# MC9S08 PL16 Reference Manual

Supports: MC9S08PL16 and MC9S08PL8

Document Number: MC9S08PL16RM
Rev. 2, 07/2018

# Contents

**Section number** **Title** **Page**

## Chapter 1
## Device Overview


1.1 Introduction....23

1.2 MCU block diagram....24

1.3 System clock distribution....25


## Chapter 2
## Pins and connections


2.1 Device pin assignment....29

2.2 Pin functions....30

2.2.1 Power (VDD, VSS)....30

2.2.2 Oscillator (XTAL, EXTAL)....31

2.2.3 External reset pin (RESET) and interrupt pin (IRQ)....32

2.2.4 Background/mode select (BKGD/MS)....33

2.2.5 Port A input/output (I/O) pins (PTA7–PTA0)....34

2.2.6 Port B input/output (I/O) pins (PTB7–PTB0)....34

2.2.7 Port C input/output (I/O) pins (PTC7–PTC0)....34

2.2.8 Port D input/output (I/O) pins (PTD7-PTD6, PTD2–PTD0)....35

2.2.9 True open drain pins (PTA3–PTA2)....35

2.3 Peripheral pinouts....35


## Chapter 3
## Power management


3.1 Introduction....37

3.2 Features....37

3.2.1 Run mode....37

3.2.2 Wait mode....38

3.2.3 Stop3 mode....38

3.2.4 Active BDM enabled in stop3 mode....38

3.2.5 LVD enabled in stop mode....39

3.2.6 Power modes behaviors ... 39
3.3 Low voltage detect (LVD) system ... 40
3.3.1 Power-on reset (POR) operation ... 41
3.3.2 LVD reset operation ... 41
3.3.3 Low-voltage warning (LVW) ... 41
3.4 Bandgap reference ... 42
3.5 Power management control bits and registers ... 42
3.5.1 System Power Management Status and Control 1 Register (PMC_SPMSC1) ... 42
3.5.2 System Power Management Status and Control 2 Register (PMC_SPMSC2) ... 44

## Chapter 4
## Memory map

4.1 Memory map ... 45
4.2 Reset and interrupt vector assignments ... 46
4.3 Register addresses and bit assignments ... 47
4.4 Random-access memory (RAM) ... 56
4.5 Flash and EEPROM ... 56
4.5.1 Overview ... 56
4.5.2 Function descriptions ... 58
4.5.2.1 Modes of operation ... 58
4.5.2.2 Flash and EEPROM memory map ... 59
4.5.2.3 Flash and EEPROM initialization after system reset ... 59
4.5.2.4 Flash and EEPROM command operations ... 60
4.5.2.5 Flash and EEPROM interrupts ... 65
4.5.2.6 Protection ... 66
4.5.2.7 Security ... 69
4.5.2.8 Flash and EEPROM commands ... 71
4.5.2.9 Flash and EEPROM command summary ... 73
4.6 Flash and EEPROM registers descriptions ... 87
4.6.1 Flash Clock Divider Register (NVM_FCLKDIV) ... 87

4.6.2 Flash Security Register (NVM_FSEC)....88
4.6.3 Flash CCOB Index Register (NVM_FCCOBIX)....89
4.6.4 Flash Configuration Register (NVM_FCNFG)....89
4.6.5 Flash Error Configuration Register (NVM_FERCNFG)....90
4.6.6 Flash Status Register (NVM_FSTAT)....91
4.6.7 Flash Error Status Register (NVM_FERSTAT)....92
4.6.8 Flash Protection Register (NVM_FPROT)....93
4.6.9 EEPROM Protection Register (NVM_EEPROT)....94
4.6.10 Flash Common Command Object Register:High (NVM_FCCOBHI)....95
4.6.11 Flash Common Command Object Register: Low (NVM_FCCOBLO)....96
4.6.12 Flash Option Register (NVM_FOPT)....96

## Chapter 5
## Interrupt

5.1 Interrupts....99
5.1.1 Interrupt stack frame....100
5.1.2 Interrupt vectors, sources, and local masks....101
5.1.3 Hardware nested interrupt....101
5.1.3.1 Interrupt priority level register....103
5.1.3.2 Interrupt priority level comparator set....104
5.1.3.3 Interrupt priority mask update and restore mechanism....104
5.1.3.4 Integration and application of the IPC....105
5.2 IRQ....105
5.2.1 Features....106
5.2.1.1 Pin configuration options....106
5.2.1.2 Edge and level sensitivity....107
5.3 Interrupt pin request register....107
5.3.1 Interrupt Pin Request Status and Control Register (IRQ_SC)....108
5.4 Interrupt priority control register....109
5.4.1 IPC Status and Control Register (IPC_SC)....110

5.4.2 Interrupt Priority Mask Pseudo Stack Register (IPC_IPMPS)..... 111
5.4.3 Interrupt Level Setting Registers n (IPC_ILRS*n*)..... 111

## Chapter 6
## System control

6.1 System device identification (SDID)..... 113
6.2 Universally unique identification (UUID)..... 113
6.3 Reset and system initialization..... 113
6.4 System options..... 114
6.4.1 BKGD pin enable..... 114
6.4.2 RESET pin enable..... 114
6.4.3 SCI0 pin reassignment..... 114
6.4.4 FTM0 channels pin reassignment..... 115
6.4.5 FTM2 channels pin reassignment..... 115
6.4.6 Bus clock output pin enable..... 115
6.5 System interconnection..... 115
6.5.1 SCI0 RxD capture..... 116
6.5.2 ADC hardware trigger..... 116
6.6 System Control Registers..... 117
6.6.1 System Reset Status Register (SYS_SRS)..... 117
6.6.2 System Background Debug Force Reset Register (SYS_SBDFR)..... 119
6.6.3 System Device Identification Register: High (SYS_SDIDH)..... 120
6.6.4 System Device Identification Register: Low (SYS_SDIDL)..... 120
6.6.5 System Options Register 1 (SYS_SOPT1)..... 121
6.6.6 System Options Register 2 (SYS_SOPT2)..... 122
6.6.7 System Options Register 3 (SYS_SOPT3)..... 123
6.6.8 Illegal Address Register: High (SYS_ILLAH)..... 124
6.6.9 Illegal Address Register: Low (SYS_ILLAL)..... 125
6.6.10 Universally Unique Identifier Register 1 (SYS_UUID1)..... 125
6.6.11 Universally Unique Identifier Register 2 (SYS_UUID2)..... 126

6.6.12 Universally Unique Identifier Register 3 (SYS_UUID3)..... 126
6.6.13 Universally Unique Identifier Register 4 (SYS_UUID4)..... 127
6.6.14 Universally Unique Identifier Register 5 (SYS_UUID5)..... 127
6.6.15 Universally Unique Identifier Register 6 (SYS_UUID6)..... 128
6.6.16 Universally Unique Identifier Register 7 (SYS_UUID7)..... 128
6.6.17 Universally Unique Identifier Register 8 (SYS_UUID8)..... 129

## Chapter 7
## Parallel input/output

7.1 Introduction..... 131
7.2 Port data and data direction..... 132
7.3 Internal pullup enable..... 133
7.4 Input glitch filter setting..... 133
7.5 Pin behavior in stop mode..... 134
7.6 Port data registers..... 134
7.6.1 Port A Data Register (PORT_PTAD)..... 135
7.6.2 Port B Data Register (PORT_PTBD)..... 135
7.6.3 Port C Data Register (PORT_PTCD)..... 136
7.6.4 Port D Data Register (PORT_PTDD)..... 136
7.6.5 Port E Data Register (PORT_PTED)..... 137
7.6.6 Port A Output Enable Register (PORT_PTAOE)..... 138
7.6.7 Port B Output Enable Register (PORT_PTBOE)..... 139
7.6.8 Port C Output Enable Register (PORT_PTCOE)..... 140
7.6.9 Port D Output Enable Register (PORT_PTDOE)..... 141
7.6.10 Port E Output Enable Register (PORT_PTEOE)..... 142
7.6.11 Port A Input Enable Register (PORT_PTAIE)..... 143
7.6.12 Port B Input Enable Register (PORT_PTBIE)..... 144
7.6.13 Port C Input Enable Register (PORT_PTCIE)..... 145
7.6.14 Port D Input Enable Register (PORT_PTDIE)..... 146
7.6.15 Port E Input Enable Register (PORT_PTEIE)..... 147

7.6.16 Port Filter Register 0 (PORT_IOFLT0)....148
7.6.17 Port Filter Register 1 (PORT_IOFLT1)....149
7.6.18 Port Filter Register 2 (PORT_IOFLT2)....149
7.6.19 Port Clock Division Register (PORT_FCLKDIV)....150
7.6.20 Port A Pullup Enable Register (PORT_PTAPE)....151
7.6.21 Port B Pullup Enable Register (PORT_PTBPE)....152
7.6.22 Port C Pullup Enable Register (PORT_PTCPE)....153
7.6.23 Port D Pullup Enable Register (PORT_PTDPE)....155
7.6.24 Port E Pullup Enable Register (PORT_PTEPE)....156

## Chapter 8
## Clock management

8.1 Clock module....157
8.2 Internal clock source (ICS)....157
8.2.1 Function description....158
8.2.1.1 Bus frequency divider....159
8.2.1.2 Low power bit usage....159
8.2.1.3 Internal reference clock (ICSIRCLK)....159
8.2.1.4 Fixed frequency clock (ICSFFCLK)....160
8.2.1.5 BDC clock....161
8.2.2 Modes of operation....161
8.2.2.1 FLL engaged internal (FEI)....162
8.2.2.2 FLL engaged external (FEE)....163
8.2.2.3 FLL bypassed internal (FBI)....163
8.2.2.4 FLL bypassed internal low power (FBILP)....163
8.2.2.5 FLL bypassed external (FBE)....164
8.2.2.6 FLL bypassed external low power (FBELP)....164
8.2.2.7 Stop (STOP)....165
8.2.3 FLL lock and clock monitor....166
8.2.3.1 FLL clock lock....166

8.2.3.2 External reference clock monitor ... 166
8.3 Initialization / application information ... 166
8.3.1 Initializing FEI mode ... 167
8.3.2 Initializing FBI mode ... 167
8.3.3 Initializing FEE mode ... 167
8.3.4 Initializing FBE mode ... 168
8.3.5 External oscillator (OSC) ... 168
8.3.5.1 Bypass mode ... 169
8.3.5.2 Low-power configuration ... 169
8.3.5.3 High-gain configuration ... 170
8.3.5.4 Initializing external oscillator for peripherals ... 170
8.4 1 kHz low-power oscillator (LPO) ... 171
8.5 Peripheral clock gating ... 171
8.6 ICS control registers ... 171
8.6.1 ICS Control Register 1 (ICS_C1) ... 172
8.6.2 ICS Control Register 2 (ICS_C2) ... 173
8.6.3 ICS Control Register 3 (ICS_C3) ... 174
8.6.4 ICS Control Register 4 (ICS_C4) ... 174
8.6.5 ICS Status Register (ICS_S) ... 175
8.6.6 OSC Status and Control Register (ICS_OSCSC) ... 176
8.7 System clock gating control registers ... 177
8.7.1 System Clock Gating Control 1 Register (SCG_C1) ... 178
8.7.2 System Clock Gating Control 2 Register (SCG_C2) ... 179
8.7.3 System Clock Gating Control 3 Register (SCG_C3) ... 180
8.7.4 System Clock Gating Control 4 Register (SCG_C4) ... 180

## Chapter 9
## Chip configurations

9.1 Introduction ... 183
9.2 Core modules ... 183

9.2.1 Central processor unit (CPU) ... 183
9.2.2 Debug module (DBG) ... 183
9.3 System modules ... 184
9.3.1 Watchdog (WDOG) ... 184
9.4 Clock module ... 184
9.5 Memory ... 184
9.5.1 Random-access-memory (RAM) ... 184
9.5.2 Non-volatile memory (NVM) ... 185
9.6 Power modules ... 185
9.7 Security ... 185
9.7.1 Cyclic redundancy check (CRC) ... 185
9.8 Timers ... 186
9.8.1 FlexTimer module (FTM) ... 186
9.8.1.1 FTM0 interconnection ... 186
9.8.1.2 FTM2 interconnection ... 186
9.8.2 8-bit modulo timer (MTIM) ... 187
9.8.2.1 MTIM0 as ADC hardware trigger ... 187
9.8.3 Real-time counter (RTC) ... 187
9.9 Communication interfaces ... 187
9.9.1 Serial communications interface (SCI) ... 187
9.9.1.1 SCI0 infrared functions ... 188
9.10 Analog ... 188
9.10.1 Analog-to-digital converter (ADC) ... 188
9.10.1.1 ADC channel assignments ... 188
9.10.1.2 Alternate clock ... 189
9.10.1.3 Hardware trigger ... 190
9.10.1.4 Temperature sensor ... 190
9.11 Human-machine interfaces HMI ... 191
9.11.1 Keyboard interrupts (KBI) ... 191

## Chapter 10
## Central processor unit

10.1 Introduction....193
10.1.1 Features....193
10.2 Programmer's Model and CPU Registers....194
10.2.1 Accumulator (A)....194
10.2.2 Index Register (H:X)....195
10.2.3 Stack Pointer (SP)....195
10.2.4 Program Counter (PC)....196
10.2.5 Condition Code Register (CCR)....196
10.3 Addressing Modes....197
10.3.1 Inherent Addressing Mode (INH)....198
10.3.2 Relative Addressing Mode (REL)....198
10.3.3 Immediate Addressing Mode (IMM)....198
10.3.4 Direct Addressing Mode (DIR)....199
10.3.5 Extended Addressing Mode (EXT)....199
10.3.6 Indexed Addressing Mode....200
10.3.6.1 Indexed, No Offset (IX)....200
10.3.6.2 Indexed, No Offset with Post Increment (IX+)....200
10.3.6.3 Indexed, 8-Bit Offset (IX1)....200
10.3.6.4 Indexed, 8-Bit Offset with Post Increment (IX1+)....201
10.3.6.5 Indexed, 16-Bit Offset (IX2)....201
10.3.6.6 SP-Relative, 8-Bit Offset (SP1)....201
10.3.6.7 SP-Relative, 16-Bit Offset (SP2)....202
10.3.7 Memory to memory Addressing Mode....202
10.3.7.1 Direct to Direct....202
10.3.7.2 Immediate to Direct....202
10.3.7.3 Indexed to Direct, Post Increment....202
10.3.7.4 Direct to Indexed, Post-Increment....203

10.4 Operation modes ... 203
10.4.1 Stop mode ... 203
10.4.2 Wait mode ... 203
10.4.3 Background mode ... 204
10.4.4 Security mode ... 205
10.5 HCS08 V6 Opcodes ... 207
10.6 Special Operations ... 207
10.6.1 Reset Sequence ... 207
10.6.2 Interrupt Sequence ... 207
10.7 Instruction Set Summary ... 208

## Chapter 11
## Keyboard Interrupts (KBI)

11.1 Introduction ... 221
11.1.1 Features ... 221
11.1.2 Modes of Operation ... 221
11.1.2.1 KBI in Wait mode ... 221
11.1.2.2 KBI in Stop modes ... 222
11.1.2.3 KBI in Active Background mode ... 222
11.1.3 Block Diagram ... 222
11.2 External signals description ... 223
11.3 Register definition ... 223
11.4 Memory Map and Registers ... 223
11.4.1 KBI Status and Control Register (KBI*x*_SC) ... 224
11.4.2 KBI Pin Enable Register (KBI*x*_PE) ... 225
11.4.3 KBI Edge Select Register (KBI*x*_ES) ... 225
11.5 Functional Description ... 226
11.5.1 Edge-only sensitivity ... 226
11.5.2 Edge and level sensitivity ... 226
11.5.3 KBI Pullup Resistor ... 227

11.5.4 KBI initialization....227

## Chapter 12
## FlexTimer Module (FTM)

12.1 Introduction....229
12.1.1 FlexTimer philosophy....229
12.1.2 Features....229
12.1.3 Modes of operation....230
12.1.4 Block diagram....230
12.2 Signal description....231
12.2.1 EXTCLK — FTM external clock....232
12.2.2 CHn — FTM channel (n) I/O pin....232
12.3 Memory map and register definition....232
12.3.1 Module memory map....232
12.3.2 Register descriptions....232
12.3.3 Status and Control (FTM*x*_SC)....234
12.3.4 Counter High (FTM*x*_CNTH)....235
12.3.5 Counter Low (FTM*x*_CNTL)....236
12.3.6 Modulo High (FTM*x*_MODH)....236
12.3.7 Modulo Low (FTM*x*_MODL)....237
12.3.8 Channel Status and Control (FTM*x*_C*n*SC)....238
12.3.9 Channel Value High (FTM*x*_C*n*VH)....239
12.3.10 Channel Value Low (FTM*x*_C*n*VL)....240
12.4 Functional Description....241
12.4.1 Clock Source....241
12.4.1.1 Counter Clock Source....241
12.4.2 Prescaler....242
12.4.3 Counter....242
12.4.3.1 Up counting....243
12.4.3.2 Up-down counting....243

12.4.3.3 Free running counter ... 244
12.4.3.4 Counter reset ... 244
12.4.4 Input capture mode ... 244
12.4.5 Output compare mode ... 245
12.4.6 Edge-aligned PWM (EPWM) mode ... 247
12.4.7 Center-aligned PWM (CPWM) mode ... 248
12.4.8 Update of the registers with write buffers ... 250
12.4.8.1 MODH:L registers ... 250
12.4.8.2 CnVH:L registers ... 251
12.4.9 BDM mode ... 251
12.5 Reset overview ... 251
12.6 FTM Interrupts ... 253
12.6.1 Timer overflow interrupt ... 253
12.6.2 Channel (n) interrupt ... 253

## Chapter 13
## 8-bit modulo timer (MTIM)

13.1 Introduction ... 255
13.2 Features ... 255
13.3 Modes of operation ... 255
13.3.1 MTIM in wait mode ... 256
13.3.2 MTIM in stop mode ... 256
13.3.3 MTIM in active background mode ... 256
13.4 Block diagram ... 256
13.5 External signal description ... 257
13.6 Register definition ... 257
13.6.1 MTIM Status and Control Register (MTIM*x*_SC) ... 257
13.6.2 MTIM Clock Configuration Register (MTIM*x*_CLK) ... 258
13.6.3 MTIM Counter Register (MTIM*x*_CNT) ... 259
13.6.4 MTIM Modulo Register (MTIM*x*_MOD) ... 260

13.7 Functional description....260
13.7.1 MTIM operation example....261

## Chapter 14
## Real-time counter (RTC)

14.1 Introduction....263
14.2 Features....263
14.2.1 Modes of operation....263
14.2.1.1 Wait mode....263
14.2.1.2 Stop modes....264
14.2.2 Block diagram....264
14.3 External signal description....264
14.4 Register definition....265
14.4.1 RTC Status and Control Register 1 (RTC_SC1)....265
14.4.2 RTC Status and Control Register 2 (RTC_SC2)....266
14.4.3 RTC Modulo Register: High (RTC_MODH)....267
14.4.4 RTC Modulo Register: Low (RTC_MODL)....267
14.4.5 RTC Counter Register: High (RTC_CNTH)....268
14.4.6 RTC Counter Register: Low (RTC_CNTL)....268
14.5 Functional description....269
14.5.1 RTC operation example....270
14.6 Initialization/application information....271

## Chapter 15
## Serial communications interface (SCI)

15.1 Introduction....273
15.1.1 Features....273
15.1.2 Modes of operation....273
15.1.3 Block diagram....274
15.2 SCI signal descriptions....276
15.2.1 Detailed signal descriptions....276

15.3 Register definition....276
15.3.1 SCI Baud Rate Register: High (SCI*x*_BDH)....277
15.3.2 SCI Baud Rate Register: Low (SCI*x*_BDL)....278
15.3.3 SCI Control Register 1 (SCI*x*_C1)....279
15.3.4 SCI Control Register 2 (SCI*x*_C2)....280
15.3.5 SCI Status Register 1 (SCI*x*_S1)....281
15.3.6 SCI Status Register 2 (SCI*x*_S2)....283
15.3.7 SCI Control Register 3 (SCI*x*_C3)....285
15.3.8 SCI Data Register (SCI*x*_D)....286
15.4 Functional description....287
15.4.1 Baud rate generation....287
15.4.2 Transmitter functional description....288
15.4.2.1 Send break and queued idle....288
15.4.3 Receiver functional description....289
15.4.3.1 Data sampling technique....290
15.4.3.2 Receiver wake-up operation....291
15.4.4 Interrupts and status flags....292
15.4.5 Baud rate tolerance....293
15.4.5.1 Slow data tolerance....294
15.4.5.2 Fast data tolerance....295
15.4.6 Additional SCI functions....296
15.4.6.1 8- and 9-bit data modes....296
15.4.6.2 Stop mode operation....296
15.4.6.3 Loop mode....297
15.4.6.4 Single-wire operation....297

## Chapter 16
## Analog-to-digital converter (ADC)

16.1 Introduction....299
16.1.1 Features....299

16.1.2 Block Diagram ..... 300
16.2 External Signal Description ..... 300
16.2.1 Analog Power (VDDA) ..... 301
16.2.2 Analog Ground (VSSA) ..... 301
16.2.3 Voltage Reference High (VREFH) ..... 301
16.2.4 Voltage Reference Low (VREFL) ..... 301
16.2.5 Analog Channel Inputs (ADx) ..... 301
16.3 ADC Control Registers ..... 302
16.3.1 Status and Control Register 1 (ADC_SC1) ..... 302
16.3.2 Status and Control Register 2 (ADC_SC2) ..... 304
16.3.3 Status and Control Register 3 (ADC_SC3) ..... 305
16.3.4 Status and Control Register 4 (ADC_SC4) ..... 306
16.3.5 Conversion Result High Register (ADC_RH) ..... 307
16.3.6 Conversion Result Low Register (ADC_RL) ..... 308
16.3.7 Compare Value High Register (ADC_CVH) ..... 309
16.3.8 Compare Value Low Register (ADC_CVL) ..... 309
16.3.9 Pin Control 1 Register (ADC_APCTL1) ..... 310
16.3.10 Pin Control 2 Register (ADC_APCTL2) ..... 311
16.4 Functional description ..... 312
16.4.1 Clock select and divide control ..... 312
16.4.2 Input select and pin control ..... 313
16.4.3 Hardware trigger ..... 313
16.4.4 Conversion control ..... 314
16.4.4.1 Initiating conversions ..... 314
16.4.4.2 Completing conversions ..... 314
16.4.4.3 Aborting conversions ..... 315
16.4.4.4 Power control ..... 315
16.4.4.5 Sample time and total conversion time ..... 316
16.4.5 Automatic compare function ..... 317

16.4.6 FIFO operation....318
16.4.7 MCU wait mode operation....321
16.4.8 MCU Stop mode operation....322
16.4.8.1 Stop mode with ADACK disabled....322
16.4.8.2 Stop mode with ADACK enabled....322
16.5 Initialization information....323
16.5.1 ADC module initialization example....323
16.5.1.1 Initialization sequence....323
16.5.1.2 Pseudo-code example....324
16.5.2 ADC FIFO module initialization example....324
16.5.2.1 Pseudo-code example....325
16.6 Application information....326
16.6.1 External pins and routing....326
16.6.1.1 Analog supply pins....326
16.6.1.2 Analog reference pins....326
16.6.1.3 Analog input pins....327
16.6.2 Sources of error....328
16.6.2.1 Sampling error....328
16.6.2.2 Pin leakage error....328
16.6.2.3 Noise-induced errors....328
16.6.2.4 Code width and quantization error....329
16.6.2.5 Linearity errors....330
16.6.2.6 Code jitter, non-monotonicity, and missing codes....330

## Chapter 17
## Cyclic redundancy check (CRC)

17.1 Introduction....333
17.2 Features....333
17.3 Block diagram....333
17.4 Modes of operation....334

17.5 Register definition....334
17.5.1 CRC Data 0 Register (CRC_D0)....335
17.5.2 CRC Data 1 Register (CRC_D1)....335
17.5.3 CRC Data 2 Register (CRC_D2)....336
17.5.4 CRC Data 3 Register (CRC_D3)....337
17.5.5 CRC Polynomial 0 Register (CRC_P0)....337
17.5.6 CRC Polynomial 1 Register (CRC_P1)....338
17.5.7 CRC Polynomial 2 Register (CRC_P2)....338
17.5.8 CRC Polynomial 3 Register (CRC_P3)....339
17.5.9 CRC Control Register (CRC_CTRL)....339
17.6 Functional description....340
17.6.1 16-bit CRC calculation....340
17.6.2 32-bit CRC calculation....340
17.6.3 Bit reverse....341
17.6.4 Result complement....341
17.6.5 CCITT compliant CRC example....341

## Chapter 18
## Watchdog (WDOG)

18.1 Introduction....343
18.1.1 Features....343
18.1.2 Block diagram....344
18.2 Memory map and register definition....345
18.2.1 Watchdog Control and Status Register 1 (WDOG_CS1)....345
18.2.2 Watchdog Control and Status Register 2 (WDOG_CS2)....347
18.2.3 Watchdog Counter Register: High (WDOG_CNTH)....348
18.2.4 Watchdog Counter Register: Low (WDOG_CNTL)....348
18.2.5 Watchdog Timeout Value Register: High (WDOG_TOVALH)....349
18.2.6 Watchdog Timeout Value Register: Low (WDOG_TOVALL)....349
18.2.7 Watchdog Window Register: High (WDOG_WINH)....350

18.2.8 Watchdog Window Register: Low (WDOG_WINL)..... 350
18.3 Functional description.....351
18.3.1 Watchdog refresh mechanism.....351
18.3.1.1 Window mode..... 352
18.3.1.2 Refreshing the Watchdog.....352
18.3.1.3 Example code: Refreshing the Watchdog.....353
18.3.2 Configuring the Watchdog.....353
18.3.2.1 Reconfiguring the Watchdog..... 353
18.3.2.2 Unlocking the Watchdog..... 354
18.3.2.3 Example code: Reconfiguring the Watchdog..... 354
18.3.3 Clock source.....354
18.3.4 Using interrupts to delay resets.....356
18.3.5 Backup reset.....356
18.3.6 Functionality in debug and low-power modes.....356
18.3.7 Fast testing of the watchdog.....357
18.3.7.1 Testing each byte of the counter..... 357
18.3.7.2 Entering user mode..... 358

## Chapter 19
## Development support

19.1 Introduction.....359
19.1.1 Forcing active background.....359
19.1.2 Features..... 359
19.2 Background debug controller (BDC).....360
19.2.1 BKGD pin description..... 361
19.2.2 Communication details..... 362
19.2.3 BDC commands..... 364
19.2.4 BDC hardware breakpoint..... 367
19.3 On-chip debug system (DBG)..... 367
19.3.1 Comparators A and B.....368

19.3.2 Bus capture information and FIFO operation ... 368
19.3.3 Change-of-flow information ... 369
19.3.4 Tag vs. force breakpoints and triggers ... 370
19.3.5 Trigger modes ... 371
19.3.6 Hardware breakpoints ... 372
19.4 Memory map and register description ... 373
19.4.1 BDC Status and Control Register (BDC_SCR) ... 373
19.4.2 BDC Breakpoint Match Register: High (BDC_BKPTH) ... 375
19.4.3 BDC Breakpoint Register: Low (BDC_BKPTL) ... 376
19.4.4 System Background Debug Force Reset Register (BDC_SBDFR) ... 376

## Chapter 20
## Debug module (DBG)

20.1 Introduction ... 379
20.1.1 Features ... 379
20.1.2 Modes of operation ... 380
20.1.3 Block diagram ... 380
20.2 Signal description ... 381
20.3 Memory map and registers ... 381
20.3.1 Debug Comparator A High Register (DBG_CAH) ... 382
20.3.2 Debug Comparator A Low Register (DBG_CAL) ... 383
20.3.3 Debug Comparator B High Register (DBG_CBH) ... 384
20.3.4 Debug Comparator B Low Register (DBG_CBL) ... 384
20.3.5 Debug Comparator C High Register (DBG_CCH) ... 385
20.3.6 Debug Comparator C Low Register (DBG_CCL) ... 386
20.3.7 Debug FIFO High Register (DBG_FH) ... 386
20.3.8 Debug FIFO Low Register (DBG_FL) ... 387
20.3.9 Debug Comparator A Extension Register (DBG_CAX) ... 388
20.3.10 Debug Comparator B Extension Register (DBG_CBX) ... 389
20.3.11 Debug Comparator C Extension Register (DBG_CCX) ... 390

20.3.12 Debug FIFO Extended Information Register (DBG_FX)....391
20.3.13 Debug Control Register (DBG_C)....391
20.3.14 Debug Trigger Register (DBG_T)....392
20.3.15 Debug Status Register (DBG_S)....394
20.3.16 Debug Count Status Register (DBG_CNT)....395
20.4 Functional description....396
20.4.1 Comparator....396
20.4.1.1 RWA and RWAEN in full modes....396
20.4.1.2 Comparator C in loop1 capture mode....396
20.4.2 Breakpoints....397
20.4.2.1 Hardware breakpoints....397
20.4.3 Trigger selection....398
20.4.4 Trigger break control (TBC)....398
20.4.4.1 Begin- and end-trigger....399
20.4.4.2 Arming the DBG module....399
20.4.4.3 Trigger modes....400
20.4.5 FIFO....402
20.4.5.1 Storing data in FIFO....403
20.4.5.2 Storing with begin-trigger....403
20.4.5.3 Storing with end-trigger....403
20.4.5.4 Reading data from FIFO....403
20.4.6 Interrupt priority....404
20.5 Resets....405

# Chapter 1
# Device Overview

## 1.1 Introduction

These devices are members of the low-cost, high-performance HCS08 family of 8-bit microcontroller units (MCUs). All MCUs in the family use the enhanced HCS08 central processor unit and are available with a variety of modules, memory sizes and types, and package types. The following table summarizes the peripheral availability per package type for the devices available.

**Table 1-1. Memory and package availability**

| Feature | MC9S08PL16 | MC9S08PL8 |
|---|---|---|
| Flash size (bytes) | 16,384 | 8,192 |
| EEPROM size (bytes) | 256 | 256 |
| RAM size (bytes) | 2,048 | 2,048 |
| LQFP-32 | Yes | Yes |
| TSSOP-20 | Yes | Yes |
| TSSOP-16 | Yes | Yes |

**Table 1-2. Feature availability**

| Pin number | 32-pin | 20-pin | 16-pin |
|---|---|---|---|
| Bus frequency (MHz) | 20 | 20 | 20 |
| IRQ | Yes | Yes | Yes |
| WDOG | Yes | Yes | Yes |
| DBG | Yes | Yes | Yes |
| IPC | Yes | Yes | Yes |
| CRC | Yes | Yes | Yes |
| ICS | Yes | Yes | Yes |
| XOSC | Yes | Yes | Yes |
| RTC | Yes | Yes | Yes |

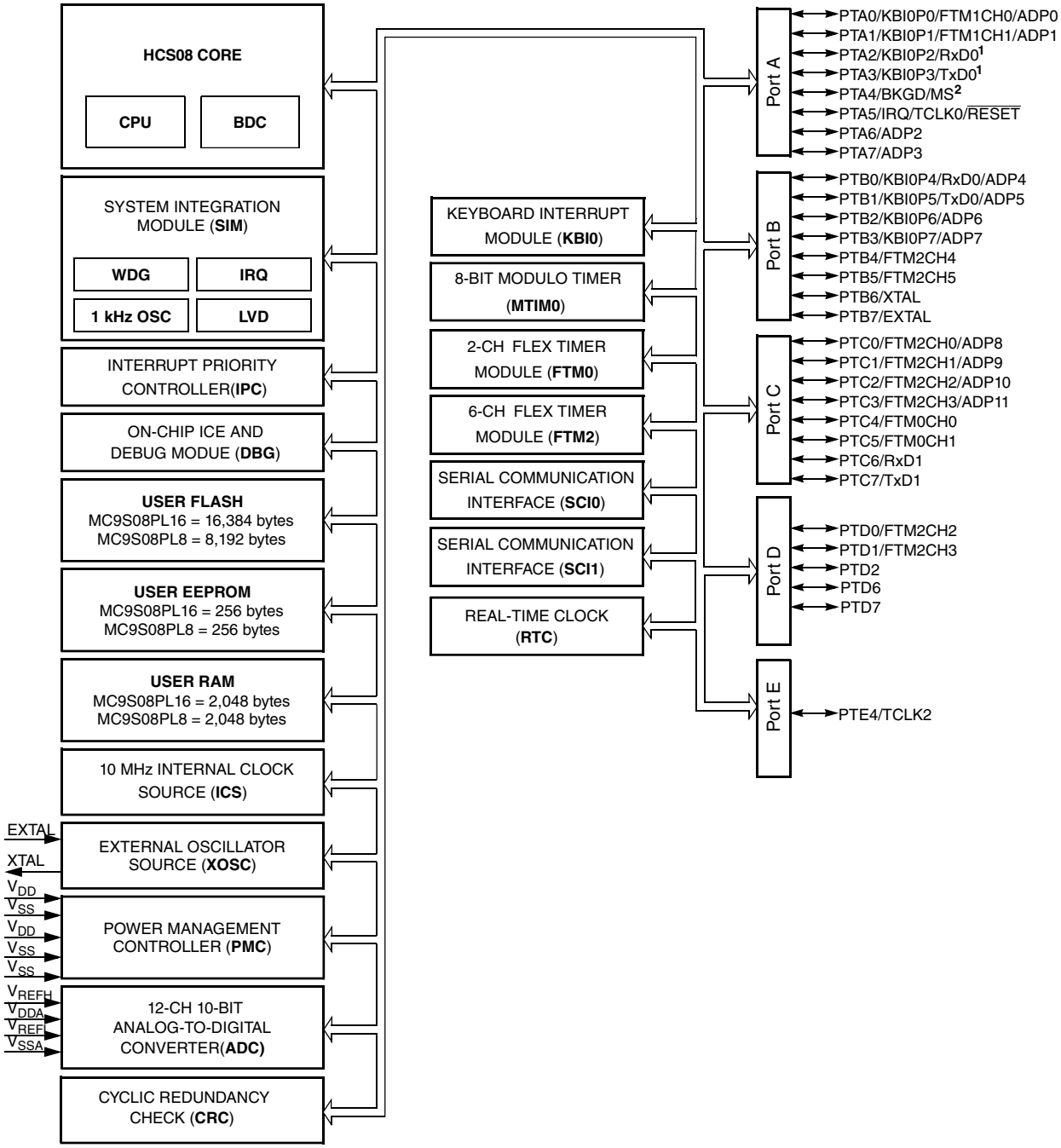
*Table continues on the next page...*

**Table 1-2. Feature availability (continued)**

| Pin number | 32-pin | 20-pin | 16-pin |
|---|---|---|---|
| FTM0 channels | 2-ch | 2-ch | 2-ch |
| FTM2 channels | 6-ch | 6-ch | 2-ch |
| MTIM0 | Yes | Yes | Yes |
| SCI0 | Yes | Yes | Yes |
| SCI1 | Yes | No | No |
| ADC | 12-channel, 10-bit | 10-channel, 10-bit | 6-channel, 10-bit |
| KBI pins | 8 | 8 | 8 |
| GPIO | 30 | 18 | 14 |

## 1.2 MCU block diagram

The block diagram below shows the structure of the MCUs.




**1. PTA2 and PTA3 operate as true-open drain when working as output.**
**2. PTA4/BKGD/MS is an output-only pin when used as port pin.**

**Figure 1-1. MCU block diagram**

## 1.3 System clock distribution

These series contain three on-chip clock sources:

- Internal clock source (ICS) module — The main clock source generator providing bus clock and other reference clocks to peripherals
- External oscillator (XOSC) module — The external oscillator providing reference clock to internal clock source (ICS), the real-time clock counter clock module (RTC) and other MCU sub-systems.
- Low-power oscillator (LPO) module — The on-chip low-power oscillator providing 1 kHz reference clock to RTC and watchdog (WDOG).

**NOTE**

For this device, the system clock is the bus clock.

The following figure shows a simplified clock connection diagram.

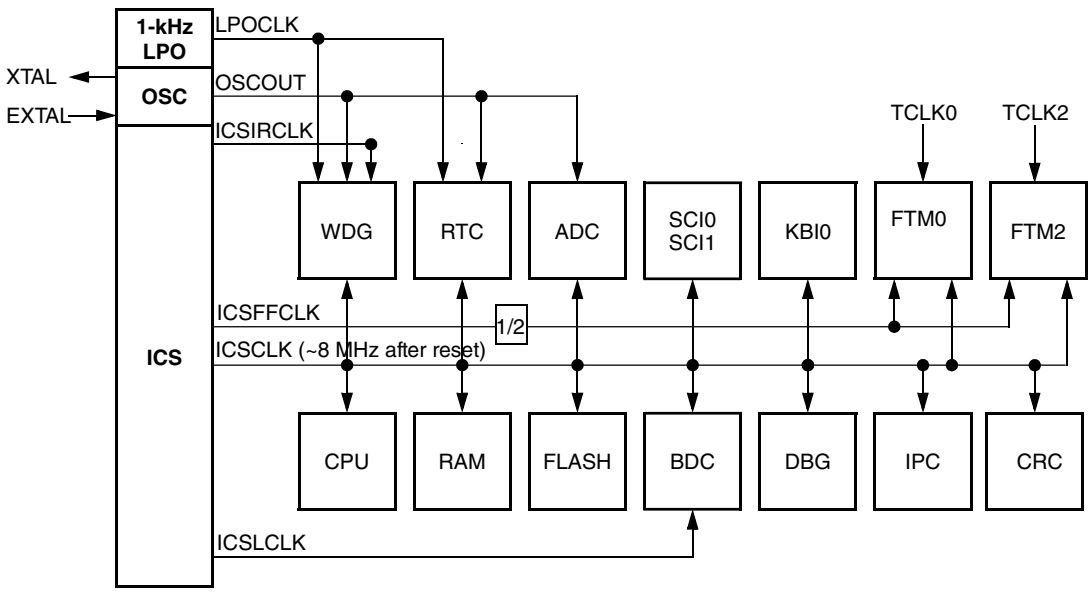



**Figure 1-2. System clock distribution diagram**

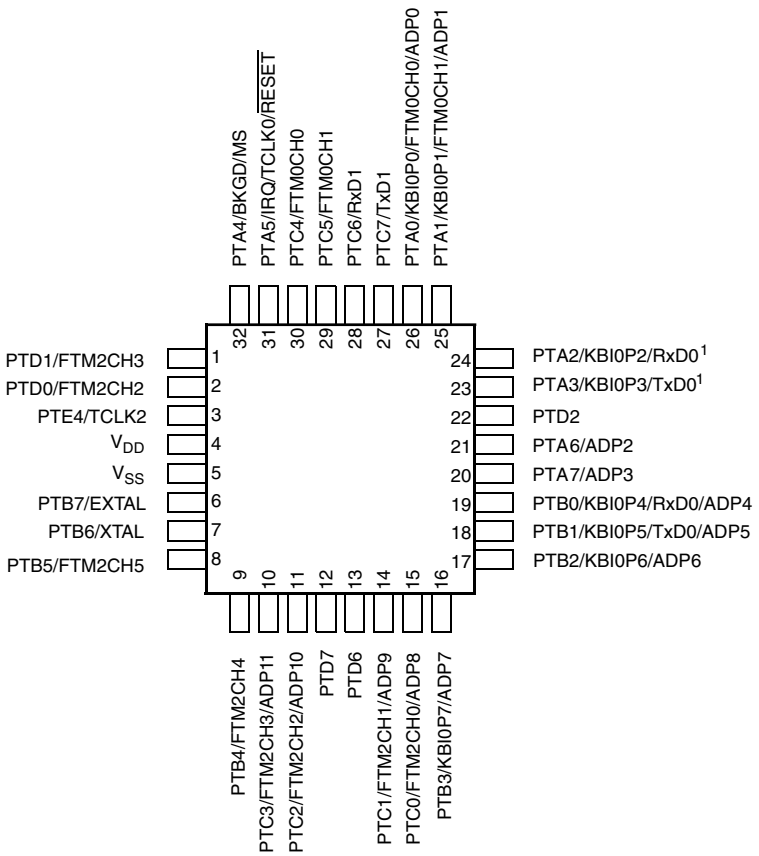
The clock system supplies:

- ICSCLK(BUS) — This up to 10 MHz clock source is used as the bus clock that is the reference to CPU and all peripherals. Control bits in the ICS control registers determine which of the clock sources is connected:
  - Internal reference clock
  - External reference clock
  - Frequency-locked loop (FLL) output

- ICSLCLK — This clock source is derived from the digitally controlled oscillator (DCO) of the ICS when the ICS is configured to run off of the internal or external reference clock. Development tools can select this internal self-clocked source (8 MHz) to speed up BDC communications in systems where the bus clock is slow.
- ICSIRCLK — This is the internal reference clock and can be selected as the clock source to the WDOG module.
- ICSFFCLK — This generates the fixed frequency clock (FFCLK) after being synchronized to the bus clock. It can be selected as clock source to the FTM and MTIM modules. The frequency of the ICSFFCLK is determined by the setting of the ICS.
- LPOCLK — This clock is generated from an internal low power oscillator (≈1 kHz) that is completely independent of the ICS module. The LPOCLK can be selected as the clock source to the RTC or WDOG modules.
- OSCOUT — This is the direct output of the external oscillator module and can be selected as the clock source for RTC, WDOG and ADC.
- TCLK0 — This is an optional external clock source for the FTM0 and MTIM0 modules. The TCLK0 must be limited to 1/4th frequency of the bus clock for synchronization.
- TCLK2 — This is an optional external clock source for the FTM2 module. The TCLK2 must be limited to 1/4th frequency of the bus clock for synchronization.

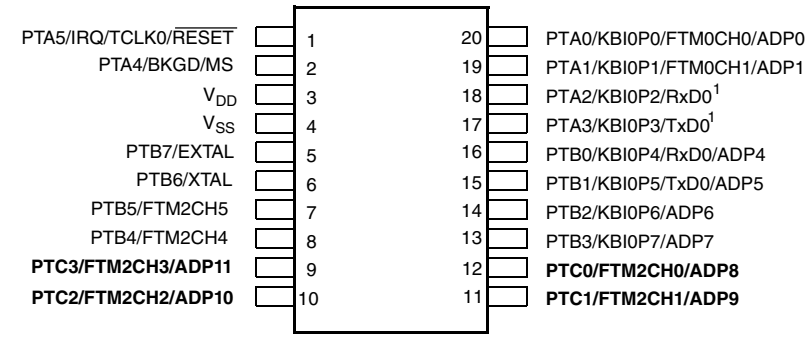
# Chapter 2
# Pins and connections

## 2.1 Device pin assignment



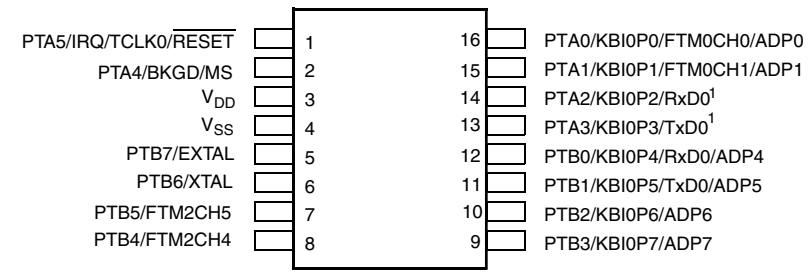

1. True open drain pins

**Figure 2-1. 32-pin LQFP package**




Pins in **bold** are not available on less pin-count packages.
1. True open drain pins

**Figure 2-2. 20-pin TSSOP package**




Pins in **bold** are not available on less pin-count packages.
1. True open drain pins

**Figure 2-3. 16-pin TSSOP package**

## 2.2 Pin functions

### 2.2.1 Power ($V_{DD}$, $V_{SS}$)

$V_{DD}$ and $V_{SS}$ are the primary power supply pins for the MCU. This voltage source supplies power to all I/O buffer circuitry and to an internal voltage regulator. The internal voltage regulator provides a regulated lower-voltage source to the CPU and to the MCU's other internal circuitry.

Typically, application systems have two separate capacitors across the power pins. In this case, there should be a bulk electrolytic capacitor, such as a 10 µF tantalum capacitor, that provides bulk charge storage for the overall system and a 0.1 µF ceramic bypass capacitor located as near to the paired $V_{DD}$ and $V_{SS}$ power pins as practical to suppress high-frequency noise.

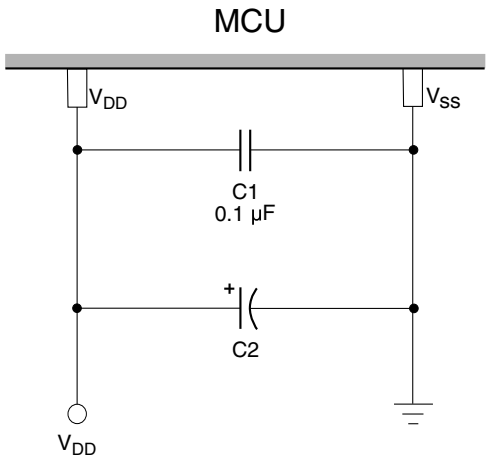



**Figure 2-4. Power supply bypassing**

## 2.2.2 Oscillator (XTAL, EXTAL)

The XTAL and EXTAL pins are used to provide the connections for the on-chip oscillator. The oscillator (XOSC) in this MCU is a Pierce oscillator that can accommodate a crystal or ceramic resonator. Optionally, an external clock source can be connected to the EXTAL input pin. The oscillator can be configured to run in stop3 mode.

Refer to the following figure, $R_S$ (when used) and $R_F$ must be low-inductance resistors such as carbon composition resistors. Wire-wound resistors, and some metal film resistors, have too much inductance. C1 and C2 normally must be high-quality ceramic capacitors that are specifically designed for high-frequency applications.

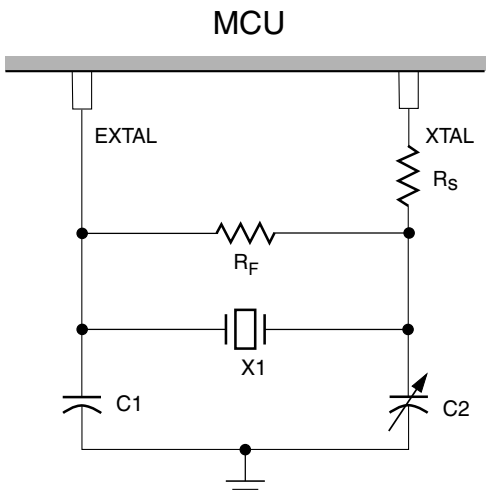



**Figure 2-5. Typical crystal or resonator circuit**

$R_F$ is used to provide a bias path to keep the EXTAL input in its linear range during crystal startup; its value is not generally critical. Typical systems use 1 M to 10 M. Higher values are sensitive to humidity and lower values reduce gain and (in extreme cases) could prevent startup.

C1 and C2 are typically in the 5 pF to 25 pF range and are chosen to match the requirements of a specific crystal or resonator. Take into account printed circuit board (PCB) capacitance and MCU pin capacitance when selecting C1 and C2. The crystal manufacturer typically specifies a load capacitance, which is the series combination of C1 and C2 (which are usually the same size). As a first-order approximation, use 10 pF as an estimate of combined pin and PCB capacitance for each oscillator pin (EXTAL and XTAL).

### 2.2.3 External reset pin ($\overline{\text{RESET}}$) and interrupt pin (IRQ)

A low on the $\overline{\text{RESET}}$ pin forces the MCU to an known startup state. $\overline{\text{RESET}}$ is bidirectional, allowing a reset of the entire system. It is driven low when any internal reset source is asserted. This pin contains an internal pullup resistor.

RESET shares an I/O pin with PTA5. The $\overline{\text{RESET}}$ pin function is enabled by default after POR reset, because internal power-on reset and low-voltage reset circuitry typically make external reset circuitry unnecessary. This pin is normally connected to the standard 6-pin background debug connector so that a development system can directly reset the MCU system. If $\overline{\text{RESET}}$ function of PTA5/IRQ/TCLK0/$\overline{\text{RESET}}$ pin is enabled, a manual external reset can be added by supplying a simple switch to ground (pull reset pin low to force a reset). When the $\overline{\text{RESET}}$ pin function is enabled, an internal pullup resistor is connected to this pin and a reset signal can feed into MCU with an input hysteresis. POR reset brings $\overline{\text{RESET}}$ pin into its default configuration, reset other than POR has no effect on the $\overline{\text{RESET}}$ pin function configuration.

When PTA5/IRQ/TCLK0/$\overline{\text{RESET}}$ is enabled as IRQ pin, it is the input source for the IRQ interrupt and is also the input for the BIH and BIL instructions. IRQ is asynchronous external interrupt pins.

In EMC-sensitive applications, an external RC filter is recommended on the reset pin. See the following figure for example.

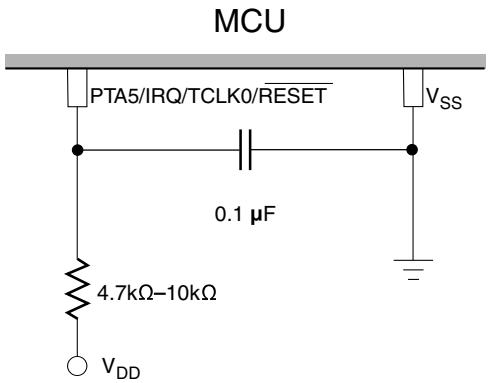



**Figure 2-6. PTA5/IRQ/TCLK0/$\overline{\text{RESET}}$ external RC filter**

## 2.2.4 Background/mode select (BKGD/MS)

During a power-on-reset (POR) or background debug force reset, the PTA4/BKGD/MS pin functions as a mode select pin. Immediately after internal reset rises the pin functions as the background pin and can be used for background debug communication. While the pin functions as a background/mode selection pin, it includes an internal pullup device and a standard output driver.

The background debug communication function is enabled when SOPT1[BKGDPE] bit is set. SOPT1[BKGDPE] is set following any reset of the MCU and must be cleared to use the PTA4/BKGD/MS pin's alternative pin functions.

If this pin is floating, the MCU will enter normal operating mode at the rising edge of reset. If a debug system is connected to the 6-pin standard background debug header, it can hold BKGD/MS low during the POR or immediately after issuing a background debug force reset, which will force the MCU into active background mode.

The BKGD pin is used primarily for background debug controller (BDC) communications using a custom protocol that uses 16 clock cycles of the target MCU's BDC clock per bit time. The target MCU's BDC clock can run as fast as the bus clock, so there should never be any significant capacitance connected to the BKGD/MS pin that interferes with background serial communications. When the pin performs output only PTA4, it can drive only capacitance-limited MOSFET. Driving a bipolar transistor directly by PTA4 is prohibited because this can cause mode entry fault and BKGD errors.

Although the BKGD pin is a pseudo open-drain pin, the background debug communication protocol provides brief, actively driven, high speedup pulses to ensure fast rise time. Small capacitances from cables and the absolute value of the internal pullup device play almost no role in determining rise and fall time on the BKGD pin.

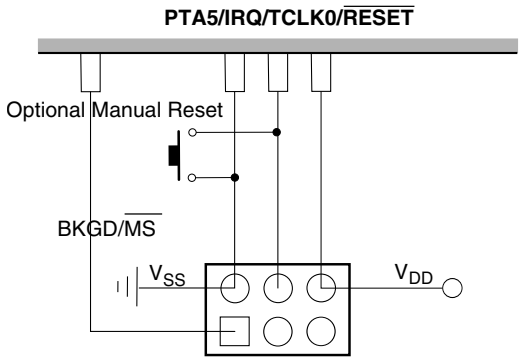



**Figure 2-7. Typical debug circuit**

## 2.2.5 Port A input/output (I/O) pins (PTA7–PTA0)

PTA7–PTA0 except PTA4 are general-purpose, bidirectional I/O port pins. These port pins also have selectable pullup devices when configured for input mode except PTA4. The pullup devices are selectable on an individual port bit basis. The pulling devices are disengaged when configured for output mode.

PTA4 is output only when used as port pin. The pulling device is disabled at this condition.

PTA3 and PTA2 provide true open drain when operated as output.

## 2.2.6 Port B input/output (I/O) pins (PTB7–PTB0)

PTB7–PTB0 are general-purpose, bidirectional I/O port pins. These port pins also have selectable pullup devices when configured for input mode, the pullup devices are selectable on an individual port bit basis. The pulling devices are disengaged when configured for output mode.

## 2.2.7 Port C input/output (I/O) pins (PTC7–PTC0)

PTC7–PTC0 are general-purpose, bidirectional I/O port pins. These port pins also have selectable pullup devices when configured for input mode, and the pullup devices are selectable on an individual port bit basis. The pulling devices are disengaged when configured for output mode.

### 2.2.8 Port D input/output (I/O) pins (PTD7-PTD6, PTD2–PTD0)

PTD7-PTD6, PTD2–PTD0are general-purpose, bidirectional I/O port pins. These port pins also have selectable pullup devices when configured for input mode, the pullup devices are selectable on an individual port bit basis. The pulling devices are disengaged when configured for output mode.

### 2.2.9 True open drain pins (PTA3–PTA2)

PTA3 and PTA2 operate in true open drain mode.

## 2.3 Peripheral pinouts

These MCUs support up to 30 general-purpose I/O pins, which are shared with on-chip peripheral functions (FTM, ADC, SCI, KBI, etc.). These 30 general-purpose I/O pins include one output-only pin (PTA4).

When a port pin is configured as general-purpose input, or when a peripheral uses the port pin as an input, the software can enable a pullup device.

For information about controlling these pins as general-purpose I/O pins, see the Parallel input/output. For information about how and when on-chip peripheral systems use these pins, see the appropriate module chapter.

Immediately after reset, all pins are configured as high-impedance general-purpose IO with internal pullup devices disabled.

**Table 2-1. Pin availability by package pin-count**

| Pin Number | | | Lowest Priority <-- --> Highest | | | | |
|---|---|---|---|---|---|---|---|
| 32-LQFP | 20-TSSOP | 16-TSSOP | Port Pin | Alt 1 | Alt 2 | Alt 3 | Alt 4 |
| 1 | — | — | PTD1 | — | FTM2CH3 | — | — |
| 2 | — | — | PTD0 | — | FTM2CH2 | — | — |
| 3 | — | — | PTE4 | — | TCLK2 | — | — |
| 4 | 3 | 3 | — | — | — | — | $V_{DD}$ |
| 5 | 4 | 4 | — | — | — | — | $V_{SS}$ |
| 6 | 5 | 5 | PTB7 | — | — | — | EXTAL |
| 7 | 6 | 6 | PTB6 | — | — | — | XTAL |
| 8 | 7 | 7 | PTB5 | — | FTM2CH5 | — | — |
| 9 | 8 | 8 | PTB4 | — | FTM2CH4 | — | — |

*Table continues on the next page...*

**Table 2-1. Pin availability by package pin-count (continued)**

| Pin Number | | | Lowest Priority <-- --> Highest | | | | |
|---|---|---|---|---|---|---|---|
| 32-LQFP | 20-TSSOP | 16-TSSOP | Port Pin | Alt 1 | Alt 2 | Alt 3 | Alt 4 |
| 10 | 9 | — | PTC3 | — | FTM2CH3 | ADP11 | — |
| 11 | 10 | — | PTC2 | — | FTM2CH2 | ADP10 | — |
| 12 | — | — | PTD7 | — | — | — | — |
| 13 | — | — | PTD6 | — | — | — | — |
| 14 | 11 | — | PTC1 | — | FTM2CH1 | ADP9 | — |
| 15 | 12 | — | PTC0 | — | FTM2CH0 | ADP8 | — |
| 16 | 13 | 9 | PTB3 | KBI0P7 | — | ADP7 | — |
| 17 | 14 | 10 | PTB2 | KBI0P6 | — | ADP6 | — |
| 18 | 15 | 11 | PTB1 | KBI0P5 | TXD0 | ADP5 | — |
| 19 | 16 | 12 | PTB0 | KBI0P4 | RXD0 | ADP4 | — |
| 20 | — | — | PTA7 | — | — | ADP3 | — |
| 21 | — | — | PTA6 | — | — | ADP2 | — |
| 22 | — | — | PTD2 | — | — | — | — |
| 23 | 17 | 13 | PTA3[1] | KBI0P3 | TXD0 | — | — |
| 24 | 18 | 14 | PTA2[1] | KBI0P2 | RXD0 | — | — |
| 25 | 19 | 15 | PTA1 | KBI0P1 | FTM0CH1 | — | ADP1 |
| 26 | 20 | 16 | PTA0 | KBI0P0 | FTM0CH0 | — | ADP0 |
| 27 | — | — | PTC7 | — | TxD1 | — | — |
| 28 | — | — | PTC6 | — | RxD1 | — | — |
| 29 | — | — | PTC5 | — | FTM0CH1 | — | — |
| 30 | — | — | PTC4 | — | FTM0CH0 | — | — |
| 31 | 1 | 1 | PTA5 | IRQ | TCLK0 | — | $\overline{\text{RESET}}$ |
| 32 | 2 | 2 | PTA4 | — | — | BKGD | MS |

1. This is a true open-drain pin when operated as output.

## Note

When an alternative function is first enabled, it is possible to get a spurious edge to the module. User software must clear any associated flags before interrupts are enabled. The table above illustrates the priority if multiple modules are enabled. The highest priority module will have control over the pin. Selecting a higher priority pin function with a lower priority function already enabled can cause spurious edges to the lower priority module. Disable all modules that share a pin before enabling another module.

# Chapter 3
# Power management

## 3.1 Introduction

The operating modes of the device are described in this chapter. Entry into each mode, exit from each mode, and functionality while in each of the modes are described.

## 3.2 Features

These MCUs feature the following power modes:

- Run mode
- Wait mode
  - CPU shuts down to conserve power
  - Bus clocks are running
  - Full voltage regulation is maintained
- Stop3 modes
  - System clocks stopped; voltage regulator in standby
  - all internal circuits powered for fast recovery

### 3.2.1 Run mode

This is the normal operating mode. In this mode, the CPU executes code from internal memory with execution beginning at the address fetched from memory at 0xFFFE: 0xFFFF after reset. The power supply is fully regulating and all peripherals can be active in run mode.

### 3.2.2 Wait mode

Wait mode is entered by executing a WAIT instruction. Upon execution of the WAIT instruction, the CPU enters a low-power state in which it is not clocked. The I bit in CCR is cleared when the CPU enters the wait mode, enabling interrupts. When an interrupt request occurs, the CPU exits the wait mode and resumes processing, beginning with the stacking operations leading to the interrupt service routine.

While the MCU is in wait mode, there are some restrictions on which background debug commands can be used. Only the BACKGROUND command and memory-access-with-status commands are available when the MCU is in wait mode. The memory-access-with-status commands do not allow memory access, but they report an error indicating that the MCU is in either stop or wait mode. The BACKGROUND command can be used to wake the MCU from wait mode and enter active background mode.

### 3.2.3 Stop3 mode

To enter stop3, the user must execute a STOP instruction with stop mode enabled (SOPT1[STOPE] = 1). Upon entering the stop3 mode, all of the clocks in the MCU are halted by default, but OSC clock and internal reference clock can be turned on by setting the ICS control registers. The ICS enters its standby state, as does the voltage regulator and the ADC. The states of all of the internal registers and logic, as well as the RAM content, are maintained. The I/O pin states are not latched at the pin. Instead they are maintained by virtue of the states of the internal logic driving the pins being maintained.

Exit from stop3 is done by asserting reset or through an interrupt. The interrupt include the asynchronous interrupt from the IRQ or KBI pins, the SCI receive interrupt, the ADC or LVI interrupt and the real-time interrupt.

If stop3 is exited by means of the $\overline{\text{RESET}}$ pin, then the MCU will be reset and operation will resume after taking the reset vector. Exit by means of an asynchronous interrupt or the real-time interrupt will result in the MCU taking the appropriate interrupt vector.

The LPO (≈1 kHz) for the real-time counter clock allows a wakeup from stop3 mode with no external components. When RTC_SC2[RTCPS] is clear, the real-time counter clock function is disabled.

### 3.2.4 Active BDM enabled in stop3 mode

Entry into the active background mode from run mode is enabled if the BDC_SCR[ENBDM] bit is set. This register is described in the development support. If BDC_SCR[ENBDM] is set when the CPU executes a STOP instruction, the system clocks to the background debug logic remain active when the MCU enters stop mode, so background debug communication is still possible. In addition, the voltage regulator does not enter its low-power standby state but maintains full internal regulation.

Most background commands are not available in stop mode. The memory-access-with-status commands do not allow memory access, but they report an error indicating that the MCU is in either stop or wait mode. The BACKGROUND command can be used to wake the MCU from stop and enter active background mode if the BDC_SCR[ENBDM] bit is set. After entering background debug mode, all background commands are available.

### 3.2.5 LVD enabled in stop mode

The LVD system is capable of generating either an interrupt or a reset when the supply voltage drops below the LVD voltage. If the LVD is enabled in stop (LVDE and LVDSE bits in SPMSC1 both set) at the time the CPU executes a STOP instruction, then the voltage regulator remains active during stop3 mode.

### 3.2.6 Power modes behaviors

Executing the WAIT or STOP command puts the MCU in a low power consumption mode for standby situations. The system integration module (SIM) holds the CPU in a non-clocked state. The operation of each of these modes is described in the following subsections. Both STOP and WAIT clear the interrupt mask (I) in the condition code register, allowing interrupt to occur. The following table shows the low power mode behaviors.

**Table 3-1. Low power mode behavior**

| Mode | Run | Wait | Stop3 |
|---|---|---|---|
| PMC | Full regulation | Full regulation | Loose regulation |
| ICS | On | On | Optional on |
| OSC | On | On | Optional on |
| LPO | On | On | On |
| CPU | On | Standby | Standby |
| FLASH | On | On | Standby |

*Table continues on the next page...*

**Table 3-1. Low power mode behavior (continued)**

| Mode | Run | Wait | Stop3 |
|---|---|---|---|
| RAM | On | Standby | Standby |
| ADC | On | On | Optional on |
| I/O | On | On | States held |
| SCI | On | On | Standby |
| FTM | On | On | Standby |
| MTIM | On | On | Standby |
| WDOG | On | On | Optional on |
| DBG | On | On | Standby |
| IPC | On | On | Standby |
| RTC | On | On | Optional on |
| LVD | On | On | Optional on |

## 3.3 Low voltage detect (LVD) system

This device includes a system to protect against low voltage conditions in order to protect memory contents and control MCU system states during supply voltage variations. This system consists of a power-on reset (POR) circuit and an LVD circuit with a user selectable trip voltage, either high ($V_{LVDH}$) or low ($V_{LVDL}$). The LVD circuit is enabled when SPMSC1[LVDE] is set and the trip voltage is selected by SPMSC2[LVDV]. The LVD is disabled upon entering the stop modes unless the SPMSC1[LVDSE] bit is set or active BDM enabled (BDCSCR[ENBDM]=1). If SPMSC1[LVDSE] and SPMSC1[LVDE] are both set, the current consumption in stop3 with the LVD enabled will be greater.

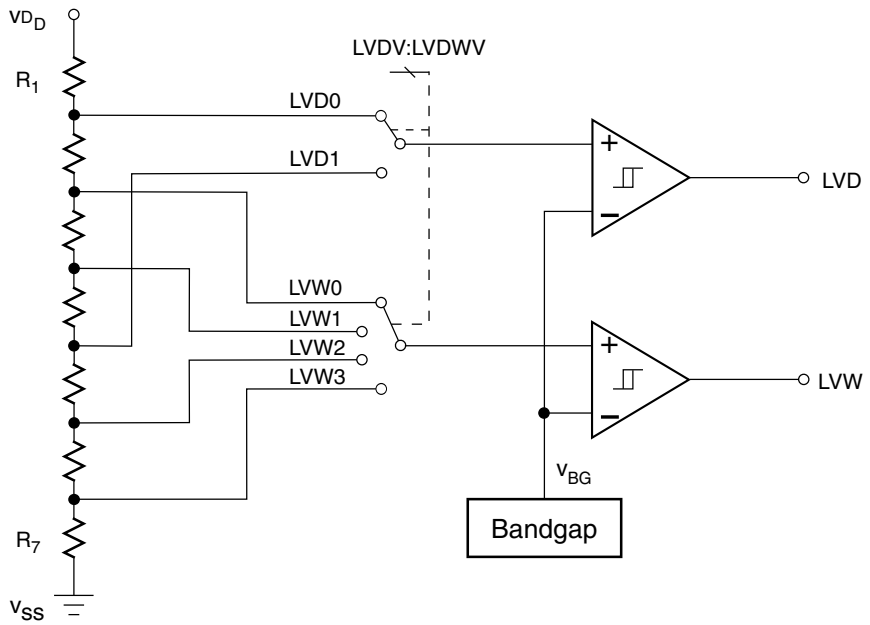



**Figure 3-1. Low voltage detect (LVD) block diagram**

## 3.3.1 Power-on reset (POR) operation

When power is initially applied to the MCU, or when the supply voltage drops below the $V_{POR}$ level, the POR circuit will cause a reset condition. As the supply voltage rises, the LVD circuit will hold the chip in reset until the supply has risen above the $V_{LVDL}$ level. Both the SRS[POR] and SRS[LVD] are set following a POR.

## 3.3.2 LVD reset operation

The LVD can be configured to generate a reset upon detection of a low voltage condition by setting SPMSC1[LVDRE] to 1. After an LVD reset has occurred, the LVD system will hold the MCU in reset until the supply voltage has risen above the level determined by LVDV. The SRS[LVD] bit is set following either an LVD reset or POR.

## 3.3.3 Low-voltage warning (LVW)

The LVD system has a low voltage warning flag to indicate that the supply voltage is approaching the LVD voltage. When a low voltage condition is detected and the LVD circuit is configured for interrupt operation (SPMSC1[LVDE] set, SPMSC1[LVWIE] set), SPMSC1[LVWF] will be set and LVW interrupt will occur. There are four user-selectable trip voltages for the LVW upon each LVDV configuration. The trip voltage is selected by SPMSC2[LVWV].

## 3.4 Bandgap reference

This device includes an on-chip bandgap reference (≈1.2V) connected to ADC channel. The bandgap reference voltage will not drop under the full operating voltage even when the operating voltage is falling. This reference voltage acts as an ideal reference voltage for accurate measurements.

## 3.5 Power management control bits and registers

**PMC memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 3040 | System Power Management Status and Control 1 Register (PMC_SPMSC1) | 8 | R/W | 1Ch | 3.5.1/42 |
| 3041 | System Power Management Status and Control 2 Register (PMC_SPMSC2) | 8 | R/W | 00h | 3.5.2/44 |

### 3.5.1 System Power Management Status and Control 1 Register (PMC_SPMSC1)

This high page register contains status and control bits to support the low-voltage detection function, and to enable the bandgap voltage reference for use by the ADC module. This register should be written during the user's reset initialization program to set the desired controls, even if the desired settings are the same as the reset settings.

Address: 3040h base + 0h offset = 3040h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | LVWF | 0 | LVWIE | LVDRE | LVDSE | LVDE | BGBDS | BGBE |
| Write | | LVWACK | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

**PMC_SPMSC1 field descriptions**

| Field | Description |
|---|---|
| 7<br>LVWF | Low-Voltage Warning Flag<br><br>The LVWF bit indicates the low-voltage warning status. |

*Table continues on the next page...*

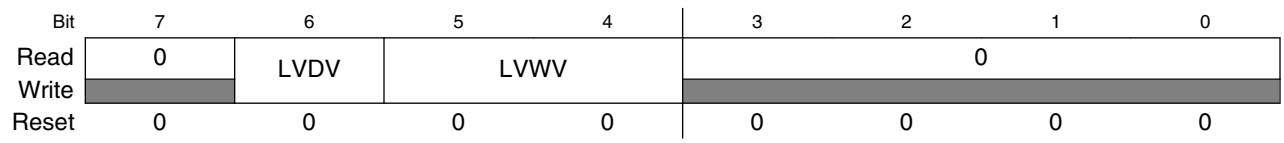
## PMC_SPMSC1 field descriptions (continued)

| Field | Description |
|---|---|
| | **NOTE:** LVWF will be set in the case when $V_{Supply}$ transitions below the trip point or after reset and $V_{Supply}$ is already below $V_{LVW}$. LVWF bit may be 1 after power on reset, therefore, to use LVW interrupt function, before enabling LVWIE, LVWF must be cleared by writing LVWACK first.<br><br>0 Low-voltage warning is not present.<br>1 Low-voltage warning is present or was present. |
| 6<br>LVWACK | Low-Voltage Warning Acknowledge<br><br>If LVWF = 1, a low-voltage condition has occurred. To acknowledge this low-voltage warning, write 1 to LVWACK, which automatically clears LVWF to 0 if the low-voltage warning is no longer present. |
| 5<br>LVWIE | Low-Voltage Warning Interrupt Enable<br><br>This bit enables hardware interrupt requests for LVWF.<br><br>0 Hardware interrupt disabled (use polling).<br>1 Request a hardware interrupt when LVWF = 1. |
| 4<br>LVDRE | Low-Voltage Detect Reset Enable<br><br>This write-once bit enables LVD events to generate a hardware reset (provided LVDE = 1).<br><br>**NOTE:** This bit can be written only one time after reset. Additional writes are ignored.<br><br>0 LVD events do not generate hardware resets.<br>1 Force an MCU reset when an enabled low-voltage detect event occurs. |
| 3<br>LVDSE | Low-Voltage Detect Stop Enable<br><br>Provided LVDE = 1, this read/write bit determines whether the low-voltage detect function operates when the MCU is in stop mode.<br><br>0 Low-voltage detect disabled during stop mode.<br>1 Low-voltage detect enabled during stop mode. |
| 2<br>LVDE | Low-Voltage Detect Enable<br><br>This write-once bit enables low-voltage detect logic and qualifies the operation of other bits in this register.<br><br>**NOTE:** This bit can be written only one time after reset. Additional writes are ignored.<br><br>0 LVD logic disabled.<br>1 LVD logic enabled. |
| 1<br>BGBDS | Bandgap Buffer Drive Select<br><br>This bit is used to select the high drive mode of the bandgap buffer.<br><br>0 Bandgap buffer enabled in low drive mode if BGBE = 1.<br>1 Bandgap buffer enabled in high drive mode if BGBE = 1. |
| 0<br>BGBE | Bandgap Buffer Enable<br><br>This bit enables an internal buffer for the bandgap voltage reference for use by the ADC module on one of its internal channels.<br><br>0 Bandgap buffer disabled.<br>1 Bandgap buffer enabled. |

## 3.5.2 System Power Management Status and Control 2 Register (PMC_SPMSC2)

This register is used to report the status of the low-voltage warning function, and to configure the stop mode behavior of the MCU. This register should be written during the user's reset initialization program to set the desired controls, even if the desired settings are the same as the reset settings.

Address: 3040h base + 1h offset = 3041h




**PMC_SPMSC2 field descriptions**

| Field | Description |
|---|---|
| 7<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 6<br>LVDV | Low-Voltage Detect Voltage Select<br><br>This write-once bit selects the low-voltage detect (LVD) trip point setting. See data sheet for details.<br><br>0 Low trip point selected ($V_{LVD} = V_{LVDL}$).<br>1 High trip point selected ($V_{LVD} = V_{LVDH}$). |
| 5–4<br>LVWV | Low-Voltage Warning Voltage Select<br><br>This bit selects the low-voltage warning (LVW) trip point voltage. See data sheet for details.<br><br>00 Low trip point selected ($V_{LVW} = V_{LVW1}$).<br>01 Middle 1 trip point selected ($V_{LVW} = V_{LVW2}$).<br>10 Middle 2 trip point selected ($V_{LVW} = V_{LVW3}$).<br>11 High trip point selected ($V_{LVW} = V_{LVW4}$). |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

# Chapter 4
# Memory map

## 4.1 Memory map

The HCS08 core processor can address 64 KB of memory space. The memory map, shown in the following figure, includes:

- User flash memory (flash)
  - MC9S08PL16: 16,384 bytes; 32 pages of 512 bytes each
  - MC9S08PL8: 8,192 bytes; 16 pages of 512 bytes each
- Random-access memory (RAM)
  - MC9S08PL16: 2,048 bytes
  - MC9S08PL8: 2,048 bytes
- Electrically erasable programmable read-only memory (EEPROM)
  - MC9S08PL16: 256 bytes; 128 pages of 2 bytes each
  - MC9S08PL8: 256 bytes; 128 pages of 2 bytes each
- Direct-page registers (0x0000 through 0x003F)
- High-page registers (0x3000 through 0x30FF)

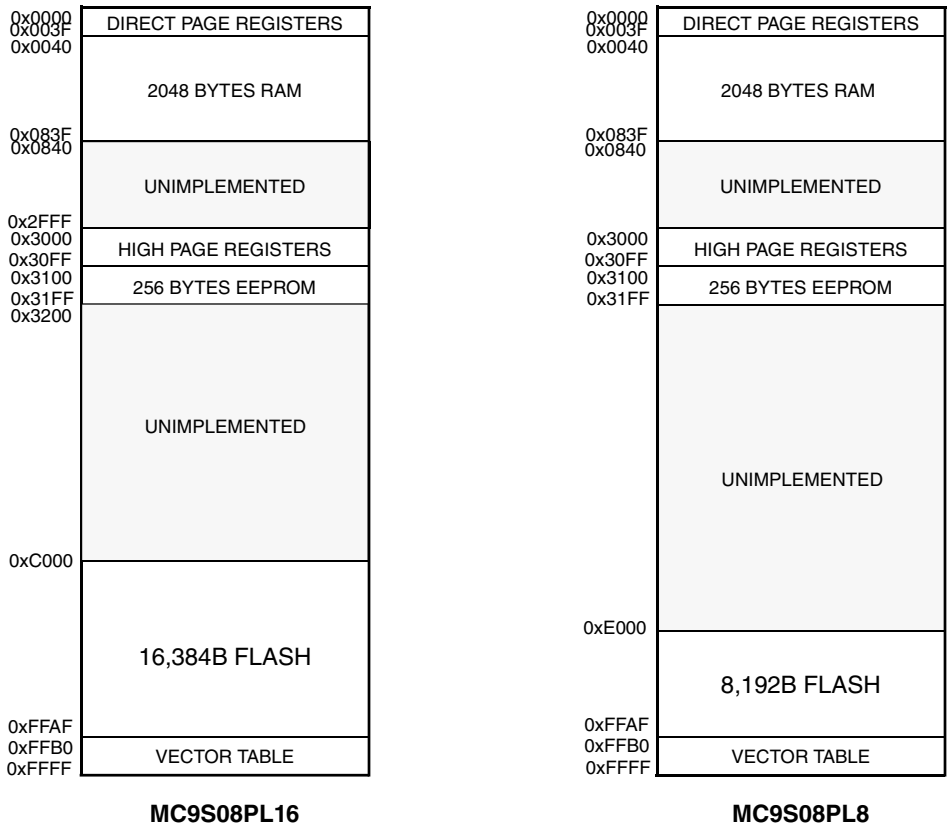



**Figure 4-1. Memory map**

## 4.2 Reset and interrupt vector assignments

The following table shows address assignments for reset and interrupt vectors. The vector names shown in this table are the labels used in the header files for the device.

**Table 4-1. Reset and interrupt vectors**

| Address (high/low) | Vector | Vector name |
|---|---|---|
| 0xFFB0:FFB1 | NVM | Vnvm |
| 0xFFB2:FFB3 | Reserved | Reserved |
| 0xFFB4:FFB5 | KBI0 | Vkbi0 |
| 0xFFB6:FFB7 | Reserved | Reserved |
| 0xFFB8:FFB9 | RTC | Vrtc |
| 0xFFBA:FFBB | Reserved | Reserved |
| 0xFFBC:FFBD | Reserved | Reserved |
| 0xFEBE:FFBF | Reserved | Reserved |
| 0xFFC0:FFC1 | Reserved | Reserved |
| 0xFFC2:FFC3 | Reserved | Reserved |

*Table continues on the next page...*

**Table 4-1. Reset and interrupt vectors (continued)**

| Address (high/low) | Vector | Vector name |
|---|---|---|
| 0xFFC4:FFC5 | Reserved | Reserved |
| 0xFFC6:FFC7 | SCI1 transmit | Vsci1txd |
| 0xFFC8:FFC9 | SCI1 receive | Vsci1rxd |
| 0xFFCA:FFCB | SCI1 error | Vsci1err |
| 0xFFCC:FFCD | SCI0 transmit | Vsci0txd |
| 0xFFCE:FFCF | SCI0 receive | Vsci0rxd |
| 0xFFD0:FFD1 | SCI0 error | Vsci0err |
| 0xFFD2:FFD3 | ADC | Vadc |
| 0xFFD4:FFD5 | Reserved | Reserved |
| 0xFFD6:FFD7 | Reserved | Reserved |
| 0xFFD8:FFD9 | MTIM0 | Vmtim0 |
| 0xFFDA:FFDB | FTM0 overflow | Vftm0ovf |
| 0xFFDC:FFDD | FTM0 channel 1 | Vftm0ch1 |
| 0xFFDE:FFDF | FTM0 channel 0 | Vftm0ch0 |
| 0xFFE0:FFE1 | Reserved | Reserved |
| 0xFFE2:FFE3 | Reserved | Reserved |
| 0xFFE4:FFE5 | Reserved | Reserved |
| 0xFFE6:FFE7 | FTM2 overflow | Vftm2ovf |
| 0xFFE8:FFE9 | FTM2 channel 5 | Vftm2ch5 |
| 0xFFEA:FFEB | FTM2 channel 4 | Vftm2ch4 |
| 0xFFEC:FFED | FTM2 channel 3 | Vftm2ch3 |
| 0xFFEE:FFEF | FTM2 channel 2 | Vftm2ch2 |
| 0xFFF0:FFF1 | FTM2 channel 1 | Vftm2ch1 |
| 0xFFF2:FFF3 | FTM2 channel 0 | Vftm2ch0 |
| 0xFFF4:FFF5 | Reserved | Reserved |
| 0xFFF6:FFF7 | Clock loss of lock | Vclk |
| 0xFFF8:FFF9 | Low voltage warning | Vlvw |
| 0xFFFA:FFFB | IRQ or Watchdog | Virq or Vwdog |
| 0xFFFC:FFFD | SWI | Vswi |
| 0xFFFE:FFFF | Reset | Vreset |

## 4.3 Register addresses and bit assignments

The register definitions vary in different memory sizes. The register addresses of unused peripherals are reserved. The following table shows the register availability of the devices.

**Table 4-2. Peripheral registers availability**

| Address | Bytes | Peripheral registers |
|---|---|---|
| 0x0000—0x0004 | 5 | Port data |
| 0x0010—0x0017 | 8 | ADC |
| 0x0018—0x001B | 4 | MTIM0 |
| 0x0020—0x002A | 11 | FTM0 |
| 0x003B—0x003B | 1 | IRQ |
| 0x003C—0x003C | 1 | KBI0 |
| 0x003E—0x003F | 2 | IPC |
| 0x3000—0x300B | 12 | SIM |
| 0x300C—0x300F | 4 | SCG |
| 0x3010—0x301F | 16 | DBG |
| 0x3020—0x302C | 13 | NVM |
| 0x3030—0x3037 | 8 | WDOG |
| 0x3038—0x303E | 7 | ICS, XOSC |
| 0x3040—0x3041 | 2 | PMC |
| 0x304A—0x304B | 2 | SYS |
| 0x3050—0x3059 | 10 | IPC |
| 0x3060—0x3068 | 9 | CRC |
| 0x306A—0x306F | 6 | RTC |
| 0x307C—0x307D | 2 | KBI0 |
| 0x3080—0x3087 | 8 | SCI0 |
| 0x3088—0x308F | 8 | SCI1 |
| 0x30AC—0x30AD | 2 | ADC |
| 0x30B0—0x30B4 | 5 | Port output enable |
| 0x30B8—0x30BC | 5 | Port input enable |
| 0x30C0—0x30EA | 43 | FTM2 |
| 0x30EC—0x30EF | 4 | Port filter |
| 0x30F0—0x30F4 | 5 | Port pullup |
| 0x30F8—0x30FF | 8 | SYS |

The registers in the devices are divided into two groups:

- Direct-page registers are located in the first 64 locations in the memory map, so they can be accessed with efficient direct addressing mode instructions.
- High-page registers are used much less often, so they are located above 0x3000 in the memory map. This leaves room in the direct page for more frequently used registers and variables.

Direct-page registers can be accessed with efficient direct addressing mode instructions. Bit manipulation instructions can be used to access any bit in a direct-page register.

The direct page registers can use the more efficient direct addressing mode, which requires only the lower byte of the address.

The following tables are summaries of all user-accessible direct-page and high-page registers and control bits. Cells that are not associated with named bits are shaded. A shaded cell with a 0 indicates this unused bit always reads as a 0; and a shaded cell with a 1 indicates this unused bit always reads as a 1. Shaded cells with dashes indicate unused or reserved bit locations that could read as 1s or 0s.

**Table 4-3. Direct-page register allocation**

| Address | Register name | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x0000 | PORT_PTAD | PTAD7 | PTAD6 | PTAD5 | PTAD4 | PTAD3 | PTAD2 | PTAD1 | PTAD0 |
| 0x0001 | PORT_PTBD | PTBD7 | PTBD6 | PTBD5 | PTBD4 | PTBD3 | PTBD2 | PTBD1 | PTBD0 |
| 0x0002 | PORT_PTCD | PTCD7 | PTCD6 | PTCD5 | PTCD4 | PTCD3 | PTCD2 | PTCD1 | PTCD0 |
| 0x0003 | PORT_PTDD | PTDD7 | PTDD6 | — | — | — | PTDD2 | PTDD1 | PTDD0 |
| 0x0004 | PORT_PTED | — | — | — | PTED4 | — | — | — | — |
| 0x0005-0x0007 | Reserved | — | — | — | — | — | — | — | — |
| 0x0008-0x000F | Reserved | — | — | — | — | — | — | — | — |
| 0x0010 | ADC_SC1 | COCO | AIEN | ADCO | ADCH | | | | |
| 0x0011 | ADC_SC2 | ADACT | ADTRG | ACFE | ACFGT | FEMPTY | FFULL | — | — |
| 0x0012 | ADC_SC3 | ADLPC | ADIV | | ADLSMP | MODE | | ADICLK | |
| 0x0013 | ADC_SC4 | — | ASCANE | ACFSEL | — | — | AFDEP | | |
| 0x0014 | ADC_RH | — | — | — | — | — | — | 9 | Bit 8 |
| 0x0015 | ADC_RL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x0016 | ADC_CVH | — | — | — | — | — | — | 9 | Bit 8 |
| 0x0017 | ADC_CVL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x0018 | MTIM0_SC | TOF | TOIE | TRST | TSTP | — | — | — | — |
| 0x0019 | MTIM0_CLK | — | — | CLKS | | PS | | | |
| 0x001A | MTIM0_CNT | COUNT | | | | | | | |
| 0x001B | MTIM0_MOD | MOD | | | | | | | |
| 0x001C-0x001F | Reserved | — | — | — | — | — | — | — | — |
| 0x0020 | FTM0_SC | TOF | TOIE | CPWMS | CLKS1 | CLKS0 | PS2 | PS1 | PS0 |
| 0x0021 | FTM0_CNTH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x0022 | FTM0_CNTL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x0023 | FTM0_MODH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x0024 | FTM0_MODL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x0025 | FTM0_C0SC | CHF | CHIE | MSB | MSA | ELSB | ELSA | — | — |
| 0x0026 | FTM0_C0VH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x0027 | FTM0_C0VL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x0028 | FTM0_C1SC | CHF | CHIE | MSB | MSA | ELSB | ELSA | — | — |

*Table continues on the next page...*

## Table 4-3. Direct-page register allocation (continued)

| Address | Register name | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x0029 | FTM0_C1VH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x002A | FTM0_C1VL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x002B | Reserved | — | — | — | — | — | — | — | — |
| 0x002C-0x002F | Reserved | — | — | — | — | — | — | — | — |
| 0x0030-0x003A | Reserved | — | — | — | — | — | — | — | — |
| 0x003B | IRQ_SC | — | IRQPDD | IRQEDG | IRQPE | IRQF | IRQACK | IRQIE | IRQMOD |
| 0x003C | KBI0_SC | — | — | — | — | KBF | KBACK | KBIE | KBMOD |
| 0x003D | Reserved | — | — | — | — | — | — | — | — |
| 0x003E | IPC_SC | IPCE | — | PSE | PSF | PULIPM | — | IPM | |
| 0x003F | IPC_IPMPS | IPM3 | | IPM2 | | IPM1 | | IPM0 | |

## Table 4-4. High-page register allocation

| Address | Register name | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x3000 | SYS_SRS | POR | PIN | WDOG | ILOP | ILAD | LOC | LVD | — |
| 0x3001 | SYS_SBDFR | — | — | — | — | — | — | — | BDFR |
| 0x3002 | SYS_SDIDH | — | — | — | — | ID11 | ID10 | ID9 | ID8 |
| 0x3003 | SYS_SDIDL | ID7 | ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 |
| 0x3004 | SYS_SOPT1 | SCI0PS | — | — | FTM2PS | BKGDPE | RSTPE | FWAKE | STOPE |
| 0x3005 | SYS_SOPT2 | TXDME | — | — | RXDCE | — | — | ADHWTS | |
| 0x3006 | SYS_SOPT3 | — | FTM0PS | — | — | CLKOE | BUSREF | | |
| 0x3007-0x300B | Reserved | — | — | — | — | — | — | — | — |
| 0x300C | SCG_C1 | FTM2 | — | FTM0 | — | — | — | MTIM0 | RTC |
| 0x300D | SCG_C2 | — | — | DBG | NVM | IPC | CRC | — | — |
| 0x300E | SCG_C3 | — | — | SCI1 | SCI0 | — | — | — | — |
| 0x300F | SCG_C4 | — | — | ADC | — | IRQ | — | — | KBI0 |
| 0x3010 | DBG_CAH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x3011 | DBG_CAL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x3012 | DBG_CBH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x3013 | DBG_CBL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x3014 | DBG_CCH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x3015 | DBG_CCL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x3016 | DBG_FH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x3017 | DBG_FL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x3018 | DBG_CAX | RWAEN | RWA | — | — | — | — | — | — |
| 0x3019 | DBG_CBX | RWBEN | RWB | — | — | — | — | — | — |
| 0x301A | DBG_CCX | RWCEN | RWC | — | — | — | — | — | — |

*Table continues on the next page...*

## Table 4-4. High-page register allocation (continued)

| Address | Register name | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x301B | DBG_FX | PPACC | — | — | — | — | — | — | Bit 16 |
| 0x301C | DBG_C | DBGEN | ARM | TAG | BRKEN | — | — | — | LOOP1 |
| 0x301D | DBG_T | TRGSEL | BEGIN | — | — | TRG | | | |
| 0x301E | DBG_S | AF | BF | CF | — | — | — | — | ARMF |
| 0x301F | DBG_CNT | — | — | — | — | CNT | | | |
| 0x3020 | NVM_FCLKDIV | FDIVLD | FDIVLCK | FDIV5 | FDIV4 | FDIV3 | FDIV2 | FDIV1 | FDIV0 |
| 0x3021 | NVM_FSEC | KEYEN1 | KEYEN0 | 1 | 1 | 1 | 1 | SEC1 | SEC0 |
| 0x3022 | NVM_FCCOBIX | — | — | — | — | — | CCOBIX2 | CCOBIX1 | CCOBIX0 |
| 0x3023 | Reserved | — | — | — | — | — | — | — | — |
| 0x3024 | NVM_FCNFG | CCIE | — | — | IGNSF | — | — | FDFD | FSFD |
| 0x3025 | NVM_FERCNFG | — | — | — | — | — | — | DFDIE | SFDIE |
| 0x3026 | NVM_FSTAT | CCIF | — | ACCERR | FPVIOL | MGBUSY | — | MGSTAT1 | MGSTAT0 |
| 0x3027 | NVM_FERSTAT | — | — | — | — | — | — | DFDIF | SFDIF |
| 0x3028 | NVM_FPROT | FPOEN | — | FPHDIS | FPHS1 | FPHS0 | — | — | — |
| 0x3029 | NVM_EEPROT | DPOPEN | — | — | — | — | DPS2 | DPS1 | DPS0 |
| 0x302A | NVM_FCCOBHI | CCOB15 | CCOB14 | CCOB13 | CCOB12 | CCOB11 | CCOB10 | CCOB9 | CCOB8 |
| 0x302B | NVM_FCCOBLO | CCOB7 | CCOB6 | CCOB5 | CCOB4 | CCOB3 | CCOB2 | CCOB1 | CCOB0 |
| 0x302C | NVM_FOPT | NV7 | NV6 | NV5 | NV4 | NV3 | NV2 | NV1 | NV0 |
| 0x302D-0x302F | Reserved | — | — | — | — | — | — | — | — |
| 0x3030 | WDOG_CS1 | EN | INT | UPDATE | TST | | DBG | WAIT | STOP |
| 0x3031 | WDOG_CS2 | WIN | FLG | — | PRES | — | — | CLK | |
| 0x3032 | WDOG_CNTH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x3033 | WDOG_CNTL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x3034 | WDOG_TOVALH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x3035 | WDOG_TOVALL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x3036 | WDOG_WINH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x3037 | WDOG_WINL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x3038 | ICS_C1 | CLKS | | RDIV | | | IREFS | IRCLKEN | IREFSTEN |
| 0x3039 | ICS_C2 | BDIV | | | LP | — | — | — | — |
| 0x303A | ICS_C3 | SCTRIM | | | | | | | |
| 0x303B | ICS_C4 | LOLIE | — | CME | — | — | — | — | SCFTRIM |
| 0x303C | ICS_S | LOLS | LOCK | — | IREFST | CLKST | | — | — |
| 0x303D | Reserved | — | — | — | — | — | — | — | — |

*Table continues on the next page...*

## Table 4-4. High-page register allocation (continued)

| Address | Register name | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x303E | ICS_OSCSC | OSCEN | — | OSCSTEN | OSCOS | — | RANGE | HGO | OSCINIT |
| 0x303F | Reserved | — | — | — | — | — | — | — | — |
| 0x3040 | PMC_SPMSC1 | LVWF | LVWACK | LVWIE | LVDRE | LVDSE | LVDE | BGBDS | BGBE |
| 0x3041 | PMC_SPMSC2 | — | LVDV | LVWV | | — | — | — | — |
| 0x3042-0x3049 | Reserved | — | — | — | — | — | — | — | — |
| 0x304A | SYS_ILLAH | Bit 15 | 14 | 13 | 12 | 11 | 19 | 9 | Bit 8 |
| 0x304B | SYS_ILLAL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x304C-0x304F | Reserved | — | — | — | — | — | — | — | — |
| 0x3050 | IPC_ILRS0 | ILR3 | | ILR2 | | ILR1 | | ILR0 | |
| 0x3051 | IPC_ILRS1 | ILR7 | | ILR6 | | ILR5 | | ILR4 | |
| 0x3052 | IPC_ILRS2 | ILR11 | | ILR10 | | ILR9 | | ILR8 | |
| 0x3053 | IPC_ILRS3 | ILR15 | | ILR14 | | ILR13 | | ILR12 | |
| 0x3054 | IPC_ILRS4 | ILR19 | | ILR18 | | ILR17 | | ILR16 | |
| 0x3055 | IPC_ILRS5 | ILR23 | | ILR22 | | ILR21 | | ILR20 | |
| 0x3056 | IPC_ILRS6 | ILR27 | | ILR26 | | ILR25 | | ILR24 | |
| 0x3057 | IPC_ILRS7 | ILR31 | | ILR30 | | ILR29 | | ILR28 | |
| 0x3058 | IPC_ILRS8 | ILR35 | | ILR34 | | ILR33 | | ILR32 | |
| 0x3059 | IPC_ILRS9 | ILR39 | | ILR38 | | ILR37 | | ILR36 | |
| 0x305A-0x305F | Reserved | — | — | — | — | — | — | — | — |
| 0x3060 | CRC_D0 | Bit 31 | 30 | 29 | 28 | 27 | 26 | 25 | Bit 24 |
| 0x3061 | CRC_D1 | Bit 23 | 22 | 21 | 20 | 19 | 18 | 17 | Bit 16 |
| 0x3062 | CRC_D2 | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x3063 | CRC_D3 | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x3064 | CRC_P0 | Bit 31 | 30 | 29 | 28 | 27 | 26 | 25 | Bit 24 |
| 0x3065 | CRC_P1 | Bit 23 | 22 | 21 | 20 | 19 | 18 | 17 | Bit 16 |
| 0x3066 | CRC_P2 | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x3067 | CRC_P3 | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x3068 | CRC_CTRL | TOT | | TOTR | | 0 | FXOR | WAS | TCRC |
| 0x3069 | Reserved | — | — | — | — | — | — | — | — |
| 0x306A | RTC_SC1 | RTIF | RTIE | — | RTCO | — | — | — | — |
| 0x306B | RTC_SC2 | RTCLKS | | — | — | — | RTCPS | | |
| 0x306C | RTC_MODH | MODH | | | | | | | |
| 0x306D | RTC_MODL | MODL | | | | | | | |
| 0x306E | RTC_CNTH | CNTH | | | | | | | |
| 0x306F | RTC_CNTL | CNTL | | | | | | | |
| 0x3070-0x307B | Reserved | — | — | — | — | — | — | — | — |
| 0x307C | KBI0_PE | KBIPE7 | KBIPE6 | KBIPE5 | KBIPE4 | KBIPE3 | KBIPE2 | KBIPE1 | KBIPE0 |

*Table continues on the next page...*

## Table 4-4. High-page register allocation (continued)

| Address | Register name | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x307D | KBI0_ES | KBEDG7 | KBEDG6 | KBEDG5 | KBEDG4 | KBEDG3 | KBEDG2 | KBEDG1 | KBEDG0 |
| 0x307E-0x307F | Reserved | — | — | — | — | — | — | — | — |
| 0x3080 | SCI0_BDH | LBKDIE | RXEDGIE | SBNS | SBR12 | SBR11 | SBR10 | SBR9 | SBR8 |
| 0x3081 | SCI0_BDL | SBR7 | SBR6 | SBR5 | SBR4 | SBR3 | SBR2 | SBR1 | SBR0 |
| 0x3082 | SCI0_C1 | LOOPS | SCISWAI | RSRC | M | WAKE | ILT | PE | PT |
| 0x3083 | SCI0_C2 | TIE | TCIE | RIE | ILIE | TE | RE | RWU | SBK |
| 0x3084 | SCI0_S1 | TDRE | TC | RDRF | IDLE | OR | NF | FE | PF |
| 0x3085 | SCI0_S2 | LBKDIF | RXEDGIF | — | RXINV | RWUID | BRK13 | LBKDE | RAF |
| 0x3086 | SCI0_C3 | R8 | T8 | TXDIR | TXINV | ORIE | NEIE | FEIE | PEIE |
| 0x3087 | SCI0_D | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 0x3088 | SCI1_BDH | LBKDIE | RXEDGIE | SBNS | SBR12 | SBR11 | SBR10 | SBR9 | SBR8 |
| 0x3089 | SCI1_BDL | SBR7 | SBR6 | SBR5 | SBR4 | SBR3 | SBR2 | SBR1 | SBR0 |
| 0x308A | SCI1_C1 | LOOPS | SCISWAI | RSRC | M | WAKE | ILT | PE | PT |
| 0x308B | SCI1_C2 | TIE | TCIE | RIE | ILIE | TE | RE | RWU | SBK |
| 0x308C | SCI1_S1 | TDRE | TC | RDRF | IDLE | OR | NF | FE | PF |
| 0x308D | SCI1_S2 | LBKDIF | RXEDGIF | — | RXINV | RWUID | BRK13 | LBKDE | RAF |
| 0x308E | SCI1_C3 | R8 | T8 | TXDIR | TXINV | ORIE | NEIE | FEIE | PEIE |
| 0x308F | SCI1_D | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 0x3090-0x3097 | Reserved | — | — | — | — | — | — | — | — |
| 0x3098-0x30AB | Reserved | — | — | — | — | — | — | — | — |
| 0x30AC | ADC_APCTL1 | ADPC7 | ADPC6 | ADPC5 | ADPC4 | ADPC3 | ADPC2 | ADPC1 | ADPC0 |
| 0x30AD | ADC_APCTL2 | — | — | — | — | ADPC11 | ADPC10 | ADPC9 | ADPC8 |
| 0x30AE | Reserved | — | — | — | — | — | — | — | — |
| 0x30AF | Reserved | — | — | — | — | — | — | — | — |
| 0x30B0 | PORT_PTAOE | PTAOE7 | PTAOE6 | PTAOE5 | PTAOE4 | PTAOE3 | PTAOE2 | PTAOE1 | PTAOE0 |
| 0x30B1 | PORT_PTBOE | PTBOE7 | PTBOE6 | PTBOE5 | PTBOE4 | PTBOE3 | PTBOE2 | PTBOE1 | PTBOE0 |
| 0x30B2 | PORT_PTCOE | PTCOE7 | PTCOE6 | PTCOE5 | PTCOE4 | PTCOE3 | PTCOE2 | PTCOE1 | PTCOE0 |
| 0x30B3 | PORT_PTDOE | PTDOE7 | PTDOE6 | — | — | — | PTDOE2 | PTDOE1 | PTDOE0 |
| 0x30B4 | PORT_PTEOE | — | — | — | PTEOE4 | — | — | — | — |
| 0x30B5-0x30B7 | Reserved | — | — | — | — | — | — | — | — |
| 0x30B8 | PORT_PTAIE | PTAIE7 | PTAIE6 | PTAIE5 | — | PTAIE3 | PTAIE2 | PTAIE1 | PTAIE0 |
| 0x30B9 | PORT_PTBIE | PTBIE7 | PTBIE6 | PTBIE5 | PTBIE4 | PTBIE3 | PTBIE2 | PTBIE1 | PTBIE0 |
| 0x30BA | PORT_PTCIE | PTCIE7 | PTCIE6 | PTCIE5 | PTCIE4 | PTCIE3 | PTCIE2 | PTCIE1 | PTCIE0 |

*Table continues on the next page...*

## Table 4-4. High-page register allocation (continued)

| Address | Register name | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x30BB | PORT_PTDIE | PTDIE7 | PTDIE6 | — | — | — | PTDIE2 | PTDIE1 | PTDIE0 |
| 0x30BC | PORT_PTEIE | — | — | — | PTEIE4 | — | — | — | — |
| 0x30BD-0x30BF | Reserved | — | — | — | — | — | — | — | — |
| 0x30C0 | FTM2_SC | TOF | TOIE | CPWMS | CLKS1 | CLKS0 | PS2 | PS1 | PS0 |
| 0x30C1 | FTM2_CNTH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x30C2 | FTM2_CNTL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x30C3 | FTM2_MODH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x30C4 | FTM2_MODL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x30C5 | FTM2_C0SC | CHF | CHIE | MSB | MSA | ELSB | ELSA | — | — |
| 0x30C6 | FTM2_C0VH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x30C7 | FTM2_C0VL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x30C8 | FTM2_C1SC | CHF | CHIE | MSB | MSA | ELSB | ELSA | — | — |
| 0x30C9 | FTM2_C1VH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x30CA | FTM2_C1VL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x30CB | FTM2_C2SC | CHF | CHIE | MSB | MSA | ELSB | ELSA | — | — |
| 0x30CC | FTM2_C2VH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x30CD | FTM2_C2VL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x30CE | FTM2_C3SC | CHF | CHIE | MSB | MSA | ELSB | ELSA | — | — |
| 0x30CF | FTM2_C3VH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x30D0 | FTM2_C3VL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x30D1 | FTM2_C4SC | CHF | CHIE | MSB | MSA | ELSB | ELSA | — | — |
| 0x30D2 | FTM2_C4VH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x30D3 | FTM2_C4VL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x30D4 | FTM2_C5SC | CHF | CHIE | MSB | MSA | ELSB | ELSA | — | — |
| 0x30D5 | FTM2_C5VH | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| 0x30D6 | FTM2_C5VL | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| 0x30D7-0x30EB | Reserved | — | — | — | — | — | — | — | — |
| 0x30EC | PORT_IOFLT0 | FLTD | | FLTC | | FLTB | | FLTA | |
| 0x30ED | PORT_IOFLT1 | — | — | — | — | — | — | FLTE | |
| 0x30EE | PORT_IOFLT2 | — | — | — | — | FLTKBI0 | | FLTRST | |
| 0x30EF | PORT_FCLKDIV | FLTDIV3 | | | FLTDIV2 | | | FLTDIV1 | |
| 0x30F0 | PORT_PTAPE | PTAPE7 | PTAPE6 | PTAPE5 | — | PTAPE3 | PTAPE2 | PTAPE1 | PTAPE0 |
| 0x30F1 | PORT_PTBPE | PTBPE7 | PTBPE6 | PTBPE5 | PTBPE4 | PTBPE3 | PTBPE2 | PTBPE1 | PTBPE0 |
| 0x30F2 | PORT_PTCPE | PTCPE7 | PTCPE6 | PTCPE5 | PTCPE4 | PTCPE3 | PTCPE2 | PTCPE1 | PTCPE0 |
| 0x30F3 | PORT_PTDPE | PTDPE7 | PTDPE6 | — | — | — | PTDPE2 | PTDPE1 | PTDPE0 |
| 0x30F4 | PORT_PTEPE | — | — | — | PTEPE4 | — | — | — | — |
| 0x30F5-0x30F7 | Reserved | — | — | — | — | — | — | — | — |
| 0x30F8 | SYS_UUID1 | ID63 | ID62 | ID61 | ID60 | ID59 | ID58 | ID57 | ID56 |
| 0x30F9 | SYS_UUID2 | ID55 | ID54 | ID53 | ID52 | ID51 | ID50 | ID49 | ID48 |

*Table continues on the next page...*

**Table 4-4. High-page register allocation (continued)**

| Address | Register name | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x30FA | SYS_UUID3 | ID47 | ID46 | ID45 | ID44 | ID43 | ID42 | ID41 | ID40 |
| 0x30FB | SYS_UUID4 | ID39 | ID38 | ID37 | ID36 | ID35 | ID34 | ID33 | ID32 |
| 0x30FC | SYS_UUID5 | ID31 | ID30 | ID29 | ID28 | ID27 | ID26 | ID25 | ID24 |
| 0x30FD | SYS_UUID6 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 | ID17 | ID16 |
| 0x30FE | SYS_UUID7 | ID15 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 |
| 0x30FF | SYS_UUID8 | ID7 | ID6 | ID5 | ID4 | ID3 | ID2 | ID1 | ID0 |

Several reserved flash memory locations, shown in the following table, are used for storing values used by several registers. These registers include an 8-byte backdoor key, NV_BACKKEY, which can be used to gain access to secure memory resources. During reset events, the contents of NVPROT and NVOPT in the reserved flash memory are transferred into corresponding FPROT and FOPT registers in the high-page registers area to control security and block protection options.

**Table 4-5. Reserved flash memory addresses**

| Address | Register Name | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0xFF70 | NV_BACKKEY0 | BACKKEY0 | | | | | | | |
| 0xFF71 | NV_BACKKEY1 | BACKKEY1 | | | | | | | |
| 0xFF72 | NV_BACKKEY2 | BACKKEY2 | | | | | | | |
| 0xFF73 | NV_BACKKEY3 | BACKKEY3 | | | | | | | |
| 0xFF74 | NV_BACKKEY4 | BACKKEY4 | | | | | | | |
| 0xFF75 | NV_BACKKEY5 | BACKKEY5 | | | | | | | |
| 0xFF76 | NV_BACKKEY6 | BACKKEY6 | | | | | | | |
| 0xFF77 | NV_BACKKEY7 | BACKKEY7 | | | | | | | |
| 0xFF78 | Reserved | — | — | — | — | — | — | — | — |
| 0xFF79 | Reserved | — | — | — | — | — | — | — | — |
| 0xFF7A | Reserved | — | — | — | — | — | — | — | — |
| 0xFF7B | Reserved | — | — | — | — | — | — | — | — |
| 0xFF7C | NV_FPROT | FPOPEN | — | FPHDIS | FPHS | | — | — | — |
| 0xFF7D | NV_EEPROT | DPOPEN | — | | | | DPS | | |
| 0xFF7E | NV_FOPT | NV | | | | | | | |
| 0xFF7F | NV_FSEC | KEYEN | | 1 | 1 | 1 | 1 | SEC | |

The 8-byte comparison key can be used to temporarily disengage memory security provided the key enable field, NV_FSEC[KEYEN], is 10b. This key mechanism can be accessed only through user code running in secure memory. A security key cannot be entered directly through background debug commands. This security key can be disabled

completely by programming the NV_FSEC[KEYEN] bit to 0. If the security key is disabled, the only way to disengage security is by mass erasing the flash if needed, normally through the background debug interface and verifying that flash is blank. To avoid returning to secure mode after the next reset, program the security bits, NV_FSEC[SEC], to the unsecured state (10b).

## 4.4 Random-access memory (RAM)

This section describes the 2048 bytes of RAM (random-access memory).

These devices include static RAM. The locations in RAM below 0x0100 can be accessed using the more efficient direct addressing mode. Any single bit in this area can be accessed with the bit manipulation instructions (BCLR, BSET, BRCLR, and BRSET).

The RAM retains data when the MCU is in low-power wait, or stop3 mode. At power-on, the contents of RAM are uninitialized. RAM data is unaffected by any reset provided that the supply voltage does not drop below the minimum value for RAM retention.

For compatibility with older M68HC05 MCUs, the HCS08 resets the stack pointer to 0x00FF. In this series, re-initialize the stack pointer to the top of the RAM so that the direct-page RAM can be used for frequently accessed RAM variables and bit-addressable program variables. Include the following 2-instruction sequence in your reset initialization routine (where RamLast is equated to the highest address of the RAM in the equate file).

```
LDHX     #RamLast+1    ;point one past RAM
TXS                    ;SP<-(H:X-1)
```

When security is enabled, the RAM is considered a secure memory resource and is not accessible through BDM or code executing from non-secure memory.

## 4.5 Flash and EEPROM

### 4.5.1 Overview

This device includes various configuration of flash and EEPROM. The controller for flash and EEPROM is ideal for single-supply applications for field programming without external high voltage sources for program or erase operations.

The flash memory is ideal for single-supply applications that allow for field reprogramming without requiring external high voltage sources for program or erase operations. The flash module includes a memory controller that executes commands to

modify flash memory contents. The user interface to the memory controller consists of the indexed flash common command object (FCCOB) register, which is written to with the command, global address, data, and any required command parameters. The memory controller must complete the execution of a command before the FCCOB register is written to with a new command.

**CAUTION**

A flash byte or longword must be in the erased state before being programmed. Cumulative programming of bits within a flash byte or longword is not allowed.

The flash memory is read as bytes. Read access time is one bus cycle for bytes. For flash memory, an erased bit reads 1 and a programmed bit reads 0. It is possible to read from flash memory while commands are being executed on EEPROM memory. It is not possible to read from EEPROM memory while a command (erase/program) is executing on flash memory. Simultaneous EEPROM memory are implemented with error correction codes (ECC) that can resolve single bit faults and detect double bit faults.

The following figure shows the block diagram of the flash and EEPROM module.

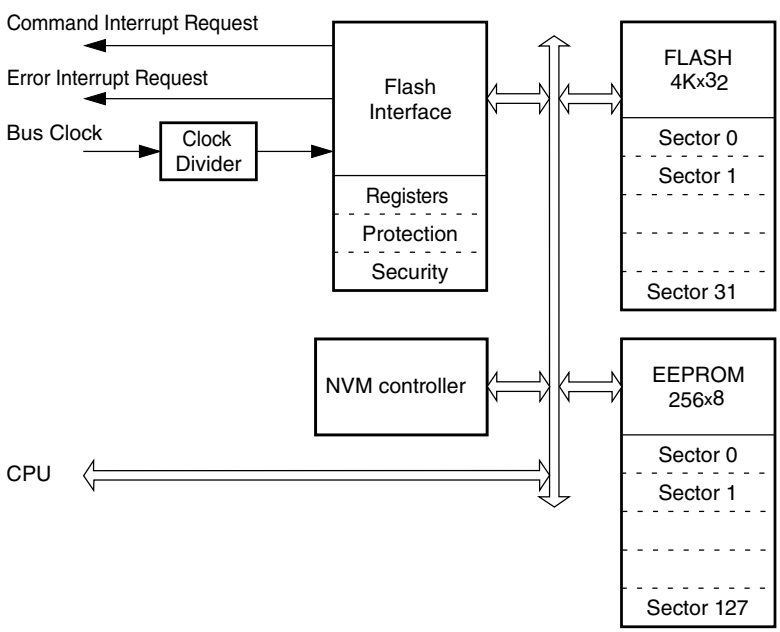



**Figure 4-2. Flash and EEPROM block diagram**

Flash features:

- 16 KB of flash memory composed of one 16 KB flash block divided into 32 sectors of 512 bytes

- Automated program and erase algorithm with verification
- Fast sector erase and longword program operation
- Ability to read the flash memory while programming a word in the EEPROM memory
- Flexible protection scheme to prevent accidental program or erase of flash memory

EEPROM features:

- 256 bytes of EEPROM memory composed of one 256 byte EEPROM block divided into 128 sectors of 2 bytes
- Single bit fault correction and double bit fault detection within a word during read operations
- Automated program and erase algorithm with verification and generation of ECC parity bits
- Fast sector erase and byte program operation
- Protection scheme to prevent accidental program or erase of EEPROM memory
- Ability to program up to four bytes in a burst sequence

Other features

- No external high-voltage power supply required for flash memory program and erase operations
- Interrupt generation on flash command completion and flash error detection
- Security mechanism to prevent unauthorized access to the flash memory

## 4.5.2 Function descriptions

### 4.5.2.1 Modes of operation

The flash and EEPROM module provides the normal user mode of operation. The operating mode is determined by module-level inputs and affects the FCLKDIV, FCNFG, and EEPROT registers.

#### 4.5.2.1.1 Wait mode

The flash and EEPROM module is not affected if the MCU enters wait mode. The flash module can recover the MCU from wait via the CCIF interrupt. See Flash and EEPROM interrupts.

#### 4.5.2.1.2 Stop mode

If a flash and EEPROM command is active, that is, FSTAT[CCIF] = 0, when the MCU requests stop mode, the current NVM operation will be completed before the MCU is allowed to enter stop mode.

### 4.5.2.2 Flash and EEPROM memory map

The MCU places the flash memory between global address 0x0000 and 0xFFFF as shown in the following table. Not all flash are available to users because some addresses are overlapped with RAM, EEPROM, and registers.

MC9S08PL16 contains a piece of 16 KB flash that is fully available for users. This flash block is divided into 32 sectors of 512 bytes. MC9S08PL8 contains a piece of 8 KB flash that is fully available for users. This flash block is divided into 16 sectors of 512 bytes.

**Table 4-6. Flash memory addressing**

| Device | Global address | Size (Bytes) | Description | User availability |
|---|---|---|---|---|
| MC9S08PL16 | 0xC000 — 0xFFFF | 16 KB | Flash block contains flash configuration field | Sector [0:31]: fully available |
| MC9S08PL8 | 0xE000 — 0xFFFF | 8 KB | Flash block contains flash configuration field | Sector [0:15]: fully available |

### 4.5.2.3 Flash and EEPROM initialization after system reset

On each system reset, the flash and EEPROM module executes an initialization sequence that establishes initial values for the flash and EEPROM block configuration parameters, the FPROT and EEPROT protection registers, and the FOPT and FSEC registers. The initialization routine reverts to built-in default values that leave the module in a fully protected and secured state if errors are encountered during execution of the reset sequence. If a double bit fault is detected during the reset sequence, both FSTAT[MGSTAT] bits will be set.

FSTAT[CCIF] is cleared throughout the initialization sequence. The NVM module holds off all CPU access for a portion of the initialization sequence. Flash and EEPROM reads are allowed after the hold is removed. Completion of the initialization sequence is marked by setting FSTAT[CCIF] high, which enables user commands.

If a reset occurs while any flash or EEPROM command is in progress, that command will be immediately aborted. The state of the word being programmed or the sector/block being erased is not guaranteed.

#### 4.5.2.4 Flash and EEPROM command operations

Flash and EEPROM command operations are used to modify flash and EEPROM memory contents.

The command operations contain three steps:

1. Configure the clock for flash or EEPROM program and erase command operations.
2. Use command write sequence to set flash and EEPROM command parameters and launch execution.
3. Execute valid flash and EEPROM commands according to MCU functional mode and MCU security state.

The figure below shows a general flowchart of the flash or EEPROM command write sequence.

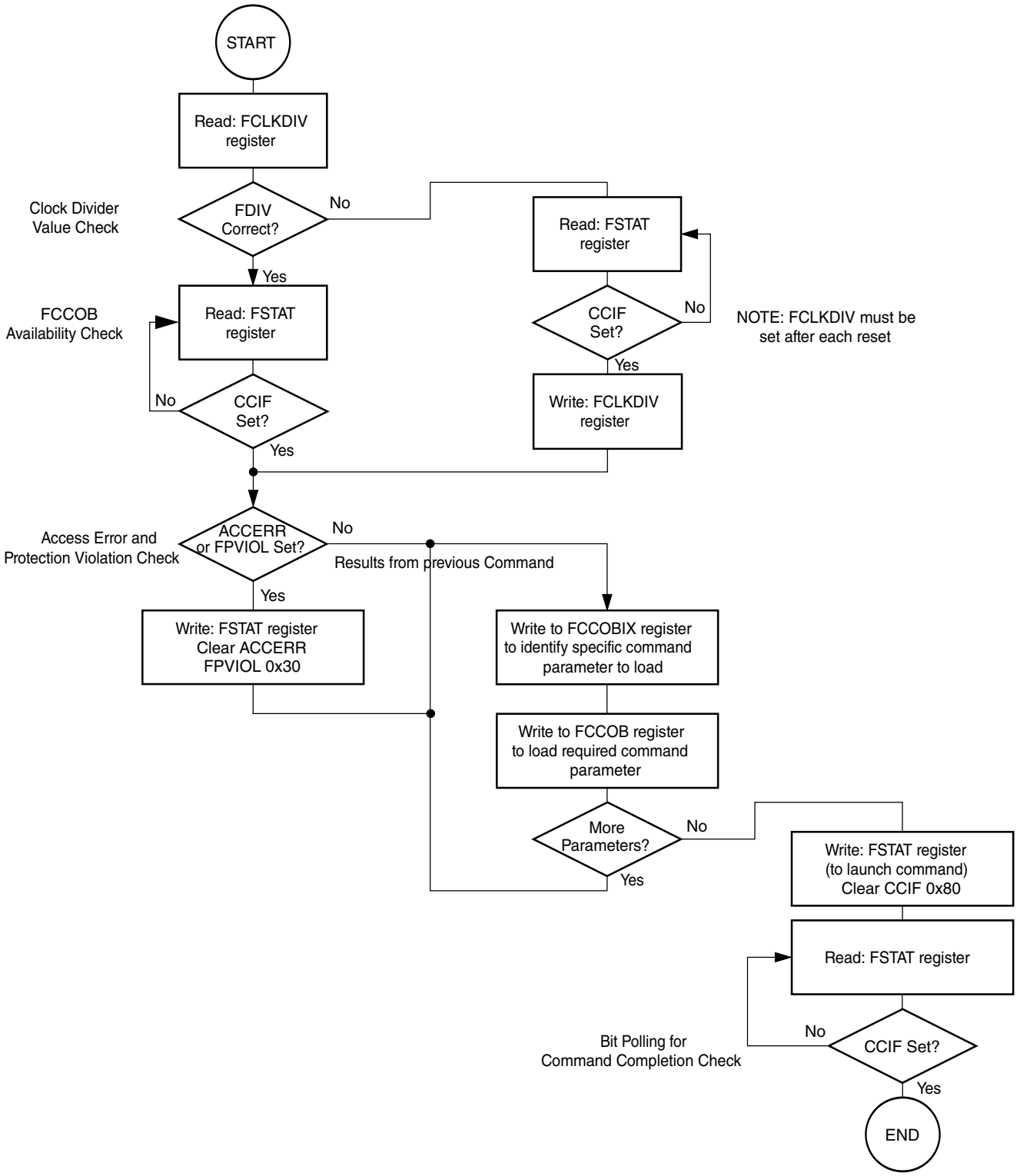



**Figure 4-3. Generic flash and EEPROM command write sequence flowchart**

#### 4.5.2.4.1 Writing the FCLKDIV register

Prior to issuing any flash and EEPROM program or erase command after a reset, the user is required to write the FCLKDIV register to divide BUSCLK down to a target FCLK of 1MHz. The following table shows recommended values for the FDIV field based on BUSCLK frequency.

**Table 4-7. FDIV values for various BUSCLK frequencies**

| BUSCLK frequency (MHz) | | FDIV[5:0] |
|---|---|---|
| MIN[1] | MAX[2] | |
| 1.0 | 1.6 | 0x00 |
| 1.6 | 2.6 | 0x01 |
| 2.6 | 3.6 | 0x02 |
| 3.6 | 4.6 | 0x03 |
| 4.6 | 5.6 | 0x04 |
| 5.6 | 6.6 | 0x05 |
| 6.6 | 7.6 | 0x06 |
| 7.6 | 8.6 | 0x07 |
| 8.6 | 9.6 | 0x08 |
| 9.6 | 10.6 | 0x09 |
| 10.6 | 11.6 | 0x0A |
| 11.6 | 12.6 | 0x0B |
| 12.6 | 13.6 | 0x0C |
| 13.6 | 14.6 | 0x0D |
| 14.6 | 15.6 | 0x0E |
| 15.6 | 16.6 | 0x0F |
| 16.6 | 17.6 | 0x10 |
| 17.6 | 18.6 | 0x11 |
| 18.6 | 19.6 | 0x12 |
| 19.6 | 20.0 | 0x13 |

1. BUSCLK is greater than this value
2. BUSCLK is less than or equal to this value

**CAUTION**

Programming or erasing the flash and EEPROM memory cannot be performed if the bus clock runs at less than 0.8 MHz. Setting FCLKDIV[FDIV] too high can destroy the flash and EEPROM memory due to overstress. Setting FCLKDIV[FDIV] too low can result in incomplete programming or erasure of the flash and EEPROM memory cells.

When the FCLKDIV register is written, the FCLKDIV[FDIVLD] bit is set automatically. If the FCLKDIV[FDIVLD] bit is 0, the FCLKDIV register has not been written since the last reset. If the FCLKDIV register has not been written, any flash and EEPROM program or erase command loaded during a command write sequence will not execute and the FSTAT[ACCERR] bit will be set.

#### 4.5.2.4.2 Command write sequence

The memory controller will launch all valid flash and EEPROM commands entered using a command write sequence.

Before launching a command, the FSTAT[ACCERR] and FSTAT[FPVIOL] bits must be clear and the FSTAT[CCIF] flag will be tested to determine the status of the current command write sequence. If FSTAT[CCIF] is 0, indicating that the previous command write sequence is still active, a new command write sequence cannot be started and all writes to the FCCOB register are ignored.

The FCCOB parameter fields must be loaded with all required parameters for the flash and EEPROM command being executed. Access to the FCCOB parameter fields is controlled via FCCOBIX[CCOBIX] bits.

Flash and EEPROM command mode uses the indexed FCCOB register to provide a command code and its relevant parameters to the memory controller. First, the user must set up all required FCCOB field. Then they can initiate the command's execution by writing a 1 to the FSTAT[CCIF] bit. This action clears the CCIF command completion flag to 0. When the user clears the FSTAT[CCIF] bit all FCCOB parameter field are locked and cannot be changed by the user until the command completes (evidenced by the memory controller returning FSTAT[CCIF] to1). Some commands return information to the FCCOB register array.

The generic format for the FCCOB parameter fields in flash and EEPROM command mode is shown in the following table. The return values are available for reading after the FSTAT[CCIF] flag has been returned to 1 by the memory controller. Writes to the unimplemented parameter fields, FCCOBIX[CCOBIX] =110b and FCCOBIX[CCOBIX] = 111b, are ignored with read from these fields returning 0x0000.

The table below shows the generic flash command format. The high byte of the first word in the CCOB array contains the command code, followed by the parameters for this specific flash command. For details on the FCCOB settings required by each command, see the flash command descriptions in Flash and EEPROM command summary .

**Table 4-8. FCCOB – flash and EEPROM command mode typical usage**

| CCOBIX[2:0] | Byte | FCCOB parameter fields in flash and EEPROM command mode |
|---|---|---|
| 000 | HI | FCMD[7:0] defining flash command |
| | LO | Global address [23:16] |
| 001 | HI | Global address [15:8] |
| | LO | Global address [7:0] |
| 010 | HI | Data 0 [15:8] |
| | LO | Data 0 [7:0] |
| 011 | HI | Data 1 [15:8] |
| | LO | Data 1 [7:0] |
| 100 | HI | Data 2 [15:8] |
| | LO | Data 2 [7:0] |
| 101 | HI | Data 3 [15:8] |
| | LO | Data 3 [7:0] |

The contents of the FCCOB parameter fields are transferred to the memory controller when the user clears the FSTAT[CCIF] command completion flag by writing 1. The CCIF flag will remain clear until the flash and EEPROM command has completed. Upon completion, the memory controller will return FSTAT[CCIF] to 1 and the FCCOB register will be used to communicate any results.

The following table presents the valid flash and EEPROM commands, as enabled by the combination of the functional MCU mode with the MCU security state of unsecured or secured.

MCU secured state is selected by NVM_FSEC[SEC].

**Table 4-9. Flash and EEPROM commands by mode and security state**

| FCMD | Command | Unsecured | Secured |
|---|---|---|---|
| | | U[1] | U[2] |
| 0x01 | Erase verify all blocks | * | * |
| 0x02 | Erase verify block | * | * |
| 0x03 | Erase verify flash section | * | N/A |
| 0x04 | Read once | * | N/A |
| 0x06 | Program flash | * | N/A |
| 0x07 | Program once | * | N/A |
| 0x08 | Erase all block | * | * |
| 0x09 | Erase flash block | * | * |
| 0x0A | Erase flash sector | * | N/A |
| 0x0B | Unsecure NVM | N/A | * |
| 0x0C | Verify backdoor access key | * | * |

*Table continues on the next page...*

**Table 4-9. Flash and EEPROM commands by mode and security state (continued)**

| FCMD | Command | Unsecured | Secured |
|---|---|---|---|
| | | U[1] | U[2] |
| 0x0D | Set user margin level | * | N/A |
| 0x10 | Erase verify EEPROM section | * | * |
| 0x11 | Program EEPROM | * | N/A |
| 0x12 | Erase EEPROM sector | * | N/A |

1. Unsecured User mode
2. Secured User mode

### 4.5.2.5 Flash and EEPROM interrupts

The flash and EEPROM module can generate an interrupt when a flash command operation has completed or when a flash and EEPROM command operation has detected an ECC fault.

**Table 4-10. Flash interrupt source**

| Interrupt source | Interrupt flag | Local enable | Global (CCR) mask |
|---|---|---|---|
| Flash and EEPROM command complete | CCIF (FSTAT register) | CCIE (FCNFG register) | I Bit |
| ECC double bit fault on flash and EEPROM read | DFDIF (FERSTAT register) | DFDIE (FERCNFG register) | I Bit |
| ECC single bit fault on flash and EEPROM read | SFDIF (FERSTAT register) | SFDIE (FERCNFG register) | I Bit |

#### 4.5.2.5.1 Description of flash and EEPROM interrupt operation

The flash module uses the FSTAT[CCIF] flag in combination with the FCNFG[CCIE] interrupt enable bit to generate the flash command interrupt request. The flash module uses the DFDIF and SFDIF flags in combination with the FERSTAT[DFDIE] and FERSTAT[SFDIE] interrupt enable bits to generate the flash error interrupt request.

The logic used for generating the flash module interrupts is shown in the following figure.

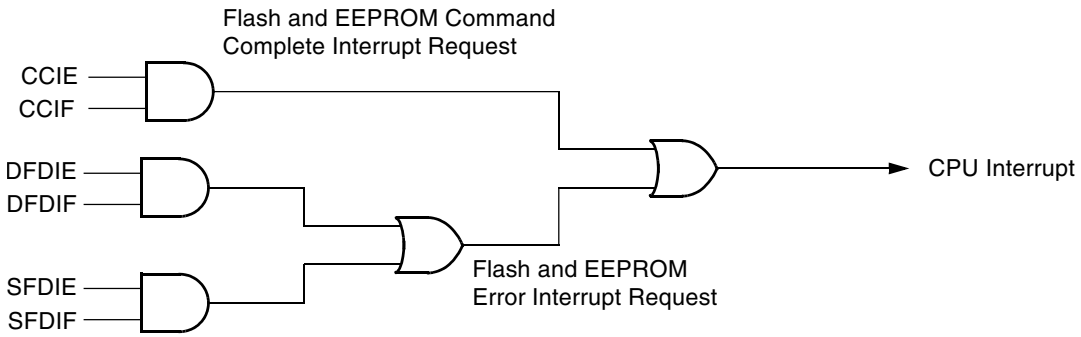



**Figure 4-4. Flash and EEPROM module interrupts implementation**

### 4.5.2.6 Protection

The FPROT register can be set to protect regions in the flash memory from accidental programming or erasing. Two separate memory regions, one growing downward from global address 0xFFFF in the flash memory, called the higher region; and the remaining addresses in the flash memory, can be activated for protection. The flash memory addresses covered by these protectable regions are shown in the flash memory map. The higher address region is mainly targeted to hold the boot loader code because it covers the vector space.

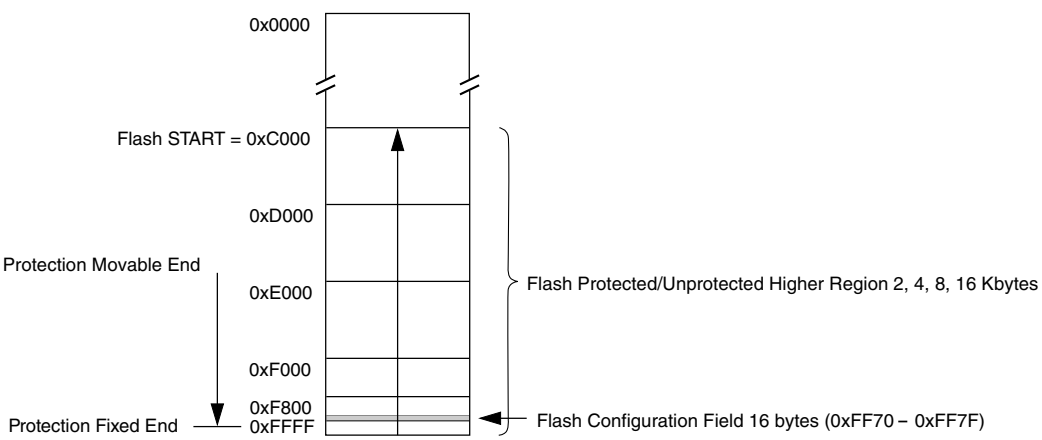



**Figure 4-5. Flash protection memory map**

Default protection settings as well as security information that allows the MCU to restrict access to the flash module are stored in the flash configuration field as described in the table below.

**Table 4-11. Flash configuration field**

| Global address | Size (Bytes) | Description |
|---|---|---|
| 0xFF70 — 0xFF77[1] | 8 | Backdoor comparison key. See Verify backdoor access key command and Unsecuring the MCU using backdoor key access. |

*Table continues on the next page...*

**Table 4-11. Flash configuration field (continued)**

| Global address | Size (Bytes) | Description |
|---|---|---|
| 0xFF78 — 0xFF7B | 4 | Reserved |
| 0xFF7C[1] | 1 | Flash protection byte |
| 0xFF7D[1] | 1 | EEPROM protection byte |
| 0xFF7E[1] | 1 | Flash nonvolatile byte |
| 0xFF7F[1] | 1 | Flash security byte |

1. 0xFF78–0xFF7F for a flash phrase and must be programmed in a single command write sequence. Each byte in the 0xFF78-0xFF7B reserved field must be programmed to 0xFF.

The flash and EEPROM module provides protection to the MCU. During the reset sequence, the FPROT register is loaded with the contents of the flash protection byte in the flash configuration field at global address 0xFF7C in flash memory. The protection functions depend on the configuration of bit settings in FPORT register.

**Table 4-12. Flash protection function**

| FPOPEN | FPHDIS | Function[1] |
|---|---|---|
| 1 | 1 | No flash protection |
| 1 | 0 | Protected high range |
| 0 | 1 | Full flash memory protected |
| 0 | 0 | Unprotected high range |

1. For range sizes, see Table 4.

The flash protection scheme can be used by applications requiring reprogramming in single chip mode while providing as much protection as possible if reprogramming is not required.

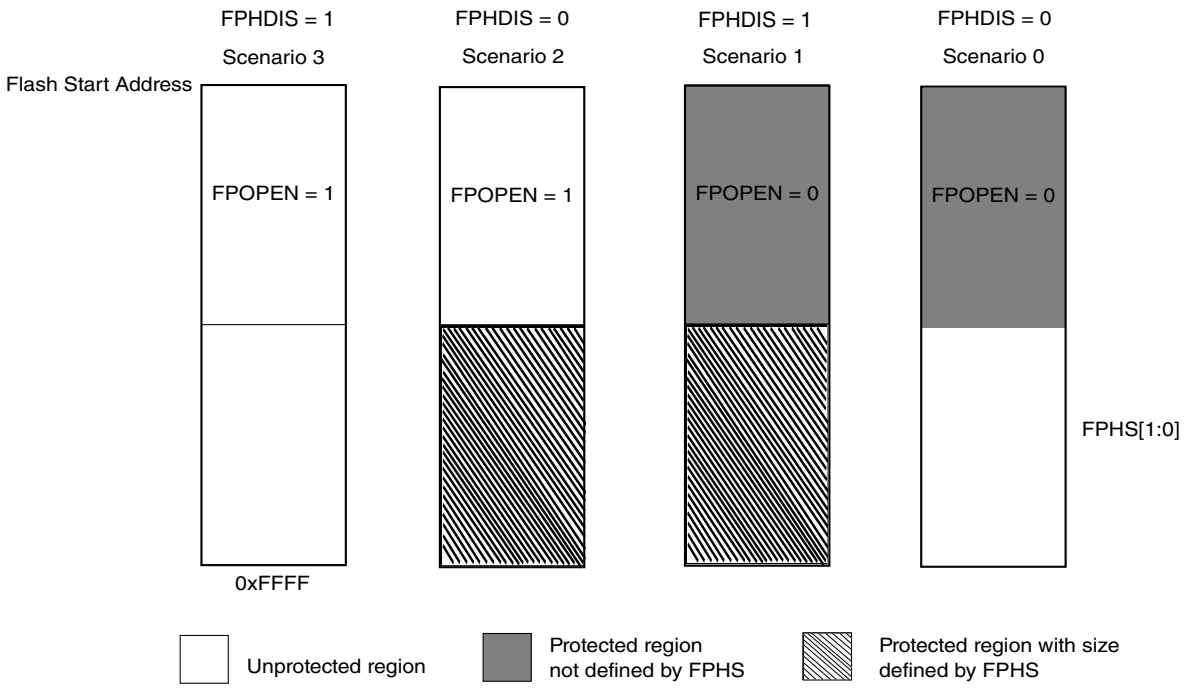



**Figure 4-6. Flash protection scenarios**

The general guideline is that flash protection can only be added and not removed. The following table specifies all valid transitions between flash protection scenarios. Any attempt to write an invalid scenario to the FPROT register will be ignored. The contents of the FPROT register reflect the active protection scenario. See the FPROT[FPHS] and FPROT[FPLS] bit descriptions for additional restrictions.

**Table 4-13. Flash protection scenario transitions**

| From protection scenario | To protection scenario | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | × | × | | |
| 1 | | × | | |
| 2 | | × | × | |
| 3 | × | × | × | × |

The flash protection address range is listed in the following two tables regarding the scenarios in the table above.

**Table 4-14. Flash protection higher address range**

| FPHS[1:0] | Global address range | Protected size |
|---|---|---|
| 00 | 0xF800 – 0xFFFF | 2 Kbytes |
| 01 | 0xF000 – 0xFFFF | 4 Kbytes |
| 10 | 0xE000 – 0xFFFF | 8 Kbytes |
| 11 | 0xC000 – 0xFFFF | 16 Kbytes |

During the reset sequence, fields NVM_EEPROT[DPOPEN] and NVM_EEPROT[DPS] are loaded with the contents of the EEPROM protection byte in the flash configuration field at global address 0xFF7D located in flash memory. EEPROM protection address range is specified by the NVM_EEPROT[DPS].

**Table 4-15. EEPROM protection address range**

| DPS[2:0] | Global address range | Protected size |
|---|---|---|
| 000 | 0x3100 – 0x311F | 32 bytes |
| 001 | 0x3100 – 0x313F | 64 bytes |
| 010 | 0x3100 – 0x315F | 96 bytes |
| 011 | 0x3100 – 0x317F | 128 bytes |
| 100 | 0x3100 – 0x319F | 160 bytes |
| 101 | 0x3100 – 0x31BF | 192 bytes |
| 110 | 0x3100 – 0x31DF | 224 bytes |
| 111 | 0x3100 – 0x31FF | 256 bytes |

All possible flash protection scenarios are shown in Figure 4-6. Although the protection scheme is loaded from the flash memory at global address 0xFF7C during the reset sequence, it can be changed by the user.

### 4.5.2.7 Security

The flash and EEPROM module provides security information to the MCU. The flash security state is defined by the NVM_FSEC[SEC] bits. During reset, the flash module initializes the NVM_FSEC register using data read from the security byte of the flash and EEPROM configuration field at global address 0xFF7F. The security state out of reset can be permanently changed by programming the security byte, assuming that the MCU is starting from a mode where the necessary flash and EEPROM erase and program commands are available and that the upper region of the flash is unprotected. If the flash security byte is successfully programmed, its new value will take affect after the next MCU reset.

The following subsections describe these security-related subjects:

- Unsecuring the MCU using backdoor key access
- Unsecuring the MCU using BDM
- Mode and security effects on flash and EEPROM command availability

#### 4.5.2.7.1 Unsecuring the MCU using backdoor key access

The MCU may be unsecured by using the backdoor key access feature which requires knowledge of the contents of the backdoor keys, which are four 16-bit words programmed at addresses 0xFF70–0xFF77. If the KEYEN[1:0] bits are in the enabled state, the verify backdoor access key command – see Verify backdoor access key command, allows the user to present four prospective keys for comparison to the keys stored in the flash and EEPROM memory via the memory controller. If the keys presented in the verify backdoor access key command match the backdoor keys stored in the flash and EEPROM memory, the FSEC[SEC] bits will be changed to unsecure the MCU. Key values of 0x0000 and 0xFFFF are not permitted as backdoor keys. While the Verify Backdoor Access Key command is active, flash memory and EEPROM memory will not be available for read access and will return invalid data.

The user code stored in the flash memory must have a method of receiving the backdoor keys from an external stimulus. This external stimulus would typically be through one of the on-chip serial ports.

If the KEYEN[1:0] bits are in the enabled state, the MCU can be unsecured by the backdoor key access sequence described below:

1. Follow the command sequence for the verify backdoor access key command as explained in Verify backdoor access key command.
2. If the verify backdoor access key command is successful, the MCU is unsecured and the FSEC[SEC] bits are forced to the unsecure state of 10.

The verify backdoor access key command is monitored by the memory controller and an illegal key will prohibit future use of the verify backdoor access key command. A reset of the MCU is the only method to re-enable the verify backdoor access key command. The security as defined in the flash and EEPROM security byte (0xFF7F) is not changed by using the verify backdoor access key command sequence. The backdoor keys stored in addresses 0xFF70–0xFF77 are unaffected by the verify backdoor access key command sequence. The verify backdoor access key command sequence has no effect on the program and erase protections defined in the flash and EEPROM protection register, FPORT.

After the backdoor keys have been correctly matched, the MCU will be unsecured. After the MCU is unsecured, the sector containing the flash and EEPROM security byte can be erased and the flash and EEPROM security byte can be reprogrammed to the unsecure state, if desired. In the unsecure state, the user has full control of the contents of the backdoor keys by programming addresses 0xFF70–0xFF77 in the flash configuration field.

#### 4.5.2.7.2 Unsecuring the MCU using BDM

A secured MCU can be unsecured by using the following method to erase the flash and EEPROM memory:

1. Reset the MCU.
2. Set FCDIV register as described in Writing the FCLKDIV register.
3. Configure registers NVM_FERSTAT and NVM_FPROT to disable protection in the flash and EEPROM memory.
4. Execute the erase all blocks command write sequence to erase the flash and EEPROM memory. Alternately, the unsecure NVM command can be executed.

   If the flash and EEPROM memory are verified as erased, the MCU will be unsecured. All BDM. commands will now be enabled and the flash security byte may be programmed to the unsecure state by continuing with the steps that follow.
5. Execute the program flash command write sequence to program the flash security byte to the unsecured state.
6. Reset the MCU.

#### 4.5.2.7.3 Mode and security effects on flash and EEPROM command availability

The availability of flash and EEPROM module commands depends on the MCU operating mode and security state as shown in Table 4-9.

### 4.5.2.8 Flash and EEPROM commands

#### 4.5.2.8.1 Flash commands

The following table summarizes the valid flash commands as well as the effects of the commands on the flash block and other resources within the flash and EEPROM module.

**Table 4-16. Flash commands**

| FCMD | Command | Function on flash memory |
|---|---|---|
| 0x01 | Erase verify all blocks | Verify that all flash (and EEPROM) blocks are erased |
| 0x02 | Erase verify block | Verify that a flash block is erased |
| 0x03 | Erase verify flash section | Verify that a given number of words starting at the address provided are erased |
| 0x04 | Read Once | Read a dedicated 64 byte field in the nonvolatile information register in flash block that was previously programmed using the program once command |

*Table continues on the next page...*

**Table 4-16. Flash commands (continued)**

| FCMD | Command | Function on flash memory |
|---|---|---|
| 0x06 | Program flash | Program up to two longwords in a flash block |
| 0x07 | Program once | Program a dedicated 64 byte field in the nonvolatile information register in flash block that is allowed to be programmed only once |
| 0x08 | Erase all block | Erase all flash and EEPROM blocks<br>An erase of all flash blocks is possible only when the FPROT[FPHDIS], and FPROT[FPOEN] bits and the EEPROT[DPOPEN] bit are set prior to launching the command |
| 0x09 | Erase flash block | Erase a flash or EEPROM block<br>An erase of the full flash block is possible only when FPROT[FPHDIS], and FPROT[FPOEN] bits are set prior to launching the command |
| 0x0A | Erase flash sector | Erase all bytes in a flash sector |
| 0x0B | Unsecure flash | Supports a method of releasing MCU security by erasing all flash (and EEPROM) blocks and verifying that all flash (and EEPROM) blocks are erased |
| 0x0C | Verify backdoor access key | Supports a method of releasing MCU security by verifying a set of security keys |
| 0x0D | Set user margin level | Specifies a user margin read level for all flash blocks |

### 4.5.2.8.2 EEPROM commands

The following table summarizes the valid EEPROM commands along with the effects of the commands on the EEPROM block.

**Table 4-17. EEPROM commands**

| FCMD | Command | Function on flash memory |
|---|---|---|
| 0x01 | Erase verify all blocks | Verify that all EEPROM (and flash) blocks are erased. |
| 0x02 | Erase verify block | Verify that an EEPROM block is erased. |
| 0x08 | Erase all block | Erase all EEPROM and flash blocks<br>An erase of all EEPROM blocks is possible only when the FPROT[FPHDIS], and FPROT[FPOEN] bits and the DPOPEN bit in the EEPORT register are set prior to launching the command. |
| 0x09 | Erase EEPROM Block | Erase a EEPROM and flash block<br>An erase of the full flash block is possible only when FPROT[FPHDIS] and FPROT[FPOPEN] bits are set prior to launching the command. |
| 0x0B | Unsecure EEPROM | Supports a method of releasing MCU security by erasing all EEPROM and flash blocks and verifying that all EEPROM and flash blocks are erased. |
| 0x0D | Set User Margin Level | Specifies a user margin read level for all flash blocks. |
| 0x10 | Erase Verify EEPROM Section | Verify that a given number of bytes starting at the address provided are erased. |
| 0x11 | Program EEPROM | Program up to four bytes in the EEPROM block. |
| 0x12 | Erase EEPROM Sector | Erase all bytes in a sector of the EEPROM block. |

#### 4.5.2.8.3 Allowed simultaneous flash and EEPROM operations

Only the operations marked 'OK' in the following table are permitted to be run simultaneously on the flash and EEPROM blocks. Some operations cannot be executed simultaneously because certain hardware resources are shared by the two memories. The priority has been placed on permitting flash reads while program and erase operations execute on the EEPROM, providing read (flash) while write (EEPROM) functionality.

**Table 4-18. Allowed simultaneous flash and EEPROM operations**

| Program flash | EEPROM | | | | |
|---|---|---|---|---|---|
| | Read | Margin read | Program | Sector erase | Mass erase |
| Read | | OK | OK | OK | |
| Margin Read[1] | | | | | |
| Program | | | | | |
| Sector Erase | | | | | |
| Mass Erase[2] | | | | | OK |

1. A 'Margin read' is any read after executing the margin setting commands 'Set user margin level' or 'Set field margin level' with anything but the 'normal' level specified. See the Note on margin settings in
2. The 'Mass erase' operations are commands 'Erase all blocks' and 'Erase flash block'

### 4.5.2.9 Flash and EEPROM command summary

This section provides details of all available flash commands launched by a command write sequence. The FSTAT[ACCERR] bit will be set during the command write sequence if any of the following illegal steps are performed, causing the command not to be processed by the memory controller:

- Starting any command write sequence that programs or erases flash memory before initializing the FLCKDIV register.
- Writing an invalid command as part of the command write sequence.
- For additional possible errors, refer to the error handling table provided for each command.

If a flash block is read during the execution of an algorithm (FSTAT[CCIF] = 0) on that same block, the read operation will return invalid data if both flags FERSTAT[SFDIF] and FERSTAT[DFDIF] are set. If the FERSTAT[SFDIF] or FERSTAT[DFDIF] flags were not previously set when the invalid read operation occurred, both the FERSTAT[SFDIF] and FERSTAT[DFDIF] flags will be set.

If the FSTAT[ACCERR] or FSTAT[FPVIOL] bits are set, the user must clear these bits before starting any command write sequence.

**CAUTION**

An EEPROM byte or flash longword must be in the erased state before being programmed. Cumulative programming of bits within an EEPROM byte or flash longword is not allowed.

#### 4.5.2.9.1 Erase verify all blocks command

The erase verify all blocks command will verify that all flash and EEPROM blocks have been erased.

**Table 4-19. Erase verify all blocks command FCCOB requirements**

| CCOBIX[2:0] | NVM_FCCOBHI parameters | NVM_FCCOBLO parameters |
|---|---|---|
| 000 | 0x01 | Not required |

Upon clearing NVM_FSTAT[CCIF] to launch the erase verify all blocks command, the memory controller will verify that the entire flash memory space is erased. The NVM_FSTAT[CCIF] flag will set after the erase verify all blocks operation has completed. If all blocks are not erased, it means blank check failed and both NVM_FSTAT[MGSTAT] bits will be set.

**Table 4-20. Erase verify all blocks command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| NVM_FSTAT | ACCERR | Set if CCOBIX[2:0] != 000 at command launch |
| | FPVIOL | None |
| | MGSTAT1 | Set if any errors have been encountered during the read[1] or if blank check failed |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the read or if blank check failed |

1. As found in the memory map for NVM

#### 4.5.2.9.2 Erase verify block command

The erase verify block command allows the user to verify that an entire flash or EEPROM block has been erased. The FCCOB global address [23:0] bits determine which block must be verified.

**Table 4-21. Erase verify block command FCCOB requirements**

| CCOBIX[2:0] | NVM_FCCOBHI parameters | NVM_FCCOBLO parameters |
|---|---|---|
| 000 | 0x02 | Global address [23:16] to identify Flash block[1] |
| 001 | Global address [15:0] in flash block to be verified | |

1. Global address [23] selects between flash (0) or EEPROM (1) block, that can otherwise eventually share the same address on the MCU global memory map.

Upon clearing NVM_FSTAT[CCIF] to launch the erase verify block command, the memory controller will verify that the selected flash or EEPROM block is erased. The NVM_FSTAT[CCIF] flag will set after the erase verify block operation has completed. If the block is not erased, it means blank check failed and both NVM_FSTAT[MGSTAT] bits will be set.

**Table 4-22. Erase verify block command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| FSTAT | ACCERR | Set if CCOBIX[2:0] != 000 at command launch |
| | | Set if an invalid global address [23:0] is supplied[1] |
| | FPVIOL | None |
| | MGSTAT1 | Set if any errors have been encountered during the read or if blank check failed |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the read or if blank check failed |

1. As found in the memory map for NVM

### 4.5.2.9.3 Erase verify flash section command

The erase verify flash section command will verify that a section of code in the flash memory is erased. The erase verify flash section command defines the starting point of the code to be verified and the number of longwords.

**Table 4-23. Erase verify flash section command FCCOB requirements**

| CCOBIX[2:0] | NVM_FCCOBHI parameters | NVM_FCCOBLO parameters |
|---|---|---|
| 000 | 0x03 | Global address [23:16] of flash block |
| 001 | Global address [15:0] of the first longwords to be verified | |
| 010 | Number of long words to be verified | |

Upon clearing NVM_FSTAT[CCIF] to launch the erase verify flash section command, the memory controller will verify that the selected section of flash memory is erased. The NVM_FSTAT[CCIF] flag will set after the erase verify flash section operation has completed. If the section is not erased, it means blank check failed and both FSTAT[MGSTAT] bits will be set.

**Table 4-24. Erase verify flash section command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| FSTAT | ACCERR | Set if CCOBIX[2:0] != 010 at command launch |

*Table continues on the next page...*

**Table 4-24. Erase verify flash section command error handling (continued)**

| Register | Error bit | Error condition |
|---|---|---|
| | | Set if command not available in current mode (see Table 4-9) |
| | | Set if an invalid global address [23:0] is supplied (see Table 4-6)[1] |
| | | Set if a misaligned long words address is supplied (global address[1:0] != 00) |
| | | Set if the requested section crosses flash address boundary |
| | FPVIOL | None |
| | MGSTAT1 | Set if any errors have been encountered during the read[2] or if blank check failed |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the read[2] or if blank check failed |

1. As defined by the memory map for NVM
2. As found in the memory map for NVM

### 4.5.2.9.4 Read once command

The read once command provides read access to a reserved 64 byte field (8 phrase) located in the nonvolatile information register of flash. The read once field can only be programmed once and can not be erased. It can be used to store the product ID or any other information that can be written only once. It is programmed using the program once command described in Program once command. To avoid code runaway, the read once command must not be executed from the flash block containing the program once reserved field.

**Table 4-25. Read once command FCCOB requirements**

| CCOBIX[2:0] | FCCOB parameters | |
|---|---|---|
| 000 | 0x04 | Not required |
| 001 | Read once phrase index (0x0000 – 0x0007) | |
| 010 | Read once word 0 value | |
| 011 | Read once word 1 value | |
| 100 | Read once word 2 value | |
| 101 | Read once word 3 value | |

Upon clearing FSTAT[CCIF] to launch the read once command, a read once phrase is fetched and stored in the FCCOB indexed register. The FSTAT[CCIF] flag will set after the read once operation has completed. Valid phrase index values for the read once command range from 0x0000 to 0x0007. During execution of the read once command, any attempt to read addresses within flash block will return invalid data.

**Table 4-26. Read once command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| FSTAT | ACCERR | Set if CCOBIX[2:0] != 001 at command launch |
| | | Set if command is not available in current mode (see Table 4-9) |
| | | Set if an invalid phrase index is supplied |
| | FPVIOL | None |
| | MGSTAT1 | Set if any errors have been encountered during the read |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the read |

#### 4.5.2.9.5 Program flash command

The program flash operation will program up to two previously erased longwords in the flash memory using an embedded algorithm.

**Note**

A flash phrase must be in the erased state before being programmed. Cumulative programming of bits within a flash phrase is not allowed.

**Table 4-27. Program flash command FCCOB requirements**

| CCOBIX[2:0] | NVM_FCCOBHI parameters | NVM_FCCOBLO parameters |
|---|---|---|
| 000 | 0x06 | Global address [23:16] to identify flash block |
| 001 | Global address [15:0] of longwords location to be programmed[1] | |
| 010 | Word 0 (longword 0) program value | |
| 011 | Word 1 (longword 0) program value | |
| 100 | Word 2 (longword 1) program value | |
| 101 | Word 3 (longword 1) program value | |

1. Global address [1:0] must be 00.

Upon clearing NVM_FSTAT[CCIF] to launch the program flash command, the memory controller will program the data words to the supplied global address and will then proceed to verify the data words read back as expected. The NVM_FSTAT[CCIF] flag will set after the program flash operation has completed.

**Table 4-28. Program flash command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| NVM_FSTAT | ACCERR | Set if CCOBIX[2:0] ≠ 011 or 101 at command launch |
| | | Set if command not available in current mode (see Table 4-9) |
| | | Set if an invalid global address [23:0] is supplied (see Table 4-6)[1] |

*Table continues on the next page...*

**Table 4-28. Program flash command error handling (continued)**

| Register | Error bit | Error condition |
|---|---|---|
| | | Set if a misaligned longword address is supplied (global address [1:0] != 00) |
| | | Set if the requested group of words breaches the end of the flash block. |
| | FPVIOL | Set if the global address [23:0] points to a protected data |
| | MGSTAT1 | Set if any errors have been encountered during the verify operation |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the verify operation |

1. As defined by the memory map of NVM

### 4.5.2.9.6 Program once command

The program once command restricts programming to a reserved 64 byte field (8 phrases) in the nonvolatile information register located in flash. The program once reserved field can be read using the read once command as described in Read once command. The program once command must be issued only because the nonvolatile information register in flash cannot be erased. To avoid code runaway, the read once command must not be executed from the flash block containing the program once reserved field.

**Table 4-29. Program once command FCCOB requirements**

| CCOBIX[2:0] | FCCOB parameters | |
|---|---|---|
| 000 | 0x07 | Not required |
| 001 | Program Once phrase index (0x000 – 0x0007) | |
| 010 | Program once Word 0 value | |
| 011 | Program once Word 1value | |
| 100 | Program once Word 2 value | |
| 101 | Program once Word 3 value | |

Upon clearing FSTAT[CCIF] to launch the program once command, the memory controller first verifies that the selected phrase is erased. If erased, then the selected phrase will be programmed and then verified with read back. The FSTAT[CCIF] flag will remain clear, setting only after the program once operation has completed.

The reserved nonvolatile information register accessed by the program once command cannot be erased, and any attempt to program one of these phrases a second time will not be allowed. Valid phrase index values for the program once command range from 0x0000 to 0x0007. During execution of the program once command, any attempt to read addresses within flash will return invalid data.

**Table 4-30. Program once command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| FSTAT | ACCERR | Set if CCOBIX[2:0] != 101 at command launch |
| | | Set if command not available in current mode (see Table 4-9) |
| | | Set if an invalid phrase index is supplied |
| | | Set if the requested phrase has already been programmed[1] |
| | FPVIOL | None |
| | MGSTAT1 | Set if any errors have been encountered during the verify operation |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the verify operation |

1. If a program once phrase is initially programmed to 0xFFFF_FFFF_FFFF_FFFF, the program once command will be allowed to execute again on that same phrase.

### 4.5.2.9.7 Erase all blocks command

The erase all blocks operation will erase the entire flash and EEPROM memory space.

**Table 4-31. Erase all blocks command FCCOB requirements**

| CCOBIX[2:0] | NVM_FCCOBHI parameters | NVM_FCCOBLO parameters |
|---|---|---|
| 000 | 0x08 | Not required |

Upon clearing NVM_FSTAT[CCIF] to launch the erase all blocks command, the memory controller will erase the entire NVM memory space and verify that it is erased. If the memory controller verifies that the entire NVM memory space was properly erased, security will be released. Therefore, the device is in unsecured state. During the execution of this command (NVM_FSTAT[CCIF] = 0) the user must not write to any NVM module register. The NVM_FSTAT[CCIF] flag will set after the erase all blocks operation has completed.

**Table 4-32. Erase all blocks command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| NVM_FSTAT | ACCERR | Set if CCOBIX[2:0] ≠ 000 at command launch |
| | | Set if command not available in current mode (see Table 4-9) |
| | FPVIOL | Set if any area of the flash or EEPROM memory is protected |
| | MGSTAT1 | Set if any errors have been encountered during the verify operation[1] |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the verify operation[1] |

1. As found in the memory map for NVM

#### 4.5.2.9.8 Erase flash block command

The erase flash block operation will erase all addresses in a flash or EEPROM block.

**Table 4-33. Erase flash block command FCCOB requirements**

| CCOBIX[2:0] | FCCOB parameters | |
|---|---|---|
| 000 | 0x09 | Global address [23:16] to identify flash block[1] |
| 001 | Global address[15:0] in flash block to be erased | |

1. Global address [23] selects between flash (0) or EEPROM (1) block, that can otherwise eventually share the same address on the MCU global memory map.

Upon clearing FSTAT[CCIF] to launch the erase flash block command, the memory controller will erase the selected flash block and verify that it is erased. The FSTAT[CCIF] flag will set after the erase flash block operation has completed.

**Table 4-34. Erase flash block command error handling**

| Register | Error Bit | Error Condition |
|---|---|---|
| FSTAT | ACCERR | Set if CCOBIX[2:0] != 001 at command launch |
| | | Set if command not available in current mode (see Table 4-9) |
| | | Set if an invalid global address [23:16] is supplied[1] |
| | FPVIOL | Set if an area of the selected flash block is protected |
| | MGSTAT1 | Set if any errors have been encountered during the verify operation[2] |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the verify operation[2] |

1. As defined by the memory map for NVM
2. As found in the memory map for NVM

#### 4.5.2.9.9 Erase flash sector command

The erase flash sector operation will erase all addresses in a flash sector.

**Table 4-35. Erase flash sector command FCCOB requirements**

| CCOBIX[2:0] | FCCOB parameters | |
|---|---|---|
| 000 | 0x0A | Global address [23:16] to identify flash block to be erased |
| 001 | Global address [15:0] anywhere within the sector to be erased. Refer to Overview for the flash sector size | |

Upon clearing FSTAT[CCIF] to launch the erase flash sector command, the memory controller will erase the selected flash sector and then verify that it is erased. The FSTAT[CCIF] flag will be set after the erase flash sector operation has completed.

**Table 4-36. Erase flash sector command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| FSTAT | ACCERR | Set if CCOBIX[2:0] != 001 at command launch |
| | | Set if command not available in current mode (see Table 4-9) |
| | | Set if an invalid global address [23:16] is supplied.[1] (see Table 4-6) |
| | | Set if a misaligned longword address is supplied (global address [1:0] != 00) |
| | FPVIOL | Set if the selected flash sector is protected |
| | MGSTAT1 | Set if any errors have been encountered during the verify operation |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the verify operation |

1. As defined by the memory map for NVM

### 4.5.2.9.10 Unsecure flash command

The unsecure flash command will erase the entire flash and EEPROM memory space, and if the erase is successful, will release security.

**Table 4-37. Unsecure flash command FCCOB requirements**

| CCOBIX[2:0] | FCCOB parameters | |
|---|---|---|
| 000 | 0x0B | Not required |

Upon clearing FSTAT[CCIF] to launch the unsecure flash command, the memory controller will erase the entire flash and EEPROM memory space and verify that it is erased. If the memory controller verifies that the entire flash and EEPROM memory space was properly erased, security will be released. If the erase verify is not successful, the unsecure flash operation sets FSTAT[MGSTAT1] and terminates without changing the security state. During the execution of this command (FSTAT[CCIF] = 0), the user must not write to any flash and EEPROM module register. The FSTAT[CCIF] flag is set after the unsecure flash operation has completed.

**Table 4-38. Unsecure flash command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| FSTAT | ACCERR | Set if CCOBIX[2:0] != 000 at command launch |
| | | Set if command is not available in current mode (see Table 4-9) |
| | FPVIOL | Set if any area of the flash or EEPROM memory is protected |
| | MGSTAT1 | Set if any errors have been encountered during the verify operation[1] |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the verify operation[1] |

1. As found in the memory map for NVM

#### 4.5.2.9.11 Verify backdoor access key command

The verify backdoor access key command will execute only if it is enabled by the NVM_FSEC[KEYEN] bits. The verify backdoor access key command releases security if user-supplied keys match those stored in the flash security bytes of the flash configuration field. See Table 4-6 for details. The code that performs verifying backdoor access command must be running from RAM or EEPROM.

**Table 4-39. Verify backdoor access key command FCCOB requirements**

| CCOBIX[2:0] | NVM_FCCOBHI parameters | NVM_FCCOBLO parameters |
|---|---|---|
| 000 | 0x0C | Not required |
| 001 | Key 0 | |
| 010 | Key 1 | |
| 011 | Key 2 | |
| 100 | Key 3 | |

Upon clearing NVM_FSTAT[CCIF] to launch the verify backdoor access key command, the memory controller will check the NVM_FSEC[KEYEN] bits to verify that this command is enabled. If not enabled, the memory controller sets the NVM_FSTAT[ACCERR] bit. If the command is enabled, the memory controller compares the key provided in FCCOB to the backdoor comparison key in the flash configuration field with Key 0 compared to 0xFF70, and so on. If the backdoor keys match, security will be released. If the backdoor keys do not match, security is not released and all future attempts to execute the verify backdoor access key command are aborted (set NVM_FSTAT[ACCERR]) until a reset occurs. The NVM_FSTAT[CCIF] flag is set after the verify backdoor access key operation has completed.

**Table 4-40. Verify backdoor access key command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| NVM_FSTAT | ACCERR | Set if CCOBIX[2:0] ≠ 100 at command launch |
| | | Set if an incorrect backdoor key is supplied |
| | | Set if backdoor key access has not been enabled (KEYEN[1:0] ≠ 10 |
| | | Set if the backdoor key has mismatched since the last reset |
| | FPVIOL | None |
| | MGSTAT1 | None |
| | MGSTAT0 | None |

#### 4.5.2.9.12 Set user margin level command

The user margin is a small delta to the normal read reference level and, in effect, is a minimum safety margin. That is, if the reads pass at the tighter tolerances of the user margins, the normal reads have at least that much safety margin before users experience data loss.

The set user margin level command causes the memory controller to set the margin level for future read operations of the flash or EEPROM block.

**Table 4-41. Set user margin level command FCCOB requirements**

| CCOBIX[2:0] | NVM_FCCOBHI parameters | NVM_FCCOBLO parameters |
|---|---|---|
| 000 | 0x0D | Global address [23:16] to identify flash block[1] |
| 001 | Global address [15:0] to identify flash block | |
| 010 | Margin level setting | |

1. Global Address [23] selects between flash (0) or EEPROM (1) block, that can otherwise eventually share the same address on the MCU global memory map.

Upon clearing NVM_FSTAT[CCIF] to launch the set user margin level command, the memory controller will set the user margin level for the targeted block and then set the NVM_FSTAT[CCIF] flag.

**Note**

When the EEPROM block is targeted, the EEPROM user margin levels are applied only to the EEPROM reads. However, when the Flash block is targeted, the flash user margin levels are applied to both Flash and EEPROM reads. It is not possible to apply user margin levels to the flash block only.

Valid margin level settings for the set user margin level command are defined in the following tables.

**Table 4-42. Valid set user margin level settings**

| CCOB (CCOBIX = 010) | Level description |
|---|---|
| 0x0000 | Return to normal level |
| 0x0001 | User margin-1 level[1] |
| 0x0002 | User margin-0 level[2] |

1. Read margin to the erased state
2. Read margin to the programmed state

**Table 4-43. Set user margin level command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| NVM_FSTAT | ACCERR | Set if CCOBIX[2:0] != 010 at command launch |
| | | Set if command is not available in current mode (see Table 4-9) |
| | | Set if an invalid global address [23:0] is supplied |
| | | Set if an invalid margin level setting is supplied |
| | FPVIOL | None |
| | MGSTAT1 | None |
| | MGSTAT0 | None |

**Note**

User margin levels can be used to check that NVM memory contents have adequate margin for normal level read operations. If unexpected results are encountered when checking NVM memory contents at user margin levels, a potential loss of information has been detected.

### 4.5.2.9.13 Erase verify EEPROM section command

The erase verify EEPROM section command will verify that a section of code in the EEPROM is erased. The erase verify EEPROM section command defines the starting point of the data to be verified and the number of bytes.

**Table 4-44. Erase verify EEPROM section command FCCOB requirements**

| CCOBIX[2:0] | NVM_FCCOBHI parameters | NVM_FCCOBLO parameters |
|---|---|---|
| 000 | 0x10 | Global address [23:16] to identify the EEPROM block |
| 001 | Global address [15:0] of the first byte to be verified | |
| 010 | Number of bytes to be verified | |

Upon clearing NVM_FSTAT[CCIF] to launch the erase verify that EEPROM section command, the memory controller will verify the selected section of EEPROM memory is erased. The NVM_FSTAT[CCIF] flag will set after the erase verify EEPROM section operation has completed. If the section is not erased, which means that blank check failed, both NVM_FSTAT[MGSTAT] bits will be set.

**Table 4-45. Erase verify EEPROM section command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| FSTAT | ACCERR | Set if CCOBIX[2:0] ≠ 010 at command launch |
| | | Set if command is not available in current mode (see Table 4-9) |

*Table continues on the next page...*

**Table 4-45. Erase verify EEPROM section command error handling (continued)**

| Register | Error bit | Error condition |
|---|---|---|
| | | Set if an invalid global address [23:0] is supplied |
| | | Set if the requested section breaches the end of the EEPROM block |
| | FPVIOL | None |
| | MGSTAT1 | Set if any errors have been encountered during the read or if blank check failed |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the read or it blank check failed. |

### 4.5.2.9.14 Program EEPROM command

The program EEPROM operation programs one to four previously erased bytes in the EEPROM block. The program EEPROM operation will confirm that the targeted location(s) were successfully programmed upon completion.

**Note**

A EEPROM byte must be in the erased state before being programmed. Cumulative programming of bits within a EEPROM byte is not allowed.

**Table 4-46. Program EEPROM command FCCOB requirements**

| CCOBIX[2:0] | NVM_FCCOBHI parameters | NVM_FCCOBLO parameters |
|---|---|---|
| 000 | 0x11 | Global address [23:16] to identify the EEPROM block |
| 001 | Global address [15:0] of the first word to be verified | |
| 010 | | Byte 0 program value |
| 011 | | Byte 1 program value, if desired |
| 100 | | Byte 2 program value, if desired |
| 101 | | Byte 3 program value, if desired |

Upon clearing NVM_FSTAT[CCIF] to launch the program EEPROM command, the user-supplied words will be transferred to the memory controller and be programmed if the area is unprotected. The CCOBIX index value at program EEPROM command launch determines how many bytes will be programmed in the EEPROM block. The NVM_FSTAT[CCIF] flag is set when the operation has completed.

**Table 4-47. Program EEPROM command error handling**

| Register | Error Bit | Error condition |
|---|---|---|
| NVM_FSTAT | ACCERR | Set if CCOBIX[2:0] < 010 at command launch |
| | | Set if CCOBIX[2:0] >101 at command launch |

*Table continues on the next page...*

**Table 4-47. Program EEPROM command error handling (continued)**

| Register | Error Bit | Error condition |
|---|---|---|
| | | Set if command is not available in current mode (see Table 4-9) |
| | | Set if an invalid global address [23:0] is supplied |
| | | Set if the requested group of words breaches the end of the EEPROM block |
| | FPVIOL | Set if the selected area of the EEPROM memory is protected |
| | MGSTAT1 | Set if any errors have been encountered during the verify operation |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the verify operation |

### 4.5.2.9.15 Erase EEPROM sector command

The erase EEPROM sector operation will erase all addresses in a sector of the EEPROM block.

**Table 4-48. Erase EEPROM sector command FCCOB requirements**

| CCOBIX[2:0] | NVM_FCCOBHI parameters | NVM_FCCOBLO parameters |
|---|---|---|
| 000 | 0x12 | Global address [23:16] to identify EEPROM block |
| 001 | Global address [15:0] anywhere within the sector to be erased. See Overview for EEPROM sector size | |

Upon clearing NVM_FSTAT[CCIF] to launch the erase EEPROM sector command, the memory controller will erase the selected EEPROM sector and verify that it is erased. The NVM_FSTAT[CCIF] flag will set after the erase EEPROM sector operation has completed.

**Table 4-49. Erase EEPROM sector command error handling**

| Register | Error bit | Error condition |
|---|---|---|
| NVM_FSTAT | ACCERR | Set if CCOBIX[2:0] ≠ 001 at command launch |
| | | Set if command is not available in current mode (see Table 4-9) |
| | | Set if an invalid global address [23:0] is supplied (see Table 4-6) |
| | FPVIOL | Set if the selected area of the EEPROM memory is protected |
| | MGSTAT1 | Set if any errors have been encountered during the verify operation |
| | MGSTAT0 | Set if any non-correctable errors have been encountered during the verify operation |

## 4.6 Flash and EEPROM registers descriptions

The flash module contains a set of 16 user control and status registers located between 0x3020 and 0x302F. In the case of the writable registers, the write accesses are forbidden during flash command execution. For more details, see Caution note in Flash and EEPROM memory map. A summary of the flash module registers is given in the following table with detailed descriptions in the following subsections.

**NVM memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 3020 | Flash Clock Divider Register (NVM_FCLKDIV) | 8 | R/W | 00h | 4.6.1/87 |
| 3021 | Flash Security Register (NVM_FSEC) | 8 | R | Undefined | 4.6.2/88 |
| 3022 | Flash CCOB Index Register (NVM_FCCOBIX) | 8 | R/W | 00h | 4.6.3/89 |
| 3024 | Flash Configuration Register (NVM_FCNFG) | 8 | R/W | 00h | 4.6.4/89 |
| 3025 | Flash Error Configuration Register (NVM_FERCNFG) | 8 | R/W | 00h | 4.6.5/90 |
| 3026 | Flash Status Register (NVM_FSTAT) | 8 | R/W | 80h | 4.6.6/91 |
| 3027 | Flash Error Status Register (NVM_FERSTAT) | 8 | R/W | 00h | 4.6.7/92 |
| 3028 | Flash Protection Register (NVM_FPROT) | 8 | R | Undefined | 4.6.8/93 |
| 3029 | EEPROM Protection Register (NVM_EEPROT) | 8 | R/W | Undefined | 4.6.9/94 |
| 302A | Flash Common Command Object Register:High (NVM_FCCOBHI) | 8 | R/W | 00h | 4.6.10/95 |
| 302B | Flash Common Command Object Register: Low (NVM_FCCOBLO) | 8 | R/W | 00h | 4.6.11/96 |
| 302C | Flash Option Register (NVM_FOPT) | 8 | R | Undefined | 4.6.12/96 |

### 4.6.1 Flash Clock Divider Register (NVM_FCLKDIV)

The FCLKDIV register is used to control timed events in program and erase algorithms.

**NOTE**

The FCLKDIV register must not be written while a flash command is executing (NVM_FSTAT[CCIF] = 0)

Address: 3020h base + 0h offset = 3020h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | FDIVLD | FDIVLCK | FDIV | | | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**NVM_FCLKDIV field descriptions**

| Field | Description |
|---|---|
| 7<br>FDIVLD | Clock Divider Loaded<br><br>0 FCLKDIV register has not been written since the last reset.<br>1 FCLKDIV register has been written since the last reset. |
| 6<br>FDIVLCK | Clock Divider Locked<br><br>0 FDIV field is open for writing.<br>1 FDIV value is locked and cannot be changed. After the lock bit is set high, only reset can clear this bit and restore writability to the FDIV field in user mode. |
| FDIV | Clock Divider Bits<br><br>FDIV[5:0] must be set to effectively divide BUSCLK down to 1MHz to control timed events during flash program and erase algorithms. Refer to the table in the Writing the FCLKDIV register for the recommended values of FDIV based on the BUSCLK frequency. |

## 4.6.2 Flash Security Register (NVM_FSEC)

The FSEC register holds all bits associated with the security of the MCU and NVM module. All bits in the FSEC register are readable but not writable. During the reset sequence, the FSEC register is loaded with the contents of the flash security byte in the flash configuration field at global address 0xFF7F located in flash memory.

See Security for security function.

Address: 3020h base + 1h offset = 3021h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | KEYEN | | Reserved | | | | SEC | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**NVM_FSEC field descriptions**

| Field | Description |
|---|---|
| 7–6<br>KEYEN | Backdoor Key Security Enable Bits<br><br>The KEYEN[1:0] bits define the enabling of backdoor key access to the flash module.<br><br>**NOTE:** 01 is the preferred KEYEN state to disable backdoor key access.<br><br>00 Disabled<br>01 Disabled<br>10 Enabled<br>11 Disabled |

*Table continues on the next page...*

**NVM_FSEC field descriptions (continued)**

| Field | Description |
|---|---|
| 5–2<br>Reserved | This field is reserved. |
| SEC | Flash Security Bits<br><br>The SEC[1:0] bits define the security state of the MCU. If the flash module is unsecured using backdoor key access, the SEC bits are forced to 10.<br><br>**NOTE:** 01 is the preferred SEC state to set MCU to secured state.<br><br>00 Secured<br>01 Secured<br>10 Unsecured<br>11 Secured |

## 4.6.3 Flash CCOB Index Register (NVM_FCCOBIX)

The FCCOBIX register is used to index the FCCOB register for NVM memory operations.

Address: 3020h base + 2h offset = 3022h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | | | CCOBIX | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**NVM_FCCOBIX field descriptions**

| Field | Description |
|---|---|
| 7–3<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| CCOBIX | Common Command Register Index<br><br>The CCOBIX bits are used to select which word of the FCCOB register array is being read or written to. |

## 4.6.4 Flash Configuration Register (NVM_FCNFG)

The FCNFG register enables the flash command complete interrupt and forces ECC faults on flash array read access from the CPU.

Address: 3020h base + 4h offset = 3024h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | CCIE | 0 | | IGNSF | 0 | | FDFD | FSFD |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**NVM_FCNFG field descriptions**

| Field | Description |
|---|---|
| 7<br>CCIE | Command Complete Interrupt Enable<br><br>The CCIE bit controls interrupt generation when a flash command has completed.<br><br>0 Command complete interrupt disabled.<br>1 An interrupt will be requested whenever the CCIF flag in the FSTAT register is set. |
| 6–5<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 4<br>IGNSF | Ignore Single Bit Fault<br><br>The IGNSF controls single bit fault reporting in the FERSTAT register.<br><br>0 All single bit faults detected during array reads are reported.<br>1 Single bit faults detected during array reads are not reported and the single bit fault interrupt will not be generated. |
| 3–2<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 1<br>FDFD | Force Double Bit Fault Detect<br><br>The FDFD bit allows the user to simulate a double bit fault during flash array read operations and check the associated interrupt routine. The FDFD bit is cleared by writing a 0 to FDFD.<br><br>0 Flash array read operations will set the FERSTAT[DFDIF] flag only if a double bit fault is detected.<br>1 Any flash array read operation will force the FERSTAT[DFDIF] flag to be set and an interrupt will be generated as long as the DFDIE interrupt enable in the FERCNFG register is set. |
| 0<br>FSFD | Force Single Bit Fault Detect<br><br>The FSFD bit allows the user to simulate a single bit fault during flash array read operations and check the associated interrupt routine. The FSFD bit is cleared by writing a 0 to FSFD.<br><br>0 Flash array read operations will set the SFDIF flag in the FERSTAT register only if a single bit fault is detected.<br>1 Flash array read operation will force the SFDIF flag in the FERSTAT register to be set and an interrupt will be generated as long as the SFDIE interrupt enable in the FERCNFG register is set. |

## 4.6.5 Flash Error Configuration Register (NVM_FERCNFG)

The FERCNFG register enables the flash error interrupts for the FERSTAT flags.

Address: 3020h base + 5h offset = 3025h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | | | | DFDIE | SFDIE |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**NVM_FERCNFG field descriptions**

| Field | Description |
|---|---|
| 7–2<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 1<br>DFDIE | Double Bit Fault Detect Interrupt Enable<br><br>The DFDIE bit controls interrupt generation when a double bit fault is detected during a flash block read operation.<br><br>0 DFDIF interrupt disabled.<br>1 An interrupt will be requested whenever the DFDIF flag is set. |
| 0<br>SFDIE | Single Bit Fault Detect Interrupt Enable<br><br>The SFDIE bit controls interrupt generation when a single bit fault is detected during a flash block read operation.<br><br>0 SFDIF interrupt disabled whenever the SFDIF flag is set.<br>1 An interrupt will be requested whenever the SFDIF flag is set. |

## 4.6.6 Flash Status Register (NVM_FSTAT)

The FSTAT register reports the operational status of the flash and EEPROM module.

Address: 3020h base + 6h offset = 3026h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | CCIF | 0 | ACCERR | FPVIOL | MGBUSY | 0 | MGSTAT | |
| Write | | | | | | | | |
| Reset | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**NVM_FSTAT field descriptions**

| Field | Description |
|---|---|
| 7<br>CCIF | Command Complete Interrupt Flag<br><br>The CCIF flag indicates that a flash command has completed. The CCIF flag is cleared by writing a 1 to CCIF to launch a command and CCIF will stay low until command completion or command violation.<br><br>0 Flash command in progress.<br>1 Flash command has completed. |
| 6<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 5<br>ACCERR | Flash Access Error Flag<br><br>The ACCERR bit indicates an illegal access has occurred to the flash memory caused by either a violation of the command write sequence or issuing an illegal flash command. While ACCERR is set, the CCIF flag cannot be cleared to launch a command. The ACCERR bit is cleared by writing a 1 to ACCERR. Writing a 0 to the ACCERR bit has no effect on ACCERR. |

*Table continues on the next page...*

**NVM_FSTAT field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 No access error detected.<br>1 Access error detected. |
| 4<br>FPVIOL | Flash Protection Violation Flag<br><br>The FPVIOL bit indicates an attempt was made to program or erase an address in a protected area of flash or EEPROM memory during a command write sequence. The FPVIOL bit is cleared by writing a 1 to FPVIOL. Writing a 0 to the FPIOL bit has no effect on FPIOL. While FPIOL is set, it is not possible to launch a command or start a command write sequence.<br><br>0 No protection violation detected.<br>1 Protection violation detected. |
| 3<br>MGBUSY | Memory Controller Busy Flag<br><br>The MGBUSY flag reflects the active state of the memory controller.<br><br>0 Memory controller is idle.<br>1 Memory controller is busy executing a flash command (CCIF = 0). |
| 2<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| MGSTAT | Memory Controller Command Completion Status Flag<br><br>One or more MGSTAT flag bits are set if an error is detected during execution of a flash command or during the flash reset sequence.<br><br>**NOTE:** Reset value can deviate from the value shown if a double bit fault is detected during the reset sequence. |

## 4.6.7 Flash Error Status Register (NVM_FERSTAT)

The FERSTAT register reflects the error status of internal flash and EEPROM operations.

Address: 3020h base + 7h offset = 3027h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | | | | DFDIF | SFDIF |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**NVM_FERSTAT field descriptions**

| Field | Description |
|---|---|
| 7–2<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 1<br>DFDIF | Double Bit Fault Detect Interrupt Flag<br><br>The setting of the DFDIF flag indicates that a double bit fault was detected in the stored parity and data bits during a flash array read operation or that a flash array read operation returning invalid data was |

*Table continues on the next page...*

**NVM_FERSTAT field descriptions (continued)**

| Field | Description |
|---|---|
| | attempted on a flash block that was under a flash command operation. The DFDIF flag is cleared by writing a 1 to DFDIF. Writing a 0 to DFDIF has no effect on DFDIF.<br><br>**NOTE:** The single bit fault and double bit fault flags are mutually exclusive for parity errors, meaning that an ECC fault occurrence can be either single fault or double fault but never both. A simultaneous access collision, when the flash array read operation is returning invalid data attempted while a command is running, is indicated when both SFDIF and DFDIF flags are high.<br><br>**NOTE:** There is a one cycle delay in storing the ECC DFDIF and SFDIF fault flags in the register. At least one NOP is required after a flash memory read before checking FERSTAT for the occurrence of EEC errors.<br><br>0 No double bit fault detected.<br>1 Double bit fault detected or a flash array read operation returning invalid data was attempted while command running. |
| 0<br>SFDIF | Single Bit Fault Detect Interrupt Flag<br><br>With the IGNSF bit in the FCNFG register clear, the SFDIF flag indicates that a single bit fault was detected in the stored parity and data bits during a flash array read operation or that a flash array read operation returning invalid data was attempted on a flash block that was under a flash command operation. The SFDIF flag is cleared by writing a 1 to SFDIF. Writing a 0 to SFDIF has no effect on SRFDIF.<br><br>0 No single bit fault detected.<br>1 Single bit fault detected and corrected or a flash array read operation returning invalid data was attempted while command running. |

## 4.6.8 Flash Protection Register (NVM_FPROT)

The FPROT register defines which flash sectors are protected against program and erase operations.

The unreserved bits of the FPROT register are writable with the restriction that the size of the protected region can only be increased (see Protection).

During the reset sequence, the FPROT register is loaded with the contents of the flash protection byte in the flash configuration field at global address 0xFF7C located in flash memory. To change the flash protection that will be loaded during the reset sequence, the upper sector of the flash memory must be unprotected, then the flash protection byte must be reprogrammed.

Trying to alter data in any protected area in the flash memory will result in a protection violation error and the FPVIOL bit will be set in the FSTAT register. The block erase of a flash block is not possible if any of the flash sectors contained in the same flash block are protected.

Address: 3020h base + 8h offset = 3028h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | FPOPEN | 1 | FPHDIS | FPHS | | 0 | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**NVM_FPROT field descriptions**

| Field | Description |
|---|---|
| 7<br>FPOPEN | Flash Protection Operation Enable<br><br>The FPOPEN bit determines the protection function for program or erase operations.<br><br>0 When FPOPEN is clear, the FPHDIS and FPLDIS bits define unprotected address ranges as specified by the corresponding FPHS and FPLS bits.<br>1 When FPOPEN is set, the FPHDIS and FPLDIS bits enable protection for the address range specified by the corresponding FPHS and FPLS bits. |
| 6<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 1. |
| 5<br>FPHDIS | Flash Protection Higher Address Range Disable<br><br>The FPHDIS bit determines whether there is a protected/unprotected area in a specific region of the flash memory ending with global address 0xFFFF.<br><br>0 Protection/Unprotection enabled.<br>1 Protection/Unprotection disabled. |
| 4–3<br>FPHS | Flash Protection Higher Address Size<br><br>The FPHS bits determine the size of the protected/unprotected area in flash memory. The FPHS bits can be written to only while the FPHDIS bit is set. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 4.6.9 EEPROM Protection Register (NVM_EEPROT)

The EEPROT register defines which EEPROM sectors are protected against program and erase operations.

The unreserved bits of the EEPROT register are writable with the restriction that protection can be added but not removed. Writes must increase the DPS value and the DPOPEN bit can only be written from 1, protection disabled, to 0, protection enabled. If the DPOPEN bit is set, the state of the DPS bits is irrelevant.

During the reset sequence, fields DPOPEN and DPS of the EEPROT register are loaded with the contents of the EEPROM protection byte in the flash configuration field at global address 0xFF7D located in flash memory. To change the EEPROM protection that

will be loaded during the reset sequence, the flash sector containing the EEPROM protection byte must be unprotected. Then the EEPROM protection byte must be programmed.

Trying to alter data in any protected area in the EEPROM memory will result in a protection violation error and the FPVIOL bit will be set in the FSTAT register. Block erase of the EEPROM memory is not possible if any of the EEPROM sectors are protected.

Address: 3020h base + 9h offset = 3029h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | DPOPEN | 0 | | | | | DPS | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**NVM_EEPROT field descriptions**

| Field | Description |
|---|---|
| 7<br>DPOPEN | EEPROM Protection Control<br><br>0 Enables EEPROM memory protection from program and erase with protected address range defined by DPS bits.<br>1 Disables EEPROM memory protection from program and erase. |
| 6–3<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| DPS | EEPROM Protection Size<br><br>These bits determine the size of the protected area in the EEPROM memory. |

## 4.6.10 Flash Common Command Object Register:High (NVM_FCCOBHI)

The FCCOB is an array of six words addressed via the CCOBIX index found in the FCCOBIX register. Byte-wide reads and writes are allowed to the FCCOB register.

Address: 3020h base + Ah offset = 302Ah

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | CCOB | | | | | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**NVM_FCCOBHI field descriptions**

| Field | Description |
|---|---|
| CCOB | Common Command Object Bit 15:8<br><br>High 8 bits of common command object register |

## 4.6.11 Flash Common Command Object Register: Low (NVM_FCCOBLO)

The FCCOB is an array of six words addressed via the CCOBIX index found in the FCCOBIX register. Byte-wide reads and writes are allowed to the FCCOB register.

Address: 3020h base + Bh offset = 302Bh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | CCOB | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**NVM_FCCOBLO field descriptions**

| Field | Description |
|---|---|
| CCOB | Common Command Object Bit 7:0<br><br>Low 8 bits of common command object register |

## 4.6.12 Flash Option Register (NVM_FOPT)

The FOPT register is the flash option register.

During the reset sequence, the FOPT register is loaded from the flash nonvolatile byte in the flash configuration field at global address 0xFF7E located in flash memory as indicated by reset condition.

Address: 3020h base + Ch offset = 302Ch

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | NV | | | | | | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

## NVM_FOPT field descriptions

| Field | Description |
|---|---|
| NV | Nonvolatile Bits<br><br>The NV[7:0] bits are available as nonvolatile bits. During the reset sequence, the FOPT register is loaded from the flash nonvolatile byte in the flash configuration field at global address 0xFF7E located in flash memory. |

# Chapter 5
# Interrupt

## 5.1 Interrupts

Interrupts save the current CPU status and registers, execute an interrupt service routine (ISR), and then restore the CPU status so that processing resumes where it left off before the interrupt. Other than the software interrupt (SWI), which is a program instruction, interrupts are caused by hardware events such as an edge on the IRQ pin or a timer-overflow event. The debug module can also generate an SWI under certain circumstances.

If an event occurs in an enabled interrupt source, an associated read-only status flag will be set. The CPU will not respond unless only the local interrupt enable is a logic 1. The I bit in the CCR is 0 to allow interrupts. The global interrupt mask (I bit) in the CCR is initially set after reset that masks (prevents) all maskable interrupt sources. The user program initializes the stack pointer and performs other system setups before clearing the I bit to allow the CPU to respond to interrupts.

When the CPU receives a qualified interrupt request, it completes the current instruction before responding to the interrupt. The interrupt sequence obeys the same cycle-by-cycle sequence as the SWI instruction and consists of:

- Saving the CPU registers on the stack.
- Setting the I bit in the CCR to mask further interrupts.
- Fetching the interrupt vector for the highest-priority interrupt that is currently pending.
- Filling the instruction queue with the first three bytes of program information starting from the address fetched from the interrupt vector locations.

While the CPU is responding to the interrupt, the I bit is automatically set to prevent another interrupt from interrupting the ISR itself, which is called nesting of interrupts. Normally, the I bit is restored to 0 when the CCR is restored from the value stacked on

entry to the ISR. In rare cases, the I bit may be cleared inside an ISR, after clearing the status flag that generated the interrupt, so that other interrupts can be serviced without waiting for the first service routine to finish. This practice is recommended only for the most experienced programmers because it can lead to subtle program errors that are difficult to debug.

The interrupt service routine ends with a return-from-interrupt (RTI) instruction that restores the CCR, A, X, and PC registers to their pre-interrupt values by reading the previously saved information off the stack.

**Note**

For compatibility with the M68HC08, the H register is not automatically saved and restored. Push H onto the stack at the start of the interrupt service routine (ISR) and restore it immediately before the RTI that is used to return from the ISR.

When two or more interrupts are pending when the I bit is cleared, the highest priority source is serviced first.

### 5.1.1 Interrupt stack frame

The following figure shows the contents and organization of a stack frame. Before the interrupt, the stack pointer (SP) points at the next available byte location on the stack. The current values of CPU registers are stored on the stack, starting with the low-order byte of the program counter (PC) and ending with the CCR. After stacking, the SP points at the next available location on the stack, which is the address that is one less than the address where the CCR was saved. The PC value that is stacked is the address of the instruction in the main program that would have executed next if the interrupt had not occurred.

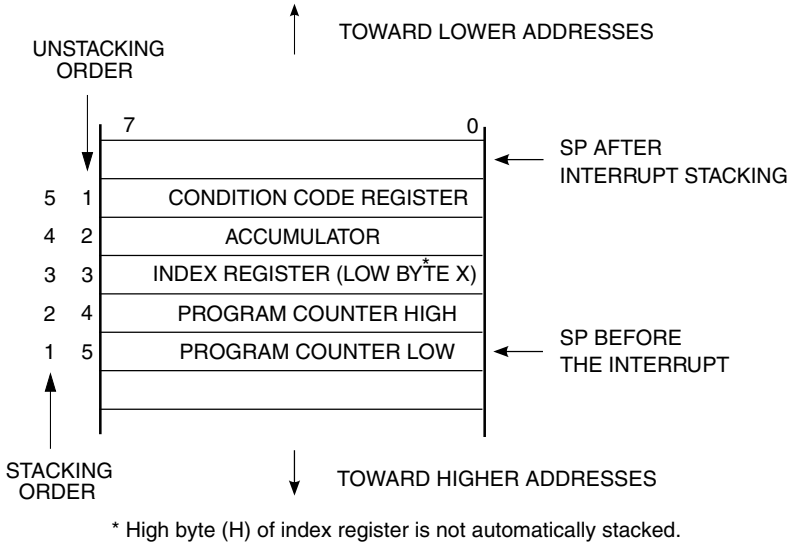



**Figure 5-1. Interrupt stack frame**

When an RTI instruction executes, these values are recovered from the stack in reverse order. As part of the RTI sequence, the CPU fills the instruction pipeline by reading three bytes of program information, starting from the PC address recovered from the stack.

The status flag causing the interrupt must be acknowledged (cleared) before returning from the ISR. Typically, the flag must be cleared at the beginning of the ISR because if another interrupt is generated by this source it will be registered so that it can be serviced after completion of the current ISR.

## 5.1.2 Interrupt vectors, sources, and local masks

The following table provides a summary of all interrupt sources. High-priority sources are located toward the bottom of the table. The high-order byte of the address for the interrupt service routine is located at the first address in the vector address column, and the low-order byte of the address for the interrupt service routine is located at the next higher address.

When an interrupt condition occurs, an associated flag bit is set. If the associated local interrupt enable is 1, an interrupt request is sent to the CPU. If the global interrupt mask (I bit in the CCR) is 0, the CPU finishes the current instruction, stacks the PCL, PCH, X, A, and CCR CPU registers, sets the I bit, and then fetches the interrupt vector for the highest priority pending interrupt. Processing then continues in the interrupt service routine.

### 5.1.3 Hardware nested interrupt

This device has interrupt priority controller (IPC) module to provide up to four-level nested interrupt capability. IPC includes the following features:

- Four-level programmable interrupt priority for each interrupt source.
- Support for prioritized preemptive interrupt service routines
  - Low-priority interrupt requests are blocked when high-priority interrupt service routines are being serviced.
  - Higher or equal priority level interrupt requests can preempt lower priority interrupts being serviced.
- Automatic update of interrupt priority mask with being serviced interrupt source priority level when the interrupt vector is being fetched.
- Interrupt priority mask can be modified during main flow or interrupt service execution.
- Previous interrupt mask level is automatically stored when interrupt vector is fetched (four levels of previous values accommodated)

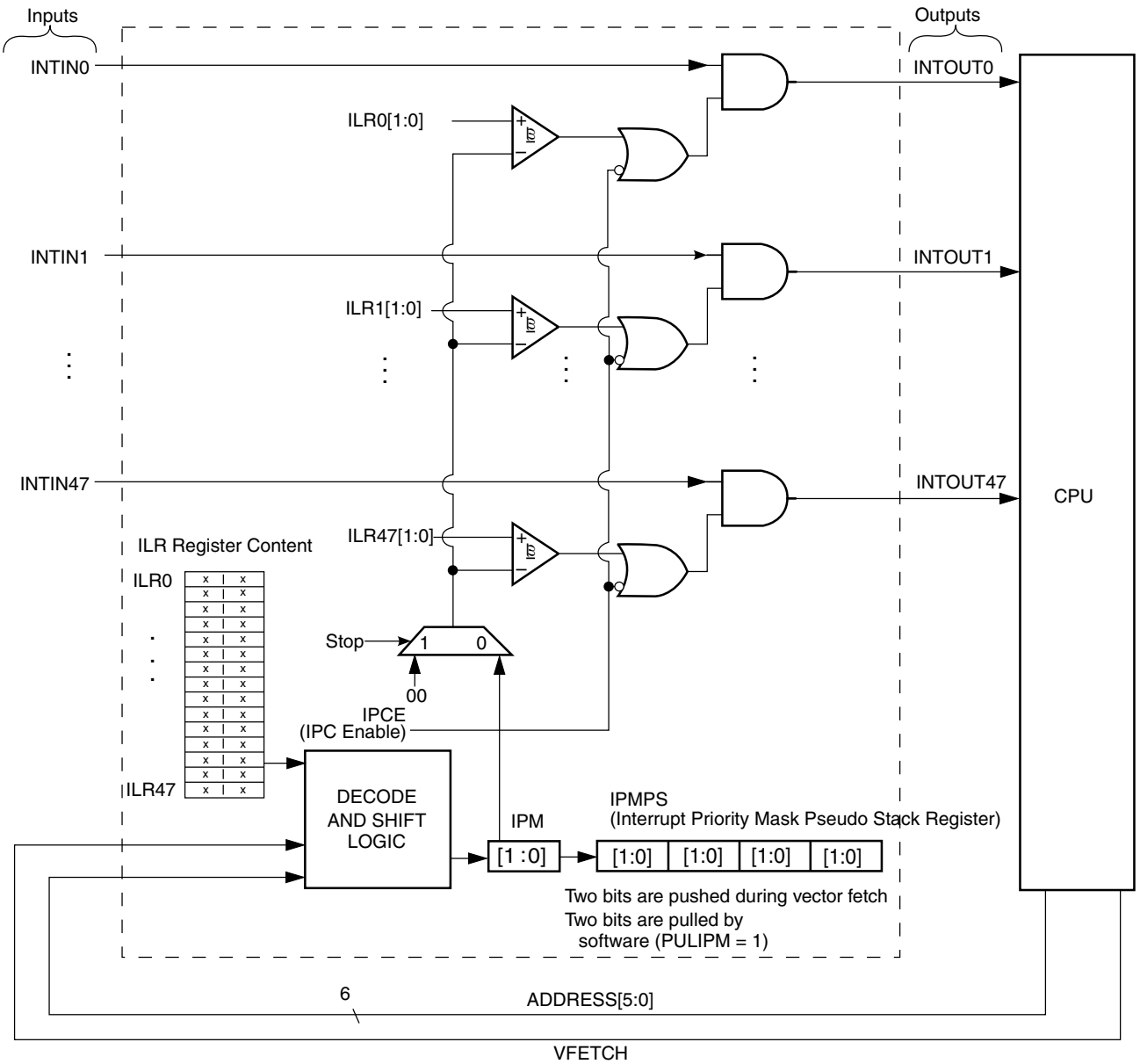



**Figure 5-2. Interrupt priority controller block diagram**

The IPC works with the existing HCS08 interrupt mechanism to allow nested interrupts with programmable priority levels. This module also allows implementation of preemptive interrupt according to the programmed interrupt priority with minimal software overhead. The IPC consists of three major functional blocks:

- The interrupt priority level registers
- The interrupt priority level comparator set
- The interrupt mask register update and restore mechanism

### 5.1.3.1 Interrupt priority level register

This set of registers is associated with the interrupt sources to the HCS08 CPU. Each interrupt priority level is a 2-bit value such that a user can program the interrupt priority level of each source to priority 0, 1, 2, or 3. Level 3 has the highest priority while level 0 has the lowest. Software can read or write to these registers at any time. The interrupt priority level comparator set, interrupt mask register update, and restore mechanism use this information.

### 5.1.3.2 Interrupt priority level comparator set

When the module is enabled, an active interrupt request forces a comparison between the corresponding ILR and the 2-bit interrupt mask IPM[1:0]. In stop3 mode, the IPM[1:0] is substituted by value 00b. If the ILR value is greater than or equal to the value of the interrupt priority mask (IPM bits in IPCSC), the corresponding interrupt out (INTOUT) signal will be asserted and signals an interrupt request to the HCS08 CPU.

When the module is disabled, the interrupt request signal from the source is directly passed to the CPU.

The interrupt priority level programmed in the interrupt priority register will not affect the inherent interrupt priority arbitration as defined by the HCS08 CPU because the IPC is an external module. Therefore, if two (or more) interrupts are present in the HCS08 CPU at the same time, the inherent priority in HCS08 CPU will perform arbitration by the inherent interrupt priority.

### 5.1.3.3 Interrupt priority mask update and restore mechanism

The interrupt priority mask (IPM) is two bits located in the least significant end of IPCSC register. These two bits control which interrupt is allowed to be presented to the HCS08 CPU. During vector fetch, the interrupt priority mask is updated automatically with the value of the ILR corresponding to that interrupt source. The original value of the IPM will be saved onto IPMPS for restoration after the interrupt service routine completes execution. When the interrupt service routine completes execution, the user restore the original value of IPM by writing 1 to the IPCSC[PULIPM] bit. In both cases, the IPMPS is a shift register functioning as a pseudo stack register for storing the IPM. When the IPM is updated, the original value is shifted into IPMPS. The IPMPS can store four levels of IPM. If the last position of IPMPS is written, the PSF flag indicates that the IPMPS is full. If all the values in the IPMPS were read, the PSE flag indicates that the IPMPS is empty.

### 5.1.3.4 Integration and application of the IPC

All interrupt inputs that comes from peripheral modules are synchronous signals. None of the asynchronous signals of the interrupts are routed to IPC. The asynchronous signals of the interrupts are routed directly to SIM module to wake system clocks in stop3 mode.

Additional care must be exercised when IRQ is reprioritized by IPC. CPU instructions BIL and BIH need input from IRQ pin. If IRQ interrupt is masked, BIL and BIH still work but the IRQ interrupt will not occur.

- The interrupt priority controller must be enabled to function. While inside an interrupt service routine, some work has to be done to enable other higher priority interrupts. The following is a pseudo code example written in assembly language:

```
INT_SER :
        BCLR        INTFLAG,INTFLAG_R ; clear flag that generate interrupt
        .                             ; do the most critical part
        .                             ; which it cannot be interrupted
        .
        .
        .
        CLI                           ; global interrupt enable and nested interrupt
enabled
        .                             ; continue the less critical
        .
        .
        .
        BSET        PULIPM, PULIPM_R  ; restore the old IPM value before leaving
        RTI                           ; then you can return
```

- A minimum overhead of six bus clock cycles is added inside an interrupt services routine to enable preemptive interrupts.
- As an interrupt of the same priority level is allowed to pass through IPC to HCS08 CPU, the flag generating the interrupt must be cleared before doing CLI to enable preemptive interrupts.
- The IPM is automatically updated to the level the interrupt is servicing and the original level is kept in IPMPS. Watch out for the full (PSF) bit if nesting for more than four levels is expected.
- Before leaving the interrupt service routine, the previous levels must be restored manually by setting PULIPM bit. Watch out for the full (PSF) bit and empty (PSE) bit.

## 5.2 IRQ

The IRQ (external interrupt) module provides a maskable interrupt input.

## 5.2.1 Features

Features of the IRQ module include:

- A Dedicated External Interrupt Pin
- IRQ Interrupt Control Bits
- Programmable Edge-only or Edge and Level Interrupt Sensitivity
- Automatic Interrupt Acknowledge
- Internal pullup device

A low level applied to the external interrupt request (IRQ) pin can latch a CPU interrupt request. The following figure shows the structure of the IRQ module:

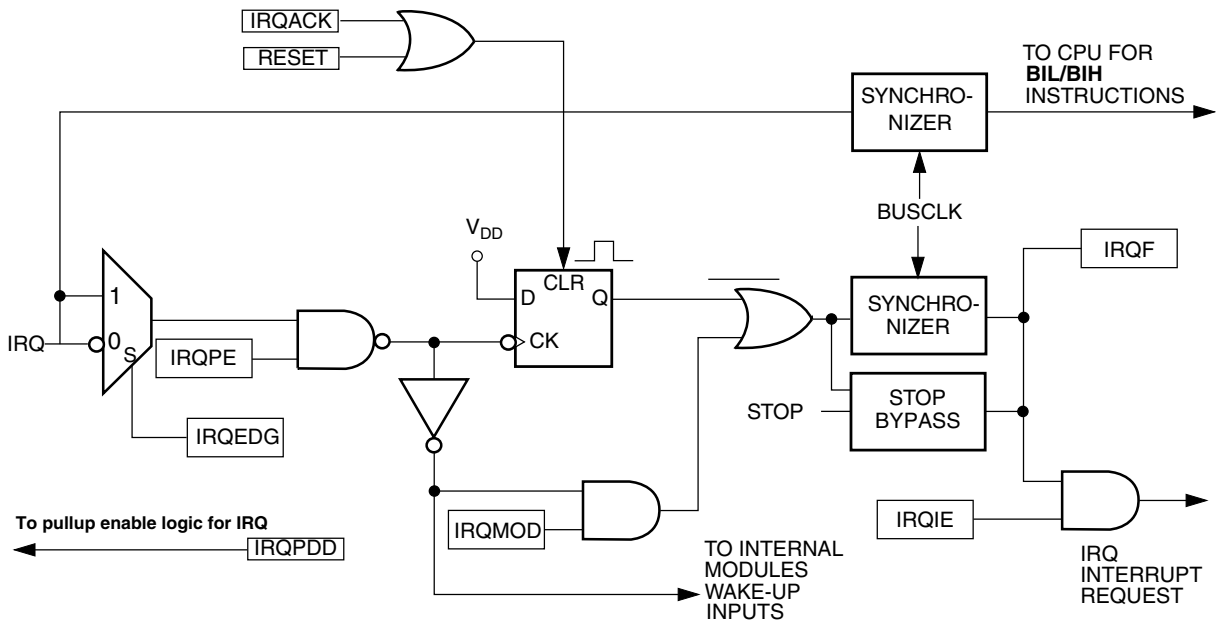



**Figure 5-3. IRQ module block diagram**

External interrupts are managed by the IRQSC status and control register. When the IRQ function is enabled, synchronous logic monitors the pin for edge-only or edge-and-level events. When the MCU is in stop mode and system clocks are shut down, a separate asynchronous path is used so that the IRQ, if enabled, can wake the MCU.

### 5.2.1.1 Pin configuration options

The IRQ pin enable control bit (IRQSC[IRQPE]) must be 1 for the IRQ pin to act as the IRQ input. The user can choose the polarity of edges or levels detected (IRQEDG), whether the pin detects edges-only or edges and levels (IRQMOD), or whether an event causes an interrupt or only sets the IRQF flag, which can be polled by software.

When enabled, the IRQ pin defaults to use an internal pullup device (IRQSC[IRQPDD] = 0). If the user uses an external pullup or pulldown, the IRQSC[IRQPDD] can be written to a 1 to turn off the internal device.

BIH and BIL instructions may be used to detect the level on the IRQ pin when it is configured to act as the IRQ input.

**Note**

This pin does not contain a clamp diode to $V_{DD}$ and must not be driven above $V_{DD}$. The voltage measured on the internally pullup IRQ pin may be as low as $V_{DD}$ – 0.7 V. The internal gates connected to this pin are pulled all the way to $V_{DD}$.

When enabling the IRQ pin for use, the IRQF will be set, and must be cleared prior to enabling the interrupt. When configuring the pin for falling edge and level sensitivity in a 3 V system, it is necessary to wait at least cycles between clearing the flag and enabling the interrupt.

### 5.2.1.2 Edge and level sensitivity

The IRQSC[IRQMOD] control bit reconfigures the detection logic so that it can detect edge events and pin levels. In this detection mode, the IRQF status flag is set when an edge is detected, if the IRQ pin changes from the de-asserted to the asserted level, but the flag is continuously set and cannot be cleared as long as the IRQ pin remains at the asserted level.

## 5.3 Interrupt pin request register

**IRQ memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 3B | Interrupt Pin Request Status and Control Register (IRQ_SC) | 8 | R/W | 00h | 5.3.1/108 |

## 5.3.1 Interrupt Pin Request Status and Control Register (IRQ_SC)

This direct page register includes status and control bits, which are used to configure the IRQ function, report status, and acknowledge IRQ events.

Address: 3Bh base + 0h offset = 3Bh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | IRQPDD | IRQEDG | IRQPE | IRQF | 0 | IRQIE | IRQMOD |
| Write | | | | | | IRQACK | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**IRQ_SC field descriptions**

| Field | Description |
|---|---|
| 7<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 6<br>IRQPDD | Interrupt Request (IRQ) Pull Device Disable<br><br>This read/write control bit is used to disable the internal pullup device when the IRQ pin is enabled (IRQPE = 1) allowing for an external device to be used.<br><br>0 IRQ pull device enabled if IRQPE = 1.<br>1 IRQ pull device disabled if IRQPE = 1. |
| 5<br>IRQEDG | Interrupt Request (IRQ) Edge Select<br><br>This read/write control bit is used to select the polarity of edges or levels on the IRQ pin that cause IRQF to be set. The IRQMOD control bit determines whether the IRQ pin is sensitive to both edges and levels or only edges. When the IRQ pin is enabled as the IRQ input and is configured to detect rising edges, the optional pullup resistor is disabled.<br><br>0 IRQ is falling edge or falling edge/low-level sensitive.<br>1 IRQ is rising edge or rising edge/high-level sensitive. |
| 4<br>IRQPE | IRQ Pin Enable<br><br>This read/write control bit enables the IRQ pin function. When this bit is set the IRQ pin can be used as an interrupt request.<br><br>0 IRQ pin function is disabled.<br>1 IRQ pin function is enabled. |
| 3<br>IRQF | IRQ Flag<br><br>This read-only status bit indicates when an interrupt request event has occurred.<br><br>0 No IRQ request.<br>1 IRQ event detected. |
| 2<br>IRQACK | IRQ Acknowledge<br><br>This write-only bit is used to acknowledge interrupt request events (write 1 to clear IRQF). Writing 0 has no meaning or effect. Reads always return 0. If edge-and-level detection is selected (IRQMOD = 1), IRQF cannot be cleared while the IRQ pin remains at its asserted level. |

*Table continues on the next page...*

**IRQ_SC field descriptions (continued)**

| Field | Description |
|---|---|
| 1<br>IRQIE | IRQ Interrupt Enable<br><br>This read/write control bit determines whether IRQ events generate an interrupt request.<br><br>0 Interrupt request when IRQF set is disabled (use polling).<br>1 Interrupt requested whenever IRQF = 1. |
| 0<br>IRQMOD | IRQ Detection Mode<br><br>This read/write control bit selects either edge-only detection or edge-and-level detection.<br><br>0 IRQ event on falling/rising edges only.<br>1 IRQ event on falling/rising edges and low/high levels. |

# 5.4 Interrupt priority control register

**IPC memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 3E | IPC Status and Control Register (IPC_SC) | 8 | R/W | 20h | 5.4.1/110 |
| 3F | Interrupt Priority Mask Pseudo Stack Register (IPC_IPMPS) | 8 | R | 00h | 5.4.2/111 |
| 3050 | Interrupt Level Setting Registers n (IPC_ILRS0) | 8 | R/W | 00h | 5.4.3/111 |
| 3051 | Interrupt Level Setting Registers n (IPC_ILRS1) | 8 | R/W | 00h | 5.4.3/111 |
| 3052 | Interrupt Level Setting Registers n (IPC_ILRS2) | 8 | R/W | 00h | 5.4.3/111 |
| 3053 | Interrupt Level Setting Registers n (IPC_ILRS3) | 8 | R/W | 00h | 5.4.3/111 |
| 3054 | Interrupt Level Setting Registers n (IPC_ILRS4) | 8 | R/W | 00h | 5.4.3/111 |
| 3055 | Interrupt Level Setting Registers n (IPC_ILRS5) | 8 | R/W | 00h | 5.4.3/111 |
| 3056 | Interrupt Level Setting Registers n (IPC_ILRS6) | 8 | R/W | 00h | 5.4.3/111 |
| 3057 | Interrupt Level Setting Registers n (IPC_ILRS7) | 8 | R/W | 00h | 5.4.3/111 |
| 3058 | Interrupt Level Setting Registers n (IPC_ILRS8) | 8 | R/W | 00h | 5.4.3/111 |
| 3059 | Interrupt Level Setting Registers n (IPC_ILRS9) | 8 | R/W | 00h | 5.4.3/111 |

## 5.4.1 IPC Status and Control Register (IPC_SC)

This register contains status and control bits for the IPC.

Address: 3Eh base + 0h offset = 3Eh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | IPCE | 0 | PSE | PSF | 0 | 0 | IPM | |
| Write | | | | | PULIPM | | | |
| Reset | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**IPC_SC field descriptions**

| Field | Description |
|---|---|
| 7<br>IPCE | Interrupt Priority Controller Enable<br><br>This bit enables/disables the interrupt priority controller module.<br><br>0 Disables IPCE. Interrupt generated from the interrupt source is passed directly to CPU without processing (bypass mode). The IPMPS register is not updated when the module is disabled.<br>1 Enables IPCE and interrupt generated from the interrupt source is processed by IPC before passing to CPU. |
| 6<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 5<br>PSE | Pseudo Stack Empty<br><br>This bit indicates that the pseudo stack has no valid information. This bit is automatically updated after each IPMPS register push or pull operation. |
| 4<br>PSF | Pseudo Stack Full<br><br>This bit indicates that the pseudo stack register IPMPS register is full. It is automatically updated after each IPMPS register push or pull operation. If additional interrupt is nested after this bit is set, the earliest interrupt mask value(IPM0[1:0]) stacked in IPMPS will be lost.<br><br>0 IPMPS register is not full.<br>1 IPMPS register is full. |
| 3<br>PULIPM | Pull IPM from IPMPS<br><br>This bit pulls stacked IPM value from IPMPS register to IPM bits of IPCSC. Zeros are shifted into bit positions 1 and 0 of IPMPS.<br><br>0 No operation.<br>1 Writing 1 to this bit causes a 2-bit value from the interrupt priority mask pseudo stack register to be pulled to the IPM bits of IPCSC to restore the previous IPM value. |
| 2<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| IPM | Interrupt Priority Mask<br><br>This field sets the mask for the interrupt priority control. If the interrupt priority controller is enabled, the interrupt source with an interrupt level (ILRxx) value that is greater than or equal to the value of IPM will be presented to the CPU. Writes to this field are allowed, but doing this will not push information to the |

*Table continues on the next page...*

**IPC_SC field descriptions (continued)**

| Field | Description |
|---|---|
| | IPMPS register. Writing IPM with PULIPM setting when IPCE is already set, the IPM will restore the value pulled from the IPMPS register, not the value written to the IPM register. |

## 5.4.2 Interrupt Priority Mask Pseudo Stack Register (IPC_IPMPS)

This register is used to store the previous interrupt priority mask level temporarily when the currently active interrupt is executed.

Address: 3Eh base + 1h offset = 3Fh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | IPM3 | | IPM2 | | IPM1 | | IPM0 | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**IPC_IPMPS field descriptions**

| Field | Description |
|---|---|
| 7–6<br>IPM3 | Interrupt Priority Mask pseudo stack position 3<br><br>This field is the pseudo stack register for IPM3. The most recent information is stored in IPM3. |
| 5–4<br>IPM2 | Interrupt Priority Mask pseudo stack position 2<br><br>This field is the pseudo stack register for IPM2. The most recent information is stored in IPM2. |
| 3–2<br>IPM1 | Interrupt Priority Mask pseudo stack position 1<br><br>This field is the pseudo stack register for IPM1. The most recent information is stored in IPM1. |
| IPM0 | Interrupt Priority Mask pseudo stack position 0<br><br>This field is the pseudo stack register for IPM0. The most recent information is stored in IPM0. |

## 5.4.3 Interrupt Level Setting Registers n (IPC_ILRS*n*)

This set of registers (ILRS0-ILRS9) contains the user specified interrupt level for each interrupt source, and indicates the number of the register (ILRSn is ILRS0 through ILRS9).

Address: 3Eh base + 3012h offset + (1d × i), where i=0d to 9d

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | ILRn3 | | ILRn2 | | ILRn1 | | ILRn0 | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## IPC_ILRS*n* field descriptions

| Field | Description |
|---|---|
| 7–6<br>ILRn3 | Interrupt Level Register for Source n*4+3<br><br>This field sets the interrupt level for interrupt source n*4+3. |
| 5–4<br>ILRn2 | Interrupt Level Register for Source n*4+2<br><br>This field sets the interrupt level for interrupt source n*4+2. |
| 3–2<br>ILRn1 | Interrupt Level Register for Source n*4+1<br><br>This field sets the interrupt level for interrupt source n*4+1. |
| ILRn0 | Interrupt Level Register for Source n*4+0<br><br>This field sets the interrupt level for interrupt source n*4+0. |

# Chapter 6
# System control

## 6.1 System device identification (SDID)

This device is hard coded to the value 0x0042 in SDID registers.

## 6.2 Universally unique identification (UUID)

This device contains up to 64-bit UUID to identify each device in this family. The intent of UUID is to enable distributed systems to uniquely identify information without significant central coordination.

## 6.3 Reset and system initialization

Resetting the MCU provides a way to start processing from a set of known initial conditions. During reset, most control and status registers are forced to initial values and the program counter is loaded from the reset vector (0xFFFE:0xFFFF). On-chip peripheral modules are disabled and I/O pins are initially configured as general-purpose high-impedance inputs with disabled pullup devices. The CCR[I] bit is set to block maskable interrupts so that the user program has a chance to initialize the stack pointer (SP) and system control settings. SP is forced to 0x00FF at reset.

This device has the following sources for reset:

- Power-on reset (POR)
- Low-voltage detect (LVD)
- Watchdog (WDOG) timer
- Illegal opcode detect (ILOP)

- Illegal address detect (ILAD)
- Background debug forced reset
- External reset pin (RESET)
- Internal clock source module reset (CLK)

Each of these sources, with the exception of the background debug forced reset, has an associated bit in the system reset status (SRS) register.

When the MCU is reset by ILAD, the address of illegal address is captured in illegal address register, which is a 16-bit register consisting of ILLAL and ILLAH that contains the LSB and MSB 8-bit of the address, respectively.

## 6.4 System options

### 6.4.1 BKGD pin enable

After POR, PTA4/BKGD/MS pin functions as BKGD output. The SYS_SOPT1[BKGDPE] bit must be set to enable the background debug mode pin enable function. When this bit is clear, this pin can function as PTA4 output.

### 6.4.2 $\overline{\text{RESET}}$ pin enable

After POR reset, PTA5/IRQ/TCLK0/$\overline{\text{RESET}}$ functions as $\overline{\text{RESET}}$. The SYS_SOPT1[RSTPE] bit must be set to enable the reset functions. When this bit is clear, this pin can function as PTA5, IRQ, or TCLK0.

### 6.4.3 SCI0 pin reassignment

After reset, SCI0 module pinouts of RxD and TxD are mapped on PTB0 and PTB1, respectively. SYS_SOPT1[SCI0PS] bit enables to reassign SCI0 pinouts on PTA2 and PTA3.

### 6.4.4 FTM0 channels pin reassignment

After reset, FTM0 channels pinouts of FTM0CH0 and FTM0CH1 are mapped on PTA0 and PTA1, respectively. SOPT3[FTM0PS] bit enables to reassign FTM0 channels pinouts on PTC4 and PTC5.

### 6.4.5 FTM2 channels pin reassignment

After POR reset, FTM2 channel pinouts of FTM2CH2, and FTM2CH3 are default mapped on PTC2, and PTC3. When set, SYS_SOPT1[FTM2PS] bit enables to reassignment these FTM2 channels on PTD0, and PTD1, respectively.

### 6.4.6 Bus clock output pin enable

The system bus clock can be outputted on PTE3 when the SYS_SOPT3[CLKOE] bits are set by nonzero. Before mapping on the pinout, the output of bus clock can be pre-divided by 1, 2, 4, 8, 16, 32, 64, or 128 by setting SYS_SOPT3[BUSREF].

## 6.5 System interconnection

This device contains a set of system-level logics for module-to-module interconnection for flexible configuration. These interconnections provide the hardware trigger function between modules with least software configuration, which is ideal for infrared communication, serial communication baudrate detection, low-end motor control, metering clock calibration, and other general-purpose applications.

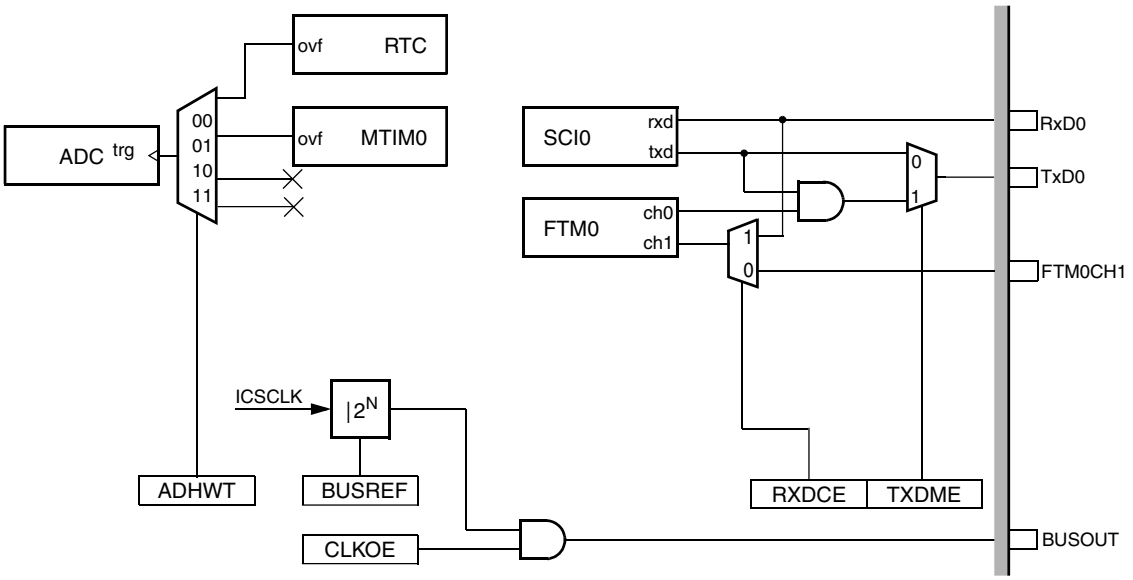



**Figure 6-1. System interconnection diagram**

## 6.5.1 SCI0 RxD capture

RxD0 pin is selectable connected to SCI0 module directly or tagged to FTM0 channel 1. When SYS_SOPT2[RXDCE] bit is set, the RxD0 pin is connected to both SCI0 and FTM0 channel 1, and the FTM0CH1 pin is released to other shared functions regardless of the configuration of FTM0 pin reassignment. When this bit is clear, the RxD0 pin is connected to SCI0 only.

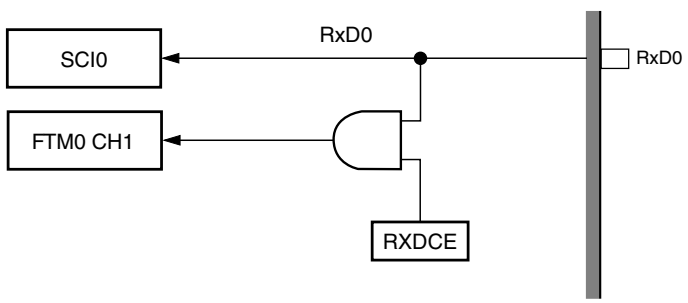



**Figure 6-2. RxD0 capture function diagram**

## 6.5.2 ADC hardware trigger

ADC module may initiate a conversion via a hardware trigger. MTIM0 overflow can be enabled as the hardware trigger for the ADC module by setting the SYS_SOPT2[ADHWT] bits. The following table shows the ADC hardware trigger setting.

**Table 6-1. ADC hardware trigger setting**

| ADHWT | ADC hardware trigger |
|---|---|
| 0:0 | RTC overflow |
| 0:1 | MTIM0 overflow |
| 1:0 | Reserved |
| 1:1 | Reserved |

## 6.6 System Control Registers

**SYS memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 3000 | System Reset Status Register (SYS_SRS) | 8 | R | 82h | 6.6.1/117 |
| 3001 | System Background Debug Force Reset Register (SYS_SBDFR) | 8 | W (always reads 0) | 00h | 6.6.2/119 |
| 3002 | System Device Identification Register: High (SYS_SDIDH) | 8 | R | 00h | 6.6.3/120 |
| 3003 | System Device Identification Register: Low (SYS_SDIDL) | 8 | R | 42h | 6.6.4/120 |
| 3004 | System Options Register 1 (SYS_SOPT1) | 8 | R/W | 0Ch | 6.6.5/121 |
| 3005 | System Options Register 2 (SYS_SOPT2) | 8 | R/W | 00h | 6.6.6/122 |
| 3006 | System Options Register 3 (SYS_SOPT3) | 8 | R/W | 00h | 6.6.7/123 |
| 304A | Illegal Address Register: High (SYS_ILLAH) | 8 | R | Undefined | 6.6.8/124 |
| 304B | Illegal Address Register: Low (SYS_ILLAL) | 8 | R | Undefined | 6.6.9/125 |
| 30F8 | Universally Unique Identifier Register 1 (SYS_UUID1) | 8 | R | Undefined | 6.6.10/125 |
| 30F9 | Universally Unique Identifier Register 2 (SYS_UUID2) | 8 | R | Undefined | 6.6.11/126 |
| 30FA | Universally Unique Identifier Register 3 (SYS_UUID3) | 8 | R | Undefined | 6.6.12/126 |
| 30FB | Universally Unique Identifier Register 4 (SYS_UUID4) | 8 | R | Undefined | 6.6.13/127 |
| 30FC | Universally Unique Identifier Register 5 (SYS_UUID5) | 8 | R | Undefined | 6.6.14/127 |
| 30FD | Universally Unique Identifier Register 6 (SYS_UUID6) | 8 | R | Undefined | 6.6.15/128 |
| 30FE | Universally Unique Identifier Register 7 (SYS_UUID7) | 8 | R | Undefined | 6.6.16/128 |
| 30FF | Universally Unique Identifier Register 8 (SYS_UUID8) | 8 | R | Undefined | 6.6.17/129 |

### 6.6.1 System Reset Status Register (SYS_SRS)

This register includes read-only status flags to indicate the source of the most recent reset. When a debug host forces reset by writing 1 to the SYS_SBDFR[BDFR] bit, none of the status bits in SRS will be set. The reset state of these bits depends on what caused the MCU to reset.

**NOTE**

For PIN, WDOG, and ILOP, any of these reset sources that are active at the time of reset (not including POR or LVR) will cause the corresponding bit(s) to be set; bits corresponding to sources that are not active at the time of reset will be cleared.

**NOTE**

The RESET values in the figure are values for power on reset; for other resets, the values depend on the trigger causes.

Address: 3000h base + 0h offset = 3000h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | POR | PIN | WDOG | ILOP | ILAD | LOC | LVD | 0 |
| Write | | | | | | | | |
| Reset | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

**SYS_SRS field descriptions**

| Field | Description |
|---|---|
| 7<br>POR | Power-On Reset<br><br>Reset was caused by the power-on detection logic. When the internal supply voltage was ramping up at the time, the low-voltage reset (LVR) status bit is also set to indicate that the reset occurred while the internal supply was below the LVR threshold.<br><br>**NOTE:** This bit POR to 1, LVR to uncertain value and reset to 0 at any other conditions.<br><br>0 Reset not caused by POR.<br>1 POR caused reset. |
| 6<br>PIN | External Reset Pin<br><br>Reset was caused by an active low level on the external reset pin.<br><br>0 Reset not caused by external reset pin.<br>1 Reset came from external reset pin. |
| 5<br>WDOG | Watchdog (WDOG)<br><br>Reset was caused by the WDOG timer timing out. This reset source may be blocked by WDOGE = 0.<br><br>0 Reset not caused by WDOG timeout.<br>1 Reset caused by WDOG timeout. |
| 4<br>ILOP | Illegal Opcode<br><br>Reset was caused by an attempt to execute an unimplemented or illegal opcode. The STOP instruction is considered illegal if stop is disabled by STOPE = 0 in the SOPT register. The BGND instruction is considered illegal if active background mode is disabled by ENBDM = 0 in the BDCSC register.<br><br>0 Reset not caused by an illegal opcode.<br>1 Reset caused by an illegal opcode. |
| 3<br>ILAD | Illegal Address |

*Table continues on the next page...*

**SYS_SRS field descriptions (continued)**

| Field | Description |
|---|---|
| | Reset was caused by an attempt to access a illegal address. The illegal address is captured in illegal address register (ILLAH:ILLAL).<br><br>0 Reset not caused by an illegal address.<br>1 Reset caused by an illegal address. |
| 2<br>LOC | Internal Clock Source Module Reset<br><br>Reset was caused by an ICS module reset.<br><br>0 Reset not caused by ICS module.<br>1 Reset caused by ICS module. |
| 1<br>LVD | Low Voltage Detect<br><br>If the LVDRE bit is set in run mode or both LVDRE and LVDSE bits are set in stop mode, and the supply drops below the LVD trip voltage, an LVD reset will occur. This bit is also set by POR.<br><br>**NOTE:** This bit reset to 1 on POR and LVR and reset to 0 on other reset.<br><br>0 Reset not caused by LVD trip or POR.<br>1 Reset caused by LVD trip or POR. |
| 0<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 6.6.2 System Background Debug Force Reset Register (SYS_SBDFR)

This register contains a single write-only control bit. A serial background command such as WRITE_BYTE must be used to write to SYS_SBDFR. Attempts to write this register from a user program are ignored. Reads always return 0x00.

**NOTE**

This register is the same as the BDC_SBDFR.

Address: 3000h base + 1h offset = 3001h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | | | | | 0 |
| Write | | | | | | | | BDFR |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SYS_SBDFR field descriptions**

| Field | Description |
|---|---|
| 7–1<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 0<br>BDFR | Background Debug Force Reset |

*Table continues on the next page...*

**SYS_SBDFR field descriptions (continued)**

| Field | Description |
|---|---|
| | A serial background command such as WRITE_BYTE may be used to allow an external debug host to force a target system reset. Writing logic 1 to this bit forces an MCU reset. This bit cannot be written from a user program.<br><br>**NOTE:** BDFR is writable only through serial background debug commands, not from user programs. |

## 6.6.3 System Device Identification Register: High (SYS_SDIDH)

This read-only register, together with SYS_SDIDL, is included so that host development systems can identify the HCS08 derivative and revision number. This allows the development software to recognize where specific memory blocks, registers, and control bits are located in a target MCU.

Address: 3000h base + 2h offset = 3002h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | Reserved | | | | ID | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SYS_SDIDH field descriptions**

| Field | Description |
|---|---|
| 7–4<br>Reserved | This field is reserved. |
| ID | Part Identification Number<br><br>These bits, together with the SYS_SDIDL, indicate part identification number. Each derivative in the HCS08 family has a unique identification number. This device is hard coded to the value . |

## 6.6.4 System Device Identification Register: Low (SYS_SDIDL)

This read-only register, together with SYS_SDIDH, is included so host development systems can identify the HCS08 derivative and revision number. This allows the development software to recognize where specific memory blocks, registers, and control bits are located in a target MCU.

Address: 3000h base + 3h offset = 3003h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ID | | | | | | | |
| Write | | | | | | | | |
| Reset | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |

**SYS_SDIDL field descriptions**

| Field | Description |
|---|---|
| ID | Part Identification Number<br><br>These bits, together with the SYS_SDIDH, indicate part identification number. Each derivative in the HCS08 family has a unique identification number. This device is hard coded to the value . |

## 6.6.5 System Options Register 1 (SYS_SOPT1)

Address: 3000h base + 4h offset = 3004h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | SCI0PS | 0 | | FTM2PS | BKGDPE | RSTPE | FWAKE | STOPE |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

**SYS_SOPT1 field descriptions**

| Field | Description |
|---|---|
| 7<br>SCI0PS | SCI0 Pin Select<br><br>This write-once bit selects the SCI0 pinouts.<br><br>0 SCI0 RxD and TxD are .<br>1 SCI0 RxD and TxD are mapped on PTA2 and PTA3. |
| 6–5<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 4<br>FTM2PS | FTM2 Port Pin Select<br><br>This write-once bit selects the FTM2 channels port pins.<br><br>0 FTM2 channels mapped on PTC0, PTC1, PTC2, PTC3, PTB4, and PTB5.<br>1 FTM2 channels mapped on PTC0, PTC1, PTD0, PTD1, PTB4, and PTB5. |
| 3<br>BKGDPE | Background Debug Mode Pin Enable<br><br>This write-once bit when set enables the PTA4/BKGD/MS pin to function as BKGD/MS. When clear, the pin functions as output only PTA4. This pin defaults to the BKGD/MS function following any MCU reset.<br><br>0 PTA4/BKGD/MS as PTA4 function.<br>1 PTA4/BKGD/MS as BKGD function. |
| 2<br>RSTPE | RESET Pin Enable<br><br>This write-once bit can be written after any reset. When RSTPE is set, the PTA5/IRQ/TCLK0/RESET pin functions as RESET. When clear, the pin functions as one of its alternative functions. This pin defaults to RESET following an MCU POR. Other resets will not affect this bit. When RSTPE is set, an internal pullup device on RESET is enabled.<br><br>0 PTA5/IRQ/TCLK0/RESET pin functions as PTA5, IRQ, or TCLK0.<br>1 PTA5/IRQ/TCLK0/RESET pin functions as RESET. |
| 1<br>FWAKE | Fast Wakeup Enable |

*Table continues on the next page...*

**SYS_SOPT1 field descriptions (continued)**

| Field | Description |
|---|---|
| | This write once bit can set CPU wakeup without any interrupt subroutine serviced. This action saved more than 11 cycles(whole interrupt subroutine time). After wake up CPU continue the address before wait or stop.<br>**NOTE:** When FWAKE is set, user should avoid generating interrupt 0~8 bus clock cycles after issuing the stop instruction, or the MCU may stuck at stop3 mode and cannot wake up by interrupts.<br>0 CPU wakes up as normal.<br>1 CPU wakes up without any interrupt subroutine serviced. |
| 0<br>STOPE | Stop Mode Enable<br>This write-once bit defaults to 0 after reset, which disables stop mode. If stop mode is disabled and a user program attempts to execute a STOP instruction, an illegal opcode reset occurs.<br>0 Stop mode disabled.<br>1 Stop mode enabled. |

## 6.6.6 System Options Register 2 (SYS_SOPT2)

This register may be read/write at any time. SYS_SOPT2 should be written during the user's reset initialization program to set the desired controls even if the desired settings are the same as the reset settings.

Address: 3000h base + 5h offset = 3005h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | TXDME | 0 | 0 | RXDCE | 0 | | ADHWTS | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SYS_SOPT2 field descriptions**

| Field | Description |
|---|---|
| 7<br>TXDME | SCI0 TxD Modulation Select<br>This bit enables the SCI0 TxD output modulated by FTM0 channel 0.<br>0 TxD0 output is connected to pinout directly.<br>1 TxD0 output is modulated by FTM0 channel 0 before mapped to pinout. |
| 6<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 5<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 4<br>RXDCE | SCI0 RxD Capture Select<br>This bit enables the SCI0 RxD is captured by FTM0 channel 1. |

*Table continues on the next page...*

**SYS_SOPT2 field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 RXD0 input signal is connected to SCI0 module only.<br>1 RXD0 input signal is connected to SCI0 module and FTM0 channel 1. |
| 3–2<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| ADHWTS | ADC Hardware Trigger Source<br><br>These bits select the ADC hardware trigger source. All trigger sources start ADC conversion on rising edge.<br><br>00 RTC overflow as the ADC hardware trigger.<br>01 MTIM0 overflow as the ADC hardware trigger.<br>10 Reserved<br>11 Reserved |

## 6.6.7 System Options Register 3 (SYS_SOPT3)

This register may be read and written at any time.

Address: 3000h base + 6h offset = 3006h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | FTM0PS | 0 | | CLKOE | BUSREF | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SYS_SOPT3 field descriptions**

| Field | Description |
|---|---|
| 7<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 6<br>FTM0PS | FTM0 Pin Select<br><br>This write-once bit selects the FTM0 Pinouts.<br><br>0 FTM0CH0 and FTM0CH1 are mapped on PTA0 and PTA1.<br>1 FTM0CH0 and FTM0CH1 are mapped on PTC4 and PTC5. |
| 5–4<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 3<br>CLKOE | CLK Output Enable<br><br>This bit enables reference clock output on PTE3<br><br>0 ICSCLK output disabled on PTE3.<br>1 ICSCLK output enabled on PTE3. |
| BUSREF | BUS Output select<br><br>This bit enables bus clock output on PTE3 via an optional prescalar. |

*Table continues on the next page...*

**SYS_SOPT3 field descriptions (continued)**

| Field | Description |
|---|---|
| | 000 Bus.<br>001 Bus divided by 2.<br>010 Bus divided by 4.<br>011 Bus divided by 8.<br>100 Bus divided by 16.<br>101 Bus divided by 32.<br>110 Bus divided by 64.<br>111 Bus divided by 128. |

## 6.6.8 Illegal Address Register: High (SYS_ILLAH)

The SYS_ILLAH is a read-only register containing the high 8-bit of the illegal address of ILAD reset.

Address: 3000h base + 4Ah offset = 304Ah

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ADDR[15:8] | | | | | | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**SYS_ILLAH field descriptions**

| Field | Description |
|---|---|
| ADDR[15:8] | High 8-bit of illegal address<br><br>**NOTE:** For ILAD, it reset to the high 8-bit of the illegal address; in other cases, the reset to values are undetermined. |

## 6.6.9 Illegal Address Register: Low (SYS_ILLAL)

The SYS_ILLAL is a read-only register containing the low 8-bit of the illegal address of ILAD reset.

Address: 3000h base + 4Bh offset = 304Bh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ADDR[7:0] | | | | | | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**SYS_ILLAL field descriptions**

| Field | Description |
|---|---|
| ADDR[7:0] | Low 8-bit of illegal address<br><br>**NOTE:** For ILAD, it resets to the low 8-bit of the illegal address; in other cases, the reset to values are undetermined. |

## 6.6.10 Universally Unique Identifier Register 1 (SYS_UUID1)

The read-only SYS_UUIDx registers contain a series of 64-bit number to identify the unique device in the family.

Address: 3000h base + F8h offset = 30F8h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ID[63:56] | | | | | | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**SYS_UUID1 field descriptions**

| Field | Description |
|---|---|
| ID[63:56] | Universally Unique Identifier |

## 6.6.11 Universally Unique Identifier Register 2 (SYS_UUID2)

The read-only SYS_UUIDx registers contain a series of 63-bit number to identify the unique device in the family.

Address: 3000h base + F9h offset = 30F9h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ID[55:48] | | | | | | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**SYS_UUID2 field descriptions**

| Field | Description |
|---|---|
| ID[55:48] | Universally Unique Identifier |

## 6.6.12 Universally Unique Identifier Register 3 (SYS_UUID3)

The read-only SYS_UUIDx registers contain a series of 63-bit number to identify the unique device in the family.

Address: 3000h base + FAh offset = 30FAh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ID[47:40] | | | | | | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**SYS_UUID3 field descriptions**

| Field | Description |
|---|---|
| ID[47:40] | Universally Unique Identifier |

## 6.6.13 Universally Unique Identifier Register 4 (SYS_UUID4)

The read-only SYS_UUIDx registers contain a series of 63-bit number to identify the unique device in the family.

Address: 3000h base + FBh offset = 30FBh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ID[39:32] | | | | | | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**SYS_UUID4 field descriptions**

| Field | Description |
|---|---|
| ID[39:32] | Universally Unique Identifier |

## 6.6.14 Universally Unique Identifier Register 5 (SYS_UUID5)

The read-only SYS_UUIDx registers contain a series of 64-bit number to identify the unique device in the family.

Address: 3000h base + FCh offset = 30FCh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ID[31:24] | | | | | | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**SYS_UUID5 field descriptions**

| Field | Description |
|---|---|
| ID[31:24] | Universally Unique Identifier |

## 6.6.15 Universally Unique Identifier Register 6 (SYS_UUID6)

The read-only SYS_UUIDx registers contain a series of 64-bit number to identify the unique device in the family.

Address: 3000h base + FDh offset = 30FDh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ID[23:16] | | | | | | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**SYS_UUID6 field descriptions**

| Field | Description |
|---|---|
| ID[23:16] | Universally Unique Identifier |

## 6.6.16 Universally Unique Identifier Register 7 (SYS_UUID7)

The read-only SYS_UUIDx registers contain a series of 64-bit number to identify the unique device in the family.

Address: 3000h base + FEh offset = 30FEh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ID[15:8] | | | | | | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**SYS_UUID7 field descriptions**

| Field | Description |
|---|---|
| ID[15:8] | Universally Unique Identifier |

## 6.6.17 Universally Unique Identifier Register 8 (SYS_UUID8)

The read-only SYS_UUIDx registers contain a series of 64-bit number to identify the unique device in the family.

Address: 3000h base + FFh offset = 30FFh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ID[7:0] | | | | | | | |
| Write | | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**SYS_UUID8 field descriptions**

| Field | Description |
|---|---|
| ID[7:0] | Universally Unique Identifier |

# Chapter 7
# Parallel input/output

## 7.1 Introduction

This device has five set of I/O ports, which include up to 37 general-purpose I/O pins.

Not all pins are available on all devices. See Table 2-1 to determine which functions are available for a specific device.

Many of the I/O pins are shared with on-chip peripheral functions, as shown in Table 2-1. The peripheral modules have priority over the I/O, so when a peripheral is enabled, the associated I/O functions are disabled.

After reset, the shared peripheral functions are disabled so that the pins are controlled by the parallel I/O except PTA4 and PTA5 that are default to BKGD/MS and $\overline{\text{RESET}}$ function. All of the parallel I/O are configured as high-impedance (Hi-Z). The pin control functions for each pin are configured as follows:

- input disabled (PTxIEn = 0),
- output disabled (PTxOEn = 0), and
- internal pullups disabled (PTxPEn = 0).

The following figures show the structures of each I/O pin.

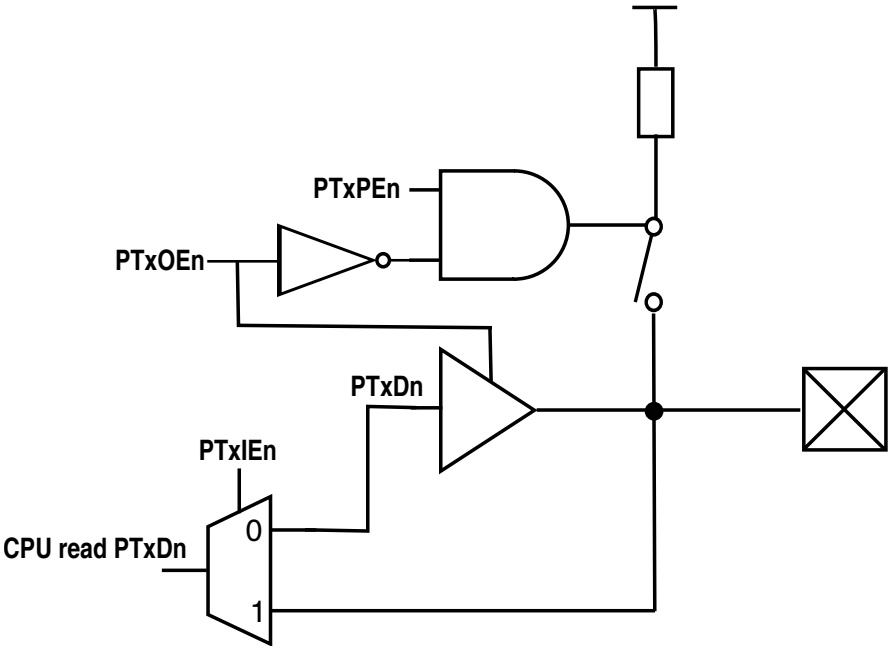



**Figure 7-1. Normal I/O structure**

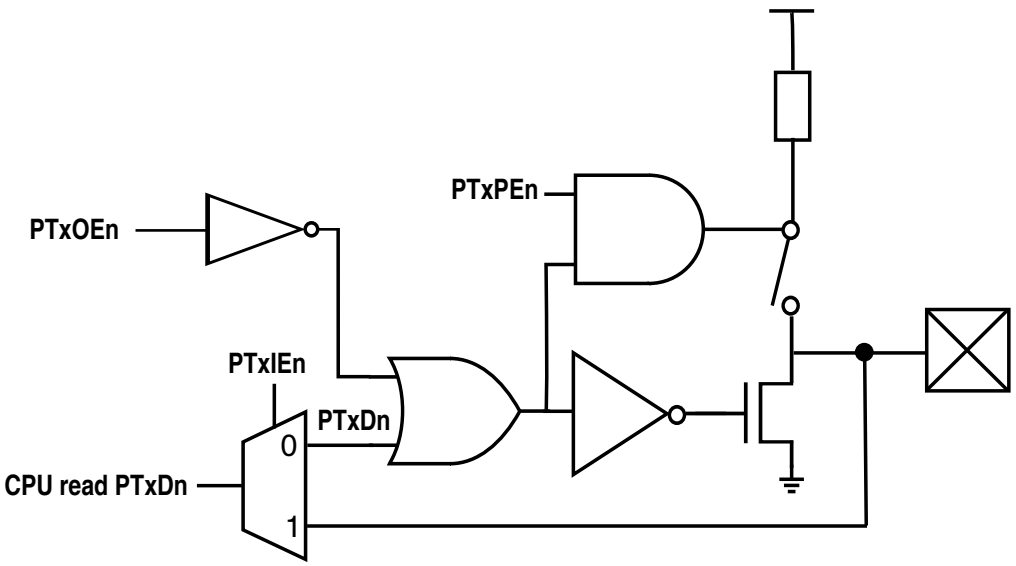



**Figure 7-2. SDA(PTA2)/SCL(PTA3) structure**

## 7.2 Port data and data direction

Reading and writing of parallel I/O is accomplished through the port data registers (PTxD). The direction, input or output, is controlled through the input enable or output enable registers.

After reset, all parallel I/O default to the Hi-Z state. The corresponding bit in output enable register (PTxOE) or input enable register (PTxIE) must be configured for output or input operation. Each port pin has an input enable bit and an output enable bit. When PTxIEn = 1, a read from PTxDn returns the input value of the associated pin; when PTxIEn = 0, a read from PTxDn returns the last value written to the port data register.

**NOTE**

The PTxOE must be clear when the corresponding pin is used as input function to avoid contention. If set the corresponding PTxOE and PTxIE bits at same time, read from PTxDn will always return the output data.

When a peripheral module or system function is in control of a port pin, the data direction register bit still controls what is returned for reads of the port data register, even though the peripheral system has overriding control of the actual pin direction.

When a shared analog function is enabled for a pin, all digital pin functions are disabled. A read of the port data register returns a value of 0 for any bits that have shared analog functions enabled. In general, whenever a pin is shared with both an alternate digital function and an analog function, the analog function has priority such that if both of the digital and analog functions are enabled, the analog function controls the pin.

A write of valid data to a port data register must occur before setting the output enable bit of an associated port pin. This ensures that the pin will not be driven with an incorrect data value.

## 7.3 Internal pullup enable

An internal pullup device can be enabled for each port pin by setting the corresponding bit in one of the pullup enable registers (PTxPEn). The internal pullup device is disabled if the pin is configured as an output by the parallel I/O control logic, or by any shared peripheral function, regardless of the state of the corresponding pullup enable register bit. The internal pullup device is also disabled if the pin is controlled by an analog function.

## 7.4 Input glitch filter setting

A filter is implemented for each port pin that is configured as a digital input. It can be used as a simple low-pass filter to filter any glitch that is introduced from the pins of GPIO, IRQ,$\overline{\text{RESET}}$, and KBI. The glitch width threshold can be adjusted easily by

setting registers PORT_IOFLTn and PORT_FCLKDIV between 1~4096 BUSCLKs (or 1~128 LPOCLKs). This configurable glitch filter can take the place of an on board external analog filter, and greatly improve the EMC performance.

Setting register PORT_IOFLTn can configure the filter of the whole port, etc. set PORT_IOFLT0[FLTA] will affect all PTAn pins.

## 7.5 Pin behavior in stop mode

In stop3 mode, all I/O is maintained because internal logic circuitry stays powered up. Upon recovery, normal I/O function is available to the user.

## 7.6 Port data registers

**PORT memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 0 | Port A Data Register (PORT_PTAD) | 8 | R/W | 00h | 7.6.1/135 |
| 1 | Port B Data Register (PORT_PTBD) | 8 | R/W | 00h | 7.6.2/135 |
| 2 | Port C Data Register (PORT_PTCD) | 8 | R/W | 00h | 7.6.3/136 |
| 3 | Port D Data Register (PORT_PTDD) | 8 | R/W | 00h | 7.6.4/136 |
| 4 | Port E Data Register (PORT_PTED) | 8 | R/W | 00h | 7.6.5/137 |
| 30B0 | Port A Output Enable Register (PORT_PTAOE) | 8 | R/W | 00h | 7.6.6/138 |
| 30B1 | Port B Output Enable Register (PORT_PTBOE) | 8 | R/W | 00h | 7.6.7/139 |
| 30B2 | Port C Output Enable Register (PORT_PTCOE) | 8 | R/W | 00h | 7.6.8/140 |
| 30B3 | Port D Output Enable Register (PORT_PTDOE) | 8 | R/W | 00h | 7.6.9/141 |
| 30B4 | Port E Output Enable Register (PORT_PTEOE) | 8 | R/W | 00h | 7.6.10/142 |
| 30B8 | Port A Input Enable Register (PORT_PTAIE) | 8 | R/W | 00h | 7.6.11/143 |
| 30B9 | Port B Input Enable Register (PORT_PTBIE) | 8 | R/W | 00h | 7.6.12/144 |
| 30BA | Port C Input Enable Register (PORT_PTCIE) | 8 | R/W | 00h | 7.6.13/145 |
| 30BB | Port D Input Enable Register (PORT_PTDIE) | 8 | R/W | 00h | 7.6.14/146 |
| 30BC | Port E Input Enable Register (PORT_PTEIE) | 8 | R/W | 00h | 7.6.15/147 |
| 30EC | Port Filter Register 0 (PORT_IOFLT0) | 8 | R/W | 00h | 7.6.16/148 |
| 30ED | Port Filter Register 1 (PORT_IOFLT1) | 8 | R/W | 00h | 7.6.17/149 |
| 30EE | Port Filter Register 2 (PORT_IOFLT2) | 8 | R/W | 00h | 7.6.18/149 |
| 30EF | Port Clock Division Register (PORT_FCLKDIV) | 8 | R/W | 00h | 7.6.19/150 |
| 30F0 | Port A Pullup Enable Register (PORT_PTAPE) | 8 | R/W | 00h | 7.6.20/151 |
| 30F1 | Port B Pullup Enable Register (PORT_PTBPE) | 8 | R/W | 00h | 7.6.21/152 |

*Table continues on the next page...*

**PORT memory map (continued)**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 30F2 | Port C Pullup Enable Register (PORT_PTCPE) | 8 | R/W | 00h | 7.6.22/153 |
| 30F3 | Port D Pullup Enable Register (PORT_PTDPE) | 8 | R/W | 00h | 7.6.23/155 |
| 30F4 | Port E Pullup Enable Register (PORT_PTEPE) | 8 | R/W | 00h | 7.6.24/156 |

## 7.6.1 Port A Data Register (PORT_PTAD)

Address: 0h base + 0h offset = 0h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | PTAD | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTAD field descriptions**

| Field | Description |
|---|---|
| PTAD | Port A Data Register Bits<br>For port A pins that are configured as inputs, a read returns the logic level on the pin.<br>For port A pins that are configured as outputs, a read returns the last value that was written to this register.<br>For port A pins that are configured as Hi-Z, a read returns uncertainty data.<br>Writes are latched into all bits of this register. For port A pins that are configured as outputs, the logic level is driven out of the corresponding MCU pin.<br>Reset forces PTAD to all 0s, but these 0s are not driven out of the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled. |

## 7.6.2 Port B Data Register (PORT_PTBD)

Address: 0h base + 1h offset = 1h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | PTBD | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTBD field descriptions**

| Field | Description |
|---|---|
| PTBD | Port B Data Register Bits<br>For port B pins that are configured as inputs, a read returns the logic level on the pin.<br>For port B pins that are configured as outputs, a read returns the last value that was written to this register.<br>For port B pins that are configured as Hi-Z, a read returns uncertainty data. |

**PORT_PTBD field descriptions (continued)**

| Field | Description |
|---|---|
| | Writes are latched into all bits of this register. For port B pins that are configured as outputs, the logic level is driven out of the corresponding MCU pin.<br>Reset forces PTBD to all 0s, but these 0s are not driven out of the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled. |

## 7.6.3 Port C Data Register (PORT_PTCD)

Address: 0h base + 2h offset = 2h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | PTCD | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTCD field descriptions**

| Field | Description |
|---|---|
| PTCD | Port C Data Register Bits<br>For port C pins that are configured as inputs, a read returns the logic level on the pin.<br>For port C pins that are configured as outputs, a read returns the last value that was written to this register.<br>For port C pins that are configured as Hi-Z, a read returns uncertainty data.<br>Writes are latched into all bits of this register. For port C pins that are configured as outputs, the logic level is driven out of the corresponding MCU pin.<br>Reset forces PTCD to all 0s, but these 0s are not driven out of the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled. |

## 7.6.4 Port D Data Register (PORT_PTDD)

For port D pins that are configured as inputs, a read returns the logic level on the pin.

For port D pins that are configured as outputs, a read returns the last value that was written to this register.

For port D pins that are configured as Hi-Z, a read returns uncertainty data.

Writes are latched into all bits of this register. For port D pins that are configured as outputs, the logic level is driven out of the corresponding MCU pin.

Reset forces PTDD to all 0s, but these 0s are not driven out of the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled.

Address: 0h base + 3h offset = 3h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | PTDD7 | PTDD6 | 0 | | | PTDD2 | PTDD1 | PTDD0 |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTDD field descriptions**

| Field | Description |
|---|---|
| 7<br>PTDD7 | Port D Data Register Bit 7 |
| 6<br>PTDD6 | Port D Data Register Bit 6 |
| 5–3<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 2<br>PTDD2 | Port D Data Register Bit 2 |
| 1<br>PTDD1 | Port D Data Register Bit 1 |
| 0<br>PTDD0 | Port D Data Register Bit 0 |

## 7.6.5 Port E Data Register (PORT_PTED)

Address: 0h base + 4h offset = 4h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | PTED | 0 | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTED field descriptions**

| Field | Description |
|---|---|
| 7–5<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 4<br>PTED | Port E Data Register Bits<br>For port E pins that are configured as inputs, a read returns the logic level on the pin.<br>For port E pins that are configured as outputs, a read returns the last value that was written to this register.<br>For port E pins that are configured as Hi-Z, a read returns uncertainty data.<br>Writes are latched into all bits of this register. For port E pins that are configured as outputs, the logic level is driven out of the corresponding MCU pin.<br>Reset forces PTED to all 0s, but these 0s are not driven out of the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 7.6.6 Port A Output Enable Register (PORT_PTAOE)

Address: 0h base + 30B0h offset = 30B0h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | PTAOE7 | PTAOE6 | PTAOE5 | PTAOE4 | PTAOE3 | PTAOE2 | PTAOE1 | PTAOE0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTAOE field descriptions**

| Field | Description |
|---|---|
| 7<br>PTAOE7 | Output Enable for Port A Bit 7<br><br>This read/write bit enables the port A pin as an output.<br><br>0 Output Disabled for port A bit 7.<br>1 Output Enabled for port A bit 7. |
| 6<br>PTAOE6 | Output Enable for Port A Bit 6<br><br>This read/write bit enables the port A pin as an output.<br><br>0 Output Disabled for port A bit 6.<br>1 Output Enabled for port A bit 6. |
| 5<br>PTAOE5 | Output Enable for Port A Bit 5<br><br>This read/write bit enables the port A pin as an output.<br><br>0 Output Disabled for port A bit 5.<br>1 Output Enabled for port A bit 5. |
| 4<br>PTAOE4 | Output Enable for Port A Bit 4<br><br>This read/write bit enables the port A pin as an output.<br><br>0 Output Disabled for port A bit 4.<br>1 Output Enabled for port A bit 4. |
| 3<br>PTAOE3 | Output Enable for Port A Bit 3<br><br>This read/write bit enables the port A pin as an output.<br><br>0 Output Disabled for port A bit 3.<br>1 Output Enabled for port A bit 3. |
| 2<br>PTAOE2 | Output Enable for Port A Bit 2<br><br>This read/write bit enables the port A pin as an output.<br><br>0 Output Disabled for port A bit 2.<br>1 Output Enabled for port A bit 2. |
| 1<br>PTAOE1 | Output Enable for Port A Bit 1<br><br>This read/write bit enables the port A pin as an output. |

*Table continues on the next page...*

**PORT_PTAOE field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Output Disabled for port A bit 1.<br>1 Output Enabled for port A bit 1. |
| 0<br>PTAOE0 | Output Enable for Port A Bit 0<br><br>This read/write bit enables the port A pin as an output.<br><br>0 Output Disabled for port A bit 0.<br>1 Output Enabled for port A bit 0. |

## 7.6.7 Port B Output Enable Register (PORT_PTBOE)

Address: 0h base + 30B1h offset = 30B1h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | PTBOE7 | PTBOE6 | PTBOE5 | PTBOE4 | PTBOE3 | PTBOE2 | PTBOE1 | PTBOE0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTBOE field descriptions**

| Field | Description |
|---|---|
| 7<br>PTBOE7 | Output Enable for Port B Bit 7<br><br>This read/write bit enables the port B pin as an output.<br><br>0 Output Disabled for port B bit 7.<br>1 Output Enabled for port B bit 7. |
| 6<br>PTBOE6 | Output Enable for Port B Bit 6<br><br>This read/write bit enables the port B pin as an output.<br><br>0 Output Disabled for port B bit 6.<br>1 Output Enabled for port B bit 6. |
| 5<br>PTBOE5 | Output Enable for Port B Bit 5<br><br>This read/write bit enables the port B pin as an output.<br><br>0 Output Disabled for port B bit 5.<br>1 Output Enabled for port B bit 5. |
| 4<br>PTBOE4 | Output Enable for Port B Bit 4<br><br>This read/write bit enables the port B pin as an output.<br><br>0 Output Disabled for port B bit 4.<br>1 Output Enabled for port B bit 4. |
| 3<br>PTBOE3 | Output Enable for Port B Bit 3<br><br>This read/write bit enables the port B pin as an output. |

*Table continues on the next page...*

**PORT_PTBOE field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Output Disabled for port B bit 3.<br>1 Output Enabled for port B bit 3. |
| 2<br>PTBOE2 | Output Enable for Port B Bit 2<br><br>This read/write bit enables the port B pin as an output.<br><br>0 Output Disabled for port B bit 2.<br>1 Output Enabled for port B bit 2. |
| 1<br>PTBOE1 | Output Enable for Port B Bit 1<br><br>This read/write bit enables the port B pin as an output.<br><br>0 Output Disabled for port B bit 1.<br>1 Output Enabled for port B bit 1. |
| 0<br>PTBOE0 | Output Enable for Port B Bit 0<br><br>This read/write bit enables the port B pin as an output.<br><br>0 Output Disabled for port B bit 0.<br>1 Output Enabled for port B bit 0. |

## 7.6.8 Port C Output Enable Register (PORT_PTCOE)

Address: 0h base + 30B2h offset = 30B2h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | PTCOE7 | PTCOE6 | PTCOE5 | PTCOE4 | PTCOE3 | PTCOE2 | PTCOE1 | PTCOE0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTCOE field descriptions**

| Field | Description |
|---|---|
| 7<br>PTCOE7 | Output Enable for Port C Bit 7<br><br>This read/write bit enables the port C pin as an output.<br><br>0 Output Disabled for port C bit 7.<br>1 Output Enabled for port C bit 7. |
| 6<br>PTCOE6 | Output Enable for Port C Bit 6<br><br>This read/write bit enables the port C pin as an output.<br><br>0 Output Disabled for port C bit 6.<br>1 Output Enabled for port C bit 6. |
| 5<br>PTCOE5 | Output Enable for Port C Bit 5<br><br>This read/write bit enables the port C pin as an output. |

*Table continues on the next page...*

**PORT_PTCOE field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Output Disabled for port C bit 5.<br>1 Output Enabled for port C bit 5. |
| 4<br>PTCOE4 | Output Enable for Port C Bit 4<br><br>This read/write bit enables the port C pin as an output.<br><br>0 Output Disabled for port C bit 4.<br>1 Output Enabled for port C bit 4. |
| 3<br>PTCOE3 | Output Enable for Port C Bit 3<br><br>This read/write bit enables the port C pin as an output.<br><br>0 Output Disabled for port C bit 3.<br>1 Output Enabled for port C bit 3. |
| 2<br>PTCOE2 | Output Enable for Port C Bit 2<br><br>This read/write bit enables the port C pin as an output.<br><br>0 Output Disabled for port C bit 2.<br>1 Output Enabled for port C bit 2. |
| 1<br>PTCOE1 | Output Enable for Port C Bit 1<br><br>This read/write bit enables the port C pin as an output.<br><br>0 Output Disabled for port C bit 1.<br>1 Output Enabled for port C bit 1. |
| 0<br>PTCOE0 | Output Enable for Port C Bit 0<br><br>This read/write bit enables the port C pin as an output.<br><br>0 Output Disabled for port C bit 0.<br>1 Output Enabled for port C bit 0. |

## 7.6.9 Port D Output Enable Register (PORT_PTDOE)

Address: 0h base + 30B3h offset = 30B3h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | PTDOE7 | PTDOE6 | 0 | | | PTDOE2 | PTDOE1 | PTDOE0 |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTDOE field descriptions**

| Field | Description |
|---|---|
| 7<br>PTDOE7 | Output Enable for Port D Bit 7<br><br>This read/write bit enables the port D pin as an output. |

*Table continues on the next page...*

**PORT_PTDOE field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Output Disabled for port D bit 7.<br>1 Output Enabled for port D bit 7. |
| 6<br>PTDOE6 | Output Enable for Port D Bit 6<br>This read/write bit enables the port D pin as an output.<br>0 Output Disabled for port D bit 6.<br>1 Output Enabled for port D bit 6. |
| 5–3<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 2<br>PTDOE2 | Output Enable for Port D Bit 2<br>This read/write bit enables the port D pin as an output.<br>0 Output Disabled for port D bit 2.<br>1 Output Enabled for port D bit 2. |
| 1<br>PTDOE1 | Output Enable for Port D Bit 1<br>This read/write bit enables the port D pin as an output.<br>0 Output Disabled for port D bit 1.<br>1 Output Enabled for port D bit 1. |
| 0<br>PTDOE0 | Output Enable for Port D Bit 0<br>This read/write bit enables the port D pin as an output.<br>0 Output Disabled for port D bit 0.<br>1 Output Enabled for port D bit 0. |

## 7.6.10 Port E Output Enable Register (PORT_PTEOE)

Address: 0h base + 30B4h offset = 30B4h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | PTEOE4 | 0 | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTEOE field descriptions**

| Field | Description |
|---|---|
| 7–5<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 4<br>PTEOE4 | Output Enable for Port E Bit 4<br>This read/write bit enables the port E pin as an output. |

*Table continues on the next page...*

**PORT_PTEOE field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Output Disabled for port E bit 4.<br>1 Output Enabled for port E bit 4. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 7.6.11 Port A Input Enable Register (PORT_PTAIE)

Address: 0h base + 30B8h offset = 30B8h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | PTAIE7 | PTAIE6 | PTAIE5 | 0 | PTAIE3 | PTAIE2 | PTAIE1 | PTAIE0 |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTAIE field descriptions**

| Field | Description |
|---|---|
| 7<br>PTAIE7 | Input Enable for Port A Bit 7<br><br>This read/write bit enables the port A pin as an input.<br><br>0 Input disabled for port A bit 7.<br>1 Input enabled for port A bit 7. |
| 6<br>PTAIE6 | Input Enable for Port A Bit 6<br><br>This read/write bit enables the port A pin as an input.<br><br>0 Input disabled for port A bit 6.<br>1 Input enabled for port A bit 6. |
| 5<br>PTAIE5 | Input Enable for Port A Bit 5<br><br>This read/write bit enables the port A pin as an input.<br><br>0 Input disabled for port A bit 5.<br>1 Input enabled for port A bit 5. |
| 4<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 3<br>PTAIE3 | Input Enable for Port A Bit 3<br><br>This read/write bit enables the port A pin as an input.<br><br>0 Input disabled for port A bit 3.<br>1 Input enabled for port A bit 3. |
| 2<br>PTAIE2 | Input Enable for Port A Bit 2<br><br>This read/write bit enables the port A pin as an input. |

*Table continues on the next page...*

**PORT_PTAIE field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Input disabled for port A bit 2.<br>1 Input enabled for port A bit 2. |
| 1<br>PTAIE1 | Input Enable for Port A Bit 1<br><br>This read/write bit enables the port A pin as an input.<br><br>0 Input disabled for port A bit 1.<br>1 Input enabled for port A bit 1. |
| 0<br>PTAIE0 | Input Enable for Port A Bit 0<br><br>This read/write bit enables the port A pin as an input.<br><br>0 Input disabled for port A bit 0.<br>1 Input enabled for port A bit 0. |

## 7.6.12 Port B Input Enable Register (PORT_PTBIE)

Address: 0h base + 30B9h offset = 30B9h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | PTBIE7 | PTBIE6 | PTBIE5 | PTBIE4 | PTBIE3 | PTBIE2 | PTBIE1 | PTBIE0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTBIE field descriptions**

| Field | Description |
|---|---|
| 7<br>PTBIE7 | Input Enable for Port B Bit 7<br><br>This read/write bit enables the port B pin as an input.<br><br>0 Input disabled for port B bit 7.<br>1 Input enabled for port B bit 7. |
| 6<br>PTBIE6 | Input Enable for Port B Bit 6<br><br>This read/write bit enables the port B pin as an input.<br><br>0 Input disabled for port B bit 6.<br>1 Input enabled for port B bit 6. |
| 5<br>PTBIE5 | Input Enable for Port B Bit 5<br><br>This read/write bit enables the port B pin as an input.<br><br>0 Input disabled for port B bit 5.<br>1 Input enabled for port B bit 5. |
| 4<br>PTBIE4 | Input Enable for Port B Bit 4<br><br>This read/write bit enables the port B pin as an input. |

*Table continues on the next page...*

**PORT_PTBIE field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Input disabled for port B bit 4.<br>1 Input enabled for port B bit 4. |
| 3<br>PTBIE3 | Input Enable for Port B Bit 3<br><br>This read/write bit enables the port B pin as an input.<br><br>0 Input disabled for port B bit 3.<br>1 Input enabled for port B bit 3. |
| 2<br>PTBIE2 | Input Enable for Port B Bit 2<br><br>This read/write bit enables the port B pin as an input.<br><br>0 Input disabled for port B bit 2.<br>1 Input enabled for port B bit 2. |
| 1<br>PTBIE1 | Input Enable for Port B Bit 1<br><br>This read/write bit enables the port B pin as an input.<br><br>0 Input disabled for port B bit 1.<br>1 Input enabled for port B bit 1. |
| 0<br>PTBIE0 | Input Enable for Port B Bit 0<br><br>This read/write bit enables the port B pin as an input.<br><br>0 Input disabled for port B bit 0.<br>1 Input enabled for port B bit 0. |

## 7.6.13 Port C Input Enable Register (PORT_PTCIE)

Address: 0h base + 30BAh offset = 30BAh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | PTCIE7 | PTCIE6 | PTCIE5 | PTCIE4 | PTCIE3 | PTCIE2 | PTCIE1 | PTCIE0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTCIE field descriptions**

| Field | Description |
|---|---|
| 7<br>PTCIE7 | Input Enable for Port C Bit 7<br><br>This read/write bit enables the port C pin as an input.<br><br>0 Input disabled for port C bit 7.<br>1 Input enabled for port C bit 7. |
| 6<br>PTCIE6 | Input Enable for Port C Bit 6<br><br>This read/write bit enables the port C pin as an input. |

*Table continues on the next page...*

**PORT_PTCIE field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Input disabled for port C bit 6.<br>1 Input enabled for port C bit 6. |
| 5<br>PTCIE5 | Input Enable for Port C Bit 5<br>This read/write bit enables the port C pin as an input.<br>0 Input disabled for port C bit 5.<br>1 Input enabled for port C bit 5. |
| 4<br>PTCIE4 | Input Enable for Port C Bit 4<br>This read/write bit enables the port C pin as an input.<br>0 Input disabled for port C bit 4.<br>1 Input enabled for port C bit 4. |
| 3<br>PTCIE3 | Input Enable for Port C Bit 3<br>This read/write bit enables the port C pin as an input.<br>0 Input disabled for port C bit 3.<br>1 Input enabled for port C bit 3. |
| 2<br>PTCIE2 | Input Enable for Port C Bit 2<br>This read/write bit enables the port C pin as an input.<br>0 Input disabled for port C bit 2.<br>1 Input enabled for port C bit 2. |
| 1<br>PTCIE1 | Input Enable for Port C Bit 1<br>This read/write bit enables the port C pin as an input.<br>0 Input disabled for port C bit 1.<br>1 Input enabled for port C bit 1. |
| 0<br>PTCIE0 | Input Enable for Port C Bit 0<br>This read/write bit enables the port C pin as an input.<br>0 Input disabled for port C bit 0.<br>1 Input enabled for port C bit 0. |

## 7.6.14 Port D Input Enable Register (PORT_PTDIE)

Address: 0h base + 30BBh offset = 30BBh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | PTDIE7 | PTDIE6 | 0 | | | PTDIE2 | PTDIE1 | PTDIE0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTDIE field descriptions**

| Field | Description |
|---|---|
| 7<br>PTDIE7 | Input Enable for Port D Bit 7<br><br>This read/write bit enables the port D pin as an input.<br><br>0 Input disabled for port D bit 7.<br>1 Input enabled for port D bit 7. |
| 6<br>PTDIE6 | Input Enable for Port D Bit 6<br><br>This read/write bit enables the port D pin as an input.<br><br>0 Input disabled for port D bit 6.<br>1 Input enabled for port D bit 6. |
| 5–3<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 2<br>PTDIE2 | Input Enable for Port D Bit 2<br><br>This read/write bit enables the port D pin as an input.<br><br>0 Input disabled for port D bit 2.<br>1 Input enabled for port D bit 2. |
| 1<br>PTDIE1 | Input Enable for Port D Bit 1<br><br>This read/write bit enables the port D pin as an input.<br><br>0 Input disabled for port D bit 1.<br>1 Input enabled for port D bit 1. |
| 0<br>PTDIE0 | Input Enable for Port D Bit 0<br><br>This read/write bit enables the port D pin as an input.<br><br>0 Input disabled for port D bit 0.<br>1 Input enabled for port D bit 0. |

## 7.6.15 Port E Input Enable Register (PORT_PTEIE)

Address: 0h base + 30BCh offset = 30BCh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | PTEIE4 | 0 | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTEIE field descriptions**

| Field | Description |
|---|---|
| 7–5<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

*Table continues on the next page...*

**PORT_PTEIE field descriptions (continued)**

| Field | Description |
|---|---|
| 4<br>PTEIE4 | Input Enable for Port E Bit 4<br><br>This read/write bit enables the port E pin as an input.<br><br>0 Input disabled for port E bit 4.<br>1 Input enabled for port E bit 4. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 7.6.16 Port Filter Register 0 (PORT_IOFLT0)

This register sets the filters for input from PTA to PTD.

Address: 0h base + 30ECh offset = 30ECh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | FLTD | | FLTC | | FLTB | | FLTA | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_IOFLT0 field descriptions**

| Field | Description |
|---|---|
| 7–6<br>FLTD | Filter selection for input from PTD<br><br>00 BUSCLK<br>01 FLTDIV1<br>10 FLTDIV2<br>11 FLTDIV3 |
| 5–4<br>FLTC | Filter selection for input from PTC<br><br>00 BUSCLK<br>01 FLTDIV1<br>10 FLTDIV2<br>11 FLTDIV3 |
| 3–2<br>FLTB | Filter selection for input from PTB<br><br>00 BUSCLK<br>01 FLTDIV1<br>10 FLTDIV2<br>11 FLTDIV3 |
| FLTA | Filter selection for input from PTA<br><br>00 BUSCLK<br>01 FLTDIV1<br>10 FLTDIV2<br>11 FLTDIV3 |

## 7.6.17 Port Filter Register 1 (PORT_IOFLT1)

This register sets the filters for input from PTE.

Address: 0h base + 30EDh offset = 30EDh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | | | | FLTE | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_IOFLT1 field descriptions**

| Field | Description |
|---|---|
| 7–2<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| FLTE | Filter selection for input from PTE<br><br>00 BUSCLK<br>01 FLTDIV1<br>10 FLTDIV2<br>11 FLTDIV3 |

## 7.6.18 Port Filter Register 2 (PORT_IOFLT2)

This register sets the filters for input.

Address: 0h base + 30EEh offset = 30EEh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | | FLTKBI0 | | FLTRST | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_IOFLT2 field descriptions**

| Field | Description |
|---|---|
| 7–4<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 3–2<br>FLTKBI0 | Filter selection for input from KBI0<br><br>00 BUSCLK<br>01 Select FLTDIV1, and will switch to FLTDIV3 in stop mode automatically.<br>10 Select FLTDIV2, and will switch to FLTDIV3 in stop mode automatically.<br>11 FLTDIV3 |
| FLTRST | Filter selection for input from RESET/IRQ |

*Table continues on the next page...*

**PORT_IOFLT2 field descriptions (continued)**

| Field | Description |
|---|---|
| | 00 No filter.<br>01 Select FLTDIV1, and will switch to FLTDIV3 in stop mode automatically.<br>10 Select FLTDIV2, and will switch to FLTDIV3 in stop mode automatically.<br>11 FLTDIV3 |

## 7.6.19 Port Clock Division Register (PORT_FCLKDIV)

Configure the high/low level glitch width threshold. Glitches that are shorter than the selected clock width will be filtered out; glitches that are more than twice the selected clock width will not be filtered out (they will pass to the internal circuitry).

Address: 0h base + 30EFh offset = 30EFh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | FLTDIV3 | | | FLTDIV2 | | | FLTDIV1 | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_FCLKDIV field descriptions**

| Field | Description |
|---|---|
| 7–5<br>FLTDIV3 | Filter Division Set 3<br><br>Port Filter Division Set 3<br><br>000 LPOCLK.<br>001 LPOCLK/2.<br>010 LPOCLK/4.<br>011 LPOCLK/8.<br>100 LPOCLK/16.<br>101 LPOCLK/32.<br>110 LPOCLK/64.<br>111 LPOCLK/128. |
| 4–2<br>FLTDIV2 | Filter Division Set 2<br><br>Port Filter Division Set 2<br><br>000 BUSCLK/32.<br>001 BUSCLK/64.<br>010 BUSCLK/128.<br>011 BUSCLK/256.<br>100 BUSCLK/512.<br>101 BUSCLK/1024.<br>110 BUSCLK/2048.<br>111 BUSCLK/4096. |
| FLTDIV1 | Filter Division Set 1 |

*Table continues on the next page...*

**PORT_FCLKDIV field descriptions (continued)**

| Field | Description |
|---|---|
| | Port Filter Division Set 1<br><br>00 BUSCLK/2.<br>01 BUSCLK/4.<br>10 BUSCLK/8.<br>11 BUSCLK/16. |

## 7.6.20 Port A Pullup Enable Register (PORT_PTAPE)

Address: 0h base + 30F0h offset = 30F0h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | PTAPE7 | PTAPE6 | PTAPE5 | 0 | PTAPE3 | PTAPE2 | PTAPE1 | PTAPE0 |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTAPE field descriptions**

| Field | Description |
|---|---|
| 7<br>PTAPE7 | Pull Enable for Port A Bit 7<br><br>This control bit determines if the internal pullup device is enabled for the associated PTA pin. For port A pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port A bit 7.<br>1 Pullup enabled for port A bit 7. |
| 6<br>PTAPE6 | Pull Enable for Port A Bit 6<br><br>This control bit determines if the internal pullup device is enabled for the associated PTA pin. For port A pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port A bit 6.<br>1 Pullup enabled for port A bit 6. |
| 5<br>PTAPE5 | Pull Enable for Port A Bit 5<br><br>This control bit determines if the internal pullup device is enabled for the associated PTA pin. For port A pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port A bit 5.<br>1 Pullup enabled for port A bit 5. |
| 4<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 3<br>PTAPE3 | Pull Enable for Port A Bit 3<br><br>This control bit determines if the internal pullup device is enabled for the associated PTA pin. For port A pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port A bit 3.<br>1 Pullup enabled for port A bit 3. |

*Table continues on the next page...*

**PORT_PTAPE field descriptions (continued)**

| Field | Description |
|---|---|
| 2<br>PTAPE2 | Pull Enable for Port A Bit 2<br><br>This control bit determines if the internal pullup device is enabled for the associated PTA pin. For port A pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port A bit 2.<br>1 Pullup enabled for port A bit 2. |
| 1<br>PTAPE1 | Pull Enable for Port A Bit 1<br><br>This control bit determines if the internal pullup device is enabled for the associated PTA pin. For port A pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port A bit 1.<br>1 Pullup enabled for port A bit 1. |
| 0<br>PTAPE0 | Pull Enable for Port A Bit 0<br><br>This control bit determines if the internal pullup device is enabled for the associated PTA pin. For port A pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port A bit 0.<br>1 Pullup enabled for port A bit 0. |

## 7.6.21 Port B Pullup Enable Register (PORT_PTBPE)

Address: 0h base + 30F1h offset = 30F1h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | PTBPE7 | PTBPE6 | PTBPE5 | PTBPE4 | PTBPE3 | PTBPE2 | PTBPE1 | PTBPE0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTBPE field descriptions**

| Field | Description |
|---|---|
| 7<br>PTBPE7 | Pull Enable for Port B Bit 7<br><br>This control bit determines if the internal pullup device is enabled for the associated PTB pin. For port B pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port B bit 7.<br>1 Pullup enabled for port B bit 7. |
| 6<br>PTBPE6 | Pull Enable for Port B Bit 6<br><br>This control bit determines if the internal pullup device is enabled for the associated PTB pin. For port B pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port B bit 6.<br>1 Pullup enabled for port B bit 6. |
| 5<br>PTBPE5 | Pull Enable for Port B Bit 5 |

*Table continues on the next page...*

**PORT_PTBPE field descriptions (continued)**

| Field | Description |
|---|---|
| | This control bit determines if the internal pullup device is enabled for the associated PTB pin. For port B pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port B bit 5.<br>1 Pullup enabled for port B bit 5. |
| 4<br>PTBPE4 | Pull Enable for Port B Bit 4<br><br>This control bit determines if the internal pullup device is enabled for the associated PTB pin. For port B pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port B bit 4.<br>1 Pullup enabled for port B bit 4. |
| 3<br>PTBPE3 | Pull Enable for Port B Bit 3<br><br>This control bit determines if the internal pullup device is enabled for the associated PTB pin. For port B pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port B bit 3.<br>1 Pullup enabled for port B bit 3. |
| 2<br>PTBPE2 | Pull Enable for Port B Bit 2<br><br>This control bit determines if the internal pullup device is enabled for the associated PTB pin. For port B pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port B bit 2.<br>1 Pullup enabled for port B bit 2. |
| 1<br>PTBPE1 | Pull Enable for Port B Bit 1<br><br>This control bit determines if the internal pullup device is enabled for the associated PTB pin. For port B pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port B bit 1.<br>1 Pullup enabled for port B bit 1. |
| 0<br>PTBPE0 | Pull Enable for Port B Bit 0<br><br>This control bit determines if the internal pullup device is enabled for the associated PTB pin. For port B pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port B bit 0.<br>1 Pullup enabled for port B bit 0. |

## 7.6.22 Port C Pullup Enable Register (PORT_PTCPE)

Address: 0h base + 30F2h offset = 30F2h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | PTCPE7 | PTCPE6 | PTCPE5 | PTCPE4 | PTCPE3 | PTCPE2 | PTCPE1 | PTCPE0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## PORT_PTCPE field descriptions

| Field | Description |
|---|---|
| 7<br>PTCPE7 | Pull Enable for Port C Bit 7<br><br>This control bit determines if the internal pullup device is enabled for the associated PTC pin. For port C pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port C bit 7.<br>1 Pullup enabled for port C bit 7. |
| 6<br>PTCPE6 | Pull Enable for Port C Bit 6<br><br>This control bit determines if the internal pullup device is enabled for the associated PTC pin. For port C pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port C bit 6.<br>1 Pullup enabled for port C bit 6. |
| 5<br>PTCPE5 | Pull Enable for Port C Bit 5<br><br>This control bit determines if the internal pullup device is enabled for the associated PTC pin. For port C pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port C bit 5.<br>1 Pullup enabled for port C bit 5. |
| 4<br>PTCPE4 | Pull Enable for Port C Bit 4<br><br>This control bit determines if the internal pullup device is enabled for the associated PTC pin. For port C pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port C bit 4.<br>1 Pullup enabled for port C bit 4. |
| 3<br>PTCPE3 | Pull Enable for Port C Bit 3<br><br>This control bit determines if the internal pullup device is enabled for the associated PTC pin. For port C pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port C bit 3.<br>1 Pullup enabled for port C bit 3. |
| 2<br>PTCPE2 | Pull Enable for Port C Bit 2<br><br>This control bit determines if the internal pullup device is enabled for the associated PTC pin. For port C pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port C bit 2.<br>1 Pullup enabled for port C bit 2. |
| 1<br>PTCPE1 | Pull Enable for Port C Bit 1<br><br>This control bit determines if the internal pullup device is enabled for the associated PTC pin. For port C pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port C bit 1.<br>1 Pullup enabled for port C bit 1. |
| 0<br>PTCPE0 | Pull Enable for Port C Bit 0 |

*Table continues on the next page...*

**PORT_PTCPE field descriptions (continued)**

| Field | Description |
|---|---|
| | This control bit determines if the internal pullup device is enabled for the associated PTC pin. For port C pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port C bit 0.<br>1 Pullup enabled for port C bit 0. |

## 7.6.23 Port D Pullup Enable Register (PORT_PTDPE)

Address: 0h base + 30F3h offset = 30F3h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | PTDPE7 | PTDPE6 | 0 | | | PTDPE2 | PTDPE1 | PTDPE0 |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTDPE field descriptions**

| Field | Description |
|---|---|
| 7<br>PTDPE7 | Pull Enable for Port D Bit 7<br><br>This control bit determines if the internal pullup device is enabled for the associated PTD pin. For port D pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port D bit 7.<br>1 Pullup enabled for port D bit 7. |
| 6<br>PTDPE6 | Pull Enable for Port D Bit 6<br><br>This control bit determines if the internal pullup device is enabled for the associated PTD pin. For port D pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port D bit 6.<br>1 Pullup enabled for port D bit 6. |
| 5–3<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 2<br>PTDPE2 | Pull Enable for Port D Bit 2<br><br>This control bit determines if the internal pullup device is enabled for the associated PTD pin. For port D pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port D bit 2.<br>1 Pullup enabled for port D bit 2. |
| 1<br>PTDPE1 | Pull Enable for Port D Bit 1<br><br>This control bit determines if the internal pullup device is enabled for the associated PTD pin. For port D pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port D bit 1.<br>1 Pullup enabled for port D bit 1. |

*Table continues on the next page...*

**PORT_PTDPE field descriptions (continued)**

| Field | Description |
|---|---|
| 0<br>PTDPE0 | Pull Enable for Port D Bit 0<br><br>This control bit determines if the internal pullup device is enabled for the associated PTD pin. For port D pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port D bit 0.<br>1 Pullup enabled for port D bit 0. |

## 7.6.24 Port E Pullup Enable Register (PORT_PTEPE)

Address: 0h base + 30F4h offset = 30F4h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | PTEPE4 | 0 | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORT_PTEPE field descriptions**

| Field | Description |
|---|---|
| 7–5<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 4<br>PTEPE4 | Pull Enable for Port E Bit 4<br><br>This control bit determines if the internal pullup device is enabled for the associated PTE pin. For port E pins that are configured as outputs or Hi-Z, these bits have no effect.<br><br>0 Pullup disabled for port E bit 4.<br>1 Pullup enabled for port E bit 4. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

# Chapter 8
# Clock management

## 8.1 Clock module

This device has ICS, XOSC, and LPO clock modules.

The internal clock source (ICS) module provides several clock source options for this device. The module contains a frequency-locked loop (FLL) that is controllable by either an internal or external reference clock. The module can select clock from the FLL or bypass the FLL as a source of the MCU system clock. The selected clock source is passed through a reduced bus divider, which allows a lower output clock frequency to be derived.

The external oscillator (XOSC) module allows an external crystal, ceramic resonator, or other external clock source to produce the external reference clock. The output of XOSC module can be used as the reference of ICS to generate system bus clock, and/or clock source of watchdog (WDOG), real-time counter (RTC), and analog-to-digital (ADC) modules.

The low-power oscillator (LPO) module is an on-chip low-power oscillator providing 1 kHz reference clock to RTC and watchdog (WDOG).

## 8.2 Internal clock source (ICS)

The internal clock source (ICS) module provides clock source options for the MCU. The module contains a frequency-locked loop (FLL) as a clock source that is controllable by an internal or external reference clock. The module can provide this FLL clock or the internal reference clock as a source for the MCU system clock, ICSCLK.

Whichever clock source is chosen, ICSCLK is the output from a bus clock divider (BDIV), which allows a lower clock frequency to be derived.

Key features of the ICS module are:

- Frequency-locked loop (FLL) is trimmable for accuracy
- Internal or external reference clocks can be used to control the FLL
- Reference divider is provided for external clock
- Internal reference clock has nine trim bits available
- Internal or external reference clocks can be selected as the clock source for the MCU
- Whichever clock is selected as the source can be divided down by 1, 2, 4, 8, 16, 32, 64 or 128
- FLL Engaged Internal mode is automatically selected out of reset
- A constant divide by 2 of the DCO output that can be select as BDC clock.
- Digitally-controlled oscillator (DCO) optimized for 16 MHz to 20 MHz frequency range
- FLL lock detector and external clock monitor
  - FLL lock detector with interrupt capability
  - External reference clock monitor with reset capability

### 8.2.1 Function description

The following figure shows the ICS block diagram.

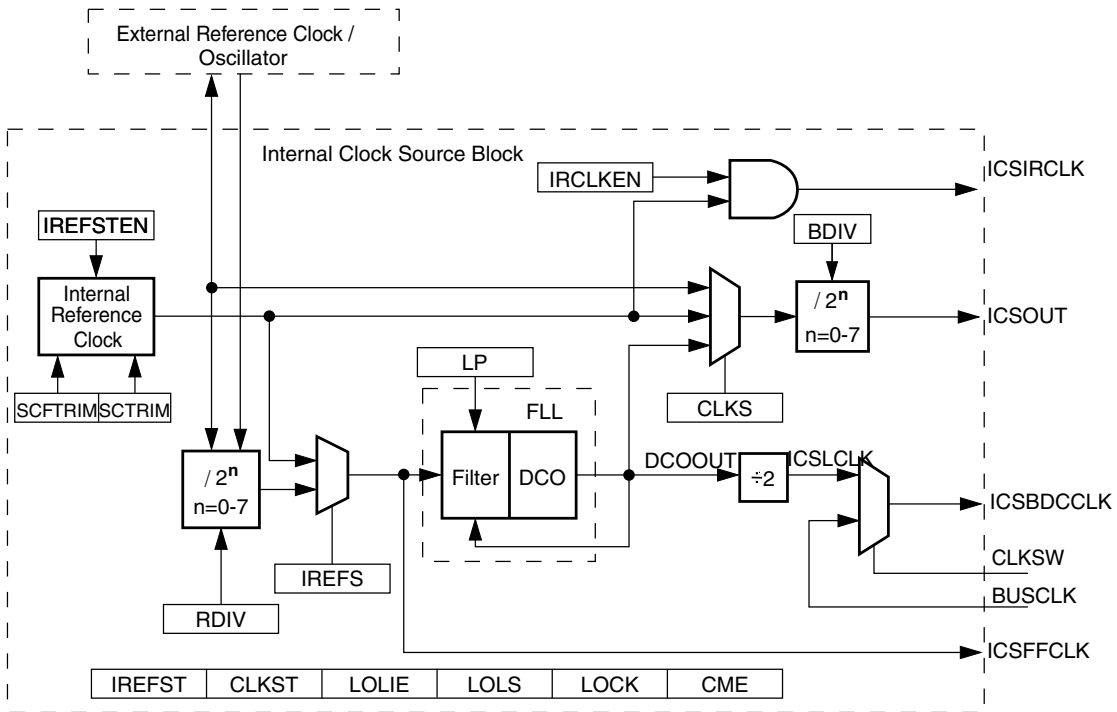



**Figure 8-1. Internal clock source (ICS)**

### 8.2.1.1 Bus frequency divider

The ICS_C2[BDIV] bits can be changed at anytime and the actual switch to the new frequency occurs immediately.

### 8.2.1.2 Low power bit usage

The low power bit (ICS_C2[LP]) is provided to allow the FLL to be disabled and thus conserve power when it is not used.

However, in some applications it may be desirable to allow the FLL to be enabled and to lock for maximum accuracy before switching to an FLL engaged mode. To do this, write the ICS_C2[LP] bit to 0.

### 8.2.1.3 Internal reference clock (ICSIRCLK)

When ICS_C1[IRCLKEN] is set the internal reference clock signal is presented as ICSIRCLK, which can be used as an additional clock source. To re-target the ICSIRCLK frequency, write a new value to the ICS_C3[SCTRIM] and ICS_C4[SCFTRIM] bits to trim the period of the internal reference clock:

- Writing a larger value slows down the ICSIRCLK frequency.
- Writing a smaller value to the ICSTRM register speeds up the ICSIRCLK frequency.

The TRIM bits affect the ICSOUT frequency if the ICS is in FLL engaged internal (FEI), FLL bypassed internal (FBI), or FLL bypassed internal low power (FBILP) mode.

Until ICSIRCLK is trimmed, programming low reference divider (BDIV) factors may result in ICSOUT frequencies that exceed the maximum chip-level frequency and violate the chip-level clock timing specifications.

If ICS_C1[IREFSTEN] is set and the ICS_C1[IRCLKEN] bit is written to 1, the internal reference clock keeps running during stop mode in order to provide a fast recovery upon exiting stop.

All MCU devices are factory programmed with a trim value in a factory reserved memory location (i.e. reserved nonvolatile information registers that can not be accessed by users). This value is uploaded to the ICS_C3 register and ICS_C4 register during any reset initialization. For finer precision, trim the internal oscillator in the application and set the ICS_C4[SCFTRIM] bit accordingly.

**NOTE**

Some tools like ProcessorExpert or USB Multilink may use flash memory location, such as 0xFF6F and/or 0xFF6E, to store the temporary trim value.

#### 8.2.1.4 Fixed frequency clock (ICSFFCLK)

The ICS presents the divided FLL reference clock as ICSFFCLK for use as an additional clock source. ICSFFCLK frequency must be no more than 1/4 of the ICSOUT frequency to be valid. When ICSFFCLK is valid, ICS output signal (ICSFFE) gets asserted high. Because of this requirement, in bypass modes the ICSFFCLK is valid only in bypass external modes (FBE and FBELP) for the following combinations of BDIV, RDIV, and RANGE values:

- RANGE=1
- BDIV=000 (divide by 1), RDIV ≥ 010
- BDIV=001 (divide by 2), RDIV ≥ 011
- BDIV=010 (divide by 4), RDIV ≥ 100
- BDIV=011 (divide by 8), RDIV ≥ 101

- BDIV=100 (divide by 16), NA
- BDIV=101 (divide by 32), NA
- BDIV=110 (divide by 64), NA
- BDIV=111 (divide by 128), NA

#### 8.2.1.5 BDC clock

The ICS presents the DCO output clock divided by two as ICSLCLK for use as a clock source for BDC communications. ICSLCLK is not available in FLL bypassed internal low power (FBILP) and FLL bypassed external low power (FBELP) modes. The ICSLCLK can be selected as BDC clock.

### 8.2.2 Modes of operation

There are seven modes of operation for the ICS: FEI, FEE, FBI, FBILP, FBE, FBELP, and stop. The following figure shows the seven states of the ICS as a state diagram. The arrows indicate the allowed movements between the states.

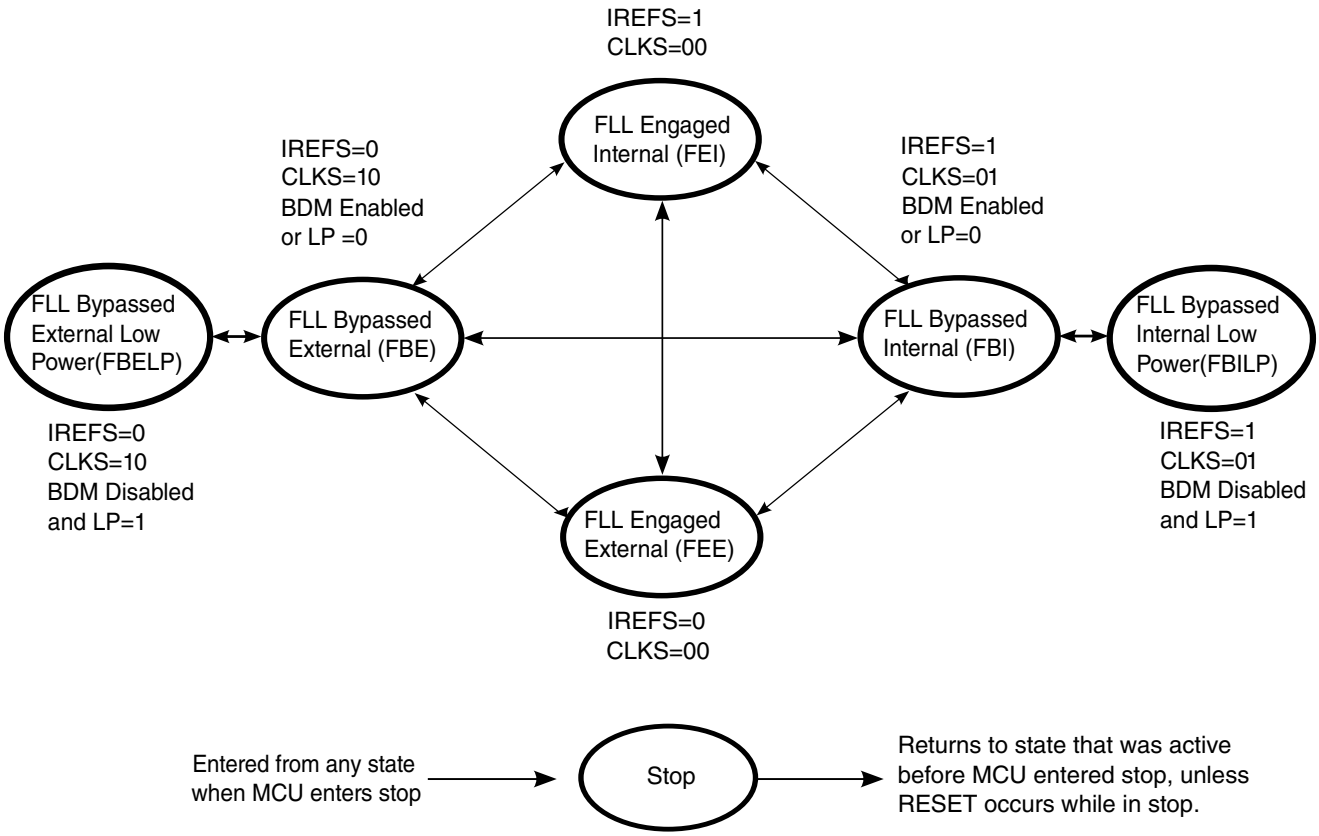



**Figure 8-2. ICS clocking switching modes**

The ICS_C1[IREFS] bit can be changed at anytime, but the actual switch to the newly selected clock is shown by the ICS_S[IREFST] bit. When switching between FLL engaged internal (FEI) and FLL engaged external (FEE) modes, the FLL will lock again after the switch is completed.

The ICS_C1[CLKS] bits can also be changed at anytime, but the actual switch to the newly selected clock is shown by the ICS_S[CLKST] bits. If the newly selected clock is not available, the previous clock remains selected.

### 8.2.2.1 FLL engaged internal (FEI)

FLL engaged internal (FEI) is the default mode of operation and is entered when all of the following conditions occur:

- ICS_C1[CLKS] bits are written to 0b
- ICS_C1[IREFS] bit is written to 1b

In FLL engaged internal mode, the ICSOUT clock is derived from the FLL clock, which is controlled by the internal reference clock. The FLL loop locks the frequency to the 512 times the internal reference frequency. The ICSLCLK is available for BDC communications, and the internal reference clock is enabled.

#### 8.2.2.2 FLL engaged external (FEE)

The FLL engaged external (FEE) mode is entered when all of the following conditions occur:

- ICS_C1[CLKS] bits are written to 00b
- ICS_C1[IREFS] bit written to 0b
- ICS_C1[RDIV] bits are written to divide external reference clock to be within the range of 31.25 kHz to 39.0625 kHz

In FLL engaged external mode, the ICSOUT clock is derived from the FLL clock, which is controlled by the external reference clock source. The FLL loop locks the frequency to the 512 times the external reference frequency, as selected by the ICS_C1[RDIV] bits. The ICSLCLK is available for BDC communications, and the external reference clock is enabled.

#### 8.2.2.3 FLL bypassed internal (FBI)

The FLL bypassed internal (FBI) mode is entered when all of the following conditions occur:

- ICS_C1[CLKS] bits are written to 01
- ICS_C1[IREFS] bit is written to 1
- BDM mode is active or ICS_C2[LP] bit is written to 0

In FLL bypassed internal mode, the ICSOUT clock is derived from the internal reference clock. The FLL clock is controlled by the internal reference clock, and the FLL loop locks the FLL frequency to the 512 times the internal reference frequency. The ICSLCLK will be available for BDC communications, and the internal reference clock is enabled.

### 8.2.2.4 FLL bypassed internal low power (FBILP)

The FLL bypassed internal low power (FBILP) mode is entered when all the following conditions occur:

- ICS_C1[CLKS] bits are written to 01
- ICS_C1[IREFS] bit is written to 1
- BDM mode is not active and ICS_C2[LP] bit is written to 1

In FLL bypassed internal low power mode, the ICSOUT clock is derived from the internal reference clock and the FLL is disabled. The ICSLCLK will be not be available for BDC communications, and the internal reference clock is enabled.

### 8.2.2.5 FLL bypassed external (FBE)

The FLL bypassed external (FBE) mode is entered when all of the following conditions occur:

- ICS_C1[CLKS] bits are written to 10
- ICS_C1[IREFS] bit is written to 0
- ICS_C1[RDIV] bits are written to divide external reference clock to be within the range of 31.25 kHz to 39.0625 kHz
- BDM mode is active or ICS_C2[LP] bit is written to 0

In FLL bypassed external mode, the ICSOUT clock is derived from the external reference clock source. The FLL clock is controlled by the external reference clock, and the FLL loop locks the FLL frequency to the 512 times the external reference frequency, as selected by the ICS_C1[RDIV] bits, so that the ICSLCLK will be available for BDC communications, and the external reference clock is enabled.

### 8.2.2.6 FLL bypassed external low power (FBELP)

The FLL bypassed external low power (FBELP) mode is entered when all of the following conditions occur:

- ICS_C1[CLKS] bits are written to 10

- ICS_C1[IREFS] bit is written to 0
- BDM mode is not active and ICS_C2[LP] bit is written to 1

In FLL bypassed external low power mode, the ICSOUT clock is derived from the external reference clock source and the FLL is disabled. The ICSLCLK will be not available for BDC communications. The external reference clock source is enabled.

### 8.2.2.7 Stop (STOP)

In stop mode, the FLL is disabled and the internal clock source can be enabled or disabled. The BDC clock is not available and the ICS does not provide MCU clock source.

Stop mode is entered whenever the MCU enters a stop state. In this mode, all ICS clock signals are static except in the following cases:

- ICSIRCLK will be active in stop mode when all of the following conditions occur:
  - ICS_C1[IRCLKEN] bit is written to 1
  - ICS_C1[IREFSTEN] bit is written to 1
- OSCOUT will be active in stop mode when all of the following conditions occur:
  - ICS_OSCSC[OSCEN] bit is written to 1
  - ICS_OSCSC[OSCSTEN] bit is written to 1

**NOTE**

The DCO frequency changes from the pre-stop value to its reset value and the FLL need to re-acquire the lock before the frequency is stable. Timing sensitive operations must wait for the FLL acquisition time, $t_{Aquire}$, before executing.

## 8.2.3 FLL lock and clock monitor

### 8.2.3.1 FLL clock lock

In FBE and FEE modes, the clock detector source uses the external reference as the reference. When FLL is detected from lock to unlock, the ICS_S[LOLS] bit is set. An interrupt will be generated if ICS_C4[LOLIE] bit is set. ICS_S[LOLS] bit is cleared by reset or by writing a logic 1 to ICS_S[LOLS] when ICS_S[LOLS] is set. Writing a logic 0 to ICS_S[LOLS] has no effect.

In FBI and FEI modes, the lock detector source uses the internal reference as the reference. When FLL is detected from lock to unlock, the ICS_S[LOLS] bit is set. An interrupt will be generated if ICS_S[LOLS] bit is set. ICS_S[LOLS] bit is cleared by reset or by writing a logic 1 to ICS_S[LOLS] when ICS_S[LOLS] is set. Writing a logic 0 to ICS_S[LOLS] has no effect.

In FBELP and FBILP modes, the FLL is not on, therefore, lock detect function is not applicable.

### 8.2.3.2 External reference clock monitor

In FBE, FEE, FEI, or FBI modes, if ICS_C4[CME] bit is written to 1, the clock monitor is enabled. If the external reference falls below a certain frequency, such as $f_{loc_high}$ or $f_{loc_low}$ depending on the ICS_OSCSC[RANGE] bit, the MCU will reset. The SYS_SRS[CLK] bit will be set to indicate the error.

In FBELP or FBILP modes, the FLL is not on, so the external reference clock monitor will not function even if ICS_C4[CME] bit is written to 1.

External reference clock monitor uses FLL as the internal reference clock. The FLL must be functional before ICS_C4[CME] bit is set.

## 8.3 Initialization / application information

This section provides example code to give some basic direction to a user on how to initialize and configure the ICS module. The example software is implemented in C language.

## 8.3.1 Initializing FEI mode

The following code segment demonstrates setting ICS to FEI mode.

### Example: 8.3.1.1 FEI mode initialization routine

```
/* the following code segment demonstrates setting ICS to FEI mode generating 20MHz bus*/
ICS_C2 = 0x00;
ICS_C1 = 0x04; /* internal reference clock to FLL */
ICS_C2 = 0x00; /* BDIV = 0, no prescalar */
ICS_C3 = TRIM_VALUE_39K0625HZ; /* FLL output 20MHz, TRIM_VALUE_39K0625HZ is ~0x50  typically
*/
/* the following code segment demonstrates setting ICS to FEI mode generating 5MHz bus*/
ICS_C2 = 0x00;
ICS_C1 = 0x04; /* internal reference clock to FLL */
ICS_C2 = 0x40; /* BDIV = 2, prescalar = 4 */
ICS_C3 = TRIM_VALUE_39K0625HZ; /* FLL output 20MHz, TRIM_VALUE_39K0625HZ is ~0x50 typically
*/
/* the following code segment demonstrates setting ICS to FEI mode generating 500kHz bus*/
ICS_C2 = 0x00;
ICS_C1 = 0x04; /* internal reference clock to FLL */
ICS_C2 = 0xA0; /* BDIV = 5, prescalar = 32 */
ICS_C3 = TRIM_VALUE_31K25HZ; /* FLL output 16MHz, TRIM_VALUE_31K25HZ is ~0x90 typically */
```

## 8.3.2 Initializing FBI mode

The following code segment demonstrates setting ICS to FBI mode.

### Example: 8.3.2.1 FBI mode initialization routine

```
/* the following code segment demonstrates setting ICS to FBI mode generating 32768Hz bus*/
ICS_C2 = 0x00;
ICS_C1 = 0x40;
ICS_C2 = 0x00;
ICS_C3 = TRIM_VALUE_32K768HZ; /* TRIM_VALUE_31K25HZ is ~0x90 typically */
```

## 8.3.3 Initializing FEE mode

The following code segment demonstrates setting ICS to FEE mode.

### Example: 8.3.3.1 FEE mode initialization routine

```
/* the following code segment demonstrates setting ICS to FEE mode generating 8MHZ bus*/
/* supposing external 4MHZ crystal is installed in high gain mode */
ICS_OSCSC = 0x96; /* high-range, high-gain, oscillator required */
while (ICS_OSCINIT == 0); /* waiting until oscillator is ready */
ICS_C1 = 0x10; /* external clock reference (31.25kHz) to FLL, RDIV = 2, external prescalar =
128 */
ICS_C2 = 0x20; /* BDIV = 1, prescalar = 2 */
```

## 8.3.4 Initializing FBE mode

The following code segment demonstrates setting ICS to FBE mode.

### Example: 8.3.4.1 FBE mode initialization routine

```
/* the following code segment demonstrates setting ICS to FBE mode generating 20MHZ bus*/
/* supposing external 20MHZ crystal is installed in high gain mode */
ICS_OSCSC = 0x96; /* high-range, high-gain, oscillator required */
while (ICS_OSCINIT == 0); /* waiting until oscillator is ready */
ICS_C1 = 0x80; /* external clock reference (20MHZ) to FLL output */
ICS_C2 = 0x00; /* BDIV = 0, prescalar = 1 */
```

## 8.3.5 External oscillator (OSC)

The oscillator module provides the reference clock for internal reference clock module (ICS), the real time counter clock module, and other MCU sub-systems.

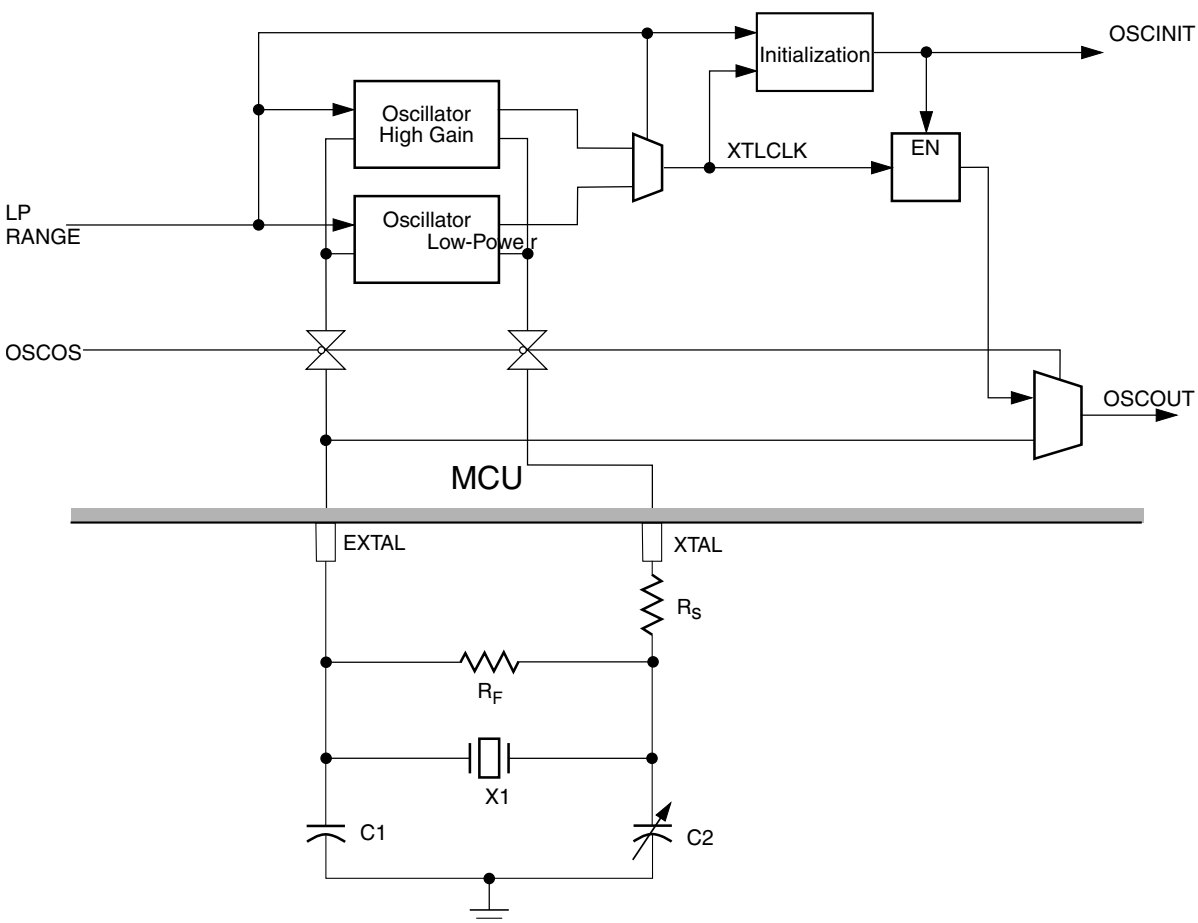



**Figure 8-3. Oscillator module block diagram**

The external oscillator circuit is designed for use with an external crystal or ceramic resonator to provide an accurate clock source. In its typical configuration, the oscillator is connected in a Pierce oscillator configuration, as shown in the above figure. This figure shows only the logical representation of the internal components and may not represent actual circuitry. The oscillator configuration uses five components:

- Crystal or ceramic, X1
- Fixed capacitor, C1
- Tuning capacitor, C2, which can also be a fixed capacitor
- Feedback resistor, $R_B$
- Series resistor, $R_S$ (optional)

### 8.3.5.1 Bypass mode

In bypass mode (ICS_OSCSC[OSCEN] = 0, ICS_OSCSC[OSCOS] = 0), external clock module is disabled. EXTAL can be used as the input of external clock source. When external clock source is not used in this mode, the EXTAL can be used as GPIO or other function muxed with this pinout. XTAL can be used as GPIO or other function muxed with its pinout even EXTAL is used as external clock source. The following figure shows the typical OSC bypass mode connection.

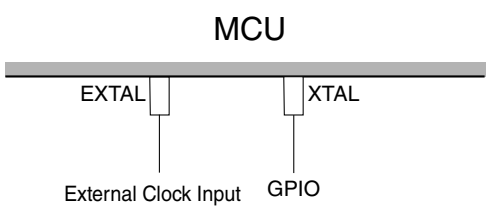



**Figure 8-4. OSC bypass mode connection**

### 8.3.5.2 Low-power configuration

In low-power mode, when ICS_OSCSC[OSCEN] = 1, ICS_OSCSC[OSCOS] = 1, and ICS_OSCSC[HGO] = 0, the series resistor $R_S$ is not used. The feedback resistor $R_F$ must be carefully selected to get best performance. The figure below shows the typical OSC low-gain mode connection.

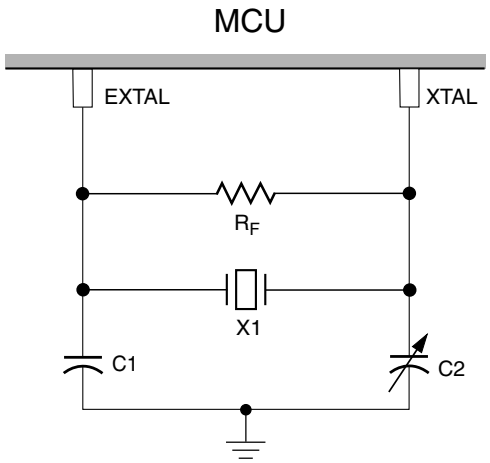



**Figure 8-5. OSC low-power mode connection**

### 8.3.5.3 High-gain configuration

In high-gain mode, when ICS_OSCSC[OSCEN] = 1, ICS_OSCSC[OSCOS] = 1, and ICS_OSCSC[HGO] = 1, the series resistor $R_S$ must be used. The series resistor $R_S$ and feedback resistor $R_F$ must be carefully selected to get best performance. The following figure shows the typical OSC high-gain mode connection.

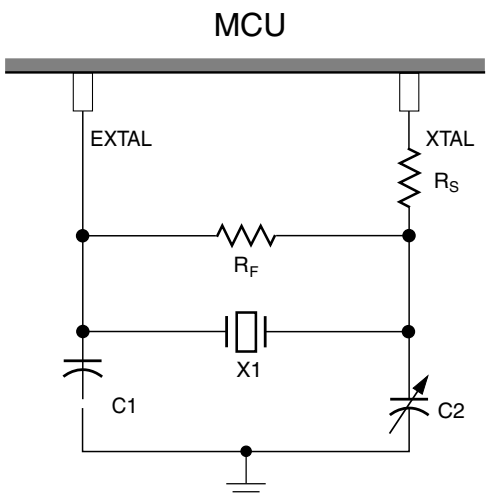



**Figure 8-6. OSC high-gain mode connection**

### 8.3.5.4 Initializing external oscillator for peripherals

The following code segment demonstrates initializing external oscillator.

#### Example: 8.3.5.4.1 External oscillator initialization routine

```
/* the following code segment demonstrates initializing external oscillator */
/* supposing external 32768Hz crystal is installed in low power mode */
ICS_OSCSC = 0xA0; /* low-range, low-power, oscillator required, ERCLK enabled in stop mode */
while (ICS_OSCSC_OSCINIT == 0); /* waiting until oscillator is ready */
```

## 8.4 1 kHz low-power oscillator (LPO)

The 1 kHz low-power oscillator acts as a standalone low-frequency clock source in all run, wait, and stop3 modes.

## 8.5 Peripheral clock gating

This device includes a clock gating system to manage the bus clock sources to the individual peripherals. Using this system, the user can enable or disable the bus clock to each of the peripherals at the clock source, eliminating unnecessary clocks to peripherals that are not in use, thereby reducing the overall run and wait mode currents.

Out of reset, all peripheral clocks will be enabled. For lowest possible run wait currents, user software should disable the clock source to any peripheral not in use. The actual clock will be enabled or disabled immediately following the write to the Clock Gating Control registers (SCG_Cx). Any peripheral with a gated clock cannot be used unless its clock is enabled. Writing to the registers of a peripheral with a disabled clock has no effect.

**Note**

User software should disable the peripheral before disabling the clocks to the peripheral. When clocks are re-enabled to a peripheral, the peripheral registers need to be re-initialized by user software.

In stop modes, the bus clock is disabled for all gated peripherals, regardless of the setting in SCG_Cx registers.

## 8.6 ICS control registers

## ICS memory map

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 3038 | ICS Control Register 1 (ICS_C1) | 8 | R/W | 04h | 8.6.1/172 |
| 3039 | ICS Control Register 2 (ICS_C2) | 8 | R/W | 20h | 8.6.2/173 |
| 303A | ICS Control Register 3 (ICS_C3) | 8 | R/W | Undefined | 8.6.3/174 |
| 303B | ICS Control Register 4 (ICS_C4) | 8 | R/W | 00h | 8.6.4/174 |
| 303C | ICS Status Register (ICS_S) | 8 | R | 10h | 8.6.5/175 |
| 303E | OSC Status and Control Register (ICS_OSCSC) | 8 | R/W | 00h | 8.6.6/176 |

## 8.6.1 ICS Control Register 1 (ICS_C1)

Address: 3038h base + 0h offset = 3038h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read Write | CLKS | | RDIV | | | IREFS | IRCLKEN | IREFSTEN |
| Reset | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

## ICS_C1 field descriptions

| Field | Description |
|---|---|
| 7–6 CLKS | Clock Source Select<br><br>Selects the clock source that controls the bus frequency. The actual bus frequency depends on the value of the BDIV bits.<br><br>00 Output of FLL is selected.<br>01 Internal reference clock is selected.<br>10 External reference clock is selected.<br>11 Reserved. |
| 5–3 RDIV | Reference Divider<br><br>Selects the amount to divide down the FLL reference clock selected by the IREFS bits. Resulting frequency must be in the range 31.25 kHz to 39.0625 kHz. |

| RDIV | ICS_OSCSC[RANGE]= 0 | ICS_OSCSC[RANGE]= 1 |
|---|---|---|
| 000 | 1[1] | 32 |
| 001 | 2 | 64 |
| 010 | 4 | 128 |
| 011 | 8 | 256 |
| 100 | 16 | 512 |
| 101 | 32 | 1024 |
| 110 | 64 | Reserved |
| 111 | 128 | Reserved |

*Table continues on the next page...*

**ICS_C1 field descriptions (continued)**

| Field | Description |
|---|---|
| | 1. Reset default |
| 2<br>IREFS | Internal Reference Select<br><br>The IREFS bit selects the reference clock source for the FLL.<br><br>0 External reference clock selected.<br>1 Internal reference clock selected. |
| 1<br>IRCLKEN | Internal Reference Clock Enable<br><br>The IRCLKEN bit enables the internal reference clock for use as ICSIRCLK.<br><br>0 ICSIRCLK inactive.<br>1 ICSIRCLK active. |
| 0<br>IREFSTEN | Internal Reference Stop Enable<br><br>The IREFSTEN bit controls whether or not the internal reference clock remains enabled when the ICS enters stop mode.<br><br>0 Internal reference clock is disabled in stop.<br>1 Internal reference clock stays enabled in stop if IRCLKEN is set or if ICS is in FEI, FBI, or FBILP mode before entering stop. |

1. Reset default

## 8.6.2 ICS Control Register 2 (ICS_C2)

Address: 3038h base + 1h offset = 3039h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | BDIV | | | LP | 0 | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**ICS_C2 field descriptions**

| Field | Description |
|---|---|
| 7–5<br>BDIV | Bus Frequency Divider<br><br>Selects the amount to divide down the clock source selected by the CLKS bits. This controls the bus frequency.<br><br>000 Encoding 0 - Divides selected clock by 1.<br>001 Encoding 1 - Divides selected clock by 2.<br>010 Encoding 2 - Divides selected clock by 4.<br>011 Encoding 3 - Divides selected clock by 8.<br>100 Encoding 4 - Divides selected clock by 16.<br>101 Encoding 5 - Divides selected clock by 32.<br>110 Encoding 6 - Divides selected clock by 64.<br>111 Encoding 7 - Divides selected clock by 128. |

*Table continues on the next page...*

**ICS_C2 field descriptions (continued)**

| Field | Description |
|---|---|
| 4<br>LP | Low Power Select<br><br>The LP bit controls whether the FLL is disabled in FLL bypassed modes.<br><br>0 FLL is not disabled in bypass mode.<br>1 FLL is disabled in bypass modes unless BDM is active. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 8.6.3 ICS Control Register 3 (ICS_C3)

Address: 3038h base + 2h offset = 303Ah

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | SCTRIM | | | | | | | |
| Reset | x* | x* | x* | x* | x* | x* | x* | x* |

* Notes:
- x = Undefined at reset.

**ICS_C3 field descriptions**

| Field | Description |
|---|---|
| SCTRIM | Slow Internal Reference Clock Trim Setting<br><br>The SCTRIM bits control the slow internal reference clock frequency by controlling the internal reference clock period. The bits are binary weighted. In other words, bit 1 adjusts twice as much as bit 0. Increasing the binary value in SCTRIM will increase the period, and decreasing the value will decrease the period. An additional fine trim bit is available in ICSC4 as the SCFTRIM bit. |

## 8.6.4 ICS Control Register 4 (ICS_C4)

Address: 3038h base + 3h offset = 303Bh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | LOLIE | 0 | CME | 0 | | | | SCFTRIM |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ICS_C4 field descriptions**

| Field | Description |
|---|---|
| 7<br>LOLIE | Loss of Lock Interrupt<br><br>Determines if an interrupt request is made following a loss of lock indication. The LOLIE bit has an effect only when LOLS is set. |

*Table continues on the next page...*

**ICS_C4 field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 No request on loss of lock.<br>1 Generate an interrupt request on loss of lock. |
| 6<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 5<br>CME | Clock Monitor Enable<br><br>Determines if a reset request is made following a loss of external clock indication. The CME bit should be set to a logic 1 only when the ICS is in an operational mode that uses the external clock (FEE or FBE).<br><br>0 Clock monitor is disabled.<br>1 Generate a reset request on loss of external clock. |
| 4–1<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 0<br>SCFTRIM | Slow Internal Reference Clock Fine Trim<br><br>The SCFTRIM bit controls the smallest adjustment of the internal reference clock frequency. Setting SCFTRIM will increase the period and clearing SCFTRIM will decrease the period by the smallest amount possible.<br><br>**NOTE:** SCFTRIM is loaded during reset from a factory programmed location when not in any BDM mode. |

## 8.6.5 ICS Status Register (ICS_S)

Address: 3038h base + 4h offset = 303Ch

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | LOLS | LOCK | 0 | IREFST | CLKST | | 0 | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

**ICS_S field descriptions**

| Field | Description |
|---|---|
| 7<br>LOLS | Loss of Lock Status<br><br>This bit is a sticky indication of lock status for the FLL. LOLS is set when lock detection is enabled and after acquiring lock, the FLL output frequency has fallen outside the lock exit frequency tolerance, $D_{unl}$. LOLIE determines whether an interrupt request is made when set. LOLS is cleared by reset or by writing a logic 1 to LOLS when LOLS is set. Writing a logic 0 to LOLS has no effect.<br><br>0 FLL has not lost lock since LOLS was last cleared.<br>1 FLL has lost lock since LOLS was last cleared. |
| 6<br>LOCK | Lock Status<br><br>Indicates whether the FLL has acquired lock. Lock detection is disabled when FLL is disabled. If the lock status bit is set then changing the value of any of the following bits IREFS, RDIV[2:0], or, if in FEI or FBI modes, TRIM[7:0] will cause the lock status bit to clear and stay cleared until the FLL has reacquired lock. |

*Table continues on the next page...*

**ICS_S field descriptions (continued)**

| Field | Description |
|---|---|
| | Stop mode entry will also cause the lock status bit to clear and stay cleared until the FLL has reacquired lock.<br>**NOTE:** Wait at least for Tqcquire after wake from stop mode to start timing critical tasks like serial communication. Do not need to wait for LOCK bit to set after wake from stop mode.<br>0 FLL is currently unlocked.<br>1 FLL is currently locked. |
| 5<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 4<br>IREFST | Internal Reference Status<br>The IREFST bit indicates the current source for the reference clock. The IREFST bit does not update immediately after a write to the IREFS bit due to internal synchronization between clock domains.<br>0 Source of reference clock is external clock.<br>1 Source of reference clock is internal clock. |
| 3–2<br>CLKST | Clock Mode Status<br>The CLKST bits indicate the current clock mode. The CLKST bits don't update immediately after a write to the CLKS bits due to internal synchronization between clock domains.<br>00 Output of FLL is selected.<br>01 FLL Bypassed, internal reference clock is selected.<br>10 FLL Bypassed, external reference clock is selected.<br>11 Reserved. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 8.6.6 OSC Status and Control Register (ICS_OSCSC)

Address: 3038h base + 6h offset = 303Eh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | OSCEN | 0 | OSCSTEN | OSCOS | 0 | RANGE | HGO | OSCINIT |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ICS_OSCSC field descriptions**

| Field | Description |
|---|---|
| 7<br>OSCEN | OSC Enable<br>The OSCEN bit enables the external clock for use as ICSERCLK.<br>0 OSC module disabled.<br>1 OSC module enabled. |

*Table continues on the next page...*

**ICS_OSCSC field descriptions (continued)**

| Field | Description |
|---|---|
| 6<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 5<br>OSCSTEN | OSC Enable in Stop mode<br><br>The OSCSTEN bit controls whether or not the OSC clock remains enabled when MCU enters stop mode.<br><br>0 OSC clock is disabled in stop.<br>1 OSC stays enabled in stop if OSCEN is set or if ICS is set or ICS requests to be active before entering stop. |
| 4<br>OSCOS | OSC Output Select<br><br>This bit is used to select the output clock of OSC module.<br><br>0 External clock source from EXTAL pin is selected.<br>1 Oscillator clock source is selected. |
| 3<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 2<br>RANGE | Frequency Range Select<br><br>Selects the frequency range for the OSC module.<br><br>0 Low frequency range of 31.25kHz - 39.0625kHz.<br>1 High frequency range of 4 - 20MHz. |
| 1<br>HGO | High Gain Oscillator Select<br><br>The HGO bit controls the OSC mode of operation.<br><br>0 Low gain mode.<br>1 High gain mode. |
| 0<br>OSCINIT | OSC Initialization<br><br>This bit set after the initialization cycles of oscillator completes.<br><br>0 Oscillator initialization not completes.<br>1 Oscillator initialization completed. |

# 8.7 System clock gating control registers

**SCG memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 300C | System Clock Gating Control 1 Register (SCG_C1) | 8 | R/W | A3h | 8.7.1/178 |
| 300D | System Clock Gating Control 2 Register (SCG_C2) | 8 | R/W | 3Ch | 8.7.2/179 |
| 300E | System Clock Gating Control 3 Register (SCG_C3) | 8 | R/W | 30h | 8.7.3/180 |
| 300F | System Clock Gating Control 4 Register (SCG_C4) | 8 | R/W | 29h | 8.7.4/180 |

## 8.7.1 System Clock Gating Control 1 Register (SCG_C1)

This high page register contains control bits to enable or disable the bus clock to the FTMs, MTIMs, and RTC modules. Gating off the clocks to unused peripherals is used to reduce the MCU's run and wait currents.

**NOTE**

User software should disable the peripheral before disabling the clocks to the peripheral. When clocks are re-enabled to a peripheral, the peripheral registers need to be re-initialized by user software.

Address: 300Ch base + 0h offset = 300Ch

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | FTM2 | 0 | FTM0 | 0 | | | MTIM0 | RTC |
| Write | | | | | | | | |
| Reset | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

**SCG_C1 field descriptions**

| Field | Description |
|---|---|
| 7<br>FTM2 | FTM2 Clock Gate Control<br><br>This bit controls the clock gate to the FTM2 module.<br><br>0 Bus clock to the FTM2 module is disabled.<br>1 Bus clock to the FTM2 module is enabled. |
| 6<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 5<br>FTM0 | FTM0 Clock Gate Control<br><br>This bit controls the clock gate to the FTM0 module.<br><br>0 Bus clock to the FTM0 module is disabled.<br>1 Bus clock to the FTM0 module is enabled. |
| 4–2<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 1<br>MTIM0 | MTIM0 Clock Gate Control<br><br>This bit controls the clock gate to the MTIM0 module.<br><br>0 Bus clock to the MTIM0 module is disabled.<br>1 Bus clock to the MTIM0 module is enabled. |
| 0<br>RTC | RTC Clock Gate Control<br><br>This bit controls the clock gate to the RTC module.<br><br>0 Bus clock to the MTRTCIM1 module is disabled.<br>1 Bus clock to the RTC module is enabled. |

## 8.7.2 System Clock Gating Control 2 Register (SCG_C2)

This high-page register contains control bits to enable or disable the bus clock to the DBG, NVM, CRC, and IPC modules. Gating off the clocks to unused peripherals is used to reduce the MCU's run and wait currents.

**NOTE**

User software should disable the peripheral before disabling the clocks to the peripheral. When clocks are re-enabled to a peripheral, the peripheral registers need to be re-initialized by user software.

Address: 300Ch base + 1h offset = 300Dh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | DBG | NVM | IPC | CRC | 0 | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

**SCG_C2 field descriptions**

| Field | Description |
|---|---|
| 7–6<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 5<br>DBG | DBG Clock Gate Control<br><br>This bit controls the clock gate to the DBG module.<br><br>0 Bus clock to the DBG module is disabled.<br>1 Bus clock to the DBG module is enabled. |
| 4<br>NVM | NVM Clock Gate Control<br><br>This bit controls the clock gate to the NVM module.<br><br>0 Bus clock to the NVM module is disabled.<br>1 Bus clock to the NVM module is enabled. |
| 3<br>IPC | IPC Clock Gate Control<br><br>This bit controls the clock gate to the IPC module.<br><br>0 Bus clock to the IPC module is disabled.<br>1 Bus clock to the IPC module is enabled. |
| 2<br>CRC | CRC Clock Gate Control<br><br>This bit controls the clock gate to the CRC module.<br><br>0 Bus clock to the CRC module is disabled.<br>1 Bus clock to the CRC module is enabled. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 8.7.3 System Clock Gating Control 3 Register (SCG_C3)

This high page register contains control bits to enable or disable the bus clock to the SCI modules. Gating off the clocks to unused peripherals is used to reduce the MCU's run and wait currents.

**NOTE**

User software should disable the peripheral before disabling the clocks to the peripheral. When clocks are re-enabled to a peripheral, the peripheral registers need to be re-initialized by user software.

Address: 300Ch base + 2h offset = 300Eh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | SCI1 | SCI0 | 0 | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

**SCG_C3 field descriptions**

| Field | Description |
|---|---|
| 7–6<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 5<br>SCI1 | SCI1 Clock Gate Control<br><br>This bit controls the clock gate to the SCI1 module.<br><br>0 Bus clock to the SCI1 module is disabled.<br>1 Bus clock to the SCI1 module is enabled. |
| 4<br>SCI0 | SCI0 Clock Gate Control<br><br>This bit controls the clock gate to the SCI0 module.<br><br>0 Bus clock to the SCI0 module is disabled.<br>1 Bus clock to the SCI0 module is enabled. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 8.7.4 System Clock Gating Control 4 Register (SCG_C4)

This high page register contains control bits to enable or disable the bus clock to the ADC, IRQ, and KBI modules. Gating off the clocks to unused peripherals is used to reduce the MCU's run and wait currents.

## NOTE

User software should disable the peripheral before disabling the clocks to the peripheral. When clocks are re-enabled to a peripheral, the peripheral registers need to be re-initialized by user software.

Address: 300Ch base + 3h offset = 300Fh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | ADC | 0 | IRQ | 0 | | KBI0 |
| Write | | | | | | | | |
| Reset | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |

### SCG_C4 field descriptions

| Field | Description |
|---|---|
| 7–6<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 5<br>ADC | ADC Clock Gate Control<br><br>This bit controls the clock gate to the ADC module.<br><br>0 Bus clock to the ADC module is disabled.<br>1 Bus clock to the ADC module is enabled. |
| 4<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 3<br>IRQ | IRQ Clock Gate Control<br><br>This bit controls the clock gate to the IRQ module.<br><br>0 Bus clock to the IRQ module is disabled.<br>1 Bus clock to the IRQ module is enabled. |
| 2–1<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 0<br>KBI0 | KBI0 Clock Gate Control<br><br>This bit controls the clock gate to the KBI0 module.<br><br>0 Bus clock to the KBI0 module is disabled.<br>1 Bus clock to the KBI0 module is enabled. |

# Chapter 9
# Chip configurations

## 9.1 Introduction

This chapter provides details on the individual modules of the device. It includes:

- device block diagrams highlighting the specific modules and pin-outs
- specific module-to-module interactions not necessarily discussed in the individual module chapters, and
- links for more information

## 9.2 Core modules

### 9.2.1 Central processor unit (CPU)

The HCS08 CPU is fully source- and object-code-compatible with the M68HC08 CPU. Several instructions and enhanced addressing modes were added to improve C compiler efficiency and to support a new background debug system which replaces the monitor mode of earlier M68HC08 microcontrollers.

### 9.2.2 Debug module (DBG)

The DBG module implements an on-chip ICE (in-circuit emulation) system and allows non-intrusive debug of application software by providing an on-chip trace buffer with flexible triggering capability. The trigger can also provide extended breakpoint capacity. The on-chip ICE system is optimized for the HCS08 8-bit architecture and supports 64 KB of memory space.

## 9.3 System modules

### 9.3.1 Watchdog (WDOG)

The watchdog timer (WDOG) module triggers a system reset if it is allowed to time out. The program is expected to periodically reload the watchdog timer, thereby preventing it from timing out. However, if a fault occurs that causes the program to stop working, the timer will not be reloaded and it will time out. The resulting trigger of a system reset brings the system back from an unresponsive state into a normal state.

## 9.4 Clock module

This device has ICS, XOSC, and LPO clock modules.

The internal clock source (ICS) module provides several clock source options for this device. The module contains a frequency-locked loop (FLL) that is controllable by either an internal or external reference clock. The module can select clock from the FLL or bypass the FLL as a source of the MCU system clock. The selected clock source is passed through a reduced bus divider, which allows a lower output clock frequency to be derived.

The external oscillator (XOSC) module allows an external crystal, ceramic resonator, or other external clock source to produce the external reference clock. The output of XOSC module can be used as the reference of ICS to generate system bus clock, and/or clock source of watchdog (WDOG), real-time counter (RTC), and analog-to-digital (ADC) modules.

The low-power oscillator (LPO) module is an on-chip low-power oscillator providing 1 kHz reference clock to RTC and watchdog (WDOG).

## 9.5 Memory

### 9.5.1 Random-access-memory (RAM)

This device contains 2,048 byte static RAM and addresses 0x0040 through 0x083F. The location of the stack RAM is programmable. The 16-bit stack pointer allows the stack to be anywhere in the 64 KB memory space.

### 9.5.2 Non-volatile memory (NVM)

The NVM is ideal for single-supply applications allowing for field programming without requiring external high voltage sources from program or erase operations. The NVM module includes a memory controller that executes commands to modify NVM contents.

This device contains two types of non-volatile memory: Flash memory and EEPROM. The Flash is mostly used for the storage of program and constant. The EEPROM is used for storing frequently modified non-volatile data.

Non-volatile memory (NVM) includes:

- Flash memory
  - MC9S08PL16—16,384 bytes: 32 sectors of 512 bytes
  - MC9S08PL8—8,192 bytes: 16 sectors of 512 bytes
- EEPROM memory
  - MC9S08PL16—256 bytes: 128 sector of 2 bytes
  - MC9S08PL8—256 bytes: 128 sector of 2 bytes

## 9.6 Power modules

This device contains on-chip regulator for various operational power modes of run, wait, and stop3 modes. The low voltage detect (LVD) system allows the system to protect against low voltage conditions in order to protect memory contents and control MCU system states during supply voltage variations. The on-chip bandgap reference (≈1.2V), which is internally connected to ADC channel, provides independent accuracy reference which will not drop over the full operating voltage even when the operating voltage is falling.

## 9.7 Security

### 9.7.1 Cyclic redundancy check (CRC)

The CRC generator module uses a programmable polynomial to generate CRC code for error detection. The 16-bit code is calculated for 8-bit of data at a time, and provides a simple check for all accessible memory locations in flash and RAM.

# 9.8 Timers

## 9.8.1 FlexTimer module (FTM)

The FlexTimer module is an up to six-channel timer that supports input capture, output compare, and the generation of PWM signals to control electric motor and power management applications. FTM time reference is a 16-bit counter that can be used as an unsigned or signed counter.

This MCU contains up to two FTM modules with one 6-channel FTM and one 2-channel FTMs. FTM0 has two independent channels, are fully compatible to the TPM. FTM2, which has six independent channels, is also fully compatible to the TPM.

Each FTM module has independent external clock input. The following table summarizes the external signals of FTM modules.

**Table 9-1. FTM module external signals**

| FTM | Functions | Default | Alternate |
|---|---|---|---|
| FTM0 | channel 0 | PTA0/FTM0CH0 | PTC4/FTM0CH0 |
| | channel 1 | PTA1/FTM0CH1 | PTC5/FTM0CH1 |
| | alternate clock | PTA5/TCLK0 | |
| FTM2 | channel 0 | PTC0/FTM2CH0 | |
| | channel 1 | PTC1/FTM2CH1 | |
| | channel 2 | PTC2/FTM2CH2 | PTD0/FTM2CH2 |
| | channel 3 | PTC3/FTM2CH3 | PTD1/FTM2CH3 |
| | channel 4 | PTB4/FTM2CH4 | |
| | channel 5 | PTB5/FTM2CH5 | |
| | alternate clock | PTE4/TCLK2 | |

### 9.8.1.1 FTM0 interconnection

SCI0 TxD signal can be modulated by FTM0 channel 0 PWM output. Please refer to SCI0 TxD modulation.

### 9.8.1.2 FTM2 interconnection

Please refer to System interconnection.

### 9.8.2 8-bit modulo timer (MTIM)

MTIM0 8-bit modulo timer module that provide a circuit of selectable clock sources and a programmable interrupt. The MTIM module contain an 8-bit modulo counter, which can operate as a free-running counter or a modulo counter. A timer overflow interrupt can be enabled to generate periodic interrupts for time-based software events. MTIM module may also use external clock source.

#### 9.8.2.1 MTIM0 as ADC hardware trigger

MTIM0 overflow can be used as ADC hardware trigger. See ADC hardware trigger for details.

### 9.8.3 Real-time counter (RTC)

The real-time counter (RTC) consists of one 16-bit counter, one 16-bit comparator, several binary-based and decimal-based prescaler dividers, two clock sources, and one programmable periodic interrupt. This module can be used for time-of-day, calendar or any task scheduling functions. It can also serve as a cyclic wakeup from low power modes without external components. RTC overflow trigger can be used as hardware trigger for ADC module. Furthermore, when the trigger is enabled, RTC can toggle external pin function if the counter overflows.

## 9.9 Communication interfaces

### 9.9.1 Serial communications interface (SCI)

This device includes two independent serial communications interface (SCI) modules. Typically, these systems are used to connect to the RS232 serial input/output port of a personal computer or workstation. They can also be used to communicate with other embedded controllers.

A flexible, 13-bit, modulo-based baud rate generator supports a broad range of standard baud rates beyond 115.2 kBd. Transmit and receive within the same SCI use a common baud rate, and each SCI module has a separate baud rate generator.

This SCI system offers many advanced features not commonly found on other asynchronous serial I/O peripherals of the embedded controllers. The receiver employs an advanced data sampling technique that ensures reliable communication and noise detection. Hardware parity, receiver wakeup, and double buffering on transmit and receive are also included.

#### 9.9.1.1 SCI0 infrared functions

##### 9.9.1.1.1 SCI0 TxD modulation

SCI0 TxD output can be modulated by FTM0 channel 0 PWM output. Please refer to SCI0 TxD modulation.

## 9.10 Analog

### 9.10.1 Analog-to-digital converter (ADC)

This device contains an analog-to-digital converter (ADC) of 10-bit, a successive approximation ADC for operation within an integrated microcontroller system-on-chip. The ADC channel assignments, alternate clock function, and hardware trigger function are configured as described following sections.

#### 9.10.1.1 ADC channel assignments

The ADC channel assignments for the device are shown in the following table. Reserved channels convert to an unknown value.

**Table 9-2. ADC channel assignments**

| ADCH | Channel | Input |
|---|---|---|
| 00000 | AD0 | PTA0/ADP0 |
| 00001 | AD1 | PTA1/ADP1 |
| 00010 | AD2 | PTA6/ADP2 |
| 00011 | AD3 | PTA7/ADP3 |
| 00100 | AD4 | PTB0/ADP4 |
| 00101 | AD5 | PTB1/ADP5 |
| 00110 | AD6 | PTB2/ADP6 |
| 00111 | AD7 | PTB3/ADP7 |
| 01000 | AD8 | PTC0/ADP8 |

*Table continues on the next page...*

**Table 9-2. ADC channel assignments (continued)**

| ADCH | Channel | Input |
|---|---|---|
| 01001 | AD9 | PTC1/ADP9 |
| 01010 | AD10 | PTC2/ADP10 for 32-pin packages and $V_{SS}$ for 20-pin and 16-pin packages |
| 01011 | AD11 | PTC3/ADP11 for 32-pin packages and $V_{SS}$ for 20-pin and 16-pin packages |
| 01100 | AD12 | $V_{SS}$ |
| 01101 | AD13 | $V_{SS}$ |
| 01110 | AD14 | $V_{SS}$ |
| 01111 | AD15 | $V_{SS}$ |
| 10000 | AD16 | $V_{SS}$ |
| 10001 | AD17 | $V_{SS}$ |
| 10010 | AD18 | $V_{SS}$ |
| 10011 | AD19 | Reserved |
| 10100 | AD20 | Reserved |
| 10101 | AD21 | Reserved |
| 10110 | AD22 | Temperature sensor |
| 10111 | AD23 | Bandgap |
| 11000 | AD24 | Reserved |
| 11001 | AD25 | Reserved |
| 11010 | AD26 | Reserved |
| 11011 | AD27 | Reserved |
| 11100 | AD28 | Reserved |
| 11101 | AD29 | $V_{REFH}$ |
| 11110 | AD30 | $V_{REFL}$ |
| 11111 | Module disabled | None |

### 9.10.1.2 Alternate clock

The ADC module is capable of performing conversions using the MCU bus clock, the bus clock divided by two, the local asynchronous clock (ADACK) within the module, or the alternate clock, ALTCLK. The alternate clock for the devices is the external oscillator output (OSCOUT).

The selected clock source must run at a frequency such that the ADC conversion clock (ADCK) runs at a frequency within its specified range ($f_{ADCK}$) after being divided down from the ALTCLK input as determined by the ADIV bits.

ALTCLK is active while the MCU is in wait mode provided the conditions described above are met. This allows ALTCLK to be used as the conversion clock source for the ADC while the MCU is in wait mode.

ALTCLK cannot be used as the ADC conversion clock source while the MCU is in stop3 mode.

### 9.10.1.3 Hardware trigger

The ADC hardware trigger is selectable from MTIM0 overflow, RTC overflow. The MCU can be configured to use any of those hardware trigger sources in run and wait modes. The RTC overflow can be used as ADC hardware trigger in STOP3 mode. Please refer to ADC hardware trigger.

### 9.10.1.4 Temperature sensor

The ADC module integrates an on-chip temperature sensor. Following actions must be performed to use this temperature sensor.

- Configure ADC for long sample with a maximum of 1 MHz clock
- Convert the bandgap voltage reference channel (AD23)
  - By converting the digital value of the bandgap voltage reference channel using the value of $V_{BG}$ the user can determine $V_{DD}$.
- Convert the temperature sensor channel (AD22)
  - By using the calculated value of $V_{DD}$, convert the digital value of AD22 into a voltage, $V_{TEMP}$

The following equation provides an approximate transfer function of the on-chip temperature sensor for $V_{DD}$ = 5.0V, Temp = 25°C, using the ADC at $f_{ADCK}$ = 1.0 MHz and configured for long sample.

$$\text{Temp} = 25 - ((V_{TEMP} - V_{TEMP25}) \div m)$$

where:

- $V_{TEMP}$ is the voltage of the temperature sensor channel at the ambient temperature
- $V_{TEMP25}$ is the voltage of the temperature sensor channel at 25 °C
- m is the hot or cold voltage versus temperature slope in V/°C

For temperature calculations, use the $V_{TEMP25}$ and m values in the data sheet.

In application code, you read the temperature sensor channel, calculate $V_{TEMP}$, and compare it to $V_{TEMP25}$. If $V_{TEMP}$ is greater than $V_{TEMP25}$, the cold slope value is applied in the above equation. If $V_{TEMP}$ is less than $V_{TEMP25}$ the hot slope value is applied.

Calibrating at 25°C will improve accuracy to ±4.5°C.

Calibration at three points -40°C, 25°C, and 85°C will improve accuracy to ±2.5°C. After calibration has been completed, you will need to calculate the slope for both hot and cold. In application code, you can calculate the temperature as detailed above and determine if it is above or below 25°C. After you have determined whether the temperature is above or below 25 °C. you can recalculate the temperature using the hot or cold slope value obtained during calibration.

## 9.11 Human-machine interfaces HMI

### 9.11.1 Keyboard interrupts (KBI)

This device has one KBI modules with up to 8 keyboard interrupt inputs grouped in a KBI modules available depending on packages.

# Chapter 10
# Central processor unit

## 10.1 Introduction

This section provides summary information about the registers, addressing modes, special operations, instructions and exceptions processing of the HCS08 V6 CPU.

The HCS08 V6 CPU is fully source- and object-code-compatible with the HCS08 CPU.

### 10.1.1 Features

Features of the HCS08 V6 CPU include:

- Object code fully upward-compatible with M68HC05 and M68HC08 families
- 16-bit stack pointer (any size stack anywhere in 64 KB CPU address space)
- 16-bit index register (H:X) with powerful indexed addressing modes
- 8-bit accumulator (A)
- Many instructions treat X as a second general-purpose 8-bit register
- Seven addressing modes:
  - Inherent — Operands in internal registers
  - Relative — 8-bit signed offset to branch destination
  - Immediate — Operand in next object code byte(s)
  - Direct — Operand in memory at 0x0000–0x00FF
  - Extended — Operand anywhere in 64-Kbyte address space

- Indexed relative to H:X — Five submodes including auto increment
- Indexed relative to SP — Improves C efficiency dramatically

- Memory-to-memory data move instructions with four address mode combinations
- Overflow, half-carry, negative, zero, and carry condition codes support conditional branching on the results of signed, unsigned, and binary-coded decimal (BCD) operations
- Efficient bit manipulation instructions
- STOP and WAIT instructions to invoke low-power operating modes

## 10.2 Programmer's Model and CPU Registers

Figure 10-1 shows the five CPU registers. CPU registers are not part of the memory map.

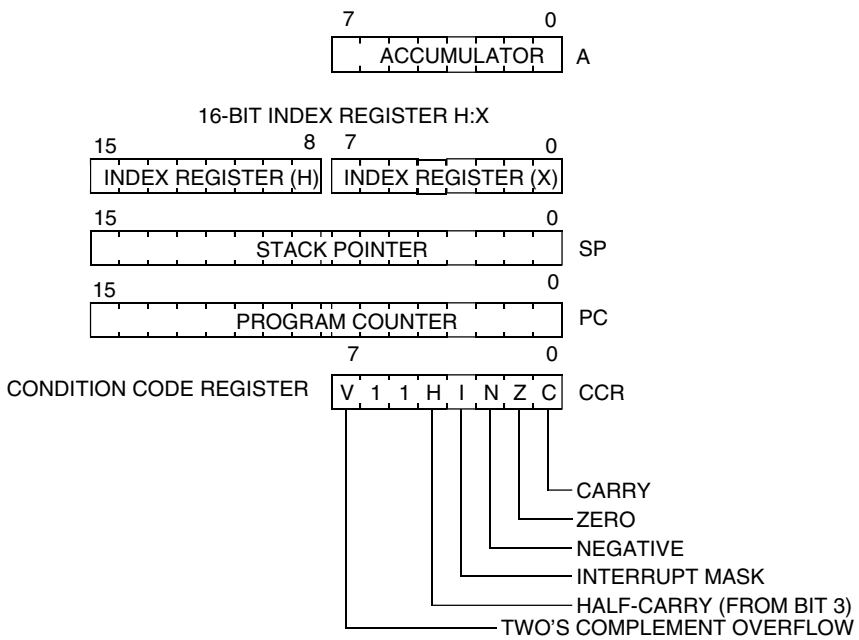



**Figure 10-1. CPU Registers**

### 10.2.1 Accumulator (A)

The A accumulator is a general-purpose 8-bit register. One input operand from the arithmetic logic unit (ALU) is connected to the accumulator, and the ALU results are often stored into the A accumulator after arithmetic and logical operations. The

accumulator can be loaded from memory using various addressing modes to specify the address where the loaded data comes from, or the contents of A can be stored to memory using various addressing modes to specify the address where data from A will be stored.

Reset has no effect on the contents of the A accumulator.

### 10.2.2 Index Register (H:X)

This 16-bit register is actually two separate 8-bit registers (H and X), which often work together as a 16-bit address pointer where H holds the upper byte of an address and X holds the lower byte of the address. All indexed addressing mode instructions use the full 16-bit value in H:X as an index reference pointer; however, for compatibility with the earlier M68HC05 family, some instructions operate only on the low-order 8-bit half (X).

Many instructions treat X as a second general-purpose 8-bit register that can be used to hold 8-bit data values. X can be cleared, incremented, decremented, complemented, negated, shifted, or rotated. Transfer instructions allow data to be transferred from A or transferred to A where arithmetic and logical operations can then be performed.

For compatibility with the earlier M68HC05 family, H is forced to 0x00 during reset. Reset has no effect on the contents of X.

### 10.2.3 Stack Pointer (SP)

This 16-bit address pointer register points at the next available location on the automatic last-in-first-out (LIFO) stack. The stack may be located anywhere in the 64 KB address space that has RAM, and can be any size up to the amount of available RAM. The stack is used to automatically save the return address for subroutine calls, the return address and CPU registers during interrupts, and for local variables. The AIS (add immediate to stack pointer) instruction adds an 8-bit signed immediate value to SP. This is most often used to allocate or deallocate space for local variables on the stack.

SP is forced to 0x00FF at reset for compatibility with the earlier M68HC05 family. HCS08 V6 programs normally change the value in SP to the address of the last location (highest address) in on-chip RAM during reset initialization to free up direct page RAM (from the end of the on-chip registers to 0x00FF).

The RSP (reset stack pointer) instruction was included for compatibility with the M68HC05 family and is seldom used in new HCS08 V6 programs because it affects only the low-order half of the stack pointer.

## 10.2.4 Program Counter (PC)

The program counter is a 16-bit register that contains the address of the next instruction or operand to be fetched.

During normal program execution, the program counter automatically increments to the next sequential memory location every time an instruction or operand is fetched. Jump, branch, interrupt, and return operations load the program counter with an address other than that of the next sequential location. This is called a change-of-flow.

During reset, the program counter is loaded with the reset vector that is located at 0xFFFE and 0xFFFF. The vector stored there is the address of the first instruction that will be executed after exiting the reset state.

## 10.2.5 Condition Code Register (CCR)

The 8-bit condition code register contains the interrupt mask (I) and five flags that indicate the results of the instruction just executed. Bits 6 and 5 are set permanently to 1. The following paragraphs describe the functions of the condition code bits in general terms.

**Table 10-1. CCR Register Field Descriptions**

| Field | Description |
|---|---|
| 7<br>V | **Two's Complement Overflow Flag** — The CPU sets the overflow flag when a two's complement overflow occurs. The signed branch instructions BGT, BGE, BLE, and BLT use the overflow flag.<br>0 No overflow<br>1 Overflow |
| 4<br>H | **Half-Carry Flag** — The CPU sets the half-carry flag when a carry occurs between accumulator bits 3 and 4 during an add-without-carry (ADD) or add-with-carry (ADC) operation. The half-carry flag is required for binary-coded decimal (BCD) arithmetic operations. The DAA instruction uses the states of the H and C condition code bits to automatically add a correction value to the result from a previous ADD or ADC on BCD operands to correct the result to a valid BCD value.<br>0 No carry between bits 3 and 4<br>1 Carry between bits 3 and 4 |
| 3<br>I | **Interrupt Mask Bit** — When the interrupt mask is set, all maskable CPU interrupts are disabled. CPU interrupts are enabled when the interrupt mask is cleared. When a CPU interrupt occurs, the interrupt mask is set automatically after the CPU registers are saved on the stack, but before the first instruction of the interrupt service routine is executed.<br>Interrupts are not recognized at the instruction boundary after any instruction that clears I (CLI or TAP). This ensures that the next instruction after a CLI or TAP will always be executed without the possibility of an intervening interrupt, provided I was set.<br>0 Interrupts enabled |

*Table continues on the next page...*

**Table 10-1. CCR Register Field Descriptions (continued)**

| Field | Description |
|---|---|
| | 1 Interrupts disabled |
| 2<br>N | **Negative Flag** — The CPU sets the negative flag when an arithmetic operation, logic operation, or data manipulation produces a negative result, setting bit 7 of the result. Simply loading or storing an 8-bit or 16-bit value causes N to be set if the most significant bit of the loaded or stored value was 1.<br>0 Non-negative result<br>1 Negative result |
| 1<br>Z | **Zero Flag** — The CPU sets the zero flag when an arithmetic operation, logic operation, or data manipulation produces a result of 0x00 or 0x0000. Simply loading or storing an 8-bit or 16-bit value causes Z to be set if the loaded or stored value was all 0s.<br>0 Non-zero result<br>1 Zero result |
| 0<br>C | **Carry/Borrow Flag** — The CPU sets the carry/borrow flag when an addition operation produces a carry out of bit 7 of the accumulator or when a subtraction operation requires a borrow. Some instructions — such as bit test and branch, shift, and rotate — also clear or set the carry/borrow flag.<br>0 No carry out of bit 7<br>1 Carry out of bit 7 |

## 10.3 Addressing Modes

Addressing modes define the way the CPU accesses operands and data. In the HCS08 V6, memory, status and control registers, and input/output (I/O) ports share a single 64 KB CPU address space. This arrangement means that the same instructions that access variables in RAM can also be used to access I/O and control registers or nonvolatile program space.

Some instructions use more than one addressing mode. For instance, move instructions use one addressing mode to specify the source operand and a second addressing mode to specify the destination address. Instructions such as BRCLR, BRSET, CBEQ, and DBNZ use one addressing mode to specify the location of an operand for a test and then use relative addressing mode to specify the branch destination address when the tested condition is true. For BRCLR, BRSET, CBEQ, and DBNZ, the addressing mode listed in the instruction set tables is the addressing mode needed to access the operand to be tested, and relative addressing mode is implied for the branch destination.

Every addressing mode, except inherent, generates a 16-bit effective address. The effective address is the address of the memory location that the instruction acts on. Effective address computations do not require extra execution cycles. The HCS08 V6 CPU uses the 16 addressing modes described in the following sections.

### 10.3.1 Inherent Addressing Mode (INH)

In this addressing mode, instructions either have no operands or all operands are in internal CPU registers. In either case, the CPU does not need to access any memory locations to complete the instruction. Examples:

```
NOP          ;this instruction has no operands
CLRA         ;operand is a CPU register
```

### 10.3.2 Relative Addressing Mode (REL)

Relative addressing mode is used to specify the destination location for branch instructions. A signed two's complement byte offset value is located in the memory location immediately following the opcode. The offset gives a branching range of -128 to +127 bytes. In most assemblers, the programmer does not need to calculate the offset, because the assembler determines the proper offset and verifies that it is within the span of the branch.

During program execution, if a branch condition is true, the signed offset is sign-extended to a 16-bit value and is added to the current contents of the program counter, which causes program execution to continue at the branch destination address. If a branch condition is false, the CPU executes the next instruction.

### 10.3.3 Immediate Addressing Mode (IMM)

The operand for instructions with the immediate addressing mode is contained in the byte(s) immediately following the opcode. The byte or bytes that follow the opcode are the value of the statement rather than the address of the value. The pound symbol (#) is used to indicate an immediate addressing mode operand. One very common programming error is to accidentally omit the # symbol. This causes the assembler to misinterpret the following expression as an address rather than explicitly provided data. For example LDA #$55 means to load the immediate value $55 into the accumulator, while LDA $55 means to load the value from address $0055 into the accumulator. Without the # symbol, the instruction is erroneously interpreted as a direct addressing instruction.

Example:

```
LDA      #$55
CPHX     #$FFFF
LDHX     #$67
```

The size of the immediate operand is implied by the instruction context. In the third example, the instruction implies a 16-bit immediate value, but only an 8-bit value is supplied. In this case the assembler generates the 16-bit value $0067 because the CPU expects a 16-bit value in the instruction stream.

## 10.3.4 Direct Addressing Mode (DIR)

This addressing mode is sometimes called zero-page addressing because it accesses operands in the address range $0000 through $00FF. Since these addresses always begin with $00, only the low byte of the address needs to be included in the instruction, which saves program space and execution time. A system can be optimized by placing the most commonly accessed data in this area of memory. The low byte of the operand address is supplied with the instruction and the high byte of the address is assumed to be zero.

Examples:

```
LDA       $55
```

The value $55 is taken to be the low byte of an address in the range $0000 through $00FF. The high byte of the address is assumed to be zero. During execution, the CPU combines the value $55 from the instruction with the assumed value of $00 to form the address $0055, which is then used to access the data to be loaded into accumulator.

```
LDHX      $20
```

In this example, the value $20 is combined with the assumed value of $00 to form the address $0020. Since the LDHX instruction requires a 16-bit value, a 16-bit word of data is read from addresses $0020 and $0021. After execution, the H:X index register has the value from address $0020 in its high byte and the value from address $0021 in its low byte. The same happens for CPHX and STHX.

```
BRSET     0,$80,foo
```

In this example, direct addressing is used to access the operand and relative addressing is used to identify the destination address of a branch, in case the branch-taken conditions are met. This is also the case for BRCLR.

## 10.3.5 Extended Addressing Mode (EXT)

In extended addressing, the full 16-bit address of the memory location to be operated on is provided in the instruction. Extended addressing can access any location in the 64 KB memory map.

Example:

```
LDA     $F03B
```

This instruction uses extended addressing because $F03B is above the zero page. In most assemblers, the programmer does not need to specify whether an instruction is direct or extended. The assembler automatically selects the shortest form of the instruction.

## 10.3.6 Indexed Addressing Mode

Indexed addressing mode has seven variations, including five that use the 16-bit H:X index register pair and two that use the stack pointer as the base reference.

### 10.3.6.1 Indexed, No Offset (IX)

Instructions using the indexed, no offset addressing mode are one-byte instructions that can access data with variable addresses. The X (Index register low byte) register contains the low byte of the conditional address of the operand and the H (Index register high byte) register contains the high byte of the address.

Indexed, no offset instructions can move a pointer through a table or hold the address of a frequently used RAM or input/output (I/O) location.

### 10.3.6.2 Indexed, No Offset with Post Increment (IX+)

Instructions using the indexed, no offset with post increment addressing mode are two-byte instructions that address the operands and then increment the Index register (H:X). The X (Index register low byte) register contains the low byte of the conditional address of the operand and the H (Index register high byte) register contains the high byte of the address. This addressing mode is usually used for table searches. MOV and CBEQ instructions use this addressing mode as well.

### 10.3.6.3 Indexed, 8-Bit Offset (IX1)

Indexed with 8-bit offset instructions are two-byte instructions that can access data with a variable address. The CPU adds the unsigned bytes in the H:X register to the unsigned byte immediately following the opcode. The sum is the effective address.

Indexed, 8-bit offset instructions are useful in selecting the k-th element in an n-element table. The table can begin anywhere and can extend as far as the address map allows. The k value would typically be in H:X, and the address of the beginning of the table would be

in the byte following the opcode. Using H:X in this way, this addressing mode is limited to the first 256 addresses in memory. Tables can be located anywhere in the address map when H:X is used as the base address, and the byte following the opcode is the offset.

### 10.3.6.4 Indexed, 8-Bit Offset with Post Increment (IX1+)

Indexed, 8-bit offset with post-increment instructions are three-byte instructions that access the operands with variable addresses, then increment H:X. The CPU adds the unsigned bytes in the H:X register to the byte immediately following the opcode. The sum is the effective address. This addressing mode is generally used for table searches. This addressing mode is used for CBEQ instruction.

### 10.3.6.5 Indexed, 16-Bit Offset (IX2)

Indexed, 16-bit offset instructions are three-byte instructions that can access data with variable addresses at any location in memory. The CPU adds the unsigned contents of H:X to the 16-bit unsigned word formed by the two bytes following the opcode. The sum is the effective address of the operand. The first byte after the opcode is the most significant byte of the 16-bit offset; the second byte is the least significant byte of the 16-bit offset. As with direct and extended addressing, most assemblers determine the shortest form of indexed addressing.

Indexed, 16-bit offset instructions are useful in selecting the k-th element in an n-element table. The table can begin anywhere and can extend as far as the address map allows. The k value would typically be in H:X, and the address of the beginning of the table would be in the bytes following the opcode.

### 10.3.6.6 SP-Relative, 8-Bit Offset (SP1)

Stack pointer, 8-bit offset instructions are three-byte instructions that address operands in much the same way as indexed 8-bit offset instructions, except that the 8-bit offset is added to the value of the stack pointer instead of the index register.

The stack pointer, 8-bit offset addressing mode permits easy addressing of data on the stack. The CPU adds the unsigned byte in the 16-bit stack pointer (SP) register to the unsigned byte following the opcode. The sum is the effective address of the operand. If interrupts are disabled, this addressing mode allows the stack pointer to be used as a second "index" register.

Stack pointer relative instructions require a pre-byte for access. Consequently, all SP relative instructions take one cycle longer than their index relative counterparts.

#### 10.3.6.7 SP-Relative, 16-Bit Offset (SP2)

Stack pointer, 16-bit offset instructions are four-byte instructions used to access data relative to the stack pointer with variable addresses at any location in memory. The CPU adds the unsigned contents of the 16-bit stack pointer to the 16-bit unsigned word formed by the two bytes following the opcode. The sum is the effective address of the operand.

As with direct and extended addressing, most assemblers determine the shortest form of stack pointer addressing. Due to the pre-byte, stack pointer relative instructions take one cycle longer than their index relative counterparts.

Stack pointer, 16-bit offset instructions are useful in selecting the k-th element a an n-element table. The table can begin anywhere and can extend anywhere in memory. The k value would typically be in the stack pointer register, and the address of the beginning of the table is located in the two bytes following the two-byte opcode.

### 10.3.7 Memory to memory Addressing Mode

Memory to memory addressing mode has the following four variations.

#### 10.3.7.1 Direct to Direct

This addressing mode is used to move data within the direct page of memory. Both the source operand and the destination operand are in the direct page. The source data is addressed by the first byte immediately following the opcode, and the destination location is addressed by the second byte following the opcode.

#### 10.3.7.2 Immediate to Direct

This addressing mode is used to move an 8-bit constant to any location in the direct page memory. The source data is the byte immediately following the opcode, and the destination is addressed by the second byte following the opcode.

#### 10.3.7.3 Indexed to Direct, Post Increment

Used only by the MOV instruction, this addressing mode accesses a source operand addressed by the H:X register, and a destination location within the direct page addressed by the byte following the opcode. H:X is incremented after the source operand is accessed.

#### 10.3.7.4 Direct to Indexed, Post-Increment

Used only with the MOV instruction, this addressing mode accesses a source operand addressed by the byte following the opcode, and a destination location addressed by the H:X register. H:X is incremented after the destination operand is written.

## 10.4 Operation modes

The CPU can be placed into the following operation modes: stop, wait, background and security.

### 10.4.1 Stop mode

Usually, all system clocks, including the crystal oscillator (when used), are halted during stop mode to minimize power consumption. In such systems, external circuitry is needed to control the time spent in stop mode and to issue a signal to wake up the target MCU when it is time to resume processing. Unlike the earlier M68HC05 and M68HC08 MCUs, the HCS08 V6 can be configured to keep a minimum set of clocks running in stop mode. This optionally allows an internal periodic signal to wake the target MCU from stop mode.

When a host debug system is connected to the background debug pin (BKGD) and the ENBDM control bit has been set by a serial command through the background interface (or because the MCU was reset into active background mode), the oscillator is forced to remain active when the MCU enters stop mode. In this case, if a serial BACKGROUND command is issued to the MCU through the background debug interface while the CPU is in stop mode, CPU clocks will resume and the CPU will enter active background mode where other serial background commands can be processed. This ensures that a host development system can still gain access to a target MCU even if it is in stop mode.

### 10.4.2 Wait mode

The WAIT instruction enables interrupts by clearing the I bit in the CCR. It then halts the clocks to the CPU to reduce overall power consumption while the CPU is waiting for the interrupt or reset event that will wake the CPU from wait mode. When an interrupt or reset event occurs, the CPU clocks will resume and the interrupt or reset event will be processed normally.

If a serial BACKGROUND command is issued to the MCU through the background debug interface while the CPU is in wait mode, CPU clocks will resume and the CPU will enter active background mode where other serial background commands can be processed. This ensures that a host development system can still gain access to a target MCU even if it is in wait mode.

While in wait mode, there are some restrictions on which background debug commands can be used. Only the BACKGROUND command and memory-access-with-status commands are available while in wait mode. The memory-access-with-status commands do not allow memory access, but they report an error indicating that the CPU is in either stop or wait mode. The BACKGROUND command can be used to wake the CPU from wait mode and enter active background mode.

### 10.4.3 Background mode

Background instruction (BGND) is not used in normal user programs because it forces the CPU to stop processing user instructions and enter the active background mode waiting for serial background commands. The only way to resume execution of the user program is through reset or by a host debug system issuing a GO, TRACE1, or TAGGO serial command through the background debug interface.

Software-based breakpoints can be set by replacing an opcode at the desired breakpoint address with the BGND opcode. When the program reaches this breakpoint address, the CPU is forced to active background mode rather than continuing the user program.

The active background mode functions are managed through the background debug controller (BDC) in the HCS08 V6 core. The BDC provides the means for analyzing MCU operation during software development. Active background mode is entered in any of the following ways:

- When the BKGD/MS pin is low at the time the MCU exits reset.
- When a BACKGROUND command is received through the BKGD pin.

- When a BGND instruction is executed.
- When encountering a BDC breakpoint.

Background commands are of two types:

- Non-intrusive commands, defined as commands that can be issued while the user program is running. Non-intrusive commands can be issued through the BKGD pin while the MCU is in run mode; non-intrusive commands can also be executed when the MCU is in the active background mode. Non-intrusive commands include:
  - Memory access commands
  - Memory-access-with-status commands
  - BDC register access commands
  - The BACKGROUND command
- Active background commands, which can be executed only while the MCU is in active background mode. Active background commands include commands to:
  - Read or write CPU registers
  - Trace one user program instruction at a time
  - Leave active background mode to return to the user's application program (GO)

The active background mode is used to program a bootloader or user application program into the flash program memory before the MCU is operated in run mode for the first time. The active background mode can also be used to erase and reprogram the flash memory after it has been previously programmed.

### 10.4.4 Security mode

Usually HCS08 V6 MCUs are implemented with a secure operating mode. When in secure mode, external access to internal memory is restricted, so that only instructions fetched from secure memory can access secure memory.

The method by which the MCU is put into secure mode is not defined by the HCS08 V6 Core. The core receives an external input signal that, when asserted, informs to the core that the MCU is in secure mode.

While in secure mode, the core controls the following set of conditions:

1. The RAM, flash, and EEPROM arrays are considered secure memory. All registers in Direct Page or High Page are considered non-secure memory.
2. Read data is tagged as either secure or non-secure during a program read, depending on whether the read is from secure or non-secure memory.
3. A data read of secure memory returns a value of $00 when the current instruction is tagged as non-secure or the access is a BDC access.
4. A data write to secure memory is blocked and data at the target address does not change state when the current instruction is tagged as non-secure or the access is through BDC.
5. A data write to secure memory is never blocked during the stacking cycles of interrupt service routines.
6. Data accesses to either secure or non-secure memory are allowed when the current instruction is tagged as secure.
7. BDC accesses to non-secure memory are allowed.

When the device is in the non-secure mode, secure memory is treated the same as non-secure memory, and all accesses are allowed.

Table 10-2 details the security conditions for allowing or disabling a read access.

**Table 10-2. Security conditions for read access**

| Inputs conditions | | | | | Read control |
|---|---|---|---|---|---|
| Security enabled | Ram, flash or EEPROM access | Program or vector read | Current CPU instruction from secure memory | Current access is via BDC | Read access allowed |
| 0 | x | x | x | x | 1 |
| 1 | 0 | x | x | x | 1 |
| 1 | 1 | 1 | x | x | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 |

## 10.5 HCS08 V6 Opcodes

The HCS08 V6 Core has 254 one-byte opcodes and 47 two-byte opcodes, totaling 301 opcodes. For a more detailed description of the HCS08 V6 instructions please refer to the Instruction Set Summary section.

## 10.6 Special Operations

The CPU performs a few special operations that are similar to instructions but do not have opcodes like other CPU instructions. This section provides additional information about these operations.

### 10.6.1 Reset Sequence

Reset can be caused by a power-on-reset (POR) event, internal conditions such as the COP (computer operating properly) watchdog, or by assertion of an external active-low reset pin. When a reset event occurs, the CPU immediately stops whatever it is doing (the MCU does not wait for an instruction boundary before responding to a reset event).

The reset event is considered concluded when the sequence to determine whether the reset came from an internal source is done and when the reset pin is no longer asserted. At the conclusion of a reset event, the CPU performs a 6-cycle sequence to fetch the reset vector from $FFFE and $FFFF and to fill the instruction queue in preparation for execution of the first program instruction.

### 10.6.2 Interrupt Sequence

When an interrupt is requested, the CPU completes the current instruction before responding to the interrupt. At this point, the program counter is pointing at the start of the next instruction, which is where the CPU should return after servicing the interrupt. The CPU responds to an interrupt by performing the same sequence of operations as for a software interrupt (SWI) instruction, except the address used for the vector fetch is determined by the highest priority interrupt that is pending when the interrupt sequence started.

The CPU sequence for an interrupt is:

1. Store the contents of PCL, PCH, X, A, and CCR on the stack, in that order.
2. Set the I bit in the CCR.
3. Fetch the high-order half of the interrupt vector.
4. Fetch the low-order half of the interrupt vector.
5. Delay for one free bus cycle.

Fetch three bytes of program information starting at the address indicated by the interrupt vector to fill the instruction queue in preparation for execution of the first instruction in the interrupt service routine.

After the CCR contents are pushed onto the stack, the I bit in the CCR is set to prevent other interrupts while in the interrupt service routine. Although it is possible to clear the I bit with an instruction in the interrupt service routine, this would allow nesting of interrupts (which is not recommended because it leads to programs that are difficult to debug and maintain).

For compatibility with the earlier M68HC05 MCUs, the high-order half of the H:X index register pair (H) is not saved on the stack as part of the interrupt sequence. The user must use a PSHH instruction at the beginning of the service routine to save H and then use a PULH instruction just before the RTI that ends the interrupt service routine. It is not necessary to save H if you are certain that the interrupt service routine does not use any instructions or auto-increment addressing modes that might change the value of H.

The software interrupt (SWI) instruction is like a hardware interrupt except that it is not masked by the global I bit in the CCR and it is associated with an instruction opcode within the program so it is not asynchronous to program execution.

## 10.7 Instruction Set Summary

**Table 10-3. Instruction Set Summary**

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| ADC #opr8i | Add with Carry | A ← (A) + (M) + (C) | ↕ | ↕ | – | ↕ | ↕ | ↕ | IMM | A9 | ii | 2 |
| ADC opr8a | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | DIR | B9 | dd | 3 |
| ADC opr16a | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | EXT | C9 | hh ll | 4 |
| ADC oprx16,X | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | IX2 | D9 | ee ff | 4 |
| ADC oprx8,X | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | IX1 | E9 | ff | 3 |
| ADC ,X | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | IX | F9 | | 3 |
| ADC oprx16,SP | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | SP2 | 9ED9 | ee ff | 5 |
| ADC oprx8,SP | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | SP1 | 9EE9 | ff | 4 |

*Table continues on the next page...*

## Table 10-3. Instruction Set Summary (continued)

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| ADD #opr8i | Add without Carry | A ← (A) + (M) | ↕ | ↕ | – | ↕ | ↕ | ↕ | IMM | AB | ii | 2 |
| ADD opr8a | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | DIR | BB | dd | 3 |
| ADD opr16a | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | EXT | CB | hh ll | 4 |
| ADD oprx16,X | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | IX2 | DB | ee ff | 4 |
| ADD oprx8,X | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | IX1 | EB | ff | 3 |
| ADD ,X | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | IX | FB | | 3 |
| ADD oprx16,SP | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | SP2 | 9EDB | ee ff | 5 |
| ADD oprx8,SP | | | ↕ | ↕ | – | ↕ | ↕ | ↕ | SP1 | 9EEB | ff | 4 |
| AIS #opr8i | Add Immediate Value (Signed) to Stack Pointer | SP ← (SP) + (M) where M is sign extended to a 16-bit value | – | – | – | – | – | – | IMM | A7 | ii | 2 |
| AIX #opr8i | Add Immediate Value (Signed) to Index Register (H:X) | H:X ← (H:X) + (M) where M is sign extended to a 16-bit value | – | – | – | – | – | – | IMM | AF | ii | 2 |
| AND #opr8i | Logical AND | A ← (A) & (M) | 0 | – | – | ↕ | ↕ | – | IMM | A4 | ii | 2 |
| AND opr8a | | | 0 | – | – | ↕ | ↕ | – | DIR | B4 | dd | 3 |
| AND opr16a | | | 0 | – | – | ↕ | ↕ | – | EXT | C4 | hh ll | 4 |
| AND oprx16,X | | | 0 | – | – | ↕ | ↕ | – | IX2 | D4 | ee ff | 4 |
| AND oprx8,X | | | 0 | – | – | ↕ | ↕ | – | IX1 | E4 | ff | 3 |
| AND ,X | | | 0 | – | – | ↕ | ↕ | – | IX | F4 | | 3 |
| AND oprx16,SP | | | 0 | – | – | ↕ | ↕ | – | SP2 | 9ED4 | ee ff | 5 |
| AND oprx8,SP | | | 0 | – | – | ↕ | ↕ | – | SP1 | 9EE4 | ff | 4 |
| ASL opr8a | Arithmetic Shift Left (same as LSL) | C ← MSB, LSB ← 0 | ↕ | – | – | ↕ | ↕ | ↕ | DIR | 38 | dd | 5 |
| ASLA | | | ↕ | – | – | ↕ | ↕ | ↕ | INH | 48 | | 1 |
| ASLX | | | ↕ | – | – | ↕ | ↕ | ↕ | INH | 58 | | 1 |
| ASL oprx8,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX1 | 68 | ff | 5 |
| ASL ,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX | 78 | | 4 |
| ASL oprx8,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP1 | 9E68 | ff | 6 |
| ASR opr8a | Arithmetic Shift Right | MSB → MSB, LSB → C | ↕ | – | – | ↕ | ↕ | ↕ | DIR | 37 | dd | 5 |
| ASRA | | | ↕ | – | – | ↕ | ↕ | ↕ | INH | 47 | | 1 |
| ASRX | | | ↕ | – | – | ↕ | ↕ | ↕ | INH | 57 | | 1 |
| ASR oprx8,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX1 | 67 | ff | 5 |
| ASR ,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX | 77 | | 4 |
| ASR oprx8,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP1 | 9E67 | ff | 6 |
| BCC rel | Branch if Carry Bit Clear | Branch if (C) = 0 | – | – | – | – | – | – | REL | 24 | rr | 3 |
| | | | – | – | – | – | – | – | DIR (b0) | 11 | dd | 5 |

*Table continues on the next page...*

## Table 10-3. Instruction Set Summary (continued)

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| BCLR n,opr8a | Clear Bit n in Memory | Mn ← 0 | – | – | – | – | – | – | DIR (b1) | 13 | dd | 5 |
| | | | – | – | – | – | – | – | DIR (b2) | 15 | dd | 5 |
| | | | – | – | – | – | – | – | DIR (b3) | 17 | dd | 5 |
| | | | – | – | – | – | – | – | DIR (b4) | 19 | dd | |
| | | | – | – | – | – | – | – | DIR (b5) | 1B | dd | 5 |
| | | | – | – | – | – | – | – | DIR (b6) | 1D | dd | 5 |
| | | | – | – | – | – | – | – | DIR (b7) | 1F | dd | 5 |
| BCS rel | Branch if Carry Bit Set (same as BLO) | Branch if (C) = 1 | – | – | – | – | – | – | REL | 25 | rr | 3 |
| BEQ rel | Branch if Equal | Branch if (Z) = 1 | – | – | – | – | – | – | REL | 27 | rr | 3 |
| BGE rel | Branch if Greater Than or Equal To (Signed Operands) | Branch if (N ⊕ V) = 0 | – | – | – | – | – | – | REL | 90 | rr | 3 |
| BGND | Enter Active Background if ENBDM = 1 | Waits For and Processes BDM Commands Until GO, TRACE1, or TAGGO | – | – | – | – | – | – | INH | 82 | | 5+ |
| BGT rel | Branch if Greater Than (Signed Operands) | Branch if (Z) ǀ (N ⊕ V) = 0 | – | – | – | – | – | – | REL | 92 | rr | 3 |
| BHCC rel | Branch if Half Carry Bit Clear | Branch if (H) = 0 | – | – | – | – | – | – | REL | 28 | rr | 3 |
| BHCS rel | Branch if Half Carry Bit Set | Branch if (H) = 1 | – | – | – | – | – | – | REL | 29 | rr | 3 |
| BHI rel | Branch if Higher | Branch if (C) ǀ (Z) = 0 | – | – | – | – | – | – | REL | 22 | rr | 3 |
| BHS rel | Branch if Higher or Same (same as BCC) | Branch if (C) = 0 | – | – | – | – | – | – | REL | 24 | rr | 3 |
| BIH rel | Branch if IRQ Pin High | Branch if IRQ pin = 1 | – | – | – | – | – | – | REL | 2F | rr | 3 |
| BIL rel | Branch if IRQ Pin Low | Branch if IRQ pin = 0 | – | – | – | – | – | – | REL | 2E | rr | 3 |
| BIT #opr8i | Bit Test | (A) & (M), (CCR Updated but Operands Not Changed) | 0 | – | – | ↕ | ↕ | – | IMM | A5 | ii | 2 |
| BIT opr8a | | | 0 | – | – | ↕ | ↕ | – | DIR | B5 | dd | 3 |
| BIT opr16a | | | 0 | – | – | ↕ | ↕ | – | EXT | C5 | hh ll | 4 |
| BIT oprx16,X | | | 0 | – | – | ↕ | ↕ | – | IX2 | D5 | ee ff | 4 |
| BIT oprx8,X | | | 0 | – | – | ↕ | ↕ | – | IX1 | E5 | ff | 3 |
| BIT ,X | | | 0 | – | – | ↕ | ↕ | – | IX | F5 | | 3 |
| BIT oprx16,SP | | | 0 | – | – | ↕ | ↕ | – | SP2 | 9ED5 | ee ff | 5 |
| BIT oprx8,SP | | | 0 | – | – | ↕ | ↕ | – | SP1 | 9EE5 | ff | 4 |

*Table continues on the next page...*

## Table 10-3. Instruction Set Summary (continued)

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| BLE rel | Branch if Less Than or Equal To (Signed Operands) | Branch if (Z) ∣ (N ⊕ V) = 1 | – | – | – | – | – | – | REL | 93 | rr | 3 |
| BLO rel | Branch if Lower (Same as BCS) | Branch if (C) = 1 | – | – | – | – | – | – | REL | 25 | rr | 3 |
| BLS rel | Branch if Lower or Same | Branch if (C) ∣ (Z) = 1 | – | – | – | – | – | – | REL | 23 | rr | 3 |
| BLT rel | Branch if Less Than (Signed Operands) | Branch if (N ⊕ V ) = 1 | – | – | – | – | – | – | REL | 91 | rr | 3 |
| BMC rel | Branch if Interrupt Mask Clear | Branch if (I) = 0 | – | – | – | – | – | – | REL | 2C | rr | 3 |
| BMI rel | Branch if Minus | Branch if (N) = 1 | – | – | – | – | – | – | REL | 2B | rr | 3 |
| BMS rel | Branch if Interrupt Mask Set | Branch if (I) = 1 | – | – | – | – | – | – | REL | 2D | rr | 3 |
| BNE rel | Branch if Not Equal | Branch if (Z) = 0 | – | – | – | – | – | – | REL | 26 | rr | 3 |
| BPL rel | Branch if Plus | Branch if (N) = 0 | – | – | – | – | – | – | REL | 2A | rr | 3 |
| BRA rel | Branch Always | No Test | – | – | – | – | – | – | REL | 20 | rr | 3 |
| BRCLR n,opr8a,rel | Branch if Bit n in Memory Clear | Branch if (Mn) = 0 | – | – | – | – | – | ↕ | DIR (b0) | 01 | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b1) | 03 | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b2) | 05 | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b3) | 07 | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b4) | 09 | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b5) | 0B | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b6) | 0D | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b7) | 0F | dd rr | 5 |
| BRN rel | Branch Never | Uses 3 Bus Cycles | – | – | – | – | – | – | REL | 21 | rr | 3 |
| BRSET n,opr8a,rel | Branch if Bit n in Memory Set | Branch if (Mn) = 1 | – | – | – | – | – | ↕ | DIR (b0) | 00 | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b1) | 02 | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b2) | 04 | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b3) | 06 | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b4) | 08 | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b5) | 0A | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b6) | 0C | dd rr | 5 |
| | | | – | – | – | – | – | ↕ | DIR (b7) | 0E | dd rr | 5 |
| | | | – | – | – | – | – | – | DIR (b0) | 10 | dd | 5 |
| | | | – | – | – | – | – | – | DIR (b1) | 12 | dd | 5 |
| | | | – | – | – | – | – | – | DIR (b2) | 14 | dd | 5 |

*Table continues on the next page...*

## Table 10-3. Instruction Set Summary (continued)

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| BSET n,opr8a | Set Bit n in Memory | Mn ← 1 | – | – | – | – | – | – | DIR (b3) | 16 | dd | 5 |
| | | | – | – | – | – | – | – | DIR (b4) | 18 | dd | 5 |
| | | | – | – | – | – | – | – | DIR (b5) | 1A | dd | 5 |
| | | | – | – | – | – | – | – | DIR (b6) | 1C | dd | 5 |
| | | | – | – | – | – | – | – | DIR (b7) | 1E | dd | 5 |
| BSR rel | Branch to Subroutine | PC ← (PC) + 0x0002<br>push (PCL)<br>SP ← (SP) – 0x0001<br>push (PCH)<br>SP ← (SP) – 0x0001<br>PC ← (PC) + rel | – | – | – | – | – | – | REL | AD | rr | 5 |
| CBEQ opr8a,rel | Compare and Branch if Equal | Branch if (A) = (M) | – | – | – | – | – | – | DIR | 31 | dd rr | 5 |
| CBEQA #opr8i,rel | | Branch if (A) = (M) | – | – | – | – | – | – | IMM | 41 | ii rr | 4 |
| CBEQX #opr8i,rel | | Branch if (X) = (M) | – | – | – | – | – | – | IMM | 51 | ii rr | 4 |
| CBEQ oprx8,X+,rel | | Branch if (A) = (M) | – | – | – | – | – | – | IX1+ | 61 | ff rr | 5 |
| CBEQ ,X+,rel | | Branch if (A) = (M) | – | – | – | – | – | – | IX+ | 71 | rr | 5 |
| CBEQ oprx8,SP,rel | | Branch if (A) = (M) | – | – | – | – | – | – | SP1 | 9E61 | ff rr | 6 |
| CLC | Clear Carry Bit | C ← 0 | – | – | – | – | – | 0 | INH | 98 | | 1 |
| CLI | Clear Interrupt Mask Bit | I ← 0 | – | – | 0 | – | – | – | INH | 9A | | 1 |
| CLR opr8a | Clear | M ← 0x00 | 0 | – | – | 0 | 1 | – | DIR | 3F | dd | 5 |
| CLRA | | A ← 0x00 | 0 | – | – | 0 | 1 | – | INH | 4F | | 1 |
| CLRX | | X ← 0x00 | 0 | – | – | 0 | 1 | – | INH | 5F | | 1 |
| CLRH | | H ← 0x00 | 0 | – | – | 0 | 1 | – | INH | 8C | | 1 |
| CLR oprx8,X | | M ← 0x00 | 0 | – | – | 0 | 1 | – | IX1 | 6F | ff | 5 |
| CLR ,X | | M ← 0x00 | 0 | – | – | 0 | 1 | – | IX | 7F | | 4 |
| CLR oprx8,SP | | M ← 0x00 | 0 | – | – | 0 | 1 | – | SP1 | 9E6F | ff | 6 |
| CMP #opr8i | Compare Accumulator with Memory | (A) – (M); (CCR Updated But Operands Not Changed) | ↕ | – | – | ↕ | ↕ | ↕ | IMM | A1 | ii | 2 |
| CMP opr8a | | | ↕ | – | – | ↕ | ↕ | ↕ | DIR | B1 | dd | 3 |
| CMP opr16a | | | ↕ | – | – | ↕ | ↕ | ↕ | EXT | C1 | hh ll | 4 |
| CMP oprx16,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX2 | D1 | ee ff | 4 |
| CMP oprx8,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX1 | E1 | ff | 3 |
| CMP ,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX | F1 | | 3 |
| CMP oprx16,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP2 | 9ED1 | ee ff | 5 |

*Table continues on the next page...*

## Table 10-3. Instruction Set Summary (continued)

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| CMP oprx8,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP1 | 9EE1 | ff | 4 |
| COM opr8a | One's Complement | M ← (M) = 0xFF – (M) | 0 | – | – | ↕ | ↕ | 1 | DIR | 33 | dd | 5 |
| COMA | | A ← (A) = 0xFF – (A) | 0 | – | – | ↕ | ↕ | 1 | INH | 43 | | 1 |
| COMX | | X ← (X) = 0xFF – (X) | 0 | – | – | ↕ | ↕ | 1 | INH | 53 | | 1 |
| COM oprx8,X | | M ← (M) = 0xFF – (M) | 0 | – | – | ↕ | ↕ | 1 | IX1 | 63 | ff | 5 |
| COM ,X | | M ← (M) = 0xFF – (M) | 0 | – | – | ↕ | ↕ | 1 | IX | 73 | | 4 |
| COM oprx8,SP | | M ← (M) = 0xFF – (M) | 0 | – | – | ↕ | ↕ | 1 | SP1 | 9E63 | ff | 6 |
| CPHX opr16a | Compare Index Register (H:X) with Memory | (H:X) – (M:M + 0x0001); (CCR Updated But Operands Not Changed) | ↕ | – | – | ↕ | ↕ | ↕ | EXT | 3E | hh ll | 6 |
| CPHX #opr16i | | | ↕ | – | – | ↕ | ↕ | ↕ | IMM | 65 | jj kk | 3 |
| CPHX opr8a | | | ↕ | – | – | ↕ | ↕ | ↕ | DIR | 75 | dd | 5 |
| CPHX oprx8,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP1 | 9EF3 | ff | 6 |
| CPX #opr8i | Compare X (Index Register Low) with Memory | (X) – (M); (CCR Updated But Operands Not Changed) | ↕ | – | – | ↕ | ↕ | ↕ | IMM | A3 | ii | 2 |
| CPX opr8a | | | ↕ | – | – | ↕ | ↕ | ↕ | DIR | B3 | dd | 3 |
| CPX opr16a | | | ↕ | – | – | ↕ | ↕ | ↕ | EXT | C3 | hh ll | 4 |
| CPX oprx16,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX2 | D3 | ee ff | 4 |
| CPX oprx8,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX1 | E3 | ff | 3 |
| CPX ,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX | F3 | | 3 |
| CPX oprx16,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP2 | 9ED3 | ee ff | 5 |
| CPX oprx8,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP1 | 9EE3 | ff | 4 |
| DAA | Decimal Adjust Accumulator After ADD or ADC of BCD Values | $(A)_{10}$ | U | – | – | ↕ | ↕ | ↕ | INH | 72 | | 1 |
| DBNZ opr8a,rel | Decrement and Branch if Not Zero | Decrement A, X, or M Branch if (result) ≠ 0 Affects X, Not H | – | – | – | – | – | – | DIR | 3B | dd rr | 7 |
| DBNZA rel | | | – | – | – | – | – | – | INH | 4B | rr | 4 |
| DBNZX rel | | | – | – | – | – | – | – | INH | 5B | rr | 4 |
| DBNZ oprx8,X,rel | | | – | – | – | – | – | – | IX1 | 6B | ff rr | 7 |
| DBNZ ,X,rel | | | – | – | – | – | – | – | IX | 7B | rr | 6 |
| DBNZ oprx8,SP,rel | | | – | – | – | – | – | – | SP1 | 9E6B | ff rr | 8 |
| DEC opr8a | | M ← (M) – 0x01 | ↕ | – | – | ↕ | ↕ | – | DIR | 3A | dd | 5 |
| DECA | | A ← (A) – 0x01 | ↕ | – | – | ↕ | ↕ | – | INH | 4A | | 1 |
| DECX | | X ← (X) – 0x01 | ↕ | – | – | ↕ | ↕ | – | INH | 5A | | 1 |

*Table continues on the next page...*

## Table 10-3. Instruction Set Summary (continued)

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| DEC oprx8,X | Decrement | M ← (M) – 0x01 | ↕ | – | – | ↕ | ↕ | – | IX1 | 6A | ff | 5 |
| DEC ,X | | M ← (M) – 0x01 | ↕ | – | – | ↕ | ↕ | – | IX | 7A | | 4 |
| DEC oprx8,SP | | M ← (M) – 0x01 | ↕ | – | – | ↕ | ↕ | – | SP1 | 9E6A | ff | 6 |
| DIV | Divide | A ← (H:A)÷(X), H ← Remainder | – | – | – | – | ↕ | ↕ | INH | 52 | | 6 |
| EOR #opr8i | Exclusive OR Memory with Accumulator | A ← (A ⊕ M) | 0 | – | – | ↕ | ↕ | – | IMM | A8 | ii | 2 |
| EOR opr8a | | | 0 | – | – | ↕ | ↕ | – | DIR | B8 | dd | 3 |
| EOR opr16a | | | 0 | – | – | ↕ | ↕ | – | EXT | C8 | hh ll | 4 |
| EOR oprx16,X | | | 0 | – | – | ↕ | ↕ | – | IX2 | D8 | ee ff | 4 |
| EOR oprx8,X | | | 0 | – | – | ↕ | ↕ | – | IX1 | E8 | ff | 3 |
| EOR ,X | | | 0 | – | – | ↕ | ↕ | – | IX | F8 | | 3 |
| EOR oprx16,SP | | | 0 | – | – | ↕ | ↕ | – | SP2 | 9ED8 | ee ff | 5 |
| EOR oprx8,SP | | | 0 | – | – | ↕ | ↕ | – | SP1 | 9EE8 | ff | 4 |
| INC opr8a | Increment | M ← (M) + 0x01 | ↕ | – | – | ↕ | ↕ | – | DIR | 3C | dd | 5 |
| INCA | | A ← (A) + 0x01 | ↕ | – | – | ↕ | ↕ | – | INH | 4C | | 1 |
| INCX | | X ← (X) + 0x01 | ↕ | – | – | ↕ | ↕ | – | INH | 5C | | 1 |
| INC oprx8,X | | M ← (M) + 0x01 | ↕ | – | – | ↕ | ↕ | – | IX1 | 6C | ff | 5 |
| INC ,X | | M ← (M) + 0x01 | ↕ | – | – | ↕ | ↕ | – | IX | 7C | | 4 |
| INC oprx8,SP | | M ← (M) + 0x01 | ↕ | – | – | ↕ | ↕ | – | SP1 | 9E6C | ff | 6 |
| JMP opr8a | Jump | PC ← Jump Address | | | | | | | DIR | BC | dd | 3 |
| JMP opr16a | | | | | | | | | EXT | CC | hh ll | 4 |
| JMP oprx16,X | | | – | – | – | – | – | – | IX2 | DC | ee ff | 4 |
| JMP oprx8,X | | | | | | | | | IX1 | EC | ff | 3 |
| JMP ,X | | | | | | | | | IX | FC | | 3 |
| JSR opr8a | Jump to Subroutine | PC ← (PC) + n (n = 1, 2, or 3) Push (PCL)<br>SP ← (SP) – 0x0001 Push (PCH)<br>SP ← (SP) – 0x0001<br>PC ← Unconditional Address | – | – | – | – | – | – | DIR | BD | dd | 5 |
| JSR opr16a | | | – | – | – | – | – | – | EXT | CD | hh ll | 6 |
| JSR oprx16,X | | | – | – | – | – | – | – | IX2 | DD | ee ff | 6 |
| JSR oprx8,X | | | – | – | – | – | – | – | IX1 | ED | ff | 5 |
| JSR ,X | | | – | – | – | – | – | – | IX | FD | | 5 |
| LDA #opr8i | Load Accumulator from Memory | A ← (M) | 0 | – | – | ↕ | ↕ | – | IMM | A6 | ii | 2 |
| LDA opr8a | | | | | | | | | DIR | B6 | dd | 3 |
| LDA opr16a | | | | | | | | | EXT | C6 | hh ll | 4 |
| LDA oprx16,X | | | | | | | | | IX2 | D6 | ee ff | 4 |
| LDA oprx8,X | | | | | | | | | IX1 | E6 | ff | 3 |

*Table continues on the next page...*

## Table 10-3. Instruction Set Summary (continued)

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| LDA ,X | | | | | | | | | IX | F6 | | 3 |
| LDA oprx16,SP | | | | | | | | | SP2 | 9ED6 | ee ff | 5 |
| LDA oprx8,SP | | | | | | | | | SP1 | 9EE6 | ff | 4 |
| LDHX #opr16i | Load Index Register (H:X) from Memory | H:X ← (M:M + 0x0001) | 0 | – | – | ↕ | ↕ | – | IMM | 45 | jj kk | 3 |
| LDHX opr8a | | | 0 | – | – | ↕ | ↕ | – | DIR | 55 | dd | 4 |
| LDHX opr16a | | | 0 | – | – | ↕ | ↕ | – | EXT | 32 | hh ll | 5 |
| LDHX ,X | | | 0 | – | – | ↕ | ↕ | – | IX | 9EAE | | 5 |
| LDHX oprx16,X | | | 0 | – | – | ↕ | ↕ | – | IX2 | 9EBE | ee ff | 6 |
| LDHX oprx8,X | | | 0 | – | – | ↕ | ↕ | – | IX1 | 9ECE | ff | 5 |
| LDHX oprx8,SP | | | 0 | – | – | ↕ | ↕ | – | SP1 | 9EFE | ff | 5 |
| LDX #opr8i | Load X (Index Register Low) from Memory | X ← (M) | 0 | – | – | ↕ | ↕ | – | IMM | AE | ii | 2 |
| LDX opr8a | | | 0 | – | – | ↕ | ↕ | – | DIR | BE | dd | 3 |
| LDX opr16a | | | 0 | – | – | ↕ | ↕ | – | EXT | CE | hh ll | 4 |
| LDX oprx16,X | | | 0 | – | – | ↕ | ↕ | – | IX2 | DE | ee ff | 4 |
| LDX oprx8,X | | | 0 | – | – | ↕ | ↕ | – | IX1 | EE | ff | 3 |
| LDX ,X | | | 0 | – | – | ↕ | ↕ | – | IX | FE | | 3 |
| LDX oprx16,SP | | | 0 | – | – | ↕ | ↕ | – | SP2 | 9EDE | ee ff | 5 |
| LDX oprx8,SP | | | 0 | – | – | ↕ | ↕ | – | SP1 | 9EEE | ff | 4 |
| LSL opr8a | Logical Shift Left (Same as ASL) | C ← MSB, LSB ← 0 | ↕ | – | – | ↕ | ↕ | ↕ | DIR | 38 | dd | 5 |
| LSLA | | | ↕ | – | – | ↕ | ↕ | ↕ | INH | 48 | | 1 |
| LSLX | | | ↕ | – | – | ↕ | ↕ | ↕ | INH | 58 | | 1 |
| LSL oprx8,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX1 | 68 | ff | 5 |
| LSL ,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX | 78 | | 4 |
| LSL oprx8,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP1 | 9E68 | ff | 6 |
| LSR opr8a | Logical Shift Right | 0 → MSB, LSB → C | ↕ | – | – | 0 | ↕ | ↕ | DIR | 34 | dd | 5 |
| LSRA | | | ↕ | – | – | 0 | ↕ | ↕ | INH | 44 | | 1 |
| LSRX | | | ↕ | – | – | 0 | ↕ | ↕ | INH | 54 | | 1 |
| LSR oprx8,X | | | ↕ | – | – | 0 | ↕ | ↕ | IX1 | 64 | ff | 5 |
| LSR ,X | | | ↕ | – | – | 0 | ↕ | ↕ | IX | 74 | | 4 |
| LSR oprx8,SP | | | ↕ | – | – | 0 | ↕ | ↕ | SP1 | 9E64 | ff | 6 |
| MOV oprx8,opr8a | Move | $(M)_{destination} \leftarrow (M)_{source}$ | 0 | – | – | ↕ | ↕ | – | DIR/DIR | 4E | dd | 5 |
| MOV opr8a,X+ | | | 0 | – | – | ↕ | ↕ | – | DIR/IX+ | 5E | dd | 5 |

*Table continues on the next page...*

## Table 10-3. Instruction Set Summary (continued)

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| MOV #opr8i,opr8a | | H:X ← (H:X) + 0x0001 in IX+/DIR and DIR/IX+ Modes | 0 | – | – | ↕ | ↕ | – | IMM/DIR | 6E | ii | 4 |
| MOV ,X+,opr8a | | | 0 | – | – | ↕ | ↕ | – | IX+/DIR | 7E | dd | 5 |
| MUL | Unsigned multiply | X:A ← (X) × (A) | – | 0 | – | – | – | 0 | INH | 42 | | 5 |
| NEG opr8a | Negate (Two's Complement) | M ← – (M) = 0x00 – (M) | ↕ | – | – | ↕ | ↕ | ↕ | DIR | 30 | dd | 5 |
| NEGA | | A ← – (A) = 0x00 – (A) | ↕ | – | – | ↕ | ↕ | ↕ | INH | 40 | | 1 |
| NEGX | | X ← – (X) = 0x00 – (X) | ↕ | – | – | ↕ | ↕ | ↕ | INH | 50 | | 1 |
| NEG oprx8,X | | M ← – (M) = 0x00 – (M) | ↕ | – | – | ↕ | ↕ | ↕ | IX1 | 60 | ff | 5 |
| NEG ,X | | M ← – (M) = 0x00 – (M) | ↕ | – | – | ↕ | ↕ | ↕ | IX | 70 | | 4 |
| NEG oprx8,SP | | M ← – (M) = 0x00 – (M) | ↕ | – | – | ↕ | ↕ | ↕ | SP1 | 9E60 | ff | 6 |
| NOP | No Operation | Uses 1 Bus Cycle | – | – | – | – | – | – | INH | 9D | | 1 |
| NSA | Nibble Swap Accumulator | A ← (A[3:0]:A[7:4]) | – | – | – | – | – | – | INH | 62 | | 1 |
| ORA #opr8i | Inclusive OR Accumulator and Memory | A ← (A) \| (M) | 0 | – | – | ↕ | ↕ | – | IMM | AA | ii | 2 |
| ORA opr8a | | | 0 | – | – | ↕ | ↕ | – | DIR | BA | dd | 3 |
| ORA opr16a | | | 0 | – | – | ↕ | ↕ | – | EXT | CA | hh ll | 4 |
| ORA oprx16,X | | | 0 | – | – | ↕ | ↕ | – | IX2 | DA | ee ff | 4 |
| ORA oprx8,X | | | 0 | – | – | ↕ | ↕ | – | IX1 | EA | ff | 3 |
| ORA ,X | | | 0 | – | – | ↕ | ↕ | – | IX | FA | | 3 |
| ORA oprx16,SP | | | 0 | – | – | ↕ | ↕ | – | SP2 | 9EDA | ee ff | 5 |
| ORA oprx8,SP | | | 0 | – | – | ↕ | ↕ | – | SP1 | 9EEA | ff | 4 |
| PSHA | Push Accumulator onto Stack | Push (A); SP ← (SP) – 0x0001 | – | – | – | – | – | – | INH | 87 | | 2 |
| PSHH | Push H (Index Register High) onto Stack | Push (H); SP ← (SP) – 0x0001 | – | – | – | – | – | – | INH | 8B | | 2 |
| PSHX | Push X (Index Register Low) onto Stack | Push (X); SP ← (SP) – 0x0001 | – | – | – | – | – | – | INH | 89 | | 2 |
| PULA | Pull Accumulator from Stack | SP ← (SP + 0x0001); Pull (A) | – | – | – | – | – | – | INH | 86 | | 3 |
| PULH | Pull H (Index Register High) from Stack | SP ← (SP + 0x0001); Pull (H) | – | – | – | – | – | – | INH | 8A | | 3 |
| PULX | Pull X (Index Register Low) from Stack | SP ← (SP + 0x0001); Pull (X) | – | – | – | – | – | – | INH | 88 | | 3 |
| ROL opr8a | | | ↕ | – | – | ↕ | ↕ | ↕ | DIR | 39 | dd | 5 |
| ROLA | | | ↕ | – | – | ↕ | ↕ | ↕ | INH | 49 | | 1 |

*Table continues on the next page...*

## Table 10-3. Instruction Set Summary (continued)

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| ROLX | Rotate Left through Carry | C ← MSB, LSB ← C | ↕ | – | – | ↕ | ↕ | ↕ | INH | 59 | | 1 |
| ROL oprx8,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX1 | 69 | ff | 5 |
| ROL ,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX | 79 | | 4 |
| ROL oprx8,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP1 | 9E69 | ff | 6 |
| ROR opr8a | Rotate Right through Carry | LSB → C, C → MSB | ↕ | – | – | ↕ | ↕ | ↕ | DIR | 36 | dd | 5 |
| RORA | | | ↕ | – | – | ↕ | ↕ | ↕ | INH | 46 | | 1 |
| RORX | | | ↕ | – | – | ↕ | ↕ | ↕ | INH | 56 | | 1 |
| ROR oprx8,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX1 | 66 | ff | 5 |
| ROR ,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX | 76 | | 4 |
| ROR oprx8,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP1 | 9E66 | ff | 6 |
| RSP | Reset Stack Pointer | SP ← 0xFF (High Byte Not Affected) | – | – | – | – | – | – | INH | 9C | | 1 |
| RTI | Return from Interrupt | SP ← (SP) + 0x0001, Pull (CCR)<br>SP ← (SP) + 0x0001, Pull (A)<br>SP ← (SP) + 0x0001, Pull (X)<br>SP ← (SP) + 0x0001, Pull (PCH)<br>SP ← (SP) + 0x0001, Pull (PCL) | ↕ | ↕ | ↕ | ↕ | ↕ | ↕ | INH | 80 | | 9 |
| RTS | Return from Subroutine | SP ← SP + 0x0001, Pull (PCH)<br>SP ← SP + 0x0001, Pull (PCL) | – | – | – | – | – | – | INH | 81 | | 6 |
| SBC #opr8i | Subtract with Carry | A ← (A) – (M) – (C) | ↕ | – | – | ↕ | ↕ | ↕ | IMM | A2 | ii | 2 |
| SBC opr8a | | | ↕ | – | – | ↕ | ↕ | ↕ | DIR | B2 | dd | 3 |
| SBC opr16a | | | ↕ | – | – | ↕ | ↕ | ↕ | EXT | C2 | hh ll | 4 |
| SBC oprx16,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX2 | D2 | ee ff | 4 |
| SBC oprx8,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX1 | E2 | ff | 3 |
| SBC ,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX | F2 | | 3 |
| SBC oprx16,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP2 | 9ED2 | ee ff | 5 |
| SBC oprx8,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP1 | 9EE2 | ff | 4 |
| SEC | Set Carry Bit | C ← 1 | – | – | – | – | – | 1 | INH | 99 | | 1 |
| SEI | Set Interrupt Mask Bit | I ← 1 | – | – | 1 | – | – | – | INH | 9B | | 1 |

*Table continues on the next page...*

## Table 10-3. Instruction Set Summary (continued)

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| STA opr8a | Store Accumulator in Memory | M ← (A) | 0 | – | – | ↕ | ↕ | – | DIR | B7 | dd | 3 |
| STA opr16a | | | 0 | – | – | ↕ | ↕ | – | EXT | C7 | hh ll | 4 |
| STA oprx16,X | | | 0 | – | – | ↕ | ↕ | – | IX2 | D7 | ee ff | 4 |
| STA oprx8,X | | | 0 | – | – | ↕ | ↕ | – | IX1 | E7 | ff | 3 |
| STA ,X | | | 0 | – | – | ↕ | ↕ | – | IX | F7 | | 2 |
| STA oprx16,SP | | | 0 | – | – | ↕ | ↕ | – | SP2 | 9ED7 | ee ff | 5 |
| STA oprx8,SP | | | 0 | – | – | ↕ | ↕ | – | SP1 | 9EE7 | ff | 4 |
| STHX opr8a | Store H:X (Index Reg.) | (M:M + 0x0001) ← (H:X) | 0 | – | – | ↕ | ↕ | – | DIR | 35 | dd | 4 |
| STHX opr16a | | | 0 | – | – | ↕ | ↕ | – | EXT | 96 | hh ll | 5 |
| STHX oprx8,SP | | | 0 | – | – | ↕ | ↕ | – | SP1 | 9EFF | ff | 5 |
| STOP | Enable Interrupts: Stop Processing. Refer to MCU Documentation. | I bit ← 0; Stop Processing | – | – | 0 | – | – | – | INH | 8E | | 3+ |
| STX opr8a | Store X (Low 8 Bits of Index Register) in Memory | M ← (X) | 0 | – | – | ↕ | ↕ | – | DIR | BF | dd | 3 |
| STX opr16a | | | 0 | – | – | ↕ | ↕ | – | EXT | CF | hh ll | 4 |
| STX oprx16,X | | | 0 | – | – | ↕ | ↕ | – | IX2 | DF | ee ff | 4 |
| STX oprx8,X | | | 0 | – | – | ↕ | ↕ | – | IX1 | EF | ff | 3 |
| STX ,X | | | 0 | – | – | ↕ | ↕ | – | IX | FF | | 2 |
| STX oprx16,SP | | | 0 | – | – | ↕ | ↕ | – | SP2 | 9EDF | ee ff | 5 |
| STX oprx8,SP | | | 0 | – | – | ↕ | ↕ | – | SP1 | 9EEF | ff | 4 |
| SUB #opr8i | Subtract | A ← (A) – (M) | ↕ | – | – | ↕ | ↕ | ↕ | IMM | A0 | ii | 2 |
| SUB opr8a | | | ↕ | – | – | ↕ | ↕ | ↕ | DIR | B0 | dd | 3 |
| SUB opr16a | | | ↕ | – | – | ↕ | ↕ | ↕ | EXT | C0 | hh ll | 4 |
| SUB oprx16,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX2 | D0 | ee ff | 4 |
| SUB oprx8,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX1 | E0 | ff | 3 |
| SUB ,X | | | ↕ | – | – | ↕ | ↕ | ↕ | IX | F0 | | 3 |
| SUB oprx16,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP2 | 9ED0 | ee ff | 5 |
| SUB oprx8,SP | | | ↕ | – | – | ↕ | ↕ | ↕ | SP1 | 9EE0 | ff | 4 |
| | | PC ← (PC) + 0x0001<br>Push (PCL)<br>SP ← (SP) – 0x0001<br>Push (PCH)<br>SP ← (SP) – 0x0001,<br>Push (X)<br>SP ← (SP) – 0x0001<br>Push (A) | | | | | | | | | | |

*Table continues on the next page...*

## Table 10-3. Instruction Set Summary (continued)

| Source Form | Operation | Description | Effect on CCR | | | | | | Address Mode | Opcode | Operand | Bus Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | H | I | N | Z | C | | | | |
| SWI | Software Interrupt | SP ← (SP) – 0x0001<br>Push (CCR)<br>SP ← (SP) – 0x0001 I ← 1<br>PCH ← Interrupt Vector High Byte<br>PCL ← Interrupt Vector Low Byte | – | – | 1 | – | – | – | INH | 83 | | 11 |
| TAP | Transfer Accumulator to CCR | CCR ← (A) | ↕ | ↕ | ↕ | ↕ | ↕ | ↕ | INH | 84 | | 1 |
| TAX | Transfer Accumulator to X (Index Register Low) | X ← (A) | – | – | – | – | – | – | INH | 97 | | 1 |
| TPA | Transfer CCR to Accumulator | A ← (CCR) | – | – | – | – | – | – | INH | 85 | | 1 |
| TST opr8a | Test for Negative or Zero | (M) – 0x00 | 0 | – | – | ↕ | ↕ | – | DIR | 3D | dd | 4 |
| TSTA | | (A) – 0x00 | 0 | – | – | ↕ | ↕ | – | INH | 4D | | 1 |
| TSTX | | (X) – 0x00 | 0 | – | – | ↕ | ↕ | – | INH | 5D | | 1 |
| TST oprx8,X | | (M) – 0x00 | 0 | – | – | ↕ | ↕ | – | IX1 | 6D | ff | 4 |
| TST ,X | | (M) – 0x00 | 0 | – | – | ↕ | ↕ | – | IX | 7D | | 3 |
| TST oprx8,SP | | (M) – 0x00 | 0 | – | – | ↕ | ↕ | – | SP1 | 9E6D | ff | 5 |
| TSX | Transfer SP to Index Register | H:X ← (SP) + 0x0001 | – | – | – | – | – | – | INH | 95 | | 2 |
| TXA | Transfer X (Index Reg. Low) to Accumulator | A ← (X) | – | – | – | – | – | – | INH | 9F | | 1 |
| TXS | Transfer Index Register to SP | SP ← (H:X) – 0x0001 | – | – | – | – | – | – | INH | 94 | | 2 |
| WAIT | Enable Interrupts Wait for Interrupt | I bit ← 0, Halt CPU | – | – | 0 | – | – | – | INH | 8F | | 3+ |

# Chapter 11
# Keyboard Interrupts (KBI)

## 11.1 Introduction

### 11.1.1 Features

The KBI features include:

- Up to eight keyboard interrupt pins with individual pin enable bits
- Each keyboard interrupt pin is programmable as:
  - falling-edge sensitivity only
  - rising-edge sensitivity only
  - both falling-edge and low-level sensitivity
  - both rising-edge and high-level sensitivity
- One software-enabled keyboard interrupt
- Exit from low-power modes

### 11.1.2 Modes of Operation

This section defines the KBI operation in:

- Wait mode
- Stop mode
- Background debug mode

### 11.1.2.1 KBI in Wait mode

Executing the Wait instruction places the MCU into Wait mode. The KBI interrupt should be enabled (KBI_SC[KBIE] = 1), if desired, before executing the Wait instruction, allowing the KBI to continue to operate while the MCU is in Wait mode. An enabled KBI pin (KBI_PE[KBIPEn] = 1) can be used to bring the MCU out of Wait mode if the KBI interrupt is enabled (KBI_SC[KBIE] = 1).

### 11.1.2.2 KBI in Stop modes

Executing the Stop instruction places the MCU into Stop mode (when Stop is selected), where the KBI can operate asynchronously. If this is the desired behavior, the KBI interrupt must be enabled (KBI_SC[KBIE] = 1) before executing the Stop instruction, allowing the KBI to continue to operate while the MCU is in Stop mode. An enabled KBI pin (KBI_PE[KBIPEn] = 1) can be used to bring the MCU out of Stop mode if the KBI interrupt is enabled (KBI_SC[KBIE] = 1).

### 11.1.2.3 KBI in Active Background mode

When the MCU is in Active Background mode, the KBI will continue to operate normally.

## 11.1.3 Block Diagram

The block diagram for the keyboard interrupt module is shown below..

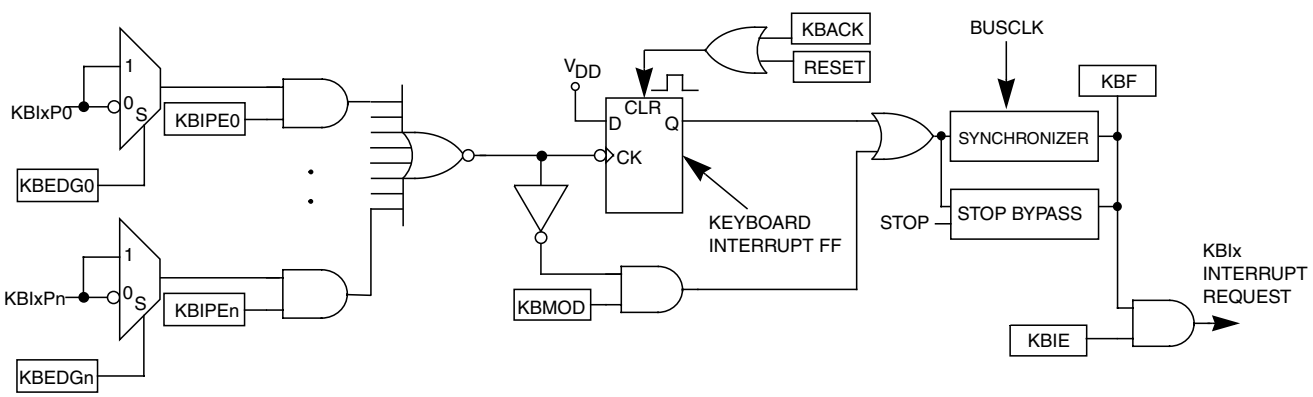



**Figure 11-1. KBI block diagram**

## 11.2 External signals description

The KBI input pins can be used to detect either falling edges, or both falling edge and low-level interrupt requests. The KBI input pins can also be used to detect either rising edges, or both rising edge and high-level interrupt requests.

The signal properties of KBI are shown in the following table:

**Table 11-1. External signals description**

| Signal | Function | I/O |
|---|---|---|
| KBIxPn | Keyboard interrupt pins | I |

## 11.3 Register definition

The KBI includes following registers:

- A pin status and control register, KBIx_SC
- A pin enable register, KBIx_PE
- An edge select register, KBIx_ES

See the direct-page register summary in the Memory chapter for the absolute address assignments for all KBIx registers. This section refers to registers and control bits only by their names.

Some MCUs may have more than one KBI, so register names include placeholder characters to identify which KBI is being referenced.

## 11.4 Memory Map and Registers

**KBI memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 3C | KBI Status and Control Register (KBI0_SC) | 8 | R/W | 00h | 11.4.1/224 |
| 307C | KBI Pin Enable Register (KBI0_PE) | 8 | R/W | 00h | 11.4.2/225 |
| 307D | KBI Edge Select Register (KBI0_ES) | 8 | R/W | 00h | 11.4.3/225 |

## 11.4.1 KBI Status and Control Register (KBIx_SC)

KBIx_SC contains the status flag and control bits, which are used to configure the KBI.

Address: 3Ch base + 0h offset = 3Ch

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | | KBF | | KBIE | KBMOD |
| Write | | | | | | KBACK | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**KBIx_SC field descriptions**

| Field | Description |
|---|---|
| 7–4<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 3<br>KBF | KBI Interrupt Flag<br><br>KBF indicates when a KBIx interrupt request is detected. Writes have no effect on KBF.<br><br>0 KBI interrupt request not detected.<br>1 KBI interrupt request detected. |
| 2<br>KBACK | KBI Acknowledge<br><br>Writing a 1 to KBACK is part of the flag clearing mechanism. |
| 1<br>KBIE | KBI Interrupt Enable<br><br>KBIE determines whether a KBI interrupt is enabled or not.<br><br>0 KBI interrupt not enabled.<br>1 KBI interrupt enabled. |
| 0<br>KBMOD | KBI Detection Mode<br><br>KBMOD (along with the KBEDG bits) controls the detection mode of the KBI interrupt pins.<br><br>0 Keyboard detects edges only.<br>1 Keyboard detects both edges and levels. |

## 11.4.2 KBI Pin Enable Register (KBIx_PE)

KBIx_PE contains the pin enable control bits.

Address: 3Ch base + 3040h offset = 307Ch

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | KBIPE | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**KBIx_PE field descriptions**

| Field | Description |
|---|---|
| KBIPE | KBI Pin Enables<br><br>Each of the KBIPEn bits enable the corresponding KBI interrupt pin.<br><br>0 Pin is not enabled as KBI interrupt.<br>1 Pin is enabled as KBI interrupt. |

## 11.4.3 KBI Edge Select Register (KBIx_ES)

KBIx_ES contains the edge select control bits.

Address: 3Ch base + 3041h offset = 307Dh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | KBEDG | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**KBIx_ES field descriptions**

| Field | Description |
|---|---|
| KBEDG | KBI Edge Selects<br><br>Each of the KBEDGn bits selects the falling edge/low-level or rising edge/high-level function of the corresponding pin.<br><br>0 Falling edge/low level.<br>1 Rising edge/high level. |

## 11.5 Functional Description

This on-chip peripheral module is called a keyboard interrupt module because originally it was designed to simplify the connection and use of row-column matrices of keyboard switches. However, these inputs are also useful as extra external interrupt inputs and as an external means of waking the MCU from stop or wait low-power modes.

The KBI module allows up to eight pins to act as additional interrupt sources. Writing to the KBIx_PE[KBIPEn] bits independently enables or disables each KBI pin. Each KBI pin can be configured as edge sensitive or edge and level sensitive based on the KBIx_SC[KBMOD] bit. Edge sensitive can be software programmed to be either falling or rising; the level can be either low or high. The polarity of the edge or edge and level sensitivity is selected using the KBIx_ES[KBEDGn] bits.

### 11.5.1 Edge-only sensitivity

Synchronous logic is used to detect edges. A falling edge is detected when an enabled keyboard interrupt (KBIx_PE[KBIPEn]=1) input signal is seen as a logic 1 (the deasserted level) during one bus cycle and then a logic 0 (the asserted level) during the next cycle. A rising edge is detected when the input signal is seen as a logic 0 (the deasserted level) during one bus cycle and then a logic 1 (the asserted level) during the next cycle.

Before the first edge is detected, all enabled keyboard interrupt input signals must be at the deasserted logic levels. After any edge is detected, all enabled keyboard interrupt input signals must return to the deasserted level before any new edge can be detected.

A valid edge on an enabled KBI pin will set KBIx_SC[KBF]. If KBIx_SC[KBIE] is set, an interrupt request will be presented to the MPU. Clearing of KBIx_SC[KBF] is accomplished by writing a 1 to KBIx_SC[KBACK].

### 11.5.2 Edge and level sensitivity

A valid edge or level on an enabled KBI pin will set KBIx_SC[KBF]. If KBIx_SC[KBIE] is set, an interrupt request will be presented to the MCU. Clearing of KBIx_SC[KBF] is accomplished by writing a 1 to KBIx_SC[KBACK], provided all enabled keyboard inputs are at their deasserted levels. KBIx_SC[KBF] will remain set if any enabled KBI pin is asserted while attempting to clear KBIx_SC[KBF] by writing a 1 to KBIx_SC[KBACK].

### 11.5.3 KBI Pullup Resistor

Each KBI pin, if enabled by KBIx_PE, can be configured via the associated I/O port pull enable register, see chapter, to use:

- an internal pullup resistor, or
- no resistor

If an internal pullup resistor is enabled for an enabled KBI pin, the associated I/O port pull select register (see I/O Port chapter) can be used to select an internal pullup resistor.

### 11.5.4 KBI initialization

When a keyboard interrupt pin is first enabled, it is possible to get a false keyboard interrupt flag. To prevent a false interrupt request during keyboard initialization, the user should do the following:

1. Mask keyboard interrupts by clearing KBIx_SC[KBIE].
2. Enable the KBI polarity by setting the appropriate KBIx_ES[KBEDGn] bits.
3. Before using internal pullup resistors, configure the associated bits in PORT_.
4. Enable the KBI pins by setting the appropriate KBIx_PE[KBIPEn] bits.
5. Write to KBIx_SC[KBACK] to clear any false interrupts.
6. Set KBIx_SC[KBIE] to enable interrupts.

# Chapter 12
# FlexTimer Module (FTM)

## 12.1 Introduction

**NOTE**

For the chip-specific implementation details of this module's instances, see the chip configuration information.

The FlexTimer module is a two to eight channel timer which supports input capture, output compare, and the generation of PWM signals to control electric motor and power management applications. The FTM time reference is a 16-bit unsigned counter.

### 12.1.1 FlexTimer philosophy

The FlexTimer is built upon a very simple timer used for many years on NXP's 8-bit microcontrollers, the HCS08 Timer PWM Module – TPM. The FlexTimer extends the functionality to meet the demands of motor control, digital lighting solutions, and power conversion, while providing low cost and backwards compatibility with the TPM module.

All of the features common with the TPM module have fully backwards compatible register assignments and the FlexTimer can use code on the same core platform without change to perform the same functions.

### 12.1.2 Features

The FTM features include:

- Selectable FTM source clock:
  - Source clock can be the system clock, the fixed frequency clock, or an external clock

  - Fixed frequency clock is an additional clock input to allow the selection of an on chip clock source other than the system clock
  - Selecting external clock connects FTM clock to a chip level input pin therefore allowing to synchronize the FTM counter with an off chip clock source
- Prescaler divide-by 1, 2, 4, 8, 16, 32, 64, or 128
- FTM has a 16-bit counter
  - It can be a free-running counter or a counter with selectable final value
  - The counting can be up or up-down
- Each channel can be configured for input capture, output compare, or edge-aligned PWM mode
- In input capture mode:
  - The capture can occur on rising edges, falling edges or both edges
- In output compare mode the output signal can be set, cleared, or toggled on match
- All channels can be configured for center-aligned PWM mode
- The generation of an interrupt per channel
- The generation of an interrupt when the counter overflows
- Backwards compatible with TPM

### 12.1.3 Modes of operation

When the MCU is in active BDM background or BDM foreground mode, the FTM temporarily suspends all counting until the MCU returns to normal user operating mode. During stop mode, all FTM input clocks are stopped, so the FTM is effectively disabled until clocks resume. During wait mode, the FTM continues to operate normally. If the FTM does not need to produce a real time reference or provide the interrupt sources needed to wake the MCU from wait mode, the power can then be saved by disabling FTM functions before entering wait mode.

### 12.1.4 Block diagram

The FTM uses one input/output (I/O) pin per channel, CHn (FTM channel (n)) where n is the channel number (0–7).

The following figure shows the FTM structure. The central component of the FTM is the 16-bit counter with programmable final value and its counting can be up or up-down.

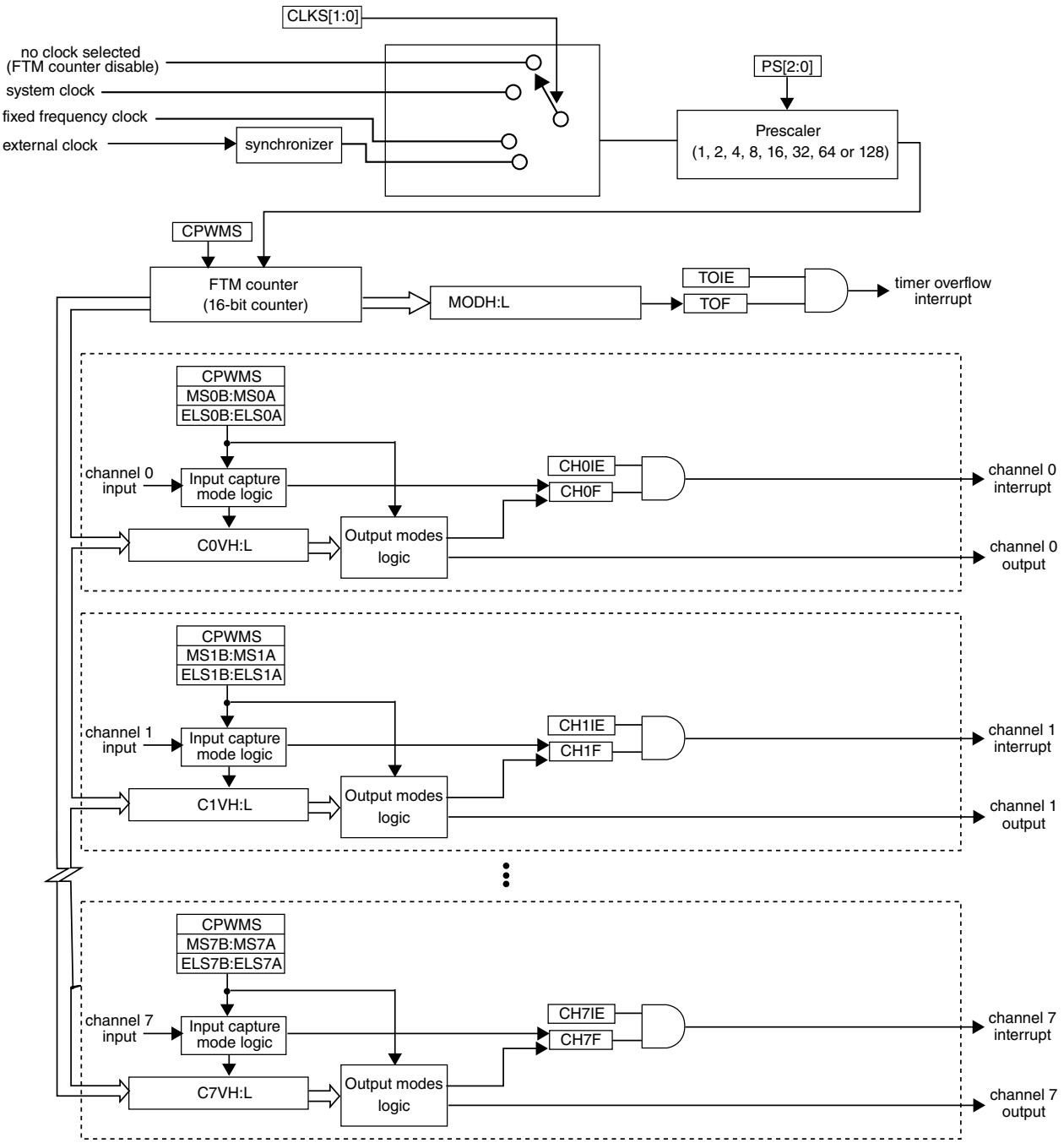



**Figure 12-1. FTM block diagram**

## 12.2 Signal description

The following table shows the user-accessible signals for the FTM.

**Table 12-1. Signal properties**

| Name | Function |
|---|---|
| EXTCLK | External clock – FTM external clock can be selected to drive the FTM counter. |
| CHn[1] | Channel (n) – I/O pin associated with FTM channel (n). |

1. n = channel number (0 to 7)

### 12.2.1 EXTCLK — FTM external clock

The external clock input signal is used as the FTM counter clock if selected by CLKS[1:0] bits in the SC register. This clock signal must not exceed 1/4 of system clock frequency. The FTM counter prescaler selection and settings are also used when an external clock is selected.

### 12.2.2 CHn — FTM channel (n) I/O pin

Each FTM channel can be configured to operate either as input or output. The direction associated with each channel, input or output, is selected according to the mode assigned for that channel.

## 12.3 Memory map and register definition

This section provides a detailed description of all FTM registers.

### 12.3.1 Module memory map

This section presents a high-level summary of the FTM registers and how they are mapped.

### 12.3.2 Register descriptions

This section consists of register descriptions in address order.

## FTM memory map

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 20 | Status and Control (FTM0_SC) | 8 | R/W | 00h | 12.3.3/234 |
| 21 | Counter High (FTM0_CNTH) | 8 | R/W | 00h | 12.3.4/235 |
| 22 | Counter Low (FTM0_CNTL) | 8 | R/W | 00h | 12.3.5/236 |
| 23 | Modulo High (FTM0_MODH) | 8 | R/W | 00h | 12.3.6/236 |
| 24 | Modulo Low (FTM0_MODL) | 8 | R/W | 00h | 12.3.7/237 |
| 25 | Channel Status and Control (FTM0_C0SC) | 8 | R/W | 00h | 12.3.8/238 |
| 26 | Channel Value High (FTM0_C0VH) | 8 | R/W | 00h | 12.3.9/239 |
| 27 | Channel Value Low (FTM0_C0VL) | 8 | R/W | 00h | 12.3.10/240 |
| 28 | Channel Status and Control (FTM0_C1SC) | 8 | R/W | 00h | 12.3.8/238 |
| 29 | Channel Value High (FTM0_C1VH) | 8 | R/W | 00h | 12.3.9/239 |
| 2A | Channel Value Low (FTM0_C1VL) | 8 | R/W | 00h | 12.3.10/240 |
| 30C0 | Status and Control (FTM2_SC) | 8 | R/W | 00h | 12.3.3/234 |
| 30C1 | Counter High (FTM2_CNTH) | 8 | R/W | 00h | 12.3.4/235 |
| 30C2 | Counter Low (FTM2_CNTL) | 8 | R/W | 00h | 12.3.5/236 |
| 30C3 | Modulo High (FTM2_MODH) | 8 | R/W | 00h | 12.3.6/236 |
| 30C4 | Modulo Low (FTM2_MODL) | 8 | R/W | 00h | 12.3.7/237 |
| 30C5 | Channel Status and Control (FTM2_C0SC) | 8 | R/W | 00h | 12.3.8/238 |
| 30C6 | Channel Value High (FTM2_C0VH) | 8 | R/W | 00h | 12.3.9/239 |
| 30C7 | Channel Value Low (FTM2_C0VL) | 8 | R/W | 00h | 12.3.10/240 |
| 30C8 | Channel Status and Control (FTM2_C1SC) | 8 | R/W | 00h | 12.3.8/238 |
| 30C9 | Channel Value High (FTM2_C1VH) | 8 | R/W | 00h | 12.3.9/239 |
| 30CA | Channel Value Low (FTM2_C1VL) | 8 | R/W | 00h | 12.3.10/240 |
| 30CB | Channel Status and Control (FTM2_C2SC) | 8 | R/W | 00h | 12.3.8/238 |
| 30CC | Channel Value High (FTM2_C2VH) | 8 | R/W | 00h | 12.3.9/239 |
| 30CD | Channel Value Low (FTM2_C2VL) | 8 | R/W | 00h | 12.3.10/240 |
| 30CE | Channel Status and Control (FTM2_C3SC) | 8 | R/W | 00h | 12.3.8/238 |

*Table continues on the next page...*

**FTM memory map (continued)**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 30CF | Channel Value High (FTM2_C3VH) | 8 | R/W | 00h | 12.3.9/239 |
| 30D0 | Channel Value Low (FTM2_C3VL) | 8 | R/W | 00h | 12.3.10/240 |
| 30D1 | Channel Status and Control (FTM2_C4SC) | 8 | R/W | 00h | 12.3.8/238 |
| 30D2 | Channel Value High (FTM2_C4VH) | 8 | R/W | 00h | 12.3.9/239 |
| 30D3 | Channel Value Low (FTM2_C4VL) | 8 | R/W | 00h | 12.3.10/240 |
| 30D4 | Channel Status and Control (FTM2_C5SC) | 8 | R/W | 00h | 12.3.8/238 |
| 30D5 | Channel Value High (FTM2_C5VH) | 8 | R/W | 00h | 12.3.9/239 |
| 30D6 | Channel Value Low (FTM2_C5VL) | 8 | R/W | 00h | 12.3.10/240 |

## 12.3.3 Status and Control (FTMx_SC)

SC contains the overflow status flag and control bits used to configure the interrupt enable, FTM configuration, clock source, and prescaler factor. These controls relate to all channels within this module.

Address: Base address + 0h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | TOF | TOIE | CPWMS | CLKS | | PS | | |
| Write | 0 | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**FTMx_SC field descriptions**

| Field | Description |
|---|---|
| 7<br>TOF | Timer Overflow Flag<br><br>Set by hardware when the FTM counter passes the value in the Counter Modulo registers. The TOF bit is cleared by reading the SC register while TOF is set and then writing a 0 to TOF bit. Writing a 1 to TOF has no effect.<br><br>If another FTM overflow occurs between the read and write operations, the write operation has no effect; therefore, TOF remains set indicating an overflow has occurred. In this case a TOF interrupt request is not lost due to the clearing sequence for a previous TOF.<br><br>0 FTM counter has not overflowed.<br>1 FTM counter has overflowed. |
| 6<br>TOIE | Timer Overflow Interrupt Enable<br><br>Enables FTM overflow interrupts.<br><br>0 Disable TOF interrupts. Use software polling.<br>1 Enable TOF interrupts. An interrupt is generated when TOF equals one. |

*Table continues on the next page...*

**FTMx_SC field descriptions (continued)**

| Field | Description |
|---|---|
| 5<br>CPWMS | Center-aligned PWM Select<br><br>Selects CPWM mode. This mode configures the FTM to operate in up-down counting mode.<br><br>0 FTM counter operates in up counting mode.<br>1 FTM counter operates in up-down counting mode. |
| 4–3<br>CLKS | Clock Source Selection<br><br>Selects one of the three FTM counter clock sources.<br><br>00 No clock selected (this in effect disables the FTM counter).<br>01 System clock<br>10 Fixed frequency clock<br>11 External clock |
| PS | Prescale Factor Selection<br><br>Selects one of 8 division factors for the clock source selected by CLKS. The new prescaler factor affects the clock source on the next system clock cycle after the new value is updated into the register bits.<br><br>000 Divide by 1<br>001 Divide by 2<br>010 Divide by 4<br>011 Divide by 8<br>100 Divide by 16<br>101 Divide by 32<br>110 Divide by 64<br>111 Divide by 128 |

## 12.3.4 Counter High (FTMx_CNTH)

The Counter registers contain the high and low bytes of the counter value. Reading either byte latches the contents of both bytes into a buffer where they remain latched until the other half is read. This allows coherent 16-bit reads in either big-endian or little-endian order which makes this more friendly to various compiler implementations. The coherency mechanism is automatically restarted by an MCU reset or any write to the Status and Control register.

Writing any value to COUNT_H or COUNT_L updates the FTM counter with its initial 16-bit value (all zeroes) and resets the read coherency mechanism, regardless of the data involved in the write.

When BDM is active, the FTM counter is frozen (this is the value that you may read); the read coherency mechanism is frozen such that the buffer latches remain in the state they were in when the BDM became active, even if one or both counter bytes are read while

BDM is active. This assures that if you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution.

Address: Base address + 1h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | COUNT_H | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**FTMx_CNTH field descriptions**

| Field | Description |
|---|---|
| COUNT_H | Counter value high byte |

## 12.3.5 Counter Low (FTMx_CNTL)

See the description for the Counter High register.

Address: Base address + 2h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | COUNT_L | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**FTMx_CNTL field descriptions**

| Field | Description |
|---|---|
| COUNT_L | Counter value low byte |

## 12.3.6 Modulo High (FTMx_MODH)

The Modulo registers contain the high and low bytes of the modulo value for the FTM counter. After the FTM counter reaches the modulo value, the overflow flag (TOF) becomes set at the next clock, and the next value of FTM counter depends on the selected counting method (Counter).

Writing to either byte latches the value into a buffer. The register is updated with the value of their write buffer according to Update of the registers with write buffers.

This write coherency mechanism may be manually reset by writing to the SC register whether BDM is active or not.

When BDM is active, this write coherency mechanism is frozen such that the buffer latches remain in the state they were in when the BDM became active, even if one or both bytes of the modulo register are written while BDM is active. Any write to the modulo register bypasses the buffer latches and directly writes to the modulo register while BDM is active.

It is recommended to initialize the FTM counter, by writing to CNTH or CNTL, before writing to the FTM modulo register to avoid confusion about when the first counter overflow will occur.

Address: Base address + 3h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | MOD_H | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**FTMx_MODH field descriptions**

| Field | Description |
|---|---|
| MOD_H | High byte of the modulo value |

## 12.3.7 Modulo Low (FTMx_MODL)

See the description for the Modulo High register.

Address: Base address + 4h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | MOD_L | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**FTMx_MODL field descriptions**

| Field | Description |
|---|---|
| MOD_L | Low byte of the modulo value |

## 12.3.8 Channel Status and Control (FTM*x*_C*n*SC)

CnSC contains the channel-interrupt-status flag and control bits used to configure the interrupt enable, channel configuration, and pin function.

**Table 12-2. Mode, edge, and level selection**

| CPWMS | MSnB:MSnA | ELSnB:ELSnA | Mode | Configuration |
|---|---|---|---|---|
| X | XX | 00 | None | Pin not used for FTM |
| 0 | 00 | 01 | Input capture | Capture on Rising Edge Only |
| | | 10 | | Capture on Falling Edge Only |
| | | 11 | | Capture on Rising or Falling Edge |
| | 01 | 01 | Output compare | Toggle Output on match |
| | | 10 | | Clear Output on match |
| | | 11 | | Set Output on match |
| | 1X | 10 | Edge-aligned PWM | High-true pulses (clear Output on match) |
| | | X1 | | Low-true pulses (set Output on match) |
| 1 | XX | 10 | Center-aligned PWM | High-true pulses (clear Output on match-up) |
| | | X1 | | Low-true pulses (set Output on match-up) |

Address: Base address + 5h offset + (3d × i), where i=0d to 5d

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | CHF | CHIE | MSB | MSA | ELSB | ELSA | 0 | 0 |
| Write | 0 | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**FTM*x*_C*n*SC field descriptions**

| Field | Description |
|---|---|
| 7<br>CHF | Channel Flag<br>Set by hardware when an event occurs on the channel. CHF is cleared by reading the CnSC register while CHnF is set and then writing a 0 to the CHF bit. Writing a 1 to CHF has no effect.<br>If another event occurs between the read and write operations, the write operation has no effect; therefore, CHF remains set indicating an event has occurred. In this case a CHF interrupt request is not lost due to the clearing sequence for a previous CHF. |

*Table continues on the next page...*

**FTMx_C*n*SC field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 No channel event has occurred.<br>1 A channel event has occurred. |
| 6<br>CHIE | Channel Interrupt Enable<br><br>Enables channel interrupts.<br><br>0 Disable channel interrupts. Use software polling.<br>1 Enable channel interrupts. |
| 5<br>MSB | Channel Mode Select<br><br>Used for further selections in the channel logic. Its functionality is dependent on the channel mode. See the table in the register description. |
| 4<br>MSA | Channel Mode Select<br><br>Used for further selections in the channel logic. Its functionality is dependent on the channel mode. See the table in the register description. |
| 3<br>ELSB | Edge or Level Select<br><br>The functionality of ELSB and ELSA depends on the channel mode. See the table in the register description. |
| 2<br>ELSA | Edge or Level Select<br><br>The functionality of ELSB and ELSA depends on the channel mode. See the table in the register description. |
| 1<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 0<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 12.3.9 Channel Value High (FTMx_C*n*VH)

These registers contain the captured FTM counter value of the input capture function or the match value for the output modes.

In input capture mode, reading a single byte in CnV latches the contents into a buffer where they remain latched until the other byte is read. This latching mechanism also resets, or becomes unlatched, when the CnSC register is written whether BDM mode is active or not. Any write to the channel registers is ignored during this mode.

When BDM is active, the read coherency mechanism is frozen such that the buffer latches remain in the state they were in when the BDM became active, even if one or both bytes of the channel value register are read while BDM is active. This ensures that if you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution. Any read of the CnV registers in BDM mode bypasses the buffer latches and returns the value of these registers and not the value of their read buffer.

In output modes, writing to CnV latches the value into a buffer. The registers are updated with the value of their write buffer according to Update of the registers with write buffers.

This write coherency mechanism may be manually reset by writing to the CnSC register whether BDM mode is active or not.

When BDM is active, the write coherency mechanism is frozen such that the buffer latches remain in the state they were in when the BDM became active even if one or both bytes of the channel value register are written while BDM is active. Any write to the CnV registers bypasses the buffer latches and writes directly to the register while BDM is active. The values written to the channel value registers while BDM is active are used in output modes operation after normal execution resumes. Writes to the channel value registers while BDM is active do not interfere with the partial completion of a coherency sequence. After the write coherency mechanism has been fully exercised, the channel value registers are updated using the buffered values while BDM was not active.

Address: Base address + 6h offset + (3d × i), where i=0d to 5d

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | VAL_H | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**FTMx_C*n*VH field descriptions**

| Field | Description |
|---|---|
| VAL_H | Channel Value High Byte<br><br>Captured FTM counter value of the input capture function or the match value for the output modes |

## 12.3.10 Channel Value Low (FTMx_C*n*VL)

See the description for the Channel Value High register.

Address: Base address + 7h offset + (3d × i), where i=0d to 5d

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | VAL_L | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**FTMx_C*n*VL field descriptions**

| Field | Description |
|---|---|
| VAL_L | Channel Value Low Byte<br><br>Captured FTM counter value of the input capture function or the match value for the output modes |

**FTMx_C*n*VL field descriptions (continued)**

| Field | Description |
|---|---|

## 12.4 Functional Description

The following sections describe the FTM features.

The notation used in this document to represent the counters and the generation of the signals is shown in the following figure.

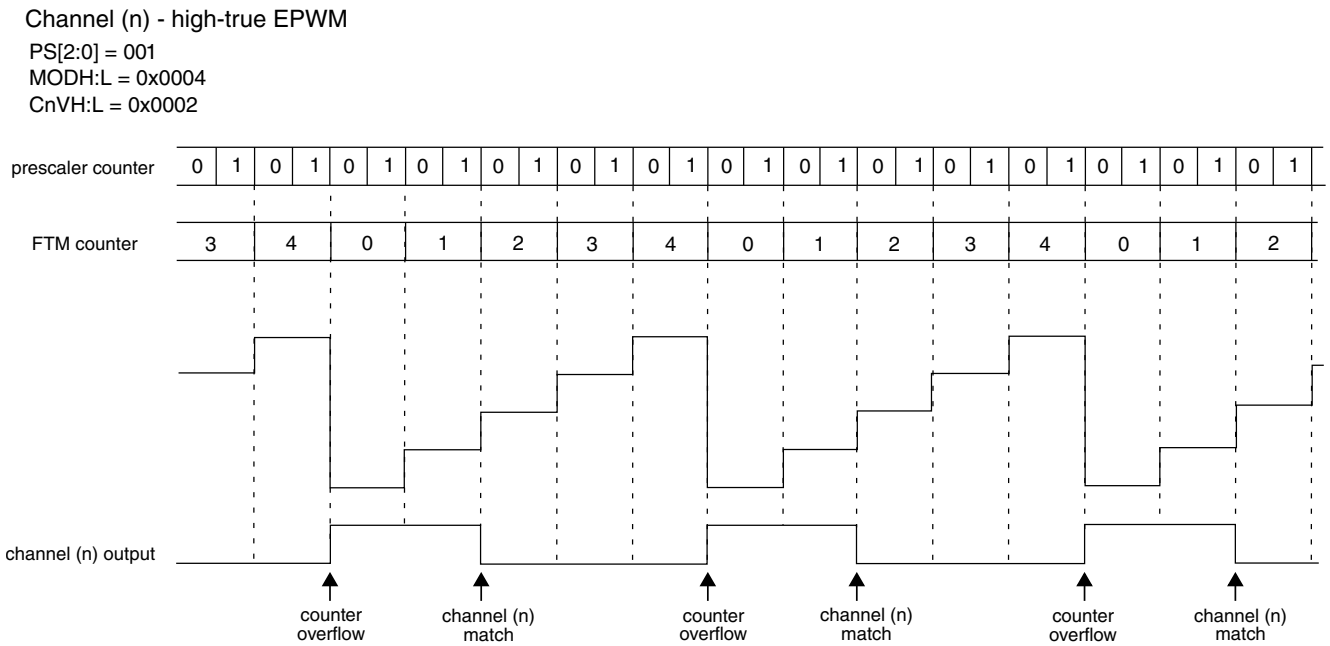



**Figure 12-2. Notation used**

### 12.4.1 Clock Source

FTM module has only one clock domain that is the system clock.

#### 12.4.1.1 Counter Clock Source

The CLKS[1:0] bits in the SC register select one of three possible clock sources for the FTM counter or disable the FTM counter. After any MCU reset, CLKS[1:0] = 0:0 so no clock source is selected.

The CLKS[1:0] bits may be read or written at any time. Disabling the FTM counter by writing 0:0 to the CLKS[1:0] bits does not affect the FTM counter value or other registers.

The fixed frequency clock is an alternative clock source for the FTM counter that allows the selection of a clock other than the system clock or an external clock. This clock input is defined by chip integration. Refer to chip specific documentation for further information. Due to FTM hardware implementation limitations, the frequency of the fixed frequency clock must not exceed the system clock frequency.

The external clock passes through a synchronizer clocked by the system clock to ensure that counter transitions are properly aligned to system clock transitions. Therefore, to meet the Nyquist criteria and account for jitter, the frequency of the external clock source must not exceed 1/4 of the system clock frequency.

## 12.4.2 Prescaler

The selected counter clock source passes through a prescaler that is a 7-bit counter. The value of the prescaler is selected by the PS[2:0] bits. The following figure shows an example of the prescaler counter and FTM counter.

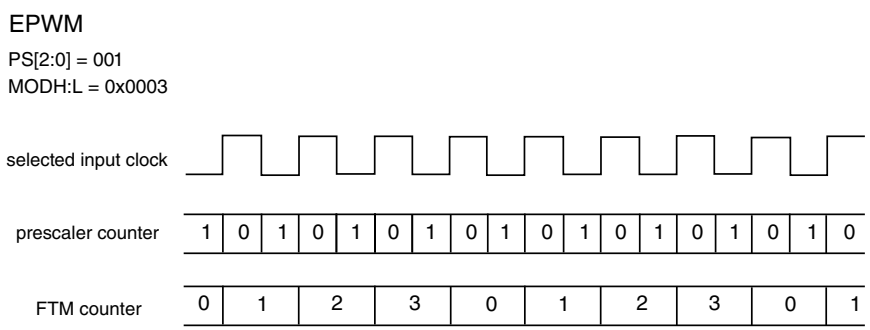



**Figure 12-3. Example of the prescaler counter**

## 12.4.3 Counter

The FTM has a 16-bit counter that is used by the channels either for input or output modes. The FTM counter clock is the selected clock divided by the prescaler (see Prescaler).

The FTM counter has these modes of operation:

- up counting (see Up counting)
- up-down counting (see Up-down counting)

### 12.4.3.1 Up counting

Up counting is selected when (CPWMS = 0).

The starting value of the count is 0x0000 and MODH:L defines the final value of the count; see the following figure. The value of 0x0000 is loaded into the FTM counter, and the counter increments until the value of MODH:L is reached, at which point the counter is reloaded with 0x0000.

The FTM period when using up counting is (MODH:L + 0x0001) × period of the FTM counter clock.

The TOF bit is set when the FTM counter changes from MODH:L to 0x0000.

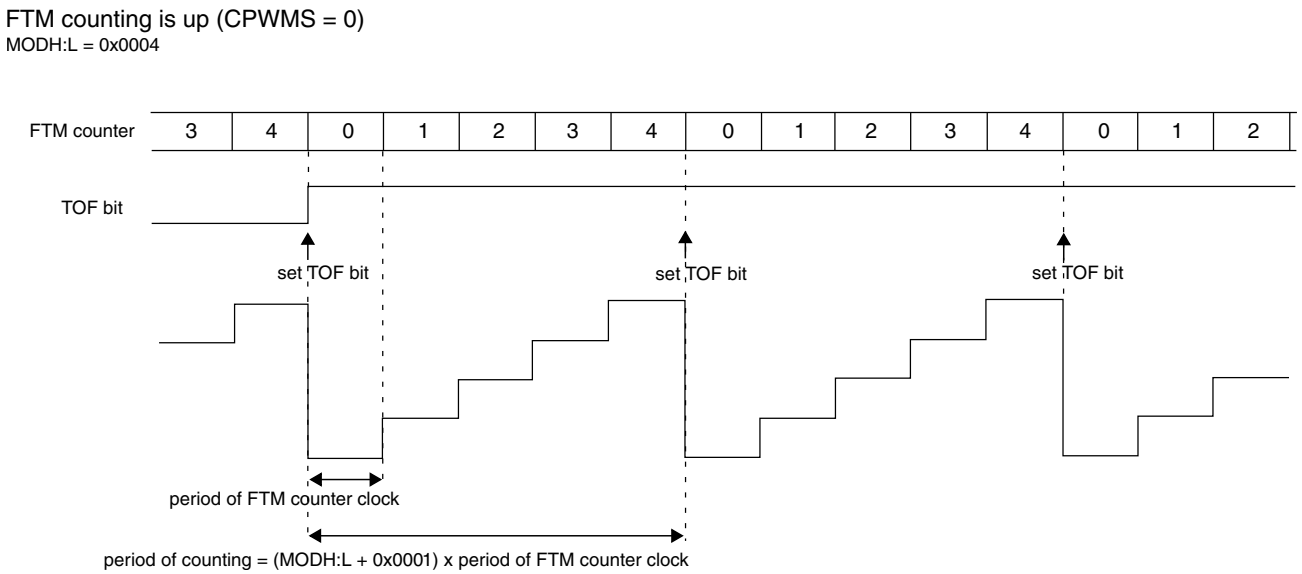



**Figure 12-4. Example of FTM up counting**

### 12.4.3.2 Up-down counting

Up-down counting is selected when (CPWMS = 1).

The starting value of the count is 0x0000 and MODH:L defines the final value of the count. The value of 0x0000 is loaded into the FTM counter, and the counter increments until the value of MODH:L is reached, at which point the counter is decremented until it returns to the value of 0x0000 and the up-down counting restarts.

The FTM period when using up-down counting is 2 × (MODH:L) × period of the FTM counter clock.

The TOF bit is set when the FTM counter changes from MODH:L to (MODH:L – 1).

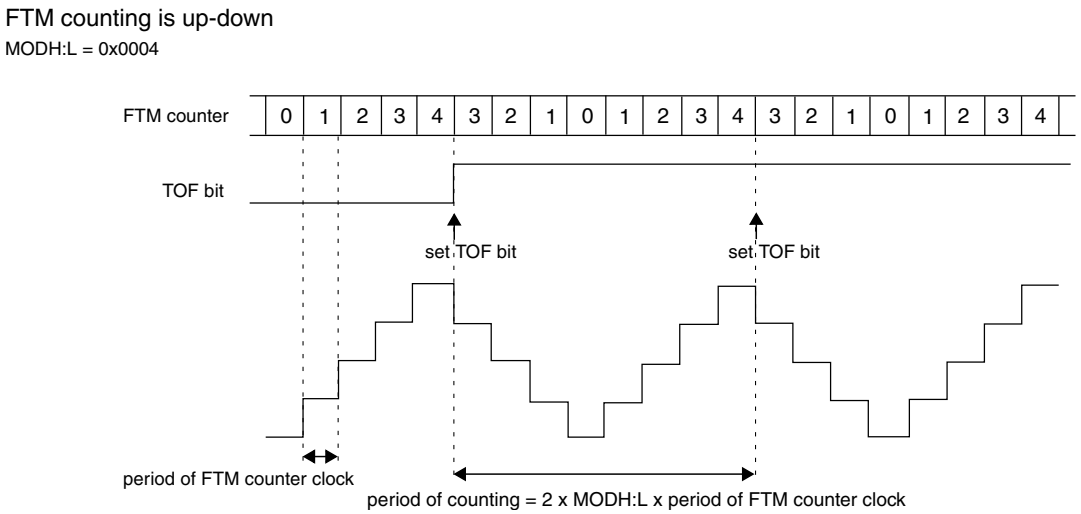



**Figure 12-5. Example of up-down counting**

### 12.4.3.3 Free running counter

If (MODH:L = 0x0000 or MODH:L = 0xFFFF), the FTM counter is a free running counter. In this case, the FTM counter runs free from 0x0000 through 0xFFFF and the TOF bit is set when the FTM counter changes from 0xFFFF to 0x0000 See the following figure.

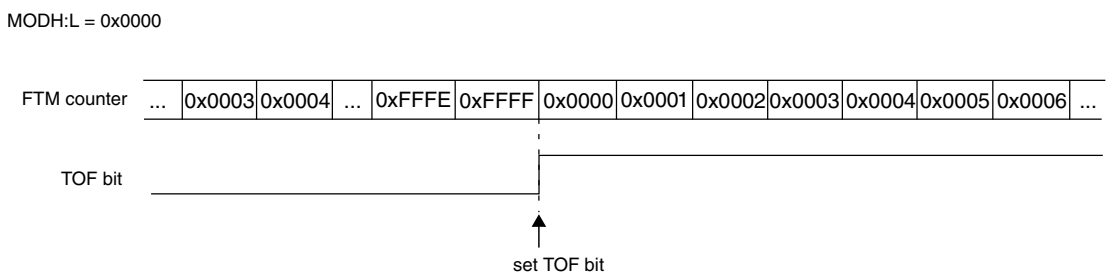



**Figure 12-6. Example when the FTM counter is a free running**

### 12.4.3.4 Counter reset

Any write to CNTH or CNTL register resets the FTM counter to the value of 0x0000 and the channels output to its initial value, except for channels in output compare mode.

## 12.4.4 Input capture mode

The input capture mode is selected when (CPWMS = 0), (MSnB:MSnA = 0:0), and (ELSnB:ELSnA ≠ 0:0).

When a selected edge occurs on the channel input, the current value of the FTM counter is captured into the CnVH:L registers. At the same time, the CHnF bit is set and the channel interrupt is generated if enabled by CHnIE = 1. See the following figure.

When a channel is configured for input capture, the CHn pin is an edge-sensitive input. ELSnB:ELSnA control bits determine which edge, falling or rising, triggers input-capture event. Note that the maximum frequency for the channel input signal to be detected correctly is system clock divided by four, which is required to meet Nyquist criteria for signal sampling.

When either half of the 16-bit capture register (CnVH:L) is read, the other half is latched into a buffer to support coherent 16-bit access in big-endian or little-endian order. This read coherency mechanism can be manually reset by writing to CnSC register.

Writes to the CnVH:L registers are ignored in input capture mode.

While in BDM, the input capture function works as configured. When a selected edge event occurs, the FTM counter value, which is frozen because of BDM, is captured into the CnVH:L registers and the CHnF bit is set.

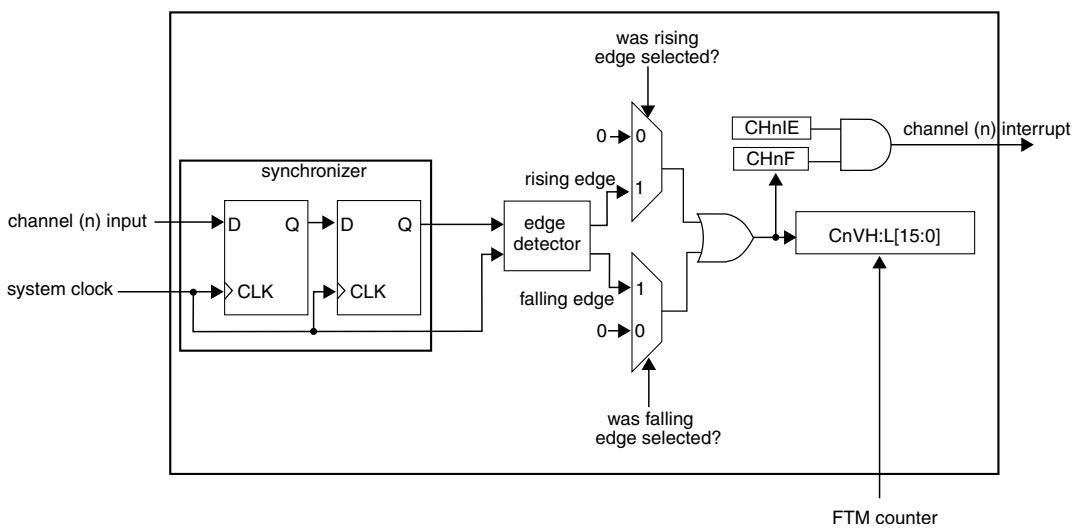



**Figure 12-7. Input capture mode**

The input signal is always delayed three rising edges of the system clock; that is, two rising edges to the synchronizer plus one more rising edge to the edge detector. In other words, the CHnF bit is set on the third rising edge of the system clock after a valid edge occurs on the channel input.

## 12.4.5 Output compare mode

The output compare mode is selected when (CPWMS = 0) and (MSnB:MSnA = 0:1).

In output compare mode, the FTM can generate timed pulses with programmable position, polarity, duration, and frequency. When the counter matches the value in the CnVH:CnVL registers of an output compare channel, the channel (n) output can be set, cleared, or toggled.

When a channel is initially configured to toggle mode, the previous value of the channel output is held until the first output compare event occurs.

The CHnF bit is set and the channel (n) interrupt is generated (if CHnIE = 1) at the channel (n) match (FTM counter = CnVH:CnVL).

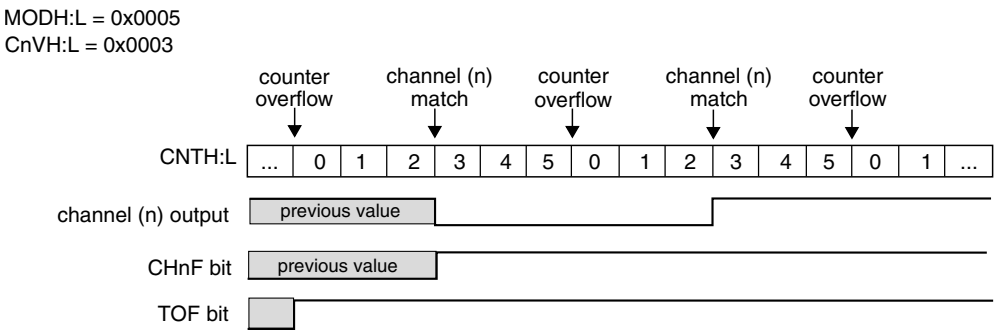



**Figure 12-8. Example of the output compare mode when the match toggles the channel output**

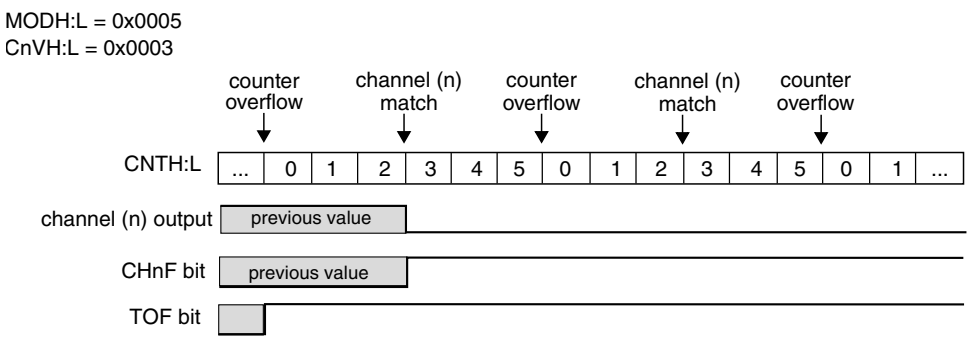



**Figure 12-9. Example of the output compare mode when the match clears the channel output**

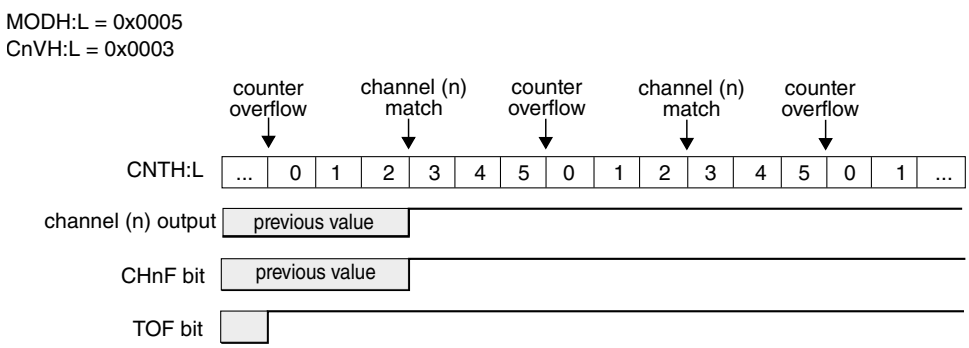



**Figure 12-10. Example of the output compare mode when the match sets the channel output**

It is possible to use the output compare mode with (ELSnB:ELSnA = 0:0). In this case, when the counter reaches the value in the CnVH:CnVL registers, the CHnF bit is set and the channel (n) interrupt is generated, if CHnIE = 1. However, the channel (n) output is not modified and controlled by FTM.

## 12.4.6 Edge-aligned PWM (EPWM) mode

The edge-aligned mode is selected when all of the following apply:

- (CPWMS = 0)
- (MSnB = 1)

The EPWM period is determined by (MODH:L + 0x0001) and the pulse width (duty cycle) is determined by (CnVH:L).

The CHnF bit is set and the channel (n) interrupt is generated (if CHnIE = 1) at the channel (n) match (FTM counter = CnVH:L), that is, at the end of the pulse width.

This type of PWM signal is called edge-aligned because the leading edges of all PWM signals are aligned with the beginning of the period, which is the same for all channels within an FTM.

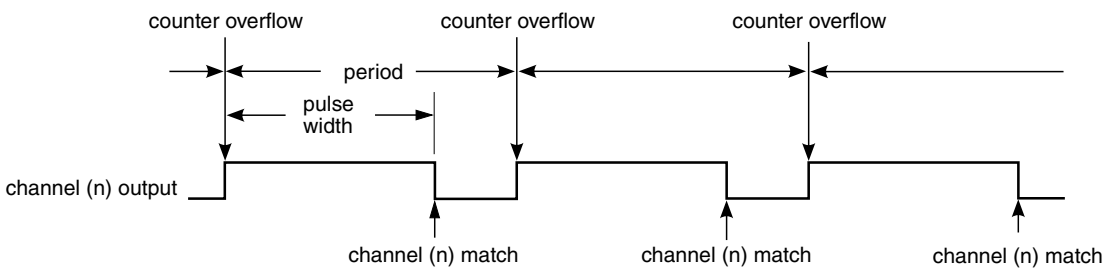



**Figure 12-11. EPWM period and pulse width with ELSnB:ELSnA = 1:0**

If (ELSnB:ELSnA = 0:0) when the counter reaches the value in the CnVH:L registers, the CHnF bit is set and the channel (n) interrupt is generated (if CHnIE = 1), however, the channel (n) output is not controlled by FTM.

If (ELSnB:ELSnA = 1:0), then the channel (n) output is forced high at the counter overflow, when the value of 0x0000 is loaded into the FTM counter. Additionally, it is forced low at the channel (n) match, when the FTM counter = CnVH:L. See the following figure.

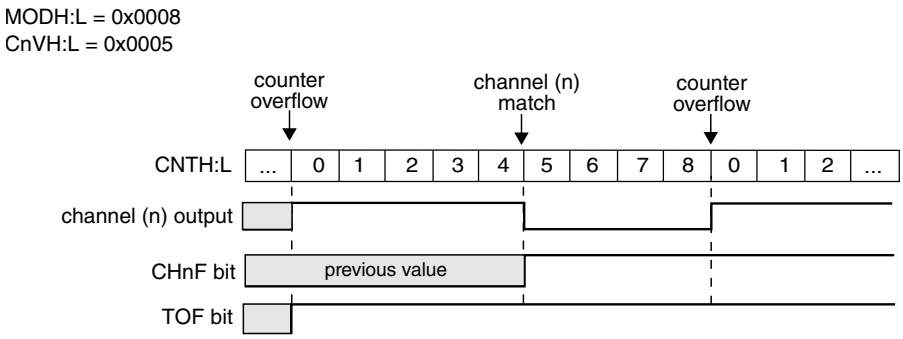



**Figure 12-12. EPWM signal with ELSnB:ELSnA = 1:0**

If (ELSnB:ELSnA = X:1), then the channel (n) output is forced low at the counter overflow, when the value of 0x0000 is loaded into the FTM counter. Additionally, it is forced high at the channel (n) match, when the FTM counter = CnVH:L. See the following figure.

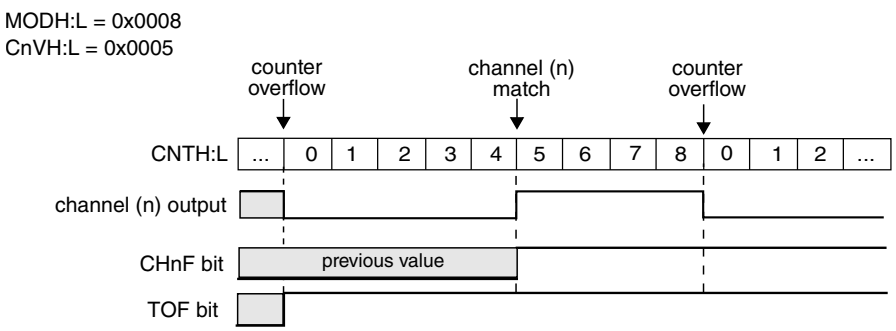



**Figure 12-13. EPWM signal with ELSnB:ELSnA = X:1**

If (CnVH:L = 0x0000), then the channel (n) output is a 0% duty cycle EPWM signal and CHnF bit is not set, even when there is the channel (n) match. If (CnVH:L > MODH:L), then the channel (n) output is a 100% duty cycle EPWM signal and CHnF bit is not set, even when there is the channel (n) match. Therefore, MODH:MODL must be less than 0xFFFF in order to get a 100% duty cycle EPWM signal.

## 12.4.7 Center-aligned PWM (CPWM) mode

The center-aligned mode is selected when:

- (CPWMS = 1)

The CPWM pulse width (duty cycle) is determined by 2 × (CnVH:L). The period is determined by 2 × (MODH:L). See the following figure. MODH:L must be kept in the range of 0x0001 to 0x7FFF because values outside this range can produce ambiguous results.

In the CPWM mode, the FTM counter counts up until it reaches MODH:L and then counts down until it reaches the value of 0x0000.

The CHnF bit is set and channel (n) interrupt is generated (if CHnIE = 1) at the channel (n) match (FTM counter = CnVH:L) when the FTM counting is down, at the begin of the pulse width, and when the FTM counting is up, at the end of the pulse width.

This type of PWM signal is called center-aligned because the pulse width centers for all channels are aligned with the value of 0x0000.

The other channel modes are not compatible with the up-down counter (CPWMS = 1). Therefore, all FTM channels must be used in CPWM mode when (CPWMS = 1).

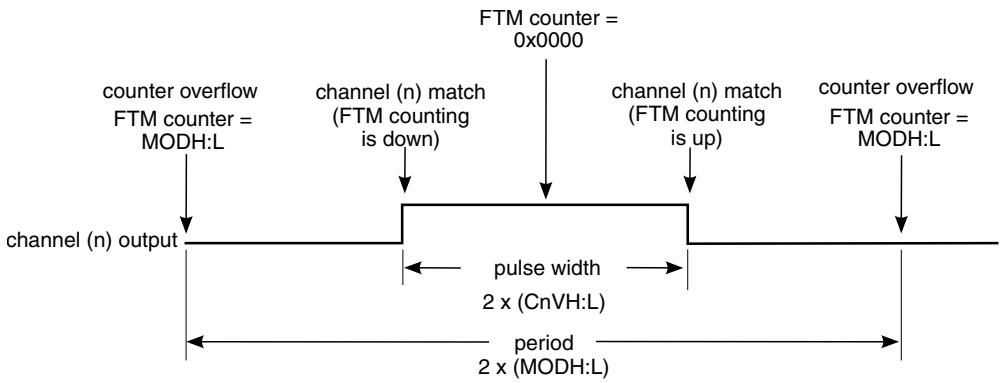



**Figure 12-14. CPWM period and pulse width with ELSnB:ELSnA = 1:0**

If (ELSnB:ELSnA = 0:0) when the counter reaches the value in the CnVH:L registers, the CHnF bit is set and the channel (n) interrupt is generated (if CHnIE = 1), however the channel (n) output is not controlled by FTM.

If (ELSnB:ELSnA = 1:0), then the channel (n) output is forced high at the channel (n) match (FTM counter = CnVH:L) when counting down, and it is forced low at the channel (n) match when counting up; see the following figure.

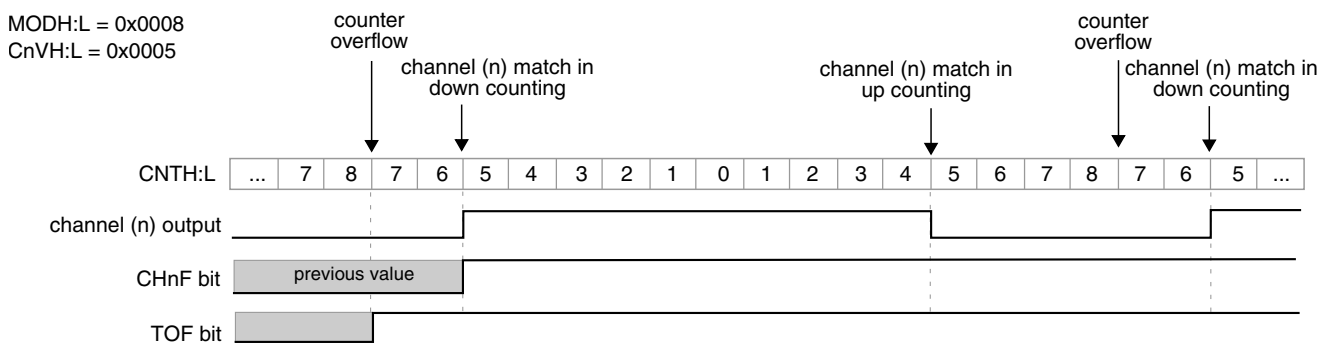



**Figure 12-15. CPWM signal with ELSnB:ELSnA = 1:0**

If (ELSnB:ELSnA = X:1), then the channel (n) output is forced low at the channel (n) match (FTM counter = CnVH:L) when counting down, and it is forced high at the channel (n) match when counting up; see the following figure.

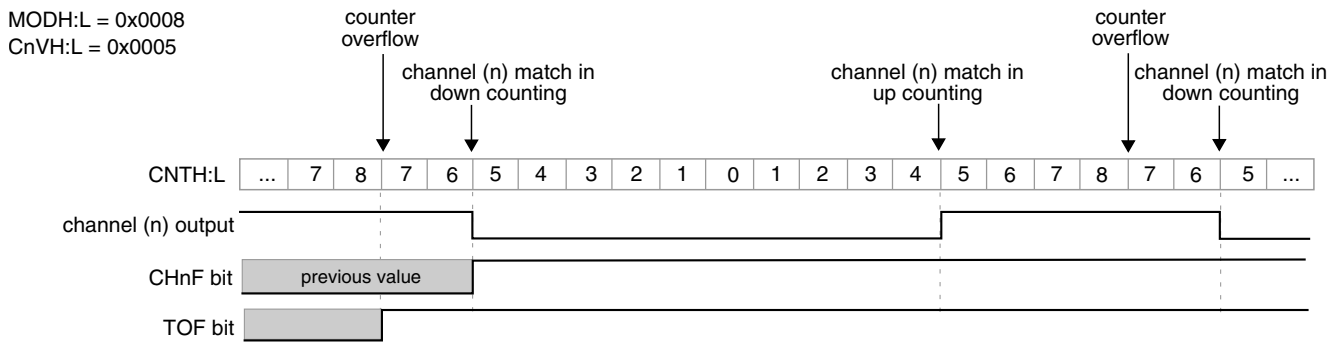



**Figure 12-16. CPWM signal with ELSnB:ELSnA = X:1**

If (CnVH:L = 0x0000) or (CnVH:L is a negative value, that is, CnVH[7] = 1) then the channel (n) output is a 0% duty cycle CPWM signal and CHnF bit is not set even when there is the channel (n) match.

If (CnVH:L is a positive value, that is, CnVH[7] = 0), (CnVH:L ≥ MODH:L), and (MODH:L ≠ 0x0000), then the channel (n) output is a 100% duty cycle CPWM signal and CHnF bit is not set even when there is the channel (n) match. This implies that the usable range of periods set by MODH:L is 0x0001 through 0x7FFE, or 0x7FFF if you do not need to generate a 100% duty cycle CPWM signal. This is not a significant limitation because the resulting period is much longer than required for normal applications.

The CPWM mode must not be used when the FTM counter is a free running counter.

## 12.4.8 Update of the registers with write buffers

This section describes the updating of registers that have write buffers.

### 12.4.8.1 MODH:L registers

If (CLKS[1:0] = 0:0), then MODH:L registers are updated when their second byte is written.

If (CLKS[1:0] ≠ 0:0), then MODH:L registers are updated according to the CPWMS bit:

- If the selected mode is not CPWM mode, then MODH:L registers are updated after both bytes have been written and the FTM counter changes from (MODH:L) to (all zeroes). If the FTM counter is a free-running counter, then this update is made when the FTM counter changes from 0xFFFF to 0x0000.
- If the selected mode is CPWM mode, then MODH:L registers are updated after both bytes have been written and the FTM counter changes from MODH:L to (MODH:L – 0x0001).

#### 12.4.8.2 CnVH:L registers

If (CLKS[1:0] = 0:0), then CnVH:L registers are updated when their second byte is written.

If (CLKS[1:0] ≠ 0:0), then CnVH:L registers are updated according to the selected mode:

- If the selected mode is output compare mode, then CnVH:L registers are updated after their second byte is written and on the next change of the FTM counter.
- If the selected mode is EPWM mode, the CnVH:L registers are updated after both bytes have been written and the FTM counter changes from MODH:L to all zeroes. If the FTM counter is a free running counter, then this update is made when the FTM counter changes from 0xFFFF to 0x0000.
- If the selected mode is CPWM mode, then CnVH:L registers are updated after both bytes have been written and the FTM counter changes from MODH:L to (MODH:L – 0x0001).

### 12.4.9 BDM mode

When BDM mode is active, the FlexTimer counter and the channels output are frozen.

However, the value of FlexTimer counter or the channels output are modified in BDM mode when:

- A write of any value to the CNTH or CNTL registers (Counter reset) resets the FTM counter to the value of 0x0000 and the channels output to their initial value, except for channels in output compare mode.

## 12.5 Reset overview

The FTM is reset whenever any chip reset occurs.

When the FTM exits from reset:

- The FTM counter and the prescaler counter are zero and are stopped (CLKS[1:0] = 0b00)
- The timer overflow interrupt is zero (Timer overflow interrupt)
- The channels interrupts are zero (Channel (n) interrupt)

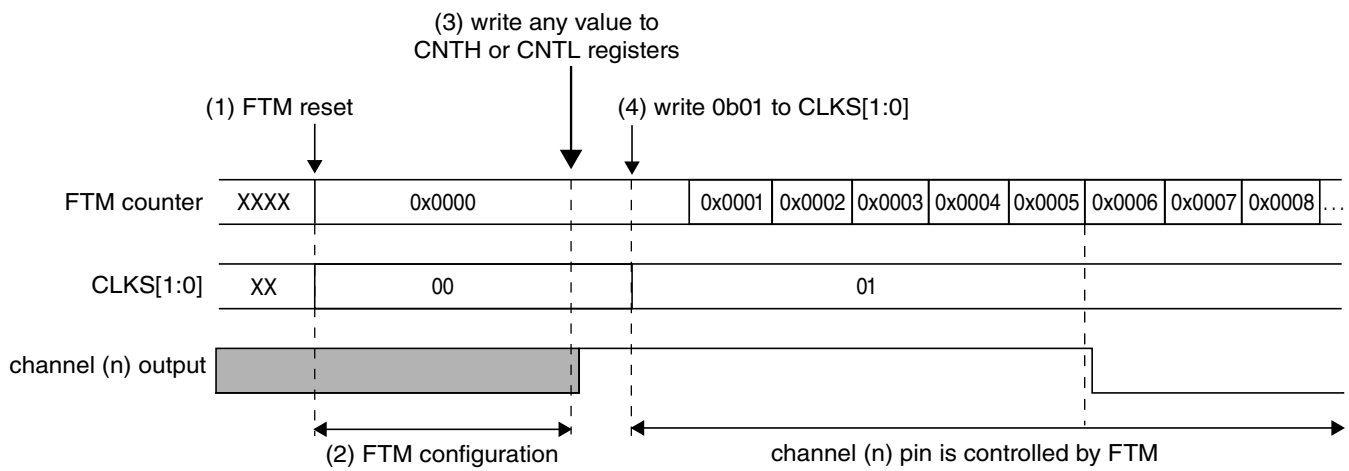
- The channels are in input capture mode (Input capture mode)
- The channels outputs are zero
- The channels pins are not controlled by FTM (ELS(n)B:ELS(n)A = 0b00). See table "Mode, Edge, and Level Selection"

The following figure shows the FTM behavior after the reset. At the reset (item 1), the FTM counter is disabled (see table "FTM Clock Source Selection"), its value is updated to zero and the pins are not controlled by FTM (table "Mode, Edge, and Level Selection").

After the reset, the FTM should be configured (item 2). It is necessary to define the FTM counter mode, the FTM counting limit (MODH:L registers value), the channels mode and CnVH:L registers value according to the channels mode.

Because of this, you should write any value to CNTH or CNTL registers (item 3). This write updates the FTM counter with the value of 0x0000 and the channels output with its initial value (except for channels in output compare mode) (Counter reset).

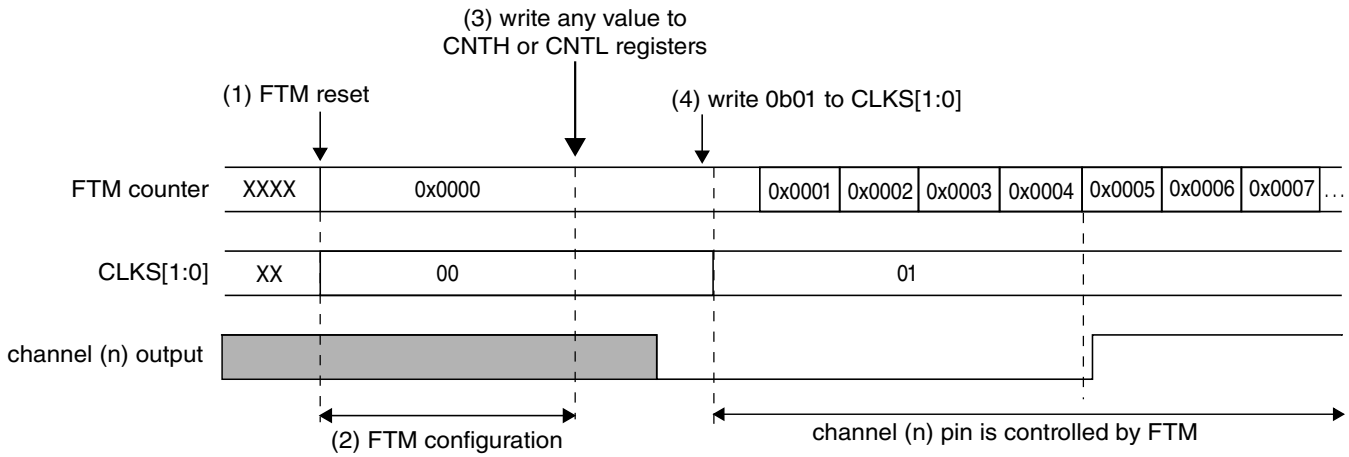
The next step is to select the FTM counter clock by the CLKS[1:0] bits (item 4). It is important to highlight that the pins are controlled only by FTM when CLKS[1:0] bits are different from zero (table "Mode, Edge, and Level Selection").




Note
– Channel (n) is in high-true EPWM mode with 0 < C(n)VH:L < MODH:L
– C(n)VH:L = 0x0005

**Figure 12-17. FTM behavior after the reset when the channel (n) is in EPWM mode**

The following figure shows an example when the channel (n) is in output compare mode and the channel (n) output is toggled when there is a match. In the output compare mode, the channel output is not updated to its initial value when there is a write to CNTH or CNTL registers (item 3).




Note
– Channel (n) is in output compare and the channel (n) output is toggled when there is a match
– C(n)VH:L = 0x0004

**Figure 12-18. FTM behavior after the reset when the channel (n) is in output compare mode**

# 12.6 FTM Interrupts

## 12.6.1 Timer overflow interrupt

The timer overflow interrupt is generated when (TOIE = 1) and (TOF = 1).

## 12.6.2 Channel (n) interrupt

The channel (n) interrupt is generated when (CHnIE = 1) and (CHnF = 1).

# Chapter 13
# 8-bit modulo timer (MTIM)

## 13.1 Introduction

The MTIM is a simple 8-bit timer with several software selectable clock sources and a programmable interrupt.

For MCUs that have more than one MTIM, the MTIMs are collectively called MTIMx. For example, MTIMx for an MCU with two MTIMs would refer to MTIM0 and MTIM1. For MCUs that have exactly one MTIM, it is always referred to as MTIM.

## 13.2 Features

Timer system features include:

- 8-bit up-counter:
  - Free-running or 8-bit modulo limit
  - Software controllable interrupt on overflow
  - Counter reset bit (TRST)
  - Counter stop bit (TSTP)
- Four software selectable clock sources for input to prescaler:
  - System bus clock - rising edge
  - Fixed frequency clock (XCLK) - rising edge
  - External clock source on the TCLK pin - rising edge
  - External clock source on the TCLK pin - falling edge
- Nine selectable clock prescale values:
  - Clock source divide by 1, 2, 4, 8, 16, 32, 64, 128, or 256

## 13.3 Modes of operation

This section defines the MTIM's operation in stop, wait, and background debug modes.

### 13.3.1 MTIM in wait mode

The MTIM continues to run in wait mode if enabled before executing the WAIT instruction. Therefore, the MTIM can be used to bring the MCU out of wait mode if the timer overflow interrupt is enabled. For lowest possible current consumption, the MTIM must be stopped by software if not needed as an interrupt source during wait mode.

### 13.3.2 MTIM in stop mode

The MTIM is disabled in stop mode, regardless of the settings before executing the STOP instruction. Therefore, the MTIM cannot be used as a wakeup source from stop modes.

If stop3 is exited with a reset, the MTIM will be put into its reset state. If stop3 is exited with an interrupt, the MTIM continues from the state it was in when stop3 was entered. If the counter was active upon entering stop3, the count will resume from the current value.

### 13.3.3 MTIM in active background mode

The MTIM suspends all counting until the microcontroller returns to normal user operating mode. Counting resumes from the suspended value as long as an MTIM reset did not occur, MTIM_SC[TRST] written to a 1 or MTIM_MOD written.

## 13.4 Block diagram

The block diagram for the modulo timer module is shown in the following figure.

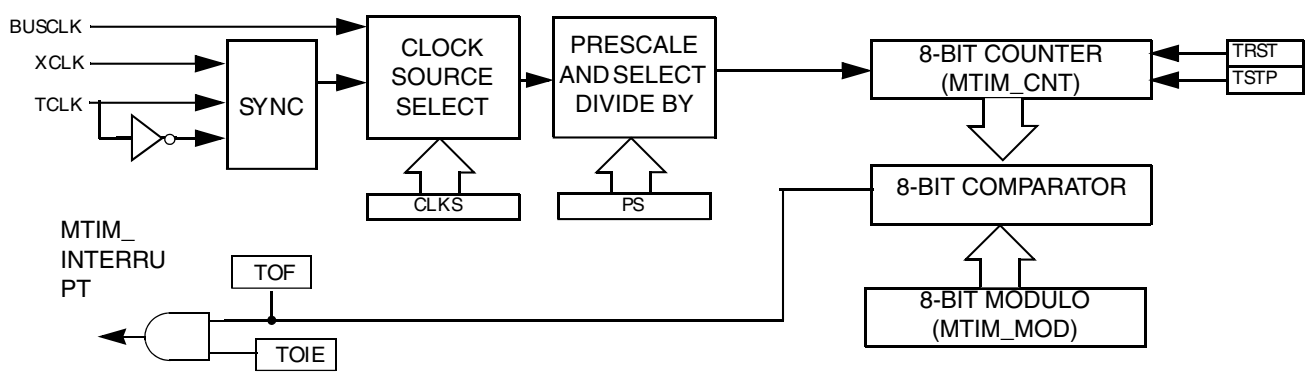



**Figure 13-1. Modulo timer (MTIM) block diagram**

## 13.5 External signal description

The MTIM includes one external signal, TCLK, used to input an external clock when selected as the MTIM clock source. The signal properties of TCLK are shown in the following table.

**Table 13-1. MTIM external signal**

| Signal | Function | I/O |
|---|---|---|
| TCLK | External clock source input into MTIM | I |

The TCLK input must be synchronized by the bus clock. Also, variations in duty cycle and clock jitter must be accommodated. Therefore, the TCLK signal must be limited to one-fourth of the bus frequency.

The TCLK pin can be muxed with a general-purpose port pin.

## 13.6 Register definition

**MTIM memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 18 | MTIM Status and Control Register (MTIM0_SC) | 8 | R/W | 10h | 13.6.1/257 |
| 19 | MTIM Clock Configuration Register (MTIM0_CLK) | 8 | R/W | 00h | 13.6.2/258 |
| 1A | MTIM Counter Register (MTIM0_CNT) | 8 | R | 00h | 13.6.3/259 |
| 1B | MTIM Modulo Register (MTIM0_MOD) | 8 | R/W | 00h | 13.6.4/260 |

### 13.6.1 MTIM Status and Control Register (MTIMx_SC)

MTIM_SC contains the overflow status flag and control bits that are used to configure the interrupt enable, reset the counter, and stop the counter.

Address: 18h base + 0h offset = 18h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | TOF | TOIE | 0 | TSTP | 0 | | | |
| Write | | | TRST | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

**MTIMx_SC field descriptions**

| Field | Description |
|---|---|
| 7<br>TOF | MTIM Overflow Flag<br><br>This bit is set when the MTIM counter register overflows to 0x00 after reaching the value in the MTIM modulo register. Clear TOF by reading the MTIM_SC register while TOF is set, then write a 0 to TOF. TOF is also cleared when TRST is written to a 1 or when any value is written to the MTIM_MOD register.<br><br>0 MTIM counter has not reached the overflow value in the MTIM modulo register.<br>1 MTIM counter has reached the overflow value in the MTIM modulo register. |
| 6<br>TOIE | MTIM Overflow Interrupt Enable<br><br>This read/write bit enables MTIM overflow interrupts. If TOIE is set, then an interrupt is generated when TOF = 1. Reset clears TOIE. Do not set TOIE if TOF = 1. Clear TOF first, then set TOIE.<br><br>0 TOF interrupts are disabled. Use software polling.<br>1 TOF interrupts are enabled. |
| 5<br>TRST | MTIM Counter Reset<br><br>When a 1 is written to this write-only bit, the MTIM counter register resets to 0x00 and TOF is cleared. Reading this bit always returns 0.<br><br>0 No effect. MTIM counter remains at current state.<br>1 MTIM counter is reset to 0x00. |
| 4<br>TSTP | MTIM Counter Stop<br><br>When set, this read/write bit stops the MTIM counter at its current value. Counting resumes from the current value when TSTP is cleared. Reset sets TSTP to prevent the MTIM from counting.<br><br>0 MTIM counter is active.<br>1 MTIM counter is stopped. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 13.6.2 MTIM Clock Configuration Register (MTIMx_CLK)

MTIM_CLK contains the clock select bits (CLKS) and the prescaler select bits (PS).

Address: 18h base + 1h offset = 19h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | CLKS | | PS | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**MTIMx_CLK field descriptions**

| Field | Description |
|---|---|
| 7–6<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

*Table continues on the next page...*

**MTIMx_CLK field descriptions (continued)**

| Field | Description |
|---|---|
| 5–4<br>CLKS | Clock Source Select<br><br>These two read/write bits select one of four different clock sources as the input to the MTIM prescaler. Changing the clock source while the counter is active does not clear the counter. The count continues with the new clock source. Reset clears CLKS to 000b.<br><br>00 Encoding 0. Bus clock (BUSCLK).<br>01 Encoding 1. Fixed-frequency clock (XCLK).<br>10 Encoding 2. External source (TCLK pin), falling edge.<br>11 Encoding 3. External source (TCLK pin), rising edge. |
| PS | Clock Source Prescaler<br><br>These four read/write bits select one of nine outputs from the 8-bit prescaler. Changing the prescaler value while the counter is active does not clear the counter. The count continues with the new prescaler value. Reset clears PS to 0000b.<br><br>0000 Encoding 0. MTIM clock source.<br>0001 Encoding 1. MTIM clock source/2.<br>0010 Encoding 2. MTIM clock source/4.<br>0011 Encoding 3. MTIM clock source/8.<br>0100 Encoding 4. MTIM clock source/16.<br>0101 Encoding 5. MTIM clock source/32.<br>0110 Encoding 6. MTIM clock source/64.<br>0111 Encoding 7. MTIM clock source/128.<br>1000 Encoding 8. MTIM clock source/256.<br>Others Default to MTIM clock source/256. |

## 13.6.3 MTIM Counter Register (MTIMx_CNT)

MTIM_CNT is the read-only value of the current MTIM count of the 8-bit counter.

Address: 18h base + 2h offset = 1Ah

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | COUNT | | | | | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**MTIMx_CNT field descriptions**

| Field | Description |
|---|---|
| COUNT | MTIM Count<br><br>These eight read-only bits contain the current value of the 8-bit counter. Writes have no effect to this register. Reset clears the count to 0x00. |

### 13.6.4 MTIM Modulo Register (MTIMx_MOD)

Address: 18h base + 3h offset = 1Bh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | MOD | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**MTIMx_MOD field descriptions**

| Field | Description |
|---|---|
| MOD | MTIM Modulo<br><br>These eight read/write bits contain the modulo value used to reset the count and set TOF. A value of 0x00 puts the MTIM in free-running mode. Writing to MTIM_MOD resets the COUNT to 0x00 and clears TOF. Reset sets the modulo to 0x00. |

## 13.7 Functional description

The MTIM consists of a main 8-bit up-counter with an 8-bit modulo register, a clock source selector, and a prescaler block with nine selectable values. The module also contains software-selectable interrupt logic.

The MTIM counter (MTIM_CNT) has three modes of operation: stopped, free-running, and modulo. Out of reset, the counter is stopped. If the counter is started without writing a new value to the modulo register, then the counter will be in free-running mode. The counter is in modulo mode when a value other than 0x00 is in the modulo register while the counter is running.

After any MCU reset, the counter is stopped and reset to 0x00, and the modulus is set to 0x00. The bus clock is selected as the default clock source and the prescale value is divide by 1. To start the MTIM in free-running mode, simply write to the MTIM status and control register (MTIM_SC), and clear the MTIM stop bit (TSTP).

Four clock sources are software selectable: the internal bus clock, the fixed frequency clock (XCLK), and an external clock on the TCLK pin, selectable as incrementing on either rising or falling edges. The MTIM clock select bits, CLKS1:CLKS0, in MTIM_CLK are used to select the desired clock source. If the counter is active (SC[TSTP] = 0) when a new clock source is selected, the counter will continue counting from the previous value using the new clock source.

Nine prescale values are software selectable: clock source divided by 1, 2, 4, 8, 16, 32, 64, 128, or 256. The prescaler select bits (CLK[PS]) in MTIM_CLK select the desired prescale value. If the counter is active (SC[TSTP] = 0) when a new prescaler value is selected, the counter will continue counting from the previous value using the new prescaler value.

The MTIM modulo register (MTIM_MOD) allows the overflow compare value to be set to any value from 0x01 to 0xFF. Reset clears the modulo value to 0x00, which results in a free running counter.

When the counter is active (SC[TSTP] = 0), the counter increments at the selected rate until the count matches the modulo value. When these values match, the counter overflows to 0x00 and continues counting. The MTIM overflow flag (SC[TOF]) is set whenever the counter overflows. The flag sets on the transition from the modulo value to 0x00. Writing to MTIM_MOD while the counter is active resets the counter to 0x00 and clears SC[TOF].

Clearing SC[TOF] is a two-step process. The first step is to read the MTIM_SC register while SC[TOF] is set. The second step is to write a 0 to SC[TOF]. If another overflow occurs between the first and second step, the clearing process is reset and SC[TOF] will remain set after the second step is performed. This will prevent the second occurrence from being missed. SC[TOF] is also cleared when a 1 is written to SC[TRST] or when any value is written to the MTIM_MOD register.

The MTIM allows for an optional interrupt to be generated whenever SC[TOF] is set. To enable the MTIM overflow interrupt, set the MTIM overflow interrupt enable bit (SC[TOIE]). SC[TOIE] must never be written to a 1 while SC[TOF] = 1. Instead, SC[TOF] must be cleared first, then the SC[TOIE] can be set to 1.

### 13.7.1 MTIM operation example

This section shows an example of the MTIM operation as the counter reaches a matching value from the modulo register.

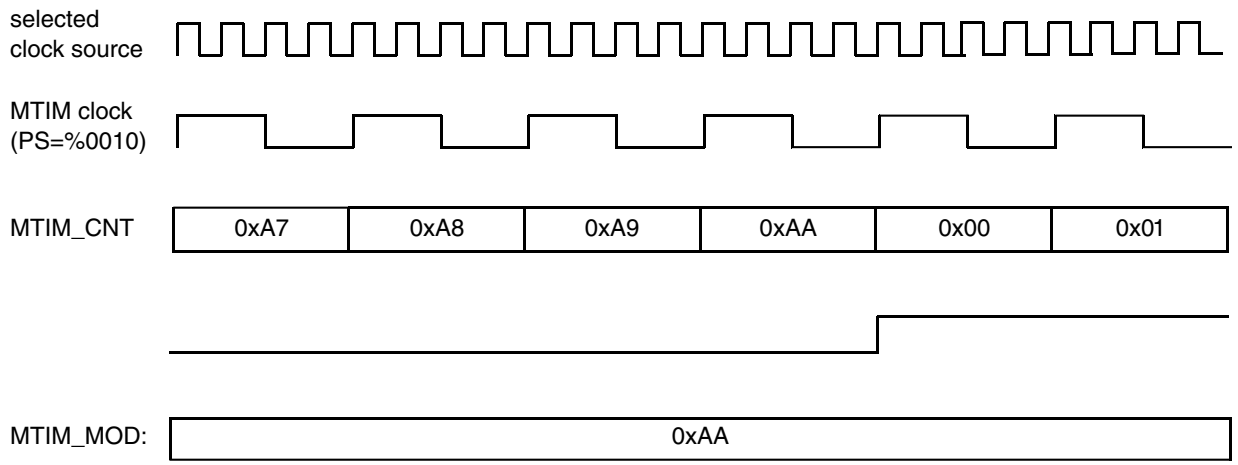



**Figure 13-2. MTIM counter overflow example**

In the above example, the selected clock source could be any of the four possible choices. The prescaler is set to CLK[PS] = 0010b or divide-by-4. The modulo value in the MTIM_MOD register is set to 0xAA. When the counter, MTIM_CNT, reaches the modulo value of 0xAA, it overflows to 0x00 and continues counting. The timer overflow flag, SC[TOF], sets when the counter value changes from 0xAA to 0x00. An MTIM overflow interrupt is generated when SC[TOF] is set, if SC[TOIE] = 1.

# Chapter 14
# Real-time counter (RTC)

## 14.1 Introduction

The real-time counter (RTC) consists of one 16-bit counter, one 16-bit comparator, several binary-based and decimal-based prescaler dividers, three clock sources, one programmable periodic interrupt, and one programmable external toggle pulse output. This module can be used for time-of-day, calendar or any task scheduling functions. It can also serve as a cyclic wake-up from low-power modes, Stop and Wait without the need of external components.

## 14.2 Features

Features of the RTC module include:

- 16-bit up-counter
  - 16-bit modulo match limit
  - Software controllable periodic interrupt on match
- Software selectable clock sources for input to prescaler with programmable 16 bit prescaler
  - OSC 32.768KHz nominal.
  - LPO (~1 kHz)
  - Bus clock

### 14.2.1 Modes of operation

This section defines the RTC operation in Stop, Wait, and Background Debug modes.

#### 14.2.1.1 Wait mode

The RTC continues to run in Wait mode if enabled before executing the WAIT instruction. Therefore, the RTC can be used to bring the MCU out of Wait mode if the real-time interrupt is enabled. For lowest possible current consumption, the RTC must be stopped by software if not needed as an interrupt source during Wait mode.

#### 14.2.1.2 Stop modes

The RTC continues to run in Stop mode if the RTC is enabled before executing the STOP instruction. Therefore, the RTC can be used to bring the MCU out of stop modes with no external components, if the real-time interrupt is enabled.

### 14.2.2 Block diagram

The block diagram for the RTC module is shown in the following figure.

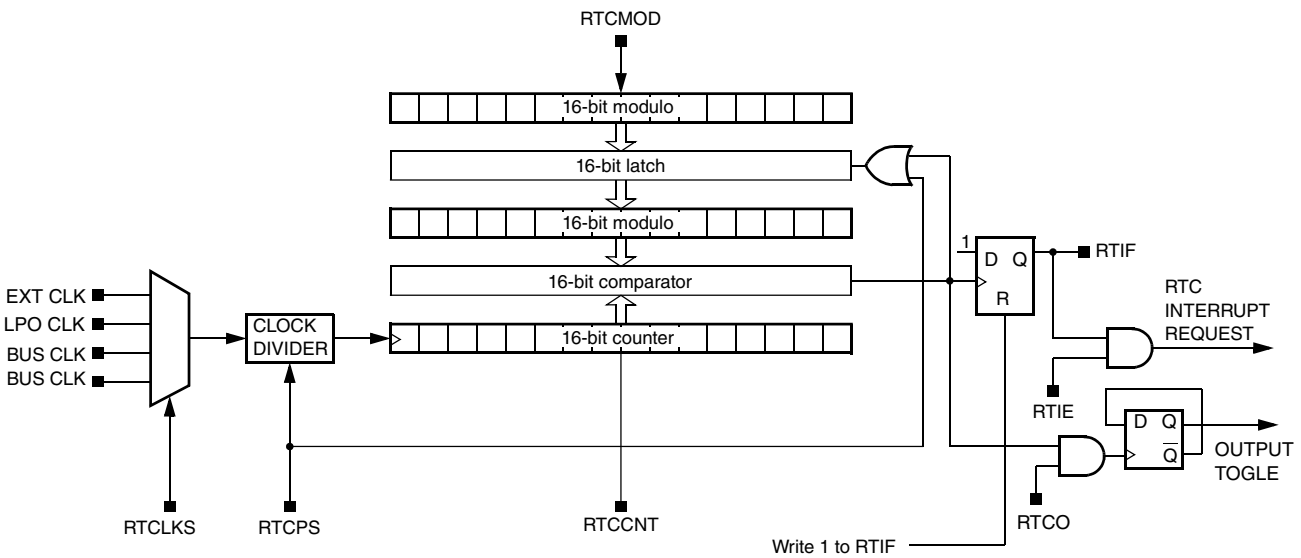



**Figure 14-1. Real-time counter (RTC) block diagram**

## 14.3 External signal description

RTCO is the output of RTC. After MCU reset, the RTC_SC1[RTCO] is set to high. When the counter overflows, the output is toggled.

## 14.4 Register definition

The RTC includes a status and control register, a 16-bit counter register, and a 16-bit modulo register.

**RTC memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 306A | RTC Status and Control Register 1 (RTC_SC1) | 8 | R/W | 00h | 14.4.1/265 |
| 306B | RTC Status and Control Register 2 (RTC_SC2) | 8 | R/W | 00h | 14.4.2/266 |
| 306C | RTC Modulo Register: High (RTC_MODH) | 8 | R/W | 00h | 14.4.3/267 |
| 306D | RTC Modulo Register: Low (RTC_MODL) | 8 | R/W | 00h | 14.4.4/267 |
| 306E | RTC Counter Register: High (RTC_CNTH) | 8 | R | 00h | 14.4.5/268 |
| 306F | RTC Counter Register: Low (RTC_CNTL) | 8 | R | 00h | 14.4.6/268 |

### 14.4.1 RTC Status and Control Register 1 (RTC_SC1)

RTC_SC1 contains the real-time interrupt status flag (RTIF), and the toggle output enable bit (RTCO).

Address: 306Ah base + 0h offset = 306Ah

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | RTIF | RTIE | 0 | RTCO | 0 | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**RTC_SC1 field descriptions**

| Field | Description |
|---|---|
| 7<br>RTIF | Real-Time Interrupt Flag<br><br>This status bit indicates the RTC counter register reached the value in the RTC modulo register. Writing a logic 0 has no effect. Writing a logic 1 clears the bit and the real-time interrupt request. Reset clears RTIF to 0.<br><br>0 RTC counter has not reached the value in the RTC modulo register.<br>1 RTC counter has reached the value in the RTC modulo register. |
| 6<br>RTIE | Real-Time Interrupt Enable<br><br>This read/write bit enables real-time interrupts. If RTIE is set, then an interrupt is generated when RTIF is set. Reset clears RTIE to 0. |

*Table continues on the next page...*

**RTC_SC1 field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Real-time interrupt requests are disabled. Use software polling.<br>1 Real-time interrupt requests are enabled. |
| 5<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 4<br>RTCO | Real-Time Counter Output<br><br>The read/write bit enables real-time to toggle output on pinout. If this bit is set, the RTCO pinout will be toggled when RTC counter overflows.<br><br>0 Real-time counter output disabled.<br>1 Real-time counter output enabled. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 14.4.2 RTC Status and Control Register 2 (RTC_SC2)

RTC_SC2 contains the clock select bits (RTCLKS) and the prescaler select bits (RTCPS).

Address: 306Ah base + 1h offset = 306Bh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | RTCLKS | | 0 | | | RTCPS | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**RTC_SC2 field descriptions**

| Field | Description |
|---|---|
| 7–6<br>RTCLKS | Real-Time Clock Source Select<br><br>These two read/write bits select the clock source input to the RTC prescaler. Changing the clock source clears the prescaler and RTCCNT counters. Reset clears RTCLKS to 00.<br><br>00 External clock source.<br>01 Real-time clock source is 1 kHz.<br>10 Bus clock.<br>11 Bus clock. |
| 5–3<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| RTCPS | Real-Time Clock Prescaler Select<br><br>These four read/write bits select binary-based or decimal-based divide-by values for the clock source. Changing the prescaler value clears the prescaler and RTCCNT counters. Reset clears RTCPS to 0000.<br><br>000 Off<br>001 If RTCLKS = x0, it is 1; if RTCLKS = x1, it is 128. |

*Table continues on the next page...*

**RTC_SC2 field descriptions (continued)**

| Field | Description |
|---|---|
| | 010 If RTCLKS = x0, it is 2; if RTCLKS = x1, it is 256.<br>011 If RTCLKS = x0, it is 4; if RTCLKS = x1, it is 512.<br>100 If RTCLKS = x0, it is 8; if RTCLKS = x1, it is 1024.<br>101 If RTCLKS = x0, it is 16; if RTCLKS = x1, it is 2048.<br>110 If RTCLKS = x0, it is 32; if RTCLKS = x1, it is 100.<br>111 If RTCLKS = x0, it is 64; if RTCLKS = x1, it is 1000. |

## 14.4.3 RTC Modulo Register: High (RTC_MODH)

RTC_MODH, together with RTC_MODL, indicates the value of the 16-bit modulo value.

Address: 306Ah base + 2h offset = 306Ch

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | MODH | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**RTC_MODH field descriptions**

| Field | Description |
|---|---|
| MODH | RTC Modulo High<br><br>These sixteen read/write bits, MODH and MODL, contain the modulo value used to reset the count to 0x0000 upon a compare match and set the RTIF status bit. A value of 0x00 of the MODH an MODL sets the RTIF bit on each rising edge of the prescaler output. Reset sets the modulo to 0x00. |

## 14.4.4 RTC Modulo Register: Low (RTC_MODL)

RTC_MODL, together with RTC_MODH, indicates the value of the 16-bit modulo value.

Address: 306Ah base + 3h offset = 306Dh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | MODL | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**RTC_MODL field descriptions**

| Field | Description |
|---|---|
| MODL | RTC Modulo Low |

**RTC_MODL field descriptions (continued)**

| Field | Description |
|---|---|
| | These sixteen read/write bits, MODH and MODL, contain the modulo value used to reset the count to 0x0000 upon a compare match and set the RTIF status bit. A value of 0x00 of the MODH an MODL sets the RTIF bit on each rising edge of the prescaler output. Reset sets the modulo to 0x00. |

## 14.4.5 RTC Counter Register: High (RTC_CNTH)

RTC_CNTH, together with RTC_CNTL, indicates the read-only value of the current RTC count of the 16-bit counter.

**NOTE**

The RTC_CNTL must be read first to lock the counter and then read RTC_CNTH to correctly read 16-bit counter.

Address: 306Ah base + 4h offset = 306Eh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | CNTH | | | | | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**RTC_CNTH field descriptions**

| Field | Description |
|---|---|
| CNTH | RTC Count High<br><br>CNTH and CNTL contain the current value of the 16-bit counter. Writes have no effect to this register. Reset or writing different values to RTCLKS and RTCPS clear the count to 0x00. |

## 14.4.6 RTC Counter Register: Low (RTC_CNTL)

RTC_CNTL, together with RTC_CNTH, indicates the read-only value of the current RTC count of the 16-bit counter.

Address: 306Ah base + 5h offset = 306Fh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | CNTL | | | | | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**RTC_CNTL field descriptions**

| Field | Description |
|---|---|
| CNTL | RTC Count Low<br><br>CNTH and CNTL contain the current value of the 16-bit counter. Writes have no effect to this register. Reset or writing different values to RTCLKS and RTCPS clear the count to 0x00. |

## 14.5 Functional description

The RTC is composed of a main 16-bit up-counter with a 16-bit modulo register, a clock source selector, and a prescaler block with binary-based and decimal-based selectable values. The module also contains software selectable interrupt logic and toggle logic for pinout.

After any MCU reset, the counter is stopped and reset to 0x0000, the modulus register is set to 0x0000, and the prescaler is off. The external oscillator clock is selected as the default clock source. To start the prescaler, write any value other than 0 to the Prescaler Select field (RTC_SC2[RTCPS]).

The clock sources are software selectable: the external oscillator (OSC), on-chip low power oscillator (LPO), and bus clock. The RTC Clock Select field (RTC_SC2[RTCLKS]) is used to select the desired clock source to the prescaler dividers. If a different value is written to RTC_SC2[RTCLKS], the prescaler and CNTH and RTC_CNTL counters are reset to 0x00.

RTC_SC2[RTCPS] and RTC_SC2[RTCLKS] select the desired divide-by value. If a different value is written to RTC_SC2[RTCPS], the prescaler and RTCCNT counters are reset to 0x00. The following table shows different prescaler period values.

**Table 14-1. Prescaler period**

| RTCPS | 32768Hz OSC clock source prescaler period (RTCLKS = 00) | LPO clock (1 kHz) source prescaler period (RTCLKS = 01) | Bus clock (8 MHz) source prescaler period (RTCLKS = 10) | Bus clock (8 MHz) source prescaler period (RTCLKS = 11) |
|---|---|---|---|---|
| 000 | Off | Off | Off | Off |
| 001 | 30.5176 µs | 128 ms | 125 ns | 16 µs |
| 010 | 61.0351 µs | 256 ms | 250 ns | 32 µs |
| 011 | 122.0703 µs | 512 ms | 500 ns | 64 µs |
| 100 | 244.1406 µs | 1024 ms | 1 µs | 128 µs |
| 101 | 488.28125 µs | 2048 ms | 2 µs | 256 µs |
| 110 | 976.5625 µs | 100 ms | 4 µs | 12.5 µs |
| 111 | 1.9531 ms | 1 s | 8 µs | 125 µs |

The RTC Modulo register (RTC_MODH and RTC_MODL) allows the compare value to be set to any value from 0x0000 to 0xFFFF. When the counter is active, the counter increments at the selected rate until the count matches the modulo value. When these values match, the counter resets to 0x0000 and continues counting. The Real-Time Interrupt Flag (RTC_SC1[RTIF]) is set whenever a match occurs. The flag sets on the transition from the modulo value to 0x0000. The modulo value written to RTC_MODH and RTC_MODL is latched until the RTC counter overflows or RTC_SC2[RTCPS] is selected nonzero.

The RTC allows for an interrupt to be generated whenever RTC_SC1[RTIF] is set. To enable the real-time interrupt, set the Real-Time Interrupt Enable field (RTC_SC1[RTIE]). RTC_SC1[RTIF] is cleared by writing a 1 to RTC_SC1[RTIF].

The RTC also allows an output to external pinout by toggling the level. RTC_SC1[RTCO] must be set to enable toggling external pinout. The level depends on the previous state of the pinout when the counter overflows if this function is active.

### 14.5.1 RTC operation example

This section shows an example of the RTC operation as the counter reaches a matching value from the modulo register.

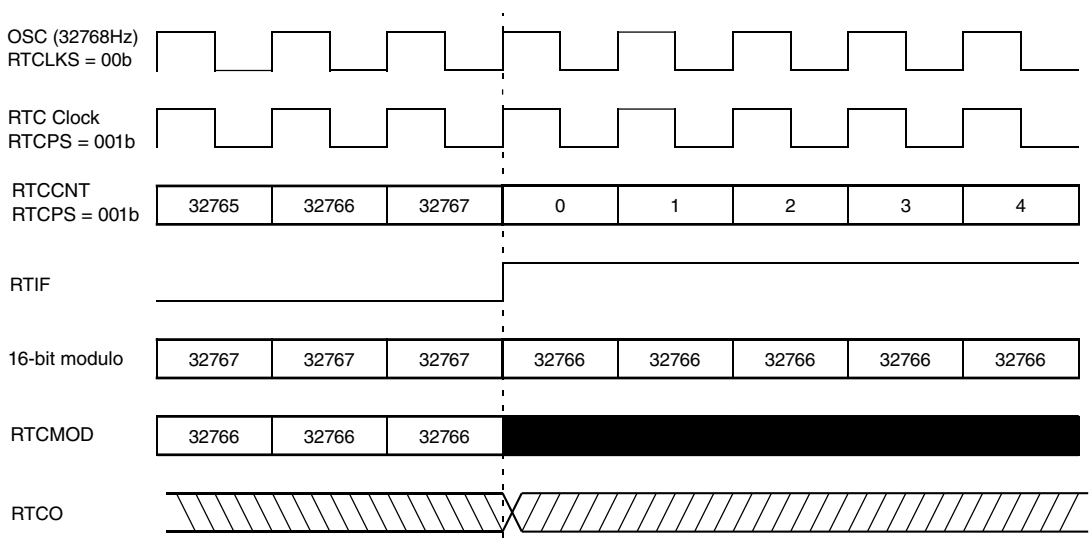



**Figure 14-2. RTC counter overflow example**

In the above example, the external clock source is selected. The prescaler is set to RTC_SC2[RTCPS] = 001b or passthrough. The actual modulo value used by 16-bit comparator is 32767, when the modulo value in the RTC_MODH and RTC_MODL registers is set to 32766. When the counter, RTC_CNTH and RTC_CNTL, reaches the

modulo value of 32767, the counter overflows to 0x00 and continues counting. The modulo value is updated by fetching from RTC_MODH and RTC_MODL registers. The real-time interrupt flag, RTC_SC1[RTIF], sets when the counter value changes from 0x7FFF to 0x0000. The RTC_SC1[RTCO] toggles as well when the RTC_SC1[RTIF] is set.

## 14.6 Initialization/application information

This section provides example code to give some basic direction to a user on how to initialize and configure the RTC module. The example software is implemented in C language.

The example below shows how to implement time of day with the RTC using the OSC clock source to achieve the lowest possible power consumption.

### Example: 14.6.1 Software calendar implementation in RTC ISR

```
/* Initialize the elapsed time counters */
Seconds = 0;
Minutes = 0;
Hours = 0;
Days=0;

/* Configure RTC to interrupt every 1 second from OSC (32.768KHz) clock source */
RTC_MOD = 511; // overflow every 32 times
RTC_SC2 = RTC_SC2_RTCPS_MASK; // external 32768 clock selected with 1/64 predivider.
RTC_SC1 = RTC_SC1_RTIF_MASK | RTC_SC1_RTIE_MASK; // interrupt cleared and enabled

/*********************************************************************
Function Name : RTC_ISR
Notes : Interrupt service routine for RTC module.
*********************************************************************/
void RTC_ISR(void)
{
/* Clears the interrupt flag, RTIF, and interrupt request */
RTC_SC1 |= RTC_SC1_RTIF_MASK;

/* RTC interrupts every 1 Second */
Seconds++;

/* 60 seconds in a minute */
if (Seconds > 59)
{
Minutes++;
Seconds = 0;
}

/* 60 minutes in an hour */
if (Minutes > 59)
{
Hours++;
Minutes = 0;
}

/* 24 hours in a day */
if (Hours > 23)
{
```

```
Days ++;
Hours = 0;
}
}
```

# Chapter 15
# Serial communications interface (SCI)

## 15.1 Introduction

### 15.1.1 Features

Features of SCI module include:

- Full-duplex, standard non-return-to-zero (NRZ) format
- Double-buffered transmitter and receiver with separate enables
- Programmable baud rates (13-bit modulo divider)
- Interrupt-driven or polled operation:
  - Transmit data register empty and transmission complete
  - Receive data register full
  - Receive overrun, parity error, framing error, and noise error
  - Idle receiver detect
  - Active edge on receive pin
  - Break detect supporting LIN
- Hardware parity generation and checking
- Programmable 8-bit or 9-bit character length
- Programmable 1-bit or 2-bit stop bits
- Receiver wakeup by idle-line or address-mark
- Optional 13-bit break character generation / 11-bit break character detection
- Selectable transmitter output polarity

### 15.1.2 Modes of operation

See Section Functional description for details concerning SCI operation in these modes:

- 8- and 9-bit data modes
- Stop mode operation

- Loop mode
- Single-wire mode

## 15.1.3 Block diagram

The following figure shows the transmitter portion of the SCI.

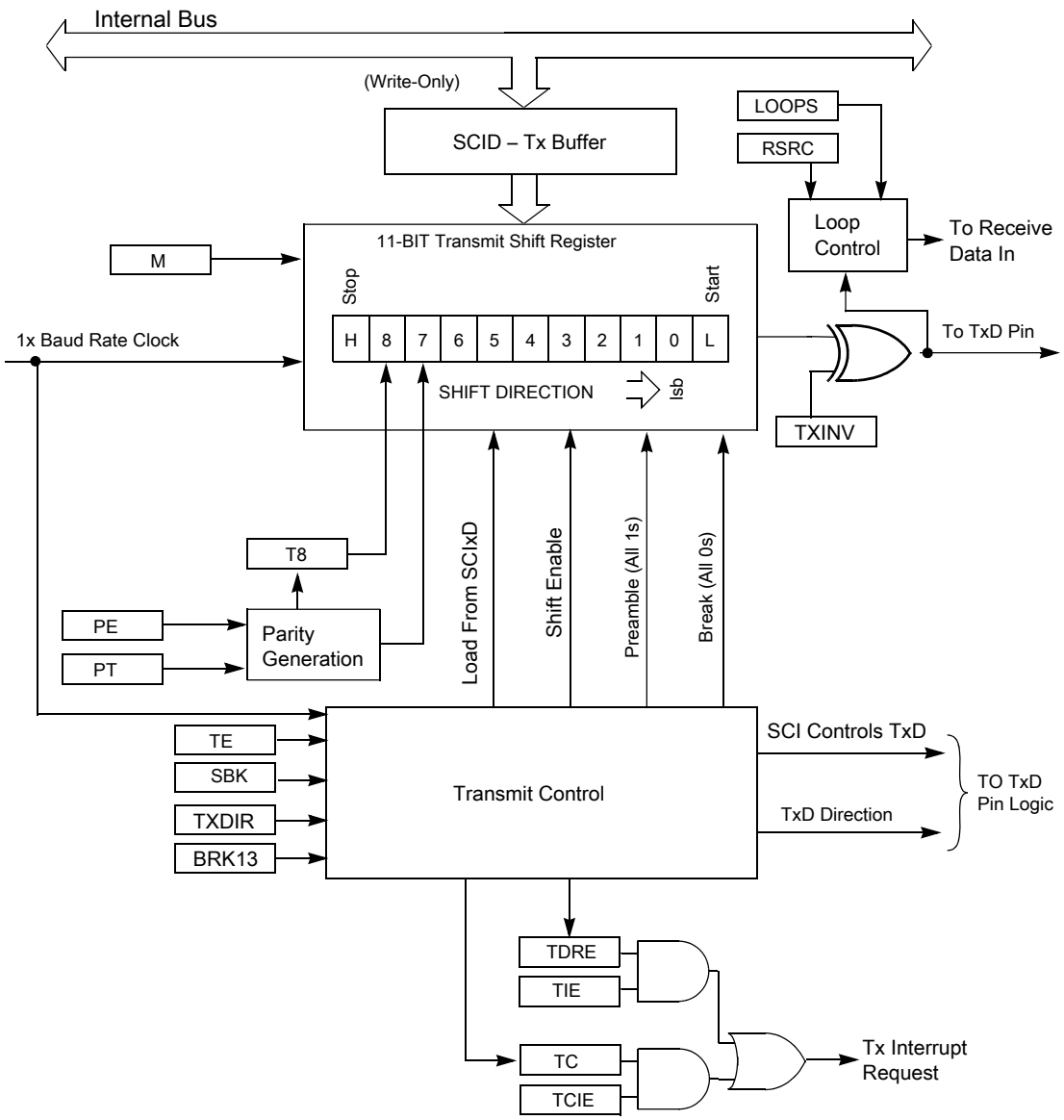



**Figure 15-1. SCI transmitter block diagram**

The following figure shows the receiver portion of the SCI.

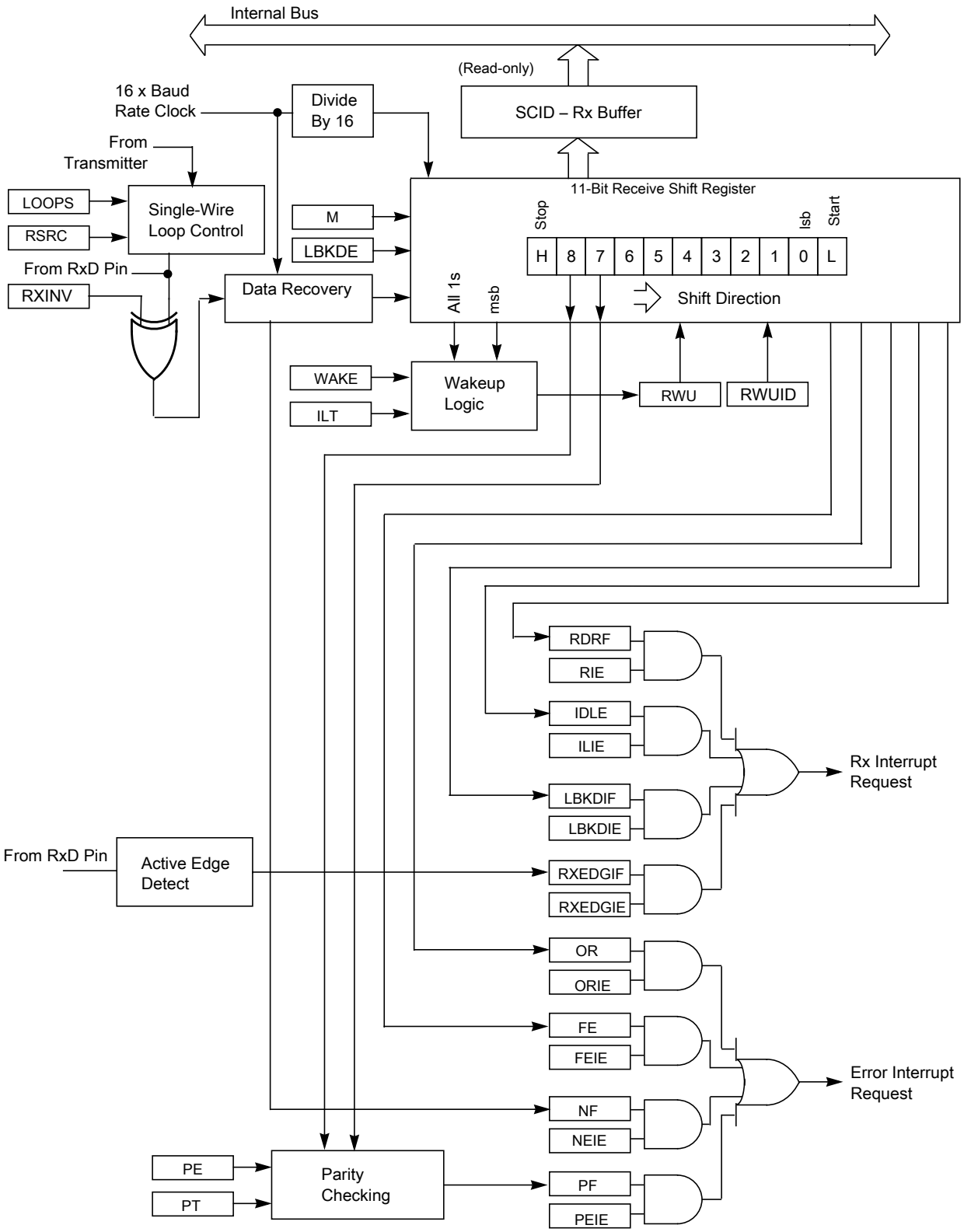



**Figure 15-2. SCI receiver block diagram**

## 15.2 SCI signal descriptions

The SCI signals are shown in the table found here.

**Table 15-1. SCI signal descriptions**

| Signal | Description | I/O |
|---|---|---|
| RxD | Receive data | I |
| TxD | Transmit data | I/O |

### 15.2.1 Detailed signal descriptions

The detailed signal descriptions of the SCI are shown in the following table.

**Table 15-2. SCI—Detailed signal descriptions**

| Signal | I/O | Description | |
|---|---|---|---|
| RxD | I | Receive data. Serial data input to receiver. | |
| | | State meaning | Whether RxD is interpreted as a 1 or 0 depends on the bit encoding method along with other configuration settings. |
| | | Timing | Sampled at a frequency determined by the module clock divided by the baud rate. |
| TxD | I/O | Transmit data. Serial data output from transmitter. | |
| | | State meaning | Whether TxD is interpreted as a 1 or 0 depends on the bit encoding method along with other configuration settings. |
| | | Timing | Driven at the beginning or within a bit time according to the bit encoding method along with other configuration settings. Otherwise, transmissions are independent of reception timing. |

## 15.3 Register definition

The SCI has 8-bit registers to control baud rate, select SCI options, report SCI status, and for transmit/receive data.

Refer to the direct-page register summary in the memory chapter of this document or the absolute address assignments for all SCI registers. This section refers to registers and control bits only by their names. An NXP-provided equate or header file is used to translate these names into the appropriate absolute addresses.

**SCI memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 3080 | SCI Baud Rate Register: High (SCI0_BDH) | 8 | R/W | 00h | 15.3.1/277 |
| 3081 | SCI Baud Rate Register: Low (SCI0_BDL) | 8 | R/W | 04h | 15.3.2/278 |
| 3082 | SCI Control Register 1 (SCI0_C1) | 8 | R/W | 00h | 15.3.3/279 |
| 3083 | SCI Control Register 2 (SCI0_C2) | 8 | R/W | 00h | 15.3.4/280 |
| 3084 | SCI Status Register 1 (SCI0_S1) | 8 | R | C0h | 15.3.5/281 |
| 3085 | SCI Status Register 2 (SCI0_S2) | 8 | R/W | 00h | 15.3.6/283 |
| 3086 | SCI Control Register 3 (SCI0_C3) | 8 | R/W | 00h | 15.3.7/285 |
| 3087 | SCI Data Register (SCI0_D) | 8 | R/W | 00h | 15.3.8/286 |
| 3088 | SCI Baud Rate Register: High (SCI1_BDH) | 8 | R/W | 00h | 15.3.1/277 |
| 3089 | SCI Baud Rate Register: Low (SCI1_BDL) | 8 | R/W | 04h | 15.3.2/278 |
| 308A | SCI Control Register 1 (SCI1_C1) | 8 | R/W | 00h | 15.3.3/279 |
| 308B | SCI Control Register 2 (SCI1_C2) | 8 | R/W | 00h | 15.3.4/280 |
| 308C | SCI Status Register 1 (SCI1_S1) | 8 | R | C0h | 15.3.5/281 |
| 308D | SCI Status Register 2 (SCI1_S2) | 8 | R/W | 00h | 15.3.6/283 |
| 308E | SCI Control Register 3 (SCI1_C3) | 8 | R/W | 00h | 15.3.7/285 |
| 308F | SCI Data Register (SCI1_D) | 8 | R/W | 00h | 15.3.8/286 |

## 15.3.1 SCI Baud Rate Register: High (SCIx_BDH)

This register, along with SCI_BDL, controls the prescale divisor for SCI baud rate generation. To update the 13-bit baud rate setting [SBR12:SBR0], first write to SCI_BDH to buffer the high half of the new value and then write to SCI_BDL. The working value in SCI_BDH does not change until SCI_BDL is written.

Address: Base address + 0h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | LBKDIE | RXEDGIE | SBNS | SBR | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SCIx_BDH field descriptions**

| Field | Description |
|---|---|
| 7<br>LBKDIE | LIN Break Detect Interrupt Enable (for LBKDIF)<br><br>0 Hardware interrupts from SCI_S2[LBKDIF] disabled (use polling).<br>1 Hardware interrupt requested when SCI_S2[LBKDIF] flag is 1. |
| 6<br>RXEDGIE | RxD Input Active Edge Interrupt Enable (for RXEDGIF) |

*Table continues on the next page...*

**SCIx_BDH field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Hardware interrupts from SCI_S2[RXEDGIF] disabled (use polling).<br>1 Hardware interrupt requested when SCI_S2[RXEDGIF] flag is 1. |
| 5<br>SBNS | Stop Bit Number Select<br><br>SBNS determines whether data characters are one or two stop bits.<br><br>0 One stop bit.<br>1 Two stop bit. |
| SBR | Baud Rate Modulo Divisor.<br><br>The 13 bits in SBR[12:0] are referred to collectively as BR, and they set the modulo divide rate for the SCI baud rate generator. When BR is cleared, the SCI baud rate generator is disabled to reduce supply current. When BR is 1 - 8191, the SCI baud rate equals BUSCLK/(16×BR). |

## 15.3.2 SCI Baud Rate Register: Low (SCIx_BDL)

This register, along with SCI_BDH, control the prescale divisor for SCI baud rate generation. To update the 13-bit baud rate setting [SBR12:SBR0], first write to SCI_BDH to buffer the high half of the new value and then write to SCI_BDL. The working value in SCI_BDH does not change until SCI_BDL is written.

SCI_BDL is reset to a non-zero value, so after reset the baud rate generator remains disabled until the first time the receiver or transmitter is enabled; that is, SCI_C2[RE] or SCI_C2[TE] bits are written to 1.

Address: Base address + 1h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | SBR | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

**SCIx_BDL field descriptions**

| Field | Description |
|---|---|
| SBR | Baud Rate Modulo Divisor<br><br>These 13 bits in SBR[12:0] are referred to collectively as BR. They set the modulo divide rate for the SCI baud rate generator. When BR is cleared, the SCI baud rate generator is disabled to reduce supply current. When BR is 1 - 8191, the SCI baud rate equals BUSCLK/(16×BR). |

## 15.3.3 SCI Control Register 1 (SCIx_C1)

This read/write register controls various optional features of the SCI system.

Address: Base address + 2h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | LOOPS | SCISWAI | RSRC | M | WAKE | ILT | PE | PT |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SCIx_C1 field descriptions**

| Field | Description |
|---|---|
| 7<br>LOOPS | Loop Mode Select<br><br>Selects between loop back modes and normal 2-pin full-duplex modes. When LOOPS is set, the transmitter output is internally connected to the receiver input.<br><br>0 Normal operation - RxD and TxD use separate pins.<br>1 Loop mode or single-wire mode where transmitter outputs are internally connected to receiver input. (See RSRC bit.) RxD pin is not used by SCI. |
| 6<br>SCISWAI | SCI Stops in Wait Mode<br><br>0 SCI clocks continue to run in wait mode so the SCI can be the source of an interrupt that wakes up the CPU.<br>1 SCI clocks freeze while CPU is in wait mode. |
| 5<br>RSRC | Receiver Source Select<br><br>This bit has no meaning or effect unless the LOOPS bit is set to 1. When LOOPS is set, the receiver input is internally connected to the TxD pin and RSRC determines whether this connection is also connected to the transmitter output.<br><br>0 Provided LOOPS is set, RSRC is cleared, selects internal loop back mode and the SCI does not use the RxD pins.<br>1 Single-wire SCI mode where the TxD pin is connected to the transmitter output and receiver input. |
| 4<br>M | 9-Bit or 8-Bit Mode Select<br><br>0 Normal - start + 8 data bits (lsb first) + stop.<br>1 Receiver and transmitter use 9-bit data characters start + 8 data bits (lsb first) + 9th data bit + stop. |
| 3<br>WAKE | Receiver Wakeup Method Select<br><br>0 Idle-line wakeup.<br>1 Address-mark wakeup. |
| 2<br>ILT | Idle Line Type Select<br><br>Setting this bit to 1 ensures that the stop bits and logic 1 bits at the end of a character do not count toward the 10 or 11 bit times of logic high level needed by the idle line detection logic.<br><br>0 Idle character bit count starts after start bit.<br>1 Idle character bit count starts after stop bit. |

*Table continues on the next page...*

**SCIx_C1 field descriptions (continued)**

| Field | Description |
|---|---|
| 1<br>PE | Parity Enable<br><br>Enables hardware parity generation and checking. When parity is enabled, the most significant bit (msb) of the data character, eighth or ninth data bit, is treated as the parity bit.<br><br>0 No hardware parity generation or checking.<br>1 Parity enabled. |
| 0<br>PT | Parity Type<br><br>Provided parity is enabled (PE = 1), this bit selects even or odd parity. Odd parity means the total number of 1s in the data character, including the parity bit, is odd. Even parity means the total number of 1s in the data character, including the parity bit, is even.<br><br>0 Even parity.<br>1 Odd parity. |

## 15.3.4 SCI Control Register 2 (SCIx_C2)

This register can be read or written at any time.

Address: Base address + 3h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | TIE | TCIE | RIE | ILIE | TE | RE | RWU | SBK |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SCIx_C2 field descriptions**

| Field | Description |
|---|---|
| 7<br>TIE | Transmit Interrupt Enable for TDRE<br><br>0 Hardware interrupts from TDRE disabled; use polling.<br>1 Hardware interrupt requested when TDRE flag is 1. |
| 6<br>TCIE | Transmission Complete Interrupt Enable for TC<br><br>0 Hardware interrupts from TC disabled; use polling.<br>1 Hardware interrupt requested when TC flag is 1. |
| 5<br>RIE | Receiver Interrupt Enable for RDRF<br><br>0 Hardware interrupts from RDRF disabled; use polling.<br>1 Hardware interrupt requested when RDRF flag is 1. |
| 4<br>ILIE | Idle Line Interrupt Enable for IDLE<br><br>0 Hardware interrupts from IDLE disabled; use polling.<br>1 Hardware interrupt requested when IDLE flag is 1. |
| 3<br>TE | Transmitter Enable |

*Table continues on the next page...*

**SCIx_C2 field descriptions (continued)**

| Field | Description |
|---|---|
| | TE must be 1 to use the SCI transmitter. When TE is set, the SCI forces the TxD pin to act as an output for the SCI system.<br>When the SCI is configured for single-wire operation (LOOPS = RSRC = 1), TXDIR controls the direction of traffic on the single SCI communication line (TxD pin).<br>TE can also queue an idle character by clearing TE then setting TE while a transmission is in progress.<br>When TE is written to 0, the transmitter keeps control of the port TxD pin until any data, queued idle, or queued break character finishes transmitting before allowing the pin to revert to a general-purpose I/O pin.<br>0 Transmitter off.<br>1 Transmitter on. |
| 2<br>RE | Receiver Enable<br>When the SCI receiver is off, the RxD pin reverts to being a general-purpose port I/O pin. If LOOPS is set the RxD pin reverts to being a general-purpose I/O pin even if RE is set.<br>0 Receiver off.<br>1 Receiver on. |
| 1<br>RWU | Receiver Wakeup Control<br>This bit can be written to 1 to place the SCI receiver in a standby state where it waits for automatic hardware detection of a selected wakeup condition. The wakeup condition is an idle line between messages, WAKE = 0, idle-line wakeup, or a logic 1 in the most significant data bit in a character, WAKE = 1, address-mark wakeup. Application software sets RWU and, normally, a selected hardware condition automatically clears RWU.<br>0 Normal SCI receiver operation.<br>1 SCI receiver in standby waiting for wakeup condition. |
| 0<br>SBK | Send Break<br>Writing a 1 and then a 0 to SBK queues a break character in the transmit data stream. Additional break characters of 10 or 11 or 12, 13 or 14 or 15 if BRK13 = 1, bit times of logic 0 are queued as long as SBK is set. Depending on the timing of the set and clear of SBK relative to the information currently being transmitted, a second break character may be queued before software clears SBK.<br>0 Normal transmitter operation.<br>1 Queue break character(s) to be sent. |

## 15.3.5 SCI Status Register 1 (SCIx_S1)

This register has eight read-only status flags. Writes have no effect. Special software sequences, which do not involve writing to this register, clear these status flags.

Address: Base address + 4h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | TDRE | TC | RDRF | IDLE | OR | NF | FE | PF |
| Write | | | | | | | | |
| Reset | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

## SCIx_S1 field descriptions

| Field | Description |
|---|---|
| 7<br>TDRE | Transmit Data Register Empty Flag<br><br>TDRE is set out of reset and when a transmit data value transfers from the transmit data buffer to the transmit shifter, leaving room for a new character in the buffer. To clear TDRE, read SCI_S1 with TDRE set and then write to the SCI data register (SCI_D).<br><br>0 Transmit data register (buffer) full.<br>1 Transmit data register (buffer) empty. |
| 6<br>TC | Transmission Complete Flag<br>TC is set out of reset and when TDRE is set and no data, preamble, or break character is being transmitted.<br>TC is cleared automatically by reading SCI_S1 with TC set and then doing one of the following:<br>• Write to the SCI data register (SCI_D) to transmit new data<br>• Queue a preamble by changing TE from 0 to 1<br>• Queue a break character by writing 1 to SCI_C2[SBK]<br><br>0 Transmitter active (sending data, a preamble, or a break).<br>1 Transmitter idle (transmission activity complete). |
| 5<br>RDRF | Receive Data Register Full Flag<br><br>RDRF becomes set when a character transfers from the receive shifter into the receive data register (SCI_D). To clear RDRF, read SCI_S1 with RDRF set and then read the SCI data register (SCI_D).<br><br>0 Receive data register empty.<br>1 Receive data register full. |
| 4<br>IDLE | Idle Line Flag<br><br>IDLE is set when the SCI receive line becomes idle for a full character time after a period of activity. When ILT is cleared, the receiver starts counting idle bit times after the start bit. If the receive character is all 1s, these bit times and the stop bits time count toward the full character time of logic high, 10 or 11 bit times depending on the M control bit, needed for the receiver to detect an idle line. When ILT is set, the receiver doesn't start counting idle bit times until after the stop bits. The stop bits and any logic high bit times at the end of the previous character do not count toward the full character time of logic high needed for the receiver to detect an idle line.<br><br>To clear IDLE, read SCI_S1 with IDLE set and then read the SCI data register (SCI_D). After IDLE has been cleared, it cannot become set again until after a new character has been received and RDRF has been set. IDLE is set only once even if the receive line remains idle for an extended period.<br><br>0 No idle line detected.<br>1 Idle line was detected. |
| 3<br>OR | Receiver Overrun Flag<br><br>OR is set when a new serial character is ready to be transferred to the receive data register (buffer), but the previously received character has not been read from SCI_D yet. In this case, the new character, and all associated error information, is lost because there is no room to move it into SCI_D. To clear OR, read SCI_S1 with OR set and then read the SCI data register (SCI_D).<br><br>0 No overrun.<br>1 Receive overrun (new SCI data lost). |
| 2<br>NF | Noise Flag<br><br>The advanced sampling technique used in the receiver takes seven samples during the start bit and three samples in each data bit and the stop bits. If any of these samples disagrees with the rest of the samples |

*Table continues on the next page...*

**SCIx_S1 field descriptions (continued)**

| Field | Description |
|---|---|
| | within any bit time in the frame, the flag NF is set at the same time as RDRF is set for the character. To clear NF, read SCI_S1 and then read the SCI data register (SCI_D).<br><br>0 No noise detected.<br>1 Noise detected in the received character in SCI_D. |
| 1<br>FE | Framing Error Flag<br><br>FE is set at the same time as RDRF when the receiver detects a logic 0 where the stop bits was expected. This suggests the receiver was not properly aligned to a character frame. To clear FE, read SCI_S1 with FE set and then read the SCI data register (SCI_D).<br><br>0 No framing error detected. This does not guarantee the framing is correct.<br>1 Framing error. |
| 0<br>PF | Parity Error Flag<br><br>PF is set at the same time as RDRF when parity is enabled (PE = 1) and the parity bit in the received character does not agree with the expected parity value. To clear PF, read SCI_S1 and then read the SCI data register (SCI_D).<br><br>0 No parity error.<br>1 Parity error. |

## 15.3.6 SCI Status Register 2 (SCIx_S2)

This register contains one read-only status flag.

When using an internal oscillator in a LIN system, it is necessary to raise the break detection threshold one bit time. Under the worst case timing conditions allowed in LIN, it is possible that a 0x00 data character can appear to be 10.26 bit times long at a slave running 14% faster than the master. This would trigger normal break detection circuitry designed to detect a 10-bit break symbol. When the LBKDE bit is set, framing errors are inhibited and the break detection threshold changes from 10 bits to 11 bits, preventing false detection of a 0x00 data character as a LIN break symbol.

Address: Base address + 5h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | LBKDIF | RXEDGIF | 0 | RXINV | RWUID | BRK13 | LBKDE | RAF |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SCIx_S2 field descriptions**

| Field | Description |
|---|---|
| 7<br>LBKDIF | LIN Break Detect Interrupt Flag<br><br>LBKDIF is set when the LIN break detect circuitry is enabled and a LIN break character is detected. LBKDIF is cleared by writing a 1 to it. |

*Table continues on the next page...*

## SCIx_S2 field descriptions (continued)

| Field | Description |
|---|---|
| | 0 No LIN break character has been detected.<br>1 LIN break character has been detected. |
| 6<br>RXEDGIF | RxD Pin Active Edge Interrupt Flag<br><br>RXEDGIF is set when an active edge, falling if RXINV = 0, rising if RXINV=1, on the RxD pin occurs. RXEDGIF is cleared by writing a 1 to it.<br><br>0 No active edge on the receive pin has occurred.<br>1 An active edge on the receive pin has occurred. |
| 5<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 4<br>RXINV | Receive Data Inversion<br><br>Setting this bit reverses the polarity of the received data input.<br><br>**NOTE:** Setting RXINV inverts the RxD input for all cases: data bits, start and stop bits, break, and idle.<br><br>0 Receive data not inverted.<br>1 Receive data inverted. |
| 3<br>RWUID | Receive Wake Up Idle Detect<br><br>RWUID controls whether the idle character that wakes up the receiver sets the IDLE bit.<br><br>0 During receive standby state (RWU = 1), the IDLE bit does not get set upon detection of an idle character.<br>1 During receive standby state (RWU = 1), the IDLE bit gets set upon detection of an idle character. |
| 2<br>BRK13 | Break Character Generation Length<br><br>BRK13 selects a longer transmitted break character length. Detection of a framing error is not affected by the state of this bit.<br><br>0 Break character is transmitted with length of 10 bit times (if M = 0, SBNS = 0) or 11 (if M = 1, SBNS = 0 or M = 0, SBNS = 1) or 12 (if M = 1, SBNS = 1).<br>1 Break character is transmitted with length of 13 bit times (if M = 0, SBNS = 0) or 14 (if M = 1, SBNS = 0 or M = 0, SBNS = 1) or 15 (if M = 1, SBNS = 1). |
| 1<br>LBKDE | LIN Break Detection Enable<br><br>LBKDE selects a longer break character detection length. While LBKDE is set, framing error (FE) and receive data register full (RDRF) flags are prevented from setting.<br><br>0 Break character is detected at length 10 bit times (if M = 0, SBNS = 0) or 11 (if M = 1, SBNS = 0 or M = 0, SBNS = 1) or 12 (if M = 1, SBNS = 1).<br>1 Break character is detected at length of 11 bit times (if M = 0, SBNS = 0) or 12 (if M = 1, SBNS = 0 or M = 0, SBNS = 1) or 13 (if M = 1, SBNS = 1). |
| 0<br>RAF | Receiver Active Flag<br><br>RAF is set when the SCI receiver detects the beginning of a valid start bit, and RAF is cleared automatically when the receiver detects an idle line. This status flag can be used to check whether an SCI character is being received before instructing the MCU to go to stop mode.<br><br>0 SCI receiver idle waiting for a start bit.<br>1 SCI receiver active (RxD input not idle). |

## 15.3.7 SCI Control Register 3 (SCIx_C3)

Address: Base address + 6h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | R8 | T8 | TXDIR | TXINV | ORIE | NEIE | FEIE | PEIE |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SCIx_C3 field descriptions**

| Field | Description |
|---|---|
| 7<br>R8 | Ninth Data Bit for Receiver<br><br>When the SCI is configured for 9-bit data (M = 1), R8 can be thought of as a ninth receive data bit to the left of the msb of the buffered data in the SCI_D register. When reading 9-bit data, read R8 before reading SCI_D because reading SCI_D completes automatic flag clearing sequences that could allow R8 and SCI_D to be overwritten with new data. |
| 6<br>T8 | Ninth Data Bit for Transmitter<br><br>When the SCI is configured for 9-bit data (M = 1), T8 may be thought of as a ninth transmit data bit to the left of the msb of the data in the SCI_D register. When writing 9-bit data, the entire 9-bit value is transferred to the SCI shift register after SCI_D is written so T8 should be written, if it needs to change from its previous value, before SCI_D is written. If T8 does not need to change in the new value, such as when it is used to generate mark or space parity, it need not be written each time SCI_D is written. |
| 5<br>TXDIR | TxD Pin Direction in Single-Wire Mode<br><br>When the SCI is configured for single-wire half-duplex operation (LOOPS = RSRC = 1), this bit determines the direction of data at the TxD pin.<br><br>0 TxD pin is an input in single-wire mode.<br>1 TxD pin is an output in single-wire mode. |
| 4<br>TXINV | Transmit Data Inversion<br><br>Setting this bit reverses the polarity of the transmitted data output.<br><br>**NOTE:** Setting TXINV inverts the TxD output for all cases: data bits, start and stop bits, break, and idle.<br><br>0 Transmit data not inverted.<br>1 Transmit data inverted. |
| 3<br>ORIE | Overrun Interrupt Enable<br><br>This bit enables the overrun flag (OR) to generate hardware interrupt requests.<br><br>0 OR interrupts disabled; use polling.<br>1 Hardware interrupt requested when OR is set. |
| 2<br>NEIE | Noise Error Interrupt Enable<br><br>This bit enables the noise flag (NF) to generate hardware interrupt requests.<br><br>0 NF interrupts disabled; use polling).<br>1 Hardware interrupt requested when NF is set. |

*Table continues on the next page...*

**SCIx_C3 field descriptions (continued)**

| Field | Description |
|---|---|
| 1<br>FEIE | Framing Error Interrupt Enable<br><br>This bit enables the framing error flag (FE) to generate hardware interrupt requests.<br><br>0 FE interrupts disabled; use polling).<br>1 Hardware interrupt requested when FE is set. |
| 0<br>PEIE | Parity Error Interrupt Enable<br><br>This bit enables the parity error flag (PF) to generate hardware interrupt requests.<br><br>0 PF interrupts disabled; use polling).<br>1 Hardware interrupt requested when PF is set. |

## 15.3.8 SCI Data Register (SCIx_D)

This register is actually two separate registers. Reads return the contents of the read-only receive data buffer and writes go to the write-only transmit data buffer. Reads and writes of this register are also involved in the automatic flag clearing mechanisms for the SCI status flags.

Address: Base address + 7h offset

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | R7T7 | R6T6 | R5T5 | R4T4 | R3T3 | R2T2 | R1T1 | R0T0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**SCIx_D field descriptions**

| Field | Description |
|---|---|
| 7<br>R7T7 | Read receive data buffer 7 or write transmit data buffer 7. |
| 6<br>R6T6 | Read receive data buffer 6 or write transmit data buffer 6. |
| 5<br>R5T5 | Read receive data buffer 5 or write transmit data buffer 5. |
| 4<br>R4T4 | Read receive data buffer 4 or write transmit data buffer 4. |
| 3<br>R3T3 | Read receive data buffer 3 or write transmit data buffer 3. |
| 2<br>R2T2 | Read receive data buffer 2 or write transmit data buffer 2. |
| 1<br>R1T1 | Read receive data buffer 1 or write transmit data buffer 1. |
| 0<br>R0T0 | Read receive data buffer 0 or write transmit data buffer 0. |

**SCIx_D field descriptions (continued)**

| Field | Description |
|---|---|

## 15.4 Functional description

The SCI allows full-duplex, asynchronous, NRZ serial communication among the MCU and remote devices, including other MCUs.

The SCI comprises a baud rate generator, transmitter, and receiver block. The transmitter and receiver operate independently, although they use the same baud rate generator. During normal operation, the MCU monitors the status of the SCI, writes the data to be transmitted, and processes received data. The following describes each of the blocks of the SCI.

### 15.4.1 Baud rate generation

As shown in the figure found here, the clock source for the SCI baud rate generator is the bus-rate clock.

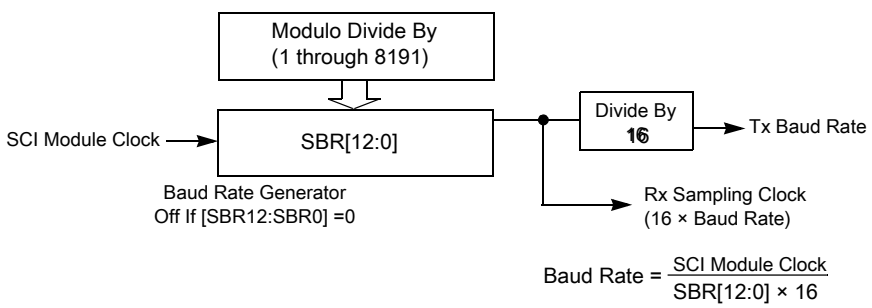



**Figure 15-3. SCI baud rate generation**

SCI communications require the transmitter and receiver, which typically derive baud rates from independent clock sources, to use the same baud rate. Allowed tolerance on this baud frequency depends on the details of how the receiver synchronizes to the leading edge of the start bit and how bit sampling is performed.

The MCU resynchronizes to bit boundaries on every high-to-low transition. In the worst case, there are no such transitions in the full 10- or 11-bit or 12-bittime character frame so any mismatch in baud rate is accumulated for the whole character time. For an NXP SCI system whose bus frequency is driven by a crystal, the allowed baud rate mismatch is about ±4.5 percent for 8-bit data format and about ±4 percent for 9-bit data format.

Although baud rate modulo divider settings do not always produce baud rates that exactly match standard rates, it is normally possible to get within a few percent, which is acceptable for reliable communications.

### 15.4.2 Transmitter functional description

This section describes the overall block diagram for the SCI transmitter, as well as specialized functions for sending break and idle characters.

The transmitter output (TxD) idle state defaults to logic high, SCI_C3[TXINV] is cleared following reset. The transmitter output is inverted by setting SCI_C3[TXINV]. The transmitter is enabled by setting the TE bit in SCI_C2. This queues a preamble character that is one full character frame of the idle state. The transmitter then remains idle until data is available in the transmit data buffer. Programs store data into the transmit data buffer by writing to the SCI data register (SCI_D).

The central element of the SCI transmitter is the transmit shift register that is 10 or 11 or 12 bits long depending on the setting in the SCI_C1[M] control bit and SCI_BDH[SBNS] bit. For the remainder of this section, assume SCI_C1[M] is cleared, SCI_BDH[SBNS] is also cleared, selecting the normal 8-bit data mode. In 8-bit data mode, the shift register holds a start bit, eight data bits, and a stop bit. When the transmit shift register is available for a new SCI character, the value waiting in the transmit data register is transferred to the shift register, synchronized with the baud rate clock, and the transmit data register empty (SCI_S1[TDRE]) status flag is set to indicate another character may be written to the transmit data buffer at SCI_D.

**NOTE**

Always read SCI_S1 before writing to SCI_D to allow data to be transmitted.

If no new character is waiting in the transmit data buffer after a stop bit is shifted out the TxD pin, the transmitter sets the transmit complete flag and enters an idle mode, with TxD high, waiting for more characters to transmit.

Writing 0 to SCI_C2[TE] does not immediately release the pin to be a general-purpose I/O pin. Any transmit activity in progress must first be completed. This includes data characters in progress, queued idle characters, and queued break characters.

#### 15.4.2.1 Send break and queued idle

SCI_C2[SBK] sends break characters originally used to gain the attention of old teletype receivers. Break characters are a full character time of logic 0, 10 bit times including the start and stop bits. A longer break of 13 bit times can be enabled by setting SCI_S2[BRK13]. Normally, a program would wait for SCI_S1[TDRE] to become set to indicate the last character of a message has moved to the transmit shifter, write 1, and then write 0 to SCI_C2[SBK]. This action queues a break character to be sent as soon as the shifter is available. If SCI_C2[SBK] remains 1 when the queued break moves into the shifter, synchronized to the baud rate clock, an additional break character is queued. If the receiving device is another NXP SCI, the break characters are received as 0s in all eight data bits and a framing error (SCI_S1[FE] = 1) occurs.

When idle-line wake-up is used, a full character time of idle (logic 1) is needed between messages to wake up any sleeping receivers. Normally, a program would wait for SCI_S1[TDRE] to become set to indicate the last character of a message has moved to the transmit shifter, then write 0 and then write 1 to the SCI_C2[TE] bit. This action queues an idle character to be sent as soon as the shifter is available. As long as the character in the shifter does not finish while SCI_C2[TE] is cleared, the SCI transmitter never actually releases control of the TxD pin. If there is a possibility of the shifter finishing while SCI_C2[TE] is cleared, set the general-purpose I/O controls so the pin shared with TxD is an output driving a logic 1. This ensures that the TxD line looks like a normal idle line even if the SCI loses control of the port pin between writing 0 and then 1 to SCI_C2[TE].

The length of the break character is affected by the SCI_S2[BRK13] and SCI_C1[M] as shown below.

**Table 15-3. Break character length**

| BRK13 | M | SBNS | Break character length |
|---|---|---|---|
| 0 | 0 | 0 | 10 bit times |
| 0 | 0 | 1 | 11 bit times |
| 0 | 1 | 0 | 11 bit times |
| 0 | 1 | 1 | 12 bit times |
| 1 | 0 | 0 | 13 bit times |
| 1 | 0 | 1 | 14 bit times |
| 1 | 1 | 0 | 14 bit times |
| 1 | 1 | 1 | 15 bit times |

### 15.4.3 Receiver functional description

In this section, the receiver block diagram is a guide for the overall receiver functional description.

Next, the data sampling technique used to reconstruct receiver data is described in more detail. Finally, two variations of the receiver wakeup function are explained.

The receiver input is inverted by setting SCI_S2[RXINV]. The receiver is enabled by setting the SCI_C2[RE] bit. Character frames consist of a start bit of logic 0, eight (or nine) data bits (lsb first), and one (or two) stop bits of logic 1. For information about 9-bit data mode, refer to 8- and 9-bit data modes. For the remainder of this discussion, assume the SCI is configured for normal 8-bit data mode.

After receiving the stop bit into the receive shifter, and provided the receive data register is not already full, the data character is transferred to the receive data register and the receive data register full (SCI_S1[RDRF]) status flag is set. If SCI_S1[RDRF] was already set indicating the receive data register (buffer) was already full, the overrun (OR) status flag is set and the new data is lost. Because the SCI receiver is double-buffered, the program has one full character time after SCI_S1[RDRF] is set before the data in the receive data buffer must be read to avoid a receiver overrun.

When a program detects that the receive data register is full (SCI_S1[RDRF] = 1), it gets the data from the receive data register by reading SCI_D. The SCI_S1[RDRF] flag is cleared automatically by a two-step sequence normally satisfied in the course of the user's program that manages receive data. Refer to Interrupts and status flags for more details about flag clearing.

#### 15.4.3.1 Data sampling technique

The SCI receiver uses a 16× baud rate clock for sampling. The oversampling ratio is fixed at 16. The receiver starts by taking logic level samples at 16 times the baud rate to search for a falling edge on the RxD serial data input pin. A falling edge is defined as a logic 0 sample after three consecutive logic 1 samples. The 16× baud rate clock divides the bit time into 16 segments labeled SCI_D[RT1] through SCI_D[RT16]. When a falling edge is located, three more samples are taken at SCI_D[RT3], SCI_D[RT5], and SCI_D[RT7] to make sure this was a real start bit and not merely noise. If at least two of these three samples are 0, the receiver assumes it is synchronized to a receive character.

The receiver then samples each bit time, including the start and stop bits, at SCI_D[RT8], SCI_D[RT9], and SCI_D[RT10] to determine the logic level for that bit. The logic level is interpreted to be that of the majority of the samples taken during the bit time. In the case of the start bit, the bit is assumed to be 0 if at least two of the samples at

SCI_D[RT3], SCI_D[RT5], and SCI_D[RT7] are 0 even if one or all of the samples taken at SCI_D[RT8], SCI_D[RT9], and SCI_D[RT10] are 1s. If any sample in any bit time, including the start and stop bits, in a character frame fails to agree with the logic level for that bit, the noise flag (SCI_S1[NF]) is set when the received character is transferred to the receive data buffer.

The falling edge detection logic continuously looks for falling edges. If an edge is detected, the sample clock is resynchronized to bit times. This improves the reliability of the receiver in the presence of noise or mismatched baud rates. It does not improve worst case analysis because some characters do not have any extra falling edges anywhere in the character frame.

In the case of a framing error, provided the received character was not a break character, the sampling logic that searches for a falling edge is filled with three logic 1 samples so that a new start bit can be detected almost immediately.

In the case of a framing error, the receiver is inhibited from receiving any new characters until the framing error flag is cleared. The receive shift register continues to function, but a complete character cannot transfer to the receive data buffer if SCI_S1[FE] remains set.

### 15.4.3.2 Receiver wake-up operation

Receiver wake-up is a hardware mechanism that allows an SCI receiver to ignore the characters in a message intended for a different SCI receiver. In such a system, all receivers evaluate the first character(s) of each message, and as soon as they determine the message is intended for a different receiver, they write logic 1 to the receiver wake up control field (SCI_C2[RWU]). When SCI_C2[RWU] is set, the status flags associated with the receiver, (with the exception of the idle bit, IDLE, when SCI_S2[RWUID] is set), are inhibited from setting, thus eliminating the software overhead for handling the unimportant message characters. At the end of a message, or at the beginning of the next message, all receivers automatically force SCI_C2[RWU] to 0, so all receivers wake up in time to look at the first character(s) of the next message.

#### 15.4.3.2.1 Idle-line wakeup

When wake is cleared, the receiver is configured for idle-line wakeup. In this mode, SCI_C2[RWU] is cleared automatically when the receiver detects a full character time of the idle-line level. The SCI_C1[M] control field selects 8-bit or 9-bit data mode and SCI_BDH[SBNS] selects 1-bit or 2-bit stop bit number that determines how many bit times of idle are needed to constitute a full character time, 10 or 11 or 12 bit times because of the start and stop bits.

When SCII_C2[RWU] is 1 and SCI_S2[RWUID] is 0, the idle condition that wakes up the receiver does not set SCI_S1[IDLE]. The receiver wakes up and waits for the first data character of the next message that sets SCI_S1[RDRF] and generates an interrupt, if enabled. When SCI_S2[RWUID] is 1, any idle condition sets SCI_S1[IDLE] flag and generates an interrupt if enabled, regardless of whether SCI_C2[RWU] is 0 or 1.

The idle-line type (SCI_C1[ILT]) control bit selects one of two ways to detect an idle line. When SCI_C1[ILT] is cleared, the idle bit counter starts after the start bit so the stop bit and any logic 1s at the end of a character count toward the full character time of idle. When SCI_C1[ILT] is set, the idle bit counter does not start until after a stop bit time, so the idle detection is not affected by the data in the last character of the previous message.

#### 15.4.3.2.2 Address-mark wakeup

When wake is set, the receiver is configured for address-mark wakeup. In this mode, SCI_C2[RWU] is cleared automatically when the receiver detects a, or two, if SCI_BDH[SBNS] = 1, logic 1 in the most significant bits of a received character, eighth bit when SCI_C1[M] is cleared and ninth bit when SCI_C1[M] is set.

Address-mark wakeup allows messages to contain idle characters, but requires the msb be reserved for use in address frames. The one, or two, if SCI_BDH[SBNS] = 1, logic 1s msb of an address frame clears the SCI_C2[RWU] bit before the stop bits are received and sets the SCI_S1[RDRF] flag. In this case, the character with the msb set is received even though the receiver was sleeping during most of this character time.

### 15.4.4 Interrupts and status flags

The SCI system has three separate interrupt vectors to reduce the amount of software needed to isolate the cause of the interrupt.

One interrupt vector is associated with the transmitter for SCI_S1[TDRE] and SCI_S1[TC] events. Another interrupt vector is associated with the receiver for RDRF, IDLE, RXEDGIF, and LBKDIF events. A third vector is used for OR, NF, FE, and PF error conditions. Each of these ten interrupt sources can be separately masked by local interrupt enable masks. The flags can be polled by software when the local masks are cleared to disable generation of hardware interrupt requests.

The SCI transmitter has two status flags that can optionally generate hardware interrupt requests. Transmit data register empty (SCI_S1[TDRE]) indicates when there is room in the transmit data buffer to write another transmit character to SCI_D. If the transmit interrupt enable (SCI_C2[TIE]) bit is set, a hardware interrupt is requested when SCI_S1[TDRE] is set. Transmit complete (SCI_S1[TC]) indicates that the transmitter is

finished transmitting all data, preamble, and break characters and is idle with TxD at the inactive level. This flag is often used in systems with modems to determine when it is safe to turn off the modem. If the transmit complete interrupt enable (SCI_C2[TCIE]) bit is set, a hardware interrupt is requested when SCI_S1[TC] is set. Instead of hardware interrupts, software polling may be used to monitor the SCI_S1[TDRE] and SCI_S1[TC] status flags if the corresponding SCI_C2[TIE] or SCI_C2[TCIE] local interrupt masks are cleared.

When a program detects that the receive data register is full (SCI_S1[RDRF] = 1), it gets the data from the receive data register by reading SCI_D. The SCI_S1[RDRF] flag is cleared by reading SCI_S1 while SCI_S1[RDRF] is set and then reading SCI_D.

When polling is used, this sequence is naturally satisfied in the normal course of the user program. If hardware interrupts are used, SCI_S1 must be read in the interrupt service routine (ISR). Normally, this is done in the ISR anyway to check for receive errors, so the sequence is automatically satisfied.

The IDLE status flag includes logic that prevents it from getting set repeatedly when the RxD line remains idle for an extended period of time. IDLE is cleared by reading SCI_S1 while SCI_S1[IDLE] is set and then reading SCI_D. After SCI_S1[IDLE] has been cleared, it cannot become set again until the receiver has received at least one new character and has set SCI_S1[RDRF].

If the associated error was detected in the received character that caused SCI_S1[RDRF] to be set, the error flags - noise flag (SCI_S1[NF]), framing error (SCI_S1[FE]), and parity error flag (SCI_S1[PF]) - are set at the same time as SCI_S1[RDRF]. These flags are not set in overrun cases.

If SCI_S1[RDRF] was already set when a new character is ready to be transferred from the receive shifter to the receive data buffer, the overrun (SCI_S1[OR]) flag is set instead of the data along with any associated NF, FE, or PF condition is lost.

At any time, an active edge on the RxD serial data input pin causes the SCI_S2[RXEDGIF] flag to set. The SCI_S2[RXEDGIF] flag is cleared by writing a 1 to it. This function depends on the receiver being enabled (SCI_C2[RE] = 1).

### 15.4.5 Baud rate tolerance

A transmitting device may operate at a baud rate below or above that of the receiver.

Accumulated bit time misalignment can cause one of the three stop bit data samples (RT8, RT9, and RT10) to fall outside the actual stop bit. A noise error will occur if the RT8, RT9, and RT10 samples are not all the same logical values. A framing error will occur if the receiver clock is misaligned in such a way that the majority of the RT8, RT9, and RT10 stop bit samples are a logic zero.

As the receiver samples an incoming frame, it re-synchronizes the RT clock on any valid falling edge within the frame. Resynchronization within frames will correct a misalignment between transmitter bit times and receiver bit times.

### 15.4.5.1 Slow data tolerance

Figure 15-4 shows how much a slow received frame can be misaligned without causing a noise error or a framing error. The slow stop bit begins at RT8 instead of RT1 but arrives in time for the stop bit data samples at RT8, RT9, and RT10.

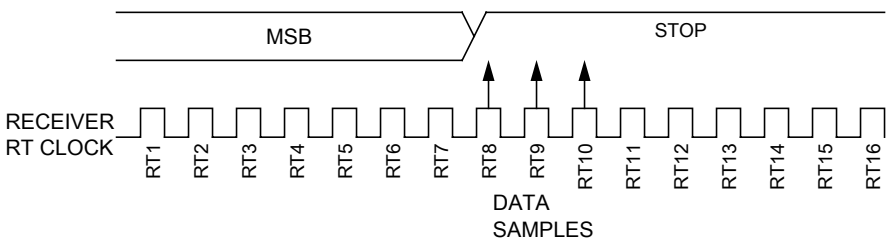



**Figure 15-4. Slow data**

For an 8-bit data and 1 stop bit character, data sampling of the stop bit takes the receiver 9 bit times x 16 RT cycles +10 RT cycles =154 RT cycles.

With the misaligned character shown in Figure 15-4, the receiver counts 154 RT cycles at the point when the count of the transmitting device is 9 bit times x 16 RT cycles + 3 RT cycles = 147 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a slow 8-bit data and 1 stop bit character with no errors is:

((154 - 147) / 154) x 100 = 4.54%

For a 9-bit data or 2 stop bits character, data sampling of the stop bit takes the receiver 10 bit times x 16 RT cycles + 10 RT cycles = 170 RT cycles.

With the misaligned character shown in Figure 15-4, the receiver counts 170 RT cycles at the point when the count of the transmitting device is 10 bit times x 16 RT cycles + 3 RT cycles = 163 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a slow 9-bit or 2 stop bits character with no errors is:

((170 - 163) / 170) X 100 = 4.12%

For a 9-bit data and 2 stop bit character, data sampling of the stop bit takes the receiver 11 bit times x 16 RT cycles + 10 RT cycles = 186 RT cycles.

With the misaligned character shown in Figure 15-4, the receiver counts 186 RT cycles at the point when the count of the transmitting device is 11 bit times x 16 RT cycles + 3 RT cycles = 179 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a slow 9-bit and 2 stop bits character with no errors is: ((186 - 179) / 186) X 100 = 3.76%

### 15.4.5.2 Fast data tolerance

Figure 15-5 shows how much a fast received frame can be misaligned. The fast stop bit ends at RT10 instead of RT16 but is still sampled at RT8, RT9, and RT10.

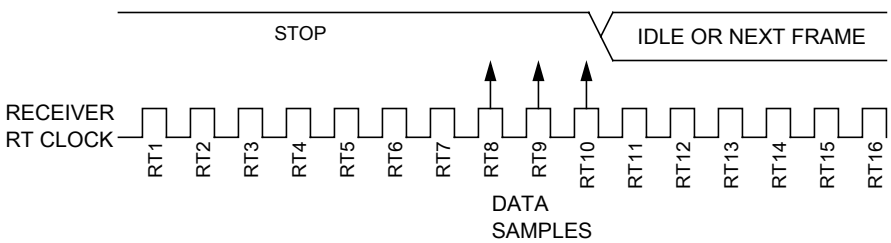



**Figure 15-5. Fast data**

For an 8-bit data and 1 stop bit character, data sampling of the stop bit takes the receiver 9 bit times x 16 RT cycles + 10 RT cycles = 154 RT cycles.

With the misaligned character shown in Figure 15-5, the receiver counts 154 RT cycles at the point when the count of the transmitting device is 10 bit times x 16 RT cycles = 160 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a fast 8-bit and 1 stop bit character with no errors is:

((154 - 160) / 154) x 100 = 3.90%

For a 9-bit data or 2 stop bits character, data sampling of the stop bit takes the receiver 10 bit times x 16 RT cycles + 10 RT cycles = 170 RT cycles.

With the misaligned character shown in, the receiver counts 170 RT cycles at the point when the count of the transmitting device is 11 bit times x 16 RT cycles = 176 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a fast 9-bit or 2 stop bits character with no errors is:

((170 - 176) / 170) x 100 = 3.53%

For a 9-bit data and 2 stop bits character, data sampling of the stop bit takes the receiver 11 bit times x 16 RT cycles + 10 RT cycles = 186 RT cycles.

With the misaligned character shown in, the receiver counts 186 RT cycles at the point when the count of the transmitting device is 12 bit times x 16 RT cycles = 192 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a fast 9-bit and 2 stop bits character with no errors is:

((186 - 192) / 186) x 100 = 3.23%

## 15.4.6 Additional SCI functions

The following sections describe additional SCI functions.

### 15.4.6.1 8- and 9-bit data modes

The SCI system, transmitter and receiver, can be configured to operate in 9-bit data mode by setting SCI_C1[M]. In 9-bit mode, there is a ninth data bit to the left of the most significant bit of the SCI data register. For the transmit data buffer, this bit is stored in T8 in SCI_C3. For the receiver, the ninth bit is held in SCI_C3[R8].

For coherent writes to the transmit data buffer, write to SCI_C3[T8] before writing to SCI_D.

If the bit value to be transmitted as the ninth bit of a new character is the same as for the previous character, it is not necessary to write to SCI_C3[T8] again. When data is transferred from the transmit data buffer to the transmit shifter, the value in SCI_C3[T8] is copied at the same time data is transferred from SCI_D to the shifter.

The 9-bit data mode is typically used with parity to allow eight bits of data plus the parity in the ninth bit, or it is used with address-mark wake-up so the ninth data bit can serve as the wakeup bit. In custom protocols, the ninth bit can also serve as a software-controlled marker.

### 15.4.6.2 Stop mode operation

During all stop modes, clocks to the SCI module are halted.

No SCI module registers are affected in Stop mode.

The receive input active edge detect circuit remains active in Stop mode. An active edge on the receive input brings the CPU out of Stop mode if the interrupt is not masked (SCI_BDH[RXEDGIE] = 1).

Because the clocks are halted, the SCI module resumes operation upon exit from stop, only in Stop mode. Software must ensure stop mode is not entered while there is a character (including preamble, break and normal data) being transmitted out of or received into the SCI module, that means SCI_S1[TC] =1, SCI_S1[TDRE] = 1, and SCI_S2[RAF] = 0 must all meet before entering stop mode.

#### 15.4.6.3 Loop mode

When SCI_C1[LOOPS] is set, the SCI_C1[RSRC] bit in the same register chooses between loop mode (SCI_C1[RSRC] = 0) or single-wire mode (SCI_C1[RSRC] = 1). Loop mode is sometimes used to check software, independent of connections in the external system, to help isolate system problems. In this mode, the internal loop back connection from the transmitter to the receiver causes the receiver to receive characters that are sent out by the transmitter.

#### 15.4.6.4 Single-wire operation

When SCI_C1[LOOPS] is set, SCI_C1[RSRC] chooses between loop mode (SCI_C1[RSRC] = 0) or single-wire mode (SCI_C1[RSRC] = 1). Single-wire mode implements a half-duplex serial connection. The receiver is internally connected to the transmitter output and to the TxD pin. The RxD pin is not used and reverts to a general-purpose port I/O pin.

In single-wire mode, the SCI_C3[TXDIR] bit controls the direction of serial data on the TxD pin. When SCI_C3[TXDIR] is cleared, the TxD pin is an input to the SCI receiver and the transmitter is temporarily disconnected from the TxD pin so an external device can send serial data to the receiver. When SCI_C3[TXDIR] is set, the TxD pin is an output driven by the transmitter. In single-wire mode, the transmitter output is internally connected to the receiver input and the RxD pin is not used by the SCI, so it reverts to a general-purpose port I/O pin.

# Chapter 16
# Analog-to-digital converter (ADC)

## 16.1 Introduction

The 10-bit analog-to-digital converter (ADC) is a successive approximation ADC designed for operation within an integrated microcontroller system-on-chip.

### 16.1.1 Features

Features of the ADC module include:

- Linear Successive Approximation algorithm with 8-, 10-bit resolution
- Up to 12 external analog inputs, external pin inputs, and 5 internal analog inputs including internal bandgap, temperature sensor, and references
- Output formatted in 8-, 10-bit right-justified unsigned format
- Single or Continuous Conversion (automatic return to idle after single conversion)
- Support up to eight result FIFO with selectable FIFO depth
- Configurable sample time and conversion speed/power
- Conversion complete flag and interrupt
- Input clock selectable from up to four sources
- Operation in Wait or Stop modes for lower noise operation
- Asynchronous clock source for lower noise operation
- Selectable asynchronous hardware conversion trigger
- Automatic compare with interrupt for less-than, or greater-than or equal-to, programmable value

## 16.1.2 Block Diagram

This figure provides a block diagram of the ADC module.

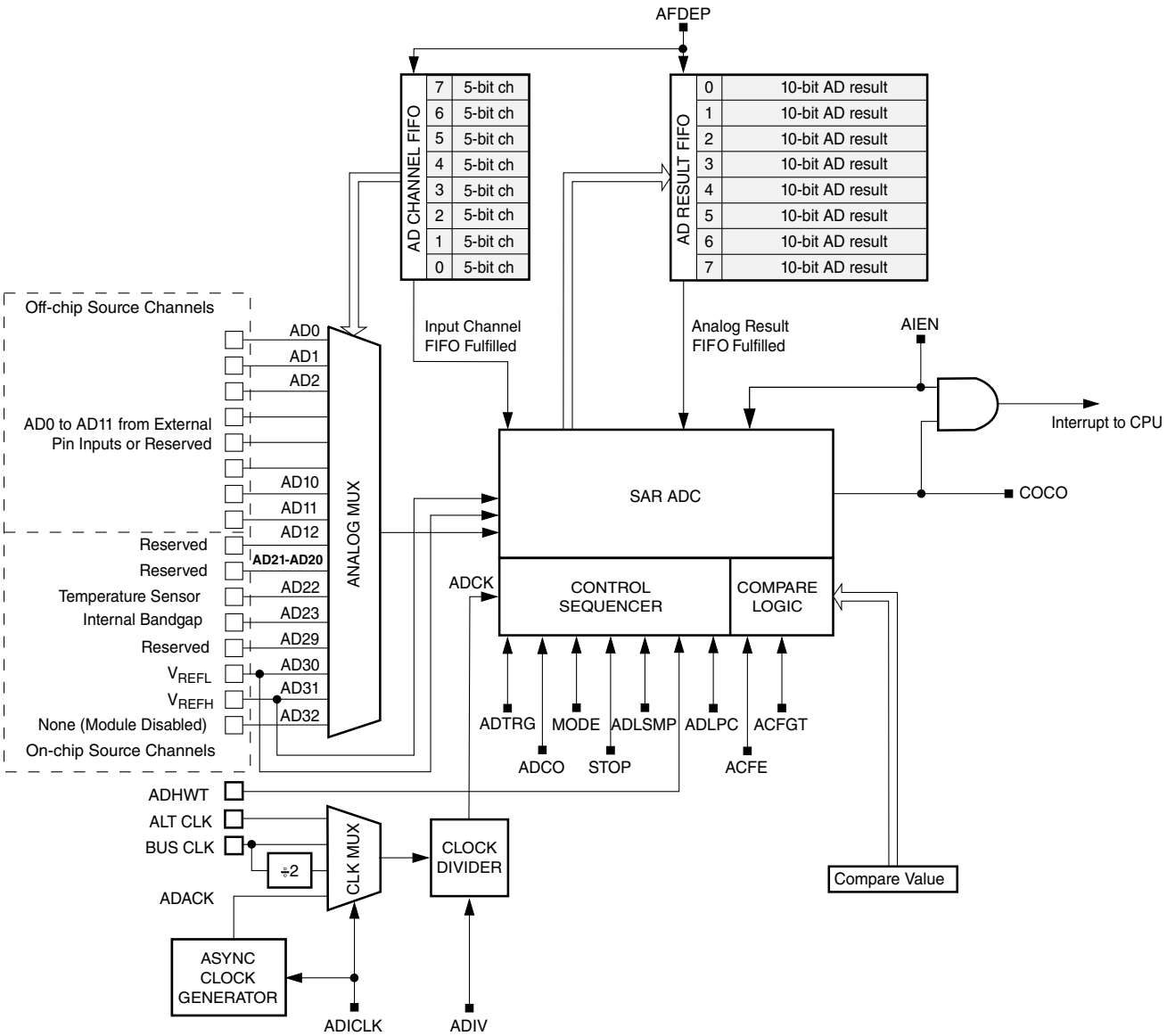



**Figure 16-1. ADC Block Diagram**

## 16.2 External Signal Description

The ADC module supports up to 24 separate analog inputs. It also requires four supply/ reference/ground connections.

**Table 16-1. Signal Properties**

| Name | Function |
|---|---|
| AD23–AD0 | Analog Channel inputs |
| $V_{REFH}$ | High reference voltage |
| $V_{REFL}$ | Low reference voltage |
| $V_{DDA}$ | Analog power supply |
| $V_{SSA}$ | Analog ground |

### 16.2.1 Analog Power ($V_{DDA}$)

The ADC analog portion uses $V_{DDA}$ as its power connection. In some packages, $V_{DDA}$ is connected internally to $V_{DD}$. If externally available, connect the $V_{DDA}$ pin to the same voltage potential as $V_{DD}$. External filtering may be necessary to ensure clean $V_{DDA}$ for good results.

### 16.2.2 Analog Ground ($V_{SSA}$)

The ADC analog portion uses $V_{SSA}$ as its ground connection. In some packages, $V_{SSA}$ is connected internally to $V_{SS}$. If externally available, connect the $V_{SSA}$ pin to the same voltage potential as $V_{SS}$.

### 16.2.3 Voltage Reference High ($V_{REFH}$)

$V_{REFH}$ is the high reference voltage for the converter. In some packages, $V_{REFH}$ is connected internally to $V_{DDA}$. If externally available, $V_{REFH}$ may be connected to the same potential as $V_{DDA}$ or may be driven by an external source between the minimum $V_{DDA}$ specified in the data sheet and the $V_{DDA}$ potential ($V_{REFH}$ must never exceed $V_{DDA}$).

### 16.2.4 Voltage Reference Low ($V_{REFL}$)

$V_{REFL}$ is the low-reference voltage for the converter. In some packages, $V_{REFL}$ is connected internally to $V_{SSA}$. If externally available, connect the $V_{REFL}$ pin to the same voltage potential as $V_{SSA}$.

### 16.2.5 Analog Channel Inputs (ADx)

The ADC module supports up to 24 separate analog inputs. An input is selected for conversion through the ADCH channel select bits.

## 16.3 ADC Control Registers

**ADC memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 10 | Status and Control Register 1 (ADC_SC1) | 8 | R/W | 1Fh | 16.3.1/302 |
| 11 | Status and Control Register 2 (ADC_SC2) | 8 | R/W | 08h | 16.3.2/304 |
| 12 | Status and Control Register 3 (ADC_SC3) | 8 | R/W | 00h | 16.3.3/305 |
| 13 | Status and Control Register 4 (ADC_SC4) | 8 | R/W | 00h | 16.3.4/306 |
| 14 | Conversion Result High Register (ADC_RH) | 8 | R | 00h | 16.3.5/307 |
| 15 | Conversion Result Low Register (ADC_RL) | 8 | R | 00h | 16.3.6/308 |
| 16 | Compare Value High Register (ADC_CVH) | 8 | R/W | 00h | 16.3.7/309 |
| 17 | Compare Value Low Register (ADC_CVL) | 8 | R/W | 00h | 16.3.8/309 |
| 30AC | Pin Control 1 Register (ADC_APCTL1) | 8 | R/W | 00h | 16.3.9/310 |
| 30AD | Pin Control 2 Register (ADC_APCTL2) | 8 | R/W | 00h | 16.3.10/311 |

### 16.3.1 Status and Control Register 1 (ADC_SC1)

This section describes the function of the ADC status and control register (ADC_SC1). Writing ADC_SC1 aborts the current conversion and initiates a new conversion (if the ADCH bits are equal to a value other than all 1s).

When FIFO is enabled, the analog input channel FIFO is written via ADCH. The analog input channel queue must be written to ADCH continuously. The resulting FIFO follows the order in which the analog input channel is written. The ADC will start conversion when the input channel FIFO is fulfilled at the depth indicated by the ADC_SC4[AFDEP]. Any write 0x1F to these bits will reset the FIFO and stop the conversion if it is active.

Address: 10h base + 0h offset = 10h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | COCO | AIEN | ADCO | ADCH | | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

## ADC_SC1 field descriptions

| Field | Description |
|---|---|
| 7<br>COCO | Conversion Complete Flag<br><br>Conversion Complete Flag. The COCO flag is a read-only bit set each time a conversion is completed when the compare function is disabled (ADC_SC2[ACFE] = 0). When the compare function is enabled (ADC_SC2[ACFE] = 1), the COCO flag is set upon completion of a conversion only if the compare result is true. When the FIFO function is enabled (ADC_SC4[AFDEP] > 0), the COCO flag is set upon completion of the set of FIFO conversion. This bit is cleared when ADC_SC1 is written or when ADC_RL is read.<br><br>0 Conversion not completed.<br>1 Conversion completed. |
| 6<br>AIEN | Interrupt Enable<br><br>AIEN enables conversion complete interrupts. When COCO becomes set while AIEN is high, an interrupt is asserted.<br><br>0 Conversion complete interrupt disabled.<br>1 Conversion complete interrupt enabled. |
| 5<br>ADCO | Continuous Conversion Enable<br><br>ADCO enables continuous conversions.<br><br>0 One conversion following a write to the ADC_SC1 when software triggered operation is selected, or one conversion following assertion of ADHWT when hardware triggered operation is selected. When the FIFO function is enabled (AFDEP > 0), a set of conversions are triggered.<br>1 Continuous conversions are initiated following a write to ADC_SC1 when software triggered operation is selected. Continuous conversions are initiated by an ADHWT event when hardware triggered operation is selected. When the FIFO function is enabled (AFDEP > 0), a set of conversions are loop triggered. |
| ADCH | Input Channel Select<br><br>The ADCH bits form a 5-bit field that selects one of the input channels.<br><br>00000-01011 AD0-AD11<br>01100-10011 $V_{SS}$<br>10100-10101 Reserved<br>10110 Temperature Sensor<br>10111 Bandgap<br>11000-11100 Reserved<br>11101 $V_{REFH}$<br>11110 $V_{REFL}$<br>11111 Module disabled<br><br>**NOTE:** Reset FIFO in FIFO mode. |

## 16.3.2 Status and Control Register 2 (ADC_SC2)

The ADC_SC2 register controls the compare function, conversion trigger, and conversion active of the ADC module.

Address: 10h base + 1h offset = 11h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ADACT | ADTRG | ACFE | ACFGT | FEMPTY | FFULL | | |
| Write | | | | | | | 0 | |
| Reset | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

**ADC_SC2 field descriptions**

| Field | Description |
|---|---|
| 7<br>ADACT | Conversion Active<br><br>Indicates that a conversion is in progress. ADACT is set when a conversion is initiated and cleared when a conversion is completed or aborted.<br><br>0 Conversion not in progress.<br>1 Conversion in progress. |
| 6<br>ADTRG | Conversion Trigger Select<br><br>Selects the type of trigger used for initiating a conversion. Two types of trigger are selectable: software trigger and hardware trigger. When software trigger is selected, a conversion is initiated following a write to ADC_SC1. When hardware trigger is selected, a conversion is initiated following the assertion of the ADHWT input.<br><br>0 Software trigger selected.<br>1 Hardware trigger selected. |
| 5<br>ACFE | Compare Function Enable<br><br>Enables the compare function.<br><br>0 Compare function disabled.<br>1 Compare function enabled. |
| 4<br>ACFGT | Compare Function Greater Than Enable<br><br>Configures the compare function to trigger when the result of the conversion of the input being monitored is greater than or equal to the compare value. The compare function defaults to triggering when the result of the compare of the input being monitored is less than the compare value.<br><br>0 Compare triggers when input is less than compare level.<br>1 Compare triggers when input is greater than or equal to compare level. |
| 3<br>FEMPTY | Result FIFO empty<br><br>0 Indicates that ADC result FIFO have at least one valid new data.<br>1 Indicates that ADC result FIFO have no valid new data. |
| 2<br>FFULL | Result FIFO full |

*Table continues on the next page...*

**ADC_SC2 field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Indicates that ADC result FIFO is not full and next conversion data still can be stored into FIFO.<br>1 Indicates that ADC result FIFO is full and next conversion will override old data in case of no read action. |
| Reserved | This field is reserved. |

## 16.3.3 Status and Control Register 3 (ADC_SC3)

ADC_SC3 selects the mode of operation, clock source, clock divide, and configure for low power or long sample time.

Address: 10h base + 2h offset = 12h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | ADLPC | ADIV | | ADLSMP | MODE | | ADICLK | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ADC_SC3 field descriptions**

| Field | Description |
|---|---|
| 7<br>ADLPC | Low-Power Configuration<br><br>ADLPC controls the speed and power configuration of the successive approximation converter. This optimizes power consumption when higher sample rates are not required.<br><br>0 High speed configuration.<br>1 Low power configuration:The power is reduced at the expense of maximum clock speed. |
| 6–5<br>ADIV | Clock Divide Select<br><br>ADIV selects the divide ratio used by the ADC to generate the internal clock ADCK.<br><br>00 Divide ration = 1, and clock rate = Input clock.<br>01 Divide ration = 2, and clock rate = Input clock ÷ 2.<br>10 Divide ration = 3, and clock rate = Input clock ÷ 4.<br>11 Divide ration = 4, and clock rate = Input clock ÷ 8. |
| 4<br>ADLSMP | Long Sample Time Configuration<br><br>ADLSMP selects between long and short sample time. This adjusts the sample period to allow higher impedance inputs to be accurately sampled or to maximize conversion speed for lower impedance inputs. Longer sample times can also be used to lower overall power consumption when continuous conversions are enabled if high conversion rates are not required.<br><br>0 Short sample time.<br>1 Long sample time. |
| 3–2<br>MODE | Conversion Mode Selection<br><br>MODE bits are used to select between 10-, or 8-bit operation.<br><br>00 8-bit conversion (N=8) |

*Table continues on the next page...*

**ADC_SC3 field descriptions (continued)**

| Field | Description |
|---|---|
| | 01 10-bit conversion (N=10)<br>10 Reserved<br>11 Reserved |
| ADICLK | Input Clock Select<br><br>ADICLK bits select the input clock source to generate the internal clock ADCK.<br><br>00 Bus clock<br>01 Bus clock divided by 2<br>10 Alternate clock (ALTCLK)<br>11 Asynchronous clock (ADACK) |

## 16.3.4 Status and Control Register 4 (ADC_SC4)

This register controls the FIFO scan mode, FIFO compare function and FIFO depth selection of the ADC module.

Address: 10h base + 3h offset = 13h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | ASCANE | ACFSEL | 0 | | AFDEP | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ADC_SC4 field descriptions**

| Field | Description |
|---|---|
| 7<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 6<br>ASCANE | FIFO Scan Mode Enable<br><br>The FIFO always use the first dummied FIFO channels when it is enabled. When this bit is set and FIFO function is enabled, ADC will repeat using the first FIFO channel as the conversion channel until the result FIFO is fulfilled. In continuous mode (ADCO = 1), ADC will start next conversion with the same channel when COCO is set.<br><br>0 FIFO scan mode disabled.<br>1 FIFO scan mode enabled. |
| 5<br>ACFSEL | Compare function select OR/AND when the FIFO function is enabled (AFDEP > 0). When this field is cleared, ADC will OR all of compare triggers and set COCO after at least one of compare trigger occurs. When this field is set, ADC will AND all of compare triggers and set COCO after all of compare triggers occur.<br><br>0 OR all of compare trigger.<br>1 AND all of compare trigger. |
| 4–3<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

*Table continues on the next page...*

**ADC_SC4 field descriptions (continued)**

| Field | Description |
|---|---|
| AFDEP | FIFO Depth enables the FIFO function and sets the depth of FIFO. When AFDEP is cleared, the FIFO is disabled. When AFDEP is set to nonzero, the FIFO function is enabled and the depth is indicated by the AFDEP bits. The ADCH in ADC_SC1 and ADC_RH:ADC_RL must be accessed by FIFO mode when FIFO function is enabled. ADC starts conversion when the analog channel FIFO is upon the level indicated by AFDEP bits. The COCO bit is set when the set of conversions are completed and the result FIFO is upon the level indicated by AFDEP bits.<br>**NOTE:** The bus clock frequency must be at least double the ADC clock when FIFO mode is enabled. It means, if ICS FBE mode is used, the ADC clock can not be ADACK.<br>000 FIFO is disabled.<br>001 2-level FIFO is enabled.<br>010 3-level FIFO is enabled..<br>011 4-level FIFO is enabled.<br>100 5-level FIFO is enabled.<br>101 6-level FIFO is enabled.<br>110 7-level FIFO is enabled.<br>111 8-level FIFO is enabled. |

## 16.3.5 Conversion Result High Register (ADC_RH)

In 10-bit mode, ADC_RH contains the upper two bits of the result of a 10-bit conversion.

ADC_RH is updated each time a conversion completes except when automatic compare is enabled and the compare condition is not met. Reading ADC_RH prevents the ADC from transferring subsequent conversion results into the result registers until ADC_RL is read. If ADC_RL is not read until after the next conversion is completed, the intermediate conversion result is lost. In 8-bit mode, there is no interlocking with ADC_RL.

When FIFO is enabled, the result FIFO is read via ADC_RH:ADC_RL. The ADC conversion completes when the input channel FIFO is fulfilled at the depth indicated by the AFDEP. The AD result FIFO can be read via ADC_RH:ADC_RL continuously by the order set in analog input channel ADCH.

If the MODE bits are changed, any data in ADC_RH becomes invalid.

Address: 10h base + 4h offset = 14h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | | | | ADR | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ADC_RH field descriptions**

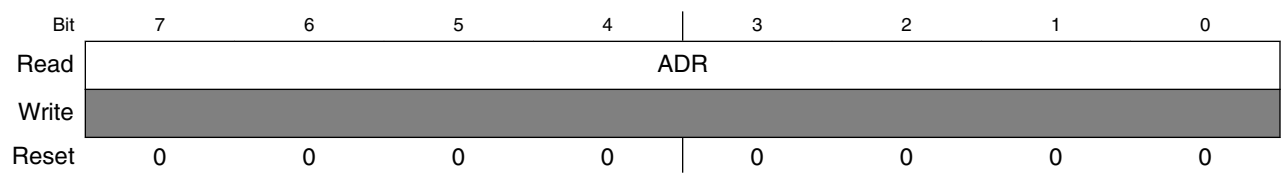
| Field | Description |
|---|---|
| 7–2<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| ADR | Conversion Result[9:8] |

## 16.3.6 Conversion Result Low Register (ADC_RL)

ADC_RL contains the lower eight bits of the result of a 10-bit conversion. This register is updated each time a conversion completes except when automatic compare is enabled and the compare condition is not met. In 10-bit mode, reading ADC_RH prevents the ADC from transferring subsequent conversion results into the result registers until ADC_RL is read. If ADC_RL is not read until the next conversion is completed, the intermediate conversion results are lost. In 8-bit mode, there is no interlocking with ADC_RH. If the MODE bits are changed, any data in ADC_RL becomes invalid.

When FIFO is enabled, the result FIFO is read via ADC_RH:ADC_RL. The ADC conversion completes when the input channel FIFO is fulfilled at the depth indicated by the AFDEP. The AD result FIFO can be read via ADC_RH:ADC_RL continuously by the order set in analog input channel FIFO.

Address: 10h base + 5h offset = 15h




**ADC_RL field descriptions**

| Field | Description |
|---|---|
| ADR | Conversion Result[7:0] |

## 16.3.7 Compare Value High Register (ADC_CVH)

In 10-bit mode, this register holds the upper two bits of the 10-bit compare value. These bits are compared to the upper two bits of the result following a conversion in 10-bit mode when the compare function is enabled.

Address: 10h base + 6h offset = 16h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | | | | CV | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ADC_CVH field descriptions**

| Field | Description |
|---|---|
| 7–2<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| CV | Conversion Result[9:8] |

## 16.3.8 Compare Value Low Register (ADC_CVL)

This register holds the lower 8 bits of the 10-bit compare value. Bits CV7:CV0 are compared to the lower 8 bits of the result following a conversion in 10-bit mode.

Address: 10h base + 7h offset = 17h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | CV | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ADC_CVL field descriptions**

| Field | Description |
|---|---|
| CV | Conversion Result[7:0] |

## 16.3.9 Pin Control 1 Register (ADC_APCTL1)

The pin control registers disable the I/O port control of MCU pins used as analog inputs. APCTL1 is used to control the pins associated with channels 0-7 of the ADC module.

Address: 10h base + 309Ch offset = 30ACh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | ADPC7 | ADPC6 | ADPC5 | ADPC4 | ADPC3 | ADPC2 | ADPC1 | ADPC0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ADC_APCTL1 field descriptions**

| Field | Description |
|---|---|
| 7<br>ADPC7 | ADC Pin Control 7<br><br>ADPC7 controls the pin associated with channel AD7.<br><br>0 AD7 pin I/O control enabled.<br>1 AD7 pin I/O control disabled. |
| 6<br>ADPC6 | ADC Pin Control 6<br><br>ADPC6 controls the pin associated with channel AD6.<br><br>0 AD6 pin I/O control enabled.<br>1 AD6 pin I/O control disabled. |
| 5<br>ADPC5 | ADC Pin Control 5<br><br>ADPC5 controls the pin associated with channel AD5.<br><br>0 AD5 pin I/O control enabled.<br>1 AD5 pin I/O control disabled. |
| 4<br>ADPC4 | ADC Pin Control 4<br><br>ADPC4 controls the pin associated with channel AD4.<br><br>0 AD4 pin I/O control enabled.<br>1 AD4 pin I/O control disabled. |
| 3<br>ADPC3 | ADC Pin Control 3<br><br>ADPC3 controls the pin associated with channel AD3.<br><br>0 AD3 pin I/O control enabled.<br>1 AD3 pin I/O control disabled. |
| 2<br>ADPC2 | ADC Pin Control 2<br><br>ADPC2 controls the pin associated with channel AD2.<br><br>0 AD2 pin I/O control enabled.<br>1 AD2 pin I/O control disabled. |

*Table continues on the next page...*

**ADC_APCTL1 field descriptions (continued)**

| Field | Description |
|---|---|
| 1<br>ADPC1 | ADC Pin Control 1<br><br>ADPC1 controls the pin associated with channel AD1.<br><br>0 AD1 pin I/O control enabled.<br>1 AD1 pin I/O control disabled. |
| 0<br>ADPC0 | ADC Pin Control 0<br><br>ADPC0 controls the pin associated with channel AD0.<br><br>0 AD0 pin I/O control enabled.<br>1 AD0 pin I/O control disabled. |

## 16.3.10 Pin Control 2 Register (ADC_APCTL2)

APCTL2 controls channels 8-15 of the ADC module.

Address: 10h base + 309Dh offset = 30ADh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | Reserved | | | | ADPC11 | ADPC10 | ADPC9 | ADPC8 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**ADC_APCTL2 field descriptions**

| Field | Description |
|---|---|
| 7–4<br>Reserved | This field is reserved. |
| 3<br>ADPC11 | ADC Pin Control 11<br><br>ADPC11 controls the pin associated with channel AD11.<br><br>0 AD11 pin I/O control enabled.<br>1 AD11 pin I/O control disabled. |
| 2<br>ADPC10 | ADC Pin Control 10<br><br>ADPC10 controls the pin associated with channel AD10.<br><br>0 AD10 pin I/O control enabled.<br>1 AD10 pin I/O control disabled. |
| 1<br>ADPC9 | ADC Pin Control 9<br><br>ADPC9 controls the pin associated with channel AD1.<br><br>0 AD9 pin I/O control enabled.<br>1 AD9 pin I/O control disabled. |
| 0<br>ADPC8 | ADC Pin Control 8 |

*Table continues on the next page...*

**ADC_APCTL2 field descriptions (continued)**

| Field | Description |
|---|---|
| | ADPC8 controls the pin associated with channel AD8.<br><br>0 AD8 pin I/O control enabled.<br>1 AD8 pin I/O control disabled. |

## 16.4 Functional description

The ADC module is disabled during reset or when the ADC_SC1[ADCH] bits are all high. The module is idle when a conversion has completed and another conversion has not been initiated. When idle, the module is in its lowest power state.

The ADC can perform an analog-to-digital conversion on any of the software selectable channels. In 10-bit mode, the selected channel voltage is converted by a successive approximation algorithm into a 10-bit digital result. In 8-bit mode, the selected channel voltage is converted by a successive approximation algorithm into a 8-bit digital result.

When the conversion is completed, the result is placed in the data registers (ADC_RH and ADC_RL). In 10-bit mode, the result is rounded to 10 bits and placed in the data registers (ADC_RH and ADC_RL). In 8-bit mode, the result is rounded to 8 bits and placed in ADC_RL. The conversion complete flag (ADC_SC1[COCO]) is then set and an interrupt is generated if the conversion complete interrupt has been enabled (ADC_SC1[AIEN] = 1).

The ADC module has the capability of automatically comparing the result of a conversion with the contents of its compare registers. The compare function is enabled by setting the ADC_SC2[ACFE] bit and operates with any of the conversion modes and configurations.

### 16.4.1 Clock select and divide control

One of four clock sources can be selected as the clock source for the ADC module. This clock source is then divided by a configurable value to generate the input clock to the converter (ADCK). The clock is selected from one of the following sources by means of the ADC_SC3[ADICLK] bits.

- The bus clock, which is equal to the frequency at which software is executed. This is the default selection following reset.

- The bus clock divided by 2: For higher bus clock rates, this allows a maximum divide by 16 of the bus clock.
- ALTCLK, that is, alternate clock which is OSCOUT
- The asynchronous clock (ADACK): This clock is generated from a clock source within the ADC module. When selected as the clock source, this clock remains active while the MCU is in Wait or Stop mode and allows conversions in these modes for lower noise operation.

Whichever clock is selected, its frequency must fall within the specified frequency range for ADCK. If the available clocks are too slow, the ADC does not perform according to specifications. If the available clocks are too fast, the clock must be divided to the appropriate frequency. This divider is specified by the ADC_SC3[ADIV] bits and can be divide-by 1, 2, 4, or 8.

### 16.4.2 Input select and pin control

The Pin Control registers ( ADC_APCTL2 and ADC_APCTL1) disables the I/O port control of the pins used as analog inputs. When a pin control register bit is set, the following conditions are forced for the associated MCU pin:

- The output buffer is forced to its high impedance state.
- The input buffer is disabled. A read of the I/O port returns a zero for any pin with its input buffer disabled.
- The pullup is disabled.

### 16.4.3 Hardware trigger

The ADC module has a selectable asynchronous hardware conversion trigger, ADHWT, that is enabled when the ADC_SC2[ADTRG] bit is set. This source is not available on all MCUs. See the module introduction for information on the ADHWT source specific to this MCU.

When ADHWT source is available and hardware trigger is enabled ( ADC_SC2[ADTRG] = 1), a conversion is initiated on the rising edge of ADHWT. If a conversion is in progress when a rising edge occurs, the rising edge is ignored. In continuous convert configuration, only the initial rising edge to launch continuous conversions is observed. The hardware trigger function operates in conjunction with any of the conversion modes and configurations.

### 16.4.4 Conversion control

Conversions can be performed in 10-bit mode, or 8-bit mode as determined by the ADC_SC3[MODE] bits. Conversions can be initiated by a software or hardware trigger. In addition, the ADC module can be configured for low power operation, long sample time, continuous conversion, and an automatic compare of the conversion result to a software determined compare value.

#### 16.4.4.1 Initiating conversions

A conversion initiates under the following conditions:

- A write to ADC_SC1 or a set of write to ADC_SC1 in FIFO mode (with ADCH bits not all 1s) if software triggered operation is selected.
- A hardware trigger (ADHWT) event if hardware triggered operation is selected.
- The transfer of the result to the data registers when continuous conversion is enabled.

If continuous conversions are enabled, a new conversion is automatically initiated after the completion of the current conversion. In software triggered operation, continuous conversions begin after ADC_SC1 is written and continue until aborted. In hardware triggered operation, continuous conversions begin after a hardware trigger event and continue until aborted.

#### 16.4.4.2 Completing conversions

A conversion is completed when the result of the conversion is transferred into the data result registers, ADC_RH and ADC_RL. This is indicated by the setting of ADC_SC1[COCO]. An interrupt is generated if ADC_SC1[AIEN] is high at the time that ADC_SC1[COCO] is set.

A blocking mechanism prevents a new result from overwriting previous data in ADC_RH and ADC_RL if the previous data is in the process of being read while in 10-bit MODE (the ADC_RH register has been read but the ADC_RL register has not). When blocking is active, the data transfer is blocked, ADC_SC1[COCO] is not set, and the new result is lost. In the case of single conversions with the compare function enabled and the compare condition false, blocking has no effect and ADC operation is terminated. In all other

cases of operation, when a data transfer is blocked, another conversion is initiated regardless of the state of ADC_SC1[ADCO] whether single or continuous conversions are enabled.

If single conversions are enabled, the blocking mechanism could result in several discarded conversions and excess power consumption. To avoid this issue, the data registers must not be read after initiating a single conversion until the conversion completes.

In fifo mode, a blocking mechanism will keep current channel conversion and no channel fifo and result fifo switching until a block mechanism is released.

#### 16.4.4.3 Aborting conversions

Any conversion in progress is aborted in the following cases:

- A write to ADC_SC1 occurs.
  - The current conversion will be aborted and a new conversion will be initiated, if ADC_SC1[ADCH] are not all 1s and ADC_SC4[AFDEP] are all 0s.
  - The current conversion and the rest of conversions will be aborted and no new conversion will be initialed, if ADC_SC4[AFDEP] are not all 0s.
  - A new conversion will be initiated when the FIFO is re-fulfilled upon the levels indicated by the ADC_SC4[AFDEP] bits).
- A write to ADC_SC2, ADC_SC3, ADC_SC4, ADC_CVH, or ADC_CVL occurs. This indicates a mode of operation change has occurred and the current and rest of conversions (when ADC_SC4[AFDEP] are not all 0s) are therefore invalid.
- The MCU is reset.
- The MCU enters Stop mode with ADACK not enabled.

When a conversion is aborted, the contents of the data registers, ADC_RH and ADC_RL, are not altered. However, they continue to be the values transferred after the completion of the last successful conversion. If the conversion was aborted by a reset, ADC_RH and ADC_RL return to their reset states.

#### 16.4.4.4 Power control

The ADC module remains in its idle state until a conversion is initiated. If ADACK is selected as the conversion clock source, the ADACK clock generator is also enabled.

Power consumption when active can be reduced by setting ADC_SC3[ADLPC]. This results in a lower maximum value for $f_{ADCK}$ (see the data sheet).

### 16.4.4.5 Sample time and total conversion time

The total conversion time depends on the sample time (as determined by ADC_SC3[ADLSMP]), the MCU bus frequency, the conversion mode (8-bit, 10-bit), and the frequency of the conversion clock ($f_{ADCK}$). After the module becomes active, sampling of the input begins.ADC_SC3[ADLSMP] selects between short (3.5 ADCK cycles) and long (23.5 ADCK cycles) sample times. When sampling is complete, the converter is isolated from the input channel and a successive approximation algorithm is performed to determine the digital value of the analog signal. The result of the conversion is transferred to ADC_RH and ADC_RL upon completion of the conversion algorithm.

If the bus frequency is less than the $f_{ADCK}$ frequency, precise sample time for continuous conversions cannot be guaranteed when short sample is enabled (ADC_SC3[ADLSMP] = 0). If the bus frequency is less than 1/11th of the $f_{ADCK}$ frequency, precise sample time for continuous conversions cannot be guaranteed when long sample is enabled (ADC_SC3[ADLSMP] = 1).

The maximum total conversion time for different conditions is summarized in the table below.

**Table 16-2. Total conversion time vs. control conditions**

| Conversion type | ADICLK | ADLSMP | Max total conversion time |
|---|---|---|---|
| Single or first continuous 8-bit | 0x, 10 | 0 | 20 ADCK cycles + 5 bus clock cycles |
| Single or first continuous 10-bit | 0x, 10 | 0 | 23 ADCK cycles + 5 bus clock cycles |
| Single or first continuous 8-bit | 0x, 10 | 1 | 40 ADCK cycles + 5 bus clock cycles |
| Single or first continuous 10-bit | 0x, 10 | 1 | 43 ADCK cycles + 5 bus clock cycles |
| Single or first continuous 8-bit | 11 | 0 | 5 µs + 20 ADCK + 5 bus clock cycles |
| Single or first continuous 10-bit | 11 | 0 | 5 µs + 23 ADCK + 5 bus clock cycles |
| Single or first continuous 8-bit | 11 | 1 | 5 µs + 40 ADCK + 5 bus clock cycles |
| Single or first continuous 10-bit | 11 | 1 | 5 µs + 43 ADCK + 5 bus clock cycles |
| Subsequent continuous 8-bit; $f_{BUS} > f_{ADCK}$ | xx | 0 | 17 ADCK cycles |
| Subsequent continuous 10-bit; $f_{BUS} > f_{ADCK}$ | xx | 0 | 20 ADCK cycles |
| Subsequent continuous 8-bit; $f_{BUS} > f_{ADCK}/11$ | xx | 1 | 37 ADCK cycles |
| Subsequent continuous 10-bit; $f_{BUS} > f_{ADCK}/11$ | xx | 1 | 40 ADCK cycles |

The maximum total conversion time is determined by the selected clock source and the divide ratio. The clock source is selectable by the ADC_SC3[ADICLK] bits, and the divide ratio is specified by the ADC_SC3[ADIV] bits. For example, in 10-bit mode, with the bus clock selected as the input clock source, the input clock divide-by-1 ratio selected, and a bus frequency of 8 MHz, then the conversion time for a single conversion as given below:

$$\text{Conversion time} = \frac{23\,\text{ADCK Cyc}}{8\,\text{MHz}/1} + \frac{5\,\text{bus Cyc}}{8\,\text{MHz}} = 3.5\,\mu\text{s}$$

The number of bus cycles at 8 MHz is:

$$\text{Bus cycles} = 3.5\mu\text{s} \times 8\text{MHz} = 28$$

**Note**

The ADCK frequency must be between $f_{ADCK}$ minimum and $f_{ADCK}$ maximum to meet ADC specifications.

## 16.4.5 Automatic compare function

The compare function can be configured to check for an upper or lower limit. After the input is sampled and converted, the result is added to the two's complement of the compare value (ADC_CVH and ADC_CVL). When comparing to an upper limit (ADC_SC2[ACFGT] = 1), if the result is greater-than or equal-to the compare value, ADC_SC1[COCO] is set. When comparing to a lower limit (ADC_SC2[ACFGT] = 0), if the result is less than the compare value, ADC_SC1[COCO] is set. The value generated by the addition of the conversion result and the two's complement of the compare value is transferred to ADC_RH and ADC_RL.

On completion of a conversion while the compare function is enabled, if the compare condition is not true, ADC_SC1[COCO] is not set and no data is transferred to the result registers. An ADC interrupt is generated on the setting of ADC_SC1[COCO] if the ADC interrupt is enabled (ADC_SC1[AIEN] = 1).

On completion of all conversions while the compare function is enabled and FIFO enabled, if none of the compare conditions are true when ADC_SC4[ACFSEL] is low or if not all of compare conditions are true when ADC_SC4[ACFSEL] is high, ADC_SC1[COCO] is not set. The compare data are transferred to the result registers regardless of compare condition true or false when FIFO enabled.

**Note**

The compare function can monitor the voltage on a channel while the MCU is in Wait or Stop mode. The ADC interrupt wakes the MCU when the compare condition is met.

**Note**

The compare function can not work in continuous conversion mode when FIFO enabled.

### 16.4.6 FIFO operation

The ADC module supports FIFO operation to minimize the interrupts to CPU in order to reduce CPU loading in ADC interrupt service routines. This module contains two FIFOs to buffer analog input channels and analog results respectively.

The FIFO function is enabled when the ADC_SC4[AFDEP] bits are set non-zero. The FIFO depth is indicated by these bits. The FIFO supports up to eight level buffer.

The analog input channel FIFO is accessed by ADC_SC1[ADCH] bits, when FIFO function is enabled. The analog channel must be written to this FIFO in order. The ADC will not start the conversion if the channel FIFO is fulfilled below the level indicated by the ADC_SC4[AFDEP] bits, no matter whether software or hardware trigger is set. Read ADC_SC1[ADCH] will read the current active channel value. Write to ADC_SC1[ADCH] will re-fill channel FIFO to initial new conversion. It will abort current conversion and any other conversions that did not start. Write to the ADC_SC1 after all the conversions are completed or ADC is in idle state.

The result of the FIFO is accessed by ADC_RH:ADC_RL registers, when FIFO function is enabled. The result must be read via these two registers by the same order of analog input channel FIFO to get the proper results. Don't read ADC_RH:ADC_RL until all of the conversions are completed in FIFO mode. The ADC_SC1[COCO] bit will be set only when all conversions indicated by the analog input channel FIFO complete whatever software or hardware trigger is set. An interrupt request will be submitted to CPU if the ADC_SC1[AIEN] is set when the FIFO conversion completes and the ADC_SC1[COCO] bit is set.

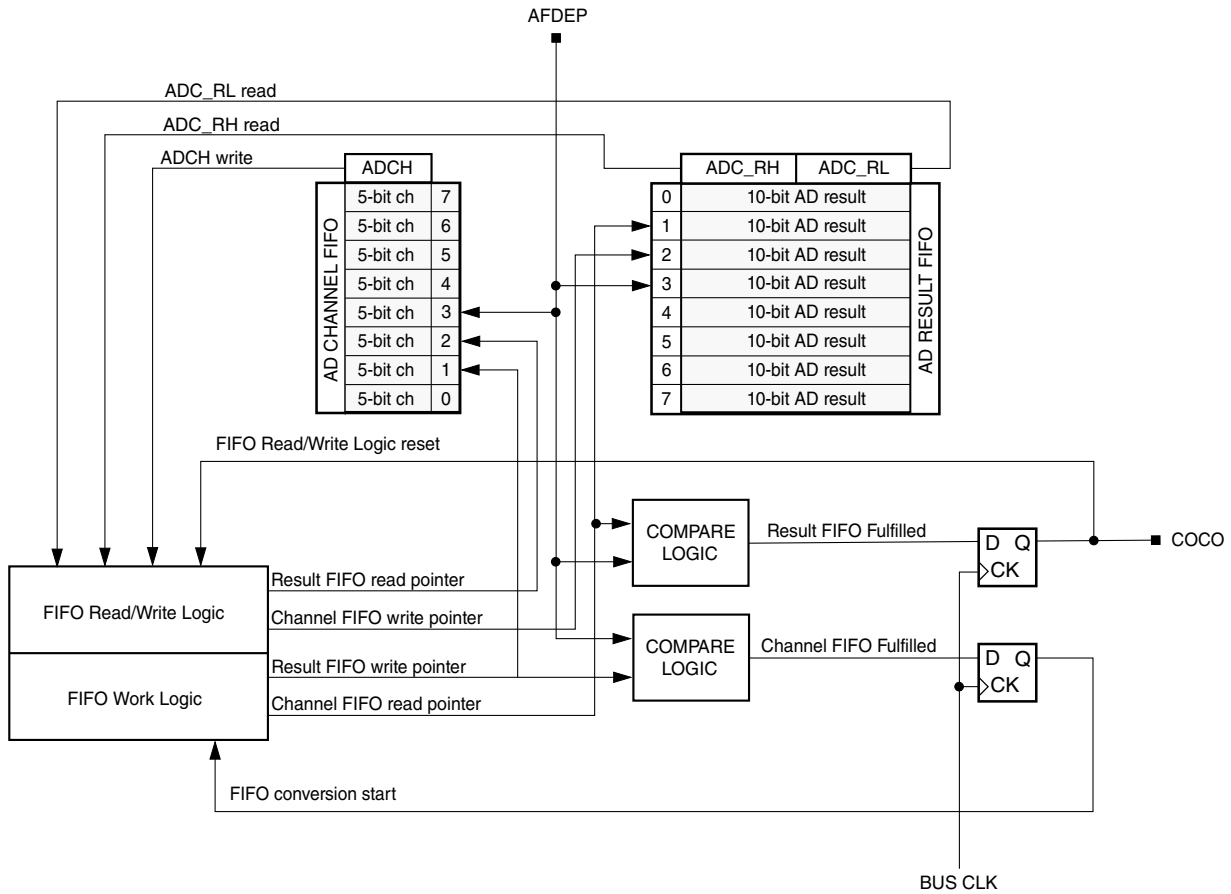



**Figure 16-2. FADC FIFO structure**

If software trigger is enabled, the next analog channel is fetched from analog input channel FIFO as soon as a conversion completes and its result is stored in the result FIFO. When all conversions set in the analog input channel FIFO completes, the ADC_SC1[COCO] bit is set and an interrupt request will be submitted to CPU if the ADC_SC1[AIEN] bit is set.

If hardware trigger mode is enabled, the next analog is fetched from analog input channel FIFO only when this conversion completes, its result is stored in the result FIFO, and the next hardware trigger is fed to ADC module. When all conversions set in the analog input channel FIFO completes, the ADC_SC1[COCO] bit is set and an interrupt request will be submitted to CPU if the ADC_SC1[AIEN] bit is set.

In single conversion in which ADC_SC1[ADCO] bit is clear, the ADC stops conversions when ADC_SC1[COCO] bit is set until the channel FIFO is fulfilled again or new hardware trigger occur.

The FIFO also provides scan mode to simplify the dummy work of input channel FIFO. When the ADC_SC4[ASCANE] bit is set in FIFO mode, the FIFO will always use the first dummied channel in spite of the value in the input channel FIFO. The ADC conversion start to work in FIFO mode as soon as the first channel is dummied. The

following write operation to the input channel FIFO will cover the first channel element in this FIFO. In scan FIFO mode, the ADC_SC1[COCO] bit is set when the result FIFO is fulfilled according to the depth indicated by the ADC_SC4[AFDEP] bits.

In continuous conversion in which the ADC_SC1[ADCO] bit is set, the ADC starts next conversion immediately when all conversions are completed. ADC module will fetch the analog input channel from the beginning of analog input channel FIFO.

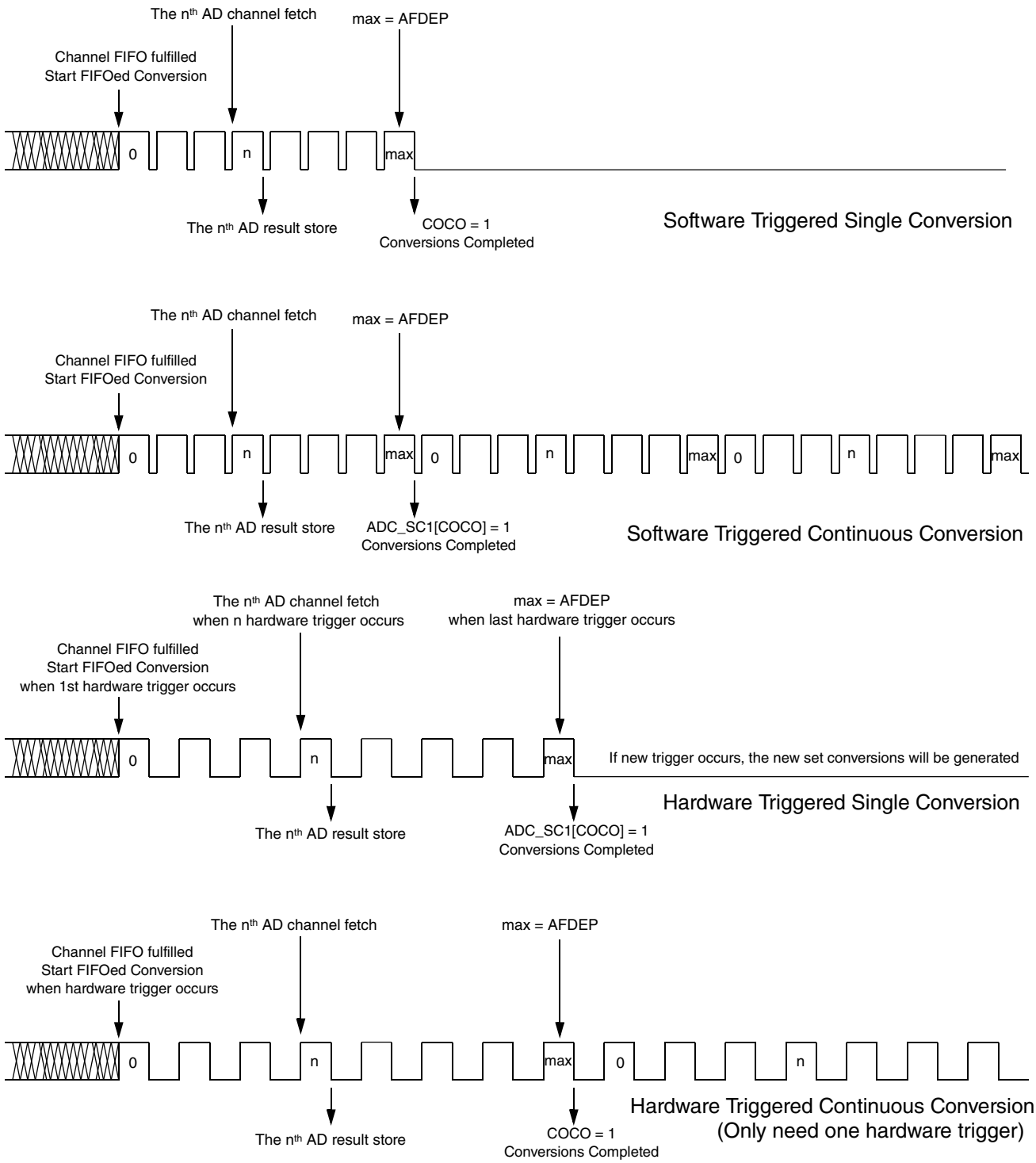



**Figure 16-3. ADC FIFO conversion sequence**

### 16.4.7 MCU wait mode operation

Wait mode is a low-power consumption standby mode from which recovery is fast because the clock sources remain active. If a conversion is in progress when the MCU enters wait mode, it continues until completion. Conversions can be initiated while the MCU is in wait mode by means of the hardware trigger or if continuous conversions are enabled.

The bus clock, bus clock divided by two, ALTCLK and ADACK are available as conversion clock sources while in wait mode.

ADC_SC1[COCO] is set by a conversion complete event that generates an ADC interrupt to wake the MCU from wait mode if the ADC interrupt is enabled (ADC_SC1[AIEN] = 1).

### 16.4.8 MCU Stop mode operation

Stop mode is a low-power consumption standby mode during which most or all clock sources on the MCU are disabled.

#### 16.4.8.1 Stop mode with ADACK disabled

If the asynchronous clock, ADACK, is not selected as the conversion clock, executing a STOP instruction aborts the current conversion and places the ADC in its idle state. The contents of ADC_RH and ADC_RL are unaffected by Stop mode. After exiting from Stop mode, a software or hardware trigger is required to resume conversions.

#### 16.4.8.2 Stop mode with ADACK enabled

If ADACK is selected as the conversion clock, the ADC continues operation during Stop mode. For guaranteed ADC operation, the MCU's voltage regulator must remain active during Stop mode. See the module introduction for configuration information for this MCU.

If a conversion is in progress when the MCU enters Stop mode, it continues until completion. Conversions can be initiated while the MCU is in Stop mode by means of the hardware trigger or if continuous conversions are enabled.

A conversion complete event sets the ADC_SC1[COCO] and generates an ADC interrupt to wake the MCU from Stop mode if the ADC interrupt is enabled (ADC_SC1[AIEN] = 1). In fifo mode, ADC cannot complete the conversion operation fully or wake the MCU from Stop mode.

**Note**

The ADC module can wake the system from low-power stop and cause the MCU to begin consuming run-level currents without generating a system level interrupt. To prevent this scenario, the data transfer blocking mechanism must be cleared when entering Stop and continuing ADC conversions.

## 16.5 Initialization information

This section gives an example that provides some basic direction on how to initialize and configure the ADC module. You can configure the module for 8-, 10-bit resolution, single or continuous conversion, and a polled or interrupt approach, among many other options. Refer to ADC_SC3 register for information used in this example.

**Note**

Hexadecimal values prefixed by a 0x, binary values prefixed by a %, and decimal values have no preceding character.

### 16.5.1 ADC module initialization example

Before the ADC module can be used to complete conversions, it must be initialized. Given below is a method to initialize ADC module.

#### 16.5.1.1 Initialization sequence

A typical initialization sequence is as follows:

1. Update the configuration register (ADC_SC3) to select the input clock source and the divide ratio used to generate the internal clock, ADCK. This register is also used for selecting sample time and low-power configuration.
2. Update status and control register 2 (ADC_SC2) to select the hardware or software conversion trigger and compare function options, if enabled.

3. Update status and control register 1 (ADC_SC1) to select whether conversions will be continuous or completed only once, and to enable or disable conversion complete interrupts. The input channel on which conversions will be performed is also selected here.

### 16.5.1.2 Pseudo-code example

In this example, the ADC module is set up with interrupts enabled to perform a single 10-bit conversion at low power with a long sample time on input channel 1, where the internal ADCK clock is derived from the bus clock divided by 1.

#### Example: 16.5.1.2.1 General ADC initialization routine

```
void ADC_init(void)
{
      /* The following code segment demonstrates how to initialize ADC by low-power mode,
long
      sample time, bus frequency, software triggered from AD1 external pin without FIFO
enabled
      */
 ADC_APCTL1 = ADC_APCTL1_ADPC1_MASK;
      ADC_SC3 = ADC_SC3_ADLPC_MASK | ADC_SC3_ADLSMP_MASK | ADC_SC3_MODE0_MASK;
      ADC_SC2 = 0x00;
      ADC_SC1 = ADC_SC1_AIEN_MASK | ADC_SC1_ADCH0_MASK;
}
```

## 16.5.2 ADC FIFO module initialization example

Before the ADC module can be used to start FIFOed conversions, an initialization procedure must be performed. A typical sequence is as follows:

1. Update the configuration register (ADC_SC3) to select the input clock source and the divide ratio used to generate the internal clock, ADCK. This register is also used to select sample time and low-power configuration.
2. Update the configuration register (ADC_SC4) to select the FIFO scan mode, FIFO compare function selection (OR or AND function) and FIFO depth.
3. Update status and control register 2 (ADC_SC2) to select the hardware or software conversion trigger, compare function options if enabled.
4. Update status and control register 1 (ADC_SC1) to select whether conversions will be continuous or completed only once, and to enable or disable conversion complete interrupts. The input channel on which conversions will be performed is also selected here.

### 16.5.2.1 Pseudo-code example

In this example, the ADC module is set up with interrupts enabled to perform a single hardware triggered 10-bit 4-level-FIFO conversion at low power with a long sample time on input channels of 1, 3, 5, and 7. Here the internal ADCK clock is derived from the bus clock divided by 1.

#### Example: 16.5.2.1.1 FIFO ADC initialization routine

```
void ADC_init(void)
{
/* The following code segment demonstrates how to initialize ADC by low-power mode, long
sample time, bus frequency, hardware triggered from AD1, AD3, AD5, and AD7 external pins
with 4-level FIFO enabled */

ADC_APCTL1 = ADC_APCTL1_ADPC6_MASK | ADC_APCTL1_ADPC5_MASK | ADC_APCTL1_ADPC3_MASK |
ADC_APCTL1_ADPC1_MASK; ADC_SC3 = ADC_SC3_ADLPC_MASK | ADC_SC3_ADLSMP_MASK |
ADC_SC3_MODE1_MASK;


// setting hardware trigger
ADC_SC2 = ADC_SC2_ADTRG_MASK ;

//4-Level FIFO
ADC_SC4 = ADC_SC4_AFDEP1_MASK | ADC_SC4_AFDEP0_MASK;

// dummy the 1st channel
ADC_SC1 = ADC_SC1_ADCH0_MASK;

// dummy the 2nd channel
ADC_SC1 = ADC_SC1_ADCH1_MASK | ADC_SC1_ADCH0_MASK;

// dummy the 3rd channel
ADC_SC1 = ADC_SC1_ADCH2_MASK | ADC_SC1_ADCH0_MASK;

// dummy the 4th channel and ADC starts conversion
ADC_SC1 = ADC_SC1_AIEN_MASK | ADC_SC1_ADCH2_MASK | ADC_SC1_ADCH1_MASK | ADC_SC1_ADCH0_MASK;

}
```

#### Example: 16.5.2.1.2 FIFO ADC interrupt service routine

```
unsigned short buffer[4];
interrupt VectorNumber_Vadc void ADC_isr(void)
{
      /* The following code segment demonstrates read AD result FIFO */
      // read conversion result of channel 1 and COCO bit is cleared
      buffer[0] = ADC_R;
      // read conversion result of channel 3
      buffer[1] = ADC_R;
      // read conversion result of channel 5
      buffer[2] = ADC_R;
      // read conversion result of channel 7
      buffer[3] = ADC_R;
}
```

**NOTE**

ADC_R is 16-bit ADC result register, combined from ADC_RH and ADC_RL

## 16.6 Application information

This section contains information for using the ADC module in applications. The ADC has been designed to be integrated into a microcontroller for use in embedded control applications requiring an A/D converter.

### 16.6.1 External pins and routing

The following sections discuss the external pins associated with the ADC module and how they are used for best results.

#### 16.6.1.1 Analog supply pins

The ADC module has analog power and ground supplies ($V_{DDA}$ and $V_{SSA}$) available as separate pins on some devices. $V_{SSA}$ is shared on the same pin as the MCU digital $V_{SS}$ on some devices. On other devices, $V_{SSA}$ and $V_{DDA}$ are shared with the MCU digital supply pins. In these cases, there are separate pads for the analog supplies bonded to the same pin as the corresponding digital supply so that some degree of isolation between the supplies is maintained.

When available on a separate pin, both $V_{DDA}$ and $V_{SSA}$ must be connected to the same voltage potential as their corresponding MCU digital supply ($V_{DD}$ and $V_{SS}$) and must be routed carefully for maximum noise immunity and bypass capacitors placed as near as possible to the package.

If separate power supplies are used for analog and digital power, the ground connection between these supplies must be at the $V_{SSA}$ pin. This should be the only ground connection between these supplies if possible. The $V_{SSA}$ pin makes a good single point ground location.

### 16.6.1.2 Analog reference pins

In addition to the analog supplies, the ADC module has connections for two reference voltage inputs. The high reference is $V_{REFH}$, which may be shared on the same pin as $V_{DDA}$ on some devices. The low reference is $V_{REFL}$, which may be shared on the same pin as $V_{SSA}$ on some devices.

When available on a separate pin, $V_{REFH}$ may be connected to the same potential as $V_{DDA}$, or may be driven by an external source between the minimum $V_{DDA}$ spec and the $V_{DDA}$ potential ($V_{REFH}$ must never exceed $V_{DDA}$). When available on a separate pin, $V_{REFL}$ must be connected to the same voltage potential as $V_{SSA}$. $V_{REFH}$ and $V_{REFL}$ must be routed carefully for maximum noise immunity and bypass capacitors placed as near as possible to the package.

AC current in the form of current spikes required to supply charge to the capacitor array at each successive approximation step is drawn through the $V_{REFH}$ and $V_{REFL}$ loop. The best external component to meet this current demand is a 0.1 μF capacitor with good high frequency characteristics. This capacitor is connected between $V_{REFH}$ and $V_{REFL}$ and must be placed as near as possible to the package pins. Resistance in the path is not recommended because the current causes a voltage drop that could result in conversion errors. Inductance in this path must be minimum (parasitic only).

### 16.6.1.3 Analog input pins

The external analog inputs are typically shared with digital I/O pins on MCU devices. The pin I/O control is disabled by setting the appropriate control bit in one of the pin control registers. Conversions can be performed on inputs without the associated pin control register bit set. It is recommended that the pin control register bit always be set when using a pin as an analog input. This avoids problems with contention because the output buffer is in its high impedance state and the pullup is disabled. Also, the input buffer draws DC current when its input is not at $V_{DD}$ or $V_{SS}$. Setting the pin control register bits for all pins used as analog inputs should be done to achieve lowest operating current.

Empirical data shows that capacitors on the analog inputs improve performance in the presence of noise or when the source impedance is high. Use of 0.01 μF capacitors with good high-frequency characteristics is sufficient. These capacitors are not necessary in all cases, but when used they must be placed as near as possible to the package pins and be referenced to $V_{SSA}$.

For proper conversion, the input voltage must fall between $V_{REFH}$ and $V_{REFL}$. If the input is equal to or exceeds $V_{REFH}$, the converter circuit converts the signal to 0x3FF (full scale 10-bit representation) or 0xFF (full scale 8-bit representation). If the input is equal to or

less than $V_{REFL}$, the converter circuit converts it to 0x000. Input voltages between $V_{REFH}$ and $V_{REFL}$ are straight-line linear conversions. There is a brief current associated with $V_{REFL}$ when the sampling capacitor is charging. The input is sampled for 3.5 cycles of the ADCK source when ADC_SC3[ADLSMP] is low, or 23.5 cycles when ADC_SC3[ADLSMP] is high.

For minimal loss of accuracy due to current injection, pins adjacent to the analog input pins should not be transitioning during conversions.

## 16.6.2 Sources of error

Several sources of error exist for A/D conversions. These are discussed in the following sections.

### 16.6.2.1 Sampling error

For proper conversions, the input must be sampled long enough to achieve the proper accuracy. Given the maximum input resistance of approximately 7 kΩ and input capacitance of approximately 5.5 pF, sampling to within 1/4 LSB can be achieved within the minimum sample window (3.5 cycles at 8 MHz maximum ADCK frequency) provided the resistance of the external analog source ($R_{AS}$) is kept below 2 kΩ.

Higher source resistances or higher-accuracy sampling is possible by setting ADC_SC3[ADLSMP] (to increase the sample window to 23.5 cycles) or decreasing ADCK frequency to increase sample time.

### 16.6.2.2 Pin leakage error

Leakage on the I/O pins can cause conversion error if the external analog source resistance ($R_{AS}$) is high. If this error cannot be tolerated by the application, keep $R_{AS}$ lower than $V_{DDA} / (2^N * I_{LEAK})$ for less than 1/4 LSB leakage error (N = 8 in 8-bit, 10 in 10-bit).

### 16.6.2.3 Noise-induced errors

System noise that occurs during the sample or conversion process can affect the accuracy of the conversion. The ADC accuracy numbers are guaranteed as specified only if the following conditions are met:

- There is a 0.1 µF low-ESR capacitor from $V_{REFH}$ to $V_{REFL}$.
- There is a 0.1 µF low-ESR capacitor from $V_{DDA}$ to $V_{SSA}$.
- If inductive isolation is used from the primary supply, an additional 1 µF capacitor is placed from $V_{DDA}$ to $V_{SSA}$.
- $V_{SSA}$ (and $V_{REFL}$, if connected) is connected to $V_{SS}$ at a quiet point in the ground plane.
- Operate the MCU in wait or Stop mode before initiating (hardware triggered conversions) or immediately after initiating (hardware or software triggered conversions) the ADC conversion.
  - For software triggered conversions, immediately follow the write to ADC_SC1 with a wait instruction or stop instruction.
  - For Stop mode operation, select ADACK as the clock source. Operation in Stop reduces $V_{DD}$ noise but increases effective conversion time due to stop recovery.
- There is no I/O switching, input or output, on the MCU during the conversion.

There are some situations where external system activity causes radiated or conducted noise emissions or excessive $V_{DD}$ noise is coupled into the ADC. In these situations, or when the MCU cannot be placed in wait or Stop or I/O activity cannot be halted, these recommended actions may reduce the effect of noise on the accuracy:

- Place a 0.01 µF capacitor ($C_{AS}$) on the selected input channel to $V_{REFL}$ or $V_{SSA}$ (this improves noise issues, but affects the sample rate based on the external analog source resistance).
- Average the result by converting the analog input many times in succession and dividing the sum of the results. Four samples are required to eliminate the effect of a 1LSB, one-time error.
- Reduce the effect of synchronous noise by operating off the asynchronous clock (ADACK) and averaging. Noise that is synchronous to ADCK cannot be averaged out.

### 16.6.2.4 Code width and quantization error

The ADC quantizes the ideal straight-line transfer function into 1024 steps (in 10-bit mode). Each step ideally has the same height (1 code) and width. The width is defined as the delta between the transition points to one code and the next. The ideal code width for an N bit converter (in this case N can be 8, 10), defined as 1LSB, is:

$$1\,\text{lsb} = (V_{\text{REFH}} - V_{\text{REFL}}) \;/\; 2^N$$

There is an inherent quantization error due to the digitization of the result. For 8-bit or 10-bit conversions the code transitions when the voltage is at the midpoint between the points where the straight line transfer function is exactly represented by the actual transfer function. Therefore, the quantization error will be ± 1/2 lsb in 8- or 10-bit mode. As a consequence, however, the code width of the first (0x000) conversion is only 1/2 lsb and the code width of the last (0xFF or 0x3FF) is 1.5 lsb.

### 16.6.2.5 Linearity errors

The ADC may also exhibit non-linearity of several forms. Every effort has been made to reduce these errors but the system must be aware of them because they affect overall accuracy. These errors are:

- Zero-scale error ($E_{ZS}$) (sometimes called offset) — This error is defined as the difference between the actual code width of the first conversion and the ideal code width (1/2 lsb in 8-bit or 10-bit modes). If the first conversion is 0x001, the difference between the actual 0x001 code width and its ideal is used.
- Full-scale error ($E_{FS}$) — This error is defined as the difference between the actual code width of the last conversion and the ideal code width (1.5 lsb in 8-bit or 10-bit modes). If the last conversion is 0x3FE, the difference between the actual 0x3FE code width and its ideal is used.
- Differential non-linearity (DNL) — This error is defined as the worst-case difference between the actual code width and the ideal code width for all conversions.
- Integral non-linearity (INL) — This error is defined as the highest-value that the absolute value of the running sum of DNL achieves. More simply, this is the worst-case difference of the actual transition voltage to a given code and its corresponding ideal transition voltage, for all codes.
- Total unadjusted error (TUE) — This error is defined as the difference between the actual transfer function and the ideal straight-line transfer function and includes all forms of error.

### 16.6.2.6 Code jitter, non-monotonicity, and missing codes

Analog-to-digital converters are susceptible to three special forms of error. These are code jitter, non-monotonicity, and missing codes.

Code jitter occurs when, at certain points, a given input voltage converts to one of two values when sampled repeatedly. Ideally, when the input voltage is infinitesimally smaller than the transition voltage, the converter yields the lower code (and vice-versa). However, even small amounts of system noise can cause the converter to be indeterminate, between two codes, for a range of input voltages around the transition voltage. This range is normally around ±1/2 lsb in 8-bit or 10-bit mode, and increases with noise.

This error may be reduced by repeatedly sampling the input and averaging the result. Additionally the techniques discussed in Noise-induced errors reduces this error.

Non-monotonicity is defined when, except for code jitter, the converter converts to a lower code for a higher input voltage. Missing codes are those values that are never converted for any input value.

In 8-bit or 10-bit mode, the ADC is guaranteed to be monotonic and have no missing codes.

# Chapter 17
# Cyclic redundancy check (CRC)

## 17.1 Introduction

Cyclic redundancy check (CRC) generates 16/32-bit CRC code for error detection. The CRC can be configured to work as a standard CRC. It provides the user with programmable polynomial, SEED and other parameters required to implement a 16-bit or 32-bit CRC standard. These parameters are detailed in further sections.

## 17.2 Features

Features of the CRC module are:

- Hardware 16/32-bit CRC generator
- Programmable initial seed value
- Programmable 16/32-bit polynomial
- Optional feature to reverse input and output data by bit
- Optional final complement output of result
- High-speed CRC calculation

## 17.3 Block diagram

The following figure is the CRC block diagram.

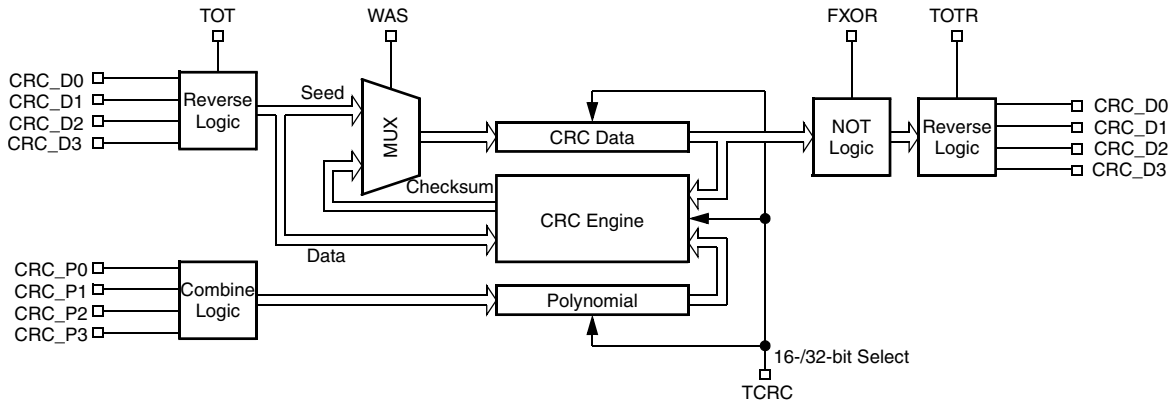



**Figure 17-1. Cyclic redundancy check (S08CRC) block diagram**

## 17.4 Modes of operation

This section defines the CRC operation in run, wait, and stop modes.

- Run mode - This is the basic mode of operation in which CRC is full functional.
- Wait mode - The CRC module is optional functional
- Stop3 mode - The CRC module is not functional in this low-power standby state. CRC calculations in progress stop and will resume after the CPU goes into run mode.

## 17.5 Register definition

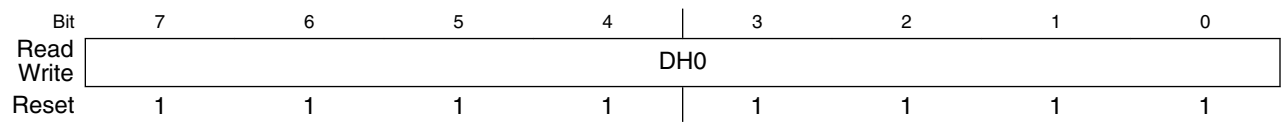
**CRC memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 3060 | CRC Data 0 Register (CRC_D0) | 8 | R/W | FFh | 17.5.1/335 |
| 3061 | CRC Data 1 Register (CRC_D1) | 8 | R/W | FFh | 17.5.2/335 |
| 3062 | CRC Data 2 Register (CRC_D2) | 8 | R/W | FFh | 17.5.3/336 |
| 3063 | CRC Data 3 Register (CRC_D3) | 8 | R/W | FFh | 17.5.4/337 |
| 3064 | CRC Polynomial 0 Register (CRC_P0) | 8 | R/W | 00h | 17.5.5/337 |
| 3065 | CRC Polynomial 1 Register (CRC_P1) | 8 | R/W | 00h | 17.5.6/338 |
| 3066 | CRC Polynomial 2 Register (CRC_P2) | 8 | R/W | 10h | 17.5.7/338 |
| 3067 | CRC Polynomial 3 Register (CRC_P3) | 8 | R/W | 21h | 17.5.8/339 |
| 3068 | CRC Control Register (CRC_CTRL) | 8 | R/W | 00h | 17.5.9/339 |

## 17.5.1 CRC Data 0 Register (CRC_D0)

D0 is one of the CRC data registers (D0:D3). The set of CRC data registers contains the value of seed, data, and checksum. When CRC_CTRL[WAS] bit is set, any write to the data registers is regarded as seed for CRC module. When CRC_CTRL[WAS] bit is clear, any write to the data registers is regarded as data for general CRC computation, in which D0:D2 does not accept any data and D3 accept 8-bit write upon the polynomial configuration. When final data are written, the final result can be read from the data register. The registers of D0:D1 contain the MSB 16-bit of CRC data, which is used only in CRC 32-bit mode. Only D3 is used to dummy data to CRC. Writing D2 will be ignored when WAS = 0.

Address: 3060h base + 0h offset = 3060h



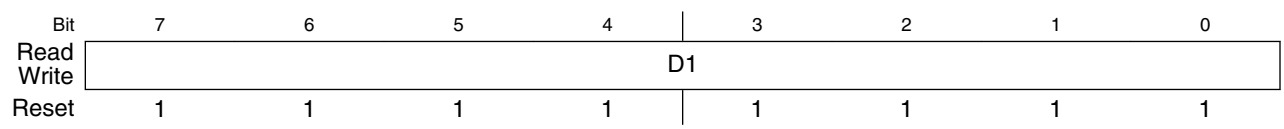

**CRC_D0 field descriptions**

| Field | Description |
|---|---|
| DH0 | CRC Data Bit 31:24 |

## 17.5.2 CRC Data 1 Register (CRC_D1)

D1 is one of the CRC data registers (D0:D3). The set of CRC data registers contains the value of seed, data, and checksum. When CRC_CTRL[WAS] bit is set, any write to the data registers is regarded as seed for CRC module. When CRC_CTRL[WAS] bit is clear, any write to the data registers is regarded as data for general CRC computation, in which D0:D2 does not accept any data and D3 accept 8-bit write upon the polynomial configuration. When final data are written, the final result can be read from the data register. The registers of D0:D1 contain the MSB 16-bit of CRC data, which is used only in CRC 32-bit mode. Only D3 is used to dummy data to CRC. Writing D2 will be ignored when WAS = 0.

Address: 3060h base + 1h offset = 3061h



**CRC_D1 field descriptions**

| Field | Description |
|---|---|
| D1 | CRC Data Bit 23:16 |

## 17.5.3 CRC Data 2 Register (CRC_D2)

D2 is one of the CRC data registers (D0:D3). The set of CRC data registers contains the value of seed, data, and checksum. When CRC_CTRL[WAS] bit is set, any write to the data registers is regarded as seed for CRC module. When CRC_CTRL[WAS] bit is clear, any write to the data registers is regarded as data for general CRC computation, in which D0:D2 does not accept any data and D3 accept 8-bit write upon the polynomial configuration. When final data are written, the final result can be read from the data register. The registers of D0:D1 contain the MSB 16-bit of CRC data, which is used only in CRC 32-bit mode. Only D3 is used to dummy data to CRC. Writing D2 will be ignored when WAS = 0.

Address: 3060h base + 2h offset = 3062h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | D2 | | | | | | | |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**CRC_D2 field descriptions**

| Field | Description |
|---|---|
| D2 | CRC Data Bit 15:8 |

## 17.5.4 CRC Data 3 Register (CRC_D3)

D3 is one of the CRC data registers (D0:D3). The set of CRC data registers contains the value of seed, data, and checksum. When CRC_CTRL[WAS] bit is set, any write to the data registers is regarded as seed for CRC module. When CRC_CTRL[WAS] bit is clear, any write to the data registers is regarded as data for general CRC computation, in which D0:D2 does not accept any data and D3 accept 8-bit write upon the polynomial configuration. When final data are written, the final result can be read from the data register. The registers of D0:D1 contain the MSB 16-bit of CRC data, which is used only in CRC 32-bit mode. Only D3 is used to dummy data to CRC. Writing D2 will be ignored when WAS = 0.

Address: 3060h base + 3h offset = 3063h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | D3 | | | | | | | |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**CRC_D3 field descriptions**

| Field | Description |
|---|---|
| D3 | CRC Data Bit 7:0 |

## 17.5.5 CRC Polynomial 0 Register (CRC_P0)

P0 is one of the CRC polynomial registers (P0:P3). The set of CRC polynominal registers contains the value of polynomial. The registers of P0:P1 contain the MSB 16-bit of CRC polynomial, which is used only in CRC 32-bit mode. The registers of P2:P3 contain the LSB 16-bit of CRC polynomial, which is used in both CRC 16- and 32-bit modes.

Address: 3060h base + 4h offset = 3064h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | P0 | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**CRC_P0 field descriptions**

| Field | Description |
|---|---|
| P0 | CRC Polynominal Bit 31:24 |

## 17.5.6 CRC Polynomial 1 Register (CRC_P1)

P1 is one of the CRC polynomial registers (P0:P3). The set of CRC polynominal registers contains the value of polynomial. The registers of P0:P1 contain the MSB 16-bit of CRC polynomial, which is used only in CRC 32-bit mode. The registers of P2:P3 contain the LSB 16-bit of CRC polynomial, which is used in both CRC 16- and 32-bit modes.

Address: 3060h base + 5h offset = 3065h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | P1 | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**CRC_P1 field descriptions**

| Field | Description |
|---|---|
| P1 | CRC Polynominal Bit 23:16 |

## 17.5.7 CRC Polynomial 2 Register (CRC_P2)

P2 is one of the CRC polynomial registers (P0:P3). The set of CRC polynominal registers contains the value of polynomial. The registers of P0:P1 contain the MSB 16-bit of CRC polynomial, which is used only in CRC 32-bit mode. The registers of P2:P3 contain the LSB 16-bit of CRC polynomial, which is used in both CRC 16- and 32-bit modes.

Address: 3060h base + 6h offset = 3066h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | P2 | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

**CRC_P2 field descriptions**

| Field | Description |
|---|---|
| P2 | CRC Polynominal Bit 15:8 |

## 17.5.8 CRC Polynomial 3 Register (CRC_P3)

P3 is one of the CRC polynomial registers (P0:P3). The set of CRC polynominal registers contains the value of polynomial. The registers of P0:P1 contain the MSB 16-bit of CRC polynomial, which is used only in CRC 32-bit mode. The registers of P2:P3 contain the LSB 16-bit of CRC polynomial, which is used in both CRC 16- and 32-bit modes.

Address: 3060h base + 7h offset = 3067h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | P3 | | | | | | | |
| Reset | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

**CRC_P3 field descriptions**

| Field | Description |
|---|---|
| P3 | CRC Polynominal Bit 7:0 |

## 17.5.9 CRC Control Register (CRC_CTRL)

Address: 3060h base + 8h offset = 3068h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | TOT | | TOTR | | 0 | FXOR | WAS | TCRC |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**CRC_CTRL field descriptions**

| Field | Description |
|---|---|
| 7–6<br>TOT | Reverse of Write<br><br>These bits identify the reverse of the input data.<br><br>00 No reverse.<br>01 Bit is reversed in byte; No byte is reversed.<br>10 Reserved.<br>11 Reserved. |
| 5–4<br>TOTR | Reverse of Read<br><br>These bits identify the reverse of the output data.<br><br>00 No reverse.<br>01 Bit is reversed in byte; No byte is reversed.<br>10 Reserved.<br>11 Reserved. |

*Table continues on the next page...*

**CRC_CTRL field descriptions (continued)**

| Field | Description |
|---|---|
| 3<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 2<br>FXOR | Complement of Read<br><br>This bit allows CRC module to output the complement of the final CRC checksum.<br><br>0 Normal checksum output.<br>1 Complement of checksum output. |
| 1<br>WAS | Write CRC data register as seed<br><br>This bit indicates the data written to the CRC data register (D0:D3) is seed or data.<br><br>0 Data is written in data registers.<br>1 Seed is written in data registers. |
| 0<br>TCRC | Width of Polynomial Generator<br><br>This bit indicates the bit width of the polynomial generator.<br><br>0 16-bit CRC Polynomial Generator.<br>1 32-bit CRC Polynomial Generator. |

# 17.6 Functional description

## 17.6.1 16-bit CRC calculation

The following steps show how to start a general 16-bit CRC calculation:

1. Clear CRC_CTRL[TCRC] bit to enable 16-bit CRC mode.
2. Optional to enable reverse and complement function. Please see Bit reverse and Result complement for details.
3. Write 16-bit polynomial to CRC_P2:CRC_P3.
4. Set CRC_CTRL[WAS] bit to allow CRC_D2:CRC_D3 to be written by seed.
5. Write 16-bit seed to CRC_D2:CRC_D3.
6. Clear CRC_CTRL[WAS] bit to start 16-bit CRC calculation.
7. Dummy CRC_D3 with 8-bit CRC raw data.
8. Get the checksum from CRC_D2:CRC_D3 when all CRC raw data dummied.

## 17.6.2 32-bit CRC calculation

The following steps show how to start a general 32-bit CRC calculation:

1. Set CRC_CTRL[TCRC] bit to enable 32-bit CRC mode.

2. Optional to enable reverse and complement function. Please see Bit reverse and Result complement for details.
3. Write 32-bit polynomial to CRC_P0:CRC_P3.
4. Set CRC_CTRL[WAS] bit to allow CRC_D0:CRC_D3 written by seed.
5. Write 32-bit seed to CRC_D0:CRC_D3.
6. Clear CRC_CTRL[WAS] bit to start 32-bit CRC calculation.
7. Dummy CRC_D3 with 8-bit CRC raw data.
8. Get the checksum from CRC_D0:CRC_D3 when all CRC raw data dummied.

## 17.6.3 Bit reverse

The bit reverse function allows the input and output data reversed by bit for different CRC standard and endian systems. The CRC_CTRL[TOT] bits control the reverse of input data and the CRC_CTRL[TOTR] bits control the reverse of output data. The following table shows how the CRC_CTRL[TOT] and CRC_CTRL[TOTR] bits work.

**Table 17-1. TOT and TOTR bit and byte reverse function**

| TOT ROW | D0 | D1 | D2 | D3 |
|---|---|---|---|---|
| 00 | b31b30b29b28b27b26b25b24 | b23b22b21b20b19b18b17b16 | b15b14b13b12b11b10b9b8 | b7b6b5b4b3b2b1b0 |
| 01 | b24b25b26b27b28b29b30b31 | b16b17b18b19b20b21b22b23 | b8b9b10b11b12b13b14b15 | b0b1b2b3b4b5b6b7 |

**NOTE**

00 is the default case that no bit is reversed.

## 17.6.4 Result complement

The result complement function allows to output the complement of the checksum in CRC data registers. When CRC_CTRL[FXOR] bit is set, the checksum is read by its complement. Otherwise, the raw checksum is accessed.

## 17.6.5 CCITT compliant CRC example

The following code segment shows CCITT CRC-16 compliant example.

### Example: 17.6.5.1 CCITT CRC-16 compliant example

```
CRC_CTRL = CRC_CTRL_WAS_MASK; // 16-bit CRC, ready to dummy seed
CRC_P2P3 = 0x1021; // Standard CCITT polynomail of (x^16 + x^12 + x^5 + 1)
CRC_D2D3 = 0xFFFF; // Set seed by 0xFFFF
CRC_CTRL = 0x00;

for ( i = 0 ; i < 128 ; i++ )
{
CRC_D3 = 'A'; // Dummy 256 `A'
CRC_D3 = 'A';
}

// Get 0xea0b in CRC_D2:CRC_D3 here
```

# Chapter 18
# Watchdog (WDOG)

## 18.1 Introduction

The Watchdog Timer (WDOG) module is an independent timer that is available for system use. It provides a safety feature to ensure that software is executing as planned and that the CPU is not stuck in an infinite loop or executing unintended code. If the WDOG module is not serviced (refreshed) within a certain period, it resets the MCU.

### 18.1.1 Features

Features of the WDOG module include:

- Configurable clock source inputs independent from the:
  - bus clock
  - Internal 32 kHz RC oscillator
  - Internal 1 kHz RC oscillator
  - External clock source
- Programmable timeout period
  - Programmable 16-bit timeout value
  - Optional fixed 256 clock prescaler when longer timeout periods are needed
- Robust write sequence for counter refresh
  - Refresh sequence of writing 0xA602 and then 0xB480 within 16 bus clocks
- Window mode option for the refresh mechanism
  - Programmable 16-bit window value

- Provides robust check that program flow is faster than expected
- Early refresh attempts trigger a reset.

- Optional timeout interrupt to allow post-processing diagnostics
  - Interrupt request to CPU with interrupt vector for an interrupt service routine (ISR)
  - Forced reset occurs 128 bus clocks after the interrupt vector fetch.
- Configuration bits are write-once-after-reset to ensure watchdog configuration cannot be mistakenly altered.
- Robust write sequence for unlocking write-once configuration bits
  - Unlock sequence of writing 0xC520 and then 0xD928 within 16 bus clocks for allowing updates to write-once configuration bits
  - Software must make updates within 128 bus clocks after unlocking and before WDOG closing unlock window.

## 18.1.2 Block diagram

The following figure provides a block diagram of the WDOG module.

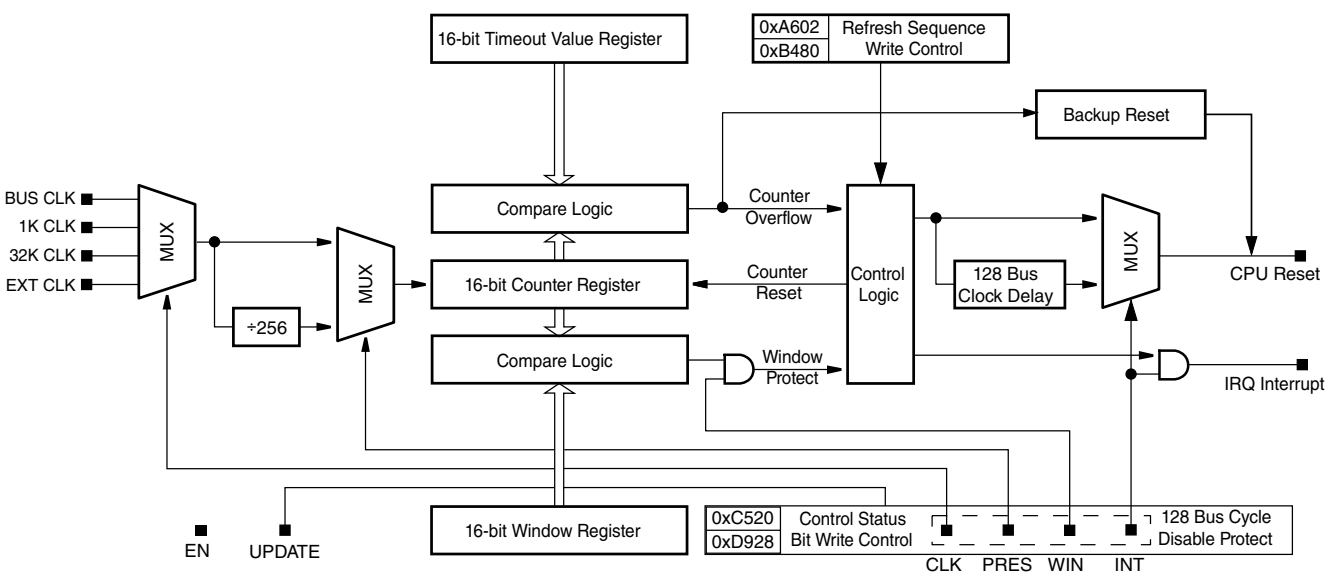



**Figure 18-1. WDOG block diagram**

## 18.2 Memory map and register definition

**WDOG memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 3030 | Watchdog Control and Status Register 1 (WDOG_CS1) | 8 | R/W | 80h | 18.2.1/345 |
| 3031 | Watchdog Control and Status Register 2 (WDOG_CS2) | 8 | R/W | 01h | 18.2.2/347 |
| 3032 | Watchdog Counter Register: High (WDOG_CNTH) | 8 | R/W | 00h | 18.2.3/348 |
| 3033 | Watchdog Counter Register: Low (WDOG_CNTL) | 8 | R/W | 00h | 18.2.4/348 |
| 3034 | Watchdog Timeout Value Register: High (WDOG_TOVALH) | 8 | R/W | 00h | 18.2.5/349 |
| 3035 | Watchdog Timeout Value Register: Low (WDOG_TOVALL) | 8 | R/W | 04h | 18.2.6/349 |
| 3036 | Watchdog Window Register: High (WDOG_WINH) | 8 | R/W | 00h | 18.2.7/350 |
| 3037 | Watchdog Window Register: Low (WDOG_WINL) | 8 | R/W | 00h | 18.2.8/350 |

### 18.2.1 Watchdog Control and Status Register 1 (WDOG_CS1)

This section describes the function of Watchdog Control and Status Register 1.

**NOTE**

TST is cleared (0:0) on POR only. Any other reset does not affect the value of this field.

Address: 3030h base + 0h offset = 3030h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | EN | INT | UPDATE | TST | | DBG | WAIT | STOP |
| Reset | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**WDOG_CS1 field descriptions**

| Field | Description |
|---|---|
| 7<br>EN | Watchdog Enable<br><br>This write-once bit enables the watchdog counter to start counting.<br><br>0 Watchdog disabled.<br>1 Watchdog enabled. |
| 6<br>INT | Watchdog Interrupt<br><br>This write-once bit configures the watchdog to generate an interrupt request upon a reset-triggering event (timeout or illegal write to the watchdog), prior to forcing a reset. After the interrupt vector fetch, the reset occurs after a delay of 128 bus clocks. |

*Table continues on the next page...*

## WDOG_CS1 field descriptions (continued)

| Field | Description |
|---|---|
| | 0 Watchdog interrupts are disabled. Watchdog resets are not delayed.<br>1 Watchdog interrupts are enabled. Watchdog resets are delayed by 128 bus clocks. |
| 5<br>UPDATE | Allow updates<br>This write-once bit allows software to reconfigure the watchdog without a reset.<br>0 Updates not allowed. After the initial configuration, the watchdog cannot be later modified without forcing a reset.<br>1 Updates allowed. Software can modify the watchdog configuration registers within 128 bus clocks after performing the unlock write sequence. |
| 4–3<br>TST | Watchdog Test<br>Enables the fast test mode. The test mode allows software to exercise all bits of the counter to demonstrate that the watchdog is functioning properly. See the Fast testing of the watchdog section.<br>This write-once field is cleared (0:0) on POR only. Any other reset does not affect the value of this field.<br>00 Watchdog test mode disabled.<br>01 Watchdog user mode enabled. (Watchdog test mode disabled.) After testing the watchdog, software should use this setting to indicate that the watchdog is functioning normally in user mode.<br>10 Watchdog test mode enabled, only the low byte is used. WDOG_CNTL is compared with WDOG_TOVALL.<br>11 Watchdog test mode enabled, only the high byte is used. WDOG_CNTH is compared with WDOG_TOVALH. |
| 2<br>DBG | Debug Enable<br>This write-once bit enables the watchdog to operate when the chip is in debug mode.<br>0 Watchdog disabled in chip debug mode.<br>1 Watchdog enabled in chip debug mode. |
| 1<br>WAIT | Wait Enable<br>This write-once bit enables the watchdog to operate when the chip is in wait mode.<br>0 Watchdog disabled in chip wait mode.<br>1 Watchdog enabled in chip wait mode. |
| 0<br>STOP | Stop Enable<br>This write-once bit enables the watchdog to operate when the chip is in stop mode.<br>0 Watchdog disabled in chip stop mode.<br>1 Watchdog enabled in chip stop mode. |

## 18.2.2 Watchdog Control and Status Register 2 (WDOG_CS2)

This section describes the function of the watchdog control and status register 2.

Address: 3030h base + 1h offset = 3031h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | WIN | FLG | 0 | PRES | 0 | | CLK | |
| Write | | w1c | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**WDOG_CS2 field descriptions**

| Field | Description |
|---|---|
| 7<br>WIN | Watchdog Window<br><br>This write-once bit enables window mode. See the Window mode section.<br><br>0 Window mode disabled.<br>1 Window mode enabled. |
| 6<br>FLG | Watchdog Interrupt Flag<br><br>This bit is an interrupt indicator when INT is set in control and status register 1. Write 1 to clear it.<br><br>0 No interrupt occurred.<br>1 An interrupt occurred. |
| 5<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 4<br>PRES | Watchdog Prescalar<br><br>This write-once bit enables a fixed 256 pre-scaling of watchdog counter reference clock. (The block diagram shows this clock divider option.)<br><br>0 256 prescalar disabled.<br>1 256 prescalar enabled. |
| 3–2<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| CLK | Watchdog Clock<br><br>This write-once field indicates the clock source that feeds the watchdog counter. See the Clock source section.<br><br>00 Bus clock.<br>01 1 kHz internal low-power oscillator (LPOCLK).<br>10 32 kHz internal oscillator (ICSIRCLK).<br>11 External clock source. |

## 18.2.3 Watchdog Counter Register: High (WDOG_CNTH)

This section describes the watchdog counter registers: high (CNTH) and low (CNTL) combined.

The watchdog counter registers CNTH and CNTL provide access to the value of the free-running watchdog counter. Software can read the counter registers at any time.

Software cannot write directly to the watchdog counter; however, two write sequences to these registers have special functions:

1. The *refresh sequence* resets the watchdog counter to 0x0000. See the Refreshing the Watchdog section.
2. The *unlock sequence* allows the watchdog to be reconfigured without forcing a reset (when WDOG_CS1[UPDATE] = 1). See the Example code: Reconfiguring the Watchdog section.

**NOTE**

All other writes to these registers are illegal and force a reset.

Address: 3030h base + 2h offset = 3032h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | CNTHIGH | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**WDOG_CNTH field descriptions**

| Field | Description |
|---|---|
| CNTHIGH | High byte of the Watchdog Counter |

## 18.2.4 Watchdog Counter Register: Low (WDOG_CNTL)

See the description of the WDOG_CNTH register.

Address: 3030h base + 3h offset = 3033h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | CNTLOW | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**WDOG_CNTL field descriptions**

| Field | Description |
|---|---|
| CNTLOW | Low byte of the Watchdog Counter |

## 18.2.5 Watchdog Timeout Value Register: High (WDOG_TOVALH)

This section describes the watchdog timeout value registers: high (WDOG_TOVALH) and low (WDOG_TOVALL) combined. WDOG_TOVALH and WDOG_TOVALL contains the 16-bit value used to set the timeout period of the watchdog.

The watchdog counter (WDOG_CNTH and WDOG_CNTL) is continuously compared with the timeout value (WDOG_TOVALH and WDOG_TOVALL). If the counter reaches the timeout value, the watchdog forces a reset.

**NOTE**

Do not write 0 to the Watchdog Timeout Value Register, otherwise, the watchdog always generates a reset.

Address: 3030h base + 4h offset = 3034h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | TOVALHIGH | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**WDOG_TOVALH field descriptions**

| Field | Description |
|---|---|
| TOVALHIGH | High byte of the timeout value |

## 18.2.6 Watchdog Timeout Value Register: Low (WDOG_TOVALL)

See the description of the WDOG_TOVALH register.

**NOTE**

All the bits reset to 0 in read.

Address: 3030h base + 5h offset = 3035h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | TOVALLOW | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

**WDOG_TOVALL field descriptions**

| Field | Description |
|---|---|
| TOVALLOW | Low byte of the timeout value |

## 18.2.7 Watchdog Window Register: High (WDOG_WINH)

This section describes the watchdog window registers: high (WDOG_WINH) and low (WDOG_WINL) combined. When window mode is enabled (WDOG_CS2[WIN] is set), WDOG_WINH and WDOG_WINL determine the earliest time that a refresh sequence is considered valid. See the Watchdog refresh mechanism section.

WDOG_WINH and WDOG_WINL must be less than WDOG_TOVALH and WDOG_TOVALL.

Address: 3030h base + 6h offset = 3036h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | WINHIGH | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**WDOG_WINH field descriptions**

| Field | Description |
|---|---|
| WINHIGH | High byte of Watchdog Window |

## 18.2.8 Watchdog Window Register: Low (WDOG_WINL)

See the description of the WDOG_WINH register.

Address: 3030h base + 7h offset = 3037h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | WINLOW | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**WDOG_WINL field descriptions**

| Field | Description |
|---|---|
| WINLOW | Low byte of Watchdog Window |

## 18.3 Functional description

The WDOG module provides a fail safe mechanism to ensure the system can be reset to a known state of operation in case of system failure, such as the CPU clock stopping or there being a run away condition in the software code. The watchdog counter runs continuously off a selectable clock source and expects to be serviced (refreshed) periodically. If it is not, it resets the system.

The timeout period, window mode, and clock source are all programmable but must be configured within 128 bus clocks after a reset.

### 18.3.1 Watchdog refresh mechanism

The watchdog resets the MCU if the watchdog counter is not refreshed. A robust refresh mechanism makes it very unlikely that the watchdog can be refreshed by runaway code.

To refresh the watchdog counter, software must execute a refresh write sequence before the timeout period expires. In addition, if window mode is used, software must not start the refresh sequence until after the time value set in the WDOG_WINH and WDOG_WINL registers. See the following figure.

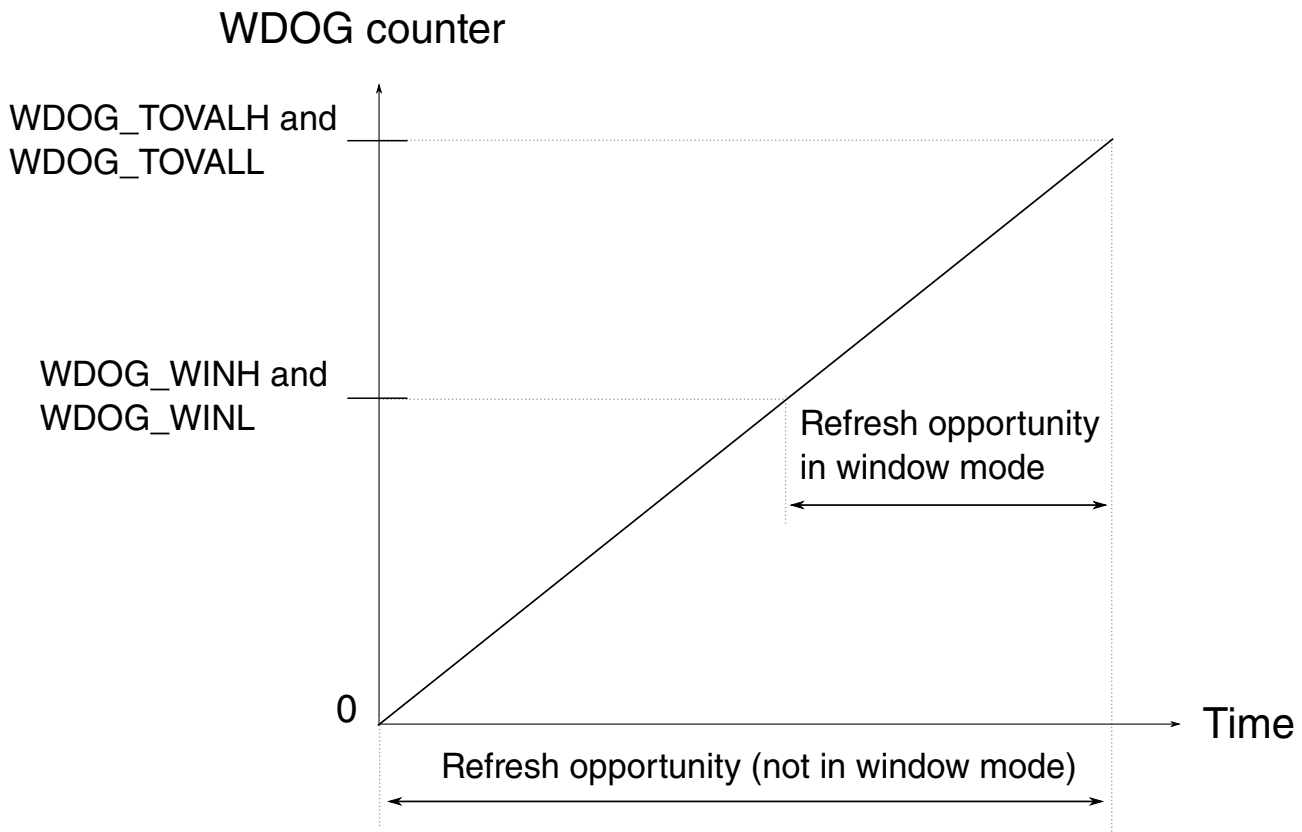



**Figure 18-2. Refresh opportunity for the Watchdog counter**

### 18.3.1.1 Window mode

Software finishing its main control loop faster than expected could be an indication of a problem. Depending on the requirements of the application, the WDOG can be programmed to force a reset when refresh attempts are early.

When Window mode is enabled, the watchdog must be refreshed after the counter has reached a minimum expected time value; otherwise, the watchdog resets the MCU. The minimum expected time value is specified in the WDOG_WINH:L registers. Setting CS1[WIN] enables Window mode.

### 18.3.1.2 Refreshing the Watchdog

The refresh write sequence is a write of 0xA602 followed by a write of 0xB480 to the WDOG_CNTH and WDOG_CNTL registers. The write of the 0xB480 must occur within 16 bus clocks after the write of 0xA602; otherwise, the watchdog resets the MCU.

**Note**

Before starting the refresh sequence, disable global interrupts. Otherwise, an interrupt could effectively invalidate the refresh sequence if writing the four bytes takes more than 16 bus clocks. Re-enable interrupts when the sequence is finished.

#### 18.3.1.3 Example code: Refreshing the Watchdog

The following code segment shows the refresh write sequence of the WDOG module.

```
/* Refresh watchdog */

for (;;) // main loop
{
   ...

   DisableInterrupts; // disable global interrupt

   WDOG_CNT = 0xA602; // write the 1st refresh word

   WDOG_CNT = 0xB480; // write the 2nd refresh word to refresh counter

   EnableInterrupts; // enable global interrupt

   ...
}
```

### 18.3.2 Configuring the Watchdog

All watchdog control bits, timeout value, and window value are write-once after reset. This means that after a write has occurred they cannot be changed unless a reset occurs. This provides a robust mechanism to configure the watchdog and ensure that a runaway condition cannot mistakenly disable or modify the watchdog configuration after configured.

This is guaranteed by the user configuring the window and timeout value first, followed by the other control bits, and ensuring that CS1[UPDATE] is also set to 0. The new configuration takes effect only after all registers except WDOG_CNTH:L are written once after reset. Otherwise, the WDOG uses the reset values by default. If window mode is not used (CS2[WIN] is 0), writing to WDOG_WINH:L is not required to make the new configuration take effect.

### 18.3.2.1 Reconfiguring the Watchdog

In some cases (such as when supporting a bootloader function), users may want to reconfigure or disable the watchdog without forcing a reset first. By setting CS1[UPDATE] to a 1 on the initial configuration of the watchdog after a reset, users can reconfigure the watchdog at any time by executing an unlock sequence. (Conversely, if CS1[UPDATE] remains 0, the only way to reconfigure the watchdog is by initiating a reset.) The unlock sequence is similar to the refresh sequence but uses different values.

### 18.3.2.2 Unlocking the Watchdog

The unlock sequence is a write to the WDOG_CNTH:L registers of 0xC520 followed by 0xD928 within 16 bus clocks at any time after the watchdog has been configured. On completing the unlock sequence, the user must reconfigure the watchdog within 128 bus clocks; otherwise, the watchdog forces a reset to the MCU.

**NOTE**

Due to 128 bus clocks requirement for reconfiguring the watchdog, some delays must be inserted before executing STOP or WAIT instructions after reconfiguring the watchdog. This ensures that the watchdog's new configuration takes effect before MCU enters low power mode. Otherwise, the MCU may not be waken up from low power mode.

### 18.3.2.3 Example code: Reconfiguring the Watchdog

The following code segment shows an example reconfiguration of the WDOG module.

```
/* Initialize watchdog with ~1-kHz clock source, ~1s time-out */

DisableInterrupts; // disable global interrupt

WDOG_CNT = 0xC520; // write the 1st unlock word

WDOG_CNT = 0xD928; // write the 2nd unlock word

WDOG_TOVAL = 1000; // setting timeout value

WDOG_CS2 = WDOG_CS2_CLK_MASK; // setting 1-kHz clock source

WDOG_CS1 = WDOG_CS1_EN_MASK; // enable counter running

EnableInterrupts; // enable global interrupt
```

### 18.3.3 Clock source

The watchdog counter has four clock source options selected by programming CS2[CLK]:

- bus clock
- internal Low-Power Oscillator (LPO) running at approximately 1 kHz (This is the default source.)
- internal 32 kHz clock
- external clock

The options allow software to select a clock source independent of the bus clock for applications that need to meet more robust safety requirements. Using a clock source other than the bus clock ensures that the watchdog counter continues to run if the bus clock is somehow halted; see Backup reset.

An optional fixed prescaler for all clock sources allows for longer timeout periods. When CS2[PRES] is set, the clock source is prescaled by 256 before clocking the watchdog counter.

The following table summarizes the different watchdog timeout periods available.

**Table 18-1. Watchdog timeout availability**

| Reference clock | Prescaler | Watchdog time-out availability |
|---|---|---|
| Internal ~1 kHz (LPO) | Pass through | ~1 ms–65.5 s[1] |
| | ÷256 | ~256 ms–16,777 s |
| Internal ~32 kHz | Pass through | ~31.25 µs–2.048 s |
| | ÷256 | ~8 ms–524.3 s |
| 1 MHz (from bus or external) | Pass through | 1 µs–65.54 ms |
| | ÷256 | 256 µs–16.777 s |
| 20 MHz (from bus or external) | Pass through | 50 ns–3.277 ms |
| | ÷256 | 12.8 µs–838.8 ms |

1. The default timeout value after reset is approximately 4 ms.

**NOTE**

When the programmer switches clock sources during reconfiguration, the watchdog hardware holds the counter at zero for 2.5 periods of the previous clock source and 2.5 periods of the new clock source after the configuration time period (128 bus clocks) ends. This delay ensures a smooth transition before restarting the counter with the new configuration.

### 18.3.4 Using interrupts to delay resets

When interrupts are enabled (CS1[INT] = 1), the watchdog first generates an interrupt request upon a reset triggering event (such as a counter timeout or invalid refresh attempt). The watchdog delays forcing a reset for 128 bus clocks to allow the interrupt service routine (ISR) to perform tasks, such as analyzing the stack to debug code.

When interrupts are disabled (CS1[INT] = 0), the watchdog does not delay forcing a reset.

### 18.3.5 Backup reset

**NOTE**

A clock source other than the bus clock must be used as the reference clock for the counter; otherwise, the backup reset function is not available.

The backup reset function is a safeguard feature that independently generates a reset in case the main WDOG logic loses its clock (the bus clock) and can no longer monitor the counter. If the watchdog counter overflows twice in succession (without an intervening reset), the backup reset function takes effect and generates a reset.

### 18.3.6 Functionality in debug and low-power modes

By default, the watchdog is not functional in Active Background mode, Wait mode, or Stop3 mode. However, the watchdog can remain functional in these modes as follows:

- For Active Background mode, set CS1[DBG]. (This way the watchdog is functional in Active Background mode even when the CPU is held by the Debug module.)
- For Wait mode, set CS1[WAIT].
- For Stop3 mode, set CS1[STOP].

**NOTE**

The watchdog can not generate interrupt in Stop3 mode even if CS1[STOP] is set and will not wake the MCU from Stop3 mode. It can generate reset during Stop3 mode.

For Active Background mode and Stop3 mode, in addition to the above configurations, a clock source other than the bus clock must be used as the reference clock for the counter; otherwise, the watchdog cannot function.

## 18.3.7 Fast testing of the watchdog

Before executing application code in safety critical applications, users are required to test that the watchdog works as expected and resets the MCU. Testing every bit of a 16-bit counter by letting it run to the overflow value takes a relatively long time (64 kHz clocks).

To help minimize the startup delay for application code after reset, the watchdog has a feature to test the watchdog more quickly by splitting the counter into its constituent byte-wide stages. The low and high bytes are run independently and tested for timeout against the corresponding byte of the timeout value register. (For complete coverage when testing the high byte of the counter, the test feature feeds the input clock via the 8th bit of the low byte, thus ensuring that the overflow connection from the low byte to the high byte is tested.)

Using this test feature reduces the test time to 512 clocks (not including overhead, such as user configuration and reset vector fetches). To further speed testing, use a faster clock (such as the bus clock) for the counter reference.

On a power-on reset, the POR bit in the system reset register is set, indicating the user should perform the WDOG fast test.

### 18.3.7.1 Testing each byte of the counter

The test procedure follows these steps:

1. Program the preferred watchdog timeout value in the WDOG_TOVALH and WDOG_TOVALL registers during the watchdog configuration time period.
2. Select a byte of the counter to test using the WDOG_CS1[TST] = 10b for the low byte; WDOG_CS1[TST] = 11b for the high byte.
3. Wait for the watchdog to timeout. Optionally, in the idle loop, increment RAM locations as a parallel software counter for later comparison. Because the RAM is not affected by a watchdog reset, the timeout period of the watchdog counter can be compared with the software counter to verify the timeout period has occurred as expected.
4. The watchdog counter times out and forces a reset.

5. Confirm the WDOG flag in the system reset register is set, indicating that the watchdog caused the reset. (The POR flag remains clear.)
6. Confirm that WDOG_CS1[TST] shows a test (10b or 11b) was performed.

If confirmed, the count and compare functions work for the selected byte. Repeat the procedure, selecting the other byte in step 2.

**NOTE**

WDOG_CS1[TST] is cleared by a POR only and not affected by other resets.

### 18.3.7.2 Entering user mode

After successfully testing the low and high bytes of the watchdog counter, the user can configure WDOG_CS1[TST] to 01b to indicate the watchdog is ready for use in application user mode. Thus if a reset occurs again, software can recognize the reset trigger as a real watchdog reset caused by runaway or faulty application code.

As an ongoing test when using the default 1 kHz clock source, software can periodically read the WDOG_CNTH and WDOG_CNTL registers to ensure the counter is being incremented.

# Chapter 19
# Development support

## 19.1 Introduction

This chapter describes the single-wire background debug mode (BDM), which uses the on-chip background debug controller (BDC) module, and the independent on-chip real-time in-circuit emulation (ICE) system, which uses the on-chip debug (DBG) module.

### 19.1.1 Forcing active background

The method for forcing active background mode depends on the specific HCS08 derivative. For the 9S08xxxx, you can force active background after a power-on reset by holding the BKGD pin low as the device exits the reset condition. You can also force active background by driving BKGD low immediately after a serial background command that writes a one to the BDFR bit in the SBDFR register. Other causes of reset including an external pin reset or an internally generated error reset ignore the state of the BKGD pin and reset into normal user mode. If no debug pod is connected to the BKGD pin, the MCU will always reset into normal operating mode.

### 19.1.2 Features

Features of the BDC module include:

- Single pin for mode selection and background communications
- BDC registers are not located in the memory map
- SYNC command to determine target communications rate
- Non-intrusive commands for memory access
- Active background mode commands for CPU register access
- GO and TRACE1 commands
- BACKGROUND command can wake CPU from stop or wait modes
- One hardware address breakpoint built into BDC

- BDC clock runs in stop mode, if BDC enabled
- Watchdog disabled by default while in active background mode. It can also be enabled by proper configuration

Features of the ICE system include:

- Two trigger comparators: Two address + read/write (R/W) or one full address + data + R/W
- Flexible 8-word by 16-bit FIFO (first-in, first-out) buffer for capture information:
  - Change-of-flow addresses or
  - Event-only data
- Two types of breakpoints:
  - Tag breakpoints for instruction opcodes
  - Force breakpoints for any address access
- Nine trigger modes:
  - Basic: A-only, A OR B
  - Sequence: A then B
  - Full: A AND B data, A AND NOT B data
  - Event (store data): Event-only B, A then event-only B
  - Range: Inside range (A ≤ address ≤ B), outside range (address < A or address > B)

## 19.2 Background debug controller (BDC)

All MCUs in the HCS08 Family contain a single-wire background debug interface that supports in-circuit programming of on-chip nonvolatile memory and sophisticated non-intrusive debug capabilities. Unlike debug interfaces on earlier 8-bit MCUs, this system does not interfere with normal application resources. It does not use any user memory or locations in the memory map and does not share any on-chip peripherals.

BDC commands are divided into two groups:

- Active background mode commands require that the target MCU is in active background mode (the user program is not running). Active background mode commands allow the CPU registers to be read or written, and allow the user to trace one user instruction at a time, or GO to the user program from active background mode.
- Non-intrusive commands can be executed at any time even while the user's program is running. Non-intrusive commands allow a user to read or write MCU memory locations or access status and control registers within the background debug controller.

Typically, a relatively simple interface pod is used to translate commands from a host computer into commands for the custom serial interface to the single-wire background debug system. Depending on the development tool vendor, this interface pod may use a standard RS-232 serial port, a parallel printer port, or some other type of communications such as a universal serial bus (USB) to communicate between the host PC and the pod. The pod typically connects to the target system with ground, the BKGD pin, RESET, and sometimes $V_{DD}$. An open-drain connection to reset allows the host to force a target system reset, which is useful to regain control of a lost target system or to control startup of a target system before the on-chip nonvolatile memory has been programmed. Sometimes $V_{DD}$ can be used to allow the pod to use power from the target system to avoid the need for a separate power supply. However, if the pod is powered separately, it can be connected to a running target system without forcing a target system reset or otherwise disturbing the running application program.

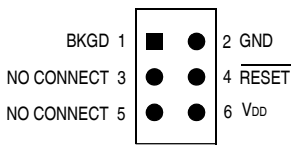



**Figure 19-1. BDM tool connector**

## 19.2.1 BKGD pin description

BKGD is the single-wire background debug interface pin. The primary function of this pin is for bidirectional serial communication of active background mode commands and data. During reset, this pin is used to select between starting in active background mode or starting the user's application program. This pin is also used to request a timed sync response pulse to allow a host development tool to determine the correct clock frequency for background debug serial communications.

BDC serial communications use a custom serial protocol first introduced on the M68HC12 Family of microcontrollers. This protocol assumes the host knows the communication clock rate that is determined by the target BDC clock rate. All communication is initiated and controlled by the host that drives a high-to-low edge to signal the beginning of each bit time. Commands and data are sent most significant bit first (MSB first). For a detailed description of the communications protocol, refer to Communication details.

If a host is attempting to communicate with a target MCU that has an unknown BDC clock rate, a SYNC command may be sent to the target MCU to request a timed sync response signal from which the host can determine the correct communication speed.

BKGD is a pseudo-open-drain pin and there is an on-chip pullup so no external pullup resistor is required. Unlike typical open-drain pins, the external RC time constant on this pin, which is influenced by external capacitance, plays almost no role in signal rise time. The custom protocol provides for brief, actively driven speedup pulses to force rapid rise times on this pin without risking harmful drive level conflicts. Refer to Communication details for more detail.

When no debugger pod is connected to the 6-pin BDM interface connector, the internal pullup on BKGD chooses normal operating mode. When a debug pod is connected to BKGD it is possible to force the MCU into active background mode after reset. The specific conditions for forcing active background depend upon the HCS08 derivative (refer to the introduction to this Development Support section). It is not necessary to reset the target MCU to communicate with it through the background debug interface.

### 19.2.2 Communication details

The BDC serial interface requires the external controller to generate a falling edge on the BKGD pin to indicate the start of each bit time. The external controller provides this falling edge whether data is transmitted or received.

BKGD is a pseudo-open-drain pin that can be driven either by an external controller or by the MCU. Data is transferred MSB first at 16 BDC clock cycles per bit (nominal speed). The interface times out if 512 BDC clock cycles occur between falling edges from the host. Any BDC command that was in progress when this timeout occurs is aborted without affecting the memory or operating mode of the target MCU system.

The custom serial protocol requires the debug pod to know the target BDC communication clock speed.

The clock switch (CLKSW) control bit in the BDC status and control register allows the user to select the BDC clock source. The BDC clock source can either be the bus or the alternate BDC clock source.

The BKGD pin can receive a high or low level or transmit a high or low level. The following diagrams show timing for each of these cases. Interface timing is synchronous to clocks in the target BDC, but asynchronous to the external host. The internal BDC clock signal is shown for reference in counting cycles.

The following figure shows an external host transmitting a logic 1 or 0 to the BKGD pin of a target HCS08 MCU. The host is asynchronous to the target so there is a 0-to-1 cycle delay from the host-generated falling edge to where the target perceives the beginning of the bit time. Ten target BDC clock cycles later, the target senses the bit level on the BKGD pin. Typically, the host actively drives the pseudo-open-drain BKGD pin during

host-to-target transmissions to speed up rising edges. Because the target does not drive the BKGD pin during the host-to-target transmission period, there is no need to treat the line as an open-drain signal during this period.

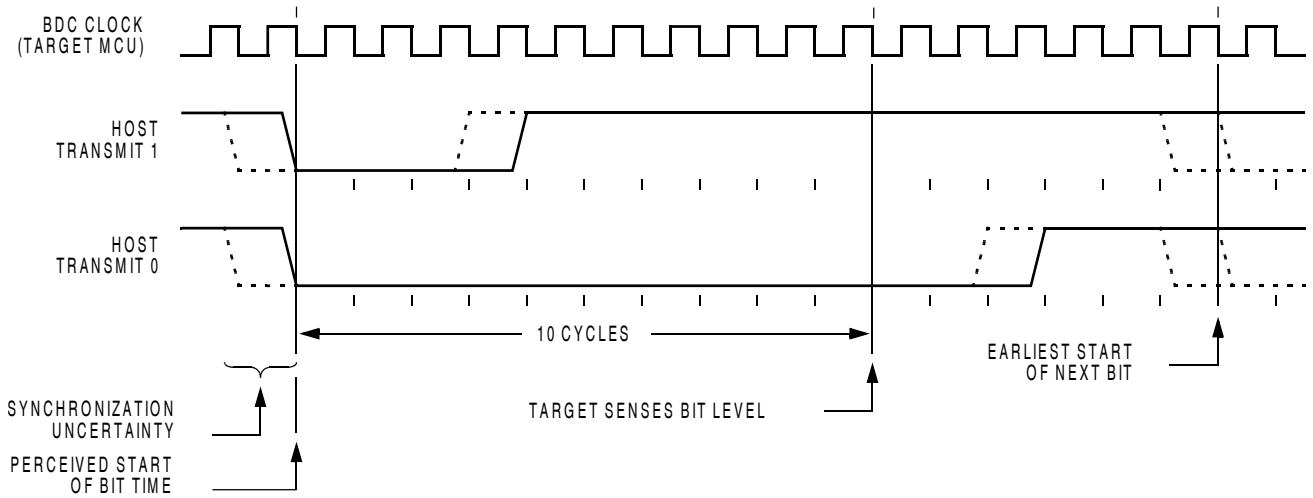



**Figure 19-2. BDC host-to-target serial bit timing**

The next figure shows the host receiving a logic 1 from the target HCS08 MCU. Because the host is asynchronous to the target MCU, there is a 0-to-1 cycle delay from the host-generated falling edge on BKGD to the perceived start of the bit time in the target MCU. The host holds the BKGD pin low long enough for the target to recognize it (at least two target BDC cycles). The host must release the low drive before the target MCU drives a brief active-high speedup pulse seven cycles after the perceived start of the bit time. The host should sample the bit level about 10 cycles after it started the bit time.

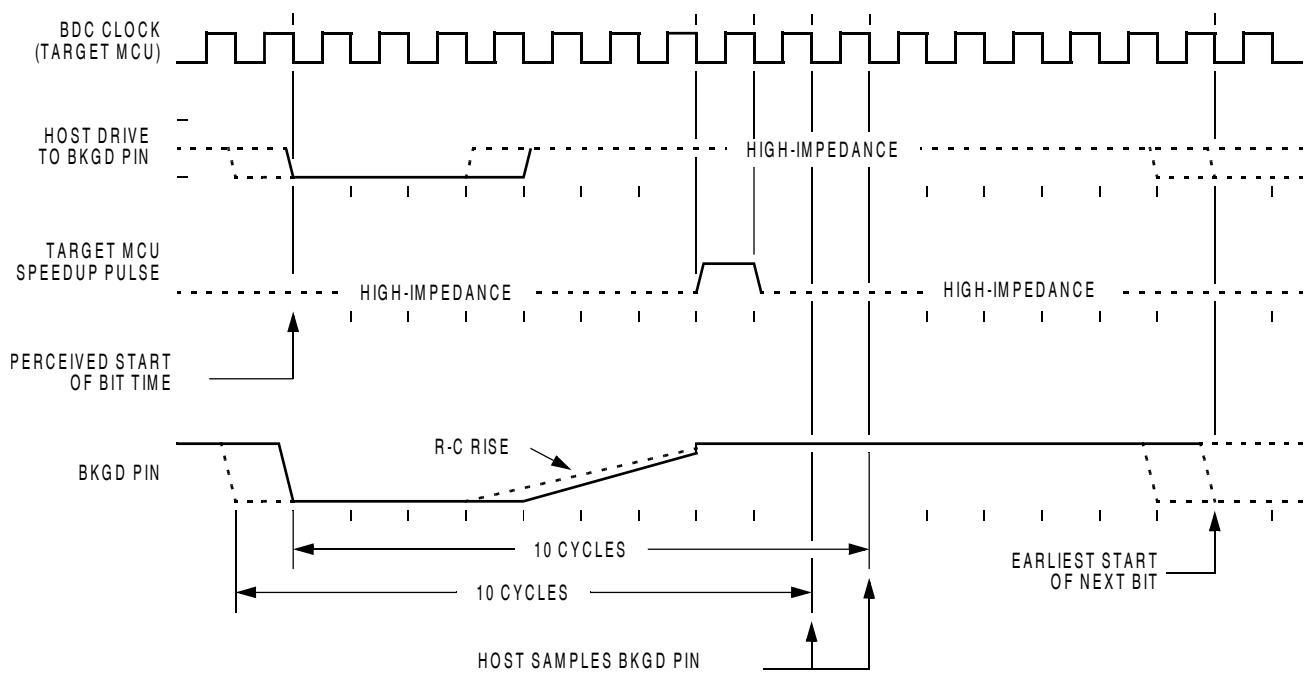



**Figure 19-3. BDC target-to-host serial bit timing (logic 1)**

The following figure shows the host receiving a logic 0 from the target HCS08 MCU. Because the host is asynchronous to the target MCU, there is a 0-to-1 cycle delay from the host-generated falling edge on BKGD to the start of the bit time as perceived by the target MCU. The host initiates the bit time but the target HCS08 finishes it. Because the target wants the host to receive a logic 0, it drives the BKGD pin low for 13 BDC clock cycles, then briefly drives it high to speed up the rising edge. The host samples the bit level about 10 cycles after starting the bit time.

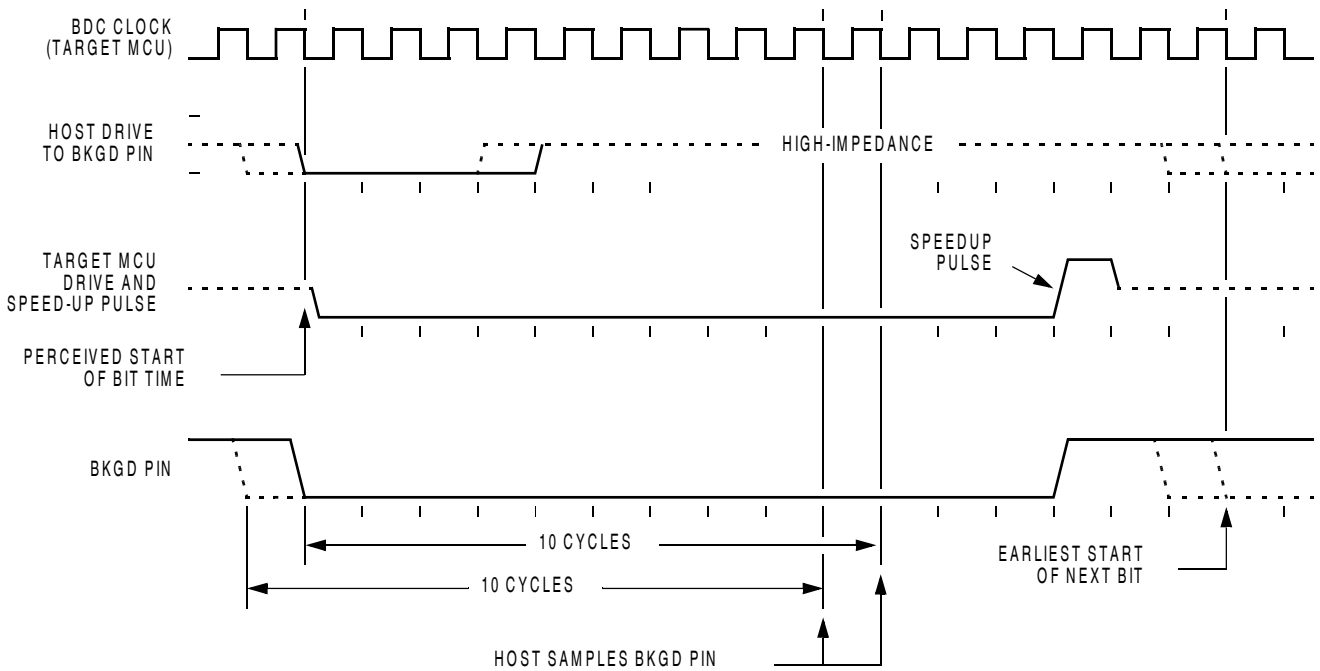



**Figure 19-4. BDM target-to-host serial bit timing (logic 0)**

## 19.2.3 BDC commands

BDC commands are sent serially from a host computer to the BKGD pin of the target HCS08 MCU. All commands and data are sent MSB-first using a custom BDC communications protocol. Active background mode commands require that the target MCU is currently in the active background mode while non-intrusive commands may be issued at any time whether the target MCU is in active background mode or running a user application program.

The following table shows all HCS08 BDC commands, a shorthand description of their coding structure, and the meaning of each command.

### Coding Structure Nomenclature

This nomenclature is used in the following table to describe the coding structure of the BDC commands.

```
Commands begin with an 8-bit hexadecimal command code in the host-to-target direction (most
significant bit first)
/ =separates parts of the command
d=delay 16 target BDC clock cycles
AAAA = a 16-bit address in the host-to-target direction
RD   = 8 bits of read data in the target-to-host direction
WD   = 8 bits of write data in the host-to-target direction
RD16 = 16 bits of read data in the target-to-host direction
WD16 = 16 bits of write data in the host-to-target direction
SS   = the contents of BDCSCR in the target-to-host direction (STATUS)
CC   = 8 bits of write data for BDCSCR in the host-to-target direction (CONTROL)
RBKP = 16 bits of read data in the target-to-host direction (from BDCBKPT breakpoint
register)
WBKP = 16 bits of write data in the host-to-target direction (for BDCBKPT breakpoint
register)
```

## Table 19-1. BDC command summary

| Command mnemonic | Active BDM/ non-intrusive | Coding structure | Description |
|---|---|---|---|
| SYNC | Non-intrusive | N/A[1] | Request a timed reference pulse to determine target BDC communication speed |
| ACK_ENABLE | Non-intrusive | D5/d | Enable acknowledge protocol. Refer to NXP document order no. HCS08RMv1/D. |
| ACK_DISABLE | Non-intrusive | D6/d | Disable acknowledge protocol. Refer to NXP document order no. HCS08RMv1/D. |
| BACKGROUND | Non-intrusive | 90/d | Enter active background mode if enabled (ignore if ENBDM bit equals 0) |
| READ_STATUS | Non-intrusive | E4/SS | Read BDC status from BDCSCR |
| WRITE_CONTROL | Non-intrusive | C4/CC | Write BDC controls in BDCSCR |
| READ_BYTE | Non-intrusive | E0/AAAA/d/RD | Read a byte from target memory |
| READ_BYTE_WS | Non-intrusive | E1/AAAA/d/SS/RD | Read a byte and report status |
| READ_LAST | Non-intrusive | E8/SS/RD | Re-read byte from address just read and report status |
| WRITE_BYTE | Non-intrusive | C0/AAAA/WD/d | Write a byte to target memory |
| WRITE_BYTE_WS | Non-intrusive | C1/AAAA/WD/d/SS | Write a byte and report status |
| READ_BKPT | Non-intrusive | E2/RBKP | Read BDCBKPT breakpoint register |
| WRITE_BKPT | Non-intrusive | C2/WBKP | Write BDCBKPT breakpoint register |
| GO | Active BDM | 08/d | Go to execute the user application program starting at the address currently in the PC |
| TRACE1 | Active BDM | 10/d | Trace 1 user instruction at the address in the PC, then return to active background mode |

*Table continues on the next page...*

**Table 19-1. BDC command summary (continued)**

| Command mnemonic | Active BDM/ non-intrusive | Coding structure | Description |
|---|---|---|---|
| TAGGO | Active BDM | 18/d | Same as GO but enable external tagging (HCS08 devices have no external tagging pin) |
| READ_A | Active BDM | 68/d/RD | Read accumulator (A) |
| READ_CCR | Active BDM | 69/d/RD | Read condition code register (CCR) |
| READ_PC | Active BDM | 6B/d/RD16 | Read program counter (PC) |
| READ_HX | Active BDM | 6C/d/RD16 | Read H and X register pair (H:X) |
| READ_SP | Active BDM | 6F/d/RD16 | Read stack pointer (SP) |
| READ_NEXT | Active BDM | 70/d/RD | Increment H:X by one then read memory byte located at H:X |
| READ_NEXT_WS | Active BDM | 71/d/SS/RD | Increment H:X by one then read memory byte located at H:X. Report status and data. |
| WRITE_A | Active BDM | 48/WD/d | Write accumulator (A) |
| WRITE_CCR | Active BDM | 49/WD/d | Write condition code register (CCR) |
| WRITE_PC | Active BDM | 4B/WD16/d | Write program counter (PC) |
| WRITE_HX | Active BDM | 4C/WD16/d | Write H and X register pair (H:X) |
| WRITE_SP | Active BDM | 4F/WD16/d | Write stack pointer (SP) |
| WRITE_NEXT | Active BDM | 50/WD/d | Increment H:X by one, then write memory byte located at H:X |
| WRITE_NEXT_WS | Active BDM | 51/WD/d/SS | Increment H:X by one, then write memory byte located at H:X. Also report status. |

1. The SYNC command is a special operation that does not have a command code.

The SYNC command is unlike other BDC commands because the host does not necessarily know the correct communications speed to use for BDC communications until after it has analyzed the response to the SYNC command.

To issue a SYNC command, the host:

- Drives the BKGD pin low for at least 128 cycles of the slowest possible BDC clock (The slowest clock is normally the reference oscillator/64 or the self-clocked rate/ 64.)
- Drives BKGD high for a brief speedup pulse to get a fast rise time (This speedup pulse is typically one cycle of the fastest clock in the system.)
- Removes all drive to the BKGD pin so it reverts to high impedance
- Monitors the BKGD pin for the sync response pulse

The target, upon detecting the SYNC request from the host (which is a much longer low time than would ever occur during normal BDC communications):

- Waits for BKGD to return to a logic high
- Delays 16 cycles to allow the host to stop driving the high speedup pulse
- Drives BKGD low for 128 BDC clock cycles
- Drives a 1-cycle high speedup pulse to force a fast rise time on BKGD
- Removes all drive to the BKGD pin so it reverts to high impedance

The host measures the low time of this 128-cycle sync response pulse and determines the correct speed for subsequent BDC communications. Typically, the host can determine the correct communication speed within a few percent of the actual target speed and the communication protocol can easily tolerate speed errors of several percent.

### 19.2.4 BDC hardware breakpoint

The BDC includes one relatively simple hardware breakpoint that compares the CPU address bus to a 16-bit match value in the BDCBKPT register. This breakpoint can generate a forced breakpoint or a tagged breakpoint. A forced breakpoint causes the CPU to enter active background mode at the first instruction boundary following any access to the breakpoint address. The tagged breakpoint causes the instruction opcode at the breakpoint address to be tagged so that the CPU will enter active background mode rather than executing that instruction if and when it reaches the end of the instruction queue. This implies that tagged breakpoints can be placed only at the address of an instruction opcode while forced breakpoints can be set at any address.

The breakpoint enable (BKPTEN) control bit in the BDC status and control register (BDCSCR) is used to enable the breakpoint logic (BKPTEN = 1). When BKPTEN = 0, its default value after reset, the breakpoint logic is disabled and no BDC breakpoints are requested regardless of the values in other BDC breakpoint registers and control bits. The force/tag select (FTS) control bit in BDCSCR is used to select forced (FTS = 1) or tagged (FTS = 0) type breakpoints.

The on-chip debug module (DBG) includes circuitry for two additional hardware breakpoints that are more flexible than the simple breakpoint in the BDC module.

## 19.3 On-chip debug system (DBG)

Because HCS08 devices do not have external address and data buses, the most important functions of an in-circuit emulator have been built onto the chip with the MCU. The debug system consists of an 8-stage FIFO that can store address or data bus information,

and a flexible trigger system to decide when to capture bus information and what information to capture. The system relies on the single-wire background debug system to access debug control registers and to read results out of the eight stage FIFO.

The debug module includes control and status registers that are accessible in the user's memory map. These registers are located in the high register space to avoid using valuable direct page memory space.

Most of the debug module's functions are used during development, and user programs rarely access any of the control and status registers for the debug module. The one exception is that the debug system can provide the means to implement a form of ROM patching. This topic is discussed in greater detail in Hardware breakpoints.

### 19.3.1 Comparators A and B

Two 16-bit comparators (A and B) can optionally be qualified with the R/W signal and an opcode tracking circuit. Separate control bits allow you to ignore R/W for each comparator. The opcode tracking circuitry optionally allows you to specify that a trigger will occur only if the opcode at the specified address is actually executed as opposed to only being read from memory into the instruction queue. The comparators are also capable of magnitude comparisons to support the inside range and outside range trigger modes. Comparators are disabled temporarily during all BDC accesses.

The A comparator is always associated with the 16-bit CPU address. The B comparator compares to the CPU address or the 8-bit CPU data bus, depending on the trigger mode selected. Because the CPU data bus is separated into a read data bus and a write data bus, the RWAEN and RWA control bits have an additional purpose, in full address plus data comparisons they are used to decide which of these buses to use in the comparator B data bus comparisons. If RWAEN = 1 (enabled) and RWA = 0 (write), the CPU's write data bus is used. Otherwise, the CPU's read data bus is used.

The currently selected trigger mode determines what the debugger logic does when a comparator detects a qualified match condition. A match can cause:

- Generation of a breakpoint to the CPU
- Storage of data bus values into the FIFO
- Starting to store change-of-flow addresses into the FIFO (begin type trace)
- Stopping the storage of change-of-flow addresses into the FIFO (end type trace)

### 19.3.2 Bus capture information and FIFO operation

The usual way to use the FIFO is to setup the trigger mode and other control options, then arm the debugger. When the FIFO has filled or the debugger has stopped storing data into the FIFO, you would read the information out of it in the order it was stored into the FIFO. Status bits indicate the number of words of valid information that are in the FIFO as data is stored into it. If a trace run is manually halted by writing 0 to ARM before the FIFO is full (CNT = 1:0:0:0), the information is shifted by one position and the host must perform ((8 - CNT) - 1) dummy reads of the FIFO to advance it to the first significant entry in the FIFO.

In most trigger modes, the information stored in the FIFO consists of 16-bit change-of-flow addresses. In these cases, read DBGFH then DBGFL to get one coherent word of information out of the FIFO. Reading DBGFL (the low-order byte of the FIFO data port) causes the FIFO to shift so the next word of information is available at the FIFO data port. In the event-only trigger modes (see Trigger modes), 8-bit data information is stored into the FIFO. In these cases, the high-order half of the FIFO (DBGFH) is not used and data is read out of the FIFO by simply reading DBGFL. Each time DBGFL is read, the FIFO is shifted so the next data value is available through the FIFO data port at DBGFL.

In trigger modes where the FIFO is storing change-of-flow addresses, there is a delay between CPU addresses and the input side of the FIFO. Because of this delay, if the trigger event itself is a change-of-flow address or a change-of-flow address appears during the next two bus cycles after a trigger event starts the FIFO, it will not be saved into the FIFO. In the case of an end-trace, if the trigger event is a change-of-flow, it will be saved as the last change-of-flow entry for that debug run.

The FIFO can also be used to generate a profile of executed instruction addresses when the debugger is not armed. When ARM = 0, reading DBGFL causes the address of the most-recently fetched opcode to be saved in the FIFO. To use the profiling feature, a host debugger would read addresses out of the FIFO by reading DBGFH then DBGFL at regular periodic intervals. The first eight values would be discarded because they correspond to the eight DBGFL reads needed to initially fill the FIFO. Additional periodic reads of DBGFH and DBGFL return delayed information about executed instructions so the host debugger can develop a profile of executed instruction addresses.

### 19.3.3 Change-of-flow information

To minimize the amount of information stored in the FIFO, only information related to instructions that cause a change to the normal sequential execution of instructions is stored. With knowledge of the source and object code program stored in the target system, an external debugger system can reconstruct the path of execution through many instructions from the change-of-flow information stored in the FIFO.

For conditional branch instructions where the branch is taken (branch condition was true), the source address is stored (the address of the conditional branch opcode). Because BRA and BRN instructions are not conditional, these events do not cause change-of-flow information to be stored in the FIFO.

Indirect JMP and JSR instructions use the current contents of the H:X index register pair to determine the destination address, so the debug system stores the run-time destination address for any indirect JMP or JSR. For interrupts, RTI, or RTS, the destination address is stored in the FIFO as change-of-flow information.

### 19.3.4 Tag vs. force breakpoints and triggers

Tagging is a term that refers to identifying an instruction opcode as it is fetched into the instruction queue, but not taking any other action until and unless that instruction is actually executed by the CPU. This distinction is important because any change-of-flow from a jump, branch, subroutine call, or interrupt causes some instructions that have been fetched into the instruction queue to be thrown away without being executed.

A force-type breakpoint waits for the current instruction to finish and then acts upon the breakpoint request. The usual action in response to a breakpoint is to go to active background mode rather than continuing to the next instruction in the user application program.

The tag vs. force terminology is used in two contexts within the debug module. The first context refers to breakpoint requests from the debug module to the CPU. The second refers to match signals from the comparators to the debugger control logic. When a tag-type break request is sent to the CPU, a signal is entered into the instruction queue along with the opcode so that if/when this opcode ever executes, the CPU will effectively replace the tagged opcode with a BGND opcode so the CPU goes to active background mode rather than executing the tagged instruction. When the TRGSEL control bit in the DBGT register is set to select tag-type operation, the output from comparator A or B is qualified by a block of logic in the debug module that tracks opcodes and produces only a trigger to the debugger if the opcode at the compare address is actually executed. There is separate opcode tracking logic for each comparator so more than one compare event can be tracked through the instruction queue at a time.

### 19.3.5 Trigger modes

The trigger mode controls the overall behavior of a debug run. The 4-bit TRG field in the DBGT register selects one of nine trigger modes. When TRGSEL = 1 in the DBGT register, the output of the comparator must propagate through an opcode tracking circuit before triggering FIFO actions. The BEGIN bit in DBGT chooses whether the FIFO begins storing data when the qualified trigger is detected (begin trace), or the FIFO stores data in a circular fashion from the time it is armed until the qualified trigger is detected (end trigger).

A debug run is started by writing a 1 to the ARM bit in the DBGC register, which sets the ARMF flag and clears the AF and BF flags and the CNT bits in DBGS. A begin-trace debug run ends when the FIFO gets full. An end-trace run ends when the selected trigger event occurs. Any debug run can be stopped manually by writing a 0 to ARM or DBGEN in DBGC.

In all trigger modes except event-only modes, the FIFO stores change-of-flow addresses. In event-only trigger modes, the FIFO stores data in the low-order eight bits of the FIFO.

The BEGIN control bit is ignored in event-only trigger modes and all such debug runs are begin type traces. When TRGSEL = 1 to select opcode fetch triggers, it is not necessary to use R/W in comparisons because opcode tags would apply only to opcode fetches that are always read cycles. It would also be unusual to specify TRGSEL = 1 while using a full mode trigger because the opcode value is normally known at a particular address.

The following trigger mode descriptions state only the primary comparator conditions that lead to a trigger. Either comparator can usually be further qualified with R/W by setting RWAEN (RWBEN) and the corresponding RWA (RWB) value to be matched against R/W. The signal from the comparator with optional R/W qualification is used to request a CPU breakpoint if BRKEN = 1 and TAG determines whether the CPU request will be a tag request or a force request.

**A-Only** – Trigger when the address matches the value in comparator A

**A OR B** – Trigger when the address matches either the value in comparator A or the value in comparator B

**A Then B** – Trigger when the address matches the value in comparator B but only after the address for another cycle matched the value in comparator A. There can be any number of cycles after the A match and before the B match.

**A AND B Data (Full Mode)**— This is called a full mode because address, data, and R/W (optionally) must match within the same bus cycle to cause a trigger event. Comparator A checks address, the low byte of comparator B checks data, and R/W is checked against RWA if RWAEN = 1. The high-order half of comparator B is not used.

In full trigger modes it is not useful to specify a tag-type CPU breakpoint (BRKEN = TAG = 1), but if you do, the comparator B data match is ignored for the purpose of issuing the tag request to the CPU and the CPU breakpoint is issued when the comparator A address matches.

**A AND NOT B Data (Full Mode)**— Address must match comparator A, data must not match the low half of comparator B, and R/W must match RWA if RWAEN = 1. All three conditions must be met within the same bus cycle to cause a trigger.

In full trigger modes it is not useful to specify a tag-type CPU breakpoint (BRKEN = TAG = 1), but if you do, the comparator B data match is ignored for the purpose of issuing the tag request to the CPU and the CPU breakpoint is issued when the comparator A address matches.

**Event-Only B (Store Data)**— Trigger events occur each time the address matches the value in comparator B. Trigger events cause the data to be captured into the FIFO. The debug run ends when the FIFO becomes full.

**A Then Event-Only B (Store Data)**— After the address has matched the value in comparator A, a trigger event occurs each time the address matches the value in comparator B. Trigger events cause the data to be captured into the FIFO. The debug run ends when the FIFO becomes full.

**Inside Range (A ≤ Address ≤ B)**— A trigger occurs when the address is greater than or equal to the value in comparator A and less than or equal to the value in comparator B at the same time.

**Outside Range (Address < A or Address > B)**— A trigger occurs when the address is either less than the value in comparator A or greater than the value in comparator B.

### 19.3.6 Hardware breakpoints

The BRKEN control bit in the DBGC register may be set to 1 to allow any of the trigger conditions described in Trigger modes to be used to generate a hardware breakpoint request to the CPU. TAG in DBGC controls whether the breakpoint request will be treated as a tag-type breakpoint or a force-type breakpoint. A tag breakpoint causes the current opcode to be marked as it enters the instruction queue. If a tagged opcode reaches

the end of the pipe, the CPU executes a BGND instruction to go to active background mode rather than executing the tagged opcode. A force-type breakpoint causes the CPU to finish the current instruction and then go to active background mode.

If the background mode has not been enabled (ENBDM = 1) by a serial WRITE_CONTROL command through the BKGD pin, the CPU will execute an SWI instruction instead of going to active background mode.

## 19.4 Memory map and register description

This section contains the descriptions of the BDCand DBG registers and control bits. Refer to the high-page register summary in the device overview chapter of this data sheet for the absolute address assignments for all DBG registers. This section refers to registers and control bits only by their names. An NXP-provided equate or header file is used to translate these names into the appropriate absolute addresses.

**BDC memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 0 | BDC Status and Control Register (BDC_SCR) | 8 | R/W | 00h | 19.4.1/373 |
| 1 | BDC Breakpoint Match Register: High (BDC_BKPTH) | 8 | R/W | 00h | 19.4.2/375 |
| 2 | BDC Breakpoint Register: Low (BDC_BKPTL) | 8 | R/W | 00h | 19.4.3/376 |
| 3 | System Background Debug Force Reset Register (BDC_SBDFR) | 8 | W (always reads 0) | 00h | 19.4.4/376 |

### 19.4.1 BDC Status and Control Register (BDC_SCR)

This register can be read or written by serial BDC commands (READ_STATUS and WRITE_CONTROL) but is not accessible to user programs because it is not located in the normal memory map of the MCU.

**NOTE**

The reset values shown in the register figure are those in the normal reset conditions. If the MCU is reset in BDM, ENBDM, BDMACT, CLKSW will be reset to 1 and others all be to 0.

Address: 0h base + 0h offset = 0h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | ENBDM | BDMACT | BKPTEN | FTS | CLKSW | WS | WSF | DVF |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## BDC_SCR field descriptions

| Field | Description |
|---|---|
| 7<br>ENBDM | Enable BDM (Permit Active Background Mode)<br><br>Typically, this bit is written to 1 by the debug host shortly after the beginning of a debug session or whenever the debug host resets the target and remains 1 until a normal reset clears it.<br><br>0 BDM cannot be made active (non-intrusive commands still allowed).<br>1 BDM can be made active to allow active background mode commands. |
| 6<br>BDMACT | Background Mode Active Status<br><br>This is a read-only status bit.<br><br>0 BDM not active (user application program running).<br>1 BDM active and waiting for serial commands. |
| 5<br>BKPTEN | BDC Breakpoint Enable<br><br>If this bit is clear, the BDC breakpoint is disabled and the FTS (force tag select) control bit and BDCBKPT match register are ignored.<br><br>0 BDC breakpoint disabled.<br>1 BDC breakpoint enabled. |
| 4<br>FTS | Force/Tag Select<br><br>When FTS = 1, a breakpoint is requested whenever the CPU address bus matches the BDCBKPT match register. When FTS = 0, a match between the CPU address bus and the BDCBKPT register causes the fetched opcode to be tagged. If this tagged opcode ever reaches the end of the instruction queue, the CPU enters active background mode rather than executing the tagged opcode.<br><br>0 Tag opcode at breakpoint address and enter active background mode if CPU attempts to execute that instruction<br>1 Breakpoint match forces active background mode at next instruction boundary (address need not be an opcode) |
| 3<br>CLKSW | Select Source for BDC Communications Clock<br><br>CLKSW defaults to 0, which selects the alternate BDC clock source.<br><br>0 Alternate BDC clock source.<br>1 MCU bus clock. |
| 2<br>WS | Wait or Stop Status<br><br>When the target CPU is in wait or stop mode, most BDC commands cannot function. However, the BACKGROUND command can be used to force the target CPU out of wait or stop and into active background mode where all BDC commands work. Whenever the host forces the target MCU into active background mode, the host should issue a READ_STATUS command to check that BDMACT = 1 before attempting other BDC commands. |

*Table continues on the next page...*

**BDC_SCR field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Target CPU is running user application code or in active background mode (was not in wait or stop mode when background became active).<br>1 Target CPU is in wait or stop mode, or a BACKGROUND command was used to change from wait or stop to active background mode. |
| 1<br>WSF | Wait or Stop Failure Status<br><br>This status bit is set if a memory access command failed due to the target CPU executing a wait or stop instruction at or about the same time. The usual recovery strategy is to issue a BACKGROUND command to get out of wait or stop mode into active background mode, repeat the command that failed, then return to the user program. (Typically, the host would restore CPU registers and stack values and re-execute the wait or stop instruction.)<br><br>0 Memory access did not conflict with a wait or stop instruction.<br>1 Memory access command failed because the CPU entered wait or stop mode. |
| 0<br>DVF | Data Valid Failure Status<br><br>0 Memory access did not conflict with a slow memory access<br>1 Memory access command failed because CPU was not finished with a slow memory access. |

## 19.4.2 BDC Breakpoint Match Register: High (BDC_BKPTH)

This register, together with BDC_BKPTL, holds the address for the hardware breakpoint in the BDC. The BKPTEN and FTS control bits in BDCSCR are used to enable and configure the breakpoint logic. Dedicated serial BDC commands (READ_BKPT and WRITE_BKPT) are used to read and write the BDCBKPT register but is not accessible to user programs because it is not located in the normal memory map of the MCU. Breakpoints are normally set while the target MCU is in active background mode before running the user application program.

Address: 0h base + 1h offset = 1h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | A[15:8] | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**BDC_BKPTH field descriptions**

| Field | Description |
|---|---|
| A[15:8] | High 8-bit of hardware breakpoint address. |

## 19.4.3 BDC Breakpoint Register: Low (BDC_BKPTL)

BDC_BKPTH and BDC_BKPTL registers hold the address for the hardware breakpoint in the BDC. The BDC_SCR[FTS] and BDC_SCR[BKPTEN] bits are used to enable and configure the breakpoint logic. Dedicated serial BDC commands (READ_BKPT and WRITE_BKPT) are used to read and write the BDC_BKPTH and BDC_BKPTL register. Breakpoints are normally set while the target MCU is in background debug mode before running the user application program. However, since READ_BKPT and WRITE_BKPT are foreground commands, they could be executed even while the user program is running.

Address: 0h base + 2h offset = 2h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | A[7:0] | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**BDC_BKPTL field descriptions**

| Field | Description |
|---|---|
| A[7:0] | Low 8-bit of hardware breakpoint address. |

## 19.4.4 System Background Debug Force Reset Register (BDC_SBDFR)

This register contains a single write-only control bit. A serial background mode command such as WRITE_BYTE must be used to write to SBDFR. Attempts to write this register from a user program are ignored. Reads always return 0x00.

Address: 0h base + 3h offset = 3h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | | | | | 0 |
| Write | | | | | | | | BDFR |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**BDC_SBDFR field descriptions**

| Field | Description |
|---|---|
| 7–1<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 0<br>BDFR | Background Debug Force Reset |

*Table continues on the next page...*

## BDC_SBDFR field descriptions (continued)

| Field | Description |
|---|---|
| | A serial active background mode command such as WRITE_BYTE allows an external debug host to force a target system reset. Writing 1 to this bit forces an MCU reset. This bit cannot be written from a user program. |

# Chapter 20
# Debug module (DBG)

## 20.1 Introduction

The DBG module implements an on-chip ICE (in-circuit emulation) system and allows non-intrusive debug of application software by providing an on-chip trace buffer with flexible triggering capability. The trigger also can provide extended breakpoint capacity. The on-chip ICE system is optimized for the S08CPUV6 8-bit architecture and supports 2 M bytes of memory space.

### 20.1.1 Features

The on-chip ICE system includes these distinctive features:

- Three comparators (A, B, and C) with ability to match addresses in 64 KB space
  - Dual mode, Comparators A and B used to compare addresses
  - Full mode, Comparator A compares address and Comparator B compares data
  - Can be used as triggers and/or breakpoints
  - Comparator C can be used as a normal hardware breakpoint
  - Loop1 capture mode, Comparator C is used to track most recent COF event captured into FIFO
- Tag and Force type breakpoints
- Nine trigger modes
  - A
  - A Or B
  - A then B
  - A and B, where B is data (full mode)
  - A and not B, where B is data (full mode)
  - Event only B, store data
  - A then event only B, store data
  - Inside range, A ≤ address ≤ B
  - Outside range, address < A or address > B

- FIFO for storing change of flow information and event only data
  - Source address of conditional branches taken
  - Destination address of indirect JMP and JSR instruction
  - Destination address of interrupts, RTI, RTC, and RTS instruction
  - Data associated with Event B trigger modes
- Ability to End-trace until reset and begin-trace from reset

### 20.1.2 Modes of operation

The on-chip ICE system can be enabled in all MCU functional modes. The DBG module is disabled if the MCU is secure. The DBG module comparators are disabled when executing a Background Debug Mode (BDM) command.

### 20.1.3 Block diagram

The following figure shows the structure of the DBG module.

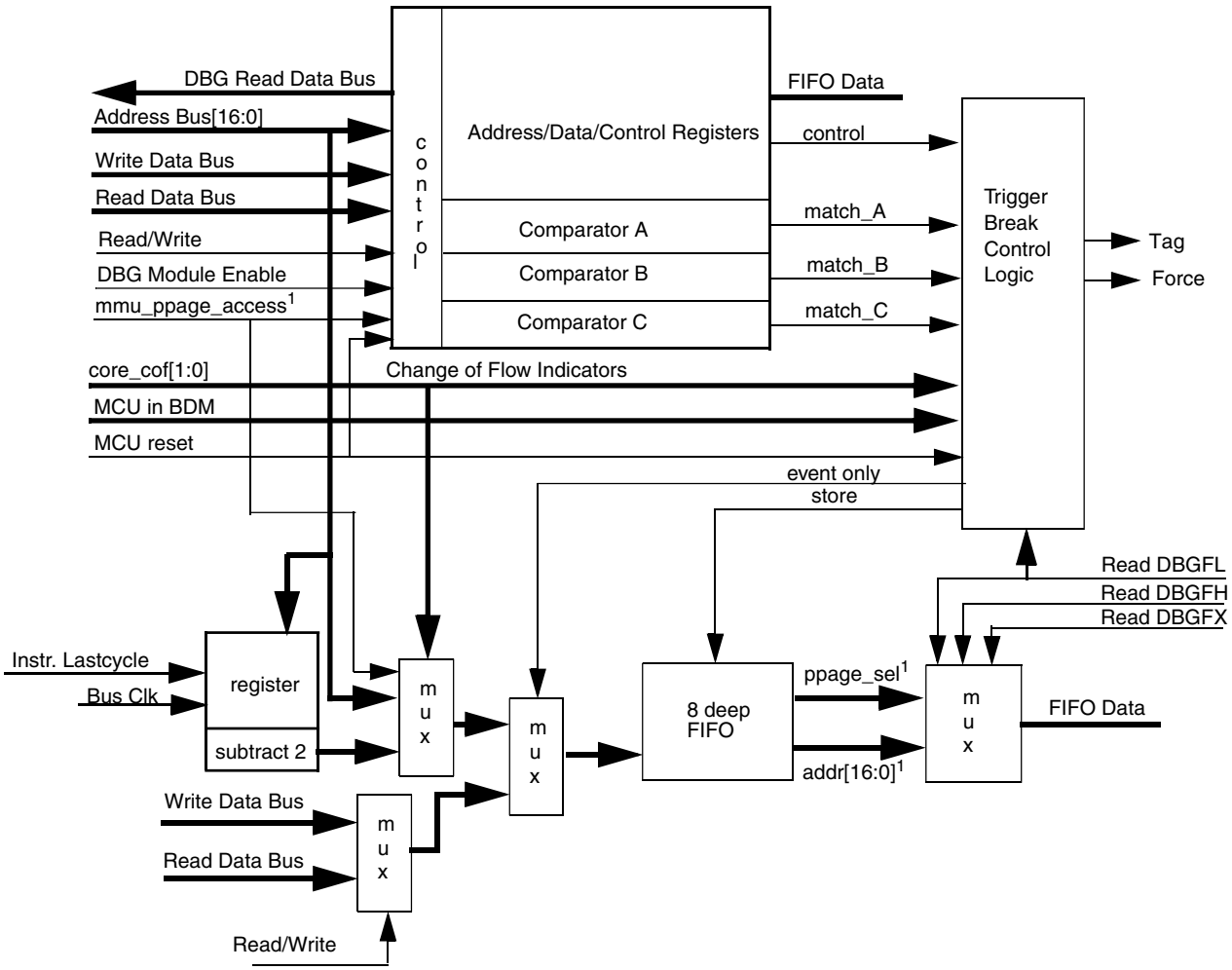



**Figure 20-1. DBG block diagram**

## 20.2 Signal description

The DBG module contains no external signals.

## 20.3 Memory map and registers

This section provides a detailed description of all DBG registers accessible to the end user.

**DBG memory map**

| Absolute address (hex) | Register name | Width (in bits) | Access | Reset value | Section/ page |
|---|---|---|---|---|---|
| 3010 | Debug Comparator A High Register (DBG_CAH) | 8 | R/W | FFh | 20.3.1/382 |
| 3011 | Debug Comparator A Low Register (DBG_CAL) | 8 | R/W | FEh | 20.3.2/383 |
| 3012 | Debug Comparator B High Register (DBG_CBH) | 8 | R/W | 00h | 20.3.3/384 |
| 3013 | Debug Comparator B Low Register (DBG_CBL) | 8 | R/W | 00h | 20.3.4/384 |
| 3014 | Debug Comparator C High Register (DBG_CCH) | 8 | R/W | 00h | 20.3.5/385 |
| 3015 | Debug Comparator C Low Register (DBG_CCL) | 8 | R/W | 00h | 20.3.6/386 |
| 3016 | Debug FIFO High Register (DBG_FH) | 8 | R | 00h | 20.3.7/386 |
| 3017 | Debug FIFO Low Register (DBG_FL) | 8 | R | 00h | 20.3.8/387 |
| 3018 | Debug Comparator A Extension Register (DBG_CAX) | 8 | R/W | 00h | 20.3.9/388 |
| 3019 | Debug Comparator B Extension Register (DBG_CBX) | 8 | R/W | 00h | 20.3.10/389 |
| 301A | Debug Comparator C Extension Register (DBG_CCX) | 8 | R/W | 00h | 20.3.11/390 |
| 301B | Debug FIFO Extended Information Register (DBG_FX) | 8 | R | 00h | 20.3.12/391 |
| 301C | Debug Control Register (DBG_C) | 8 | R/W | C0h | 20.3.13/391 |
| 301D | Debug Trigger Register (DBG_T) | 8 | R/W | 40h | 20.3.14/392 |
| 301E | Debug Status Register (DBG_S) | 8 | R | 01h | 20.3.15/394 |
| 301F | Debug Count Status Register (DBG_CNT) | 8 | R | 00h | 20.3.16/395 |

## 20.3.1 Debug Comparator A High Register (DBG_CAH)

**NOTE**

All the bits in this register reset to 1 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + 0h offset = 3010h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | CA[15:8] | | | | | | | |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**DBG_CAH field descriptions**

| Field | Description |
|---|---|
| CA[15:8] | Comparator A High Compare Bits<br><br>The Comparator A High compare bits control whether Comparator A will compare the address bus bits [15:8] to a logic 1 or logic 0. |

**DBG_CAH field descriptions (continued)**

| Field | Description |
|---|---|
| | 0 Compare corresponding address bit to a logic 0.<br>1 Compare corresponding address bit to a logic 1. |

## 20.3.2 Debug Comparator A Low Register (DBG_CAL)

**NOTE**

All the bits in this register reset to 1 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + 1h offset = 3011h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | CA[7:0] | | | | | | | |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

**DBG_CAL field descriptions**

| Field | Description |
|---|---|
| CA[7:0] | Comparator A Low<br><br>The Comparator A Low compare bits control whether Comparator A will compare the address bus bits [7:0] to a logic 1 or logic 0.<br><br>0 Compare corresponding address bit to a logic 0.<br>1 Compare corresponding address bit to a logic 1. |

## 20.3.3 Debug Comparator B High Register (DBG_CBH)

**NOTE**

All the bits in this register reset to 0 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + 2h offset = 3012h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | CB[15:8] | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_CBH field descriptions**

| Field | Description |
|---|---|
| CB[15:8] | Comparator B High Compare Bits<br><br>The Comparator B High compare bits control whether Comparator B will compare the address bus bits [15:8] to a logic 1 or logic 0.Not used in full mode.<br><br>0 Compare corresponding address bit to a logic 0.<br>1 Compare corresponding address bit to a logic 1. |

## 20.3.4 Debug Comparator B Low Register (DBG_CBL)

**NOTE**

All the bits in this register reset to 0 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + 3h offset = 3013h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | CB[7:0] | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_CBL field descriptions**

| Field | Description |
|---|---|
| CB[7:0] | Comparator B Low<br><br>The Comparator B Low compare bits control whether Comparator B will compare the address bus bits [7:0] to a logic 1 or logic 0.<br><br>0 Compare corresponding address bit to a logic 0.<br>1 Compare corresponding address bit to a logic 1. |

## 20.3.5 Debug Comparator C High Register (DBG_CCH)

**NOTE**

All the bits in this register reset to 0 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + 4h offset = 3014h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | CC[15:8] | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_CCH field descriptions**

| Field | Description |
|---|---|
| CC[15:8] | Comparator C High Compare Bits<br><br>The Comparator C High compare bits control whether Comparator C will compare the address bus bits [15:8] to a logic 1 or logic 0.<br><br>0 Compare corresponding address bit to a logic 0.<br>1 Compare corresponding address bit to a logic 1. |

## 20.3.6 Debug Comparator C Low Register (DBG_CCL)

**NOTE**

All the bits in this register reset to 0 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + 5h offset = 3015h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | CC[7:0] | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_CCL field descriptions**

| Field | Description |
|---|---|
| CC[7:0] | Comparator C Low<br><br>The Comparator C Low compare bits control whether Comparator C will compare the address bus bits [7:0] to a logic 1 or logic 0.<br><br>0 Compare corresponding address bit to a logic 0.<br>1 Compare corresponding address bit to a logic 1. |

## 20.3.7 Debug FIFO High Register (DBG_FH)

**NOTE**

All the bits in this register reset to 0 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + 6h offset = 3016h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | F[15:8] | | | | | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_FH field descriptions**

| Field | Description |
|---|---|
| F[15:8] | FIFO High Data Bits<br><br>The FIFO High data bits provide access to bits [15:8] of data in the FIFO. This register is not used in event only modes and will read a $00 for valid FIFO words. |

## 20.3.8 Debug FIFO Low Register (DBG_FL)

**NOTE**

All the bits in this register reset to 0 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + 7h offset = 3017h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | F[7:0] | | | | | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_FL field descriptions**

| Field | Description |
|---|---|
| F[7:0] | FIFO Low Data Bits<br><br>The FIFO Low data bits contain the least significant byte of data in the FIFO. When reading FIFO words, read DBGFX and DBGFH before reading DBGFL because reading DBGFL causes the FIFO pointers to advance to the next FIFO location. In event-only modes, there is no useful information in DBGFX and DBGFH so it is not necessary to read them before reading DBGFL. |

## 20.3.9 Debug Comparator A Extension Register (DBG_CAX)

**NOTE**

All the bits in this register reset to 0 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + 8h offset = 3018h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | RWAEN | RWA | 0 | | | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_CAX field descriptions**

| Field | Description |
|---|---|
| 7<br>RWAEN | Read/Write Comparator A Enable Bit<br><br>The RWAEN bit controls whether read or write comparison is enabled for Comparator A.<br><br>0 Read/Write is not used in comparison.<br>1 Read/Write is used in comparison. |
| 6<br>RWA | Read/Write Comparator A Value Bit<br><br>The RWA bit controls whether read or write is used in compare for Comparator A. The RWA bit is not used if RWAEN = 0.<br><br>0 Write cycle will be matched.<br>1 Read cycle will be matched. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 20.3.10 Debug Comparator B Extension Register (DBG_CBX)

**NOTE**

All the bits in this register reset to 0 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + 9h offset = 3019h

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | RWBEN | RWB | 0 | | | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_CBX field descriptions**

| Field | Description |
|---|---|
| 7<br>RWBEN | Read/Write Comparator B Enable Bit<br><br>The RWBEN bit controls whether read or write comparison is enabled for Comparator B. In full modes, RWAEN and RWA are used to control comparison of R/W and RWBEN is ignored.<br><br>0 Read/Write is not used in comparison.<br>1 Read/Write is used in comparison. |
| 6<br>RWB | Read/Write Comparator B Value Bit<br><br>The RWB bit controls whether read or write is used in compare for Comparator B. The RWB bit is not used if RWBEN = 0.In full modes, RWAEN and RWA are used to control comparison of R/W and RWB is ignored.<br><br>0 Write cycle will be matched.<br>1 Read cycle will be matched. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 20.3.11 Debug Comparator C Extension Register (DBG_CCX)

**NOTE**

All the bits in this register reset to 0 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + Ah offset = 301Ah

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | RWCEN | RWC | 0 | | | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_CCX field descriptions**

| Field | Description |
|---|---|
| 7<br>RWCEN | Read/Write Comparator C Enable Bit<br><br>The RWCEN bit controls whether read or write comparison is enabled for Comparator C.<br><br>0 Read/Write is not used in comparison.<br>1 Read/Write is used in comparison. |
| 6<br>RWC | Read/Write Comparator C Value Bit<br><br>The RWC bit controls whether read or write is used in compare for Comparator C. The RWC bit is not used if RWCEN = 0.<br><br>0 Write cycle will be matched.<br>1 Read cycle will be matched. |
| Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |

## 20.3.12 Debug FIFO Extended Information Register (DBG_FX)

**NOTE**

All the bits in this register reset to 0 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the bits in this register do not change after reset.

Address: 3010h base + Bh offset = 301Bh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | PPACC | 0 | | | | | | Bit16 |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_FX field descriptions**

| Field | Description |
|---|---|
| 7<br>PPACC | PPAGE Access Indicator Bit<br><br>This bit indicates whether the captured information in the current FIFO word is associated with an extended access through the PPAGE mechanism or not. This is indicated by the internal signal mmu_ppage_sel which is 1 when the access is through the PPAGE mechanism.<br><br>0 The information in the corresponding FIFO word is event-only data or an unpaged 17-bit CPU address with bit-16 = 0.<br>1 The information in the corresponding FIFO word is a 17-bit flash address with PPAGE[2:0] in the three most significant bits and CPU address[13:0] in the 14 least significant bits. |
| 6–1<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 0<br>Bit16 | Extended Address Bit 16<br><br>This bit is the most significant bit of the 17-bit core address. |

## 20.3.13 Debug Control Register (DBG_C)

Address: 3010h base + Ch offset = 301Ch

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read<br>Write | DBGEN | ARM | TAG | BRKEN | 0 | | | LOOP1 |
| Reset | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_C field descriptions**

| Field | Description |
|---|---|
| 7<br>DBGEN | DBG Module Enable Bit<br><br>The DBGEN bit enables the DBG module. The DBGEN bit is forced to zero and cannot be set if the MCU is secure.<br><br>0 DBG not enabled.<br>1 DBG enabled. |
| 6<br>ARM | Arm Bit<br><br>The ARM bit controls whether the debugger is comparing and storing data in FIFO.<br><br>0 Debugger not armed.<br>1 Debugger armed. |
| 5<br>TAG | Tag or Force Bit<br><br>The TAG bit controls whether a debugger or comparator C breakpoint will be requested as a tag or force breakpoint to the CPU. The TAG bit is not used if BRKEN = 0.<br><br>0 Force request selected.<br>1 Tag request selected. |
| 4<br>BRKEN | Break Enable Bit<br><br>The BRKEN bit controls whether the debugger will request a breakpoint to the CPU at the end of a trace run, and whether comparator C will request a breakpoint to the CPU.<br><br>0 CPU break request not enabled.<br>1 CPU break request enabled. |
| 3–1<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 0<br>LOOP1 | Select LOOP1 Capture Mode<br><br>This bit selects either normal capture mode or LOOP1 capture mode. LOOP1 is not used in event-only modes.<br><br>0 Normal operation - capture COF events into the capture buffer FIFO.<br>1 LOOP1 capture mode enabled. When the conditions are met to store a COF value into the FIFO, compare the current COF address with the address in comparator C. If these addresses match, override the FIFO capture and do not increment the FIFO count. If the address does not match comparator C, capture the COF address, including the PPACC indicator, into the FIFO and into comparator C.. |

## 20.3.14 Debug Trigger Register (DBG_T)

**NOTE**

The figure shows the values in POR or non-end-run reset. All the bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN=1 and BEGIN=0, the ARM and BRKEN bits are cleared but the remaining control bits in this register do not change after reset.

## NOTE

The DBG trigger register (DBGT) can not be changed unless ARM=0.

Address: 3010h base + Dh offset = 301Dh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | TRGSEL | BEGIN | 0 | | TRG | | | |
| Write | | | | | | | | |
| Reset | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

### DBG_T field descriptions

| Field | Description |
|---|---|
| 7<br>TRGSEL | Trigger Selection Bit<br><br>The TRGSEL bit controls the triggering condition for the comparators.<br><br>0 Trigger on any compare address access.<br>1 Trigger if opcode at compare address is execute. |
| 6<br>BEGIN | Begin/End Trigger Bit<br><br>The BEGIN bit controls whether the trigger begins or ends storing of data in FIFO.<br><br>0 Trigger at end of stored data.<br>1 Trigger before storing data. |
| 5–4<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| TRG | Trigger Mode Bits<br><br>The TRG bits select the trigger mode of the DBG module.<br><br>0000 A only.<br>0001 A or B.<br>0010 A then B.<br>0011 Event only B.<br>0100 A then event only B.<br>0101 A and B (full mode).<br>0110 A and not B (full mode).<br>0111 Inside range.<br>1000 Outside range.<br>1001-1111 No trigger. |

## 20.3.15 Debug Status Register (DBG_S)

**NOTE**

The figure shows the values in POR or non-end-run reset. The bits of AF, BF and CF are undefined and ARMF is reset to 0 in end-run reset. In the case of an end-trace to reset where DBGEN=1 and BEGIN=0, ARMF gets cleared by reset but AF, BF, and CF do not change after reset.

Address: 3010h base + Eh offset = 301Eh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | AF | BF | CF | 0 | | | | ARMF |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**DBG_S field descriptions**

| Field | Description |
|---|---|
| 7<br>AF | Trigger A Match Bit<br><br>The AF bit indicates if Trigger A match condition was met since arming.<br><br>0 Comparator A did not match.<br>1 Comparator A match. |
| 6<br>BF | Trigger B Match Bit<br><br>The BF bit indicates if Trigger B match condition was met since arming.<br><br>0 Comparator B did not match.<br>1 Comparator B match. |
| 5<br>CF | Trigger C Match Bit<br><br>The CF bit indicates if Trigger C match condition was met since arming.<br><br>0 Comparator C did not match.<br>1 Comparator C match. |
| 4–1<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| 0<br>ARMF | Arm Flag Bit<br><br>The ARMF bit indicates whether the debugger is waiting for trigger or waiting for the FIFO to fill. While DBGEN = 1, this status bit is a read-only image of the ARM bit in DBGC.<br><br>0 Debugger not armed.<br>1 Debugger armed. |

## 20.3.16 Debug Count Status Register (DBG_CNT)

**NOTE**

All the bits in this register reset to 0 in POR or non-end-run reset. The bits are undefined in end-run reset. In the case of an end-trace to reset where DBGEN = 1 and BEGIN = 0, the CNT[3:0] bits do not change after reset.

Address: 3010h base + Fh offset = 301Fh

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Read | 0 | | | | CNT | | | |
| Write | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**DBG_CNT field descriptions**

| Field | Description |
|---|---|
| 7–4<br>Reserved | This field is reserved.<br>This read-only field is reserved and always has the value 0. |
| CNT | FIFO Valid Count Bits<br><br>The CNT bits indicate the amount of valid data stored in the FIFO. Table 1-20 shows the correlation between the CNT bits and the amount of valid data in FIFO. The CNT will stop after a count to eight even if more data is being stored in the FIFO. The CNT bits are cleared when the DBG module is armed, and the count is incremented each time a new word is captured into the FIFO. The host development system is responsible for checking the value in CNT[3:0] and reading the correct number of words from the FIFO because the count does not decrement as data is read out of the FIFO at the end of a trace run.<br><br>0000 No data valid.<br>0001 1 word valid.<br>0010 2 words valid.<br>0011 3 words valid.<br>0100 4 words valid.<br>0101 5 words valid.<br>0110 6 words valid.<br>0111 7 words valid.<br>1000 8 words valid. |

## 20.4 Functional description

This section provides a complete functional description of the on-chip ICE system. The DBG module is enabled by setting the DBG_C[DBGEN] bit. Enabling the module allows the arming, triggering and storing of data in the FIFO. The DBG module is made up of three main blocks, the comparators, trigger break control logic and the FIFO.

### 20.4.1 Comparator

The DBG module contains three comparators, A, B, and C. Comparator A compares the core address bus with the address stored in the DBG_CAH and DBG_CAL registers. Comparator B compares the core address bus with the address stored in the DBG_CBH and DBG_CBL registers except in full mode, where it compares the data buses to the data stored in the DBG_CBL register. Comparator C compares the core address bus with the address stored in the DBG_CCH and DBG_CCL registers. Matches on comparators A, B, and C are signaled to the trigger break control (TBC) block.

#### 20.4.1.1 RWA and RWAEN in full modes

In full modes ("A And B" and "A And Not B") DBG_CAX[RWAEN and DBG_CAX[RWA] are used to select read or write comparisons for both comparators A and B. To select write comparisons and the write data bus in Full Modes set DBG_CAX[RWAEN] = 1 and DBG_CAX[RWA] = 0, otherwise read comparisons and the read data bus will be selected. The DBG_CBX[RWBEN] and DBG_CBX[RWB] bits are not used and will be ignored in full modes.

#### 20.4.1.2 Comparator C in loop1 capture mode

Normally comparator C is used as a third hardware breakpoint and is not involved in the trigger logic for the on-chip ICE system. In this mode, it compares the core address bus with the address stored in the DBG_CCX, DBG_CCH, and DBG_CCL registers. However, in loop1 capture mode, comparator C is managed by logic in the DBG module to track the address of the most recent change-of-flow event that was captured into the FIFO buffer. In loop1 capture mode, comparator C is not available for use as a normal hardware breakpoint.

When the DBG_C[ARM] and DBG_C[DBGEN] bits are set to one in loop1 capture mode, comparator C value registers are cleared to prevent the previous contents of these registers from interfering with the loop1 capture mode operation. When a COF event is detected, the address of the event is compared to the contents of the DBG_CCH and DBG_CCL registers to determine whether it is the same as the previous COF entry in the capture FIFO. If the values match, the capture is inhibited to prevent the FIFO from filling up with duplicate entries. If the values do not match, the COF event is captured into the FIFO and the DBG_CCH and DBG_CCL registers are updated to reflect the address of the captured COF event.

## 20.4.2 Breakpoints

A breakpoint request to the CPU at the end of a trace run can be created if the DBG_C[BRKEN] bit is set. The value of the DBG_T[BEGIN] bit determines when the breakpoint request to the CPU will occur. If the DBG_T[BEGIN] bit is set, begin-trigger is selected and the breakpoint request will not occur until the FIFO is filled with 8 words. If the DBG_T[BEGIN] bit is cleared, end-trigger is selected and the breakpoint request will occur immediately at the trigger cycle.

When traditional hardware breakpoints from comparators A or B are desired, set DBG_T[BEGIN] = 0 to select an end-trace run and set the trigger mode to either 0x0 (A-only) or 0x1 (A OR B) mode.

There are two types of breakpoint requests supported by the DBG module, tag-type and force-type. Tagged breakpoints are associated with opcode addresses and allow breaking just before a specific instruction executes. Force breakpoints are not associated with opcode addresses. The DBG_C[TAG] bit determines whether CPU breakpoint requests will be a tag-type or force-type breakpoints. When DBG_C[TAG] = 0, a force-type breakpoint is requested and it will take effect at the next instruction boundary after the request. When DBG_C[TAG] = 1, a tag-type breakpoint is registered into the instruction queue and the CPU will break if/when this tag reaches the head of the instruction queue and the tagged instruction is about to be executed.

### 20.4.2.1 Hardware breakpoints

Comparators A, B, and C can be used as three traditional hardware breakpoints whether the on-chip ICE real-time capture function is required or not. To use any breakpoint or trace run capture functions set DBG_C[DBGEN] = 1. DBG_C[BRKEN] and DBG_C[TAG] affect all three comparators. When DBG_C[BRKEN] = 0, no CPU breakpoints are enabled. When DBG_C[BRKEN] = 1, CPU breakpoints are enabled and the DBG_C[TAG] bit determines whether the breakpoints will be tag-type or force-type

breakpoints. To use comparators A and B as hardware breakpoints, set DBG_T = 0x81 for tag-type breakpoints and 0x01 for force-type breakpoints. This sets up an end-type trace with trigger mode "A OR B".

Comparator C is not involved in the trigger logic for the on-chip ICE system.

### 20.4.3 Trigger selection

The DBG_T[TRGSEL] bit is used to determine the triggering condition of the on-chip ICE system. DBG_T[TRGSEL] applies to both trigger A and B except in the event only trigger modes. By setting the DBG_T[TRGSEL] bit, the comparators will qualify a match with the output of opcode tracking logic. The opcode tracking logic is internal to each comparator and determines whether the CPU executed the opcode at the compare address. With the DBG_T[TRGSEL] bit cleared a comparator match is all that is necessary for a trigger condition to be met.

**NOTE**

If the DBG_T[TRGSEL] is set, the address stored in the comparator match address registers must be an opcode address for the trigger to occur.

### 20.4.4 Trigger break control (TBC)

The TBC is the main controller for the DBG module. Its function is to decide whether data should be stored in the FIFO based on the trigger mode and the match signals from the comparator. The TBC also determines whether a request to break the CPU should occur.

The DBG_C[TAG] bit controls whether CPU breakpoints are treated as tag-type or force-type breakpoints. The DBG_T[TRGSEL] bit controls whether a comparator A or B match is further qualified by opcode tracking logic. Each comparator has a separate circuit to track opcodes because the comparators could correspond to separate instructions that could be propagating through the instruction queue at the same time.

In end-type trace runs (DBG_T[BEGIN] = 0), when the comparator registers match, including the optional R/W match, this signal goes to the CPU break logic where DBG_C[BRKEN] determines whether a CPU break is requested and the DBG_C[TAG] control bit determines whether the CPU break will be a tag-type or force-type breakpoint. When DBG_T[TRGSEL] is set, the R/W qualified comparator match signal also passes through the opcode tracking logic. If/when it propagates through this logic, it will cause a

trigger to the ICE logic to begin or end capturing information into the FIFO. In the case of an end-type (DBG_T[BEGIN] = 0) trace run, the qualified comparator signal stops the FIFO from capturing any more information.

If a CPU breakpoint is also enabled, you would want DBG_C[TAG] and DBG_T[TRGSEL] to agree so that the CPU break occurs at the same place in the application program as the FIFO stopped capturing information. If DBG_T[TRGSEL] was 0 and DBG_C[TAG] was 1 in an end-type trace run, the FIFO would stop capturing as soon as the comparator address matched, but the CPU would continue running until a TAG signal could propagate through the CPUs instruction queue, which could take a long time in the case where changes of flow caused the instruction queue to be flushed. If DBG_T[TRGSEL] was one and DBG_C[TAG] was zero in an end-type trace run, the CPU would break before the comparator match signal could propagate through the opcode tracking logic to end the trace run.

In begin-type trace runs (DBG_T[BEGIN] = 1), the start of FIFO capturing is triggered by the qualified comparator signals, and the CPU breakpoint (if enabled by DBG_C[BRKEN]=1) is triggered when the FIFO becomes full. Since this FIFO full condition does not correspond to the execution of a tagged instruction, it would not make sense to use DBG_C[TAG] = 1 for a begin-type trace run.

#### 20.4.4.1 Begin- and end-trigger

The definition of begin- and end-trigger as used in the DBG module are as follows:

- Begin-trigger: storage in FIFO occurs after the trigger and continues until 8 locations are filled.
- End-trigger: storage in FIFO occurs until the trigger with the least recent data falling out of the FIFO if more than 8 words are collected.

#### 20.4.4.2 Arming the DBG module

Arming occurs by enabling the DBG module by setting the DBG_C[DBGEN] bit and by setting the DBG_C[ARM] bit. The DBG_C[ARM] and DBG_S[ARMF] bits are cleared when the trigger condition is met in end-trigger mode or when the FIFO is filled in begin-trigger mode. In the case of an end-trace where DBG_C[DBGEN] = 1 and DBG_T[BEGIN] = 0, DBG_C[ARM] and DBG_S[ARMF] are cleared by any reset to end the trace run that was in progress. The DBG_S[ARMF] bit is also cleared if DBG_C[ARM] is written to zero or when the DBG_C[DBGEN] bit is low. The TBC logic determines whether a trigger condition has been met based on the trigger mode and the trigger selection.

### 20.4.4.3 Trigger modes

The on-chip ICE system supports nine trigger modes. The trigger mode is used as a qualifier for either starting or ending the storing of data in the FIFO. When the match condition is met, the appropriate flag AF or BF is set in DBG_S register. Arming the DBG module clears the DBG_S[AF], DBG_S[BF], and DBG_S[CF] flags. In all trigger modes except for the event only modes change of flow addresses are stored in the FIFO. In the event only modes only the value on the data bus at the trigger event B comparator match address will be stored.

#### 20.4.4.3.1 A only

In the A only trigger mode, if the match condition for A is met, the DBG_S[AF] flag is set.

#### 20.4.4.3.2 A or B

In the A or B trigger mode, if the match condition for A or B is met, the corresponding flag(s) in the DBG_S register are set.

#### 20.4.4.3.3 A then B

In the A then B trigger mode, the match condition for A must be met before the match condition for B is compared. When the match condition for A or B is met, the corresponding flag in the DBG_S register is set.

#### 20.4.4.3.4 Event only B

In the event only B trigger mode, if the match condition for B is met, the DBG_S[BF] flag is set. The event only B trigger mode is considered a begin-trigger type and the DBG_T[BEGIN] bit is ignored.

#### 20.4.4.3.5 A then event only B

In the A then event only B trigger mode, the match condition for A must be met before the match condition for B is compared. When the match condition for A or B is met, the corresponding flag in the DBG_S register is set. The A then event only B trigger mode is considered a begin-trigger type and the DBG_T[BEGIN] bit is ignored.

#### 20.4.4.3.6 A and B (full mode)

In the A and B trigger mode, comparator A compares to the address bus and comparator B compares to the data bus. In the A and B trigger mode, if the match condition for A and B happen on the same bus cycle, both the DBG_S[AF] and DBG_S[BF] flags are set. If a match condition on only A or only B happens, no flags are set.

For breakpoint tagging operation with an end-trigger type trace, only matches from comparator A will be used to determine if the Breakpoint conditions are met and comparator B matches will be ignored.

#### 20.4.4.3.7 A and not B (full mode)

In the A and not B trigger mode, comparator A compares to the address bus and comparator B compares to the data bus. In the A and not B trigger mode, if the match condition for A and not B happen on the same bus cycle, both the DBG_S[AF] and DBG_S[BF] flags are set. If a match condition on only A or only not B occur no flags are set.

For breakpoint tagging operation with an end-trigger type trace, only matches from comparator A will be used to determine if the breakpoint conditions are met and comparator B matches will be ignored.

#### 20.4.4.3.8 Inside range, A ≤ address ≤ B

In the inside range trigger mode, if the match condition for A and B happen on the same bus cycle, both the DBG_S[AF] and DBG_S[BF] flags are set. If a match condition on only A or only B occur no flags are set.

#### 20.4.4.3.9 Outside range, address < A or address > B

In the outside range trigger mode, if the match condition for A or B is met, the corresponding flag in the DBGS register is set.

The four control bits DBG_T[BEGIN] and DBG_T[TRGSEL], and DBG_C[BRKEN] and DBG_C[TAG], determine the basic type of debug run as shown in the following table. Some of the 16 possible combinations are not used (refer to the notes at the end of the table).

**Table 20-1. Basic types of debug runs**

| BEGIN | TRGSEL | BRKEN | TAG | Type of debug run |
|---|---|---|---|---|
| 0 | 0 | 0 | x | Fill FIFO until trigger address (no CPU breakpoint - keep running) |
| 0 | 0 | 1 | 0 | Fill FIFO until trigger address, then force CPU breakpoint |
| 0 | 0 | 1 | 1 | Do not use |
| 0 | 1 | 0 | x | Fill FIFO until trigger opcode about to execute (no CPU breakpoint - keep running) |
| 0 | 1 | 1 | 0 | |
| 0 | 1 | 1 | 1 | Fill FIFO until trigger opcode about to execute (trigger causes CPU breakpoint) |
| 1 | 0 | 0 | x | Start FIFO at trigger address (No CPU breakpoint - keep running) |
| 1 | 0 | 1 | 0 | Start FIFO at trigger address, force CPU breakpoint when FIFO full |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | x | Start FIFO at trigger opcode (No CPU breakpoint - keep running) |
| 1 | 1 | 1 | 0 | Start FIFO at trigger opcode, force CPU breakpoint when FIFO full |
| 1 | 1 | 1 | 1 | |

## 20.4.5 FIFO

The FIFO is an eight word deep FIFO. In all trigger modes except for event only, the data stored in the FIFO will be change of flow addresses. In the event only trigger modes only the data bus value corresponding to the event is stored. In event only trigger modes, the high byte of the valid data from the FIFO will always read a 0x00.

### 20.4.5.1 Storing data in FIFO

In all trigger modes except for the event only modes, the address stored in the FIFO will be determined by the change of flow indicators from the core. The signal core_cof[1] indicates the current core address is the destination address of an indirect JSR or JMP instruction, or a RTS or RTI instruction or interrupt vector and the destination address should be stored. The signal core_cof[0] indicates that a conditional branch was taken and that the source address of the conditional branch should be stored.

### 20.4.5.2 Storing with begin-trigger

Storing with begin-trigger can be used in all trigger modes. Once the DBG module is enabled and armed in the begin-trigger mode, data is not stored in the FIFO until the trigger condition is met. Once the trigger condition is met the DBG module will remain armed until 8 words are stored in the FIFO. If the core_cof[1] signal becomes asserted, the current address is stored in the FIFO. If the core_cof[0] signal becomes asserted, the address registered during the previous last cycle is decremented by two and stored in the FIFO.

### 20.4.5.3 Storing with end-trigger

Storing with end-trigger cannot be used in event-only trigger modes. After the DBG module is enabled and armed in the end-trigger mode, data is stored in the FIFO until the trigger condition is met. If the core_cof[1] signal becomes asserted, the current address is stored in the FIFO. If the core_cof[0] signal becomes asserted, the address registered during the previous last cycle is decremented by two and stored in the FIFO. When the trigger condition is met, the DBG_C[ARM] and DBG_S[ARMF] will be cleared and no more data will be stored. In non-event only end-trigger modes, if the trigger is at a change of flow address the trigger event will be stored in the FIFO.

#### 20.4.5.4 Reading data from FIFO

The data stored in the FIFO can be read using BDM commands provided the DBG module is enabled and not armed (DBG_C[DBGEN] = 1 and DBG_C[ARM] = 0). The FIFO data is read out first-in-first-out. By reading the DBG_CNT[CNT] bits at the end of a trace run, the number of valid words can be determined. The FIFO data is read by optionally reading the DBG_FH register followed by the DBG_FL register. Each time the DBG_FL register is read, the FIFO is shifted to allow reading of the next word, however, the count does not decrement. In event-only trigger modes where the FIFO will contain only the data bus values stored, to read the FIFO only DBG_FL needs to be accessed.

The FIFO is normally read only while DBG_C[ARM] = 0 and DBG_S[ARMF] = 0, however, reading the FIFO while the DBG module is armed will return the data value in the oldest location of the FIFO and the TBC will not allow the FIFO to shift. This action could cause a valid entry to be lost because the unexpected read blocked the FIFO advance.

If the DBG module is not armed and the DBG_FL register is read, the TBC will store the current opcode address. Through periodic reads of the DBG_FH and DBG_FL registers while the DBG module is not armed, host software can provide a histogram of program execution This is called profile mode.

### 20.4.6 Interrupt priority

When DBG_T[TRGSEL] is set and the DBG module is armed to trigger on begin- or end-trigger types, a trigger is not detected in the condition where a pending interrupt occurs at the same time that a target address reaches the top of the instruction pipe. In these conditions, the pending interrupt has higher priority and code execution switches to the interrupt service routine.

When DBG_T[TRGSEL] is clear and the DBG module is armed to trigger on end-trigger types, the trigger event is detected on a program fetch of the target address, even when an interrupt becomes pending on the same cycle. In these conditions, the pending interrupt has higher priority, the exception is processed by the core and the interrupt vector is fetched. Code execution is halted before the first instruction of the interrupt service routine is executed. In this scenario, the DBG module will have cleared DBG_C[ARM] without having recorded the change-of-flow that occurred as part of the interrupt exception. Note that the stack will hold the return addresses and can be used to reconstruct execution flow in this scenario.

When DBG_T[TRGSEL] is clear and the DBG module is armed to trigger on begin-trigger types, the trigger event is detected on a program fetch of the target address, even when an interrupt becomes pending on the same cycle. In this scenario, the FIFO captures

the change of flow event. Because the system is configured for begin-trigger, the DBG remains armed and does not break until the FIFO has been filled by subsequent change of flow events.

## 20.5 Resets

The DBG module cannot cause an MCU reset.

There are two different ways this module will respond to reset depending upon the conditions before the reset event. If the DBG module was setup for an end trace run with DBG_C[DBGEN] = 1 and DBG_T[BEGIN] = 0, DBG_C[ARM], DBG_S[ARMF], and DBG_C[BRKEN] are cleared but the reset function on most DBG control and status bits is overridden so a host development system can read out the results of the trace run after the MCU has been reset. In all other cases including POR, the DBG module controls are initialized to start a begin trace run starting from when the reset vector is fetched. The conditions for the default begin trace run are:

- DBG_CAX = 0x00, DBG_CAH=0xFF, DBG_CAL=0xFE so comparator A is set to match when the 16-bit CPU address 0xFFFE appears during the reset vector fetch
- DBG_C = 0xC0 to enable and arm the DBG module
- DBG_T = 0x40 to select a force-type trigger, a BEGIN trigger, and A-only trigger mode

# Appendix A
# Revision History

**Table A-1. Revsion history**

| Rev. No. | Date | Substantial Changes |
|---|---|---|
| Rev.1 | 04/2018 | Initial public release. |
| Rev.2 | 07/2018 | • Updated the field name to NV_FPROT[FPHS] in Register addresses and bit assignments.<br>• Removed redundant registers in FTM.<br>• Added SCG_C2[CRC]. |

***How to Reach Us:***

**Home Page:**
nxp.com

**Web Support:**
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2018 NXP B.V.

Document Number MC9S08PL16RM
Revision 2, 07/2018