Freescale Semiconductor
Application Note

Document Number: AN4347
Rev. 0, 08/2011

Transitioning Applications from S08AC and S08FL Family to S08PT Family

by: Systems and Applications
Asia Pacific
Microcontroller Solutions Group

Freescale's S08PT family has the first S08 MCUs that employ the 0.18 micron process, a new version of the S08 CPU, and a new 5 V pad. This family also introduces many new features for different peripheral modules. With these new features, the S08PT family has lower power consumption, better transient protection, and better performance.

This application note outlines the product family differences between S08AC/FL and S08PT families, and provides recommendations for code conversion.

Contents

1 Flash and EEPROM 1
1.1 Protection 3
1.2 Commands 5
1.3 ECC parity check. 12
2 Clock gating. 13
3 Interrupt Priority Controller 13
4 S08 CPU V6 15
5 Enhanced code debug 16
6 Pinout changes 17
7 Parallel I/O structure 18
8 Safety feature enhancements 19
8.1 Watchdog timer 20
8.2 Cyclic redundancy check. 26
9 FlexTimer versus TPM 29
9.1 FTM clock source 30
9.2 TPM-compatible functions. 31
9.3 FTM-specific functions. 31
10 TSI addition 44
10.1 TSI function description. 45
10.2 TSI applications 46
11 RTC and MTIM additions. 48
11.1 Real time counter 48
11.2 Modulo timer 50
12 16-bit SPI with queue addition. 52
13 Analog-to-Digital Converter (ADC) 55
14 References 57
15 Glossary 57

1 Flash and EEPROM

The flash memory is ideal for single-supply applications, allowing for field reprogramming that does not require external high voltage sources for program or erase operations.

The S08PT family has both flash and EEPROM memory. Flash memory size is device-dependent with a maximum size of 64 KB. The EEPROM memory is 256 bytes. This

© Freescale Semiconductor, Inc., 2011. All rights reserved.

freescale™ semiconductor

family provides a security mechanism similar to that found in the S08AC and S08FL families, to prevent unauthorized access to the flash memory and EEPROM. Unsecuring the flash and EEPROM memory can also be done in similar way via Backdoor Key access, then an Erase All Blocks command (similar to Mass Erase of S08AC/FL families).

The S08PT family supports simultaneous flash and EEPROM operations. It is possible to read from flash memory while some commands are executing on EEPROM memory. However, it is not possible to read from EEPROM memory while a command such as Erase and Program is executing on flash memory.

The EEPROM memory is implemented with Error Correction Codes (ECC) that can resolve single-bit faults and detect double-bit faults, providing the best safety features among all three families.

An erase operation is performed on a sector base on devices in the S08PT family, which is similar to a page on devices in the S08AC/FL families. Each flash sector contains 512 bytes. Each EEPROM sector consists of two bytes. Therefore, the flash memory has up to 128 sectors, while EEPROM has 128 sectors.

Flash memory has these features:

- Automated program and erase algorithm with verify
- Fast sector erase and longword (32-bit) program operation
- Ability to read flash memory while programming a byte in EEPROM memory
- Flexible protection scheme to prevent accidental program or erase of flash memory
- Ability to set flash read margin levels

EEPROM memory has these features:

- 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
- Ability to set EEPROM read margin levels

Table 1 compares flash and EEPROM features among the S08AC/FL/PT families:

Table 1. Flash and EEPROM comparision

Features	S08AC	S08FL	S08PT	Unit
Flash security				
Flash protection				
EEPROM	X	X	256	Bytes
Flash size	128K	16K	64K	Bytes maximum
Flash sector/page size	512	512	512	Bytes
ECC parity check	X	X	Single fault bit resolve & Double fault bit detection on EEPROM	

Table 1. Flash and EEPROM comparision (continued)

Features	S08AC	S08FL	S08PT	Unit
Flash clock frequency (FCLK)	150k to 200K	150k to 200K	0.8M to 1M	Hz
Flash program bytes	1	1	8	Bytes maximum
EEPROM program bytes	X	X	4	Bytes maximum
Access time/byte read	1	1	1	Bus cycles
Program time	9 (byte program) 4 (burst program)	9 (byte program) 4 (burst program)	~68 (2 words) ~122 (4 words)	FCLK cycles
Page Erase time	4000	4000	~20015	FCLK cycles
Mass erase time	20,000	20000	~100066	FCLK cycles

1.1 Protection

Flash protection is performed on three separate memory regions in the flash memory: one that starts at global address 0x8000 and extends upward (low range), one that starts at global address 0xFFFF and extends downward (high range), and the remaining memory region. The protected address region can only be increased, not reduced, once it is implemented.

Flash protection and unprotection features are controlled via NVM_FPROT register bits, as listed in Table 2, Table 3, and Table 4.

Table 2. Flash protection function

FPOPEN	FPHDIS	FPLDIS	Function
1	1	1	No flash protection
1	1	0	Protect low range
1	0	1	Protect high range
1	0	0	Protect both low and high range
0	1	1	Protect full flash memory
0	1	0	Unprotect low range
0	0	1	Unprotect high range
0	0	0	Unprotect both high and low range

Table 3. Flash protection high range

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

Table 4. Flash protection low range

FPLS[1:0]	Global Address Range	Protected size
00	0x8000—0x83FF	1 KB
01	0x8000—0x87FF	2 KB
10	0x8000—0x8FFF	4 KB
11	0x8000—0x9FFF	8 KB

Figure 1 shows seven different protection schemes.

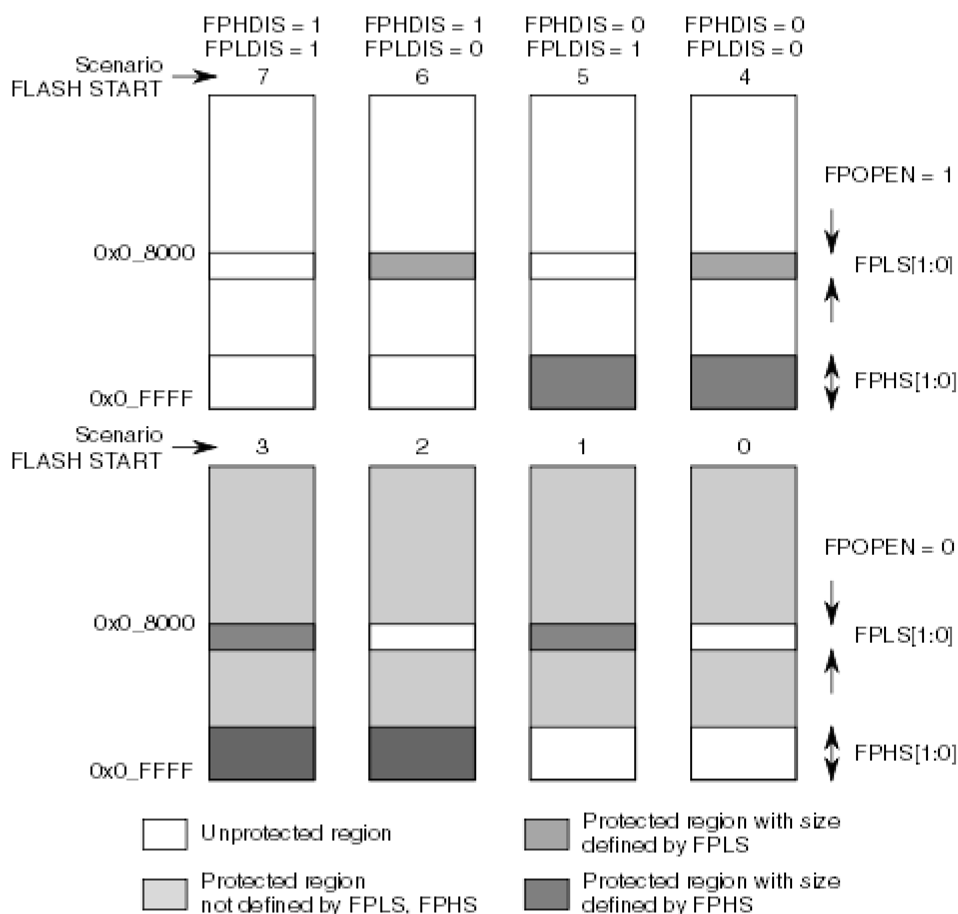


Figure 1. Flash protection schemes

Table 5 specifies all valid transitions between flash protection scenarios:

Table 5. Valid Flash protection scenario transitions

From protection scenario#	To protection scenario#							
	0	1	2	3	4	5	6	7
0	X	X	X	X				
1		X		X				
2			X	X				
3				X				
4				X	X			
5			X	X	X	X		
6		X		X	X		X	
7	X	X	X	X	X	X	X	X

EEPROM protection is controlled by NVM_EEPROT register bits. The protected region can only be increased, not reduced, once it is implemented. The NVM_EEPROT[DPOPEN] bit enables or disables the EEPROM protection. NVM_EEPROT[DPS2:0] bits define the protected address range as in Table 6.

Table 6. EEPROM protection address range

DPS[2:0]	Global address range	Protected size (bytes)
000	0x0_3100—0x0_311F	32
001	0x0_3100—0x0_313F	64
010	0x0_3100—0x0_315F	96
011	0x0_3100—0x0_317F	128
100	0x0_3100—0x0_319F	160
101	0x0_3100—0x031BF	192
The protection size continues to increase in increments of 32 bytes for each DPS value increase of one. . . .		
111	0x0_3100—0x0_31FF	256

1.2 Commands

The flash memory 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 (NVM_FCCOB) register which is written to with the command, global memory address, data, and any required command parameters. The memory controller must complete the execution of a command before the FCCOB register can be written to with a new command. The 16-bit NVM_FCCOB register includes the high-byte NVM_FCCOBHI and the low-byte NVM_FCCOBLO. When the flash command is completed and/or the flash error is detected, it will generate the corresponding interrupt, if enabled.

Table 7 shows the general command parameter format in NVM_FCCOB and the corresponding parameter index in NVM_FCCOBIX.

Table 7. General command parameter format in FCCOB

NVM_FCCOBIX[2:0]	FCCOB register meaning (the least significant bits are at the right side and most significant bits must be filled 0 if not used)	
	High byte	Low byte
	NVM_FCCOBHI	NVM_FCCOBLO
000	Flash command code(FCMD)	memory address bits[17: 16]
001	memory address bits[15 : 8]	memory address bits[7 : 0]
010	Data0 bits[15:8]	Data0 bits[7:0]
011	Data1 bits[15:8]	Data1 bits[7:0]
100	Data2 bits[15:8]	Data2 bits[7:0]
101	Data3 bits[15:8]	Data3 bits[7:0]

The command write sequence contains five steps:

1. Wait for the previous command to be completed, that is, NVM_FSTAT[CCIF] == 1.
2. Configure the flash clock to be 1 MHz by writing NVM_FCLKDIV, if it is not configured.
3. Check NVM_FSTAT[ACCERR] and NVM_FSTAT[FPVIOL] bits. Clear these bits if they are set by writing 0x30 to NVM_FSTAT.
4. Write the desired flash/EEPROM command and all the related parameters to NVM_FCCOB indexed by NVM_FCCOBIX[2:0] (the recommended write sequence is in ascending order of the command parameter format).
5. Launch the command by writing 0x80 to NVM_FSTAT to clear the NVM_FSTAT[CCIF] flag.

There are twelve commands for flash operation and nine commands for EEPROM operation, as shown in Table 8.

Table 8. Flash operation commands

Commands	Flash	EEPROM	Function Description
Erase verify all blocks	Y	Y	Verify that all blocks including flash and EEPROM are erased.
Erase verify block	Y	Y	Verify that the flash block or EEPROM block is erased.
Erase verify flash section	Y	—	Verify that a given number of words in the flash are erased.
Read Once	Y	—	Read a dedicated 64 B field in the nonvolatile information register of the flash block that was previously programmed using the program once command.
Program Once	Y	—	Program a dedicated 64 B field in the nonvolatile information register of the flash block.
Program flash	Y	—	Program up to two longwords in the flash block.
Erase all blocks	Y	Y	Erase all flash and EEPROM blocks.
Erase block	Y	Y	Erase a flash block or EEPROM block.
Erase flash sector	Y	—	Erase a flash sector.
Unsecure flash	Y	Y	Supports a method of releasing MCU security by erasing all flash and EEPROM blocks and verifying that all flash and EEPROM blocks are erased.
Verify backdoor access key	Y	—	Supports a method of releasing MCU security by verifying a set of security keys.

Table 8. Flash operation commands (continued)

Commands	Flash	EEPROM	Function Description
Set user margin level	Y	Y	Specify a user margin read level for the flash block or EEPROM block.
Erase verify EEPROM section	—	Y	Verify that a given number of bytes in EEPROM provided are erased.
Program EEPROM	—	Y	Program up to four bytes in the EEPROM block.
Erase EEPROM sector	—	Y	Erase all bytes in a sector of the EEPROM block.

The following sections describe some typical commands. For a full list of valid commands, please refer to Freescale document MC9S08PT60RM, *MC9S08PT60 Microcontroller Reference Manual.*

1.2.1 Erase flash/EEPROM

The commands in this group are:

- Erase Flash Sector
- Erase Flash Block
- Erase All Blocks
- Erase EEPROM Sector

The Erase Flash Sector command will erase all memory units in the specified flash sector, while the Erase Flash Block command will bulk-erase the entire flash memory space. The Erase All Blocks command will bulk-erase the entire flash memory space as well as the whole EEPROM memory space.

The Erase Flash/EEPROM Sector command is similar to the page erase command in the S08AC/FL families. The Erase Flash Block command and Erase All Blocks command are similar to the mass-erase command in S08AC/FL families.

The following code snippet shows how to erase a flash sector at address 0x8684:

```
// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = 0x0A;// Erase flash sector command
NVM_FCCOBLO = 0;// memory address bits[17:16]
// Write index to specify the memory address bits[15:0] to be loaded
NVM_FCCOBIX  = 0x1;
// Write the desired memory address bits[15:0] in the flash sector to be erased
NVM_FCCOB = 0x8684;

// Launch program command
NVM_FSTAT = 0x80;

// Wait till command is completed
While (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));
```

The following code snippet shows how to erase an EEPROM sector at address 0x3100:

```
// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = 0x12;// Erase EEPROM sector command
NVM_FCCOBLO = 0;// memory address bits[17:16]
// Write index to specify the memory address bits[15:0] to be loaded
NVM_FCCOBIX  = 0x1;
// Write the desired memory address bits[15:0] in the EEPROM sector to be erased
NVM_FCCOB = 0x3100;

// Launch program command
NVM_FSTAT = 0x80;

// Wait till command is completed
While (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));
```

1.2.2 Program flash/EEPROM

The Program Flash command will program up to two previously erased longwords in the flash memory, using an embedded algorithm. The memory locations to be programmed must be a longword or two longwords and their addresses must be aligned to a longword (4 B) boundary, that is, the lower two memory address bits[1:0] must be zero. The command code for Program Flash is 0x06.

The following code snippet shows how to program a longword 0x55667788 at 0x8684 in flash memory:

```
// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = 0x06;// program flash command
NVM_FCCOBLO = 0;// memory address bits[17:16]
// Write index to specify the longword memory address bits[15:0] to be loaded
NVM_FCCOBIX  = 0x1;
// Write the longword memory address bits[15:0]
NVM_FCCOB = 0x8684;
// Write index to specify the word0 of the longword to be programmed
NVM_FCCOBIX =  0x2;
// Write the word 0 of the longword
NVM_FCCOB = 0x7788;
// Write index to specify the word1 of the longword to be programmed
NVM_FCCOBIX =  0x3;
// Write the word 1 of the longword
NVM_FCCOB = 0x5566;

// Launch the command
NVM_FSTAT = 0x80;

// Wait till command is completed
While (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));
```

The following code snippet shows how to program a 4-byte 0x55667788 at address 0x3100 in EEPROM memory:

```
// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
```

```
NVM_FCCOBHI = 0x11;// program EEPROM command
NVM_FCCOBLO = 0;// memory address bits[17:16]
// Write index to specify the lower byte memory address bits[15:0] to be loaded
NVM_FCCOBIX  = 0x1;
// Write the lower byte memory address bits[15:0]
NVM_FCCOB = 0x3100;
// Write index to specify the byte0 to be programmed
NVM_FCCOBIX =  0x2;
// Write the byte 0
NVM_FCCOBLO = 0x88;
// Write index to specify the byte1 to be programmed
NVM_FCCOBIX =  0x3;
// Write the byte 1
NVM_FCCOBLO = 0x77;
// Write index to specify the byte2 to be programmed
NVM_FCCOBIX =  0x4;
// Write the byte 2
NVM_FCCOBLO = 0x66;
// Write index to specify the byte3 to be programmed
NVM_FCCOBIX =  0x5;
// Write the byte 3
NVM_FCCOBLO = 0x55;

// Launch the command
NVM_FSTAT = 0x80;

// Wait till command is completed
While (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));
```

1.2.3 Program Once and Read Once commands

The Program Once and Read Once commands are used to access a reserved 64 B (8 phrases) field in the nonvolatile information register (IFR) in the flash block. This reserved field cannot be erased and can be used to store the product ID or any other information that can only be written once. So, the Program Once command can only be used once.

The following code snippet shows how to program and read the second eight bytes, 0x1122334455667788, in the reserved 64 B field of IFR:

```
// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = 0x7;// program once command
// Write index to specify one of the 8 phrases in the reserved area once field
NVM_FCCOBIX  = 0x1;
// Write the phrase index to the reserved area once field
NVM_FCCOB = 0x0001;// 2nd phrases
// Write index to specify the word0 to be programmed
NVM_FCCOBIX =  0x2;
// Write the word 0
NVM_FCCOB = 0x7788;
// Write index to specify the word1 to be programmed
NVM_FCCOBIX =  0x3;
// Write the word1
NVM_FCCOB = 0x5566;
// Write index to specify the word2 to be programmed
```

```
NVM_FCCOBIX =  0x4;
// Write the word 2
NVM_FCCOB = 0x3344;
// Write index to specify the word3 to be programmed
NVM_FCCOBIX =  0x5;
// Write the word 3
NVM_FCCOB = 0x1122;

// Launch the command
NVM_FSTAT = 0x80;

// Wait till command is completed
While (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

/*  Use Read Once command to read the just programmed once phrases
into global variables gwIFRPhrases[0] to gwIFRPhrases[4]
*/
// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = 0x4;// read once command
// Write index to specify one of the 8 phrases in the reserved area once field
NVM_FCCOBIX  = 0x1;
// Write the phrase index to the reserved area once field
NVM_FCCOB = 0x0001;// 2nd phrases

// Launch the command
NVM_FSTAT = 0x80;

// Wait till command is completed
While (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));

// Write index to specify which word to read
NVM_FCCOBIX =  0x2;
// Read the word0 from once field
gwIFRPhrases[0] = NVM_FCCOB;
// Write index to specify which word to read
NVM_FCCOBIX =  0x3;
// Read the word1 from once field
gwIFRPhrases[1] = NVM_FCCOB;
// Write index to specify which word to read
NVM_FCCOBIX =  0x4;
// Read the word2 from once field
gwIFRPhrases[2] = NVM_FCCOB;
// Write index to specify which word to read
NVM_FCCOBIX =  0x5;
// Read the word3 from once field
gwIFRPhrases[3] = NVM_FCCOB;
```

1.2.4 User margin level

The user margin is a small delta to the normal read reference level. In effect it serves as a minimum safety margin. If the reads pass at the tighter tolerances of the user margins, then the normal reads have at least this much safety margin before a user will experience data loss. There are two user margin levels: user

margin-1 level, and user margin-0 level. User margin-1 level is the read margin to the erased state, while user margin-0 level is the read margin to the programmed state.

Users can use the Set User Margin level command to set the related user margin for the future read operation from flash/EEPROM. If user margin-1 level is set, and the future flash/EEPROM read returns a single-bit value of zero, then the flash read robustness is not guaranteed. If user margin-0 level is set, and the future flash/EEPROM read returns a bit of 1, then the flash read robustness is also not guaranteed.

This feature helps to ensure flash memory read robustness, and users can take the necessary action for safety purposes.

The NVM_FCCOBLO[7] bit is used to select between flash memory and EEPROM memory to be applied with the set user margin level:

- 0—flash memory to be applied
- 1—EEPROM memory to be applied

The following code snippet shows how to set the user margin-0 level for the EEPROM block at a specified address:

```
/*  Set user margin-0 level for the future flash operations
*/
// Write index to specify the command code to be loaded
NVM_FCCOBIX = 0x0;
// Write command code and memory address bits[17:16]
NVM_FCCOBHI = 0x0D;// set user margin level command
NVM_FCCOBLO = 0x80;// Bit 7 =1  selects EEPROM memory
// Write index to specify EEPROM address
NVM_FCCOBIX  = 0x1;
// Write the EEPROM address
NVM_FCCOB = 0x3100;//
// Write index to specify user margin level
NVM_FCCOBIX  = 0x2;
// Write the user margin-0 level
NVM_FCCOB = 0x0002;//

// Launch the command
NVM_FSTAT = 0x80;

// Wait till command is completed
While (!(NVM_FSTAT & NVM_FSTAT_CCIF_MASK));
```

1.3 ECC parity check

EEPROM supports ECC parity check. It supports single fault bit and double fault bit detection and also single fault bit resolve. This feature provides an additional safety mechanism for the system.

Both errors are reflected in the NVM_FERSTAT register. NVM_FERSTAT[DFDIF] indicates that a double fault bit error has occurred, while NVM_FERSTAT[SFDIF] indicates a single fault bit error that occurs if NVM_FCNFG[IGNSF] is cleared.

Both ECC errors can generate the corresponding interrupt if the corresponding bit in NVM_FERCNFG register is set.

There is a one-cycle delay in storing the ECC DFDIF] and SFDIF fault flags in NVM_FERSTAT register. At least one clock cycle is required after a flash memory read before checking NVM_FERSTAT for the occurrence of ECC errors.

2 Clock gating

The S08PT family 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 which are not in use and thereby reducing the MCU's 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).

In stop modes, the bus clock is disabled for all gated peripherals, regardless of the setting in the SCG_Cx registers.

The following code snippet is used to gate off the clock to TSI:

```
SCG_C4 &= ~(1<<2);
```

3 Interrupt Priority Controller

The Interrupt Priority Controller (IPC) is the same as in the S08FL family, but the S08AC family does not have it.

The IPC has these features:

- Four-level programmable interrupt priority for each interrupt source
- Support for prioritized preemptive interrupt service routines
  - — Lower priority interrupt requests are blocked when higher priority interrupts are being serviced
  - — Higher or equal priority level interrupt requests can preempt lower priority interrupts being serviced
- Automatic update of interrupt priority mask with the priority level of the interrupt source currently being serviced 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)

The interrupt priority level is programmable in the IPC_ILRSx register as below:

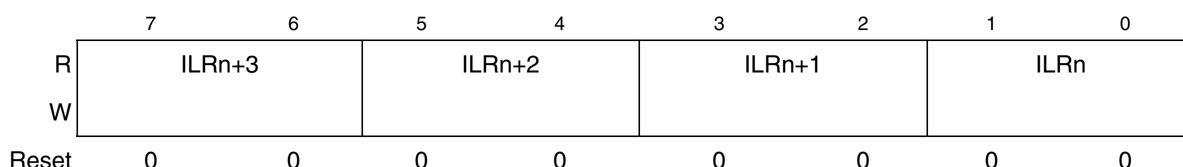


Figure 2. IPC ILRSx register

- ILRn bits corresponds to vector number <n>. To get the correct IPC_ILRSx register, divide the vector number <n> by 4 and the result quotient is <x>, that is, x = vector number <n>/4.
- Each interrupt priority level is a 2-bit value such that a user can program the interrupt priority level of each interrupt source to priority 0,1, 2, or 3.
- Level 3 has the highest priority while level 0 has the lowest.

The current interrupt priority mask (IPM) defined in IPC_SC register controls 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.

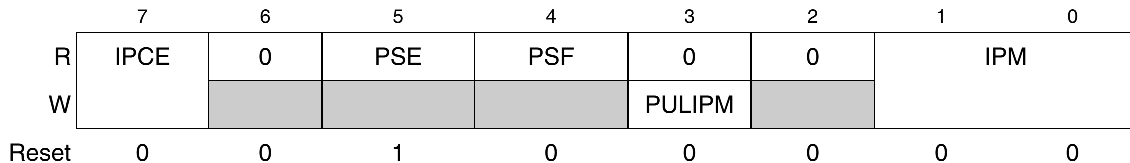


Figure 3. IPC SC register

The original value of the IPM will be saved to the Interrupt Priority Mask Pseudo Stack (IPC_IPMPS), which is a shift register for restoration after the interrupt service routine completes execution.

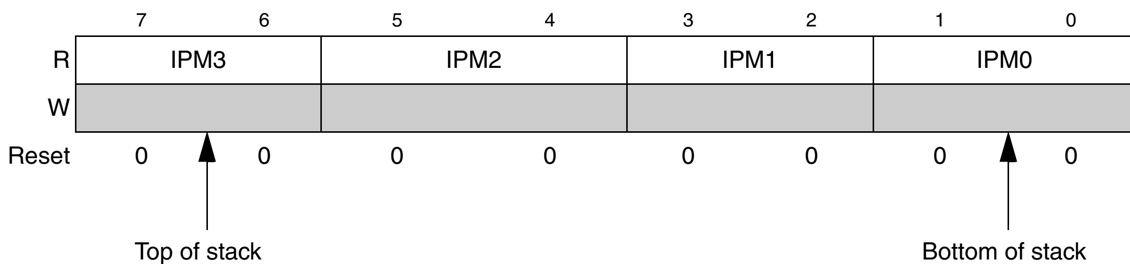


Figure 4. IPC IPM Pseudo Stack register

When the interrupt service routine completes execution, the user restores the original value of IPM by writing 1 to the IPC_SC[PULIPM] bit.

When the IPM is updated, the original value is shifted into IPC_IPMPS. The IPC_IPMPS can store four levels of IPM. If the last position of IPC_IPMPS is written to the area marked as “Top of stack” in Figure 4, the PSF flag is set and indicates that the IPC_IPMPS is full. If all the values in the IPMPS were read, the PSE flag indicates that the IPC_IPMPS is empty.

The interrupt priority arbitration scheme is:

- When the IPC is enabled by setting IPC_SC[IPCE] bit, the active interrupt source’s interrupt priority level (IPC_ILRSx[ILRn]) is compared with the current interrupt priority mask (IPC_SC[IPM]) (in stop3 mode, the IPC_SC[IPM] = 0x00); otherwise, the interrupt request signal from the source is directly passed to the CPU.
- If the ILRn value ≥ IPC_SC[IPM], the corresponding interrupt out (INTOUT) signal will be asserted and will signal an interrupt request to the HCS08 CPU.
- Because the IPC is an external module, the interrupt priority level programmed in the interrupt priority register IPC_ILRm will not affect the inherent interrupt priority arbitration as defined by

the HCS08 CPU. 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.

To support nesting interrupt, there are some requirements on an interrupt service routine (ISR):

- A minimum overhead of six bus clock cycles is added inside an ISR to enable preemptive interrupts due to added CLI and BSET instructions.
- Because interrupts of the same priority level are allowed to pass from the IPC to the HCS08 CPU, the flag generating the interrupt should 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 IPC_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 should be restored manually by setting the PULIPM bit. Watch out for the full (PSF) bit and empty (PSE) bit.

The following code snippet shows how to set up nesting interrupts:

```
/* Initializing interrupt priorities and IPC */
 /* Configure RTC IPL to 2 */
IPC_ILRS8_ILR35       = 2;

 /* Configure KBI0 PTA2 IPL to the highest level 3 */
 IPC_ILRS9_ILR37 = 3;
 IPC_SC_IPM  = 1;  /* set current IPM */
 IPC_SC_IPCE = 1; /* enable IPC */
/* Interrupt Serivce Routines (ISRs) */
interrupt VectorNumber_Vrtc void RTC_Isr(void)
{
   /*clear RTC flag by writing 1*/
   RTC_SC1 |= RTC_SC1_RTIF_MASK;
  /* do most critical work which can not be interrupted by higher priority interrupt.
asm (CLI); /* enable nesting interrupt */
```

4 S08 CPU V6

The S08PT family devices include an S08 CPU V6, which is the newest version of the S08 CPU. It has no 2× clock domain, but is still fully source - and object-code compatible with old versions of the S08 CPU.

The core clock and the system bus clock are the same. This greatly reduces the power consumption of the core by around 59% compared with the old HCS08 CPU, given the same performance.

In addition, S08 CPU V6 and V5 added two special instructions: CALL and RTC to support extended memory space exceeding 64 KB. This requires memory management unit (MMU) support. APPAGE register is used to manage 16 KB pages of memory which can be accessed by the CPU through a 16 KB window from 0x8000 through 0xBFFF. The MMU also includes a linear address pointer register and data access registers, so that the extended memory space operates as if it is a single linear block of memory.

CALL operates like the JSR instruction, except that CALL saves the current PPAGE value on the stack and provides a new PPAGE value for the destination. RTC works like the RTS instruction, except RTC restores the old PPAGE value in addition to the PC during the return from the called routine.

The S08PT family does not include MMU as the memory size is within 64 KB. As a result, both CALL and RTC instructions are of no use.

The S08FL family devices contain the S08 CPU V1. Most of the S08AC family devices like the MC9S08AC60 series include the S08 CPU V2. These devices have no CALL and RTC instructions. However, the MC9S08AC128 series devices include the S08 CPU V5, and the MMU is able to access the memory space exceeding 64 KB.

Table 9 shows the difference between these instructions among the different versions of the S08 CPUs:.

Table 9. CPU comparison

Instruction	CPU V6	CPU V5	CPU V4/V3/V2/V1	Description
RTS	6	6	5	Bus cycles
STOP	3+	2+	2	Bus cycles
CALL	9	8	N/A	Bus cycles
RTC	7	7	N/A	Bus cycles

5 Enhanced code debug

The S08PT family includes version three of the Debug Module (DBG V3). It implements an on-chip ICE (in-circuit emulation) system and allows non-intrusive debugging 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 MB of memory space.

The DBG V3 added the following enhanced features, compared with the old version:

- Support up to three hardware breakpoints (old version only supports two hardware breakpoints) by adding additional comparator (comparator C) logic
- End-trace until reset, which allows capturing change of flow (COF) information that caused reset event
- Begin-trace from reset, which enables capturing COF information as soon as reset occurs
- LOOP1 capture mode, which is efficient for tracing loop code

In the case of an end-trace to reset where DBGEN=1 and BEGIN=0, the bits in DBG A/B/C Comparator registers, FIFO registers, and Debug Count Status register, do not change after reset. The bits in Debug Trigger register do not change after reset. The Debug Status register ARMF bit gets cleared by reset but AF, BF, and CF do not change after reset. The ARM and BRKEN bits in the Debug Control register are cleared but other control bits do not change after reset. Those control and status bits unaffected by reset are overridden so a host development system can read out the results of the trace run after the MCU has been reset. This is different from a POR reset which will reset those registers to default reset values. The end-trace until reset is an important feature which allows capturing trace information that caused the reset event.

In LOOP1 capture mode, the 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 in the FIFO buffer. And in this case, the comparator C is not available for use as a normal hardware breakpoint. Therefore the number of hardware

breakpoints is reduced to two in LOOP1 capture mode. 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 Comparator C registers are updated to reflect the address of the captured COF event.

In addition to the above improved features in the debug module, the S08PT family also provides illegal address detect (ILAD) in addition to the illegal opcode detect (ILOP). This feature facilitates debugging the runaway code by capturing the errant addresses. When the MCU is reset by ILOP or ILAD, the address of the illegal opcode or illegal address is captured in the Illegal Address register. This is a 16-bit register consisting of SYS_ILLAL and SYS_ILLAH, which contain the LSB and MSB 8-bit of the address, respectively. There are two flags in the System Reset And Status register (SYS_SRS)to indicate such sources of reset: ILOP and ILAD. By checking the above register bits, users can easily know whether runaway code is caused by an illegal address and/or illegal opcode.

6 Pinout changes

On the S08PT family, the 5 V pad and internal layout are both optimized for transient protection.

These pins are general-purpose, bidirectional I/O port pins.

- PTA7–0 (except PTA4)
- PTB7–0
- PTC7–0
- PTD7–0
- PTE7–0
- PTF7–0
- PTG3–0
- PTH7–6
- PTH2–0

These port pins also have selectable pullup devices when configured for input mode (except PTA4).

The pulling devices are disengaged when configured for output mode and are selectable on an individual basis.

PTA4 is an output-only port pin. The port does not have pulling devices. PTA3 and PTA2 provide true open drain when operated as output.

PTB4–5, PTD0–1, PTE0–1, and PTH0–1 support high current function (regardless of their pin functionality) when they are configured as output. When high current function is enabled, they can drive output current. Each high current drive pin can drive a 20 mA sink/source current. These pins can directly drive devices requiring high current such as optocouplers (see Figure below). All of the other pins can offer a single 5 mA sink/source current. Only one or two pins are allowed to sink and/or source current at a time.

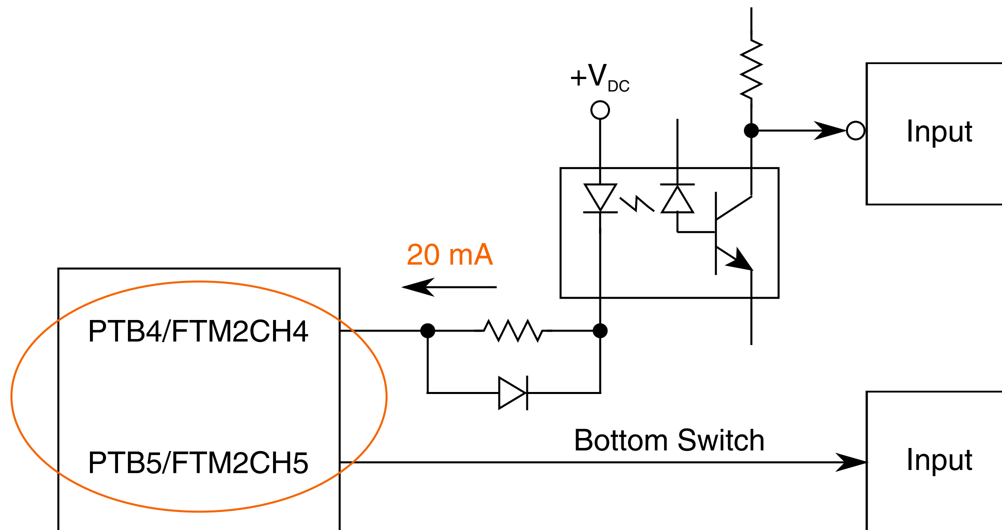


Figure 5. Direct connection between high current drive pins and optocouplers

After reset, all pins are configured as parallel I/O pins with pullup disabled and are high-impedance (Hi-Z). If any pin is not used, it should be configured to output for maximum system robustness.

7 Parallel I/O structure

S08PT family devices have eight sets of I/O ports, which include up to 57 general-purpose I/O pins. The I/O ports have these features:

- Data direction control
- Internal pullup control
- Extreme high drive control
- Input filter control

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. This provides the ability to read the actual state of a pin when configured as output by a peripheral, with the steps given here.

S08PT family devices use the PTxIEn register bit to enable the corresponding pin as input, and the PTxOEn register bit to enable the corresponding pin as output.

Both S08AC and S08FL families use PTxDD registers to control the direction of the corresponding pins.

Below is the example code snippet to configure a PTA0 pin as input and to read the pin state on the S08PT family:

```
/* read pin state on PTA0 pin */
PORT_PTAIE  |= 1;/* configure pin as input */
PORT_PTAPE |= 1;/* enable pullup on this pin */
PinState = PORT_PTAD & 1;
```

PTA4 is an output-only port pin. The port does not have pullup or pulldown devices. PTA3 and PTA2 provide true open drain when operated as output. All other ports are bidirectional I/O port pins.

These port pins also have selectable pullup devices when configured for input mode. The pulling devices are disengaged when configured for output mode. The pulling devices are selectable on an individual port bit basis in the PORT_PTxPE register. If the corresponding bit is set, the internal pullup is enabled; otherwise, it is disabled. For I^2C applications, the internal pullup on both pins can be used to save power when SDA/SCL outputs are low.

Output extreme high drive can drive 20 mA sink/source current on the enabled pins, as described in Section 7, "Parallel I/O structure."

This feature can be enabled by setting the corresponding bit in the PORT_HDRVE register for:

- PTH1
- PTH0
- PTE1
- PTE0
- PTD1
- PTD0
- PTB5
- PTB4

when they are operated as output. The extreme high drive strength is disabled if the pin is configured as an input. When configured as any shared peripheral function, the extreme high drive function strength still works on these pins, but only when they are configured as outputs.

This code shows how to enable extreme high drive on PTE1 pin:

```
/* Enabel extreme high drive on PTE1 pin */
PORT_PTEOE_PTEOE1 = 1;/* configure PTE1 pin as output pin */

PORT_HDRIVE_PTE1 = 1;/* enable extreme high drive on PTE1 pin */
```

A filter is implemented for each port pin set as digital input. It can be used as a simple low-pass filter and filters any glitch introduced from the pins of GPIO, IRQ, RESET, and KBI. The glitch width threshold can be adjusted easily by setting registers PORTP_IOFLTn and PORTP_FCLKDIV between 1–4096 bus clocks or 1–128 ms. This configurable glitch filter can take the place of all the other analog filters on board, and greatly improves the EMC performance on noise immunity.

8 Safety feature enhancements

Compared with the S08AC/FL families, the S08PT family has enhanced safety features in addition to the EEPROM ECC:

- Watchdog timer redesigned to be fully compatible with IEC60730 safety standard
- Programmable cyclic redundancy check (CRC) redesigned to support standard 16-bit and 32-bit CRC protocols for error detection

8.1 Watchdog timer

The watchdog timer module (WDOG) is an independent timer that is available for system use. If it is not updated with a certain data write within a certain period, it will reset the MCU. It is used as a safety element to make certain the software is executing as planned and that the CPU is not stuck in an infinite loop or executing unintended code.

WDOG has the following features:

- Configurable independent clock source input from bus clock
  - Internal 32 kHz RC oscillator
  - Internal 1 kHz RC oscillator
  - External clock source
- Programmable time-out period with 16-bit modulo value and optional fixed 256 clock prescaler
- Robust write sequence for counter refresh
  - Refresh sequence of writing 0xA602 and then 0xB480 within 16 bus clock cycles
- Windowed refresh option
  - Programmable 16-bit window value
  - Provides robust check that program flow is faster than expected
  - Refresh outside window leads to reset
- Programmable timeout post-processing
  - Interrupt request to CPU with interrupt vector allowing ISR
  - Force a reset after 128 bus clock cycles
- Robust write sequence for unlocking write-once control/configuration bits
  - Unlock sequence of writing 0xC520 and then 0xD928 within 16 bus clock cycles for allowing updates to write-once control/configuration bits
  - User need to update these after unlocking, within 128 bus clock cycles. Failure to update resets the system
- Configuration bits and registers are write-once-after-reset, to ensure watchdog configuration cannot be mistakenly altered (rewrite two preceding items for consistency)
- Flexible test mode enabling fast testing watchdog in the safety environment
- Backup reset to prevent hardware lockup condition driven by bus clock

S08AC/FL family COP watchdog configurations are listed in Table 10.

Table 10. Watchdog configurations for S08AC/FL

S08AC		S08FL			Clock source	Watchdog overflow count	Comments
COPCLKS	COPT	COPCLKS	COPT	COPW = 1 (COP Window)			
—	—	x	00	x	x	x	Watchdog disabled for S08FL; Watchdog disabled by clearing write-once bit SOPT[COPE] for S08AC
0	0	0	01	x	1 KHz internal	2^5 cycles	
0	1	0	10	x	1 KHz internal	2^8 cycles	
—	—	0	11	x	1 KHz internal	2^{10} cycles	Not applicable for S08AC
1	0	1	01	6144 cycles	Bus	2^{13} cycles	
—	—	1	10	49152 cycles	Bus	2^{16} cycles	Not applicable for S08AC
1	1	1	11	196,608 cycles	bus	2^{18} cycles	

For the watchdog clock source, the S08AC/FL families have two clock source options: internal 1 kHz low power oscillator, and bus clock. However, the S08PT family has additional two clock options: an internal 32 kHz RC oscillator (from ICSIRCLK), and external clock (from OSCOUT). The internal 1 kHz oscillator, 32 kHz oscillator, and external clock source allows an independent clock source from the bus clock for a robust application requirement.

On S08AC/FL families, resetting/clearing the watchdog counter is managed by writing 0x55 and 0xAA to the SRS register, in sequence. The S08FL family requires clearing the watchdog counter within the last 25% of the selected timeout period if the COP window is enabled.

Window mode allowsteh user to determine if the main loop is executing faster than expected. The S08AC family does not support watchdog window mode, while the S08FL and S08PT families support window mode.

The watchdog overflow count option is limited on both the S08AC and S08FL families, but the S08PT family has a programmable overflow count defined in a 16-bit timer register.

On the S08PT family, clearing the watchdog counter is conducted by writing 0xA602 and then 0xB480 to the WDOG_CNT register in sequence within 16 bus clock cycles. If the window mode is enabled, the watchdog must be refreshed after the watchdog counter has reached the window value and before it has reached the overflow value. Any invalid watchdog refresh done before the watchdog counter has reached the window value or after it has reached the overflow value will assert a fault, resetting the CPU.

All three families support write-once protection. This means that once a write has occurred the watchdog control bits 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, once configured. However, only the S08PT family supports unlock write-once protection, which allows reconfiguring all watchdog control bits without a reset.

The S08PT family provides another enhanced safety feature: backup reset. If the clock source (except bus clock) is selected as the reference clock, the backup reset function takes effect. In this case, if the watchdog counter continuously overflows twice, the watchdog will generate a reset by force. This function can prevent the state machine and other logic driven by the bus clock from pausing illegally.

Table 11 shows the feature comparisons among the S08PT and S08AC/FL families.

Table 11. Watchdog comparison

Features	S08PT	S08AC	S08FL
Independent clock source from bus	1K Hz 32K Hz External clock	1KHz	1KHz
Overflow count	16-bit programmable	Fixed	Fixed
Refresh sequence	Write 0xA602 to counter WDOG_CNT, then Write 0xB480 to counter	Write 0x55 to SRS, then Write 0xAA to SRS	Write 0x55 to SRS, then Write 0xAA to SRS
Window mode	Supported	Unsupported	Supported
Write-once protection	Yes	Yes	Yes
Unlock write-once protection	Supported	Unsupported	Unsupported
Test mode	Supported	Unsupported	Unsupported
Backup reset	Supported	Unsupported	Unsupported

The following sections describe S08PT watchdog usage.

8.1.1 Clock source

WDOG_CS2[WDOGC[1:0]] bits are used to select to one of four clock sources:

- Bus clock
- 1 kHz internal low-power oscillator
- 32 kHz internal RC oscillator
- External clock

The WDO_CS2[WDOGP] bit selects a fixed 256 prescaler for the clock source.

After reset, the watchdog is configured as 1 ms with internal 1 kHz clock source.

8.1.2 Window mode

WDOG_CS1[WDOGW] is used to enable window mode.

The starting point of the time window is T_{window}, and is defined by the WDOG_WIN register. Assume T_0 is the watchdog counter reset point or refreshing point, and $T_{overflow}$ is the watchdog counter overflow

point defined by watchdog timer register WDOG_TMR. Figure 6 shows the time window within which the watchdog must be refreshed:

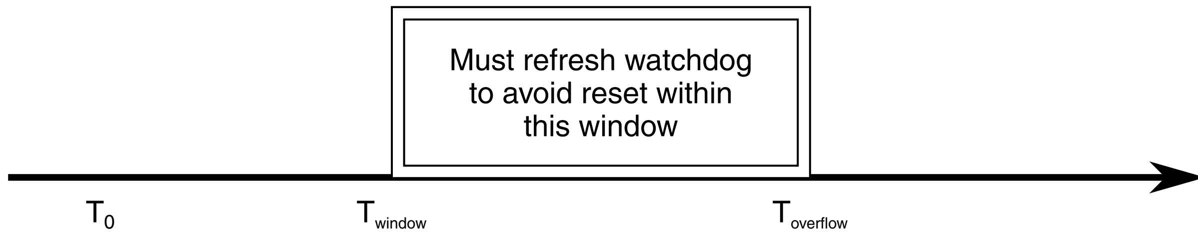


Figure 6. Watchdog window mode

8.1.3 Refreshing watchdog

Refreshing the watchdog is performed by this write sequence:

1. Write 0xA602 to WDOG_CNT.
2. Write 0xB480 to WDOG_CNT.

The second write must occur within 16 bus clock cycles after the first write happens.

8.1.4 Unlock write-once protection

All watchdog control bits, overflow value, and window value are write-once-after-reset. To update them, the unlock write-once protection mechanism must be used. To enable unlock write-once, WDOG_CS1[WDOGA] must be cleared. The default value of WDOG_CS1[WDOGA] is 0 after reset. If this bit is set, then all watchdog control bits are only write-once, which is similar to S08AC/FL families. In other words, once configured, the watchdog cannot be modified without forcing a reset.

The unlock write-once protection is performed with this unlock write sequence:

1. Write 0xC520 to WDOG_CNT.
2. Write 0xD928 to WDOG_CNT.

The second write must occur within 16 bus clock cycles after the first write happens.

Upon completion of the unlock write sequence, the watchdog write-once control bits should be reconfigured within 128 bus clock cycles. Otherwise a reset will occur.

8.1.5 Initialization

After reset, users can initialize the watchdog by performing these steps:

1. Write WDOG_WIN register and WDOG_TMR register to configure the window and overflow value first.
2. Write all other registers — except the watchdog counter register WDOG_CNT (CNTH:L) — while ensuring WDOG_CS1[WDOGA] = 0.

The new configuration will take effect only after all registers except WDOG_CNTH:WDOG_CNTL are written once.

Please note that the above procedure must be completed before watchdog timeout occurs.

This code snippet shows how to perform watchdog initialization:

```
void  wdog_user_mode(void)
{
    /* Write all 6 8-bit registers or 2 16-bit registers plus 2 8-bit registers */
    WDOG_TMR = 2000;/*  16-bit register, timeout at 2000 cycles */
    WDOG_WIN  = 1000;/* 16-bit register, window starts from 1000 cycles */
    WDOG_CS1 = 0x88;/* watchdog enabled, WDOGA = 0, WDOGT=01 (application user mode) */
    WDOG_CS2 = 0x83;/* window mode enabled, external clock source */
}
```

To disable the watchdog after reset, refer to this code snippet:

```
/* Write all 6 8-bit registers or 2 16-bit registers plus 2 8-bit registers */
WDOG_TMR = 2000;/*  16-bit register, timeout at 2000 cycles */
WDOG_WIN  = 1000;/* 16-bit register, window starts from 1000 cycles */
WDOG_CS1 = 0;/* watchdog disabled, WDOGA = 0 */
WDOG_CS2 = 0x81;/* window mode enabled, 1K Hz clock source */
```

8.1.6 Test mode

Generally watchdog timeouts are set to around 100 ms or longer. The watchdog has a feature to test the watchdog more quickly and help minimize the delay time after reset to start application code execution.

To test the watchdog quickly, the counter can be split into two blocks: low 8-bit block and high 8-bit block. In this case, the watchdog counter functions as an 8-bit counter rather than 16-bit counter.

To enable test mode, WDOG_CS1[WDOGT[1:0]] bits must be set to binary 10 or 11. These bits are cleared by a POR reset only and not affected by other resets.

Table 12. Test mode configuration

WDOG_CS1 WDOGT[1:0]	Test Mode	Comments
00	disabled	Reset only by POR
01	disabled	Application user mode
10	enabled	Test performed only on the lower 8-bit block (use only lower counter register and lower timer register): • WDOG_CNTL is compared with WDOG_TMRL • WDOG_CNTH and WDOG_TMRH are not used
11	enabled	Test performed only on the higher 8-bit block (use only higher counter register and higher timer register): • WDOG_CNTH is compared with WDOG_TMRH • WDOG_CNTL and WDOG_TMRL are not used

This code snippet shows how to do test mode after reset:

```
    void wdog_test_mode_low(void)
    {
        /* Write all 6 8-bit registers or 2 16-bit registers plus 2 8-bit registers */
        WDOG_TMR = 2000;/*  16-bit register, timeout at 2000 cycles */
        WDOG_WIN  = 1000;/* 16-bit register, window starts from 1000 cycles */
        WDOG_CS1 = 0x90;/* watchdog enabled, WDOGA = 0, WDOGT=10 ( lower 8-bit test mode)
        */
        WDOG_CS2 = 0x83;/* window mode enabled, external clock source */
    }
    void  wdog_test_mode_high(void)
    {
        /* Write all 6 8-bit registers or 2 16-bit registers plus 2 8-bit registers */
        WDOG_TMR = 2000;/*  16-bit register, timeout at 2000 cycles */
        WDOG_WIN  = 1000;/* 16-bit register, window starts from 1000 cycles */
        WDOG_CS1 = 0x98;/* watchdog enabled, WDOGA = 0, WDOGT=11 (higher 8-bit test mode)
        */
        WDOG_CS2 = 0x83;/* window mode enabled, external clock source */
    }

startup code:
    If( SYS_SRS_POR )
    {
        /* POR reset, conduct watchdog low test mode */
        wdog_test_mode_low();
    }
    else If(SYS_SRS_WDOG ) /* watchdog reset occurs but not POR */
    {
        /* check watchdog test mode bits */
        If( WDOG_CS1_WDOGT == 3)
        {
            /* higher 8-bit test mode generated watchdog reset */
            /* done test mode, and enable user mode */
```

```
                wdog_user_mode();
            }
            else if (WDOG_CS1_WDOGT == 2)
            {
                /* lower 8-bit test mode generated watchdog reset */
                /* enter watchdog high test mode */
                wdog_test_mode_high();
            }
        }
```

8.2 Cyclic redundancy check

The programmable cyclic redundancy check (CRC) generates 16/32-bit CRC code for error detection. The PCRC can be configured to work as any standard 16-bit or 32-bit CRC. It provides the user with programmable polynomial, seed value, and other parameters required to implement the required 16-bit or 32-bit CRC standard.

CRC also provides a feature to reverse input and output data by bit, and a feature to invert the final CRC result.

8.2.1 CRC initialization

To enable the CRC calculation, the user must program the SEED, POLYNOMIAL and necessary transpose type, and CRC result inversion parameters in their respective registers. Asserting the CRC_CTL[WAS] bit (SEED bit), enables the programming of the SEED value into the CRC data registers. Re-asserting the SEED bit and programming a new or same seed value after a previously completed CRC calculation will re-initialize the CRC for a new CRC computation. All other parameters must be set before programming the seed value and subsequent data values.

8.2.2 16-bit CRC calculation

These steps show the process to compute a 16-bit CRC:

1. Clear the CRC_CTRL[TCRC] bit to enable 16-bit CRC mode.
2. Program the transpose option bits (CRC_CTRL[TOT] for data write/input, CRC_CTRL[TOTR] for result read) and complement option bit (CRC_CTRL[FXOR]) in the CRC_CTRL register, as required for the CRC calculation.
3. Write 16-bit polynomial to CRC_P2:CRC_P3. CRC_P1:CRC_P0 are not usable in 16-bit CRC mode. CRC_P2 is the high byte of the 16-bit polynomial, CRC_P3 is the low byte.
4. Set SEED bit (CRC_CTRL[WAS]) to program the seed value.
5. Write 16-bit seed to CRC_D2:CRC_D3. CRC_D1:CRC_D0 are not used. CRC_D2 is the high byte of 16-bit seed, while CRC_D3 is the low byte.
6. Clear SEED bit (CRC_CTRL[WAS]) to start writing data values.
7. Write data values into CRC_D3. CRC is computed on every data value write, and the intermediate CRC result is stored back into CRC_D2:CRC_D3.
8. When all values have been written, the final CRC result can be read out from CRC_D2:CRC_D3. CRC_D2 is the high byte of the 16-bit CRC result, while CRC_D3 is the low byte.

9. Transpose and complement are performed dynamically while reading or writing values.

The next code snippet shows how to perform CCITT CRC-16:

```
CRC_CTRL  = 0;/* enable 16-bit CRC mode with no transpose */
CRC_P2P3  = 0x1021;/* write 16-bit CCITT polynomial 0x1021 */
                        /* Standard CCITT polynomial of (x^16 + x^12 + x^5 + 1) */
CRC_CTR_WAS = 1;/* enable SEED write */
CRC_D2D3 = 0xFFFF;/* write seed 0xFFFF */
CRC_CTRL_WAS = 0;/* disable SEED write, enable data write */
for ( i = 0 ; i < 256 ; i++ )
{
       CRC_D3 = 'A';/*  Dummy 256 `A' */
}
Checksum = CRC_D2D3; /* read CRC result */
```

8.2.3 32-bit CRC calculation

These steps show the process for computing a 32-bit CRC:

1. Set CRC_CTRL[TCRC] bit to enable 32-bit CRC mode.
2. Program the transpose option bits (CRC_CTRL[TOT] for data write/input, CRC_CTRL[TOTR] for result read) and complement option bit (CRC_CTRL[FXOR]) in the CRC_CTRL register, as required for the CRC calculation.
3. Write 32-bit polynomial to CRC_P0 to CRC_P3. CRC_P0 is the most significant byte (bit 31 to bit 24 of the 32-bit value) and CRC_P3 is the least significant byte (bit 7 to bit 0 of the 32-bit value).
4. Set SEED bit (CRC_CTRL[WAS]) to program the seed value.
5. Write 32-bit seed to CRC_D0 to CRC_D3. CRC_D0 is the most significant byte (bit 31 to bit 24 of the 32-bit value) and the CRC_D3 is the least significant byte (bit 7 to bit 0 of the 32-bit value).
6. Clear SEED bit (CRC_CTRL[WAS]) to start writing data values.
7. Write data values into CRC_D3. CRC is computed on every data value write and the intermediate CRC result is stored back into CRC_D0 to CRC_D3.
8. When all values have been written, the final CRC result can be read out from CRC_D0 to CRC_D3.
9. Transpose and complement are performed dynamically while reading or writing values.

8.2.4 Transpose feature

Some protocols use little endian format for the data stream to calculate CRC. In such a case, this feature comes in quite handy when performing such flipping of bits. By default, the transpose feature is disabled. Data is transposed dynamically while being read or written.

This version of CRC only supports bit transpose in a byte (CRC_CTRL[TOT]/CRC_CTRL[TOTR] = 01). For the TOT or TOTR bits, 10 and 11 are not valid values and should not be used.

The following figure shows the bit transpose/reverse in a byte:

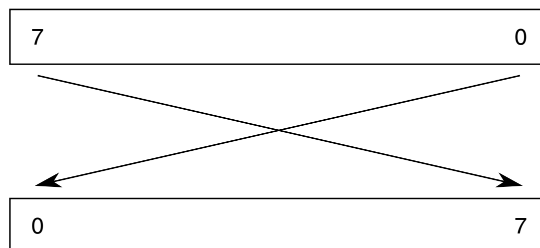


Figure 7. CRC bit transpose

8.2.5 Final XOR

The CRC result complement function outputs the complement of the checksum value in CRC data registers every time the CRC data register is read out. When FXOR bit in CRCCTL register is set, the checksum is complemented. Otherwise, the raw checksum is accessed.

8.2.6 Typical use cases

Table 13 lists some typical CRC standards and the corresponding settings:

Table 13. CRC typical use cases

Name	Poly	Seed	Final XOR?	Type of Transpose for Input	Type of Transpose for CRC Read	Standards
CRC-16	0x1021	0xFFFF (ITU-T V.41) 0x0000 (ITU-T T.30, X.25)	No	No transpose	No transpose	CRC-CCITT, ADCCP, SDLC/HDLC
CRC-16	0x1021	0	No	Transpose only bits in a byte	Transpose only bits in a byte	CRC-CCITT (Kermit)
XMODEM	0x8408	0x0000	No	Transpose only bits in a byte	Transpose only bits in a byte	XMODEM
ARC	0x8005	0x0000	No	Transpose only bits in a byte	Transpose only bits in a byte	ARC (zip file)
CRC-32	0x04C11DB 7	0xFFFFFFFF	Yes	Transpose only bits in a byte	Transpose both bits and bytes	PKZIP, AUTODIN II, Ethernet, FDDI

9 FlexTimer versus TPM

The FlexTimer module (FTM) on the S08PT family is designed based on the Timer/ PWM (TPM). It provides input capture, output compare, and generation of PWM signals. It is fully backward compatible with the TPM module used on the S08AC/FL families. It also provides these new enhanced features targeting motor control and power conversion applications:

- Complementary mode
- Combined mode, to generate asymmetric PWM
- Deadtime insertion hardware
- PWM synchronization control
- Fault control inputs
- Output masking
- Polarity control
- Enhanced triggering functionality

There are 3 FTMs on the S08PT family: FTM0, FTM1, and FTM2. Both FTM0 and FTM1 have two independent channels. FTM2 has six independent channels.

FTM register space is divided into two sets: one has fully TPM-compatible registers, and the other has FTM-specific registers.

Table 14 shows the FTM registers that are compatible with TPM.

Table 14. FlexTimer registers compatible with TPM

FlexTimer		Timer PWM (TPM)	
Register Name	**Address**	**Register Name**	**Address**
FTMx_SC	Base + 0000	TPMxSC	Base + 0000
FTMx_CNTH	Base + 0001	TPMxCNTH	Base + 0001
FTMx_CNTL	Base + 0002	TPMxCNTL	Base + 0002
FTMx_MODH	Base + 0003	TPMxMODH	Base + 0003
FTMx_MODL	Base + 0004	TPMxMODL	Base + 0004
FTMx_C0SC	Base + 0005	TPMxC0SC	Base + 0005
FTMx_C0VH	Base + 0006	TPMxC0VH	Base + 0006
FTMx_C0VL	Base + 0007	TPMxC0VL	Base + 0007
FTMx_C1SC	Base + 0008	TPMxC1SC	Base + 0008
FTMx_C1VH	Base + 0009	TPMxC1VH	Base + 0009
FTMx_C1VL	Base + 000A	TPMxC1VL	Base + 000A

Table 14. FlexTimer registers compatible with TPM (continued)

FlexTimer		Timer PWM (TPM)	
Register Name	**Address**	**Register Name**	**Address**
…	…	…	…
FTMx_C5SC	Base + 0014	TPMxC5SC	Base + 0014
FTMx_C5VH	Base + 0015	TPMxC5VH	Base + 0015
FTMx_C5VL	Base + 0016	TPMxC5VL	Base + 0016

For FTM0 and FTM1, the shaded areas in the above table are holes and are considered to be illegal when accessing them.

Table 15 shows the FTM-specific register map:

Table 15. FlexTimer-specific registers

Register Name	Address	Register Name	Address
FTMx_CNTINH	Base + 0017	Reserved	Base + 0021
FTMx_CNTINL	Base + 0018	FTMx_DEADTIME	Base + 0022
FTMx_STATUS	Base + 0019	FTMx_EXTTRIG	Base + 0023
FTMx_MODE	Base + 001A	FTMx_POL	Base + 0024
FTMx_SYNC	Base + 001B	FTMx_FMS	Base + 0025
FTMx_OUTINIT	Base + 001C	FTMx_FILTER0	Base + 0026
FTMx_OUTMASK	Base + 001D	FTMx_FILTER1	Base + 0027
FTMx_COMBINE0	Base + 001E	FTMx_FLTFILTER	Base + 0028
FTMx_COMBINE1	Base + 001F	FTMx_FLTCTRL	Base + 0029
FTMx_COMBINE2	Base + 0020		

9.1 FTM clock source

The FTM clock source can be the system clock divided by two or the system clock, depending on the FTMx_MODE[FTMEN] bit, the fixed system clock, and the external clock. The fixed system clock is ICSFFCLK. The external clock must not exceed one-fourth (¼) of the system clock.

Table 16 shows the clock source selection:

Table 16. FlexTimer-specific registers

FTMx_SC[CLKS]	FTMx_MODE[FTMEN]	FTM Clock Source
00	X	No Clock selected. FTM disabled
01	0	System Clock divided by 2 (bus clock)
	1	System Clock
10	X	Fixed System Clock
11	X	External Clock

9.2 TPM-compatible functions

To use TPM-compatible functions, make sure that FTMx_MOD[FTMEN] is zero, and do not write to FTM-specific registers. Instead, write to TPM-compatible registers as is done on S08AC/FL families.

9.3 FTM-specific functions

FTM provides specific functions which are enhanced features to TPM. Prior to accessing any FTM-specific function, FTMx_MOD[FTMEM] must be set.

9.3.1 Combined mode

In combined mode, the channel (*n*) (an even channel) and channel (*n*+1) (the adjacent odd channel) are combined together as a PWM channel pair *n*. This is then used to create a single PWM signal which can be used for asymmetrical PWM/phase shift PWM. This feature is widely used in power conversion applications.

The FTMx_COMBINEn register controls the PWM channel *n* and *n*+1 pair. In order for a PWM channel pair *n* to enter combined mode, these conditions must be met:

- FTMx_MODE[FTMEN] = 1
- FTMx_COMBINEn[COMBINE] = 1
- FTMx_COMBINEn[DECAPEN] = 0
- FTMx_SC[CPWMS] = 0

FTMx_CnSC[ELSB:ELSA] controls when channel *n* outputs high:

- = 10, when channel *n* matches
- = 1X, when channel *n*+1 matches

Figure 8 shows the channel *n* output in combined mode:

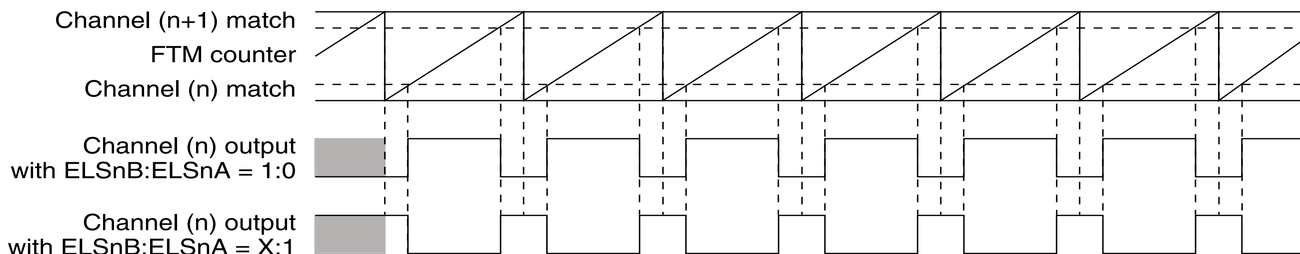


Figure 8. Program flash example

The PWM period is FTMx_MODH:L – FTMx_CNTINH:L + 1, and the PWM pulse width is FTMx_C(n+1)VH:L – FTMx_C(n)VH:L.

A channel *n* interrupt occurs when FTMx_CNT = FTMx_CnVH:L, and a channel *n*+1 interrupt occurs when FTMx_CNT = FTMx_C(n+1)VH:L.

Timer overflow interrupt occurs when FTMx_CNT changes from FTMx_MODH:L to its initial value (FTMx_CNTINH:L).

The next code snippet shows how to enter combined mode:

```
/* FTM2 runs 1000 times slower than the system clock */
FTM2_MODE_FTMEN = 1; /* Enable the FlexTimer */
FTM2_COMBINE1 = 0x01; /* Combine channels 2 and 3 */
                                   /* Set on channel 2 match, Clear on channel 3 match */
FTM2_MOD =  1000-1;  /* 1000 cycle PWM period */
FTM2_C0SC = 0x28;   /* No Interrupts; High true pulses; Clear on Match */
FTM2_C1SC = 0x28;   /* No Interrupts; High true pulses; Clear on Match */
FTM2_C2SC = 0x28;   /* No Interrupts; High true pulses */
FTM2_C3SC = 0x28;   /* No Interrupts; High true pulses */

FTM2_C0V =  200;  /* match after 200 cycles */
FTM2_C1V = 700;  /* match after 700 cycles */
FTM2_C2V = 200;  /* match after 200 cycles */
FTM2_C3V = 700;  /* match after 700 cycles */
FTM2_SC = 0x08;     /* No Interrupts; Output Compare running from SYSTEM CLK */
```

9.3.2 PWM complementary mode

PWM complementary mode is a special case of combined mode when FTMx_COMBINEn[COMP]=1. In PWM complementary mode, the channel (*n*+1) output is the inverse of channel (*n*) output.

The following conditions must be met to enter PWM complementary mode:

- FTMx_MODE[FTMEN] = 1
- FTMx_COMBINEn[COMBINE] = 1
- FTMx_COMBINEn[DECAPEN] = 0
- FTMx_COMBINEn[COMP] =1
- FTMx_SC[CPWMS] = 0

If FTMx_COMBINEn[COMP] = 0, the channel (*n*+1) output is the same as channel (*n*) output.

The next code snippet shows how to enter PWM complementary mode:

```
/* FTM2 runs 1000 times slower than the system clock */
FTM2_MODE_FTMEN = 1; /* Enable the FlexTimer */
FTM2_COMBINE1 = 0x03; /* Combine channels 2 and 3 as complementary mode */
                                  /* Set on channel 2 match, Clear on channel 3 match */
FTM2_MOD =  1000-1;  /* 1000 cycle PWM period */
FTM2_C0SC = 0x28;   /* No Interrupts; High true pulses; Clear on Match */
FTM2_C1SC = 0x28;   /* No Interrupts; High true pulses; Clear on Match */
FTM2_C2SC = 0x28;   /* No Interrupts; High true pulses */
FTM2_C3SC = 0x28;   /* No Interrupts; High true pulses */

FTM2_C0V =  200;  /* match after 200 cycles */
FTM2_C1V = 700;  /* match after 700 cycles */
FTM2_C2V = 200;  /* match after 200 cycles */
FTM2_C3V = 700;  /* match after 700 cycles */
FTM2_SC = 0x08;     /* No Interrupts; Output Compare running from SYSTEM CLK */
```

9.3.3 Deadtime insertion

Deadtime insertion ensures that no two complementary channels drive the active state at the same time. This avoids a short condition between the top transistor and the bottom transistor. It is enabled when these conditions are met:

- FTMx_COMBINEn[DTEN] = 1
- FTMx_DEADTIME[DTVAL] is not 0
- FTM is in PWM complementary mode:
  - FTMx_MODE[FTMEN] = 1
  - FTMx_COMBINEn[COMBINE] = 1
  - FTMx_COMBINEn[DECAPEN] = 0
  - FTMx_COMBINEn[COMP] =1
  - FTMx_SC[CPWMS] = 0

The actual deadtime delay is defined by the FTMx_DEADTIME register. This register can be used for all FTM channels in the same FTM module with the following equation:

$$\text{Dead time delay} = \text{deadtime prescaler} \times \text{deadtime value} \qquad \text{Eqn. 1}$$

The deadtime prescaler is the prescaler of the system clock and defined by FTMx_DEADTIME[DTPS] as shown in Table 17.

Table 17. Deadtime prescaler

DTPS	System Clock Prescaler
0X	1
10	4
11	16

The deadtime value is a number from 1 to 63, defined by FTMx_DEADTIME[DTVAL[5:0]].

The deadtime can be inserted on both complementary channel signals at:

- Rising edge if FTMx_POL[POL_n] and FTMx_POL[POL_{n+1}] = 0
- Falling edge if FTMx_POL[POL_n] and FTMx_POL[POL_{n+1}] = 1

Figure 9 shows the deadtime being inserted at the rising edge of complementary signals:

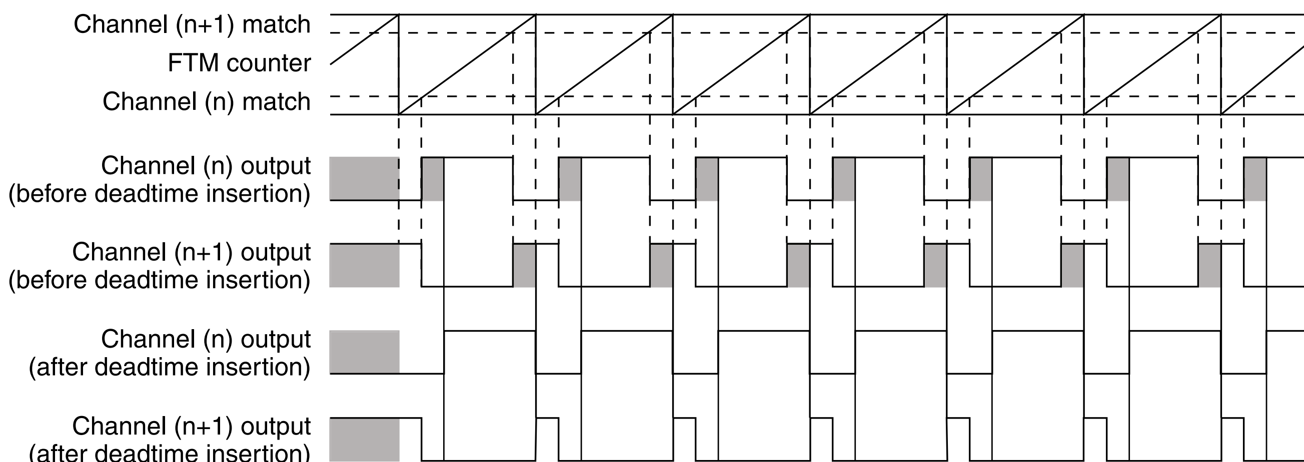


Figure 9. FlexTimer deadtime insertion waveform

This code snippet shows how to set up deadtime for FTM2:

```
/* FTM2 runs 1000 times slower than the system clock */
FTM2_MODE_FTMEN = 1; /* Enable the FlexTimer */
FTM2_COMBINE0 = 0x03; /* Combine channels 0 and 1 as complementary mode */
                                    /* Set on channel 0 match, Clear on channel 1 match */
FTM2_COMBINE1 = 0x13; /* deadtime enabled on complementary channel 2 and 3  */
FTM2_DEADTIME = 0x94; /* Prescaler=4, DTVAL=20, Deadtime=80 System Clocks */
FTM2_MOD =  1000;  /* 1000 cycle PWM period */
FTM2_C0SC = 0x28;   /* No Interrupts; High true pulses; Clear on Match */
FTM2_C1SC = 0x28;   /* No Interrupts; High true pulses; Clear on Match */
FTM2_C2SC = 0x28;   /* No Interrupts; High true pulses */
FTM2_C3SC = 0x28;   /* No Interrupts; High true pulses */

FTM2_C0V =  200;  /* match after 200 cycles */
FTM2_C1V = 700;  /* match after 700 cycles */
FTM2_C2V = 200;  /* match after 200 cycles */
FTM2_C3V = 700;  /* match after 700 cycles */
FTM2_SC = 0x08;     /* No Interrupts; Output Compare running from SYSTEM CLK */
```

9.3.4 PWM synchronization

PWM synchronization provides an opportunity to update registers FTMx_MODH:L and FTMx_CnVH:L with the contents of their write buffers.

It is also used to force the FTM counter to its initial value and force an output mask on specified channels by updating the FTMx_OUTMASK register. It can also be used to synchronize two or more FlexTimer modules.

PWM synchronization is enabled if these conditions are met:

- FTMx_MODE[FTMEN] = 1

- FTMx_COMBINEn[COMBINE] = 1
- FTMx_COMBINEn[DECAPEN] = 0
- FTMx_COMBINEn[COMP] =1
- FTMx_SC[CPWMS] = 0

There are two kinds of PWM synchronization, as listed below. They are configured by the FTMx_MODE[PWMSYNC] bit and the FTMx_SYNC register:

- Hardware trigger synchronization
- Software trigger synchronization

The FTMx_COMBINEn[SYNCEN] bit enables or disables the PWM synchronization on the PWM channel pair (FTMx_C_nV and FTMx_C_{n+1}V). This bit must be set to enable FTM CnV register synchronization.

The FTMx_MODE[PWMSYNC] bit selects which triggers can be used by FTMx_MOD, FTMx_CnV, FTMx_OUTMASK, and FTM counter synchronization:

- 0 (default value): both software and hardware triggers can be used to synchronize all register updates (FTMx_MOD, FTMx_CnV, FTMx_OUTMASK, and FTM counter synchronization)
- 1: both FTMx_MOD and FTMx_CnV register updates are synchronized only by software trigger; FTMx_OUTMASK and FTM counter updates are synchronized only by hardware triggers.

Please note that the software trigger and hardware triggers have a potential conflict if used together. It is recommended to use either hardware or software triggers but not both at the same time — otherwise unpredictable behavior may occur.

9.3.4.1 Hardware trigger synchronization

There are three hardware triggers (trigger0, trigger1, and trigger2) on FTM2 which are synchronized by system clocks. They can be enabled or disabled individually by FTMx_SYNC register, as shown below:

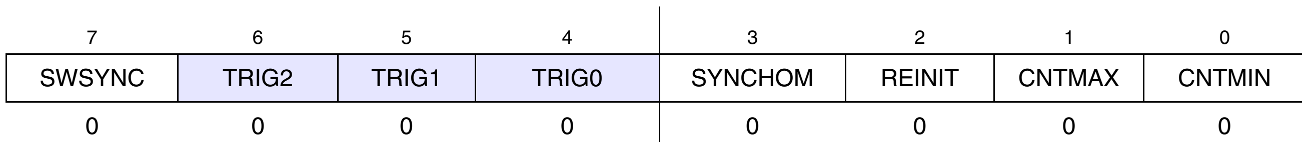


Figure 10. Flextimer FTMx_SYNC register

The hardware trigger event occurs when a rising edge on the selected triggers occurs.

FTM2 hardware triggers are interconnected to other modules as defined in Table 18.

Table 18. FTM2 hardware trigger source

Hardware Trigger	Trigger Source
TRIG2	Software trigger controlled by SYS_SOPT2[FTMSYNC]
TRIG1	FTM0 channel 0 output FTM0CH0
TRIG0	ACMP output

9.3.4.2 Software trigger synchronization

A software trigger event occurs when one is written to the FTMx_SYNC[SWSYNC] bit.

9.3.4.3 PWM synchronization boundary cycle

To synchronize FTMx_MODH:L and FTMx_CnVH:L register updates, FTMx_SYNC[CNTMAX] and FTMx_SYNC[CNTMIN] select one of two boundary cycle synchronization methods: maximum boundary cycle or minimum boundary cycle.

When FTMx_SYNC[CNTMAX] = 1, the maximum boundary cycle is enabled. In this case, FTMx_MODH:L and CnVH:L registers are updated when the FTM counter reaches its maximum value of (FTMx_MODH:L value) after the enabled trigger has occurred.

When FTMx_SYNC[CNTMIN] = 1, the minimum boundary cycle is enabled. In this case, FTMx_MODH:L and CnVH:L registers are updated when the FTM counter reaches its minimum value (FTMx_CNTINH:L value) after the enabled trigger has occurred.

Please also note that PWM synchronization with (FTMx_SYNC[CNTMAX] = 1) is not recommended, and its results are not guaranteed on the S08PT family.

9.3.4.4 PWM MOD register synchronization

MOD register synchronization occurs when the FTMx_MODH:L registers are updated with the value of their write buffers.

Table 19 shows the different possible configurations for MOD register synchronization:

Table 19. MOD register synchronization configurations

Register	Synchronization control bits						Synchronization/update description
	PWMSYNC	REINIT	SYNCHOM	CNTMAX	CNTMIN	SYNCEN	
FTMx_ MODH:L	0	0	X	1	0	X	MODH:L are updated with their write buffer contents when the counter reaches its maximum value after the enabled (hardware or software) trigger has occurred.
	0	0	X	0	1	X	MODH:L are updated with their write buffer contents when the counter reaches its minimum value after the enabled (hardware or software) trigger has occurred.
	0	1	X	X	X	X	MODH:L are updated with their write buffer contents when the enabled (hardware or software) trigger occurs.
	1	X	X	1	0	X	MODH:L are updated with their write buffer contents when the counter reaches its maximum value after the enabled software trigger has occurred.
	1	X	X	0	1	X	MODH:L are updated with their write buffer contents when the counter reaches its minimum value after the enabled software trigger has occurred.

9.3.4.5 PWM CnV register synchronization

The FTM CnV register synchronization or PWM duty cycle synchronization occurs when the FTMx_CnVH:L registers are updated with the value of their write buffers.

Table 20 shows the different possible configurations for CnV register synchronization.

Table 20. CnV register synchronization configuration

Register	Synchronization Control bits						Synchronization/update description
	PWMSYNC	REINIT	SYNCHOM	CNTMAX	CNTMIN	SYNCEN	
FTMx_ CnVH:L	0	0	X	1	0	1	CnVH:L are updated with their write buffer contents when the counter reaches its maximum value after the enabled (hardware or software) trigger has occurred.
	0	0	X	0	1	1	CnVH:L are updated with their write buffer contents when the counter reaches its minimum value after the enabled (hardware or software) trigger has occurred.
	0	1	X	X	X	1	CnVH:L are updated with their write buffer contents when the enabled (hardware or software) trigger occurs.
	1	X	X	1	0	1	CnVH:L are updated with their write buffer contents when the counter reaches its maximum value after the enabled software trigger has occurred.
	1	X	X	0	1	1	CnVH:L are updated with their write buffer contents when the counter reaches its minimum value after the enabled software trigger has occurred.

9.3.4.6 FTM counter synchronization

FTM counter synchronization occurs when the FTM counter (FTMx_CNTH:L) is updated with the value of counter initial value registers (FTMx_CNTINH:L).

Table 21 shows different configurations for PWM counter register synchronization.

Table 21. PWM counter register synchronization configuration

Register	Synchronization control bits						Synchronization/update description
	PWMSYNC	REINIT	SYNCHOM	CNTMAX	CNTMIN	SYNCEN	
FTMx_CNTH:L	0	1	X	X	X	X	CNTH:L are forced to the FTM counter initial value when the enabled (hardware or software) trigger occurs.
	1	1	X	X	X	X	CNTH:L are forced to the FTM counter initial value when the enabled hardware trigger occurs.

9.3.4.7 Output masking and synchronization

Output masking determines which channels will be masked — that is, forced into an inactive state — and is controlled by FTMx_OUTMASK, FTMx_MODE[PWMSYNC], and FTMx_SYNC[SYNCHOM], as shown in Table 22.

Table 22. Output masking synchronization

Register	Synchronization control bits						Synchronization/update description
	PWMSYNC	REINIT	SYNCHOM	CNTMAX	CNTMIN	SYNCEN	
FTMx_ OUTMASK	X	X	0	X	X	X	OUTMASK register are updated on the next rising edge of the system clock.
	0	X	1	X	X	X	OUTMASK register is updated on the next rising edge of the system clock when the enabled (hardware or software) trigger occurs.
	1	X	1	X	X	X	OUTMASK register is updated when the enabled hardware trigger occurs.

9.3.5 Initialization

Initialization preconfigures the channel outputs to their initial states, determined by the FTMx_OUTINIT[CHnOI] bit value when a one is written to the FTMx_MODE[INIT] bit.

To perform initialization on channel outputs, the following bits must be set:

- FTMx_MODE[FTMEN] = 1
- FTMx_COMBINEn[COMBINE] = 1
- FTMx_SC[CLKS] = 00 (FTM counter disabled)

The next code snippet shows how to perform initialization on channel outputs:

```
/* FTM2 runs 1000 times slower than the system clock */
```

```
FTM2_MODE_FTMEN = 1;
FTM2_COMBINE1 = 0x01; /* Combine channels 2 and 3 */
FTM2_OUTINIT = 0x08; /* channel 2 initial value = 0, channel 3 initial value = 1 */

FTM2_MOD =  1000;  /* 1000 cycles PWM period */
FTM2_C1SC = 0x68;   /* Interrupt enabled; High true pulses; Clear on Match */
FTM2_C2SC = 0x28;   /* No Interrupts; High true pulses */
FTM2_C3SC = 0x28;   /* No Interrupts; High true pulses */

FTM2_C1V = 500;  /* match after 500 cycles */
FTM2_C2V = 200;  /* match after 200 cycles */
FTM2_C3V = 700;  /* match after 700 cycles */
FTM2_SC = 0x08;     /* No Interrupts; Output Compare running from SYSTEM CLK / 1 */

interrupt VectorNumber_Vftm2ch1 void FTM2_Ch1_ISR(void)
{
    FTM2_SC = 0x00;           /* Disable FTM counter */
    FTM2_MODE_INIT = 1;  /* Force initialization */
    FTM2_SC = 0x08;           /* Enable FTM counter */
}
```

9.3.6 Fault control inputs

Channel outputs are forced to a safe value when a fault is detected. Fault control is enabled when:

- FTMx_MODE[FTMEN] = 1
- FTMx_COMBINEn[COMBINE] = 1
- FTMx_COMBINEn[DECAPEN] = 0
- FTMx_SC[CPWMS] = 0
- FTMx_COMBINEn[FAULTEN] = 1
- FTMx_MODE[FAULTM] 00

FTMx_MODE[FAULTM] register bits are defined as below:

7	6	5	4	3	2	1	0
FAULTIE	FAULTM		CAPTEST	PWMSYNC	WPDIS	INIT	FTMEN

Table 23. FAULTM bit settings

FAULTM	Fault Control Mode
00	Disabled for all channels
01	Enabled for even channels only with Manual fault clearing
10	Enabled for all channels with Manual fault clearing
11	Enabled for all channels with Automatic fault clearing

FTM2 supports up to four fault inputs. Each fault input can be enabled or disabled by the FAULTnEN bit in the FTMx_FLTCTRL register, shown below:

7	6	5	4	3	2	1	0
FFLTR3EN	FFLTR2EN	FFLTR1EN	FFLTR0EN	FAULT3EN	FAULT2EN	FAULT1EN	FAULT0EN

The fault flags are shown in the FTMx_FMS register:

7	6	5	4	3	2	1	0
FAULTF	WPEN	FAULTIN	0	FAULTF3	FAULTF2	FAULTF1	FAULTF0
0				0	0	0	0

FTMx_FMS[FAULTFn] indicates a fault condition detected on the FAULTn pin, when both the fault control and the corresponding fault pin fault control are enabled. It is cleared by reading FTMx_FMS, followed by writing zero to the flag bit. It can also be cleared when the FAULTF flag is cleared.

FTMx_FMS[FAULTF] = FAULTF0 | FAULTF1 | FAULTF2 | FAULTF3. This flag is cleared by reading FTMx_FMS register followed by writing zero to the FAULTF bit.

9.3.6.1 Automatic fault clearing

Disabled channel outputs are enabled when the fault input signal (FAULTIN) returns to zero and a new PWM cycle begins.

This code snippet shows how to use automatic fault clearing:

```
/* FTM2 runs 1000 times slower than the system clock */
FTM2_MODE = 0x63; /* FAULTM = 11 Automatic Clearing */
FTM2_COMBINE1 = 0x41; /* Enable Fault Control, Combine channels 2 and 3 */
FTM2_POL = 0x00;

FTM2_MOD = 1000;  /* 1000 cycle PWM period */
FTM2_C2SC = 0x28;   /* No Interrupts; High true pulses */
FTM2_C3SC = 0x28;   /* No Interrupts; High true pulses */

FTM2_C2V = 200;  /* match after 200 cycles */
FTM2_C3V = 700;  /* match after 700 cycles */
FTM2_SC = 0x08;     /* No Interrupts; Output Compare running from SYSTEM CLK */
```

9.3.6.2 Manual fault clearing

Disabled channel outputs are enabled when the fault flag FAULTF is cleared and a new PWM cycle begins, regardless of the fault input signal (FAULTIN). It is recommended to verify the fault input signal (FAULTIN bit) before clearing the fault flag FAULTF, to avoid unpredictable results.

This code snippet shows how to use manual fault clearing:

```
/* FTM2 runs 1000 times slower than the system clock */
FTM2_MODE = 0x23; /* FAULTM = 01, Even Channels, Manual Clearing */
FTM2_COMBINE1 = 0x41; /* Enable Fault Control, Combine channels 2 and 3 */
FTM2_POL = 0x00;

FTM2_MOD = 1000;  /* 1000 cycle PWM period */
```

```
FTM2_C2SC = 0x28;   /* No Interrupts; High true pulses */
FTM2_C3SC = 0x28;   /* No Interrupts; High true pulses */

FTM2_C2V = 200;  /* match after 200 cycles */
FTM2_C3V = 700;  /* match after 700 cycles */
FTM2_SC = 0x08;     /* No Interrupts; Output Compare running from SYSTEM CLK */

/* FAULT interrupt serivce routine */
FTM2_FAULT_ISR:
       while (FTM2FMS_FAULTIN); /* Wait for FAULTIN to clear */
        FTM2FMS;                              /* Read FTM1FMS with FAULTF set*/
        FTM2FMS_FAULTF = 0;        /* write 0 to FAULTF */
```

9.3.6.3 Fault filtering

Fault filtering is used to protect against glitches or noise on the fault input pins. Any noise with pulse width < FTMx_FLTFILTER[FFVAL[3:0]] system clocks will be ignored.

If FTMx_FLTCTRL[FFLTRnEN] =1 and FTMx_FLTFILTER[FFVAL[3:0]] != 0, then the fault filtering logic on the fault input pin *n* is enabled.

If FTMx_FLTCTRL[FFLTRnEN] = 0 or FTMx_FLTFILTER[FFVAL[3:0]] == 0, then the fault filtering logic on the fault input pin *n* is disabled.

9.3.7 Polarity control

Polarity control determines whether the signal is negated at the output and defines the channel as safe or in an inactive state.

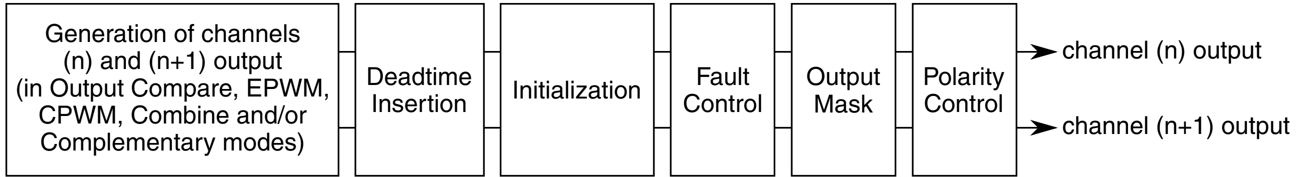
If FTMx_POL[POLn] = 0, the channel *n* polarity is active high, and one is the active state while zero is the inactive state; otherwise, the channel *n* polarity is active low and zero is the active state while one is the inactive state.

Polarity control is only available in combined mode, that is:

- FTMx_MODE[FTMEN] = 1
- FTMx_COMBINEn[COMBINE] = 1
- FTMx_COMBINEn[DECAPEN] = 0
- FTMx_SC[CPWMS] = 0

9.3.8 Features priority

Figure 11 shows the priority of FTM features that can be combined to generate correct signals on channel *n* and *n*+1 output pins:



Priority runs from left (highest) to right (lowest)

Figure 11. FlexTimer features priority

Figure 12 shows the correct waveform on the channel output pin after the priority features:

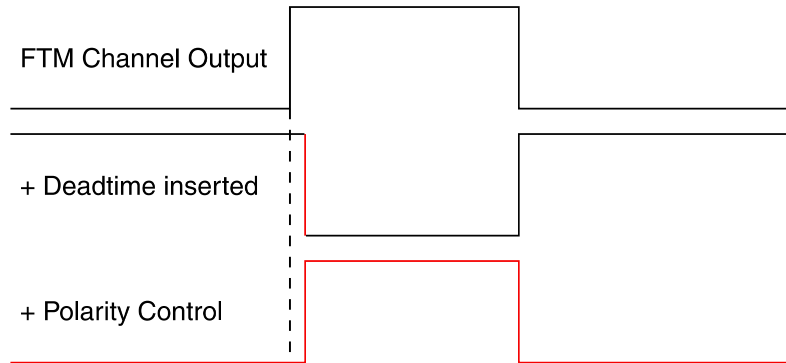


Figure 12. FlexTimer features priority example

If the feature priority is reserved, then the resulting waveform on the channel output pin is different.

9.3.9 Enhanced triggering

The external trigger provides a hardware trigger signal for the on-chip modules. It is only available in combined mode.

There are two kind of external triggers: channel trigger and initialization trigger.

The channel *n* trigger is enabled when these conditions are met:

- FTMx_MODE[FTMEN] = 1
- FTMx_COMBINEn[COMBINE] = 1
- FTMx_COMBINEn[DECAPEN] = 0
- FTMx_SC[CPWMS] = 0
- FTMx_EXTTRIG[CHnTRIG] = 1

And the channel *n* trigger output is generated if the FTM counter = FTMx_CnV register (channel *n* match).
The initialization trigger is generated when the FTM counter is updated with the CNTINH:L register value. It is enabled when these conditions are met:

- FTMx_MODE[FTMEN] = 1
- FTMx_COMBINEn[COMBINE] = 1

- FTMx_COMBINEn[DECAPEN] = 0
- FTMx_SC[CPWMS] = 0
- FTMx_EXTTRIG[INITTRIGEN] = 1

Figure 13 shows the interconnections between FTM2 external triggers and ADC hardware triggers:

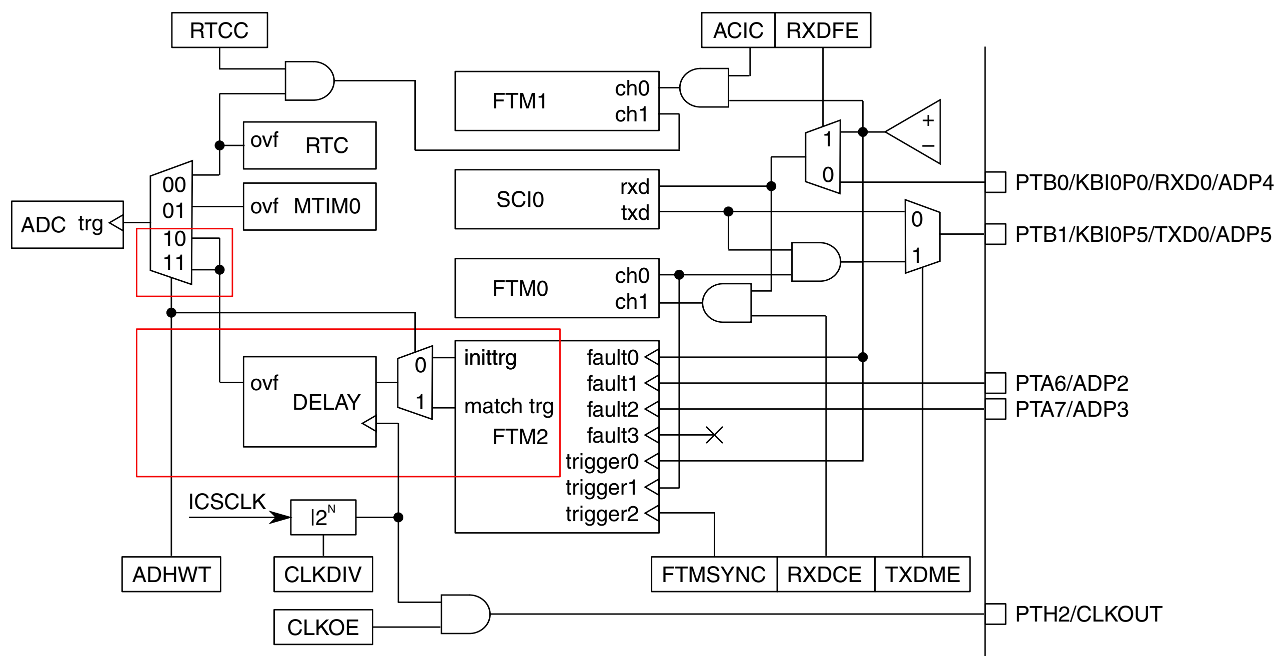


Figure 13. FTM2 to ADC hardware trigger

10 TSI addition

Devices in the S08PT family have one touch sense interface module, which allow users to detect touch in capacitive electrodes in a range from 1 to 100 pF. Features include:

- Support of up to 16 external electrodes
- Automatic detection of electrode capacitance across all operational power modes
- Internal reference oscillator for high-accuracy measurement
- Configurable software or hardware scan trigger
- Full support by Freescale touch sensing software library (TSS) with keypads, sliders, and rotaries
- Capability to wake MCU from stop 3 mode
- Compensates for temperature and supply voltage variations
- High sensitivity change with 16-bit resolution counter register
- Configurable up to 4096 scan times

10.1 TSI function description

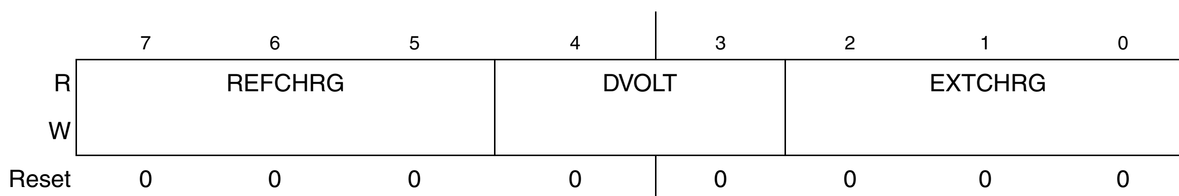
The TSI supports up to 16 external electrodes. Users can enable the external electrodes with TSI_PEN0 and TSI_PEN1 registers. For example:

```
TSI_PEN0 = 0x0f; // enable TS0 ~ TS3 external electrodes
```

There are two oscillators in the TSI module: internal and external. With the internal oscillator, there is a reference capacitance and users can configure the discharge and charge current from 500 nA to 64 mA. The frequency is directly proportional to the current, as illustrated in Equation 2.

$$Fref = Iref/2Cref*V \qquad \textit{Eqn. 2}$$

These parameters can be configured by TSI_SC2 register as below:



Configure the discharge and charge threshold voltage by DVOLT bits (for V) below:

4–3 DVOLT	DVOLT These bits indicate the oscillator's voltage rails as below: 00 $V_P = 0.80\ V_{ref}$; $V_m\ 0.20\ V_{ref}$; $DV = 0.6\ \Delta V_{ref}$ 01 $V_P = 0.70\ V_{ref}$; $V_m\ 0.30\ V_{ref}$; $DV = 0.4\ \Delta V_{ref}$ 10 $V_P = 0.62\ V_{ref}$; $V_m\ 0.28\ V_{ref}$; $DV = 0.24\ \Delta V_{ref}$ 11 $V_P = 0.58\ V_{ref}$; $V_m\ 0.42\ V_{ref}$; $DV = 0.16\ \Delta V_{ref}$

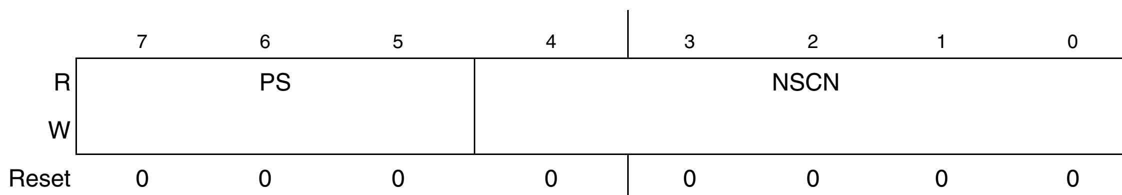
Here is the code snippet to configure parameters for the internal reference:

```
TSI_CS2_REFCHRG  = 0x07;       // setting reference oscillator current is 64uA
TS1_CS2_DVOLT = 0x01;           // setting Vp = 0.7Vref, Vm = 0.3Vref
```

In the same way, the external oscillator can be configured as below:

```
TSI_CS2_EXTCHRG  = 0x04;       // setting reference oscillator current is 8uA
```

The difference between the internal oscillator and the external oscillator is that there is a reference capacitance with the internal oscillator — but for the external oscillator, capacitance is made up of the external electrode. Through detecting the duration of both the charging and discharging of the electrodes, the TSI module can measure the external electrode's capacitance. When a finger approaches the electrode, the capacitance will be changed. In order to improve accuracy when measuring capacitance, users can set the prescaler of the external oscillator and scan times in TSI Control and Status Register 1 (TSI_CS1):



The maximum value for PS is 128 and for NSCN is 32, giving a maximum number of scans of 4096.

The NSCN and PS bit fields will be used to determine the charge and discharge time of the electrode. The relationship between NSCN, PS, and charge/discharge time will be further described later.

Configuring the register is shown as below:

```
TSI_CS1_PS = 2;  //configure prescaler is 4
TSI_CS1_NSCA = 0x0f; // scan number is 16.
```

To start a scan, users can enable the software trigger or hardware trigger. In software trigger mode, when the TSI_CS0[STM] bit is cleared after setting the TSI_CS0[SWTS] bit, the TSI module will start scanning the channel which is specified by TSI_CS3[TSICH]. The following code shows how to set the current scan channel and start scanning:

```
TSI_CS3_TSICH = 0x01;  // select channel 1
TSI_CS0_STM = 0;      // enable software trigger
TSI_CS0_SWTS = 1;   //  start a scanning
```

In hardware trigger mode when the TSI_CS0[STM] bit is set, the trigger event is selectable from RTC, MTIM0, and FTM2. This feature allows scans to be automatically scheduled instead of controlled by software, making the measurement process less dependent on CPU intervention.

As the charge and discharge times are captured by a 16-bit counter, Tcharg and Tdischarge can be represented by the reference clock Iref and the counter value Ncharge and Ndischarge, which sum Ntotal:

$$\text{Ntotal} = \text{Ps} \times \text{Nscan} \times \text{Iref}/\text{Iext} \qquad \textit{Eqn. 3}$$

Ps: setting the prescaler of the external oscillator

Nscan: setting scan numbers for electrode

Iref: reference oscillator current value

Iext: external electrode oscillator current

If Ntotal exceeds a certain threshold this indicates that there is a touch in one of the electrodes. This threshold may be dynamically updated to adapt to changes in temperature, humidity, and materials over a long period of time. This dynamic adaptation of threshold over time is called baseline tracking. The Freescale Touch Sensing Software Library (TSS) provides low-level support of the TSI measurement functions plus baseline tracking, filtering, and conversion of measurement data into accurate touch and release events. More info may be found at www.freescale.com/touchsensing.

10.2 TSI applications

The touch sensor is flexible and convenient for use. It is used in a wide range of applications, from cellular phones and PDAs to personal computers, point-of-sale devices, many medical appliances, and industrial applications.

On the associated hardware, users can lay out different patterns of electrodes on the printed circuit board (PCB). These electrodes add very little cost compared to their mechanical counterparts. Depending on requirements, simple shapes such as circles, rectangles, or ovals, with a size similar to the object to be detected, may be used. Based on different software, single key and multiplexed keys can be detected. When the number of keys exceeds the available sensor channels, multiplexing or logical combinations can be created. Consequently, detection of a valid key does not necessarily mean the change in capacitance of a single electrode, but of two or more electrodes at the same time.

Below are some typical applications for touch sensors:

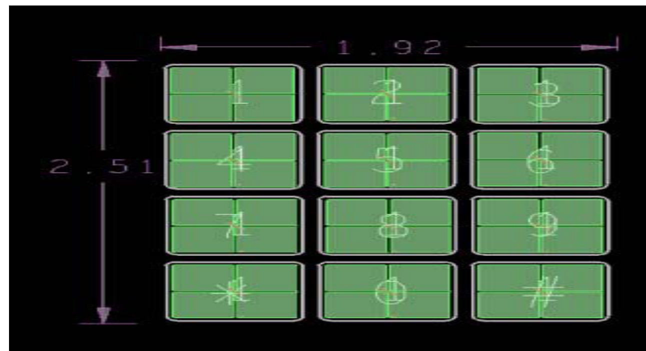


Figure 14. TSI keyboard patterns

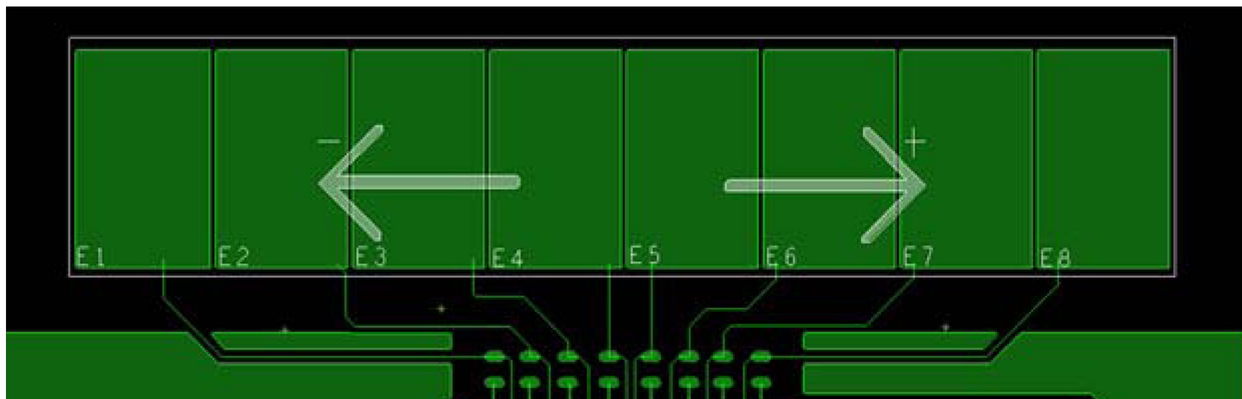


Figure 15. TSI slides patterns

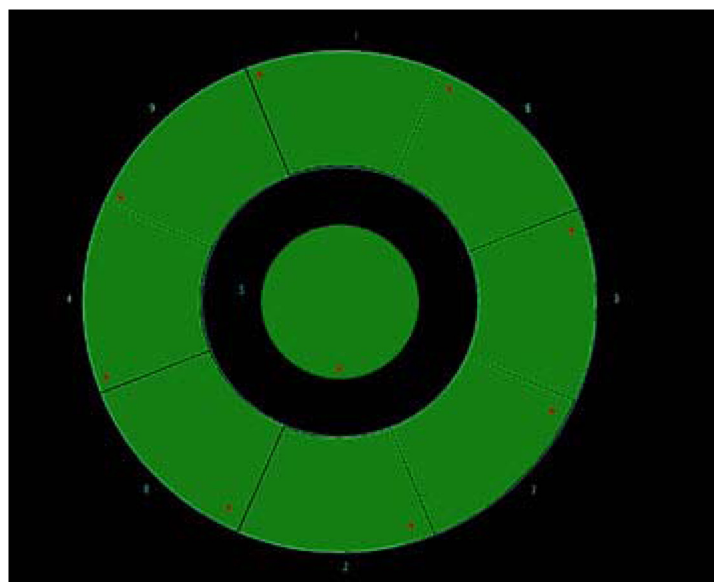


Figure 16. TSI rotary patterns

The next code shows how to initialize the TSI:

```
void TSI_Init( void ){
     TSI_CS1_PS = 2;  //configure prescaler is 4
```

```
    TSI_CS1_NSCA = 0x0f; // scan number is 16.
    TSI_CS2_EXTCHRG  = 0x04;       // setting external oscillator current is 8uA
    TSI_CS2_REFCHRG  = 0x04;       // setting reference oscillator current is 8uA
    TS1_CS2_DVOLT = 0x01;           // setting Vp = 0.7Vref, Vm = 0.3Vref
// enable TSI electrode 0 ~ 4
TSI_PEN0 = 0x0f;
// enable TSI module in stop mode for low power
TSI_CS0_STPE = 1;
// enable interrupt,when scanning complete,generate a interrupt
TSI_CS0_TSIIEN = 1;
// enable TSI module
TSI_CS0_TSIEN = 1;
}
```

The code that starts scanning and obtains the counter value is shown below:

```
     // specify the scanning channel
TSI_CS3_TSICH = 0x01;//select channel 1 start scanning
// start scanning
TSI_CS0_SWTS = 1;    // start a scanning
// wait scan complete
while(!TSI_CS0_EOSF);// wait scan complete
TSI_CS0_EOSF = 1;    // clear EOSF flag
uiScanValue = TSI_CNT;  //read counter value
```

Users need to scan the electrodes periodically and, based on the value returned, determine if the electrode is released or pressed. In slide or rotary configurations, users can decide the direction, position, and increment, according to electrode capacitance changes and additional algorithms. Freescale provides the TSS software library (with TSS support) which can be integrated into applications. This software library includes the calibration, baseline tracking, IIR, and other filtering algorithms, as well as decoding the electrode state, to keep track of environment changes or other factors that could affect measurement result.

In addition, the TSI can work in stop3 mode, and wake the CPU up when a scan is complete. It is very useful for low-power applications in which waking up via touch is preferred over mechanical buttons or other stimuli.

11 RTC and MTIM additions

There are two additional timers: the Real Time Counter (RTC) and Modulo Timer (MTIM) on the S08PT family are described in the following sections.

11.1 Real time counter

The real-time counter (RTC) can be used for time-of-day, calendar, or any task scheduling functions. It can also be used for a programmable delay before asserting a hardware trigger to other modules. In addition, it can serve as a cyclic wakeup from low power modes.

The 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.

The RTC overflow trigger can be used as a hardware trigger for the ADC and TSI modules.

If SYS_SOPT2[RTCC] bit is set, the RTC overflow is connected to FTM1 channel 1 for capture.

Figure 17 shows the interconnections between RTC, ADC, and FTM1:

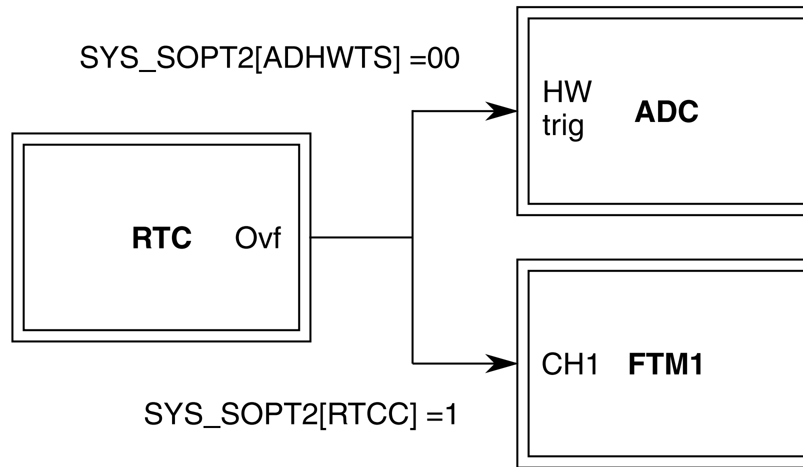


Figure 17. Interconnection from RTC to ADC and FTM1

Before accessing any RTC register, SCG_C2[RTC] must be set to enable clock to the RTC.

There are three clock sources selectable by RTC_SC2[RTCLKS] for the RTC:

- External low-power oscillator (XOSC)
- On-chip low power oscillator (LPO) (1KHz)
- Bus clock

The RTC prescaler value is determined by the RTCLKS[1:0] and RTCPS[2:0] bits in the RTC_SC2 register as shown in Table 24.

Table 24. RTC prescaler

RTCLKS[1:0]	RTCPS[2:0]	Prescaler	Description
XX	000	off	Counter is off

Table 24. RTC prescaler

RTCLKS[1:0]	RTCPS[2:0]	Prescaler	Description
X0	001	1	Binary based prescaler
	010	2	
	011	4	
	100	8	
	101	16	
	110	32	
	111	64	
X1	001	128	
	010	256	
	011	512	
	100	1024	
	101	2048	
	110	100	Decimal based prescaler
	111	1000	

Please note that if a different value is written to RTC_SC2[RTCLKS] or RTC_SC2[RTCPS], the prescaler and RTC counters (RTC_CNTH and RTC_CNTL) are reset to zero.

When the RTC counter reaches the RTC modulo value in RTC_MODH:L registers (in other words, RTC modulo match or RTC counter overflow), the RTC_SC1[RTIF] flag gets set. In this case, if RTC_SC1[RTIE] = 1 (RTC interrupt is enabled), then an RTC interrupt will be generated. RTC_SC1[RTIF] is cleared by writing a one to RTC_SC1[RTIF].

If RTC_SC1[RTCO] is set, the RTC will toggle the output on the RTCO pin whenever the RTC counter overflows (in other words, RTC modulo match).

11.2 Modulo timer

The modulo timer (MTIM) can be used to generate periodic interrupts and/or a programmable delay before asserting a hardware trigger to other modules.

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.

There are two MTIMs on the S08PT family: MTIM0 and MTIM1.

MTIM0 timer overflow can be used to trigger the ADC, as shown in Figure 18.

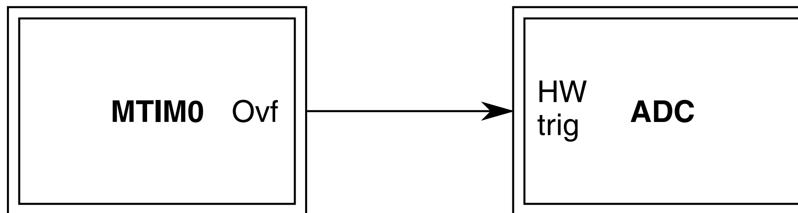


Figure 18. Interconnection between MTIM0 and ADC

There are four clock sources for the MTIM selected by MTIMx_CLK[CLKS]:

- Internal bus clock (ICSCLK)
- Fixed frequency clock (ICSFFCLK)
- External clock on the TCLK pin with counter incrementing on TCLK rising edges
- External clock on the TCLK pin with counter incrementing on TCLK falling edges

One of the nine prescalers is selected by MTIMx_CLK[PS] from the clock source divided by 1, by 2, up to by 256.

The MTIM has three modes of operation:

- Stopped
- Free-running
- Modulo

If MTIMx_SC[TSTP] = 1, the counter is in stopped mode.

To start the MTIM in free-running mode, follow steps below:

1. Stop the MTIM counter.
2. Write MTIMx_CLK to select one of the four clock sources and the related prescaler, if MTIMx clock has not been configured or needs to be changed.
3. Write zero to the modulo register (MTIMx_MOD).
4. Write to the MTIM status and control register (MTIMx_SC).
5. Clear the MTIM stop bit (TSTP).

The next code shows how to set up a free-running timer:

```
 /* MTIM0 free-running mode example */
MTIM0_SC_TSTP = 1;                                             /* stop counter */
MTIM0_CLK = 0x17;              /* use FIXED clock ICSFFCLK and prescaler /128 */
MTIM0_MOD = 0;                                       /* must clear MOD register */
MTIM0_SC |=  MTIM0_SC_TOF_MASK; /* write MTIMx_SC to clear TOF first */
MTIM0_SC_TOIE = 1;                               /* then set TOIE bit */
MTIM0_SC_TSTP = 0;                /* clear TSTP to start free-running counter */

interrupt VectorNumber_Vmtim0 void MTIM0_ISR(void)
{
   unsigned char dummy;
```

```
/* clear overflow flag by reading MTIMx_SC and then writing 0 to TOF bit */
dummy = MTIM0_SC;
MTIM0_SC_TOF = 0;
}
```

To start the MTIM in modulo mode, perform these steps:

1. Stop the MTIM counter.
2. Write MTIMx_CLK to select one of the four clock sources and the related prescaler, if the MTIMx clock has not been configured or needs to be changed.
3. Write a value other than zero to the modulo register (MTIMx_MOD).
4. Write to the MTIM status and control register (MTIMx_SC).
5. Clear the MTIM stop bit (TSTP).

This code shows how to set up modulo mode:

```
/* MTIM0 modulo mode example */
MTIM0_SC_TSTP = 1;                                             /* stop counter */
MTIM0_CLK = 0x17;                /* use FIXED clock ICSFFCLK and prescaler /128 */
MTIM0_MOD = 99;             /* generate interrupt at 100 prescaler clock cycles */
MTIM0_SC |=  MTIM0_SC_TOF_MASK; /* write MTIMx_SC to clear TOF first */
MTIM0_SC_TOIE = 1;                                /* then set TOIE bit */
MTIM0_SC_TSTP = 0;                 /* clear TSTP to start free-running counter */

interrupt VectorNumber_Vmtim0 void MTIM0_ISR(void)
{
   unsigned char dummy;
/* clear overflow flag by reading MTIMx_SC and then writing 0 to TOF bit */
dummy = MTIM0_SC;
MTIM0_SC_TOF = 0;
}
```

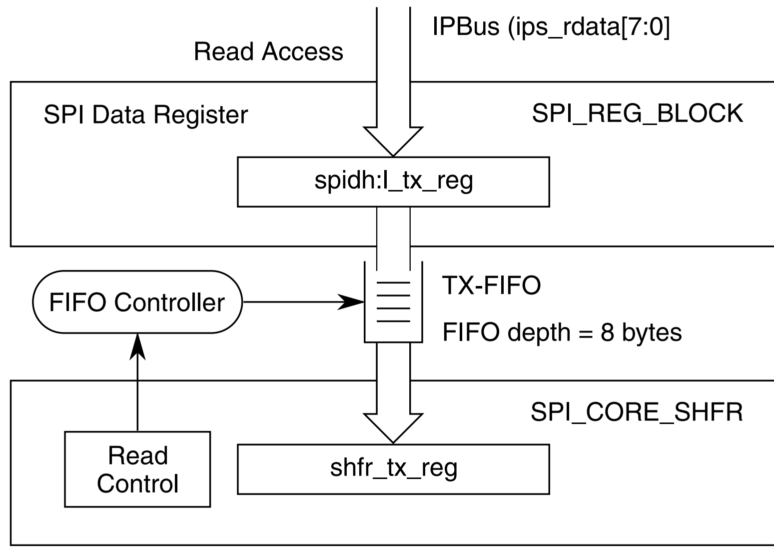
12 16-bit SPI with queue addition

Compared to the S08AC/FL family, the S08PT family adds two functions in its 16-bit SPI module: FIFO and receive data hardware matching. In addition, the S08PT family also includes another 8-bit SPI module which is the same as in the S08AC/FL families.

To use the 16-bit SPI, the bus clock connected to SPI must be enabled first (SCG_C3_SPI1=1).

In FIFO mode (FIFOMODE is 1), the SPI RX buffer and SPI TX buffer are replaced by an 8-byte-deep FIFO, which provides features to allow fewer CPU interrupts to occur when transmitting/receiving high volume/high speed data.

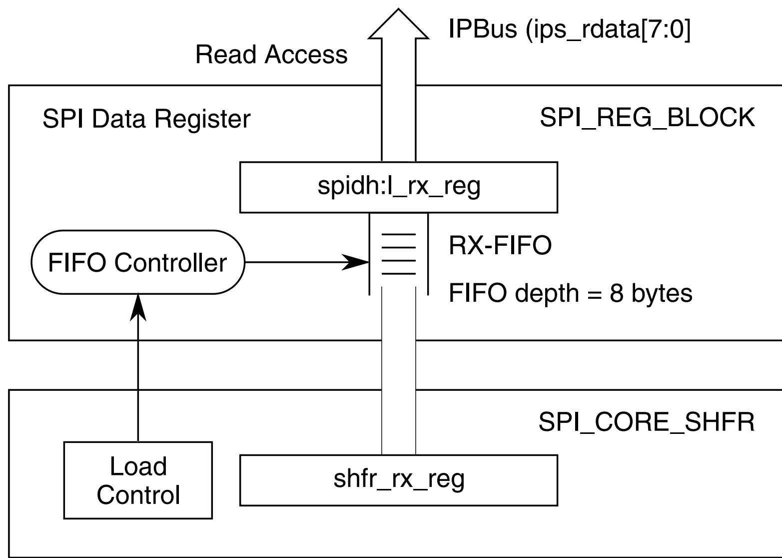
The write-side structural overview is shown in Figure 19.



SPIH:L write side structural overview in FIFO mode

Figure 19. SPI write FIFO structure

The read-side structural overview is shown in Figure 20.



SPIH:L read side structural overview in FIFO mode

Figure 20. SPI read FIFO structure

When FIFO mode is enabled, the SPI can still function in either 8-bit or 16-bit mode (as per SPIMODE bit). For FIFO function, two registers are added (SPIx_C3 and SPIx_CI), and four additional flags help monitor the FIFO status. Two of these flags can provide CPU interrupts.

The status register has four flags that provide mechanisms to support an 8-byte FIFO mode:

- RNFULLF

- TNEARF
- TXFULLF
- RFIFOEF

Also, be aware that the function of SPRF and SPTEF differs slightly from their function in the normal buffered modes, mainly regarding how the flags are cleared by the amount available in the transmit and receive FIFOs.

- The RNFULLF and TNEAREF flags help improve the efficiency of FIFO operation when transferring large amounts of data. These flags provide a “watermark” feature of the FIFOs to allow continuous transmission of data when running at high speed.
- The RNFULLF flag can generate an interrupt if the RNFULLIEN bit in the SPIx_C3 register is set, which allows the CPU to start emptying the receive FIFO without delaying the reception of subsequent bytes. The user can also determine if all data in the receive FIFO has been read by monitoring the RFIFOEF.
- The TNEAREF flag can generate an interrupt if the TNEARIEN bit in the SPIx_C3 register is set, which allows the CPU to start filling the transmit FIFO before it is empty and thus to prevent breaks in SPI transmission.

The SPI also contains a hardware match register. When the value received in the SPI receive data buffer equals this hardware compare value, the SPI match flag (SPMF) is set. An interrupt will be requested if SPMIE is set.

In 8-bit mode, only the SPIx_ML register is available. Reads of the SPIx_MH register return all zeros. Writes to the SPIx_MH register are ignored.

In 16-bit mode, reading either byte (SPIx_MH or ML register) latches the contents of both bytes into a buffer where they remain latched until the other byte is read. Writing to either byte (SPIx_MH or ML register) latches the value into a buffer. When both bytes have been written, they are transferred as a coherent value into the SPI match registers.

To conserve power, the wait mode is available, but it depends upon the state of the SPISWAI bit in SPI Control Register 2.

- If SPISWAI is clear, the SPI operates normally when the CPU is in wait mode.
- If SPISWAI is set, SPI clock generation ceases and the SPI module enters a power conservation state when the CPU is in wait mode.

The SPI is completely disabled in a stop mode, where the peripheral bus clock is stopped and internal logic states are not retained. After an exit from this type of stop mode, all registers are reset to their default values, and the SPI module must be re-initialized.

Below is the example code snippet to initialize the SPI as a master device in 16-bit FIFO mode:

```
SPI1_C1_MSTR       = 1;        /* SPI  master */
SPI1_C2_SPIMODE = 1;     /* 16 bit mode, SPI hardware match interrupt
                                               disable,polling*/
SPI1_BR                = 0x10;  /* baud rate divisor is 2, baurd rate prescaler is 2*/
SPI1_C3_FIFOMODE = 1;        /* FIFO enabled */
SPI1_C1_SPE            = 1;          /* enable SPI */
```

The next code snippet is used to transmit data with polling method in 16-bit FIFO mode:

```
for(xfer_count=0;xfer_count< 10;xfer_count++) /* SPI1 send 10 bytes */
{
        while(SPI1_S_TXFULLF);                  /* wait till Tx FIFO not full */
         SPI1_D16 = tx_data[xfer_count] ;            /* write data to Tx FIFO */
}
```

The next code snippet is used to receive data with polling method in 16-bit FIFO mode:

```
while(!SPI1_S_RFIFOEF) {/* if Rx FIFO is not empty */
        rx_data[rx_count++] = SPI1_D16;  /* read data from  Rx FIFO */
}
```

13 Analog-to-Digital Converter (ADC)

The S08PT family has a similar ADC structure as the S08AC/FL families, except the S08PT family also has added channel FIFO and result FIFO features. These FIFO operation features are used to minimize the interrupts to the CPU, in order to reduce CPU loading in ADC interrupt service routines.

The FIFO function is enabled when the ADC_SC4[AFDEP] bits are set to non-zero values. The FIFO depth is indicated by these bits. The FIFO supports up to an eight-level buffer.

The analog input channel FIFO is accessed by the ADC_SC1[ADCH] bits when the FIFO function is enabled. The analog channel must be written to this FIFO in order. The ADC starts the conversion only when the FIFO is filled to the levels indicated by the ADC_SC4[AFDEP] bits.

A write to ADC_SC1[ADCH] will re-fill channel FIFO to initiate a new conversion if all previous conversions are completed; otherwise, it will abort the current conversion and the rest of conversions not started in the channel FIFO, including the conversion by this write. Furthermore, it resets the channel FIFO.

It is recommended to write to the ADC_SC1 after all of the conversions are completed (ADC_SC1[COCO] = 1) or the ADC is in an idle state.

When the FIFO function is enabled, the result of the FIFO is accessed by the ADC_RH:ADC_RL registers. The result must be read via these two registers in the same order of analog input channel FIFO to get the proper results.

Do not read ADC_RH:ADC_RL until all of the conversions are completed in FIFO mode. Otherwise, the ADC will continuously scan the channel FIFO and make conversions until the result FIFO is full. As a result the readable data number will be more than the FIFO depth.

The ADC_SC1[COCO] bit will be set only when all conversions indicated by the analog input channel FIFO are complete. 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.

If software trigger is selected, 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 are complete, the COCO bit is set and an interrupt request will be submitted to CPU if the AIEN bit is set.

If hardware trigger is selected, the next analog channel is fetched from analog input channel FIFO. However, when this conversion completes its result is stored in the result FIFO, and the next hardware trigger is fed to the ADC module.

In single conversion mode (ADCO bit is clear), if the software trigger is enabled then the ADC will not start conversion until the channel FIFO is full again; if the hardware trigger is enabled, then the ADC will not start conversion until the channel FIFO is full again and a new hardware trigger occurs.

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 FIFO channel. The ADC conversion is started as soon as the first channel is written. The COCO bit is set when the result FIFO is fulfilled. In continuous mode (ADCO = 1), the ADC will start the next conversion with the same channel when COCO is set.

The next code snippet demonstrates how to set up ADC channel FIFO and read result FIFO:

```
/* Initializing ADC for FIFO mode */
ADC_APCTL1 = 0x0f;/* enable analog channel pins */
ADC_APCTL2 = 0x00;
ADC_SC3_ADIV   = ADC_ADIV_DIVIDE_4;  /* configure ADC clock prescaler*/
ADC_SC3_MODE   = ADC_MODE_12BIT;   /*  configure ADC resolution: 12 bits */
ADC_SC3_ADLSMP = 0;                                     /* short sample time */
ADC_SC3_ADICLK = CLOCK_SOURCE_ADACK;  /* select ADC clock source */
ADC_SC4_AFDEP = 0x02;        /*  FIFO depth is 3 levels which enable FIFO mode */
ADC_SC4_ASCANE = 0;/* enable FIFO scan mode if ASCANE = 1; use normal FIFO mode if ASCANE
= 0 */
ADC_SC2 = 0;                         /* select software trigger and no compare */
/* write ADC input channels to channel FIFO with single conversion mode */
for( i=0;i<3;i++ )
{
         switch(i)
         {
         case 0:ADC_SC1 = 0x16;             /* setting channel temperature sensor */
              break;
              case 1:ADC_SC1 = 0x1d;             /* setting channel Vrefh */
              break;
              case 2:ADC_SC1 = 0x1e;             /* setting channel Vrefl */
              break;
              }
}
while(!ADC_SC1_COCO);/* wait until all conversions in FIFO are completed */
/* Now it is safe to read result FIFO in order */
while( !ADC_SC2_FEMPTY )
{
          uiADC_Result[ucCount++] = ADC_R;
}
```

Table 25 compares ADC performance between the different families:

Table 25. ADC performance comparison

Parameters	S08AC	S08FL	S08PT	Unit
CPU clock BUS clock	40 20	20 10	20 20	MHz
Resolution	10 8	8	12 10 8	bits
Conversion time first subsequent	>=3.5 (10-bit), >=3 (8-bit) >=2.5	>=3 >= 2.125	TBD	us
E_{TUE}	<=±2.5 ±0.5	<= ±0.5	TBD	LSB
INL	<= ±1.0 ±0.5	<= ±0.5	TBD	LSB
DNL	<= ±1.0 ±0.5	<= ±0.5	TBD	LSB

14 References

MC9S08PT60RM, *MC9S08PT60 Reference Manual*

MC9S08AC60, *MC9S08AC60/AC48/AC32 Data Sheet*

MC9S08AC128RM, *MC9S08AC128 Referenence Manual*

MC9S08FL16RM, *MC9S08FL16 Reference Manual*

AN1259, “System Design and Layout Techniques for Noise Reduction in MCU-Based Systems”

15 Glossary

BDM Background Debug Mode

CRC Cyclic Redundency Check

ECC Error Correction Codes

FCCOB Flash Common Command OBject

RTC Real Time Counter

TPM Timer/PWM Module

FTM FlexTimer Module

MTIM Modulo Timer

WDOG Watchdog

TSI Touch Sense Input

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4347
Rev. 0
08/2011

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor 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 Freescale Semiconductor 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. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

For information on Freescale’s Environmental Products program, go to http://www.freescale.com/epp.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. 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
© Freescale Semiconductor, Inc. 2011. All rights reserved.

freescale™
semiconductor