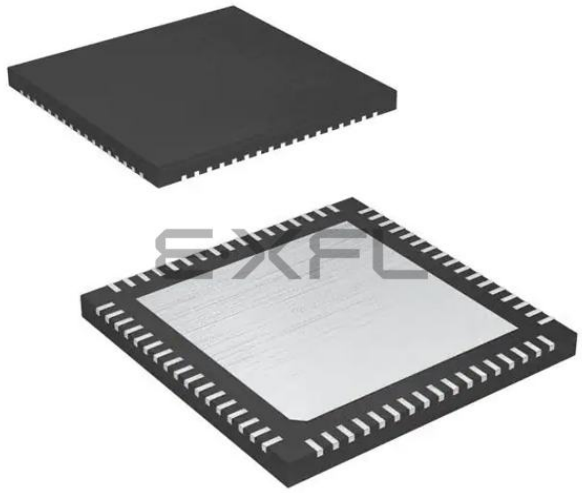
# Analog Devices Inc./Maxim Integrated - MAXQ2000-QAX+ Datasheet



**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 | - |
| Core Size | 16-Bit |
| Speed | 20MHz |
| Connectivity | 1-Wire®, SPI, UART/USART |
| Peripherals | LCD, POR, PWM, WDT |
| Number of I/O | 50 |
| Program Memory Size | 64KB (32K x 16) |
| Program Memory Type | FLASH |
| EEPROM Size | - |
| RAM Size | 2K x 8 |
| Voltage - Supply (Vcc/Vdd) | 1.8V ~ 2.75V |
| Data Converters | - |
| Oscillator Type | Internal |
| Operating Temperature | -40°C ~ 85°C (TA) |
| Mounting Type | Surface Mount |
| Package / Case | 68-VFQFN Exposed Pad |
| Supplier Device Package | 68-QFN (10x10) |
| Purchase URL | https://www.e-xfl.com/product-detail/analog-devices/maxq2000-qax |

- Real-Time Clock
- 1-Wire® Bus Master
- General-Purpose Digital I/O Ports

## 1.4 MAXQ10 and MAXQ20 Microcontrollers

This user's guide covers both the 8-bit MAXQ10 and 16-bit MAXQ20 microcontrollers. The primary difference between the MAXQ10 and MAXQ20 implementations is the width of the internal data bus and ALU. The MAXQ10 design implements an 8-bit internal data bus and ALU, while the MAXQ20 design implements a 16-bit internal data bus and ALU. This difference is most evident when comparing the instruction set, and more specifically, those operations that involve the ALU and accumulators. The registers on the MAXQ10 and MAXQ20 can be either 8 bits or 16 bits wide.

*1-Wire is a registered trademark of Dallas Semiconductor Corp.*

page (16kWords) may be logically mapped, as just defined, to either the upper or lower half of data memory. If word access mode is selected, two pages (32kWords total) may be logically mapped to data memory. To avoid memory overlapping in the byte access mode, the physical data memory should be confined to the address range x0000h to x3FFFh in word mode. The selection of physical memory page or pages to be logically mapped to data space is determined by the Code Access Bits (CDA1:0):

| CDA1:0 | SELECTED PAGE IN BYTE MODE | SELECTED PAGE IN WORD MODE |
|---|---|---|
| 00 | P0 | P0 and P1 |
| 01 | P1 | P0 and P1 |
| 10 | P2 | P2 and P3 |
| 11 | P3 | P2 and P3 |

Figure 2-3 and Figure 2-4 summarize the default memory maps for this memory structure. The primary difference lies in the reset default settings for the data pointer Word/Byte Mode Select (WBSn) bits. The WBSn bits of the MAXQ10 default to byte access mode (WBSn = 0), while the MAXQ20 WBSn bits default to word access mode (WBSn = 1).

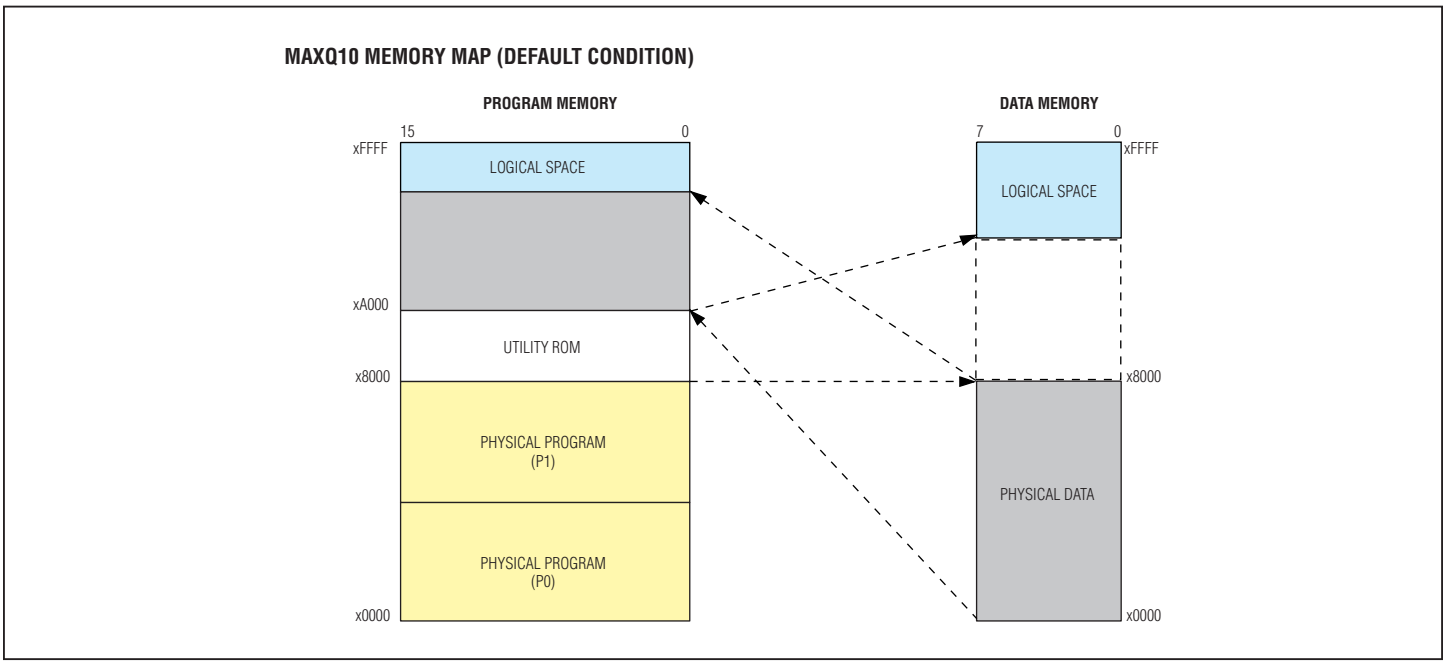



*Figure 2-3. Pseudo Von Neumann Memory Map (MAXQ10 Default)*

When executing from the data memory (only allowable when UPA = 0):

- Program flows freely between the lower 32k user code (P0 and P1) and the utility ROM segment.
- The upper half of the code segment (P2 and P3) is not accessible as program (since UPA = 0).
- The utility ROM can be accessed as data with offset at x8000h.
- One page (byte access mode) or two pages (word access mode) can be accessed as data with offset at x0000h as determined by the CDA1:0 bits.

## 2.6 Data Alignment

To support merged program and data memory operation while maintaining efficiency on memory space usage, the data memory must be able to support both byte-wide and word-wide accessing. Data is aligned in data memory as word, but the effective data address is resolved to bytes. This data alignment allows direct program fetching in its native word size while maintaining accessibility at the byte level. It is important to realize that this accessibility requires strict word alignment. All executable words must align to an even address in byte mode. Care must be taken when updating the code segment in the unified data memory space as misalignment of words will likely result in loss of program execution control. Worst yet, this situation may not be detected if the watchdog timer is also disabled.

Data memory is organized as two byte-wide memory banks with common word address decode but two 8-bit data buses. The data memory will always be read as a complete word, independent of operation, whether program fetch or data access. The program decoder always uses the full 16-bit word, whereas the data access can utilize a word or an individual byte.

In byte mode, data pointer hardware reads out the word containing the selected byte using the effective data word address pointer (the least significant bit of the byte data pointer is not initially used). Then, the least significant data pointer bit functions as the byte select that is used to place the target byte to the data path. For write access, data pointer hardware addresses a particular word using the effective data word address while the least significant bit selects the corresponding data bank for write, leaving the contents of the another memory bank unaffected.

### 2.6.1 Memory Management Unit

Memory allocation and accessing control for program and data memory can be managed by the memory management unit (MMU). A single memory management unit option is discussed in this User Guide, however the memory management unit implementation for any given product depends upon the type and amount of memory addressable by the device. Users should consult the individual product data sheet(s) and/or user's guide supplement(s) for detailed information.

Although supporting less than the maximum addressable program and data memory segments, the MMU implementation presented provides a high degree of programming and access control flexibility. It supports the following:

- User program memory up to 32k x 16 (up to 64k x 16 with inclusion of UPA bit).
- Utility ROM up to 8k x 16.
- Data memory SRAM up to 16k x 16.
- In-system and in-application programming of embedded EEPROM, Flash, or SRAM memories.
- Access to any of the three memory areas (SRAM, code memory, utility ROM) using the data memory pointers.
- Execution from any of the three memory areas (SRAM, code memory, factory written and tested utility-ROM routines).

Given these capabilities, the following rules apply to the memory map:

- A particular memory segment cannot be simultaneously accessed as both program and data.
- The offset address is xA000h when logically mapping data memory into the program space.
- The offset for logically mapping the utility ROM into the data memory space is x8000h.
- Program memory:
  - The lower half of the program memory (P0 and P1) is always accessible, starting at x0000h.
  - The upper half of the program memory (P2 and P3) must be activated by setting the UPA bit to 1 when accessing for code execution, starting at x8000h.
  - Setting the UPA bit to 1 disallows access to the utility ROM and logical data memory as program.

- if the system clock divide ratio is 2, the interrupt request is recognized after 2 system clock;
- if the system clock divide ratio is 4 or greater, the interrupt request is recognized after 1 system clock;

An interrupt request with a pulse width less than three undivided clock cycles is not recognized. Note that the granularity of interrupt source is at module level. Synchronous interrupts and sampled asynchronous interrupts assigned to the same module product a single interrupt to the interrupt handler.

External interrupts, when enabled, can be used as switchback sources from power management mode. There is no latency associated with the switchback because the circuit is being clocked by an undivided clock source versus the divide-by-256 system clock. For the same reason, there is no latency for other switchback sources that do not qualify as interrupt sources.

### 2.8.4 Interrupt Prioritization by Software

All interrupt sources of the MAXQ microcontroller naturally have the same priority. However, when CPU operation vectors to the programmed Interrupt Vector address, the order in which potential interrupt sources are interrogated is left entirely up to the user, as this often depends upon the system design and application requirements. The Interrupt Mask system register provides the ability to knowingly block interrupts from modules considered to be of lesser priority and manually re-enable the interrupt servicing by the CPU (by setting INS = 0). Using this procedure, a given interrupt service routine can continue executing, only to be interrupted by higher priority interrupts. An example demonstrating this software prioritization is provided in the *Handling Interrupts* section of *Section 3: Programming*.

### 2.8.5 Interrupt Exception Window

An interrupt exception window is a noninterruptable execution cycle. During this cycle, the interrupt handler does not respond to any interrupt requests. All interrupts that would normally be serviced during an interrupt exception window are delayed until the next execution cycle.

Interrupt exception windows are used when two or more instructions must be executed consecutively without any delays in between. Currently, there is a single condition in the MAXQ microcontroller that causes an interrupt exception window: activation of the prefix (PFX) register.

When the prefix register is activated by writing a value to it, it retains that value only for the next clock cycle. For the prefix value to be used properly by the next instruction, the instruction that sets the prefix value and the instruction that uses it must always be executed back to back. Therefore, writing to the PFX register causes an interrupt exception window on the next cycle. If an interrupt occurs during an interrupt exception window, an additional latency of one cycle in the interrupt handling will be caused as the interrupt will not be serviced until the next cycle.

## 2.9 Operating Modes

In addition to the standard program execution mode, there are three other operating modes for the MAXQ. During Reset Mode, the processor is temporarily halted by an external or internal reset source. During Power Management Mode, the processor executes instructions at a reduced clock rate to decrease power consumption. Stop Mode halts execution and all internal clocks to save power until an external stimulus indicates that processing should be resumed.

### 2.9.1 Reset Mode

When the MAXQ microcontroller is in Reset Mode, no instruction execution or other system or peripheral operations occur, and all input/output pins return to default states. Once the condition that caused the reset (whether internal or external) is removed, the processor begins executing code at address 8000h.

There are four different sources that can cause the MAXQ to enter Reset Mode:

- Power-On/Brownout Reset
- External Reset
- Watchdog Timer Reset
- Internal System Reset

#### 2.9.1.1 Power-On/Brownout Reset

An on-chip power-on reset (POR) circuit is provided to ensure proper initialization on internal device states. The power-on reset circuit provides a minimum power-on-reset delay sufficient to accomplish this initialization. For fast $V_{DD}$ supply rise times, the MAXQ device will, at a minimum, be held in reset for the power-on reset delay when initially powered up. For slow $V_{DD}$ supply rise times, the MAXQ device will be held in reset until $V_{DD}$ is above the power-on-reset voltage threshold. The minimum POR delay and POR voltage threshold can differ depending upon MAXQ device. Refer to the device data sheet(s) for specifics.

The PMME bit may not be set to 1 if any potential switchback source is active. Attempts to set the PMME bit under these conditions result in a no-op.

#### 2.9.2.1 Switchback

When Power Management Mode is active, the MAXQ operates at a reduced clock rate. Although execution continues as normal, peripherals that base their timing on the system clock such as the UART module and the SPI module may be unable to operate normally or at a high enough speed for proper application response. Additionally, interrupt latency is greatly increased.

The Switchback feature is used to allow a processor running under Power Management Mode to switch back to normal mode quickly under certain conditions that require rapid response. Switchback is enabled by setting the SWB bit to 1. If Switchback is enabled, a processor running under Power Management Mode automatically clears the PMME bit to 0 and returns to normal mode when any of the following conditions occur:

- An external interrupt condition occurs on an INTx pin and the corresponding external interrupt is enabled.
- An active-low transition occurs on the UART serial receive-input line (modes 1, 2, and 3) and data reception is enabled.
- The SBUF register is written to send an outgoing byte through the UART and transmission is enabled.
- The SPIB register is written in master mode (STBY = 1) to send an outgoing character through the SPI module and transmission is enabled.
- The SPI module's $\overline{\text{SSEL}}$ signal is asserted in slave mode.
- Time-of-Day and Subsecond interval alarms from the RTC when enabled.
- Active debug mode is entered either by break point match or issuance of the 'Debug' command from background mode.

### 2.9.3 Stop Mode

When the MAXQ is in Stop Mode, the CPU system clock is stopped, and all processing activity is halted. All on-chip peripherals requiring the system clock are also stopped. Power consumption in Stop Mode is at the lowest possible level and is basically limited to static leakage current.

Stop Mode is entered by setting the STOP bit to 1. The processor enters Stop Mode immediately once the instruction that sets the STOP bit is executed. The MAXQ exits Stop Mode when any of the following conditions occur:

- An external interrupt condition occurs on one of the INTx pins and the corresponding external interrupt is enabled. After the interrupt returns, execution resumes after the stop point.
- An external reset signal is applied to the $\overline{\text{RST}}$ pin. After the reset signal is removed, execution resumes at 8000h as it would after any reset state.

In some MAXQ devices, the brownout voltage detection circuitry can be disabled during Stop Mode, so a power-fail condition does not cause a reset as it would under normal conditions. Once the processor exits Stop Mode, it resumes execution as follows:

- If the RGSL bit is set to 0, the clock source selected by the XT/$\overline{\text{RC}}$ bit is enabled so that it may warm up/stabilize. During the warmup period, the internal ring oscillator may be used for execution. The clock source switches from the ring oscillator to the XT/$\overline{\text{RC}}$ source automatically once the warmup completes. The RGMD bit can be read by the processor to determine when the switch from the ring oscillator to the XT/$\overline{\text{RC}}$ source has occurred.
- If the RGSL bit is set to 1, the internal ring oscillator will be used to resume execution and the XT/$\overline{\text{RC}}$ selected clock source will remain disabled.

# SECTION 3: PROGRAMMING

The following section provides a programming overview of the MAXQ. For full details on the instruction set, as well as System Register and Peripheral Register detailed bit descriptions, see the appropriate sections in this user's guide.

## 3.1 Addressing Modes

The instruction set for the MAXQ provides three different addressing modes: direct, indirect, and immediate.

The direct addressing mode can be used to specify either source or destination registers, such as:

```
move   A[0], A[1]          ; copy accumulator 1 to accumulator 0
push   A[0]                ; push accumulator 0 on the stack
add    A[1]                ; add accumulator 1 to the active accumulator
```

Direct addressing is also used to specify addressable bits within registers.

```
move   C, Acc.0            ; copy bit zero of the active accumulator
                           ;   to the carry flag
move   PO0.3, #1           ; set bit three of port 0 Output register
```

Indirect addressing, in which a register contains a source or destination address, is used only in a few cases.

```
move   @DP[0], A[0]        ; copy accumulator 0 to the data memory
                           ;   location pointed to by data pointer 0
move   A[0], @SP--         ; where @SP-- is used to pop the data pointed to
                           ;   by the stack pointer register
```

Immediate addressing is used to provide values to be directly loaded into registers or used as operands.

```
move   A[0], #10h          ; set accumulator 1 to 10h/16d
```

## 3.2 Prefixing Operations

All instructions on the MAXQ are 16 bits long and execute in a single cycle. However, some operations require more data than can be specified in a single cycle or require that high-order register-index bits be set to achieve the desired transfer. In these cases, the prefix register module PFX is loaded with temporary data and/or required register index bits to be used by the following instruction. The PFX module only holds loaded data for a single cycle before it clears to zero.

Instruction prefixing is required for the following operations, which effectively makes them two-cycle operations:

- When providing a 16-bit immediate value for an operation (e.g., loading a 16-bit register, ALU operation, supplying an absolute program branch destination), the PFX module must be loaded in the previous cycle with the high byte of the 16-bit immediate value unless that high byte is zero. One exception to this rule is when supplying an absolute branch destination to 00xxh. In this case, PFX still must be written with 00h. Otherwise, the branch instruction would be considered a relative one instead of the desired absolute branch.
- When selecting registers with indexes greater than 07h within a module as destinations for a transfer or registers with indexes greater than 0Fh within a module as sources, the PFX[n] register must be loaded in the previous cycle. This can be combined with the previous item.

Generally, prefixing operations can be inserted automatically by the assembler as needed, so that (for example)

```
move   DP[0], #1234h
```

actually assembles as

```
move   PFX[0], #12h
move   DP[0], #34h
```

However, the operation

```
move   DP[0], #0055h
```

does not require a prefixing operation even though the register DP[0] is 16-bit. This is because the prefix value defaults to zero, so the line

```
move   PFX[0], #00h
```

is not required.

## 3.3 Reading and Writing Registers

All functions in the MAXQ are accessed through registers, either directly or indirectly. This section discusses loading registers with immediate values and transferring values between registers of the same size and different sizes.

### 3.3.1 Loading an 8-Bit Register With an Immediate Value

Any writeable 8-bit register with a sub-index from 0h to 7h within its module can be loaded with an immediate value in a single cycle using the MOVE instruction.

```
move  AP, #05h            ; load accumulator pointer register with 5 hex
```

Writeable 8-bit registers with sub-indexes 8h and higher can be loaded with an immediate value using MOVE as well, but an additional cycle is required to set the prefix value for the destination.

```
move  WDCN, #33h          ; assembles to:  move PFX[2], #00h
                          ;                 move (WDCN-80h), #33h
```

### 3.3.2 Loading a 16-Bit Register With a 16-Bit Immediate Value

Any writeable 16-bit register with a sub-index from 0h to 07h can be loaded with an immediate value in a single cycle if the high byte of that immediate value is zero.

```
move  LC[0], #0010h       ; prefix defaults to zero for high byte
```

If the high byte of that immediate value is not zero or if the 16-bit destination sub-index is greater than 7h, an extra cycle is required to load the prefix value for the high byte and/or the high-order register index bits.

```
                          ; high byte <> #00h
move  LC[0], #0110h       ; assembles to:  move PFX[0], #01h
                          ;                 move LC[0], #10h
                          ; destination sub-index > 7h
move  A[8], #0034h        ; assembles to:  move PFX[2], #00h
                          ;                 move (A[8]-80h), #34h
```

### 3.3.3 Moving Values Between Registers of the Same Size

Moving data between same-size registers can be done in a single-cycle MOVE if the destination register's index is from 0h to 7h and the source register index is between 0h and Fh.

```
move  A[0], A[8]          ; copy accumulator 8 to accumulator 0
move  LC[0], LC[1]        ; copy loop counter 1 to loop counter 0
```

If the destination register's index is greater than 7h or if the source register index is greater than Fh, prefixing is required.

```
move  A[15], A[0]         ; assembles to:  move PFX[2], #00h
                          ;                 move (A[15]-80h), A[0]
```

### 3.3.4 Moving Values Between Registers of Different Sizes

Before covering some transfer scenarios that might arise, a special register must be introduced that will be used in many of these cases. The 16-bit General Register (GR) is expressly provided for performing byte singulation of 16-bit words. The high and low bytes of GR are individually accessible in the GRH and GRL registers respectively. A read-only GRS register makes a byte-swapped version of GR accessible and the GRXL register provides a sign-extended version of GRL.

**8-bit destination ← low byte (16-bit source)**

The simplest transfer possibility would be loading an 8-bit register with the low byte of a 16-bit register. This transfer does not require use of GR and requires a prefix only if the destination or source register are outside of the single cycle write or read regions, 0–7h and 0–Fh, respectively.

```
move  OFFS, LC[0]         ; copy the low byte of LC[0]  to the OFFS register
move  IMR, @DP[1]         ; copy the low byte @DP[1]  to the IMR register
move  WDCN, LC[0]         ; assembles to: move PFX[2], #00h
                          ;                move (WDCON-80h), LC[0]
```

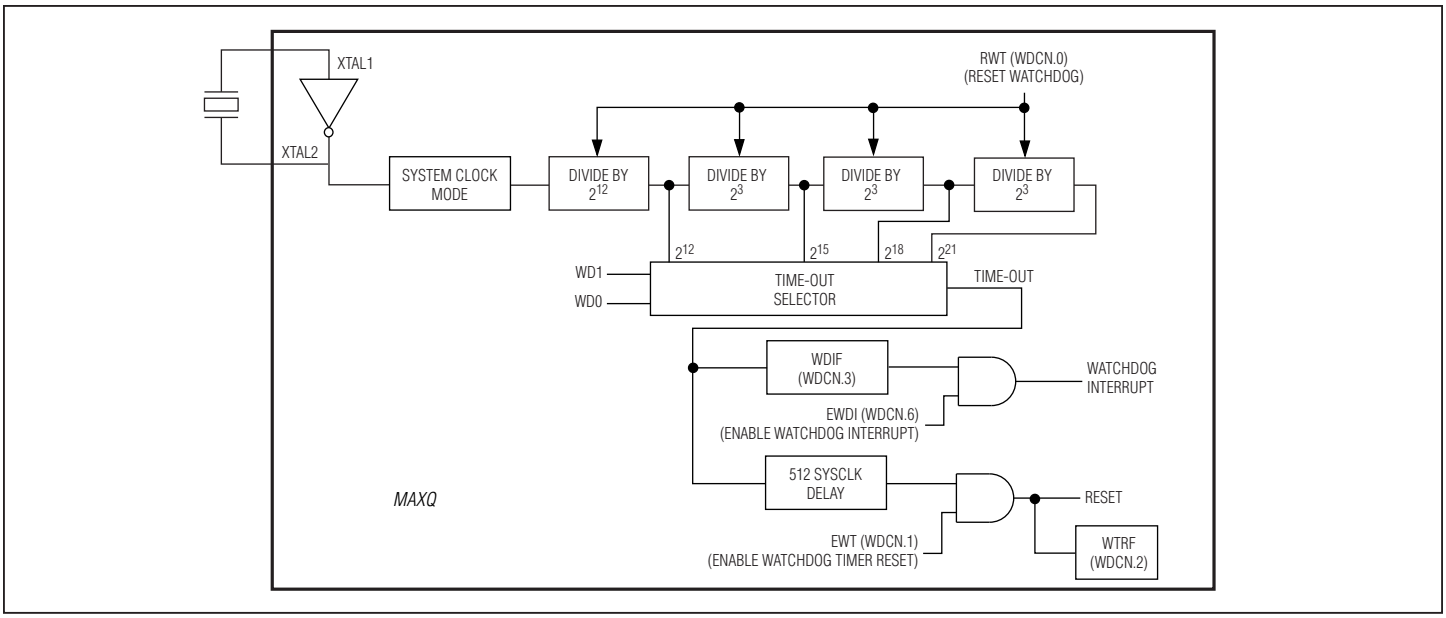



*Figure 3-1. Watchdog Timer Block Diagram*

If the timeout is reached without RWT being set, hardware will generate a Watchdog interrupt if the interrupt source has been enabled. If no further action is taken to prevent a Watchdog reset, in the 512 system clock cycles following the timeout, hardware has the ability to reset the CPU if EWT = 1. When the reset occurs, the Watchdog Timer Reset Flag (WTRF = WDCN.2) will automatically be set to indicate the cause of the reset, however software must clear this bit manually.

The Watchdog Interrupt is also available for applications that do not need a true Watchdog Reset but simply a very long timer. The interrupt is enabled using the Enable Watchdog Timer Interrupt (EWDI = WDCN.6) bit. When the timeout occurs, the Watchdog Timer will set the WDIF bit (WDCN.3), and an interrupt will occur if the interrupt global enable (IGE = IC.0) and system interrupt mask (IMS = IMR.7) are set and the interrupt in service (INS) bit is clear. Note that WDIF is set 512 clocks before a potential Watchdog Reset. The Watchdog Interrupt Flag will indicate the source of the interrupt, and must be cleared by software.

Using the Watchdog Interrupt during software development can allow the user to select ideal watchdog reset locations. Code is first developed without enabling the Watchdog Interrupt or Reset functions. Once the program is complete, the Watchdog Interrupt function is enabled to identify the required locations in code to set the RWT (WDCN.0) bit. Incrementally adding instructions to reset the Watchdog Timer prior to each address location (identified by the Watchdog Interrupt) will allow the code to eventually run without receiving a Watchdog Interrupt. At this point the Watchdog Timer Reset can be enabled without the potential of generating unwanted resets. At the same time the Watchdog Interrupt may also be disabled. Proper use of the Watchdog Interrupt with the Watchdog Reset allows interrupt software to survey the system for errant conditions.

When using the Watchdog Timer as a system monitor, the Watchdog Reset function should be used. If the Interrupt function were used, the purpose of the watchdog would be defeated. For example, assume the system is executing errant code prior to the Watchdog Interrupt. The interrupt would temporarily force the system back into control by vectoring the CPU to the interrupt service routine. Restarting the Watchdog and exiting by an RETI or RET, would return the processor to the lost position prior to the interrupt. By using the Watchdog Reset function, the processor is restarted from the beginning of the program, and therefore placed into a known state.

The Watchdog timeout selection is made using bits WD1 (WDCN.5) and WD0 (WDCN.4). The Watchdog has four timeout selections based on the system clock frequency as shown in the figure. Since the timeout is a function of the system clock, the actual timeout interval is dependent on both the crystal frequency and the system clock mode selection. Shown below is a summary of the selectable Watchdog timeout intervals for the various system clock modes and WD1:0 control bit settings. The Watchdog Reset, if enabled, is always scheduled to occur 512 system clocks following the timeout. Watchdog generated resets will last for 4 system clock cycles.

## 4.9 Watchdog Control Register (WDCN, 8h[Fh])

Initialization: Bits 5, 4, 3 and 0 are cleared to 0 on all forms of reset; for others, see individual bit descriptions.
Access: Unrestricted direct read/write access.

<table>
<tr><th>BIT</th><th>FUNCTION</th></tr>
<tr><td>WDCN.0 (RWT)</td><td>Reset Watchdog Timer. Setting this bit to 1 resets the watchdog timer count. If watchdog interrupt and/or reset modes are enabled, the software must set this bit to 1 before the watchdog timer elapses to prevent an interrupt or reset from occurring. This bit always returns 0 when read.</td></tr>
<tr><td>WDCN.1 (EWT)</td><td>Enable Watchdog Timer Reset. If this bit is set to 1 when the watchdog timer elapses, the watchdog resets the processor 512 system clock cycles later unless action is taken to disable the reset event. Clearing this bit to 0 prevents a watchdog reset from occurring but does not stop the watchdog timer or prevent watchdog interrupts from occurring if EWDI = 1. If EWT = 0 and EWDI = 0, the watchdog timer will be stopped. If the watchdog timer is stopped (EWT = 0 and EWDI = 0), setting the EWT bit will reset the watchdog interval and reset counter, and enable the watchdog timer. This bit is cleared on Power-on reset and is unaffected by other forms of reset.</td></tr>
<tr><td>WDCN.2 (WTRF)</td><td>Watchdog Timer Reset Flag. This bit is set to 1 when the watchdog resets the processor. Software can check this bit following a reset to determine if the watchdog was the source of the reset. Setting this bit to 1 in software will not cause a watchdog reset. This bit is cleared by Power-on reset only and is unaffected by other forms of reset. It should also be cleared by software following any reset so that the source of the next reset can be correctly determined by software. This bit is only set to 1 when a watchdog reset actually occurs, so if EWT is cleared to 0 when the watchdog timer elapses, this bit will not be set.</td></tr>
<tr><td>WDCN.3 (WDIF)</td><td>Watchdog Interrupt Flag. This bit will be set to 1 when the watchdog timer interval has elapsed or can be set to 1 by user software. When WDIF = 1, an interrupt request will occur if the watchdog interrupt has been enabled (EWDI = 1) and not otherwise masked or prevented by an interrupt already in service (i.e., IGE = 1, IMS = 1, and INS = 0 must be true for the interrupt to occur). This bit should be cleared by software before exiting the interrupt service routine to avoid repeated interrupts. Furthermore, if the watchdog reset has been enabled (EWT = 1), a reset is scheduled to occur 512 system clock cycles following setting of the WDIF bit.</td></tr>
<tr><td>WDCN.4 (WD0);<br>WDCN.5 (WD1)</td><td>Watchdog Timer Mode Select Bit 0; Watchdog Timer Mode Select Bit 1. These bits determine the watchdog interval or the length of time between resetting of watchdog timer and the watchdog generated interrupt in terms of system clocks. Modifying the watchdog interval via the WD1:0 bits will automatically reset the watchdog timer unless the 512 system clock reset counter is already in progress, in which case, changing the WD1:0 bits will not effect the Watchdog timer or reset counter.
<table>
<tr><th>WD1</th><th>WD0</th><th>CLOCKS UNTIL INTERRUPT</th><th>CLOCKS UNTIL RESET</th></tr>
<tr><td>0</td><td>0</td><td>$2^{12}$</td><td>$2^{12}$ + 512</td></tr>
<tr><td>0</td><td>1</td><td>$2^{15}$</td><td>$2^{15}$ + 512</td></tr>
<tr><td>1</td><td>0</td><td>$2^{18}$</td><td>$2^{18}$ + 512</td></tr>
<tr><td>1</td><td>1</td><td>$2^{21}$</td><td>$2^{21}$ + 512</td></tr>
</table></td></tr>
<tr><td>WDCN.6 (EWDI)</td><td>Watchdog Interrupt Enable. If this bit is set to 1, an interrupt request can be generated when the WDIF bit is set to 1 by any means. If this bit is cleared to 0, no interrupt will occur when WDIF is set to 1, however, it does not stop the watchdog timer or prevent watchdog resets from occurring if EWT = 1. If EWT = 0 and EWDI = 0, the watchdog timer will be stopped. If the watchdog timer is stopped (EWT = 0 and EWDI = 0), setting the EWDI bit will reset the watchdog interval and reset counter, and enable the watchdog timer. This bit is cleared to 0 by power-on reset and is unaffected by other forms of reset.</td></tr>
<tr><td>WDCN.7 (POR)</td><td>Power-On Reset Flag. This bit is set to 1 whenever a power-on/brownout reset occurs. It is unaffected by other forms of reset. This bit can be checked by software following a reset to determine if a power-on/brownout reset occurred. It should always be cleared by software following a reset to ensure that the sources of following resets can be determined correctly.</td></tr>
</table>

# SECTION 6: GENERAL-PURPOSE I/O MODULE

The General-Purpose I/O Module (GPIO) for the MAXQ supports multiple 8-bit port types, each having different I/O characteristics. From a software perspective, each port appears as a group of Peripheral Registers with unique addresses. The exact quantity and type of ports provided by the GPIO Module is product-dependent. Each of the four different types of I/O ports are described.

## 6.1 I/O Port: Type A

The Type A port can be used as a bidirectional I/O port. A port consists of eight general-purpose input/output pins and all the registers needed to control and configure them. Each pin is independently controllable. Up to six pins of each type A port can be configured as external interrupts. Each interrupt function is supported by its own interrupt flag, and each can be independently enabled.

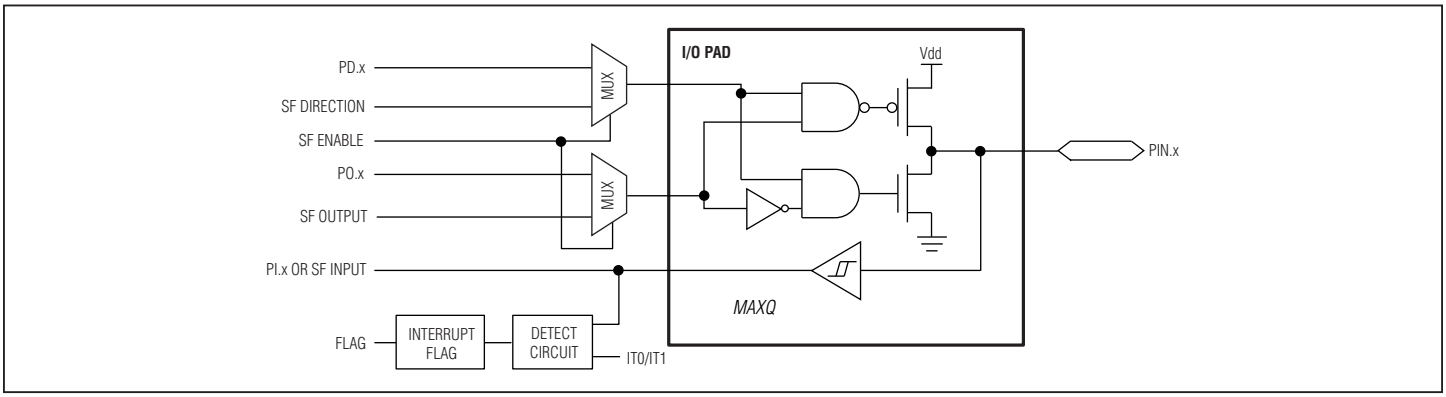



*Figure 6-1. Type A Port Pin Schematic*

## 6.2 I/O Port: Type B

The Type B port can also be used as a bidirectional I/O port. The Type B port consists of eight general-purpose input/output pins and three registers needed to control and configure them. Each pin is independently controllable. Type B port pins are intended to support secondary special functions. The special functions associated with these port pins are generally implemented in peripheral modules to the MAXQ CPU, which can be enabled, controlled, and monitored using dedicated Peripheral Registers.

Enabling the special function automatically converts the pin to that function. The I/O drive characteristics for these pins are the same no matter whether the pin is configured for general-purpose I/O or whether it is being used for the special function.

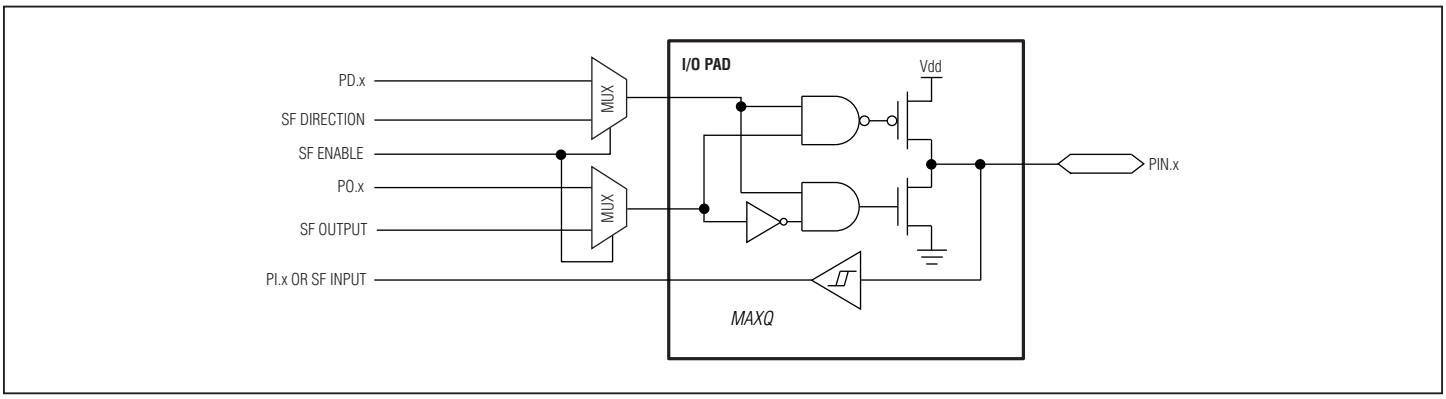



*Figure 6-2. Type B Port Pin Schematic*

### 9.2.1 16-Bit Timer: Auto-Reload/Compare

The 16-bit auto-reload/compare mode for Timer 2 is in effect when the Timer 2 mode select bit (T2MD) is cleared and the capture/compare function definition bits are both cleared (CCF[1:0] = 00b). The Timer 2 value is contained in the T2V register. The Timer 2 run control bit (TR2) starts and stops the 16-bit Timer. The input clock for 16-bit Timer 2 is defined as the system clock divided by the ratio specified by the T2DIV[2:0] prescale bits. The Timer begins counting from the value contained in the T2L:T2H register pair until overflowing. When an overflow occurs, the reload value (T2RH:T2RL) is reloaded instead of the x0000h state. The Timer 2 overflow flag (TF2) is set every time that an overflow condition (T2V = 0xFFFFh) is detected. If Timer 2 interrupts have been enabled (ET2 = 1), the TF2 flag can generate an interrupt request. When operating in compare mode, the capture/compare registers (T2CH:T2CL) are compared versus the Timer 2 value registers. Whenever a compare match occurs, the capture/compare status flag (TCC2) is set. If Timer 2 interrupts have been enabled (ET2 = 1), this event is capable of generating an interrupt request. If the capture/compare register is set to a value outside the Timer 2 counting range, a compare match is not signaled and the TCC2 flag is not set. Internally, a Timer 2 output clock is generated, which toggles on the cycle following any compare match or overflow, unless the compare match value has been set equal to the overflow condition, in which case, only one toggle will occur. This clock may be sourced by certain peripherals and/or may be output on one or more pins as permitted by the microcontroller.

#### 9.2.1.1 Output Enable (PWM Out)

The Output Enable bits (T2OE[1:0]) enable the Timer 2 output clock to be presented on the pins associated with the respective bits. If Timer 2 has a single I/O pin, the T2OE[0] bit is associated with the T2P pin and the T2OE[1] bit is not implemented (as it would serve no purpose).

#### 9.2.1.2 Polarity Control

The Polarity Control bits (T2POL[1:0]) can be used to modify (invert) the enabled clock outputs to the pin(s). The enabled clock outputs (defined by T2OE[1:0]) will toggle on each compare match or overflow. The T2POL[1:0] bits are logically XORed with the Timer 2 output signal, therefore setting a given T2POL[x] bit will result in a high starting state. The T2POL[n] bit can be changed at any time, however the assigned T2POL[n] state will take effect on the external pin only when the corresponding T2OE[n] bit is changed from 0 to 1. When generating PWM output, please note that changing the compare match register can result in a perceived duty cycle inversion if a compare match is missed or multiple compare matches occur during the reload to overflow counting.

#### 9.2.1.3 Gated

To use the T2P pin as a timer-input clock gate, the T2OE[0] bit must be cleared to 0 and the G2EN bit must be set to 1. When T2OE[0] = 1, the G2EN bit setting has no effect. When T2OE[0] is cleared to 0, the respective polarity control bit is used to modify the polarity of the input signal to the Timer. In the gated mode, the Timer 2 input clock is gated anytime that the external signal matches the state of the T2POL[0] bit. This means that the default clock gating condition for the T2P pin is logic low (since T2POL[0] = 0 default). Setting T2POL[0] = 1 results in the Timer 2 input clock being gated when the T2P pin is high. Note if multiple pins are allocated for Timer 2 (i.e., T2P, T2PB), the primary pin can be used for clock gating, while the secondary pin can be used to output the gated PWM output signal (if T2OE[1] = 1).

#### 9.2.1.4 Single Shot (and Gating)

When operating in 16-bit compare mode, the single-shot is used to automate the generation of single pulses under software control or in response to an external signal (single-shot gated). To generate single-shot output pulses solely under software control, the G2EN bit should be cleared to 0, the output enables and polarity controls should be configured as desired, and the single-shot bit should be set to 1. Writing the single-shot bit effectively overrides the TR2 = 0 condition until Timer 2 overflow/reload occurs. The single-shot bit is automatically cleared once the overflow/reload occurs.

Writing SS2 and TR2 = 1 at the same time still causes the SS2 bit to stay in effect until an overflow/reload occurs; however, since TR2 was also written to 1, the specified PWM output continues even after SS2 becomes clear.

If two pins are available for the Timer 2 implementation, an additional mode is supported: single-shot gated. Single-shot gated requires that the T2P pin be used as an input (T2OE[0] = 0). It also requires that G2EN = 1, thus differentiating it from the software controlled single-shot mode on the second output pin. If G2EN is enabled and SS2 is written to 1, the gating condition must first be removed for the single-shot enabled output to occur on the pin. When the clock gate is removed, the single-shot output occurs. Just as described, the SS2 bit = 1 state remains in effect until overflow/reload. Note that this makes it possible for the single-shot to span multiple gated/non-gated intervals. Once the SS2 = 1 conditions completes, if TR2 = 1, the gated PWM mode is in effect. Otherwise (TR2 = 0), Timer 2 is stopped.

## 9.4 Timer/Counter 2 Peripheral Registers

### 9.4.1 Timer/Counter 2 Configuration Register (T2CFG)

| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | T2CI | T2DIV2 | T2DIV1 | T2DIV0 | T2MD | CCF1 | CCF0 | C/$\overline{T2}$ |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Access | rw | rw | rw | rw | rw | rw | rw | rw |

*r = read, w = write*

**Bit 7: Timer 2 Clock Input Select Bit (T2CI).** Setting this bit enables an alternate input clock source to the Timer 2 block. The alternate input clock selection is the 32kHz clock. The alternate input clock must be sampled by the system clock, which requires that the system clock be at least 4 x 32kHz for proper operation unless the system clock is also source from the 32kHz crystal.

**Bits 6 to 4: Timer 2 Clock Divide 2:0 Bits (T2DIV[2:0]).** These three bits select the divide ratio for the timer clock-input clock (as a function of the system clock) when operating in timer mode with T2CI = 0.

| T2DIV2 | T2DIV1 | T2DIV0 | DIVIDE RATIO |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 2 |
| 0 | 1 | 0 | 4 |
| 0 | 1 | 1 | 8 |
| 1 | 0 | 0 | 16 |
| 1 | 0 | 1 | 32 |
| 1 | 1 | 0 | 64 |
| 1 | 1 | 1 | 128 |

**Bit 3: Timer 2 Mode Select (T2MD).** This bit enables the dual 8-bit mode of operation. The default-reset state is 0, which selects the 16-bit mode of operation. When the dual 8-bit mode is established, the primary timer/counter (T2H) carries all of the counter/capture functionality while the secondary 8-bit timer (T2L) must operate in timer compare mode, sourcing the defined internal clock.

0 = 16-bit mode (default)

1 = dual 8-bit mode

**Bits 2 to 1: Capture/Compare Function Select Bits (CCF[1:0]).** These bits, in conjunction with the C/$\overline{T2}$ bit, select the basic operating mode of Timer 2. In the dual 8-bit mode of operation (T2MD = 1), the T2L timer only operates in compare mode.

| CCF1 | CCF0 | EDGE(S) | C/$\overline{T2}$ = 0 (TIMER MODE) | C/$\overline{T2}$ = 1 (COUNTER MODE) |
|---|---|---|---|---|
| 0 | 0 | None | Compare Mode | Disabled |
| 0 | 1 | Rising | Capture/Reload | Counter |
| 1 | 0 | Falling | Capture/Reload | Counter |
| 0 | 1 | Rising and Falling | Capture/Reload | Counter |

**Bit 0: Counter/Timer Select (C/$\overline{T2}$).** This bit enables/disables the edge counter mode of operation for the 16-bit counter (T2H:T2L) or the 8-bit counter (T2H) when the dual 8-bit mode of operation is enabled (T2MD = 1). The edge for counting (rising/falling/both) is defined by the CCF[1:0] bits.

0 = timer mode

1 = counter mode

*Compare Mode:*

If SS2 is written to 1 while in compare mode, one cycle of the defined waveform (reload to overflow) is output to the T2P, T2PB pins as prescribed by T2POL[1:0] and T2OE[1:0] controls. The only time that this does not immediately occur is when a gating condition is also defined. If a gating condition is defined, the single-shot cycle cannot occur until the gating condition is removed. If the specified non-gated level is already in effect, the singleshot period will start. The gated single-shot output is not supported in dual 8-bit mode.

*Capture Mode:*

If SS2 is written to 1 while in capture mode, the timer is halted and the single-shot capture cycle does not begin until the edge specified by CCF[1:0] is detected, or the defined gating condition is removed. Once running, the timer continues running (as allowed by the gate condition) until the defined capture single-shot edge is detected. In this way, the SS2 bit can be used to delay the running of a timer until an edge is detected (setting both SS2 and TR2 =1) or override the TR2 = 0 bit setting for one capture cycle (setting only SS2 = 1). When both edges are defined for capture CCF[1:0] = 11b), the T2POL[0] bit serves to define the single-shot start/end edge: falling edge if T2POL[0] = 1; rising edge if T2POL[0] = 0. No interrupt flag is set when the starting edge for the single-shot capture cycle is detected. The single-shot capture cycle always ends when the next single shot edge is detected. The start/end edge is defined by T2POL[0]. This bit is intended to automate pulse-width measurement (low or high) and duty cycle/period measurement.

**Bit 0: Gating Enable (G2EN).** This bit enables the external T2P pin to gate the input clock to the 16-bit (T2MD = 0) or highest 8-bit (T2MD = 1) Timer. Gating uses T2P as an input, thus it can only be used when T2OE0 = 0 and C/$\overline{T2}$ = 0. Gating is not possible on the low 8-bit timer (T2L) when Timer 2 is operated in dual 8-bit mode. Gating is not supported for counter mode operation (C/$\overline{T2}$ = 1). The G2EN bit serves a different purpose when capture and reload have been defined for both edges (CCF[1:0] = 11b and CPRL2 = 1). For this special case, setting G2EN = 1 allows the T2POL0 bit to specify which edge does not cause a reload. If T2POL0 is 0, no reload on the falling edge; if T2POL0 is 1, no reload on the rising edge.

0 = gating disabled

1 = gating enabled

### 9.4.3 Timer/Counter 2 Control Register B (T2CNB)

| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | ET2L | T2OE1 | T2POL1 | — | TF2 | TF2L | TCC2 | TC2L |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Access | rw | rw | rw | r | rw | rw | rw | rw |

*r = read, w = write*

**Bit 7: Enable Timer 2 Low Interrupts (ET2L).** This bit serves as the local enable for Timer 2 Low interrupt sources that fall under the TF2L and TC2L interrupt flags.

**Bit 6: Timer 2 Output Enable 1 (T2OE1).** See table given under T2CNA.5 description. The T2OE1 bit is not implemented for single pin versions of Timer 2.

**Bit 5: Timer 2 Polarity Select 1 (T2POL1).** When the T2B output is enabled (T2OE1 = 1), this bit selects the starting logic level for the alternate pin output. The output that is driven on the T2PB pin can be derived from the 16-bit Timer 2 or the 8-Timer (T2L) depending upon whether operating in the 16-bit mode or the dual 8-bit mode. The T2POL1 bit can be modified anytime, but takes effect on the external pin when T2OE1 is changed from 0 to 1.

**Bit 3: Timer 2 Overflow Flag (TF2).** This flag becomes set anytime there is an overflow of the full 16-bit T2V timer/counter (when T2MD = 0) or an overflow of the 8-bit T2H timer/counter when the dual 8-bit mode of operation is selected (T2MD = 1).

**Bit 2: Timer 2 Low Overflow Flag (TF2L).** This flag is meaningful only when in the dual 8-bit mode of operation (T2MD = 1) and becomes set whenever there is an overflow of the T2L 8-bit timer.

**Bit 1: Timer 2 Capture/Compare Flag (TCC2).** This flag is set on any compare match between the Timer 2 value and compare register (T2V = T2C or T2H = T2CH, respectively, for 16-bit and 8-bit compare modes) or when a capture event is initiated by an external edge.

**Bit 0: Timer 2 Low Compare Flag (TC2L).** This flag is meaningful only for the dual 8-bit mode of operation (T2MD = 1) and becomes set only when a compare match occurs between T2CL and T2L. Timer 2 Low does not have an associated capture function.

# SECTION 14: REAL-TIME CLOCK MODULE

This section contains the following information:

14.1 RTC Alarm Functions . . . . . 14-3
14.1.1 Time-of-Day Alarm . . . . . 14-3
14.1.2 Sub-Second Alarm . . . . . 14-3
14.1.3 System Wakeup by Time-of-Day or Sub-Second Interval Alarm . . . . . 14-3
14.2 RTC Trim Function . . . . . 14-3
14.3 RTC Register Access . . . . . 14-5
14.3.1 Busy Bit Write Signaling . . . . . 14-5
14.3.2 Ready Bit Read Signaling . . . . . 14-5
14.3.3 RTC Count Register Access . . . . . 14-5
14.3.4 RTC Alarm Register Access . . . . . 14-5
14.3.5 RTC Trim Register Access . . . . . 14-5
14.4 RTC Peripheral Registers . . . . . 14-6
14.4.1 RTC Control Register (RCNT) . . . . . 14-6
14.4.2 RTC Seconds High Register (RTSH) . . . . . 14-7
14.4.3 RTC Seconds Low Register (RTSL) . . . . . 14-8
14.4.4 RTC Sub-Seconds Register (RTSS) . . . . . 14-8
14.4.5 RTC Alarm Seconds High Register (RASH) . . . . . 14-9
14.4.6 RTC Alarm Seconds Low Register (RASL) . . . . . 14-9
14.4.7 RTC Sub-Second Alarm Register (RSSA) . . . . . 14-10
14.4.8 RTC Trim Register (RTRM) . . . . . 14-10

## LIST OF FIGURES

Figure 14-1. RTC Functional Block Diagram . . . . . 14-2
Figure 14-2. RTC Digital-Trim Facility Block Diagram . . . . . 14-4
Figure 14-3. Digital Trim Pulse Calibration Diagram . . . . . 14-4

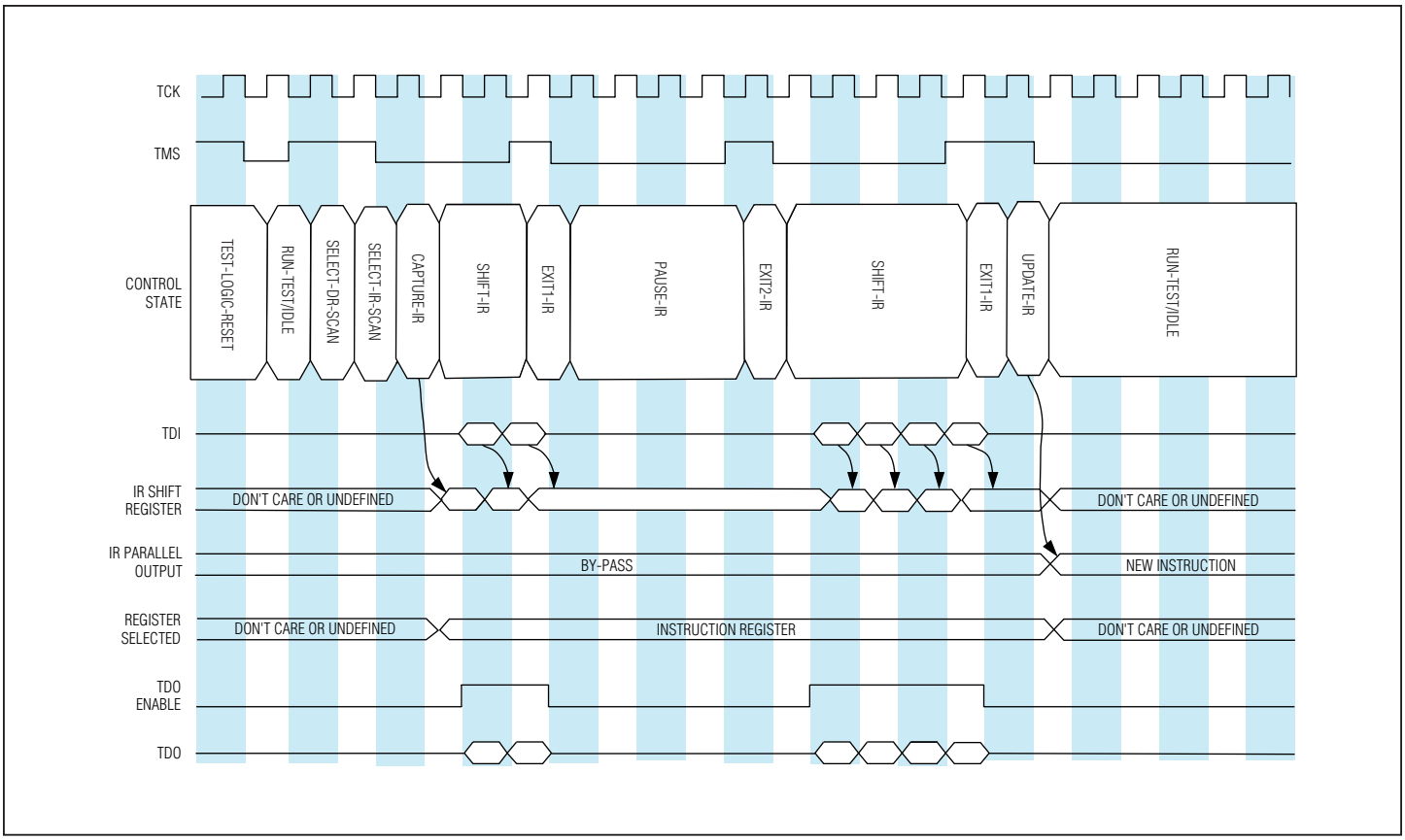



*Figure 15-3. TAP Controller Debug Mode IR-Scan Example*

### 16.1.1.7 Breakpoint 5 Register (BP5) (REGE = 0)

| Bit # | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Name (REGE = 0) | BP5.15 | BP5.14 | BP5.13 | BP5.12 | BP5.11 | BP5.10 | BP5.9 | BP5.8 |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Access | s | s | s | s | s | s | s | s* |

| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name (REGE = 0) | BP5.7 | BP5.6 | BP5.5 | BP5.4 | BP5.3 | BP5.2 | BP5.1 | BP5.0 |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Access | s* | s* | s* | s* | s** | s** | s** | s** |

*s = special, * = register index within module {0-31), ** = module specifier 3:0 {0-15}*

**Bits 15 to 0: Breakpoint 5 (BP5.[15:0]).** This register is accessible only via background mode read/write commands.

(REGE = 0) This register serves as one of the two data memory address breakpoints. When DME is set in background mode, the debug engine will monitor the data memory address bus activity while the CPU is executing the user program. If an address match is detected, a break occurs, allowing the debug engine to take over control of the CPU and enter debug mode.

### 16.1.1.8 Breakpoint 5 Register (BP5) (REGE = 1)

| Bit # | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| Name (REGE = 1) | — | — | — | — | — | — | — | BP5.8 |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Access | s | s | s | s | s | s | s | s* |

| Bit # | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name (REGE = 1) | BP5.7 | BP5.6 | BP5.5 | BP5.4 | BP5.3 | BP5.2 | BP5.1 | BP5.0 |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Access | s* | s* | s* | s* | s** | s** | s** | s** |

*s = special, * = register index within module {0-31), ** = module specifier 3:0 {0-15}*

**Bits 15 to 9: Reserved**

**Bits 8 to 0: Breakpoint 5 (BP5.[8:0]).** This register is accessible only via background mode read/write commands.

(REGE = 1) This register serves as one of the two register breakpoints. A break occurs when two conditions are met:

Condition 1: The destination register address for the executed instruction matches with the specified module and index.

Condition 2: The bit pattern written to the destination register matches those bits specified for comparison by the ICDD data register and ICDA mask register. Only those ICDD data bits with their corresponding ICDA mask bits will be compared. When all bits in the ICDA register are cleared, Condition 2 becomes a don't care.

### 16.1.2 Using Breakpoints

All breakpoint registers (BP0-BP5) default to the FFFFh state on power-on reset or when the Test-Logic-Reset TAP state is entered. The breakpoint registers are accessible only with Background mode read/write commands issued over the TAP communication link. The breakpoint registers are not read/write accessible to the CPU.

Setting the Debug Mode Enable (DME) bit in the ICDC register to logic 1 enables all six breakpoint registers for breakpoint match comparison. The state of the Break-On Register Enable (REGE) bit in the ICDC register determines whether the BP4 and BP5 breakpoints should be used as data memory address breakpoints (REGE = 0) or as register breakpoints (REGE = 1).

When using the register matching breakpoints, it is important to realize that Debug mode operations (e.g., read data memory, write data memory, etc.) require use of ICDA and ICDD for passing of information between the host and MAXQ microcontroller ROM routines. It is advised that these registers be saved and restored or be reconfigured before returning to the background mode if register breakpoints are to remain enabled.

When a breakpoint match occurs, the debug engine forces a break and the MAXQ microcontroller enters Debug Mode. If a breakpoint match occurs on an instruction that activates the PFX register, the break is held off until the prefixed operation completes. The host can assess whether Debug mode has been entered by monitoring the status bits of the 10-bit word shifted out of the TDO pin. The status bits will change from the Non-debug (00b) state associated with background mode to the Debug-Idle (01b) state when Debug Mode is entered. Debug mode can also be manually invoked by host issuance of the 'Debug' background command.

## 16.2 Debug Mode

There are two ways to enter the Debug Mode from Background Mode:

- Issuance of the Debug command directly by the host via the TAP communication port, or
- Breakpoint matching mechanism.

The host can issue the Debug background command to the debug engine. This direct Debug Mode entry is indeterministic. The response time varies dependent on system conditions when the command is issued. The breakpoint mechanism provides a more controllable response, but requires that the breakpoints be initially configured in Background mode. No matter the method of entry, the debug engine takes control of the CPU in the same manner. Debug mode entry is similar to the state machine flow of an interrupt except that the target execution address is x8010h which resides in the Utility ROM instead of the address specified by the IV register that is used for interrupts. On debug mode entry, the following actions occur:

1) block the next instruction fetch from program memory

2) push the return address onto the stack

3) set the contents of IP to x8010h

4) clear the IGE bit to 0 to disable interrupt handler if it is not already clear.

5) halt CPU operation

Once in Debug mode, further breakpoint matches or host issuance of the Debug command are treated as no operations and will not disturb debug engine operation. Entering debug mode also stops the clocks to all timers, including the Watchdog Timer. Temporarily disabling these functions allows debug mode operations without disrupting the relationship between the original user program code and hardware timed functions. No interrupt request can be granted since the interrupt handler is also halted as a result of IGE = 0.

**MOVE C, Acc.<b>** — **Move Accumulator Bit to Carry Flag**

**Description:** Replaces the Carry (C) status flag with the specified active accumulator bit.

**Status Flags:** C

**Operation:** C ← Acc.<b>

**Encoding:**

| 15 | | | 0 |
|---|---|---|---|
| 1110 | 1010 | bbbb | 1010 |

**MAXQ010 Example(s):**

```
                        ; Acc = 01h, C=0
MOVE C, Acc.0           ; C =1
```

**MAXQ020 Example(s):**

```
                        ; Acc = 01C0h, C=0
MOVE C, Acc.8           ; C =1
```

**Special Notes:** For the MAXQ10, the accumulator width is only 8 bits. Thus, only bit index encoding ('bbbb') for bits 0 ('0000') through 7 ('0111') is supported.

**MOVE C, src.<b>** — **Move Bit to Carry Flag**

**Description:** Replaces the Carry (C) status flag with the specified source bit src.<b>.

**Status Flags:** C

**Operation:** C ← src.<b>

**Encoding:**

| 15 | | | 0 |
|---|---|---|---|
| fbbb | 0111 | ssss | ssss |

**Example(s):**

```
                        ; M0[0] = FEh; C=1 (assume M0[0] is an 8-bit register)
MOVE C, M0[0].0         ; C=0
```

**MOVE C, #0** — **Clear Carry Flag**

**Description:** Clears the Carry (C) processor status flag.

**Status Flag:** C ← 0

**Operation:** C ← 0

**Encoding:**

| 15 | | | 0 |
|---|---|---|---|
| 1101 | 1010 | 0000 | 1010 |

**Example(s):**

```
                        ; C = 1
MOVE C, #0              ; C ← 0
```

**OR Acc.<b>** **Logical OR Carry Flag with Accumulator Bit**

**Description:** Performs a logical-OR between the Carry (C) status flag and a specified bit of the active accumulator (Acc.<b>) and returns the result to the Carry.

**Status Flags:** C

**Operation:** C ← C OR Acc.<b>

**Encoding:**

| 15 | | | 0 |
|---|---|---|---|
| 1010 | 1010 | bbbb | 1010 |

**MAXQ10 Example(s):**

```
                    ; Acc = 45h, C=0 at start
OR Acc.1            ; Acc.1=0  → C=0
OR Acc.2            ; Acc.2=1  → C=1
```

**MAXQ20 Example(s):**

```
                    ; Acc = 2345h, C=0 at start
OR Acc.1            ; Acc.1=0  → C=0
OR Acc.2            ; Acc.2=1  → C=1
```

**Special Notes:** For the MAXQ10, the accumulator width is only 8 bits. Thus, only bit index encoding ('bbbb') for bits 0 ('0000') through 7 ('0111') is supported.

**POP dst** **Pop Word from the Stack**

**Description:** Pops a single word from the stack (@SP) to the specified dst and decrements the stack pointer (SP).

**Status Flags:** S, Z (if dst = Acc or AP or APC)

C, E (if dst = PSF)

**Operation:** dst ← @ SP--

**Encoding:**

| 15 | | | 0 |
|---|---|---|---|
| 1ddd | dddd | 0000 | 1101 |

**Example(s):**

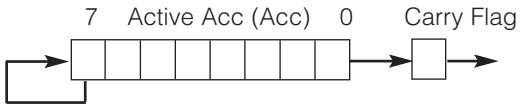
```
                    ; GR ← 1234h
POP GR              ; @DP[0] ← 76h (WBS0=0)
POP @DP[0]          ; @DP[0] ← 0876h (WBS0=1)
```

Stack Data:

| |
|---|
| xxxxh |
| 1234h |
| 0876h |
| xxxxh |
| xxxxh |

← SP (initial)
← SP (after POP GR)
← SP (after POP @DP[0])

**SRA2 Operation:**

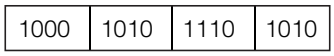



Acc.[5:0] ← Acc.[7:2]

Acc.[7:6] ← Acc.7

C ← Acc.1

**Encoding:**

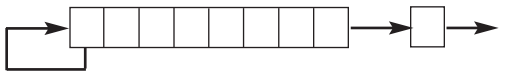



**Example(s):**

```
                        ; Acc = 03h, C=0, Z=0
SRA2                    ; Acc = 00h, C=1, Z=1
```

---

**SRA4 Operation:**

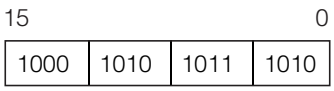



Acc.[3:0] ← Acc.[7:4]

Acc.[7:4] ← Acc.7

C ← Acc.3

**Encoding:**




**Example(s):**

```
                        ; Acc = 98h, C=0, Z=0
SRA4                    ; Acc = F9h, C=1, Z=0
```