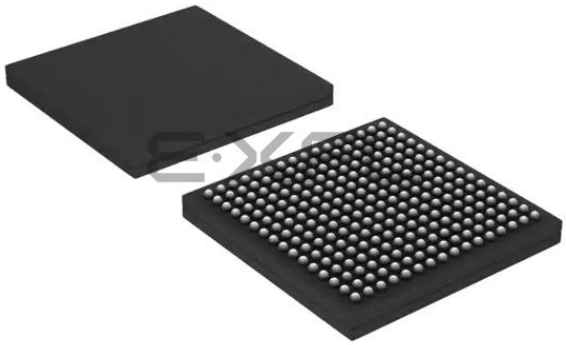


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	Coldfire V2
Core Size	32-Bit Single-Core
Speed	66MHz
Connectivity	CANbus, EBI/EMI, Ethernet, I²C, SPI, UART/USART
Peripherals	DMA, LVD, POR, PWM, WDT
Number of I/O	150
Program Memory Size	-
Program Memory Type	ROMless
EEPROM Size	-
RAM Size	64K x 8
Voltage - Supply (Vcc/Vdd)	2.7V ~ 3.6V
Data Converters	A/D 8x10b
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	256-LBGA
Supplier Device Package	256-MAPBGA (17x17)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mcf5280cvm66j

Table 2-7. Fault Status Encodings

FS[3:0]	Definition
00*xx*	Reserved
0100	Error on instruction fetch
0101	Reserved
011x	Reserved
1000	Error on operand write
1001	Attempted write to write-protected space
101x	Reserved
1100	Error on operand read
1101	Reserved
111x	Reserved

- The 8-bit vector number, vector[7:0], defines the exception type and is calculated by the processor for all internal faults and represents the value supplied by the interrupt controller in case of an interrupt. See Table 2-5.

2.3.4 Processor Exceptions

2.3.4.1 Access Error Exception

The exact processor response to an access error depends on the memory reference being performed. For an instruction fetch, the processor postpones the error reporting until the faulted reference is needed by an instruction for execution. Therefore, faults during instruction prefetches followed by a change of instruction flow do not generate an exception. When the processor attempts to execute an instruction with a faulted opword and/or extension words, the access error is signaled and the instruction aborted. For this type of exception, the programming model has not been altered by the instruction generating the access error.

If the access error occurs on an operand read, the processor immediately aborts the current instruction's execution and initiates exception processing. In this situation, any address register updates attributable to the auto-addressing modes, (for example, (An)+,-(An)), have already been performed, so the programming model contains the updated An value. In addition, if an access error occurs during a MOVEM instruction loading from memory, any registers already updated before the fault occurs contain the operands from memory.

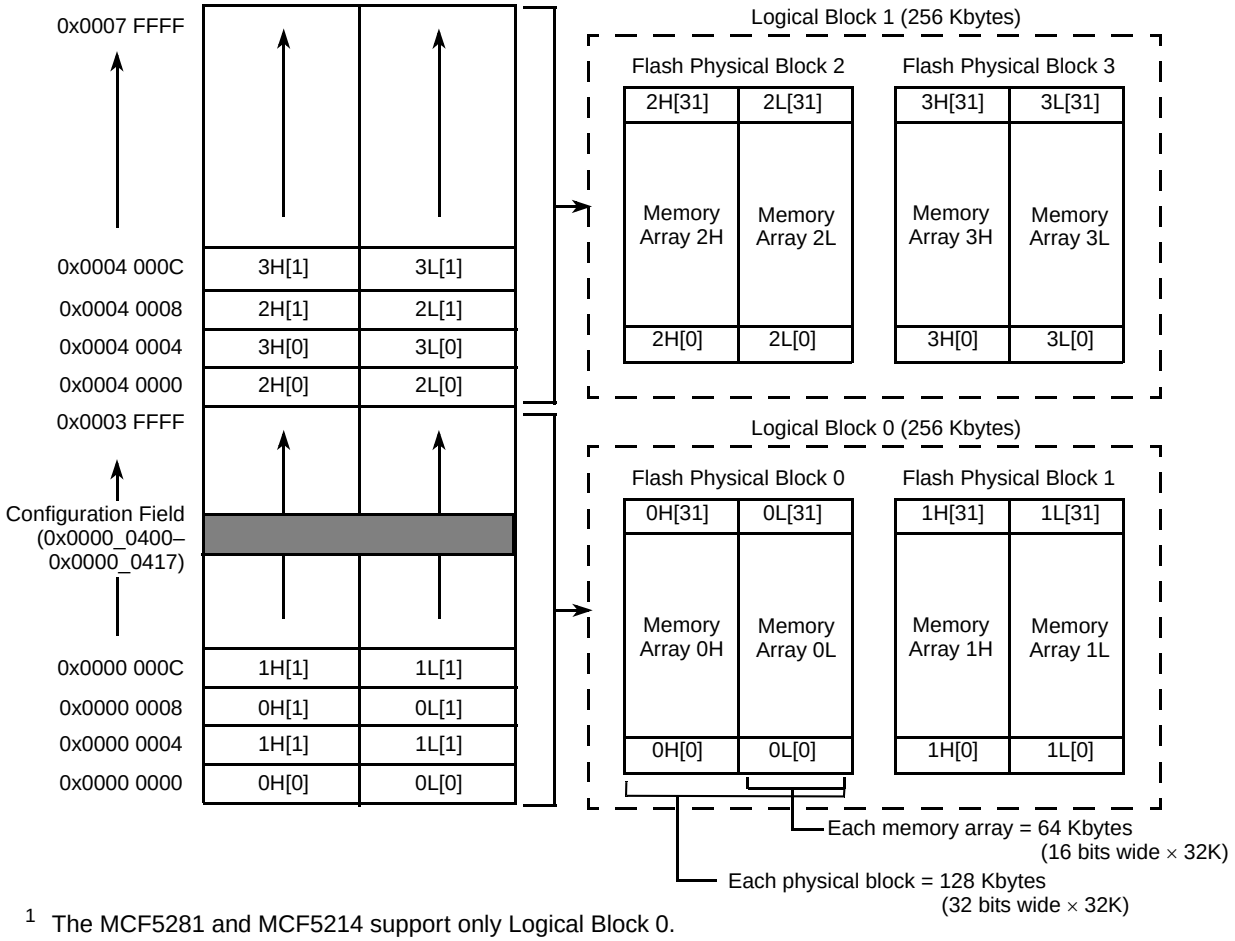
The V2 ColdFire processor uses an imprecise reporting mechanism for access errors on operand writes. Because the actual write cycle may be decoupled from the processor's issuing of the operation, the signaling of an access error appears to be decoupled from the instruction that generated the write. Accordingly, the PC contained in the exception stack frame merely represents the location in the program when the access error was signaled. All programming model updates associated with the write instruction are completed. The NOP instruction can collect access errors for writes. This instruction delays its

6.3 Memory Map

Figure 6-2 shows the memory map for the CFM array. The CFM array can reside anywhere in the memory space of the MCU. The starting address of the array is determined by the CFM array base address which must reside on a natural size boundary; that is, the CFM array base address must be an integer multiple of the array size. The CFM register space must reside on a 64 byte boundary as determined by the CFM register base address. Figure 6-2 shows how multiple 32,768 by 16-bit Flash physical blocks interleave to form a contiguous non-volatile memory space. Each pair of 32-bit blocks (even and odd) interleave every 4 bytes to form a 256-Kbyte section of memory.

NOTE

The CFM on the MCF5281 and MCF5214 is constructed with four banks of 32K x 16-bit Flash arrays to generate 256 Kbytes of 32-bit Flash memory.



[1] The MCF5281 and MCF5214 support only Logical Block 0.

Figure 6-2. CFM Array Memory Map

The CFM module has hardware interlocks to protect data from accidental corruption. The <<BLOCK NAME>> memory array is logically divided into 16-Kbyte sectors for the purpose of data protection and access control. A flexible scheme allows the protection of any combination of logical sectors (see Section 6.3.4.4, “CFM Protection Register (CFMPROT)”). A similar mechanism is available to control supervisor/user and program/data space access to these sectors.

[4] The BDM logic is clocked by a separate TCLK clock. Entering halt mode via the BDM port exits any low-power mode. Upon exit from halt mode, the previous low-power mode will be re-entered and changes made in halt mode will remain in effect.

Table 9-10. Stop Mode Operation (Sheet 2 of 5)

MODE In	LOCEN	LOCRE	LOLRE	PLL	OSC	EWKUP	Expected PLL Action at Stop	PLL Action During Stop	MODE Out	LOCKSS	LOCK	LOCS	Comments
NRM	0	0	0	Off	On	0	Lose lock	Regain	NRM	‘LK	1	‘LC	Block LOCKS from being cleared
								Lose reference clock or no lock regain	Stuck	—	—	—	
								Lose reference clock, regain	NRM	‘LK	1	‘LC	Block LOCKS from being cleared
NRM	0	0	0	Off	On	1	Lose lock	No lock regain	Unstable NRM	0–>‘LK	0–>1	‘LC	Block LOCKS until lock regained
								Lose reference clock or no f.b. clock regain	Stuck	—	—	—	
								Lose reference clock, regain	Unstable NRM	0–>‘LK	0–>1	‘LC	LOCS not set because LOCEN = 0
NRM	0	0	0	On	On	0	—	—	NRM	‘LK	1	‘LC	
								Lose lock or clock	Stuck	—	—	—	
								Lose lock, regain	NRM	0	1	‘LC	
								Lose clock and lock, regain	NRM	0	1	‘LC	LOCS not set because LOCEN = 0
NRM	0	0	0	On	On	1	—	—	NRM	‘LK	1	‘LC	
								Lose lock	Unstable NRM	0	0–>1	‘LC	
								Lose lock, regain	NRM	0	1	‘LC	
								Lose clock	Stuck	—	—	—	
								Lose clock, regain without lock	Unstable NRM	0	0–>1	‘LC	
								Lose clock, regain with lock	NRM	0	1	‘LC	
NRM	X	X	1	Off	X	X	Lose lock, f.b. clock, reference clock	RESET	RESET	—	—	—	Reset immediately

Table 12-5. Chip Select Registers

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0080	Chip select address register—bank 0 (CSAR0) [p. 12-6]		Reserved[1]	
0x00_0084	Chip select mask register—bank 0 (CSMR0) [p. 12-6]			
0x00_0088	Reserved[1]		Chip select control register—bank 0 (CSCR0) [p. 12-7]	
0x00_008C	Chip select address register—bank 1 (CSAR1) [p. 12-6]		Reserved[1]	
0x00_0090	Chip select mask register—bank 1 (CSMR1) [p. 12-6]			
0x00_0094	Reserved[1]		Chip select control register—bank 1 (CSCR1) [p. 12-7]	
0x00_0098	Chip select address register—bank 2 (CSAR2) [p. 12-6]		Reserved[1]	
0x00_009C	Chip select mask register—bank 2 (CSMR2) [p. 12-6]			
0x00_00A0	Reserved[1]		Chip select control register—bank 2 (CSCR2) [p. 12-7]	
0x00_00A4	Chip select address register—bank 3 (CSAR3) [p. 12-6]		Reserved[1]	
0x00_00A8	Chip select mask register—bank 3 (CSMR3) [p. 12-6]			
0x00_00AC	Reserved[1]		Chip select control register—bank 3 (CSCR3) [p. 12-7]	
0x00_00B0	Chip select address register—bank 4 (CSAR4) [p. 12-6]		Reserved[1]	
0x00_00B4	Chip select mask register—bank 4 (CSMR4) [p. 12-6]			
0x00_00B8	Reserved[1]		Chip select control register—bank 4 (CSCR4) [p. 12-7]	
0x00_00BC	Chip select address register—bank 5 (CSAR5) [p. 12-6]		Reserved[1]	
0x00_00C0	Chip select mask register—bank 5 (CSMR5) [p. 12-6]			
0x00_00C4	Reserved[1]		Chip select control register—bank 5 (CSCR5) [p. 12-7]	
0x00_00C8	Chip select address register—bank 6 (CSAR6) [p. 12-6]		Reserved[1]	
0x00_00CC	Chip select mask register—bank 6 (CSMR6) [p. 12-6]			
0x00_00D0	Reserved[1]		Chip select control register—bank 6 (CSCR6) [p. 12-7]	

[1] Addresses not assigned to a register and undefined register bits are reserved for expansion. Write accesses to these reserved address spaces and reserved register bits have no effect.

NOTE

Throughout this chapter “external request” and DREQ are used to refer to a DMA request from one of the on-chip UARTS or DMA timers. For details on the connections associated with DMA request inputs, see Section 16.2, “DMA Request Control (DMAREQC).”

16.1.1 DMA Module Features

The DMA controller module features are as follows:

- Four independently programmable DMA controller channels
- Auto-alignment feature for source or destination accesses
- Dual-address transfers
- Channel arbitration on transfer boundaries
- Data transfers in 8-, 16-, 32-, or 128-bit blocks using a 16-byte buffer
- Continuous-mode or cycle-steal transfers
- Independent transfer widths for source and destination
- Independent source and destination address registers

16.2 DMA Request Control (DMAREQC)

The DMAREQC register provides a software-controlled connection matrix for DMA requests. It logically routes DMA requests from the DMA timers and UARTs to the four channels of the DMA controller. Writing to this register determines the exact routing of the DMA request to the four channels of the DMA modules. If DCR*n*[EEXT] is set and the channel is idle, the assertion of the appropriate DREQ*n* activates channel *n*.

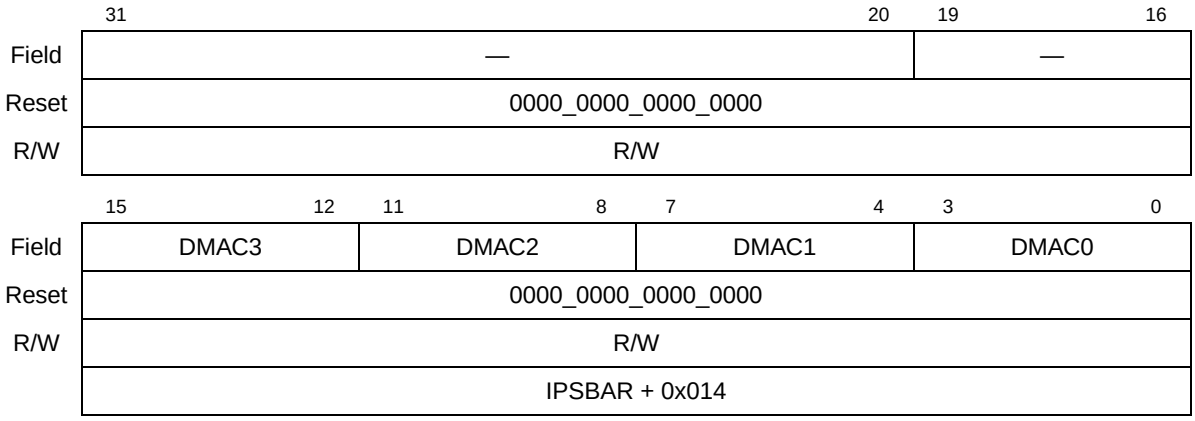


Figure 16-2. DMA Request Control Register (DMAREQC)

Table 16-1. DMAREQC Field Description

Bits	Name	Description

peripheral device or memory, the source address is the starting address of the data block. This can be any aligned byte address.

The DAR*n* should contain the destination (write) address. If the transfer is from a peripheral device to memory, or from memory to memory, the DAR*n* is loaded with the starting address of the data block to be written. If the transfer is from memory to a peripheral device, DAR*n* is loaded with the address of the peripheral data register. This address can be any aligned byte address.

SAR*n* and DAR*n* change after each cycle depending on DCR*n*[SSIZE,DSIZE, SINC,DINC] and on the starting address. Increment values can be 1, 2, 4, or 16 for byte, word, longword, or 16-byte line transfers, respectively. If the address register is programmed to remain unchanged (no count), the register is not incremented after the data transfer.

BCR*n*[BCR] must be loaded with the number of byte transfers to occur. It is decremented by 1, 2, 4, or 16 at the end of each transfer, depending on the transfer size. DSR*n*[DONE] must be cleared for channel startup.

As soon as the channel has been initialized, it is started by writing a one to DCR*n*[START] or asserting DREQ*n*, depending on the status of DCR*n*[EEXT]. Programming the channel for internal requests causes the channel to request the bus and start transferring data immediately. If the channel is programmed for external request, DREQ*n* must be asserted before the channel requests the bus.

Changes to DCR*n* are effective immediately while the channel is active. To avoid problems with changing a DMA channel setup, write a one to DSR*n*[DONE] to stop the DMA channel.

16.5.4 Data Transfer

This section describes auto-alignment and bandwidth control for DMA transfers.

16.5.4.1 Auto-Alignment

Auto-alignment allows block transfers to occur at the optimal size based on the address, byte count, and programmed size. To use this feature, DCR*n*[AA] must be set. The source is auto-aligned if DCR*n*[SSIZE] indicates a transfer size larger than DCR*n*[DSIZE]. Source alignment takes precedence over the destination when the source and destination sizes are equal. Otherwise, the destination is auto-aligned. The address register chosen for alignment increments regardless of the increment value. Configuration error checking is performed on registers not chosen for alignment.

If BCR*n* is greater than 16, the address determines transfer size. Bytes, words, or longwords are transferred until the address is aligned to the programmed size boundary, at which time accesses begin using the programmed size.

If BCR*n* is less than 16 at the start of a transfer, the number of bytes remaining dictates transfer size. For example, AA = 1, SAR*n* = 0x0001, BCR*n* = 0x00F0, SSIZE = 00 (longword), and DSIZE = 01 (byte). Because SSIZE > DSIZE, the source is auto-aligned. Error checking is performed on destination registers. The access sequence is as follows:

1. Read byte from 0x0001—write 1 byte, increment SAR*n*.
2. Read word from 0x0002—write 2 bytes, increment SAR*n*.
3. Read longword from 0x0004—write 4 bytes, increment SAR*n*.
4. Repeat longwords until SAR*n* = 0x00F0.
5. Read byte from 0x00F0—write byte, increment SAR*n*.

IPSBAR Offset: 0x10EC — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	OPCODE																PAUSE_DUR															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 17-14. Opcode/Pause Duration Register (OPD)

Table 17-18. OPD Field Descriptions

Field	Description
31–16 OPCODE	Opcode field used in PAUSE frames. These read-only bits are a constant, 0x0001.
15–0 PAUSE_DUR	Pause Duration field used in PAUSE frames.

17.4.15 Descriptor Individual Upper Address Register (IAUR)

IAUR contains the upper 32 bits of the 64-bit individual address hash table. The address recognition process uses this table to check for a possible match with the destination address (DA) field of receive frames with an individual DA. This register is not reset and you must initialize it.

IPSBAR Offset: 0x1118 — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IADDR1																															
W																																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 17-15. Descriptor Individual Upper Address Register (IAUR)

Table 17-19. IAUR Field Descriptions

Field	Description
31–0 IADDR1	The upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR1 contains hash index bit 63. Bit 0 of IADDR1 contains hash index bit 32.

17.4.16 Descriptor Individual Lower Address Register (IALR)

IALR contains the lower 32 bits of the 64-bit individual address hash table. The address recognition process uses this table to check for a possible match with the DA field of receive frames with an individual DA. This register is not reset and you must initialize it.

Table 20-16. GPTFLG2 Field Descriptions

Bit(s)	Name	Description
7	TOF	Timer overflow flag. Set when the GPT counter rolls over from 0xFFFF to 0x0000. If the TOI bit in GPTSCR2 is also set, TOF generates an interrupt request. This bit is read anytime, write anytime (writing 1 clears the flag, and writing 0 has no effect). 1 Timer overflow 0 No timer overflow **Note:** When the GPT channel 3 registers contain 0xFFFF and TCRE is set, TOF does not get set even though the GPT counter registers go from 0xFFFF to 0x0000. When TOF is set, it does not inhibit subsequent overflow events.
6–0	—	Reserved, should be cleared.

Note: When the fast flag clear all bit, GPTSCR1[TFFCA], is set, any access to the GPT counter registers clears GPT flag register 2.

20.5.14 GPT Channel Registers (GPTC*n*)

	15 … 0
Field	CCNT
Reset	0000_0000_0000_0000
R/W	R/W
Address	IPSBAR + 0x1A_0010, 0x1A_0012, 0x1A_0014, 0x1A_0016, 0x1B_0010, 0x1B_0012, 0x1B_0014, 0x1B_0016

Figure 20-16. GPT Channel[0:3] Register (GPTC*n*)

Table 20-17. GPTC*n* Field Descriptions

Bit(s)	Name	Description
15–0	CCNT	When a channel is configured for input capture (IOS*n* = 0), the GPT channel registers latch the value of the free-running counter when a defined transition occurs on the corresponding input capture pin. When a channel is configured for output compare (IOS*n* = 1), the GPT channel registers contain the output compare value. To ensure coherent reading of the GPT counter, such that a timer rollover does not occur between back-to-back 8-bit reads, it is recommended that only word (16-bit) accesses be used. These bits are read anytime, write anytime (for the output compare channel); writing to the input capture channel has no effect.

Table 20-18. GPTPACTL Field Descriptions (continued)

Bit(s)	Name	Description
1	PAOVI	Pulse accumulator overflow interrupt enable. Enables the PAOVF flag to generate interrupt requests. 1 PAOVF interrupt requests enabled 0 PAOVF interrupt requests disabled
0	PAI	Pulse accumulator input interrupt enable. Enables the PAIF flag to generate interrupt requests. 1 PAIF interrupt requests enabled 0 PAIF interrupt requests disabled

20.5.16 Pulse Accumulator Flag Register (GPTPAFLG)

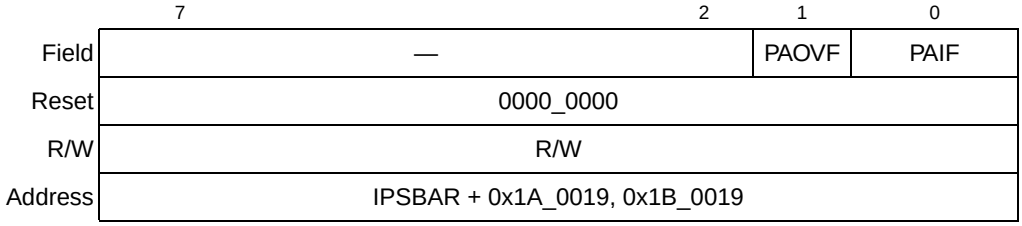


Figure 20-18. Pulse Accumulator Flag Register (GPTPAFLG)

Table 20-19. GPTPAFLG Field Descriptions

Bit(s)	Name	Description
7–2	—	Reserved, should be cleared.
1	PAOVF	Pulse accumulator overflow flag. Set when the 16-bit pulse accumulator rolls over from 0xFFFF to 0x0000. If the GPTPACTL[PAOVI] bit is also set, PAOVF generates an interrupt request. Clear PAOVF by writing a 1 to it. This bit is read anytime, write anytime. (Writing 1 clears the flag; writing 0 has no effect.) 1 Pulse accumulator overflow 0 No pulse accumulator overflow
0	PAIF	Pulse accumulator input flag. Set when the selected edge is detected at the PAI pin. In event counter mode, the event edge sets PAIF. In gated time accumulation mode, the trailing edge of the gate signal at the PAI pin sets PAIF. If the PAI bit in GPTPACTL is also set, PAIF generates an interrupt request. Clear PAIF by writing a 1 to it. 1 Active PAI input 0 No active PAI input

NOTE

When the fast flag clear all enable bit, GPTSCR1[TFFCA], is set, any access to the pulse accumulator counter registers clears all the flags in GPTPAFLG.

For example, if a 80-MHz timer clock is divided by 16, DTMR*n*[PS] equals 0x7F, and the timer is referenced at 0x1312C (78,124 decimal), the time-out period is:

$$\text{Timeout period} = \frac{1}{80 \times 10^6} \times 16 \times (127 + 1) \times (78124 + 1) = 2.00 \text{ seconds} \qquad \textit{Eqn. 21-2}$$

Table 23-3. UMR1*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>2
PT</td><td>Parity type. PM and PT together select parity type (PM = 0x) or determine whether a data or address character is transmitted (PM = 11).
<table>
<tr><th>PM</th><th>Parity Mode</th><th>Parity Type (PT= 0)</th><th>Parity Type (PT= 1)</th></tr>
<tr><td>00</td><td>With parity</td><td>Even parity</td><td>Odd parity</td></tr>
<tr><td>01</td><td>Force parity</td><td>Low parity</td><td>High parity</td></tr>
<tr><td>10</td><td>No parity</td><td colspan="2">N/A</td></tr>
<tr><td>11</td><td>Multidrop mode</td><td>Data character</td><td>Address character</td></tr>
</table></td></tr>
<tr><td>1–0
B/C</td><td>Bits per character. Selects the number of data bits per character to be sent. The values shown do not include start, parity, or stop bits.
00 5 bits
01 6 bits
10 7 bits
11 8 bits</td></tr>
</table>

23.3.2 UART Mode Register 2 (UMR2*n*)

The UMR2*n* registers control UART module configuration. UMR2*n* can be read or written when the mode register pointer points to it, which occurs after any access to UMR1*n*. UMR2*n* accesses do not update the pointer.

IPSBAR 0x00_0200 (UMR20)
Offset: 0x00_0240 (UMR21)
0x00_0280 (UMR22)

Access: User read/write[1]

	7	6	5	4	3	2	1	0
R W	CM		TXRTS	TXCTS	SB			
Reset:	0	0	0	0	0	0	0	0

1 After UMR1*n* is read or written, the pointer points to UMR2*n*

Figure 23-4. UART Mode Registers 2 (UMR2*n*)

24.3.7 Clock Synchronization and Arbitration

I²C is a true multi-master bus that allows more than one master connected to it. If two or more master devices simultaneously request control of the bus, a clock synchronization procedure determines the bus clock. Because wire-AND logic is performed on the I2C_SCL line, a high-to-low transition on the I2C_SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the I2C_SCL line low until the clock high state is reached. However, change of low to high in this device's clock may not change the state of the I2C_SCL line if another device clock remains within its low period. Therefore, synchronized clock I2C_SCL is held low by the device with the longest low period.

Devices with shorter low periods enter a high wait state during this time (see Figure 24-12). When all devices concerned have counted off their low period, the synchronized clock (I2C_SCL) line is released and pulled high. At this point, the device clocks and the I2C_SCL line are synchronized, and the devices start counting their high periods. The first device to complete its high period pulls the I2C_SCL line low again.

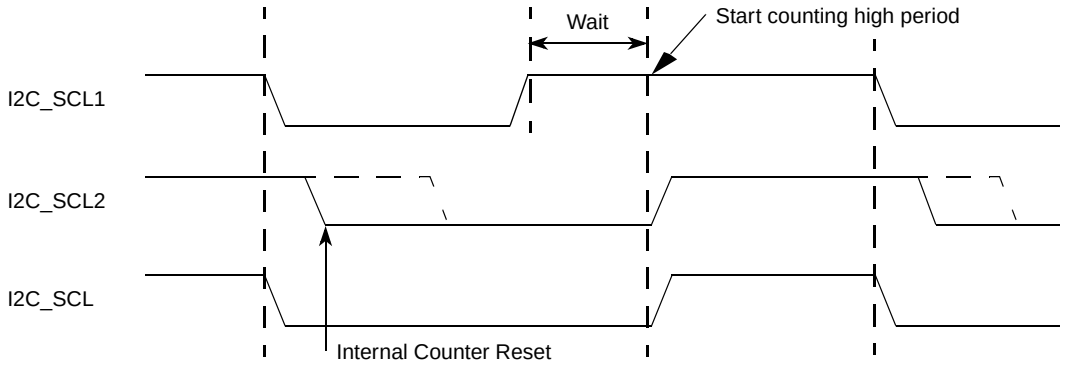


Figure 24-12. Clock Synchronization

A data arbitration procedure determines the relative priority of the contending masters. A bus master loses arbitration if it transmits logic 1 while another master transmits logic 0. The losing masters immediately switch over to slave receive mode and stop driving I2C_SDA output (see Figure 24-13). In this case, transition from master to slave mode does not generate a STOP condition. Meanwhile, hardware sets I2SR[IAL] to indicate loss of arbitration.

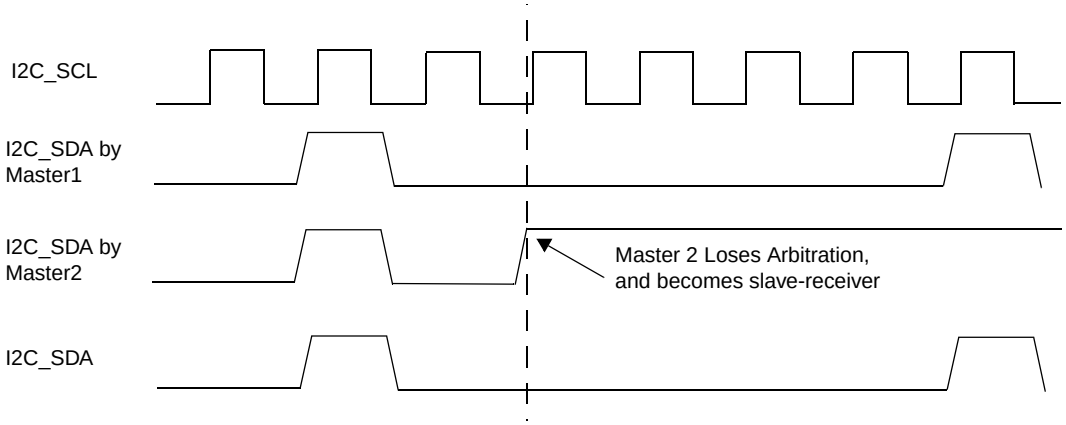


Figure 24-13. Arbitration Procedure

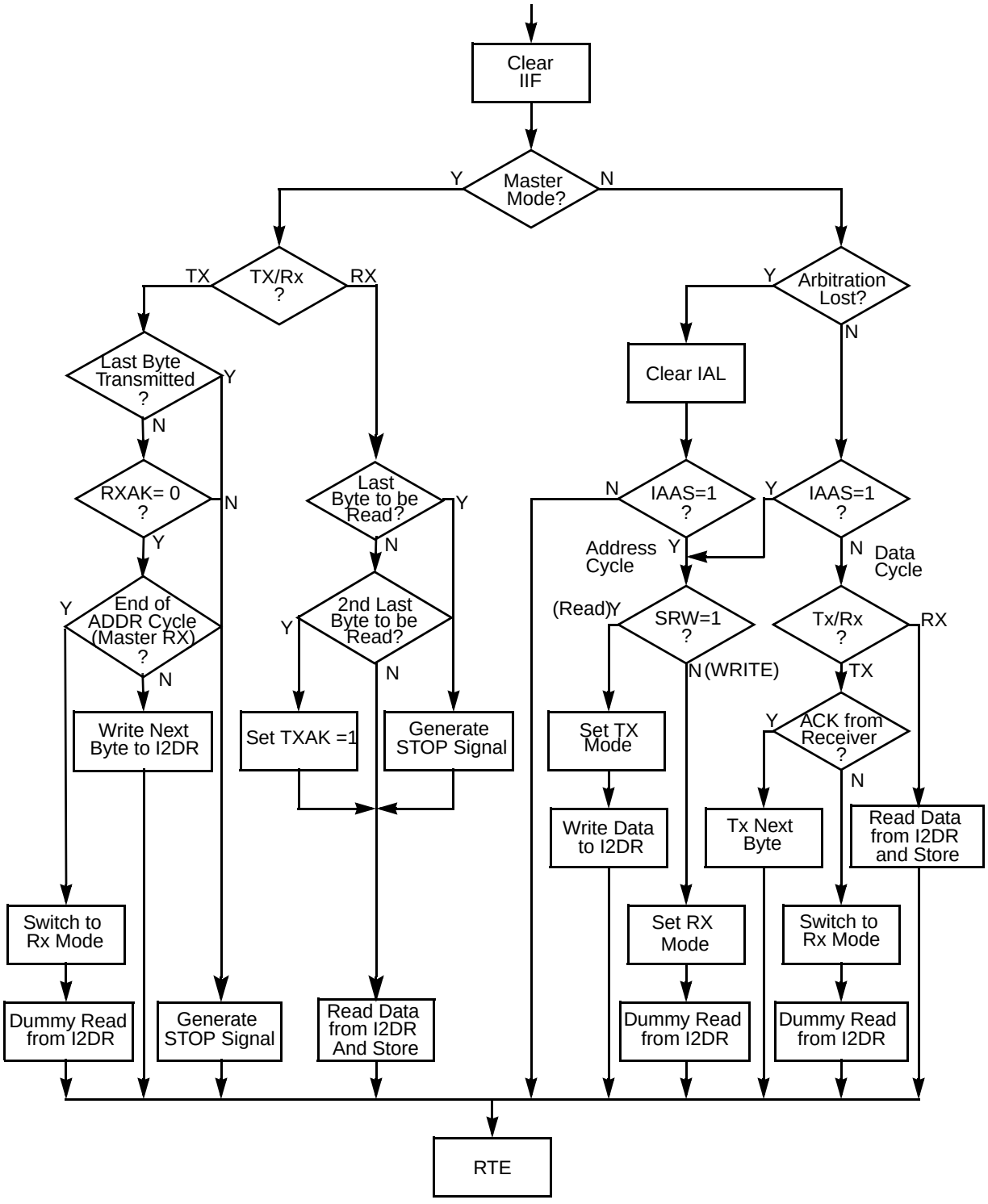


Figure 24-14. Flow-Chart of Typical I²C Interrupt Routine

Chapter 25
FlexCAN

The FlexCAN module is a communication controller implementing the controller area network (CAN) protocol, an asynchronous communications protocol used in automotive and industrial control systems. It is a high speed (1 Mbit/sec), short distance, priority based protocol which can communicate using a variety of mediums (for example, fiber optic cable or an unshielded twisted pair of wires). The FlexCAN supports both the standard and extended identifier (ID) message formats specified in the CAN protocol specification, revision 2.0, part B.

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. A general working knowledge of the CAN protocol revision 2.0 is assumed in this document. For details, refer to the CAN protocol revision 2.0 specification.

25.1 Features

- Based on and includes all existing Freescale TouCAN module features
- Freescale IP interface architecture
- Full implementation of the CAN protocol specification version 2.0
  - Standard data and remote frames (up to 109 bits long)
  - Extended data and remote frames (up to 127 bits long)
  - 0–8 bytes data length
  - Programmable bit rate up to 1Mbit/sec
- Up to 16 flexible message buffers of 0–8 bytes data length, each configurable as Rx or Tx, all supporting standard and extended messages
- Listen-only mode capability
- Content-related addressing
- No read/write semaphores
- Three programmable mask registers: global (for MBs 0-13), special for MB14, and special for MB15
- Programmable transmit-first scheme: lowest ID or lowest buffer number
- “Time Stamp”, based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Programmable I/O modes
- Maskable interrupts
- Independent of the transmission medium (external transceiver is assumed)
- Open network architecture
- Multimaster bus
- High immunity to EMI
- Short latency time for high-priority messages
- Low-power “sleep” mode, with programmable “wake up” on bus activity

The FlexCAN responds to any bus state as described in the protocol, e.g. transmit error active or error passive flag, delay its transmission start time (Error Passive) and avoid any influence on the bus when in Bus Off state. The following are the basic rules for FlexCAN bus state transitions:

- If the value of TXCTR or RXCTR increases to be greater than or equal to 128, the FCS field in the error status register is updated to reflect it (set Error Passive state).
- If the FlexCAN state is Error Passive, and either TXCTR counter or RXCTR then decrements to a value less than or equal to 127 while the other already satisfies this condition, the ESTAT[FCS] field is updated to reflect it (set Error Active state).
- If the value of the TXCTR increases to be greater than 255, the ESTAT[FCS] field is updated to reflect it (set Bus Off state) and an interrupt may be issued. The value of TXCTR is then reset to zero.
- If the FlexCAN state is Bus_Off, then TXCTR, together with an internal counter are cascaded to count the 128 occurrences of 11 consecutive recessive bits on the bus. Hence, TXCTR is reset to zero, and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the TXCTR. When TXCTR reaches the value of 128, ESTAT[FCS] is updated to be Error Active, and both error counters are reset to zero. At any instance of dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero, but does NOT affect the TXCTR value.
- If during system start-up, only one node is operating, then its TXCTR increases with each message it’s trying to transmit as a result of ACK_ERROR. A transition to bus state Error Passive should be executed as described, while this device never enters the Bus_Off state.
- If the RXCTR increases to a value greater than 127, it is no longer incremented, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127, in order to return to Error Active state.

25.4.10 FlexCAN Initialization Sequence

Initialization of the FlexCAN includes the initial configuration of the message buffers and configuration of the CAN communication parameters following a reset, as well as any reconfiguration which may be required during operation. The following is a generic initialization sequence for the FlexCAN:

1. Initialize all operation modes
   a) Initialize the transmit and receive pin modes in control register 0 (CANCTRL0).
   b) Initialize the bit timing parameters PROPSEG, PSEGS1, PSEG2, and RJW in control registers 1 and 2 (CANCTRL[1:2]).
   c) Select the S-clock rate by programming the PRESDIV register.
   d) Select the internal arbitration mode (LBUF bit in CANCTRL1).
2. Initialize message buffers
   a) The control/status word of all message buffers must be written either as an active or inactive message buffer.
   b) All other entries in each message buffer should be initialized as required.
3. Initialize mask registers for acceptance mask as needed
4. Initialize FlexCAN interrupt handler
   a) Initialize the interrupt configuration register (ICR*n*) with a specific request level and vector base address.
   b) Set the required mask bits in the IMASK register (for all message buffer interrupts), in CANCTRL0 (for bus off and error interrupts), and in CANMCR for the WAKE interrupt.

28.4.2.2 Port QB Digital I/O Signals

Port QB signals are referred to as PQB[3:0] when used as a 4-bit digital input/output port. In addition to functioning as analog input signals, the port QB signals are also connected to the input of a synchronizer during reads and may be used as general-purpose digital inputs when the applied voltages meet V_{IH} and V_{IL} requirements.

Each port QB signal is configured as an input or output by programming the port data direction register (DDRQB). The digital input signal states are read from the port QB data register (PORTQB) when DDRQB specifies that the signals are inputs. The digital data in PORTQB is driven onto the port QB signals when the corresponding bits in DDRQB specify output. See Section 28.6.4, “Port QA and QB Data Direction Register (DDRQA & DDRQB).

28.4.3 External Trigger Input Signals

The QADC has two external trigger signals, ETRIG2 and ETRIG1. Each external trigger input is associated with one of the scan queues, queue 1 or queue 2. The assignment of ETRIG[2:1] to a queue is made by the TRG bit in QADC control register 0 (QACR0). When TRG = 0, ETRIG1 triggers queue 1 and ETRIG2 triggers queue 2. When TRG = 1, ETRIG1 triggers queue 2 and ETRIG2 triggers queue 1. See Section 28.6.5, “Control Registers “Control Registers.”

28.4.4 Multiplexed Address Output Signals

In non-multiplexed mode, the QADC analog input signals are connected to an internal multiplexer which routes the analog signals into the internal A/D converter.

In externally multiplexed mode, the QADC allows automatic channel selection through up to four external 4-to-1 multiplexer chips. The QADC provides a 2-bit multiplexed address output to the external multiplexer chips to allow selection of one of four inputs. The multiplexed address output signals, MA1 and MA0, can be used as multiplexed address output bits or as general-purpose I/O when external multiplexed mode is not being used.

MA[1:0] are used as the address inputs for up to four 4-channel multiplexer chips. Because the MA[1:0] signals are digital outputs in multiplexed mode, the state of their corresponding data direction bits in DDRQA is ignored.

28.4.5 Multiplexed Analog Input Signals

In external multiplexed mode, four of the port QB signals are redefined so that each represent four analog input channels. See Table 28-1.

Table 28-1. Multiplexed Analog Input Channels

Multiplexed Analog Input	Channels
ANW	Even numbered channels from 0 to 6
ANX	Odd numbered channels from 1 to 7
ANY	Even numbered channels from 16 to 22
ANZ	Odd numbered channels from 17 to 23

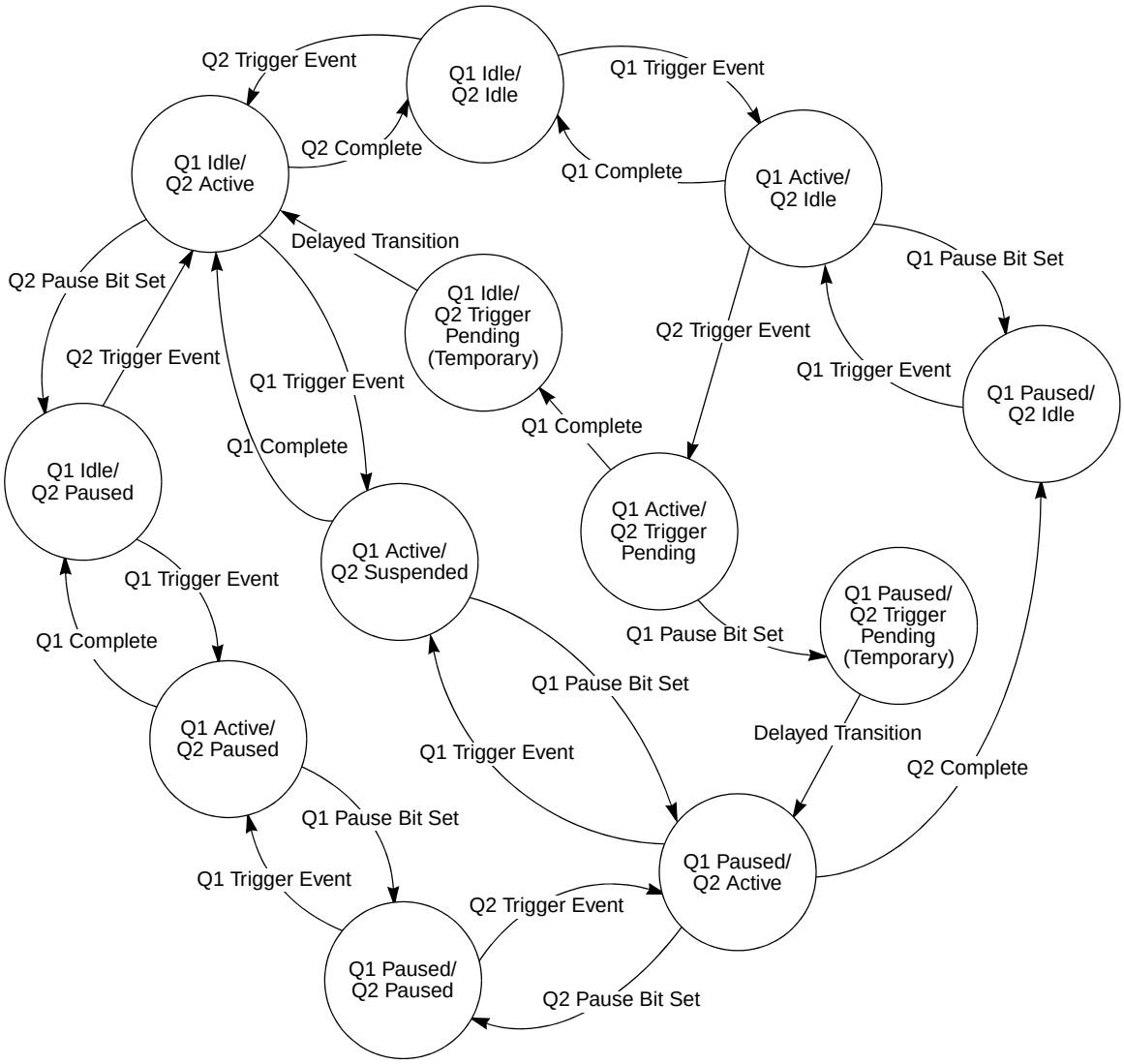


Figure 28-12. Queue Status Transition

28.6.6.2 QADC Status Register 1 (QASR1)

Stop mode resets this register .

	15	14	13	12	11	10	9	8
Field	—		CWPQ15	CWPQ14	CWPQ13	CWPQ12	CWPQ11	CWPQ10
Reset	0011_1111							
R/W:	R							

The following paragraphs and figures outline the prioritizing criteria used to determine which conversion occurs in each overlap situation.

NOTE

Each situation in Figure 28-23 through Figure 28-33 is labeled S1 through S19. In each diagram, time is shown increasing from left to right. The execution of queue 1 and queue 2 (Q1 and Q2) is shown as a string of rectangles representing the execution time of each CCW in the queue. In most of the situations, there are four CCWs (labeled C1 to C4) in both queue 1 and queue 2. In some of the situations, CCW C2 is presumed to have the pause bit set, to show the similarities of pause and end-of-queue as terminations of queue execution.

Trigger events are described in Table 28-22.

Table 28-22. Trigger Events

Trigger	Events
T1	Events that trigger queue 1 execution (external trigger, software-initiated single-scan enable bit, or completion of the previous continuous loop)
T2	Events that trigger queue 2 execution (external trigger, software-initiated single-scan enable bit, timer period/interval expired, or completion of the previous continuous loop)

When a trigger event causes a CCW execution in progress to be aborted, the aborted conversion is shown as a ragged end of a shortened CCW rectangle.

The situation diagrams also show when key status bits are set.
Table 28-23 describes the status bits.

Table 28-23. Status Bits

Bit	Function
CF flag	Set when the end of the queue is reached
PF flag	Set when a queue completes execution up through a pause bit
Trigger overrun error (TOR)	Set when a new trigger event occurs before the queue is finished servicing the previous trigger event

Below the queue execution flows are three sets of blocks that show the status information that is made available to the user. The first two rows of status blocks show the condition of each queue as:

- Idle
- Active
- Pause
- Suspended (queue 2 only)
- Trigger pending

The third row of status blocks shows the 4-bit QS status register field that encodes the condition of the two queues. Two transition status cases, QS = 0011 and QS = 0111, are not shown because they exist only very briefly between stable status conditions.

The first three examples in Figure 28-23 through Figure 28-25 (S1, S2, and S3) show what happens when a new trigger event is recognized before the queue has completed servicing the previous trigger event on the same queue.

In situation S1 (Figure 28-23), one trigger event is being recognized on each queue while that queue is still working on the previously recognized trigger event. The trigger overrun error status bit is set, and the premature trigger event is otherwise ignored. A trigger event that occurs before the servicing of the previous trigger event is through does not disturb the queue execution in progress.

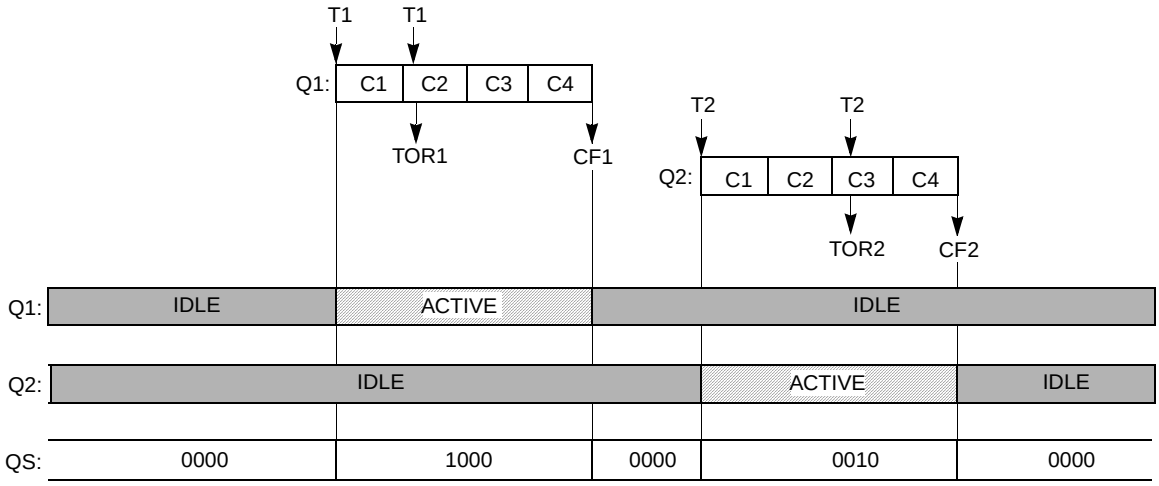


Figure 28-23. CCW Priority Situation 1

In situation S2 (Figure 28-24), more than one trigger event is recognized before servicing of a previous trigger event is complete. The trigger overrun bit is again set, but the additional trigger events are otherwise ignored. After the queue is complete, the first newly detected trigger event causes queue execution to begin again. When the trigger event rate is high, a new trigger event can be seen very soon after completion of the previous queue, leaving little time to retrieve the previous results. Also, when trigger events are occurring at a high rate for queue 1, the lower priority queue 2 channels may not get serviced at all.

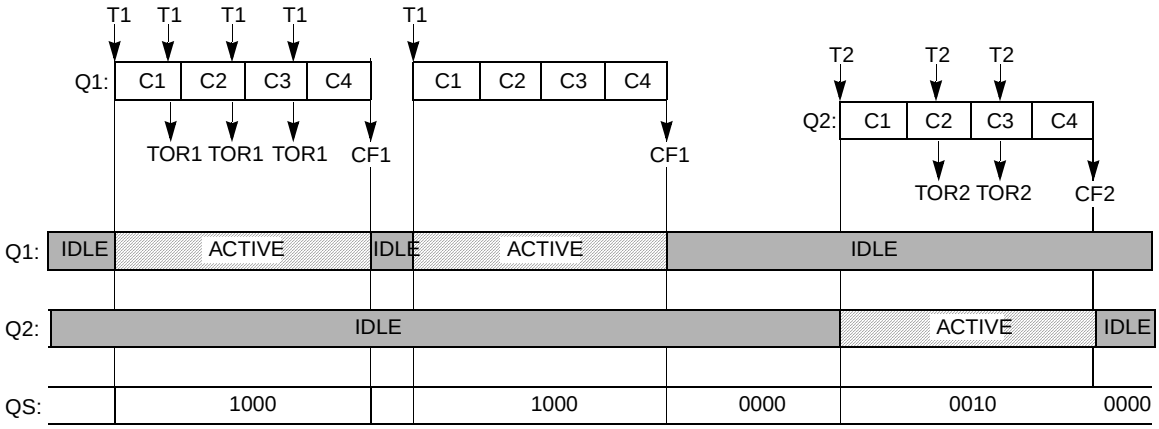


Figure 28-24. CCW Priority Situation 2

Situation S3 (Figure 28-25) shows that when the pause feature is used, the trigger overrun error status bit is set the same way and that queue execution continues unchanged.