TABLE OF CONTENTS

SECTION 1: OVERVIEW

This section contains the following information:

SECTION 1: OVERVIEW

The MAXQ® family of 16-bit reduced instruction set computing (RISC) microcontrollers is targeted toward low-cost, low-power, embedded-application designs. The flexible, modular architecture design used in these microcontrollers allows development of targeted designs for specific applications with minimal effort.

Microcontrollers in the MAXQ family provide many different combinations of program memory, data memory, and peripherals while supporting a common feature set. This shared functionality provides maximum reusability for hardware and software systems developed using these microcontrollers.

1.1 Instruction Set

All MAXQ microcontrollers share a common instruction set, with all instructions a fixed 16 bits in length. A register-based, transport-triggered architecture allows all instructions to be coded as simple transfer operations. All instructions reduce to either writing an immediate value to a destination register or memory location or moving data between registers and/or memory locations.

This simple top-level instruction decoding allows all instructions to be executed in a single cycle. Since all CPU operations are performed on registers only, any new functionality can be added by simply adding new register modules. The simple instruction set also provides maximum flexibility for code optimization by a compiler.

1.2 Harvard Memory Architecture

Program memory, data memory, and register space on MAXQ microcontrollers are separate from one another, and are each accessed by a separate bus. This type of memory architecture (known as a Harvard architecture) has some advantages.

First, the word lengths can be different for different types of memory. Program memory must be 16 bits wide to accommodate the instruction word size, but system and peripheral registers can be 8 bits wide or 16 bits wide as needed. Since data memory is not required to store program code, its width may also vary and could conceivably be targeted for a specific application.

Also, since data memory is accessed by the CPU only through appropriate registers, it is possible for register modules to access memory entirely independent from the main processor, providing the framework for direct memory-access operations. It is also possible to have more than one type of data memory, each accessed through a different register set.

1.3 Register Set

Since all functions in the MAXQ family are accessed through registers, common functionality is provided through a common register set. Many of these registers provide the equivalent of higher level op codes by directly accessing the arithmetic logic unit (ALU), the loop counter registers, and the data pointer registers. Others, such as the interrupt registers, provide common control and configuration functions that are equivalent across all MAXQ microcontrollers.

The common register set, also known as the System Registers, includes the following:

- ALU access and control registers, including working accumulator registers and the processor status flags
- Two Data Pointers and a Frame Pointer for data memory access
- Auto-decrementing Loop Counters for fast, compact looping
- Instruction Pointer and other branching control access points
- Stack Pointer and an access point to the 16-bit-wide dedicated hardware stack
- Interrupt vector, identification, and masking registers

Peripherals and other features that can vary among MAXQ microcontroller devices are accessed through Peripheral registers. These registers, grouped into register modules, provide such additional functionality as:

- Universal Asynchronous Receiver/Transmitter (UART) Serial Ports
- High-Speed Timers and Counters
- Serial Peripheral Interface (SPI™) ports
- Hardware Multiplier

- 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.

SECTION 2: ARCHITECTURE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 2: ARCHITECTURE

The MAXQ architecture is designed to be modular and expandable. Top-level instruction decoding is extremely simple and based on transfers to and from registers. The registers are organized into functional modules, which are in turn divided into the System Register and Peripheral Register groups. Figure 2-1 illustrates the modular architecture and the basic transport possibilities.

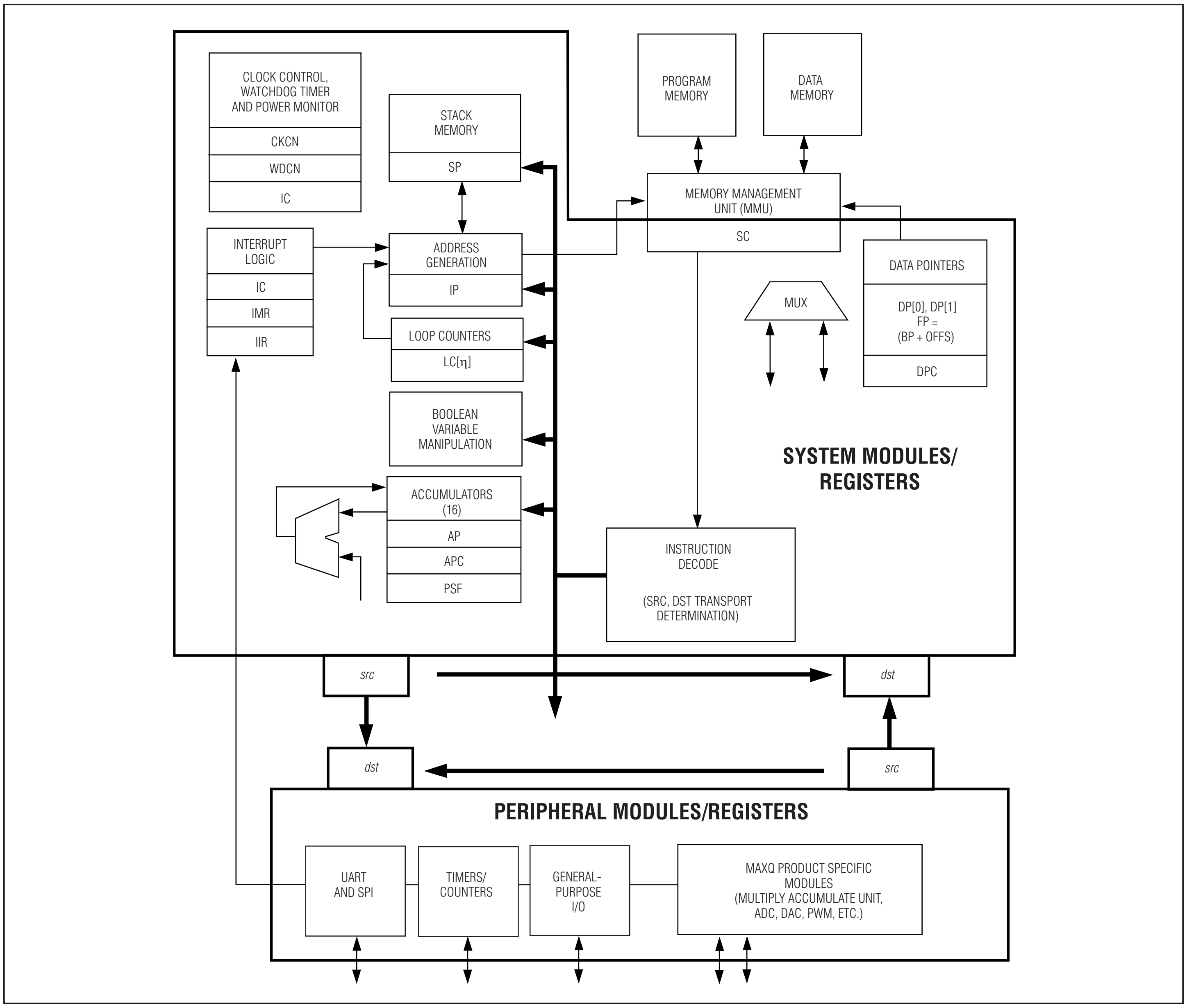

CLOCK CONTROL, WATCHDOG TIMER AND POWER MONITOR
CKCN
WDCN
IC
STACK MEMORY
SP
PROGRAM MEMORY
DATA MEMORY
MEMORY MANAGEMENT UNIT (MMU)
SC
INTERRUPT LOGIC
IC
IMR
IIR
ADDRESS GENERATION
IP
LOOP COUNTERS
LC[η]
DATA POINTERS
DP[0], DP[1]
FP = (BP + OFFS)
DPC
MUX
BOOLEAN VARIABLE MANIPULATION
SYSTEM MODULES/ REGISTERS
ACCUMULATORS (16)
AP
APC
PSF
INSTRUCTION DECODE
(SRC, DST TRANSPORT DETERMINATION)
src
dst
dst
src
PERIPHERAL MODULES/REGISTERS
UART AND SPI
TIMERS/ COUNTERS
GENERAL-PURPOSE I/O
MAXQ PRODUCT SPECIFIC MODULES (MULTIPLY ACCUMULATE UNIT, ADC, DAC, PWM, ETC.)

Figure 2-1. MAXQ Transport-Triggered Architecture

Memory access from the MAXQ is based on a Harvard architecture with separate address spaces for program and data memory. The simple instruction set and transport-triggered architecture allow the MAXQ to run in a nonpipelined execution mode where each instruction can be fetched from memory, decoded, and executed in a single clock cycle. Data memory is accessed through one of three data pointer registers. Two of these data pointers, DP[0] and DP[1], are stand-alone 16-bit pointers. The third data pointer, FP, is composed of a 16-bit base pointer (BP) and an 8-bit offset register (OFFS). All three pointers support post-increment/decrement functionality for read operations and pre-increment/decrement for write operations. For the Frame Pointer (FP=BP[Offs]), the increment/decrement operation is executed on the OFFS register and does not affect the base pointer (BP). Stack functionality is provided by dedicated memory with a 16-bit width and a typical depth of 8 (although this varies dependent upon the MAXQ product). An on-chip memory management unit (MMU) is accessible through system registers to allow logical remapping of physical program and data spaces, and thus facilitates in-system programming and fast access to data tables, arrays, and constants physically located in program memory.

2.1 Instruction Decoding

Every MAXQ instruction is encoded as a single 16-bit word according to the format in Figure 2-2.

FORMAT	DESTINATION							SOURCE							
f	d	d	d	d	d	d	d	s	s	s	s	s	s	s	s

Figure 2-2. Instruction Word Format

Bit 15 (f) indicates the format for the source field of the instruction as follows:

- If f equals 0, the instruction is an immediate source instruction, and the source field represents an immediate 8-bit value.
- If f equals 1, the instruction is a register source instruction, and the source field represents the register that the source value will be read from.

Bits 0 to 7 (ssssssss) represent the source for the transfer. Depending on the value of the format field, this can either be an immediate value or a source register. If this field represents a register, the lower four bits contain the module specifier and the upper four bits contain the register index in that module.

Bits 8 to 14 (ddddddd) represent the destination for the transfer. This value always represents a destination register, with the lower four bits containing the module specifier and the upper three bits containing the register sub-index within that module.

Since the source field is 8 bits wide and 4 bits are required to specify the module, any one of 16 registers in that module may be specified as a source. However, the destination field has one less bit, which means that only eight registers in a module can be specified as a destination in a single-cycle instruction.

While the asymmetry between source and destination fields of the op code may initially be considered a limitation, this space can be used effectively. Firstly, since read-only registers will never be specified as destinations, they can be placed in the second eight locations in a module to give single-cycle read access. Secondly, there are often critical control or configuration bits associated with system and certain peripheral modules where limited write access is beneficial (e.g., watchdog-timer enable and reset bits). By placing such bits in one of the upper 24 registers of a module, this write protection is added in a way that is virtually transparent to the assembly source code. Anytime that it is necessary to directly select one of the upper 24 registers as a destination, the prefix register PFX is used to supply the extra destination bits. This prefix register write is inserted automatically by the assembler and requires one additional execution cycle.

The MAXQ architecture is transport-triggered. This means that writing to or reading from certain register locations will also cause side effects to occur. These side effects form the basis for the higher level op codes defined by the assembler, such as ADDC, OR, JUMP, and so on. While these op codes are actually implemented as MOVE instructions between certain register locations, the encoding is handled by the assembler and need not be a concern to the programmer. The registers defined in the System Register and Peripheral Register maps operate as described in the documentation; the unused "empty" locations are the ones used for these special cases.

The MAXQ instruction set is designed to be highly orthogonal. All arithmetic and logical operations that use two registers can use any register along with the accumulator. Data can be transferred between any two registers in a single instruction.

2.2 Register Space

The MAXQ architecture provides a total of 16 register modules. Each of these modules contains 32 registers. The first eight registers in each module may be read from or written to in a single cycle; the second eight registers may be read from in a single cycle and written to in two cycles (by using the prefix register PFX); the last sixteen registers may be read or written in two cycles (always requiring use of the prefix register PFX).

Registers may be either 8 or 16 bits in length. Within a register, any number of bits can be implemented; bits not implemented are fixed at zero. Data transfers between registers of different sizes are handled as shown in Table 2-1.

- If the source and destination registers are both 8 bits wide, data is transferred bit to bit accordingly.
- If the source register is 8 bits wide and the destination register is 16 bits wide, the data from the source register is transferred into the lower 8 bits of the destination register. The upper 8 bits of the destination register are set to the current value of the prefix register; this value is normally zero, but it can be set to a different value by the previous instruction if needed. The prefix register reverts back to zero after one cycle, so this must be done by the instruction immediately before the one that will be using the value.
- If the source register is 16 bits wide and the destination register is 8 bits wide, the lower 8 bits of the source are transferred to the destination register.
- If both registers are 16 bits wide, data is copied bit to bit.

Table 2-1. Register-to-Register Transfer Operations

SOURCE REGISTER SIZE (BITS)	DESTINATION REGISTER SIZE (BITS)	PREFIX SET?	DESTINATION SET TO VALUE	
			HIGH 8 BITS	LOW 8 BITS
8	8	—		Source [7:0]
8	16	No	00h	Source [7:0]
8	16	Yes	Prefix [7:0]	Source [7:0]
16	8	—		Source [7:0]
16	16	No	Source [15:8]	Source [7:0]

The above rules apply to all data movements between defined registers. Data transfer to/from undefined register locations has the following behavior:

- If the destination is an undefined register, the MOVE is a dummy operation but may trigger an underlying operation according to the source register (e.g., @DP[n]--).
- If the destination is a defined register and the source is undefined, the source data for the transfer will depend upon the source module width. If the source is from a module containing 8-bit or 8-bit and 16-bit source registers, the source data will be equal to the prefix data concatenated with 00h. If the source is from a module containing only 16-bit source registers, 0000h source data is used for the transfer.

The 16 available register modules are broken up into two different groups. The low six modules (specifiers 0h through 5h) are known as the Peripheral Register modules, while the high 10 modules (specifiers 6h to Fh) are known as the System Register modules. These groupings are descriptive only, as there is no difference between accessing the two register groups from a programming perspective.

The System Registers define basic functionality that remains the same across all products based on the MAXQ architecture. This includes all register locations that are used to implement higher level op codes as well as the following common system features.

- ALU (MAXQ10: 8 bit; MAXQ20: 16 bit) and associated status flags (zero, equals, carry, sign, overflow)
- Eight working accumulator registers (MAXQ10: 8 bit width; MAXQ20: 16 bit width), along with associated control registers
- Instruction pointer
- Registers for interrupt control, handling, and identification
- Auto-decrementing Loop Counters for fast, compact looping
- Two Data Pointer registers and a Frame Pointer for data memory access

The Peripheral Registers define additional functionality that may be included by different products based on the MAXQ architecture. This functionality is broken up into discrete modules so that only the features that are required for a given product need to be included. Since the Peripheral Registers add functionality outside of the common MAXQ system architecture, they are not used to implement op codes.

2.3 Memory Organization

Beyond the internal register space, memory on the MAXQ microcontroller is organized according to a Harvard architecture, with a separate address space and bus for program memory and data memory. Stack memory is also separate and is accessed through a dedicated register set.

To provide additional memory map flexibility, an MMU allows data memory space to be mapped into a predefined program memory segment, thus affording the possibility of code execution from data memory. Additionally, program memory space can be made accessible as data space, allowing access to constant data stored in program memory.

2.3.1 Program Memory

Program memory begins at address x0000h and is contiguous through the internal program memory. The actual size of the on-chip program memory available for user application is product dependent. Given a 16-bit program address bus, the maximum program space is 64kWords. Since the codewords are 16 bits, the program memory is therefore a 64k x 16 linear space.

Program memory is accessed directly by the program fetching unit and is addressed by the Instruction Pointer register. From an implementation perspective, system interrupts and branching instructions simply change the contents of the Instruction Pointer and force the op code fetch from a new program location. The Instruction Pointer is direct read/write accessible by the user software; write access to the Instruction Pointer will force program flow to the new address on the next cycle following the write. The content of the Instruction Pointer will be incremented by 1 automatically after each fetch operation. The Instruction Pointer defaults to 8000h, which is the starting address of the utility ROM. The default IP setting of 8000h is assigned to allow initial in-system programming to be accomplished with utility ROM code assistance. The utility ROM code interrogates a specific register bit in order to decide whether to execute in-system programming or jump immediately to user code starting at 0000h. The user code reset vector should always be stored in the lowest bytes of the program memory.

The program memory is normally implemented using nonvolatile memory, e.g., ROM, EEPROM, or Flash. ROM memory technology requires program code to be masked into the ROM during chip fabrication; no write access to program memory is available. EEPROM and Flash provide in-system programming capability but both technologies require that the memory targeted for the write operation be unprogrammed (erased). The utility ROM provides routines to carry out the necessary operations (erase, write, verify) on these nonvolatile memories.

2.3.2 Utility ROM

A utility ROM is normally placed in the upper 32kWord program memory space starting at address 8000h. This utility ROM potentially provides the following system utility functions:

- Reset vector
- Bootstrap function for system initialization
- In-application programming
- In-circuit debug

Following each reset, the processor automatically starts execution at address 8000h in the utility ROM, allowing ROM code to perform any necessary system support functions. Next, the System Programming Enable (SPE) bit is examined to determine whether system programming should commence or whether that code should be bypassed, instead forcing execution to vector to the start of user program code. When the SPE bit is set to logic 1, the processor will execute the prescribed Bootstrap Loader mode program that resides in utility ROM. The SPE bit defaults to 0. To enter the Bootstrap Loader mode, the SPE bit can be set to 1 during reset via the JTAG interface. When in-system programming is complete, the Bootstrap Loader can clear the SPE bit and reset the device such that the in-system programming routine is subsequently bypassed.

2.3.3 Data Memory

On-chip data memory begins at address x0000h and is contiguous through the internal data memory. The actual size of the on-chip data memory available for the user application is product dependent. Data memory is accessed via indirect register addressing through a Data Pointer (@DP[n]) or Frame Pointer (@BP[Offs]). The Data Pointer is used as one of the operands in a MOVE instruction. If the Data Pointer is used as source, the core performs a Load operation that reads data from the data memory location addressed by the Data Pointer. If the Data Pointer is used as destination, the core executes a Store operation that writes data to the data memory location addressed by the Data Pointer. The Data Pointer can be directly accessed by the user software.

The core incorporates two 16-bit Data Pointers (DP[0] and DP[1]) to support data memory accessing. All Data Pointers support indirect addressing mode and indirect addressing with auto-increment or auto-decrement. Data Pointers DP[0] and DP[1] can be used as post increment/decrement source pointers by a MOVE instruction or pre increment/decrement destination pointers by a MOVE instruction. Using Data Pointer indirectly with "++" will automatically increase the content of the active Data Pointer by 1 immediately following the execution of read data transfer (@DP[n]++) or immediately preceding the execution of a write operation (@++DP[n]). Using Data Pointer indirectly with "--" will decrease the content of the active Data Pointer by 1 immediately following the execution of read data transfer (@DP[n]--) or immediately preceding the execution of a write operation (@--DP[n]).

The Frame Pointer (BP[Offs]) is formed by 16-bit unsigned addition of Frame Pointer Base Register (BP) and Frame Pointer Offset Register (Offs). Frame Pointer can be used as a post increment/decrement source pointer by a MOVE instruction or as a pre increment/decrement destination pointer. Using Frame Pointer indirectly with "++" (@BP[++Offs] for a write or @BP[Offs++] for a read) will automatically increase the content of the Frame Pointer Offset by 1 immediately before or after the execution of data transfer depending upon whether it is used as a destination or source pointer respectively. Using Frame Pointer indirectly with "--" (@BP[--Offs] for a write or @BP[Offs--] for a read) will decrease the content of the Frame Pointer Offset by 1 immediately before/after execution of data transfer depending upon whether it is used as a destination or source pointer respectively. Note that the increment/decrement function affects the content of the Offs register only, while the contents of the BP register remain unaffected by the borrow/carry out from the Offs register.

A data memory cycle contains only one system clock period to support fast internal execution. This allows read or write operations on SRAM to be completed in one clock cycle. Data memory mapping and access control are handled by the MMU. Read/write access to the data memory can be in word or in byte.

2.3.4 Stack Memory

A 16-bit wide on-chip stack is provided by the MAXQ for storage of program return addresses and general-purpose use. The stack is used automatically by the processor when the CALL, RET, and RETI instructions are executed and when an interrupt is serviced; it can also be used explicitly to store and retrieve data by using the @SP- - source, @++SP destination, or the PUSH, POP, and POPI instructions. The POPI instruction acts identically to the POP instruction except that it additionally clears the INS bit.

The width of the stack is 16 bits to accommodate the instruction pointer size. The stack depth may vary between product implementations. As the stack pointer register SP is used to hold the index of the top of the stack, the maximum size of the stack allowed for a MAXQ product is defined by the number of bits defined in the SP register (e.g., 3 bits for stack depth of 8, 4 bits for stack depth of 16).

On reset, the stack pointer SP initializes to the top of the stack (e.g. 07h for an 8-word stack, 0Fh for a 16-word stack). The CALL, PUSH, and interrupt vectoring operations increment SP and then store a value at @SP. The RET, RETI, POP, and POPI operations retrieve the value at @SP and then decrement SP.

As with the other RAM-based modules, the stack memory is initialized to indeterminate values upon reset or power-up. Stack memory is dedicated for stack operations only and cannot be accessed through program or data address spaces.

2.4 Pseudo-Von Neumann Memory Mapping

The MAXQ supports a pseudo-Von Neumann memory structure that can merge program and data into a linear memory map. This is accomplished by mapping the data memory into the program space or mapping program memory segment into the data space. Program memory from x0000h to x7FFFh is the normal user code segment, followed by the utility ROM segment. The uppermost part of the 64kWord memory is the logical area for data memory when accessed as a code segment.

The program memory is logically divided into four program pages:

- P0 contains the lower 16kWords,
- P1 contains the second 16kWords,
- P2 contains the third 16kWords, and
- P3 contains the fourth 16kWords.

By default, P2 and P3 are not accessible for program execution until they are explicitly activated by the user software. The Upper Program Access (UPA) bit must be set to logic 1 to activate P2 and P3. Once UPA is set, P2 and P3 will occupy the upper half of the 64kWord program space. In this configuration (UPA = 1), the utility ROM cannot be accessed at program memory and the physical data memory cannot be accessed logically in program space.

The logical mapping of physical program memory page(s) into data space depends upon two factors: physical memory currently in use for program execution; and word/byte data memory access selection. If execution is from the utility ROM, physical program memory page(s) can logically be mapped to the upper half of data memory space. If logical data memory is used for execution, physical program memory page(s) can logically be mapped to the lower half of data memory space. If byte access mode is selected, only one 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).

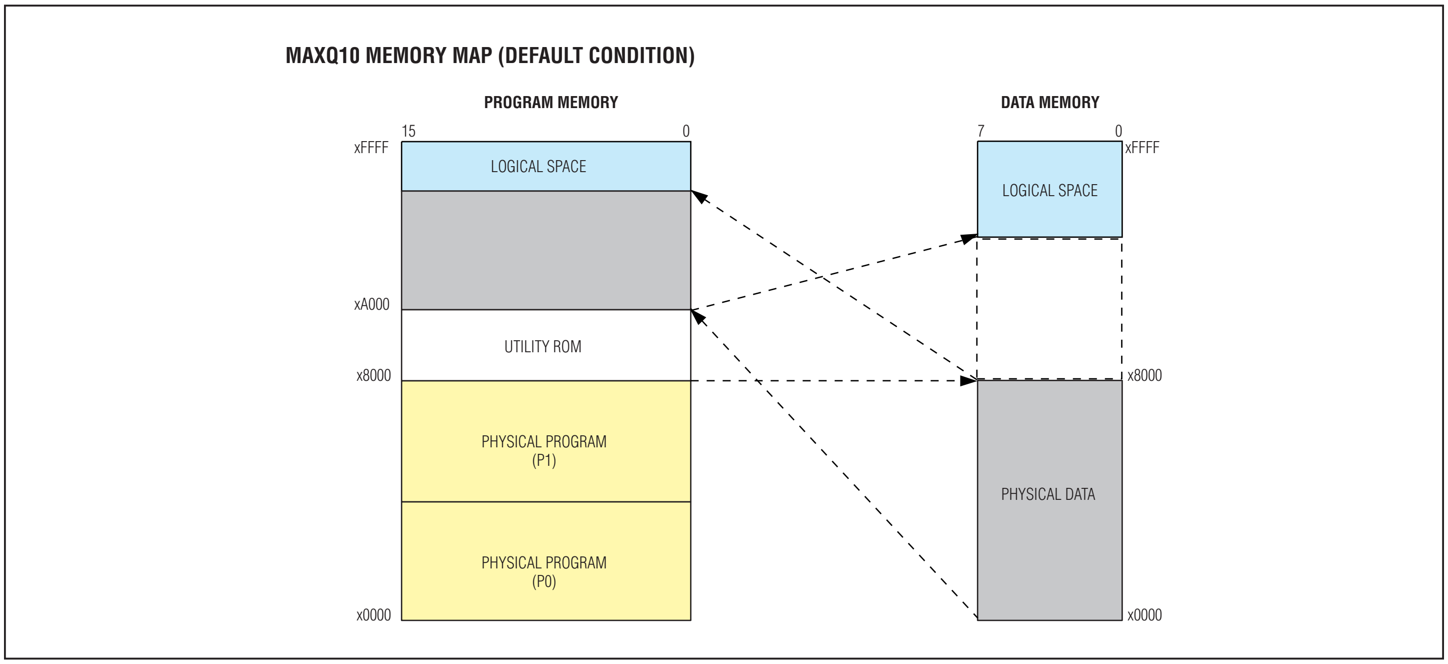


Figure 2-3. Pseudo Von Neumann Memory Map (MAXQ10 Default)

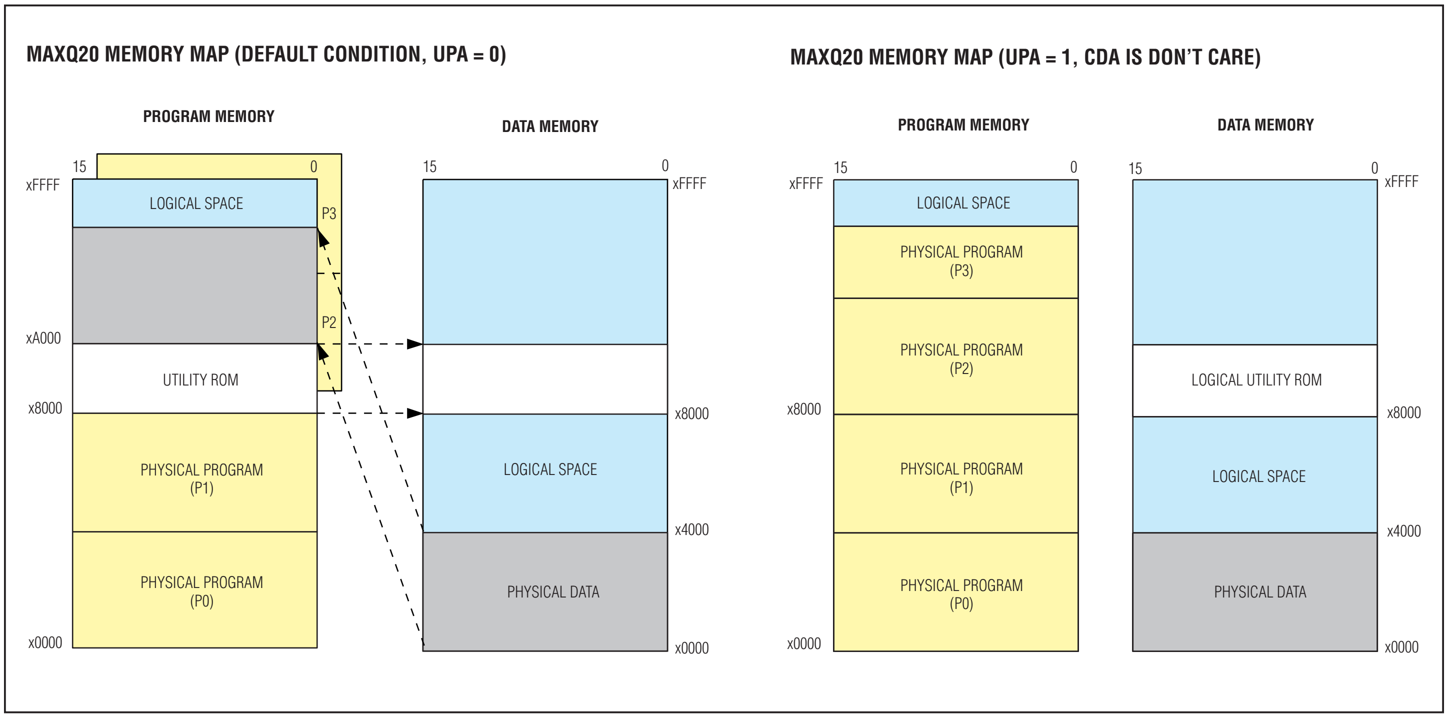


Figure 2-4. Pseudo Von Neumann Memory Map (MAXQ20 Default)

2.5 Pseudo-Von Neumann Memory Access

The pseudo-Von Neumann memory mapping is straightforward if there is no memory overlapping among the program, utility ROM, and data memory segments. However, for applications requiring large-size program memory, the paging scheme can be used to selectively activate those overlapped memory segments. The UPA bit can be used to activate the upper half of the physical program code (P2 and P3) for program execution. When accessing the program memory as data, the CDA bits can be used to select one of the four program pages as needed. Full data memory access to any of the four physical program memory pages is based on the assumption that the maximum physical data memory is in the range of 16k x 16. The other restriction for accessing the pseudo-Von Neumann map is that when program execution is in a particular memory segment, the same memory segment cannot be simultaneously be accessed as data.

When executing from the lower 32k program space (P0 and P1):

- The upper half of the code segment (P2 and P3) is accessible as program if the UPA bit is set to 1.
- The physical data memory is available for accessing as a code segment with offset at xA000h if the UPA bit is 0.
- Load and Store operations addressed to physical data memory are executed as normal.
- The utility ROM can be read as data, starting at x8000h of the data space.

When executing from the utility ROM (only allowable when UPA = 0):

- The lower 32k program space (P0 and P1) functions as normal program memory.
- The upper half of the code segment (P2 and P3) is not accessible as program (since UPA = 0).
- The physical data memory is available for accessing as a code segment with offset at xA000h.
- Load and Store operations addressed to physical data memory are executed as normal.
- One page (byte access mode) or two pages (word access mode) can be accessed as data with offset at x8000h as determined by the CDA1:0 bits.

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.

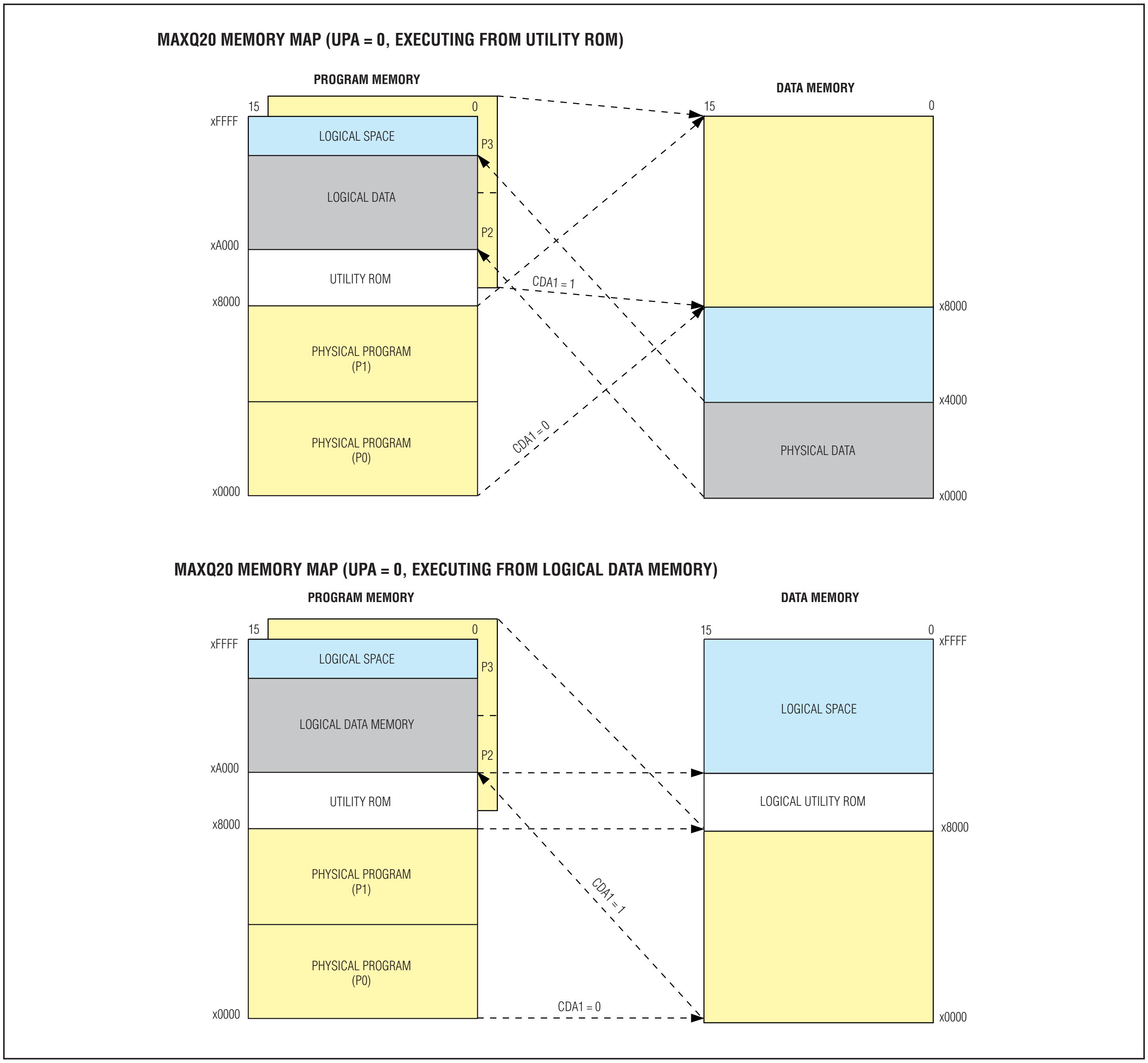


Figure 2-5. CDA Functions (Word Access Mode)

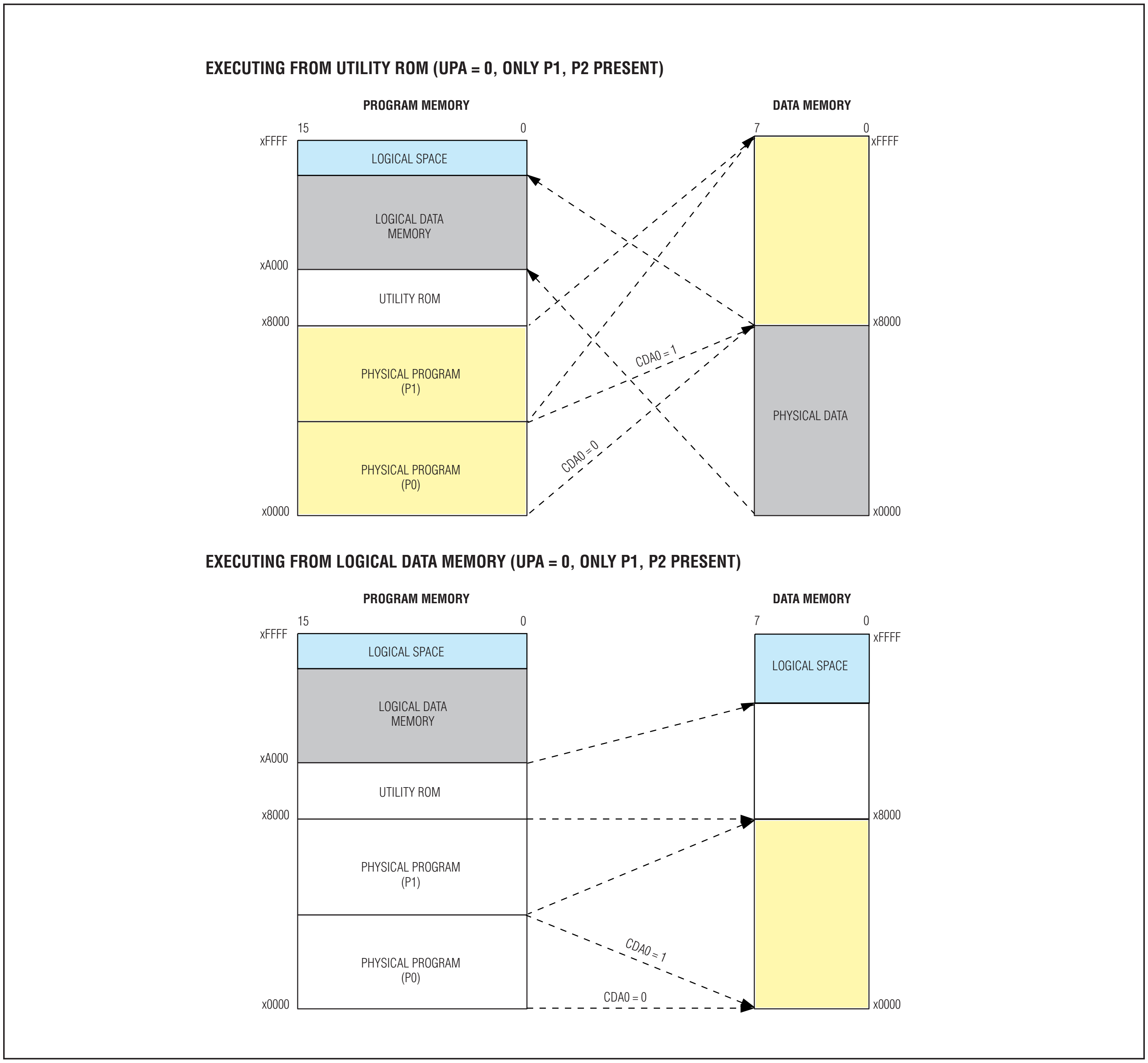


Figure 2-6. CDA Functions (Byte Access Mode)

- Physical program memory pages (P0, P1, P2, P3) are logically mapped into data space based upon the memory segment currently being used for execution, selection of byte/word access mode, and CDA1:0 bit settings (described under Pseudo Von Neumann Memory Map and Pseudo Von Neumann Memory Access.)

• Data memory

- Access can be either word or byte.
- All 16 data pointer address bits are significant in either access mode (word or byte.)

2.7 Clock Generation

All functional modules in the MAXQ are synchronized to a single system clock. The internal clock circuitry generates the system clock from one of four possible sources:

• Internal ring oscillator
• Internal oscillator, using an external crystal or resonator
• Internal relaxation oscillator, using an external RC
• External clock signal

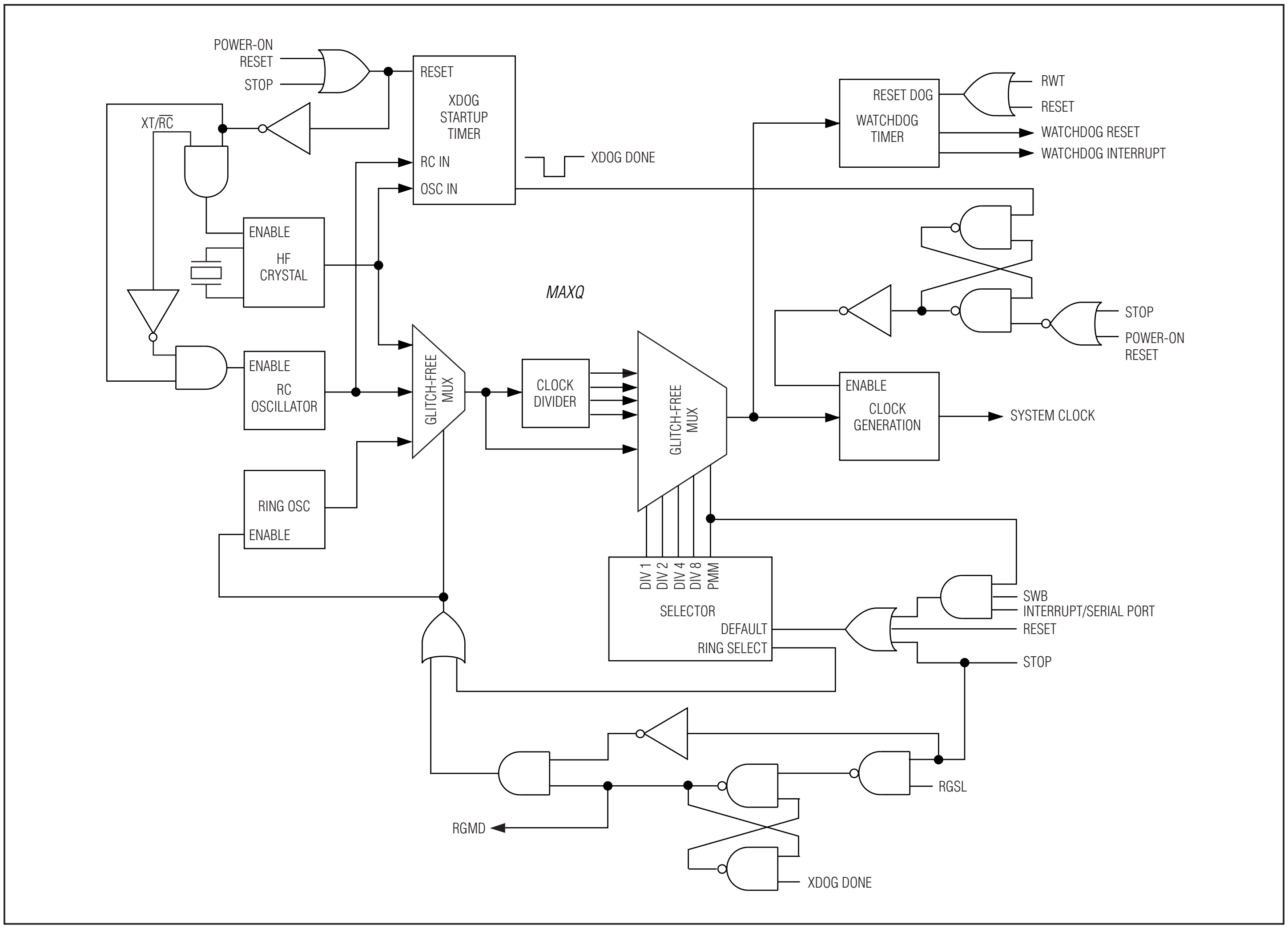


Figure 2-7. MAXQ Clock Sources

The external clock and crystal are mutually exclusive since they are input via the same clock pin. The basic clock source selection is made through two bits: RGSL and XT/$\overline{RC}$. The RGSL bit controls selection of the internal ring oscillator for system clock generation. When RGSL = 1, the internal ring oscillator is used for system clock generation. The RGSL bit is read/write accessible at any time and defaults to logic 0 on power-on reset only, allowing the internal ring oscillator to be used for system clock generation until the crystal warmup completes or until user code selects the XTAL1 pin configuration corresponding to an external RC (XT/$\overline{RC}$ =0). The XT/$\overline{RC}$ bit is writable only when the internal ring oscillator is explicitly selected for system clock generation (RGSL=1). The user code then must disable the internal ring oscillator (RGSL = 0) for the XT/$\overline{RC}$ elected clock source to take effect. Since RGSL and XT/$\overline{RC}$ reside in the same register, an external clock source selection can be made in the same instruction as the ring oscillator is disabled. Once the ring oscillator is disabled, RGSL = 0, the RGMD bit can be used to assess when the switchover to the source defined by XT/$\overline{RC}$ has occurred. The external RC clock selection (XT/$\overline{RC}$ =0) requires a 4-cycle count before it can be used, while the external crystal/resonator or external clock selection (XT/$\overline{RC}$ = 1) requires a 65,536-cycle count before it can be used. Requiring some type of warmup period for both external clock possibilities (crystal or RC) also serves as protection against an errant change of the XT/$\overline{RC}$ bit that produces a mismatch of XTAL1, XTAL2 pin function and external clock circuitry.

Each time code execution must start or restart (as may be the case when exiting stop mode) using the external clock source, the following sequence occurs:

- Reset the crystal warmup counter, and
- Allow the required warmup delay:
  - 65,536 external clock cycles if XT/$\overline{RC}$ = 1 and exiting from stop mode
  - four external clock cycles if XT/$\overline{RC}$ = 0
- During the warmup sequence, code execution may commence from the internal ring oscillator provided that one is present in the given MAXQ device. The user code may detect when the automatic switchover from the internal ring oscillator to the selected XT/$\overline{RC}$ source occurs by polling the RGMD status bit. If the RGSL bit is returned to logic 1 state (internal ring selection) while a warmup is in progress, the crystal amplifier or RC oscillator shuts down and the warmup process terminates.

2.7.1 Internal Ring Oscillator

The MAXQ microcontroller can source its main clock directly from an internal ring oscillator. The ring frequency varies over process, temperature, and supply voltage. For synchronization and timing purposes, the ring oscillator resets anytime it is selected for use as the primary system clock. The ring oscillator clock is divided down according to the PMME, CD1:0 bit selections just the same as the external non-ring clock possibilities. There is a four-cycle warmup delay associated with the internal ring oscillator when the system is going through a power-on reset or returning from the Stop mode.

2.7.2 External Clock (Crystal/Resonator)

An external quartz crystal or a ceramic resonator can be connected from XTAL1 to XTAL2 as the device determining the frequency, as illustrated in Figure 2-8. The fundamental mode of the crystal operates as inductive reactance in parallel resonance with external capacitance to the crystal.

Crystal specifications, operating temperature, operating voltage, and parasitic capacitance must be considered when designing the internal oscillator. To further reduce the effects of external noise, a guard ring can be placed around the oscillator circuitry.

Pins XTAL1 and XTAL2 are protected by clamping devices against on-chip electrostatic discharge. These clamping devices are diodes parasitic to the feedback resistor Rf in the inverter circuit of the oscillator. The inverter circuit is presented as a NAND gate, which can disable clock generation in STOP mode or if the internal ring oscillator is explicitly selected for use (i.e., RGSL = 1).

Noise at XTAL1 and XTAL2 can adversely affect on-chip clock timing. It is good design practice to place the crystal and capacitors near the oscillator circuitry and connect to XTAL1, XTAL2, and ground with direct shot trace. The typical values of external capacitors vary with the type of crystal used and should be initially selected based on the load capacitance as suggested by the crystal manufacturer.

For cost-sensitive applications, a ceramic resonator can be used instead of a crystal. Using the ceramic resonator may require a different circuit configuration and capacitance value.

2.7.3 External Clock (Direct Input)

The MAXQ CPU can also obtain the system clock signal directly from an external clock source. In this configuration, the clock generation circuitry is driven directly by an external clock.

To operate the core from an external clock, connect the clock source to the XTAL1 pin and leave the XTAL2 pin floating. The clock source should be driven through a CMOS driver. If the clock driver is a TTL gate, its output must be connected to VCC through a pullup resistor to ensure a satisfactory logic level for active clock pulses. To minimize system noise on the clock circuitry, the external clock source must meet the maximum rise and fall times and the minimum high and low times specified for the clock source. The external noise can affect clock generation circuit if these parameters do not meet the specification.

2.7.4 External RC

For timing-insensitive applications, the external RC option offers additional cost savings. The RC oscillator frequency is a function of the supply voltage, external resistor (Rext) and capacitor (Cext) values and tolerances, and the operating temperature. In addition to this, the oscillator frequency varies from unit to unit due to normal process parameter variation. Figure 2-9 shows how the external RC combination is connected to the MAXQ microcontroller.

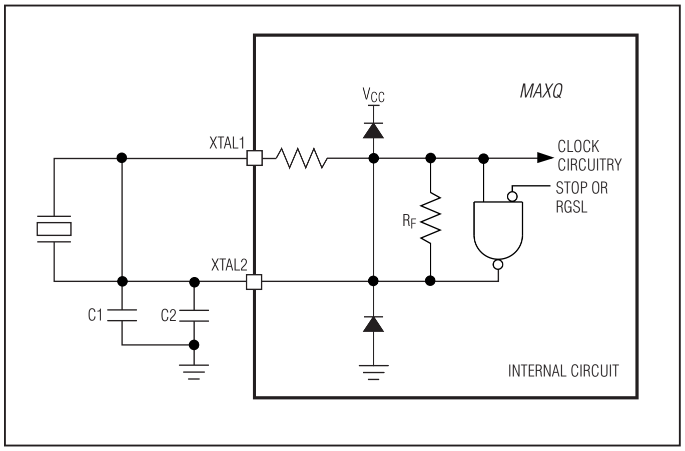


Figure 2-8. On-Chip Crystal Oscillator

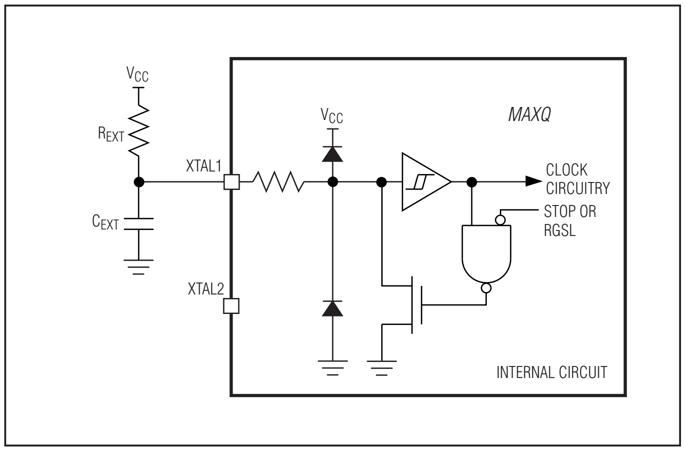


Figure 2-9. RC Relaxation Oscillator

2.7.5 Internal System Clock Generation

The internal system clock is derived from the currently selected oscillator input.

By default, one system clock cycle is generated per oscillator cycle, but the number of oscillator cycles per system clock can also be increased by setting the Power Management Mode Enable (PMME) bit and the Clock Divide Control (CD[1:0]) register bits per Table 2-2.

Table 2-2. System Clock Rate Control Settings

PMME	CD[1:0]	CYCLES PER CLOCK
0	00	1 (default)
0	01	2
0	10	4
0	11	8
1	xx	256

2.8 Interrupts

The MAXQ provides a single, programmable interrupt vector (IV) that can be used to handle internal and external interrupts. Interrupts can be generated from system level sources (e.g., watchdog timer) or by sources associated with the peripheral modules included in the specific MAXQ microcontroller. Only one interrupt can be handled at a time, and all interrupts naturally have the same priority. A programmable interrupt mask register allows software-controlled prioritization and nesting of high-priority interrupts.

2.8.1 Servicing Interrupts

For the MAXQ to service an interrupt, interrupts must be enabled globally, modularly, and locally. The Interrupt Global Enable (IGE) bit located in the Interrupt Control (IC) register acts as a global interrupt mask. This bit defaults to 0, and it must be set to 1 before any interrupt takes place.

The local interrupt-enable bit for a particular source is in one of the peripheral registers associated with that peripheral module, or in a system register for any system interrupt source. Between the global and local enables are intermediate per-module and system interrupt mask bits. These mask bits reside in the Interrupt Mask system register. By implementing intermediate per-module masking capability in a single register, interrupt sources spanning multiple modules can be selectively enabled/disabled in a single instruction. This promotes a simple, fast, and user-definable interrupt prioritization scheme. The interrupt source-enable hierarchy is illustrated in Figure 2-10.

When an interrupt condition occurs, its individual flag is set, even if the interrupt source is disabled at the local, module, or global level. Interrupt flags must be cleared within the user interrupt routine to avoid repeated interrupts from the same source.

Since all interrupts vector to the address contained in the Interrupt Vector (IV) register, the Interrupt Identification Register (IIR) may be used by the interrupt service routine to determine the module source of an interrupt. The IIR contains a bit flag for each peripheral module and one flag associated with all system interrupts; if the bit for a module is set, then an interrupt is pending that was initiated by that module. If a module is capable of generating interrupts for different reasons, then peripheral register bits inside the module provide a means to differentiate among interrupt sources.

The Interrupt Vector (IV) register provides the location of the interrupt service routine. It may be set to any location within program memory. The IV register defaults to 0000h on reset or power-up, so if it is not changed to a different address, the user program must determine whether a jump to 0000h came from a reset or interrupt source.

2.8.2 Interrupt System Operation

The interrupt handler hardware responds to any interrupt event when it is enabled. An interrupt event occurs when an interrupt flag is set. All interrupt requests are sampled at the rising edge of the clock and can be serviced by the processor one clock cycle later, assuming the request does not hit the interrupt exception window. The one-cycle stall between detection and acknowledgement/servicing is due to the fact that the current instruction may also be accessing the stack. For this reason, the CPU must allow the current instruction to complete before pushing the stack and vectoring to IV. If an interrupt exception window is generated by the currently executing instruction, the following instruction must be executed, so the interrupt service routine will be delayed an additional cycle.

Interrupt operation in the MAXQ CPU is essentially a state machine generated long CALL instruction. When the interrupt handler services an interrupt, it temporarily takes control of the CPU to perform the following sequence of actions:

1) The next instruction fetch from program memory is cancelled.

2) The return address is pushed on to the stack.

3) The INS bit is set to 1 to prevent recursive interrupt calls.

4) The instruction pointer is set to the location of the interrupt service routine (contained in the Interrupt Vector register).

5) The CPU begins executing the interrupt service routine.

Once the interrupt service routine completes, it should use the RETI instruction to return to the main program. Execution of RETI involves the following sequence of actions:

1) The return address is popped off the stack.

2) The INS bit is cleared to 0 to re-enable interrupt handling.

3) The instruction pointer is set to the return address that was popped off the stack.

4) The CPU continues execution of the main program.

Pending interrupt requests will not interrupt an RETI instruction; a new interrupt will be serviced after first being acknowledged in the execution cycle which follows the RETI instruction and then after the standard one stall cycle of interrupt latency. This means there will be at least two cycles between back-to-back interrupts.

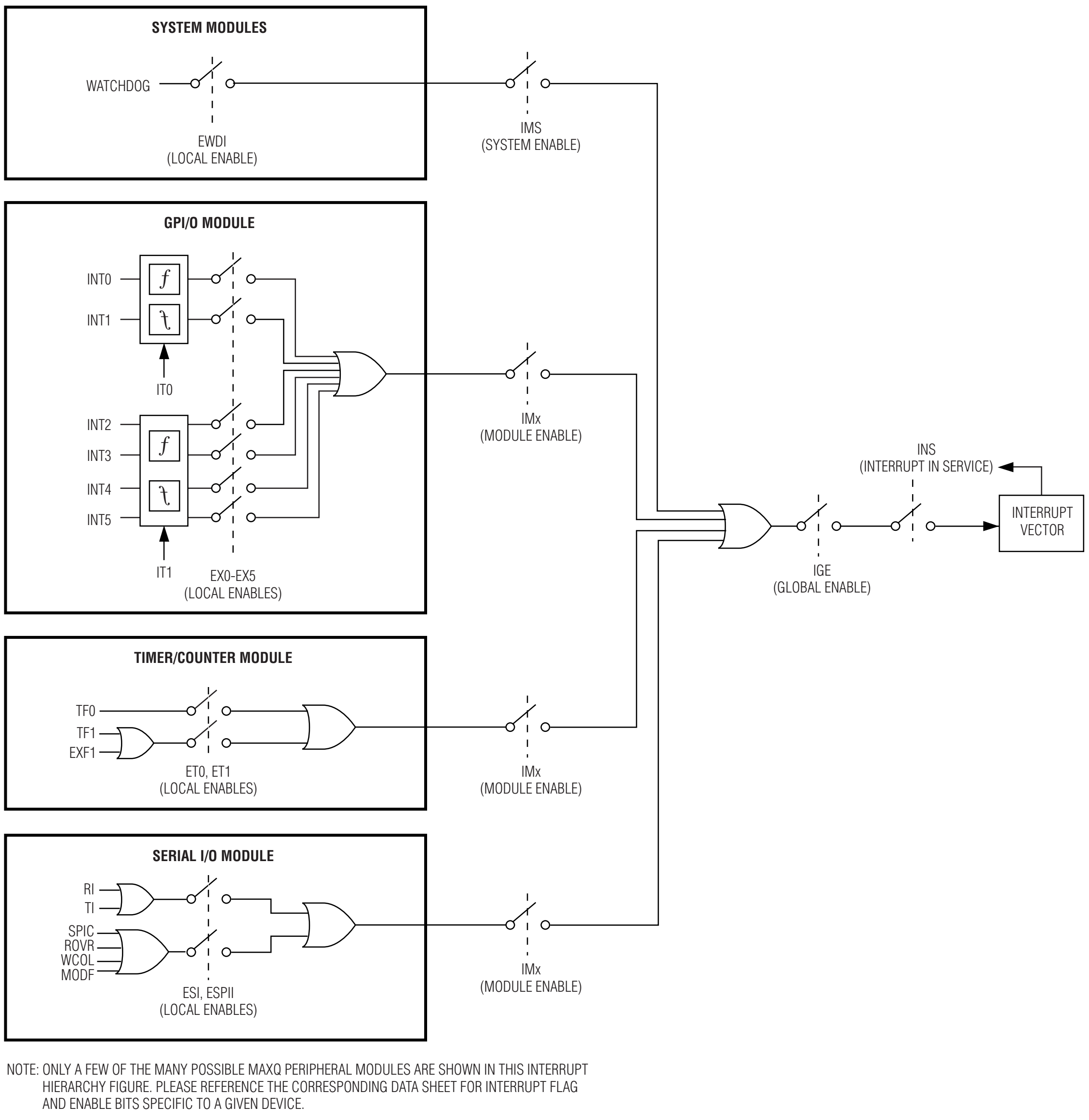


Figure 2-10. MAXQ Interrupt Source Hierarchy Example

2.8.3 Synchronous vs. Asynchronous Interrupt Sources

Interrupt sources can be classified as either asynchronous or synchronous. All internal interrupts are synchronous interrupts. An internal interrupt is directly routed to the interrupt handler that can be recognized in one cycle. All external interrupts are asynchronous interrupts by nature. When the device is not in Stop Mode, asynchronous interrupt sources are passed through a 3-clock sampling/glitch filter circuit before being routed to the interrupt handler. The sampling/glitch filter circuit is running on the undivided source clock (i.e., before PMME, CD1:0-controlled clock divide) such that the number of system clocks required to recognize an asynchronous interrupt request depends upon the system clock divide ratio:

- if the system clock divide ratio is 1, the interrupt request is recognized after 3 system clock;

- 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.

Certain MAXQ devices may also incorporate brownout detection capability. For these devices, an on-chip precision reference and comparator monitor the supply voltage V_{DD} to ensure that it is within acceptable limits. If V_{DD} is below the power-fail level, the power monitor initiates a reset condition. This can occur either when the MAXQ is first powered up when the V_{DD} supply is above the POR voltage threshold, or when V_{DD} drops out of tolerance from an acceptable level.

In either case, the reset condition is maintained until V_{DD} rises above the reset level V_{RST}. Once $V_{DD} > V_{RST}$, execution may resume following any necessary clock warmup delay.

When the processor exits from the power-on/brownout reset state, the POR bit in the Watchdog Control Register (WDCN) is set to 1 and can only be cleared by software. The user software can examine the POR bit following a reset to determine whether the reset was caused by a power-on reset or by another source.

2.9.1.2 External Reset

During normal operation, the MAXQ device is placed into external reset mode by holding the $\overline{RST}$ pin at logic 0 for at least four clock cycles. If MAXQ device is in the low-power Stop mode (i.e., system clock is not active), the $\overline{RST}$ pin becomes an asynchronous source, forcing the reset state immediately after being taken to logic 0. Once the MAXQ enters Reset mode, it remains in reset as long as the $\overline{RST}$ pin is held at logic 0. After the $\overline{RST}$ pin returns to logic 1, the processor exits the reset state within four clock cycles and begins program execution at address 8000h.

For many MAXQ devices, the $\overline{RST}$ pin is an output as well as an input. If a reset condition is caused by another source (such as a brownout reset, watchdog, or internal reset), an output reset pulse is generated at the $\overline{RST}$ pin for as long as the MAXQ remains in reset. If the $\overline{RST}$ pin is connected to an RC reset circuit or a similar circuit, it may not be able to drive the output reset signal. However, if this occurs it does not affect the internal reset condition.

2.9.1.3 Watchdog Timer Reset

The watchdog timer is a programmable hardware timer that can be set to reset the processor in the case of a software lockup or other unrecoverable error. Once the watchdog is enabled in this manner, the processor must reset the watchdog timer periodically to avoid a reset. If the processor does not reset the watchdog timer before it elapses, the watchdog will initiate a reset state.

If the watchdog resets the processor, it remains in reset, and holds the $\overline{RST}$ pin low, for four clock cycles. Once the reset condition is removed, the processor will begin executing program code at address 8000h. When a reset occurs due to a watchdog timeout, the Watchdog Timer Reset flag in the WDCN register is set to 1 and can only be cleared by software. User software can examine this bit following a reset to determine if that reset was caused by a watchdog timeout.

2.9.1.4 Internal System Reset

MAXQ devices may incorporate functions that logically warrant the ability to generate an internal system reset. This reset generation capability is assessed by MAXQ function based upon its expected use. In-system programming is a prime example of functionality that benefits by having the ability to reset the device. The exact in-system programming protocol is somewhat device and interface specific, however, it is expected that, upon completion of in-system programming, many users want the ability to reset the system. This internal (software-triggered) reset generation capability is possible following in-system programming.

2.9.2 Power Management Mode

There are two major sources of power dissipation in CMOS circuitry. The first is static dissipation caused by continuous leakage current. The second is dynamic dissipation caused by transient switching current required to charge and discharge load capacitors, as well as short circuit current produced by momentary connections between V_{DD} and ground during gate switching.

Usually, it is the dynamic switching power dissipation that dominates the total power consumption, and this power dissipation (P_D) for a CMOS circuit can be calculated in terms of load capacitance (C_L), power-supply voltage (V_{DD}) and operating frequency (f) as:

$$P_D = C_L \times V_{DD}^2 \times f$$

Capacitance and supply voltage are technology dependent and relatively fixed. However, the operating frequency determines the clock rate, and the required clock rate may be different from application to application depending on the amount of processing power required.

If an external crystal or oscillator is being used, the operating frequency can be adjusted by changing external components. However, it may be the case that a single application may require maximum processing power at some times and very little at others. Power Management mode allows an application to reduce its clock frequency, and therefore its power consumption, under software control.

Power Management Mode is invoked by setting the PMME bit to 1. Once this bit has been set, one system clock cycle will occur every 256 oscillator cycles. All operations continue as normal in this mode, but at the reduced clock rate. Power Management Mode can be deactivated by clearing the PMME bit to 0; the PMME bit will also be cleared automatically to 0 by any reset condition.

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

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

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]
```

8-bit destination ← high byte (16-bit source)

If, however, we needed to load an 8-bit register with the high byte of a 16-bit source, it would be best to use the GR register. Transferring the 16-bit source to the GR register adds a single cycle.

```
        move   GR, LC[0]            ; move LC[0] to the GR register
        move   IC, GRH              ; copy the high byte into the IC register
```

16-bit destination ← concatenation (8-bit source, 8-bit source)

Two 8-bit source registers can be concatenated and stored into a 16-bit destination by using the prefix register to hold the high-order byte for the concatenated transfer. An additional cycle may be required if either source byte register index is greater than 0Fh or the 16-bit destination is greater than 07h.

```
        move   PFX[0], IC           ; load high order source byte IC into PFX
        move   @++SP, AP            ; store @DP[0] the concatenation of IC:AP

                                    ; 16-bit destination sub-index: dst=08h
                                    ;  8-bit source sub-indexes:
                                    ;  high=10h, low=11h
        move   PFX[1], #00h         ;
        move   PFX[3], high         ; PFX=00:high
        move   dst, low             ; dst=high:low
```

Low (16-bit destination) ← 8-bit source

To modify only the low byte of a given 16-bit destination, the 16-bit register should be moved into the GR register such that the high byte can be singulated and the low byte written exclusively. An additional cycle is required if the destination index is greater than 0Fh.

```
        move   GR, DP[0]            ; move DP[0] to the GR register
        move   PFX[0], GRH          ; get the high byte of DP[0] via GRH
        move   DP[0], #20h          ; store the new DP[0] value
                                    ; 16-bit destination sub-index: dst=10h
                                    ;  8-bit source sub-index: src=11h
        move   PFX[1], #00h         ;
        move   GR, dst              ; read dst word to the GR register
        move   PFX[5], GRH          ; get the high byte of dst via GRH
        move   dst, src             ; store the new dst value
```

High (16-bit destination) ← 8-bit source

To modify only the high byte of a given 16-bit destination, the 16-bit register should be moved into the GR register such that the low byte can be singulated and the high byte can be written exclusively. Additional cycles are required if the destination index is greater than 0Fh or if the source index is greater than 0Fh.

```
        move   GR, DP[0]            ; move DP[0] to the GR register
        move   PFX[0], #20h         ; get the high byte of DP[0] via GRH
        move   DP[0], GRL           ; store the new DP[0] value
                                    ; 16-bit destination sub-index: dst=10h
                                    ;  8-bit source sub-index: src=11h
        move   PFX[1], #00h         ;
        move   GR, dst              ; read dst word to the GR register
        move   PFX[1], #00h
        move   PFX[4], src          ; get the new src byte
        move   dst, GRL             ; store the new dst value
```

If the high byte needs to be cleared to 00h, the operation can be shortened by transferring only the GRL byte to the 16-bit destination (example follows):

```
        move   GR, DP[0]            ; move DP[0] to the GR register
        move   DP[0], GRL           ; store the new DP[0] value, 00h used for high byte
```

3.4 Reading and Writing Register Bits

The MOVE instruction can also be used to directly set or clear any one of the lowest 8 bits of a peripheral register in module 0h-5h or a system register in module 8h. The set or clear operation will not affect the upper byte of a 16-bit register that is the target of the set or clear operation. If a set or clear instruction is used on a destination register that does not support this type of operation, the register high byte will be written with the prefix data and the low byte will be written with the bit mask (i.e. all 0's with a single 1 for the set bit operation or all ones with a single 0 for the clear bit operation).

Register bits can be set or cleared individually using the MOVE instruction as follows.

```
          move   IGE, #1              ; set IGE (Interrupt Global Enable) bit
          move   APC.6, #0            ; clear IDS bit (APC.6)
```

As with other instructions, prefixing is required to select destination registers beyond index 07h.

The MOVE instruction may also be used to transfer any one of the lowest 8 bits from a register source or any bit of the active accumulator (Acc) to the Carry flag. There is no restriction on the source register module for the 'MOVE C, src.bit' instruction.

```
          move   C, IIR.3             ; copy IIR.3 to Carry
          move   C, Acc.7             ; copy Acc.7 to Carry
```

Prefixing is required to select source registers beyond index 15h.

3.5 Using the Arithmetic and Logic Unit

The MAXQ provides either an 8-bit (MAXQ10) or 16-bit (MAXQ20) ALU, which allows operations to be performed between the active accumulator and any other register. The default ALU configuration provides eight accumulator registers that are also either 8-bit (MAXQ10) or 16-bit (MAXQ20) wide, of which any one may be selected as the active accumulator. Many MAXQ devices will be equipped with 16 working accumulators.

3.5.1 Selecting the Active Accumulator

Any of the 16 accumulator registers A[0] through A[15] may be selected as the active accumulator by setting the low four bits of the Accumulator Pointer Register (AP) to the index of the accumulator register you want to select.

```
          move   AP, #01h             ; select A[1] as the active accumulator
          move   AP, #0Fh             ; select A[15] as the active accumulator
```

The current active accumulator can be accessed as the Acc register, which is also the register used as the implicit destination for all arithmetic and logical operations.

```
          move   A[0], #55h           ; set A[0] =   55 hex (MAXQ10)
                                      ;          = 0055 hex (MAXQ20)
          move   AP, #00h             ; select A[0] as active accumulator
          move   Acc, #55h            ; set A[0] =   55 hex (MAXQ10)
                                      ;          = 0055 hex (MAXQ20)
```

3.5.2 Enabling Auto-Increment and Auto-Decrement

The accumulator pointer AP can be set to automatically increment or decrement after each arithmetic or logical operation. This is useful for operations involving a number of accumulator registers, such as adding or subtracting two multibyte integers.

If auto-increment/decrement is enabled, the AP register increments or decrements after any of the following operations:

- ADD src (Add source to active accumulator)
- ADDC src (Add source to active accumulator with carry)
- SUB src (Subtract source from active accumulator)
- SUBB src (Subtract source from active accumulator with borrow)
- AND src (Logical AND active accumulator with source)
- OR src (Logical OR active accumulator with source)
- XOR src (Logical XOR active accumulator with source)
- CPL (Bit-wise complement active accumulator)
- NEG (Negate active accumulator)

- SLA (Arithmetic shift left on active accumulator)
- SLA2 (Arithmetic shift left active accumulator two bit positions)
- SLA4 (Arithmetic shift left active accumulator four bit positions)
- SRA (Arithmetic shift right on active accumulator)
- SRA2 (Arithmetic shift right active accumulator two bit positions)
- SRA4 (Arithmetic shift right active accumulator four bit positions)
- RL (Rotate active accumulator left)
- RLC (Rotate active accumulator left through Carry flag)
- RR (Rotate active accumulator right)
- RRC (Rotate active accumulator right through Carry flag)
- SR (Logical shift active accumulator right)
- MOVE Acc, src (Copy data from source to active accumulator)
- MOVE dst, Acc (Copy data from active accumulator to destination)
- MOVE Acc, Acc (Recirculation of active accumulator contents)
- XCHN (Exchange nibbles within each byte of active accumulator)

(MAXQ20 Only)

- XCH (Exchange active accumulator bytes)

The active accumulator may not be the source in any instruction where it is also the implicit destination.

There is an additional notation that can be used to refer to the active accumulator for the instruction "MOVE dst, Acc." If the instruction is instead written as "MOVE dst, A[AP]," the source value is still the active accumulator, but no AP auto-increment or auto-decrement function will take place, even if this function is enabled. Note that the active accumulator may not be the destination for the MOVE dst, A[AP] instruction (i.e. MOVE Acc, A[AP] is prohibited).

So, the two instructions

```
        move   A[7], Acc
        move   A[7], A[AP]
```

are equivalent, except that the first instruction triggers auto-inc/dec (if it is enabled), while the second one will never do so.

The Accumulator Pointer Control Register (APC) controls the automatic-increment/decrement mode as well as selects the range of bits (modulo) in the AP register that will be incremented or decremented. There are nine different unique settings for the APC register, as listed in Table 3-1.

Table 3-1. Accumulator Pointer Control Register Settings

APC.2 (MOD2)	APC.1 (MOD1)	APC.0 (MOD0)	APC.6 (IDS)	APC	AUTO INCREMENT/DECREMENT SETTING
0	0	0	X	00h	No auto-increment/decrement (default mode)
0	0	1	0	01h	Increment bit 0 of AP (modulo 2)
0	0	1	1	41h	Decrement bit 0 of AP (modulo 2)
0	1	0	0	02h	Increment bits [1:0] of AP (modulo 4)
0	1	0	1	42h	Decrement bits [1:0] of AP (modulo 4)
0	1	1	0	03h	Increment bits [2:0] of AP (modulo 8)
0	1	1	1	43h	Decrement bits [2:0] of AP (modulo 8)
1	0	0	0	04h	Increment all 4 bits of AP (modulo 16)
1	0	0	1	44h	Decrement all 4 bits of AP (modulo 16)

For the modulo increment or decrement operation, the selected range of bits in AP are incremented or decremented. However, if these bits roll over or under, they simply wrap around without affecting the remaining bits in the accumulator pointer. So, the operations can be defined as follows:

- Increment modulo 2: AP = AP[3:1] + ((AP[0] + 1) mod 2)
- Decrement modulo 2: AP = AP[3:1] + ((AP[0] - 1) mod 2)
- Increment modulo 4: AP = AP[3:2] + ((AP[1:0] + 1) mod 4)
- Decrement modulo 4: AP = AP[3:2] + ((AP[1:0] - 1) mod 4)
- Increment modulo 8: AP = AP[3] + ((AP[2:0] + 1) mod 8)
- Decrement modulo 8: AP = AP[3] + ((AP[2:0] - 1) mod 8)
- Increment modulo 16: AP = (AP + 1) mod 16
- Decrement modulo 16:   AP = (AP - 1) mod 16

For this example, assume that all 16 accumulator registers are initially set to zero.

```
move  AP, #02h          ; select A[2] as active accumulator
move  APC, #02h         ; auto-increment AP[1:0] modulo 4
                        ;  AP   A[0]*  A[1]*  A[2]*  A[3]*
                        ;  02   0000   0000   0000   0000
add   #01h              ;  03   0000   0000   0001   0000
add   #02h              ;  00   0000   0000   0001   0002
add   #03h              ;  01   0003   0000   0001   0002
add   #04h              ;  02   0003   0004   0001   0002
add   #05h              ;  03   0003   0004   0006   0002
                        ;
                        ; *the upper #00h byte of each accumulator
                        ;  is only present on the MAXQ20 device
```

3.5.3 ALU Operations Using the Active Accumulator and a Source

The following arithmetic and logical operations can use any register or immediate value as a source. The active accumulator Acc is always used as the second operand and the implicit destination. Also, Acc may not be used as the source for any of these operations.

```
add   A[4]              ; Acc = Acc + A[4]
addc  #32h              ; Acc = Acc + 32h + Carry (MAXQ10)
                        ; Acc = Acc + 0032h + Carry (MAXQ20)
sub   A[15]             ; Acc = Acc – A[15]
subb  A[1]              ; Acc = Acc – A[1] - Carry
cmp   #00h              ; If (Acc == 00h), set Equals flag (MAXQ10)
                        ; If (Acc == 0000h), set Equals flag (MAXQ20)
and   A[0]              ; Acc = Acc AND A[0]
or    #55h              ; Acc = Acc OR #55h (MAXQ10)
                        ; Acc = Acc OR #0055h (MAXQ20)
xor   A[1]              ; Acc = Acc XOR A[1]
```

3.5.4 ALU Operations Using Only the Active Accumulator

The following arithmetic and logical operations operate only on the active accumulator.

```
cpl                        ; Acc = NOT Acc
neg                        ; Acc = (NOT Acc) + 1
rl                         ; Rotate accumulator left (not using Carry)
rlc                        ; Rotate accumulator left through Carry
rr                         ; Rotate accumulator right (not using Carry)
rrc                        ; Rotate accumulator right through Carry
sla                        ; Shift accumulator left arithmetically once
sla2                       ; Shift accumulator left arithmetically twice
sla4                       ; Shift accumulator left arithmetically four times
sr                         ; Shift accumulator right, set Carry to Acc.0,
                           ; set Acc.7 to zero (MAXQ10)
                           ; set Acc.15 to zero (MAXQ20)
sra                        ; Shift accumulator right arithmetically once
sra2                       ; Shift accumulator right arithmetically twice
sra4                       ; Shift accumulator right arithmetically four times
xchn                       ; Swap low and high nibbles of each Acc byte
xch  (MAXQ20 only)         ; Swap low byte and high byte of Acc
```

3.5.5 ALU Bit Operations Using Only the Active Accumulator

The following operations operate on single bits of the current active accumulator in conjunction with the Carry flag. Any of these operations may use an Acc bit from 0 to 7 (for MAXQ10) or from 0 to 15 (for MAXQ20).

```
move  C, Acc.0             ; copy bit 0 of accumulator to Carry
move  Acc.5, C             ; copy Carry to bit 5 of accumulator
and   Acc.3                ; Acc.3 = Acc.3 AND Carry
or    Acc.0                ; Acc.0 = Acc.0 OR Carry
xor   Acc.1                ; Acc.1 = Acc.1 OR Carry
```

None of the above bit operations cause the auto-increment, auto-decrement, or modulo operations defined by the accumulator pointer control (APC) register.

3.5.6 MAXQ10 Example: Adding Two 4-Byte Numbers Using Auto-Increment

```
move  A[0], #78h           ; First number – 12345678h
move  A[1], #56h
move  A[2], #34h
move  A[3], #12h
move  A[4], #0AAh          ; Second number – 0AAAAAAAh
move  A[5], #0AAh
move  A[6], #0AAh
move  A[7], #0Ah
move  AP, #00h             ; A[0] is active accumulator
move  APC, #02h            ; Increment low two bits mod 4
add   A[4]                 ; A[0] = 78h + AAh     = 22h + Carry
addc  A[5]                 ; A[1] = 56h + AAh + 1 = 01h + Carry
addc  A[6]                 ; A[2] = 34h + AAh + 1 = DFh
addc  A[7]                 ; A[3] = 12h + 0Ah     = 1Ch
; 12345678h + 0AAAAAAAh = 1CDF0122h
```

3.5.7 MAXQ20 Example: Adding Two 4-Byte Numbers Using Auto-Increment

```
          move   A[0], #5678h          ; First number – 12345678h
          move   A[1], #1234h
          move   A[2], #0AAAAh         ; Second number – 0AAAAAAAh
          move   A[3], #0AAAh
          move   APC, #81h             ; Active Acc = A[0], increment low bit = mod 2
          add    A[2]                  ; A[0]  = 5678h + AAAAh = 0122h + Carry
          addc   A[3]                  ; A[1]  = 1234h + AAAh + 1 = 1CDFh
      ; 12345678h + 0AAAAAAAh = 1CDF0122h
```

3.6 Processor Status Flag Operations

The Processor Status Flag (PSF) register contains five flags that are used to indicate and store the results of arithmetic and logical operations, four of which can also be used for conditional program branching.

3.6.1 Sign Flag

The Sign flag (PSF.6) reflects the current state of the high bit of the active accumulator (Acc.7 for the MAXQ10 or Acc.15 for the MAXQ20). If signed arithmetic is being used, this flag indicates whether the value in the accumulator is positive or negative.

Since the Sign flag is a dynamic reflection of the high bit of the active accumulator, any instruction that changes the value in the active accumulator can potentially change the value of the Sign flag. Also, any instruction that changes which accumulator is the active one (including AP auto-increment/decrement) can also change the Sign flag.

The following operation uses the Sign flag:

- JUMP S, src (Jump if Sign flag is set)

3.6.2 Zero Flag

The Zero flag (PSF.7) is a dynamic flag that reflects the current state of the active accumulator Acc. If all bits in the active accumulator are zero, the Zero flag equals 1. Otherwise, it equals 0.

Since the Zero flag is a dynamic reflection of (Acc = 0), any instruction that changes the value in the active accumulator can potentially change the value of the Zero flag. Also, any instruction that changes which accumulator is the active one (including AP auto-increment/decrement) can also change the Zero flag.

The following operations use the Zero flag:

- JUMP Z, src (Jump if Zero flag is set)
- JUMP NZ, src (Jump if Zero flag is cleared)

3.6.3 Equals Flag

The Equals flag (PSF.0) is a static flag set by the CMP instruction. When the source given to the CMP instruction is equal to the active accumulator, the Equals flag is set to 1. When the source is different from the active accumulator, the Equals flag is cleared to 0.

The following instructions use the value of the Equals flag. Please note that the 'src' for the JUMP E/NE instructions must be immediate.

- JUMP E, src (Jump if Equals flag is set)
- JUMP NE, src (Jump if Equals flag is cleared)

In addition to the CMP instruction, any instruction using PSF as the destination can alter the Equals flag.

3.6.4 Carry Flag

The Carry flag (PSF.1) is a static flag indicating that a carry or borrow bit resulted from the last ADD/ADDC or SUB/SUBB operation. Unlike the other status flags, it can be set or cleared explicitly and is also used as a generic bit operand by many other instructions.

The following instructions can alter the Carry flag:

- ADD src (Add source to active accumulator)
- ADDC src (Add source and Carry to active accumulator)
- SUB src (Subtract source from active accumulator)
- SUBB src (Subtract source and Carry from active accumulator)

- SLA, SLA2, SLA4 (Arithmetic shift left active accumulator)
- SRA, SRA2, SRA4 (Arithmetic shift right active accumulator)
- SR (Shift active accumulator right)
- RLC/RRC (Rotate active accumulator left / right through Carry)
- MOVE C, Acc.<b> (Set Carry to selected active accumulator bit)
- MOVE C, #i (Explicitly set, i = 1, or clear, i = 0, the Carry flag)
- CPL C (Complement Carry)
- AND Acc.<b>
- OR Acc.<b>
- XOR Acc.<b>
- MOVE C, src.<b> (Copy bit addressable register bit to Carry)
- any instruction using PSF as the destination

The following instructions use the value of the Carry flag:

- ADDC src (Add source and Carry to active accumulator)
- SUBB src (Subtract source and Carry from active accumulator)
- RLC/RRC (Rotate active accumulator left/right through Carry)
- CPL C (Complement Carry)
- MOVE Acc.<b>, C (Set selected active accumulator bit to Carry)
- AND Acc.<b> (Carry = Carry AND selected active accumulator bit)
- OR Acc.<b> (Carry = Carry OR selected active accumulator bit)
- XOR Acc.<b> (Carry = Carry XOR selected active accumulator bit)
- JUMP C, src (Jump if Carry flag is set)
- JUMP NC, src (Jump if Carry flag is cleared)

3.6.5 Overflow Flag

The Overflow flag (PSF.2) is a static flag indicating that the carry or borrow bit (Carry status Flag) resulting from the last ADD/ADDC or SUB/SUBB operation but did not match the carry or borrow of the high order bit of the active accumulator. The overflow flag is useful when performing signed arithmetic operations.

The following instructions can alter the Overflow flag:

- ADD src (Add source to active accumulator)
- ADDC src (Add source and Carry to active accumulator)
- SUB src (Subtract source from active accumulator)
- SUBB src (Subtract source and Carry from active accumulator)

3.7 Controlling Program Flow

The MAXQ provides several options to control program flow and branching. Jumps may be unconditional, conditional, relative, or absolute. Subroutine calls store the return address on the hardware stack for later return. Built-in counters and address registers are provided to control looping operations.

3.7.1 Obtaining the Next Execution Address

The address of the next instruction to be executed can be read at any time by reading the Instruction Pointer (IP) register. This can be particularly useful for initializing loops. Note that the value returned is actually the address of the current instruction plus 1, so this will be the address of the next instruction executed as long as the current instruction does not cause a jump.

3.7.2 Unconditional Jumps

An unconditional jump can be relative (IP +127/-128 words) or absolute (to anywhere in program space). Relative jumps must use an 8-bit immediate operand, such as

```
Label1:                     ; must be within +127/-128 words of the JUMP
...
jump  Label1
```

Absolute jumps can use a 16-bit immediate operand, a 16-bit register, or an 8-bit register.

```
   jump  LongJump           ; assembles to: move PFX[0], #high(LongJump)
                            ;               jump         #low(LongJump)
   jump  DP[0]              ; absolute jump to the address in DP[0]
```

If an 8-bit register is used as the jump destination, the prefix value is used as the high byte of the address and the register is used as the low byte.

3.7.3 Conditional Jumps

Conditional jumps transfer program execution based on the value of one of the status flags (C, E, Z, S). Except where noted for JUMP E and JUMP NE, the absolute and relative operands allowed are the same as for the unconditional JUMP command.

```
   jump c, Label1           ; jump to Label1 if Carry is set
   jump nc, LongJump        ; jump to LongJump if Carry is not set
   jump z, LC[0]            ; jump to 16-bit register destination if
                            ;    Zero is set
   jump nz, Label1          ; jump to Label1 if Zero is not set (Acc<>0)
   jump s, A[2]             ; jump to A[2] if Sign flag is set
   jump e, Label1           ; jump to Label1 if Equal is set
   jump ne, Label1          ; jump to Label1 if Equal is cleared
```

JUMP E and JUMP NE may only use immediate destinations.

3.7.4 Calling Subroutines

The CALL instruction works the same as the unconditional JUMP, except that the next execution address is pushed on the stack before transferring program execution to the branch address. The RET instruction is used to return from a normal call, and RETI is used to return from an interrupt handler routine.

```
   call  Label1             ; if Label1 is relative,
                            ; assembles to : call #immediate
   call  LongCall           ; assembles to:  move PFX[0], #high(LongCall)
                            ;                 call #low(LongCall)
   call  LC[0]              ; call to address in LC[0]
LongCall:
   ret                      ; return from subroutine
```

3.7.5 Looping Operations

Looping over a section of code can be performed by using the conditional jump instructions. However, there is built-in functionality, in the form of the 'DJNZ LC[n], src' instruction, to support faster, more compact looping code with separate loop counters. The 16-bit registers LC[0], and LC[1] are used to store these loop counts. The 'DJNZ LC[n], src' instruction automatically decrements the associated loop counter register and jumps to the loop address specified by src if the loop counter has not reached 0.

To initialize a loop, set the LC[n] register to the count you wish to use before entering the loop's main body.

The desired loop address should be supplied in the src operand of the 'DJNZ LC[n], src' instruction. When the supplied loop address is relative (+127/-128 words) to the DJNZ LC[n] instruction, as is typically the case, the assembler automatically calculates the relative offset and inserts this immediate value in the object code.

```
   move  LC[1], #10h        ; loop 16 times
LoopTop:                    ; loop addr relative to djnz LC[n],src instruction
   call  LoopSub
   djnz  LC[1], LoopTop     ; decrement LC[1] and jump if nonzero
```

When the supplied loop address is outside the relative jump range, the prefix register (PFX[0]) is used to supply the high byte of the loop address as required.

```
          move   LC[1], #10h          ; loop 16 times
       LoopTop:                       ; loop addr not relative to djnz LC[n],src
          call   LoopSub
          ...
          djnz   LC[1], LoopTop       ; decrement LC[1] and jump if nonzero
                                      ; assembles to: move PFX[0], #high(LoopTop)
                                      ;               djnz LC[1], #low(LoopTop)
```

If loop execution speed is critical and a relative jump cannot be used, one might consider preloading an internal 16-bit register with the src loop address for the 'DJNZ LC[n], src' loop. This ensures that the prefix register will not be needed to supply the loop address and always yields the fastest execution of the DJNZ instruction.

```
          move   LC[0], #LoopTop      ; using LC[0] as address holding register
                                      ; assembles to: move PFX[0], #high(LoopTop)
                                      ;               move LC[0], #low(LoopTop)
          move   LC[1], #10h          ; loop 16 times
          ...
       LoopTop:                       ; loop address not relative to djnz LC[n],src
          call   LoopSub
          ...
          djnz   LC[1], LC[0]         ; decrement LC[1] and jump if nonzero
```

If opting to preload the loop address to an internal 16-bit register, the most time and code efficient means is by performing the load in the instruction just prior to the top of the loop:

```
          move   LC[1], #10h          ; Set loop counter to 16
          move   LC[0], IP            ; Set loop address to the next address
       LoopTop:                       ; loop addr not relative to djnz LC[n],src
          ...
```

3.7.6 Conditional Returns

Similar to the conditional jumps, the MAXQ microcontroller also supports a set of conditional return operations. Based upon the value of one of the status flags, the CPU can conditionally pop the stack and begin execution at the address popped from the stack. If the condition is not true, the conditional return instruction does not pop the stack and does not change the instruction pointer. The following conditional return operations are supported:

```
          RET C                       ; if C=1, a RET is executed
          RET NC                      ; if C=0, a RET is executed
          RET Z                       ; if Z=1 (Acc=00h), a RET is executed
          RET NZ                      ; if Z=0 (Acc<>00h), a RET is executed
          RET S                       ; if S=1, a RET is executed
```

3.8 Handling Interrupts

Handling interrupts in the MAXQ is a three-part process. First, the location of the interrupt handling routine must be set by writing the address to the 16-bit Interrupt Vector (IV) register. This register defaults to 0000h on reset, but this will usually not be the desired location since this will often be the location of reset/power-up code.

```
          move   IV, IntHandler       ; move PFX[0], #high(IntHandler)
                                      ; move IV, #low(IntHandler)
                                      ; PFX[0] write not needed if IntHandler addr=00xxh
```

Next, the interrupt must be enabled. For any interrupts to be handled, the IGE bit in the Interrupt and Control register (IC) must first be set to 1. Next, the interrupt itself must be enabled at the module level and locally within the module itself. The module interrupt enable is located in the Interrupt Mask register, while the location of the local interrupt enable will vary depending on the module in which the interrupt source is located.

Once the interrupt handler receives the interrupt, the Interrupt in Service (INS) bit will be set by hardware to block further interrupts, and execution control is transferred to the interrupt service routine. Within the interrupt service routine, the source of the interrupt must be determined. Since all interrupts go to the same interrupt service routine, the Interrupt Identification Register (IIR) must be examined to determine which module initiated the interrupt. For example, the II0 (IIR.0) bit will be set if there is a pending interrupt from module 0. These bits cannot be cleared directly; instead, the appropriate bit flag in the module must be cleared once the interrupt is handled.

INS is set automatically on entry to the interrupt handler and cleared automatically on exit (RETI).

```
IntHandler:
          push  PSF                 ; save C since used in identification process
          move  C, IIR.X            ; check highest priority flag in IIR
          jump  C, ISR_X            ; if IIR.X is set, interrupt from module X
          move  C, IIR.Y            ; check next highest priority int source
          jump  C, ISR_Y            ; if IIR.Y is set, interrupt from module Y
          ...
ISR_X:
          ...
          reti
```

To support high priority interrupts while servicing another interrupt source, the IMR register may be used to create a user-defined prioritization. The IMR mask register should not be utilized when the highest priority interrupt is being serviced because the highest priority interrupt should never be interrupted. This is default condition when a hardware branch is made the Interrupt Vector address (INS is set to 1 by hardware and all other interrupt sources are blocked). The code below demonstrates how to use IMR to allow other interrupts.

```
ISR_Z:
          pop   PSF                 ; restore PSF
          push  IMR                 ; save current interrupt mask
          move  IMR, #int_mask      ; new mask to allow only higher priority ints
          move  INS, #0             ; re-enable interrupts
          ...
          (interrupt servicing code)
          ...
          pop   IMR                 ; restore previous interrupt mask
          ret                       ; back to code or lower priority interrupt
```

Please note that configuring a given IMR register mask bit to '0' only prevents interrupt conditions from the corresponding module or system from generating an interrupt request. Configuring an IMR mask bit to '0' does not prevent the corresponding IIR system or module identification flag from being set. This means that when using the IMR mask register functionality to block interrupts, there may be cases when both the mask (IMR.x) and identifier (IIR.x) bits should be considered when determining if the corresponding peripheral should be serviced.

3.8.1 Conditional Return from Interrupt

Similar to the conditional returns, the MAXQ microcontroller also supports a set of conditional return from interrupt operations. Based upon the value of one of the status flags, the CPU can conditionally pop the stack, clear the INS bit to 0, and begin execution at the address popped from the stack. If the condition is not true, the conditional return from interrupt instruction leaves the INS bit unchanged, does not pop the stack and does not change the instruction pointer. The following conditional return from interrupt operations are supported:

```
          RETI C                    ; if C=1, a RETI is executed
          RETI NC                   ; if C=0, a RETI is executed
          RETI Z                    ; if Z=1 (Acc=00h), a RETI is executed
          RETI NZ                   ; if Z=0 (Acc<>00h), a RETI is executed
          RETI S                    ; if S=1, a RETI is executed
```

3.9 Accessing the Stack

The hardware stack is used automatically by the CALL, RET and RETI instructions, but it can also be used explicitly to store and retrieve data. All values stored on the stack are 16 bits wide.

The PUSH instruction increments the stack pointer SP and then stores a value on the stack. When pushing a 16-bit value onto the stack, the entire value is stored. However, when pushing an 8-bit value onto the stack, the high byte stored on the stack comes from the prefix register. The @++SP stack access mnemonic is the associated destination specifier that generates this push behavior, thus the following two instruction sequences are equivalent:

```
          move  PFX[0], IC
          push  PSF                 ; stored on stack: IC:PSF

          move  PFX[0], IC
          move  @++SP, PSF          ; stored on stack: IC:PSF
```

The POP instruction removes a value from the stack and then decrements the stack pointer. The @SP-- stack access mnemonic is the associated source specifier that generates this behavior, thus the following two instructions are equivalent:

```
          pop   PSF
          move  PSF, @SP--
```

The POPI instruction is equivalent to the POP instruction but additionally clears the INS bit to 0. Thus, the following two instructions would be equivalent:

```
          popi  IP
          reti
```

The @SP-- mnemonic can be used by the MAXQ microcontroller so that stack values may be used directly by ALU operations (e.g. ADD src, XOR src, etc.) without requiring that the value be first popped into an intermediate register or accumulator.

```
          add   @SP--               ; sum the last three words pushed onto the stack
          add   @SP--               ;  with Acc, disregarding overflow
          add   @SP--
```

The stack pointer SP can be set explicitly, however only those least significant bits needed to represent the stack depth for the associated MAXQ device are used. For a MAXQ device that has a stack depth of 16 words, only the lowest four bits are used and setting SP to 0Fh will return it to its reset state.

Since the stack is 16 bits wide, it is possible to store two 8-bit register values on it in a single location. This allows more efficient use of the stack if it is being used to save and restore registers at the start and end of a subroutine.

```
SubOne:
          move  PFX[0], IC
          push  PSF                 ; store IC:PSF on the stack
          ...
          pop   GR                  ; 16-bit register
          move  IC, GRH             ; IC was stored as high byte
          move  PSF, GRL            ; PSF was stored as low byte
          ret
```

3.10 Accessing Data Memory

Data memory is accessed through the data pointer registers DP[0] and DP[1] or the Frame Pointer BP[Offs]. Once one of these registers is set to a location in data memory, that location can be read or written as follows, using the mnemonic @DP[0], @DP[1] or @BP[OFFS] as a source or destination.

```
          move  DP[0], #0000h       ; set pointer to location 0000h
          move  A[0], @DP[0]        ; read from data memory
          move  @DP[0], #55h        ; write to data memory
```

Either of the data pointers may be post-incremented or post-decremented following any read or may be pre-incremented or pre-decremented before any write access by using the following syntax.

```
          move  A[0], @DP[0]++      ; increment DP[0] after read
          move  @++DP[0], A[1]      ; increment DP[0] before write
          move  A[5], @DP[1]--      ; decrement DP[1] after read
          move  @--DP[1], #00h      ; decrement DP[1] before write
```

The Frame Pointer (BP[OFFS]) is actually composed of a base pointer (BP) and an offset from the base pointer (OFFS). For the frame pointer, the offset register (OFFS) is the target of any increment or decrement operation. The base pointer (BP) is unaffected by increment and decrement operations on the Frame Pointer. Similar to DP[n], the OFFS register may be pre-incremented/decremented when writing to data memory and may be post-incremented/decremented when reading from data memory.

```
move   A[0], @BP[OFFS--]     ; decrement OFFS after read
move   @BP[++OFFS], A[1]     ; increment OFFS before write
```

All three data pointers support both byte and word access to data memory. Each data pointer has its own word/byte select (WBSn) special-function register bit to control the access mode associated with the data pointer. These three register bits (WBS2, which controls BP[Offs] access; WBS1, which controls DP[1] access; and WBS0, which controls DP[0] access) reside in the Data Pointer Control (DPC) register. When a given WBSn control bit is configured to 1, the associated pointer is operated in the word access mode. When the WBSn bit is configured to 0, the pointer is operated in the byte access mode. Word access mode allows addressing of 64kWords of memory while byte access mode allows addressing of 64kBytes of memory.

Each data pointer (DP[n]) and Frame Pointer base (BP) register is actually implemented internally as a 17-bit register (e.g., 16:0). The Frame Pointer offset register (OFFS) is implemented internally as a 9-bit register (e.g., 8:0). The WBSn bit for the respective pointer controls whether the highest 16 bits (16:1) of the pointer are in use, as is the case for word mode (WBSn = 1) or whether the lowest 16 bits (15:0) are in use, as will be the case for byte mode (WBSn = 0). The WBS2 bit also controls whether the high 8 bits (8:1) of the offset register are in use (WBS2 = 1) or the low 8 bits (7:0) are used (WBS2 = 0). All data pointer register reads, writes, auto-increment/decrement operations occur with respect to the current WBSn selection. Data pointer increment and decrement operations only affect those bits specific to the current word or byte addressing mode (e.g., incrementing a byte mode data pointer from FFFFh does not carry into the internal high order bit that is utilized only for word mode data pointer access). Switching from byte to word access mode or vice versa does not alter the data pointer contents. Therefore, it is important to maintain the consistency of data pointer address value within the given access mode.

```
move   DPC, #0               ; DP[0] in byte mode
move   DP[0], #2345h         ; DP[0]=2345h (byte mode)
                             ; internal  bits 15:0 loaded
move   DPC, #4               ; DP[0] in word mode
move   DP[0], #2345h         ; DP[0]=2345h (word mode)
                             ; internal bits 16:1 loaded
move   DPC, #0               ; DP[0] in byte mode
move   GR, DP[0]             ; GR = 468Bh (looking at bits 15:0)
```

The three pointers share a single read/write port on the data memory and thus, the user must knowingly activate a desired pointer before using it for data memory read operations. This can be done explicitly using the data pointer select bits (SDPS1:0; DPC.1:0), or implicitly by writing to the DP[n], BP, or OFFS registers as shown below. Any indirect memory write operation using a data pointer will set the SDPS bits, thus activating the write pointer as the active source pointer.

```
move   DPC, #2               ; (explicit) selection of FP as the pointer
move   DP[1], DP[1]          ; (implicit) selection of DP[1]; set SDPS1:0=01b
move   OFFS, src             ; (implicit) selection of FP; set SDPS1=1
move   @DP[0], src           ; (implicit) selection of DP[0]; set SDPS1:0=00b
```

Once the pointer selection has been made, it will remain in effect until:

- the source data pointer select bits are changed via the explicit or implicit methods described above (i.e., another data pointer is selected for use)
- the memory to which the active source data pointer is addressing is enabled for code fetching using the Instruction Pointer, or
- a memory write operation is performed using a data pointer other than the current active source pointer.

```
move   DP[1], DP[1]          ; select DP[1] as the active pointer
move   dst, @DP[1]           ; read from pointer
move   @DP[1], src           ; write using a data pointer
                             ; DP[0] is needed
move   DP[0], DP[0]          ; select DP[0] as the active pointer
```

To simplify data pointer increment/decrement operations without disturbing register data, a virtual NUL destination has been assigned to system module 6, sub-index 7 to serve as a bit bucket. Data pointer increment/decrement operations can be done as follows without altering the contents of any other register:

```
move   NUL, @DP[0]++         ; increment DP[0]
move   NUL, @DP[0]--         ; decrement DP[0]
```

The following data pointer related instructions are invalid:

```
move @++DP[0], @DP[0]++
move @++DP[1], @DP[1]++
move @BP[++Offs], @BP[Offs++]
move @--DP[0], @DP[0]--
move @--DP[1], @DP[1]--
move @BP[--Offs], @BP[Offs--]
move @++DP[0], @DP[0]--
move @++DP[1], @DP[1]--
move @BP[++Offs], @BP[Offs--]
move @--DP[0], @DP[0]++
move @--DP[1], @DP[1]++
move @BP[--Offs], @BP[Offs++]
move @DP[0], @DP[0]++
move @DP[1], @DP[1]++
move @BP[Offs], @BP[Offs++]
move @DP[0], @DP[0]--
move @DP[1], @DP[1]--
move @BP[Offs], @BP[Offs--]
move DP[0], @DP[0]++
move DP[0], @DP[0]--
move DP[1], @DP[1]++
move DP[1], @DP[1]--
move Offs, @BP[Offs--]
move Offs, @BP[Offs++]
```

3.11 Using the Watchdog Timer

The Watchdog Timer is a user-programmable clock counter that can serve as a time-base generator, an event timer, or a system supervisor. As can be seen in the diagram below, the timer is driven by the main system clock and is supplied to a series of dividers. If the watchdog interrupt and the watchdog reset are disabled (EWDI = 0 and EWT = 0), the watchdog timer and its input clock are disabled. Whenever the watchdog timer is disabled, the watchdog interval timer (per WD1:0 bits) and 512 clock reset counter will be reset if either the interrupt or reset function is enabled. When the watchdog timer is initially enabled, there will be a 1-clock to 3-clock cycle delay before it starts. The divider output is selectable, and determines the interval between timeouts. When the timeout is reached, an interrupt flag is set, and if enabled, an interrupt occurs. A watchdog-reset function is also provided in addition to the watchdog interrupt. The reset and interrupt are completely discrete functions that can be acknowledged or ignored, together or separately for various applications.

Table 3-2. Watchdog Timer Register Control Bits

BIT NAME	DESCRIPTION	REGISTER LOCATION	BIT POSITION
EWDI	Enable Watchdog Timer Interrupt	WDCN (Fh, 8h)	WDCN.6
WD1, WD0	Watchdog Interval Control Bits		WDCN.5,4
WDIF	Watchdog Interrupt Flag		WDCN.3
WTRF	Watchdog Timer Reset Flag		WDCN.2
EWT	Enable Watchdog Timer Reset		WDCN.1
RWT	Reset Watchdog Timer		WDCN.0

The Watchdog Timer Reset function works as follows. After initializing the correct timeout interval (discussed below), software can enable, if desired, the reset function by setting the Enable Watchdog Timer Reset (EWT = WDCN.1) bit. Setting the EWT bit will reset/restart the Watchdog timer if the Watchdog interrupt is not already enabled. At any time prior to reaching its user selected terminal value, software can set the Reset Watchdog Timer (RWT = WDCN.0) bit. If the Watchdog Timer is reset (RWT bit written to a logic 1) before the timeout period expires, the timer will start over. Hardware will automatically clear RWT after software sets it.

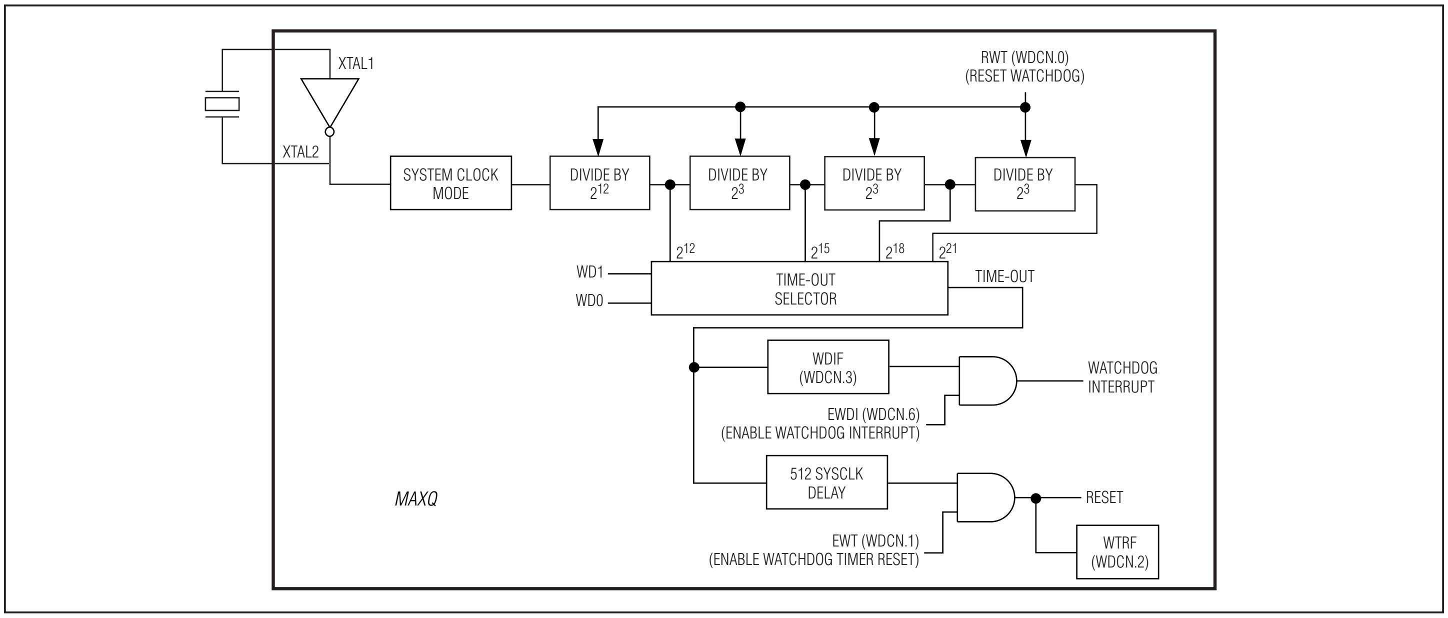


Figure 3-1. Watchdog Timer Block Diagram

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.

Table 3-3. Watchdog Timeout Period Selection

SYSTEM CLOCK MODE	SYSTEM CLOCK SELECT BITS PMME, CD1, CD0	WATCHDOG TIMEOUT (IN NUMBER OF OSCILLATOR CLOCKS)			
		WD1:0 = 00b	WD1:0 = 01b	WD1:0 = 10b	WD1:0 = 11b
Divide by 1 (default)	000	2^{12}	2^{15}	2^{18}	2^{21}
Divide by 2	001	2^{13}	2^{16}	2^{19}	2^{22}
Divide by 4	010	2^{14}	2^{17}	2^{20}	2^{23}
Divide by 8	011	2^{15}	2^{18}	2^{21}	2^{24}
Power Management Mode (Divide by 256)	1xx	2^{20}	2^{23}	2^{26}	2^{29}

Table 3-4. System Register Map

REGISTER INDEX WITHIN MODULE	MODULE SPECIFIER									
	6h	7h	8h	9h	Ah	Bh	Ch	Dh	Eh	Fh
0h	—	—	AP	**A[0]**	**Acc**	**PFX[0]**	**IP**	—	—	—
1h	—	—	APC	**A[1]**	***A[AP]***	**PFX[1]**	—	**SP**	—	—
2h	—	—	—	**A[2]**	—	**PFX[2]**	—	**IV**	—	—
3h	—	—	—	**A[3]**	—	**PFX[3]**	—	—	OFFS	**DP[0]**
4h	—	—	PSF	**A[4]**	—	**PFX[4]**	—	—	**DPC**	—
5h	—	—	IC	**A[5]**	—	**PFX[5]**	—	—	**GR**	—
6h	—	—	IMR	**A[6]**	—	**PFX[6]**	—	**LC[0]**	GRL	—
7h	—	—	—	**A[7]**	—	**PFX[7]**	—	**LC[1]**	**BP**	**DP[1]**
8h	—	—	SC	**A[8]**	—	—	—	—	***GRS***	—
9h	—	—	—	**A[9]**	—	—	—	—	GRH	—
Ah	—	—	—	**A[10]**	—	—	—	—	***GRXL***	—
Bh	—	—	*IIR*	**A[11]**	—	—	—	—	***FP***	—
Ch	—	—	—	**A[12]**	—	—	—	—	—	—
Dh	—	—	—	**A[13]**	—	—	—	—	—	—
Eh	—	—	CKCN	**A[14]**	—	—	—	—	—	—
Fh	—	—	WDCN	**A[15]**	—	—	—	—	—	—
10h	—	—	—	—	—	—	—	—	—	—
11h	—	—	—	—	—	—	—	—	—	—
12h	—	—	—	—	—	—	—	—	—	—
13h	—	—	—	—	—	—	—	—	—	—
14h	—	—	—	—	—	—	—	—	—	—
15h	—	—	—	—	—	—	—	—	—	—
16h	—	—	—	—	—	—	—	—	—	—
17h	—	—	—	—	—	—	—	—	—	—
18h	—	—	—	—	—	—	—	—	—	—
19h	—	—	—	—	—	—	—	—	—	—
1Ah	—	—	—	—	—	—	—	—	—	—
1Bh	—	—	—	—	—	—	—	—	—	—
1Ch	—	—	—	—	—	—	—	—	—	—
1Dh	—	—	—	—	—	—	—	—	—	—
1Eh	—	—	—	—	—	—	—	—	—	—
1Fh	—	—	—	—	—	—	—	—	—	—

Note: *Registers in italics are read-only. Registers in bold are 16-bit (except A[n], Acc, A[AP] for MAXQ10). Registers with indexes 8h and higher can only be accessed as destinations by using the prefix register. Similarly, registers with indexes 10h and higher can only be accessed as sources through the prefix register. All undefined or unused indexes (indicated by an em-dash ‘—‘) are either used for op code implementation or reserved for future expansion, and should not be accessed explicitly. Accessing these locations as registers can have deterministic effects, but the effects will probably not be the intended ones.*

Table 3-5. System Register Bit Map

REGISTER	BIT POSITION															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AP									—	—	—	—	AP (4 bits)			
APC									CLR	IDS	—	—	—	MOD2	MOD1	MOD0
PSF									Z	S	—	GPF1	GPF0	OV	C	E
IC									—	—	CGDS	—	—	—	INS	IGE
IMR									IMS	—	IM5	IM4	IM3	IM2	IM1	IM0
SC									TAP	—	CDA1	CDA0	UPA	ROD	PWL	—
IIR									IIS	—	II5	II4	II3	II2	II1	II0
CKCN									XT/$\overline{\text{RC}}$	RGSL	RGMD	STOP	SWB	PMME	CD1	CD0
WDCN									POR	EWDI	WD1	WD0	WDIF	WTRF	EWT	RWT
A[0]	A[0] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[1]	A[1] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[2]	A[2] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[3]	A[3] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[4]	A[4] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[5]	A[5] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[6]	A[6] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[7]	A[7] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[8]	A[8] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[9]	A[9] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[10]	A[10] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[11]	A[11] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[12]	A[12] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[13]	A[13] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[14]	A[14] (MAXQ10: 8 bits; MAXQ20:16 bits)															
A[15]	A[15] (MAXQ10: 8 bits; MAXQ20:16 bits)															
PFX[n]	PFX[n] (16 bits)															
IP	IP (16 bits)															
SP	—	—	—	—	—	—	—	—	—	—	—	—	SP (4 bits)			
IV	IV (16 bits)															
LC[0]	LC[0] (16 bits)															
LC[1]	LC[1] (16 bits)															
OFFS									OFFS (8 bits)							
DPC	—	—	—	—	—	—	—	—	—	—	—	WBS2	WBS1	WBS0	SDPS1	SDPS0
GR	GR (16 bits)															
GRL									GRL (8 bits)							
BP	BP (16 bits)															
GRS	GRS (16 bits) = (GRL, GRH)															
GRH									GRH (8 bits)							
GRXL	GRXL (16 bits) = (GRL.7, 8bits): (GRL, 8bits)															
FP	FP = BP[Offs] (16 bits)															
DP[0]	DP[0] (16 bits)															
DP[1]	DP[1] (16 bits)															

Table 3-6. System Register Bit Reset Values

REGISTER	BIT POSITION															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AP									0	0	0	0	0	0	0	0
APC									0	0	0	0	0	0	0	0
PSF									1	0	0	0	0	0	0	0
IC									0	0	0	0	0	0	0	0
IMR									0	0	0	0	0	0	0	0
SC									1	0	0	0	0	0	s	0
IIR									0	0	0	0	0	0	0	0
CKCN									s	s	s	0	0	0	0	0
WDCN									s	s	0	0	0	s	s	0
A[n] MAXQ10									0	0	0	0	0	0	0	0
A[n] MAXQ20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PFX	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IP	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SP	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
IV	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LC[0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LC[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OFFS									0	0	0	0	0	0	0	0
DPC MAXQ10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DPC MAXQ20	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
GR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GRL									0	0	0	0	0	0	0	0
BP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GRS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GRH									0	0	0	0	0	0	0	0
GRXL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DP[0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DP[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: *Bits marked ‘s’ are static across some or all resets.*

SECTION 4: SYSTEM REGISTER DESCRIPTIONS

This section contains the following information:

SECTION 4: SYSTEM REGISTER DESCRIPTIONS

Those registers currently defined in the MAXQ System Register map are described in the following pages. The addresses for each register are given in the format *module[index]*, where *module* is the module specifier from 8h to Fh and *index* is the register sub-index from 0h to Fh.

4.1 Accumulator Pointer Register (AP, 8h[0h])

The bit definitions are for 16 accumulators.
Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
AP.3 to AP.0	Active Accumulator Select. These bits select which of the 16 accumulator registers are used for arithmetic and logical operations. If the APC register has been set to perform automatic increment/decrement of the active accumulator, this setting will be automatically changed after each arithmetic or logical operation. If a 'MOVE AP, Acc' instruction is executed, any enabled AP inc/dec/modulo control will take precedence over the transfer of Acc data into AP.
AP.7 to AP.4	Reserved. All reads return 0.

4.2 Accumulator Pointer Control Register (APC, 8h[1h])

The bit definitions are for 16 accumulators.
Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION	
APC.2 to APC.0 (MOD2 to MOD0)	Accumulator Pointer Auto Increment/Decrement Modulus. If these bits are set to a nonzero value, the accumulator pointer (AP[3:0]) will be automatically incremented or decremented following each arithmetic or logical operation. The mode for the auto-increment/decrement is determined as follows:	
	MOD[2:0]	AUTO INCREMENT/DECREMENT MODE
	000	No auto-increment/decrement (default)
	001	Increment/decrement AP[0] modulo 2
	010	Increment/decrement AP[1:0] modulo 4
	011	Increment/decrement AP[2:0] modulo 8
	100	Increment/decrement AP modulo 16
	101 to 111	Reserved (modulo 16 when set)
APC.5 to APC.3	Reserved. All reads return 0.	
APC.6 (IDS)	Increment/Decrement Select. If this bit is set to 0, the accumulator pointer AP is incremented following each arithmetic or logical operation according to MOD[2:0]. If this bit is set to 1, the accumulator pointer AP is decremented following each arithmetic or logical operation according to MOD[2:0]. If MOD[2:0] is set to 000, the setting of this bit is ignored.	
APC.7 (CLR)	AP Clear. Writing this bit to 1 clears the accumulator pointer AP to 0. Once set, this bit will automatically be reset to 0 by hardware. If a 'MOVE APC, Acc' instruction is executed requesting that AP be set to 0 (i.e., CLR = 1), the AP clear function overrides any enabled inc/dec/modulo control. All reads from this bit return 0.	

4.3 Processor Status Flags Register (PSF, 8h[4h])

The OV and S bit definitions are given for the MAXQ20 (16-bit accumulators and ALU).
Initialization: This register is cleared to 80h on all forms of reset.
Access: Bit 7 (Z), bit 6 (S), and bit 2 (OV) are read only. Bits 4 and 3 (GPF1,GPF0), bit 1 (C), and bit 0 (E) are unrestricted read/write.

BIT	FUNCTION
PSF.0 (E)	Equals Flag. This bit flag is set to 1 whenever a compare operation (CMP) returns an equal result. If a CMP operation returns not equal, this bit is cleared.
PSF.1 (C)	Carry Flag. This bit flag is set to 1 whenever an add or subtract operation (ADD, ADDC, SUB, SUBB) returns a carry or borrow. This bit flag is cleared to 0 whenever an add or subtract operation does not return a carry or borrow. Many other instructions potentially affect the carry bit. Reference the instruction set documentation for details.
PSF.2 (OV)	Overflow Flag. This flag is set to 1 if there is a carry out of bit 14 but not out of bit 15, or a carry out of bit 15 but not out of bit 14 from the last arithmetic operation, otherwise, the OV flag remains as 0. OV indicates a negative number resulted as the sum of two positive operands, or a positive sum resulted from two negative operands.
PSF.3 (GPF0)	General Purpose Flag 0
PSF.4 (GPF1)	General Purpose Flag 1. These general-purpose flag bits are provided for user software control.
PSF.5	Reserved. All reads return 0.
PSF.6 (S)	Sign Flag. This bit flag mirrors the current value of the high bit of the active accumulator (Acc.15).
PSF.7 (Z)	Zero Flag. The value of this bit flag equals 1 whenever the active accumulator is equal to zero, and it equals 0 otherwise.

4.4 Interrupt and Control Register (IC, 8h[5h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
IC.0 (IGE)	Interrupt Global Enable. If this bit is set to 1, interrupts are globally enabled, but still must be locally enabled to occur. If this bit is set to 0, all interrupts are disabled.
IC.1 (INS)	Interrupt In Service. The INS is set by hardware automatically when an interrupt is acknowledged. No further interrupts occur as long as the INS remains set. The interrupt service routine can clear the INS bit to allow interrupt nesting. Otherwise, the INS bit is cleared by hardware upon execution of an RETI or POPI instruction.
IC.4 to IC.2	Reserved. All reads return 0.
IC.5 (CGDS)	System Clock Gating Disable. If this bit is set to 0 (default mode), system clock gating circuitry is active. If this bit is set to 1, the clock gating circuitry is disabled.
IC.7, IC.6	Reserved. All reads return 0.

4.5 Interrupt Mask Register (IMR, 8h[6h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted read/write access.

BIT	FUNCTION
	The first six bits in this register are interrupt mask bits for modules 0 to 5, one bit per module. The eighth bit, IMS, serves as a mask for any system module interrupt sources. Setting a mask bit allows the enabled interrupt sources for the associated module or system (for the case of IMS) to generate interrupt requests. Clearing the mask bit effectively disables all interrupt sources associated with that specific module or all system interrupt sources (for the case of IMS). The interrupt mask register is intended to facilitate user-definable interrupt prioritization.
IMR.0 (IM0)	Interrupt Mask for Register Module 0
IMR.1 (IM1)	Interrupt Mask for Register Module 1
IMR.2 (IM2)	Interrupt Mask for Register Module 2
IMR.3 (IM3)	Interrupt Mask for Register Module 3
IMR.4 (IM4)	Interrupt Mask for Register Module 4
IMR.5 (IM5)	Interrupt Mask for Register Module 5
IMR.6	Reserved. Reads return 0.
IMR.7 (IMS)	Interrupt Mask for System Modules

4.6 System Control Register (SC, 8h[8h])

Initialization: This register is reset to 100000s0b on all reset. Bit 1 (PWL) is set to 1 on a power-on reset only.
Access: Unrestricted read/write access.

BIT	FUNCTION
SC.0	Reserved. All reads return 0.
SC.1 (PWL)	Password Lock. This bit defaults to 1 on a power-on reset. When this bit is 1, it requires a 32-byte password to be matched with the password in the program space before allowing access to the password protected in-circuit debug or bootstrap loader ROM routines. Clearing this bit to 0 disables the password protection for these ROM routines.
SC.2 (ROD)	ROM Operation Done. This bit is used to signify completion of a ROM operation sequence to the control units. This allows the Debug engine to determine the status of a ROM sequence. Setting this bit to logic 1 causes an internal system reset if the JTAG SPE bit is also set. Setting the ROD bit will clear the JTAG SPE bit if it is set and the ROD bit will be automatically cleared by hardware once the control unit acknowledges the done indication.
SC.3 (UPA)	Upper Program Access. The physical program memory is logically divided into four pages; P0 and P1 occupy the lower 32kWords while P2 and P3 occupy the upper 32kWords. P0 and P1 are assigned to the lower half of the program space and are always active. P2 and P3 must be explicitly activated in the upper half of the program space by setting the UPA bit to 1. When UPA bit is cleared to 0, the upper program memory space is occupied by the Utility ROM and the logical data memory, which is accessible as program memory. Note that the UPA is not implemented if the upper 32K of the program space is not used for the user code.
SC.5 and SC.4 (CDA1, CDA0)	Code Data Access Bits 1:0. The CDA bits are used to logically map physical program memory page to the data space for read/write access:

CDA1:0	BYTE MODE ACTIVE PAGE	WORD MODE ACTIVE PAGE
00	P0	P0 and P1
01	P1	P0 and P1
10	P2	P2 and P3
11	P3	P2 and P3

BIT	FUNCTION
SC.5 and SC.4 (CDA1, CDA0)	The logical data memory addresses of the program pages depend on whether execution is from Utility ROM or logical data memory. Note that CDA1 is not implemented if the upper 32k of the program space is not used for the user code. No CDA bits are needed if only one page of program space is incorporated.
SC.6	Reserved. All reads return 0.
SC.7 (TAP)	Test Access (JTAG) Port Enable. This bit controls whether the Test Access Port special-function pins are enabled. The TAP defaults to being enabled. Clearing this bit to 0 disables the TAP special function pins.

4.7 Interrupt Identification Register (IIR, 8h[Bh])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Read only.

BIT	FUNCTION
	The first six bits in this register indicate interrupts pending in modules 0 to 5, one bit per module. The eighth bit, IIS, indicates a pending system interrupt, such as from the watchdog timer. The interrupt pending flags will be set only for enabled interrupt sources waiting for service. The interrupt pending flag will be cleared when the pending interrupt sources within that module are disabled or when the interrupt flags are cleared by software
IIR.0 (II0)	Interrupt Identifier Flag for Register Module 0
IIR.1 (II1)	Interrupt Identifier Flag for Register Module 1
IIR.2 (II2)	Interrupt Identifier Flag for Register Module 2
IIR.3 (II3)	Interrupt Identifier Flag for Register Module 3
IIR.4 (II4)	Interrupt Identifier Flag for Register Module 4
IIR.5 (II5)	Interrupt Identifier Flag for Register Module 5
IIR.6	Reserved. Reads return 0.
IIR.7 (IIS)	Interrupt Identifier Flag for System Modules

4.8 System Clock Control Register (CKCN, 8h[Eh])

Initialization: Bits 4:0 are cleared to zero on all forms of reset. See bit description for bits 7:5.
Access: Bit 5 (RGMD) is read-only. All other bits are unrestricted read/write, except for the locking mechanism on CD0 and CD1 as described below.

BIT	FUNCTION
CKCN.0 (CD0); CKCN.1 (CD1)	Clock Divide Bit 0. Clock Divide Bit 1. If the PMME bit is cleared, the CD0 and CD1 bits control the number of oscillator clocks required to generate one system clock as follows:

CD1	CD0	OSCILLATOR CLOCK CYCLES PER SYSTEM CLOCK CYCLE
0	0	1 (default)
0	1	2
1	0	4
1	1	8

BIT	FUNCTION
CKCN.0 (CD0); CKCN.1 (CD1)	If the PMME bit is set to 1, the values of CD0 and CD1 may not be altered and do not affect the system clock frequency.
CKCN.2 (PMME)	Power Management Mode Enable. If the PMME bit is cleared to 0, the values of CD0 and CD1 determine the number of oscillator clock cycles per system clock cycle. If the PMME bit is set to 1, the values of CD0 and CD1 are ignored and the system clock operates in a fixed mode of 1 cycle per 256 oscillator cycles (divide by 256). If the PMME bit is set to 1 and Switchback mode has been enabled (SWB = 1), when a Switchback source (such as an enabled external interrupt) becomes active, PMME will be cleared to 0 and cannot be set to 1 unless all Switchback sources are inactive.
CKCN.3 (SWB)	Switchback Enable. If the SWB bit is cleared to 0, Switchback mode is not active. If the SWB bit is set to 1, Switchback mode is active. Switchback mode has no effect if Power Management Mode is not active (PMME = 0). If Power Management Mode is active and Switchback mode is enabled, the PMME bit will be cleared to 0 when any of the following conditions occur. 1) An external interrupt condition occurs on an INTx pin and the corresponding external interrupt is enabled. 2) An active-low transition occurs on the UART serial receive-input line (modes 1, 2, and 3) and data reception is enabled. 3) The SBUF register is written to send an outgoing byte through the UART and transmission is enabled 4) The SPIB register is written in master mode (STBY = 1) to send an outgoing character through the SPI module and transmission is enabled. 5) The SPI module's $\overline{SSEL}$ signal is asserted in slave mode. 6) Time-of-Day and Subsecond interval alarms from the RTC when enabled. 7) Active debug mode is entered either by break point match or issuance of the 'Debug' command from background mode. When any of these conditions cause Switchback to clear PMME to 0, the system clock rate will then be determined by the settings of CD0 and CD1. After PMME is cleared to 0 by Switchback, it may not be set back to 1 as long as any of the above conditions are true.
CKCN.4 (STOP)	Stop Mode Select. Setting this bit to 1 causes the MAXQ to enter Stop Mode. This will not change the currently selected clock divide ratio (CD0, CD1, PMME).
CKCN.5 (RGMD)	Ring Oscillator Mode. This read-only bit reflects the selection of clock source. RGMD = 1 indicates the ring oscillator is providing the system clock. RGMD = 0 indicates that the clock source specified by the XT/$\overline{RC}$ bit is being used for system clock generation. If the given MAXQ device does not include an internal ring oscillator from which it can run, this read-only bit will track the value of CKCN.6.
CKCN.6 (RGSL)	Ring Oscillator Select. This bit selects the internal ring oscillator for system clock generation. When RGSL is set to 1, the internal ring oscillator (following the PMME, CD1:0 selected divide ratio) is immediately sourced as the system clock and the internal crystal amplifier is disabled (if allowed). When RGSL is cleared to 0, the internal ring oscillator (following the clock divide selection) will continue to serve as the system clock until the warm-up counter associated with the XT/$\overline{RC}$ clock selection has expired. At which point, that clock source (following the PMME, CD1:0 selected divide ratio) is sourced as the system clock. The RGSL bit is cleared to 0 on power-on reset only and is unaffected by other resets. If the given MAXQ device does not include an internal ring oscillator from which it can run, this bit can be used as a general-purpose read/write bit.
CKCN.7(XT/$\overline{RC}$)	Crystal/RC Oscillator Select. This bit selects the non-ring oscillator mode that may be used for system clock generation. The XT/$\overline{RC}$ bit can only be modified when RGSL = 1. The XT/ $\overline{RC}$ bit is set to 1 on power-on reset only and is unaffected by other resets. XT/ $\overline{RC}$ = 0: external RC configuration XT/ $\overline{RC}$ = 1: external crystal/clock configuration If the given MAXQ device does not support both the crystal and RC options, this bit can be used as a general-purpose read/write bit that is write protected when CKCN.6 is configured to 0.

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.

BIT	FUNCTION
WDCN.0 (RWT)	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.
WDCN.1 (EWT)	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.
WDCN.2 (WTRF)	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.
WDCN.3 (WDIF)	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.
WDCN.4 (WD0); WDCN.5 (WD1)	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.

WD1	WD0	CLOCKS UNTIL INTERRUPT	CLOCKS UNTIL RESET
0	0	2^{12}	$2^{12} + 512$
0	1	2^{15}	$2^{15} + 512$
1	0	2^{18}	$2^{18} + 512$
1	1	2^{21}	$2^{21} + 512$

WDCN.6 (EWDI)	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.
WDCN.7 (POR)	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.

4.10 (MAXQ10) Accumulator n Register (A[n], 9h[nh])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
A[n].7 to A[n].0	This register acts as the accumulator for all ALU arithmetic and logical operations when selected by the accumulator pointer (AP). It can also be used as a general-purpose working register.

4.11 (MAXQ20) Accumulator n Register (A[n], 9h[nh])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
A[n].15 to A[n].0	This register acts as the accumulator for all ALU arithmetic and logical operations when selected by the accumulator pointer (AP). It can also be used as a general-purpose working register.

4.12 Prefix Register (PFX[n], Bh[n])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
PFX[n].15 to PFX[n].0	The Prefix register provides a means of supplying an additional 8 bits of high-order data for use by the succeeding instruction as well as providing additional indexing capabilities. This register will only hold any data written to it for one execution cycle, after which it will revert to 0000h. Although this is a 16-bit register, only the lower 8 bits are actually used for prefixing purposes by the next instruction. Writing to or reading from any index in the Prefix module will select the same 16-bit register. However, when the Prefix register is written, the index *n* used for the PFX[*n*] write also determines the high-order bits for the register source and destination specified in the following instruction.

WRITE TO	SOURCE, DESTINATION INDEX SELECTION	
	SOURCE REGISTER RANGE	DESTINATION REGISTER RANGE
PFX[0]	0h to Fh	0h to 7h
PFX[1]	10h to 1Fh	0h to 7h
PFX[2]	0h to Fh	8h to Fh
PFX[3]	10h to 1Fh	8h to Fh
PFX[4]	0h to Fh	10h to 17h
PFX[5]	10h to 1Fh	10h to 17h
PFX[6]	0h to Fh	18h to 1Fh
PFX[7]	10h to 1Fh	18h to 1Fh

The index selection reverts to 0 (default mode allowing selection of registers 0h to 7h for destinations) after one cycle in the same manner as the contents of the Prefix register.

4.13 Instruction Pointer Register (IP, Ch[0h])

Initialization: This register is cleared to 8000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
IP.15 to IP.0	This register contains the address of the next instruction to be executed and is automatically incremented by 1 after each program fetch. Writing an address value to this register will cause program flow to jump to that address. Reading from this register will not affect program flow.

4.14 Stack Pointer Register (SP, Dh[1h])

Bits defined below for 16-word stack depth.
Initialization: This register is cleared to 000Fh on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
SP.3 to SP.0	These four bits indicate the current top of the hardware stack, from 0h to Fh. This pointer is incremented after a value is pushed on the stack and decremented before a value is popped from the stack.
SP.15 to SP.4	Reserved; all reads return 0.

4.15 Interrupt Vector Register (IV, Dh[2h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
IV.15 – IV.0	This register contains the address of the interrupt service routine. The interrupt handler will generate a CALL to this address whenever an interrupt is acknowledged.

4.16 Loop Counter 0 Register (LC[0], Dh[6h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
LC[0].15 to LC[0].0	This register is used as the loop counter for the DJNZ LC[0], src operation. This operation decrements LC[0] by one and then jumps to the address specified in the instruction by src.

4.17 Loop Counter 1 Register (LC[1], Dh[7h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
LC[1].15 to LC[1].0	This register is used as the loop counter for the DJNZ LC[1], src operation. This operation decrements LC[1] by one and then jumps to the address specified in the instruction by src.

4.18 Frame Pointer Offset Register (OFFS, Eh[3h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
OFFS.7 to OFFS.0	This 8-bit register provides the Frame Pointer (FP) offset from the base pointer (BP). The Frame Pointer is formed by unsigned addition of Frame Pointer Base Register (BP) and Frame Pointer Offset Register (Offs). The contents of this register can be post-incremented or post-decremented when using the Frame Pointer for read operations and may be pre-incremented or pre-decremented when using the Frame Pointer for write operations. A carry out or borrow resulting from an increment/decrement operation has no effect on the Frame Pointer Base Register (BP).

4.19 Data Pointer Control Register (DPC, Eh[4h])

Initialization: (MAXQ10) This register is cleared to 0000h on all forms of reset.
(MAXQ20) This register is cleared to 001Ch on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION		
DPC.1 to DPC.0 (SDPS1, SDPS0)	Source Data Pointer Select Bits 1:0. These bits select one of the three data pointers as the active source pointer for the load operation. A new data pointer must be selected before being used to read data memory:		
	SDPS1	SDPS0	SOURCE POINTER SELECTION
	0	0	DP[0]
	0	1	DP[1]
	1	0	FP (BP[Offs])
	1	1	Reserved (select FP if set)
	These bits default to 00b but do not activate DP[0] as an active source pointer until the SDPS bits are explicitly cleared to 00b or the DP[0] register is written by an instruction. Also, modifying the register contents of a data/frame pointer register (DP[0], DP[1], BP or Offs) will change the setting of the SDPS bits to reflect the active source pointer selection.		
DPC.2 (WBS0)	Word/Byte Select 0. This bit selects access mode for DP[0]. When WBS0 is set to logic 1, the DP[0] is operated in word mode for data memory access; when WBS0 is cleared to logic 0, DP[0] is operated in byte mode for data memory access.		
DPC.3 (WBS1)	Word/Byte Select 1. This bit selects access mode for DP[1]. When WBS1 is set to logic 1, the DP[1] is operated in word mode for data memory access; when WBS1 is cleared to logic 0, DP[1] is operated in byte mode for data memory access.		
DPC.4 (WBS2)	Word/Byte Select 2. This bit selects access mode for BP[Offs]. When WBS2 is set to logic 1, the BP[Offs] is operated in word mode for data memory access; when WBS2 is cleared to logic 0, BP[Offs] is operated in byte mode for data memory access.		
DPC.15 to DPC.5	Reserved. Read returns 0.		

4.20 General Register (GR, Eh[5h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
GR.15 to GR.0	This register is intended primarily for supporting byte operations on 16-bit data. The 16-bit register is byte-readable, byte-writeable through the corresponding GRL and GRH 8-bit registers and byte-swappable through the GRS 16-bit register.

4.21 General Register Low Byte (GRL, Eh[6h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
GRL.7 to GRL.0	This register reflects the low byte of the GR register and is intended primarily for supporting byte operations on 16-bit data. Any data written to the GRL register will also be stored in the low byte of the GR register.

4.22 Frame Pointer Base Register (BP, Eh[7h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
BP.15 to BP.0	This register serves as the base pointer for the Frame Pointer (FP). The Frame Pointer is formed by unsigned addition of Frame Pointer Base Register (BP) and Frame Pointer Offset Register (Offs). The content of this base pointer register is not affected by increment/decrement operations performed on the offset (OFFS) register.

4.23 General Register Byte-Swapped (GRS, Eh[8h])

Initialization: This register is cleared to 0000h on all forms of reset
Access: Unrestricted read-only access.

BIT	FUNCTION
GRS.15 to GRS.0	This register is intended primarily for supporting byte operations on 16-bit data. This 16-bit read only register returns the byte-swapped value for the data contained in the GR register.

4.24 General Register High Byte (GRH, Eh[9h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
GRH.7 to GRH.0	This register reflects the high byte of the GR register and is intended primarily for supporting byte operations on 16-bit data. Any data written to the GRH register will also be stored in the high byte of the GR register.

4.25 General Register Sign Extended Low Byte (GRXL, Eh[Ah])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read-only access.

BIT	FUNCTION
GRXL.15 to GRXL.0	This register provides the sign extended low byte of GR as a 16-bit source.

4.26 Frame Pointer Register (FP, Eh[Bh])

Initialization: This register is cleared to 0000h on all forms of reset.
Access" Unrestricted direct read-only access.

BIT	FUNCTION
FP.15 to FP.0	This register provides the current value of the frame pointer (BP[Offs]).

4.27 Data Pointer 0 Register (DP[0], Fh[3h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
DP[0].15 to DP[0].0	This register is used as a pointer to access data memory. DP[0] can be automatically incremented or decremented following each read operation or can be automatically incremented or decremented before each write operation.

4.28 Data Pointer 1 Register (DP[1], Fh[7h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	FUNCTION
DP[1].15 to DP[1].0	This register is used as a pointer to access data memory. DP[1] can be automatically incremented or decremented following each read operation or can be automatically incremented or decremented before each write operation.

SECTION 5: PERIPHERAL REGISTER MODULES

The MAXQ microcontroller uses Peripheral Registers to control and monitor peripheral modules. These registers reside in Modules 0h through 5h, with sub-index values 0h to 1Fh. While the peripherals must reside in modules 0h through 5h, they are not necessarily tied to specific index numbers inside this range and can be moved, removed, and/or duplicated for certain MAXQ-based products as space permits. For this reason, each peripheral modules and its associated registers/bits are covered separately. Consult the individual device data sheet or user's guide supplement for the exact peripheral register map.

SECTION 6: GENERAL-PURPOSE I/O MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

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.

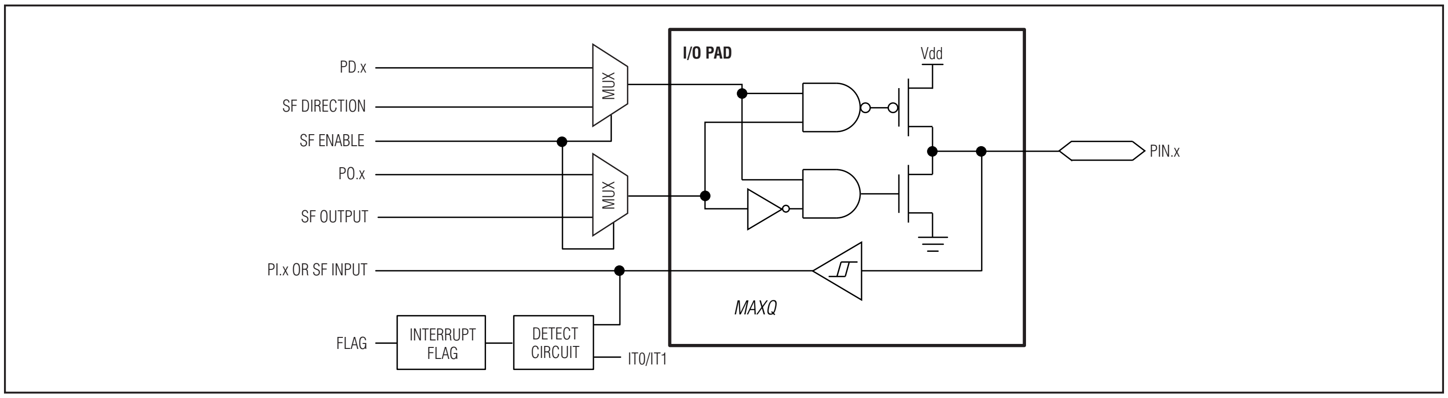


Figure 6-1. Type A Port Pin Schematic

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.

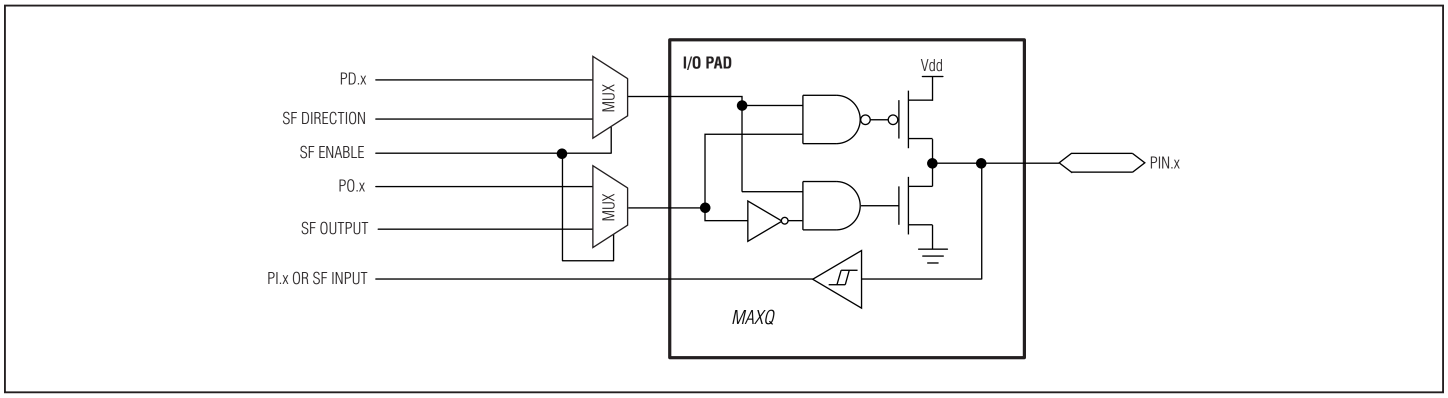


Figure 6-2. Type B Port Pin Schematic

6.3 I/O Port: Type C

The Type C I/O port is nearly identical to the Type B I/O port, but with the addition of a selectable internal, weak, P-channel, pullup device. The weak pullup device can be enabled by configuring the port pin as an input and setting the associated port output bit to logic 1 (default reset state).

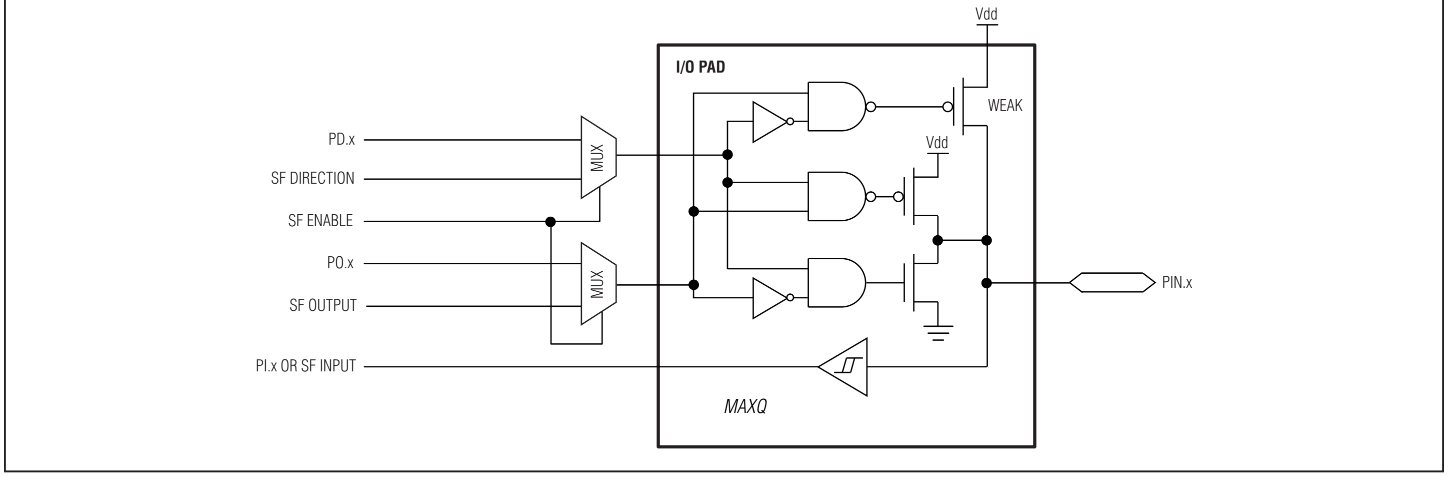


Figure 6-3. Type C Port Pin Schematic

Table 6-1. Weak Pullup Control

PDX.x (PORT DIRECTION BIT) OR SF SELECT (SF I/O CONTROL)	POX.x (PORT OUTPUT BIT)	WEAK PULLUP
X	0	OFF
1 = output	1	OFF
0 = input	1	ON

6.4 I/O Port: Type D

The Type D I/O port merges the Type C I/O port with the interrupt functionality of the Type A I/O port. Just like the Type C I/O port, the weak pullup device can be enabled by configuring the port pin as an input and setting the associated port output bit to logic 1 (default reset state).

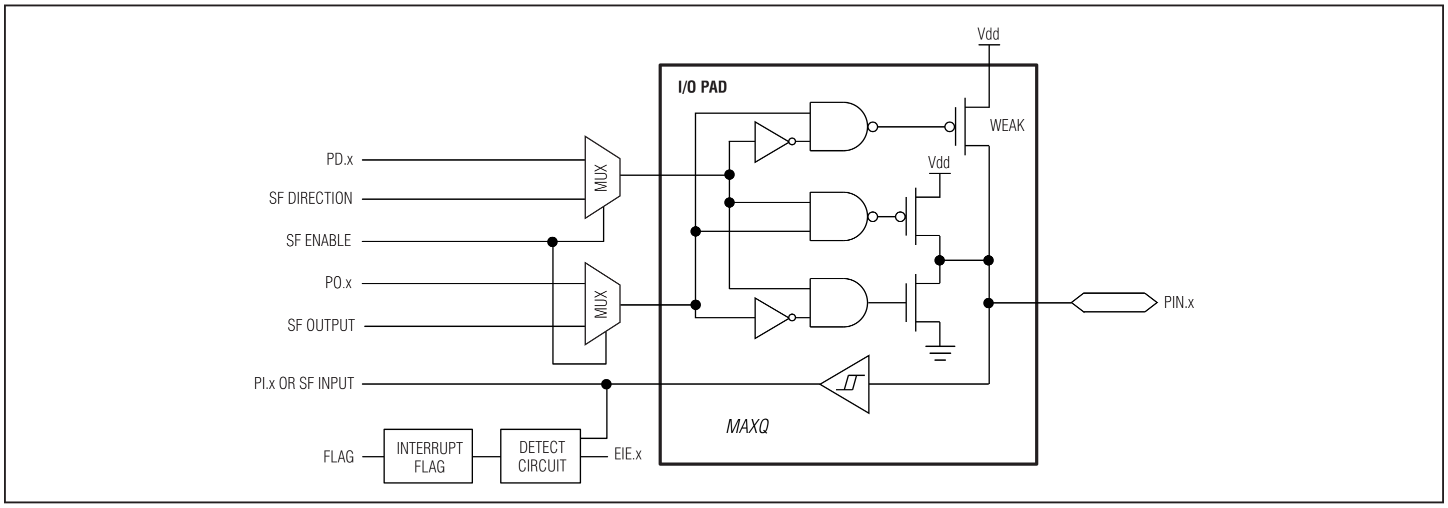


Figure 6-4. Type D Port Pin Schematic

6.5 I/O Port Peripheral Registers

6.5.1 Port Output x Register (POx)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	POx.7	POx.6	POx.5	POx.4	POx.3	POx.2	POx.1	POx.0
Reset (Type A or Type B)	0	0	0	0	0	0	0	0
Reset (Type C or Type D)	1	1	1	1	1	1	1	1
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Port Output x (POx) (POx.[7:0]). This register stores the data that is output on any of the pins of Port x that have been defined as output pins. Changing the data direction of any pins for this port (through register PDx) will not affect the value in this register.

If Port x is a Type C or Type D port that supports the weak pullup input mode, the POx register bits control the weak pullup enables for any port pins that have been configured as input pins.

6.5.2 Port Input x Register (PIx)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	PIx.7	PIx.6	PIx.5	PIx.4	PIx.3	PIx.2	PIx.1	PIx.0
Reset	s	s	s	s	s	s	s	s
Access	r	r	r	r	r	r	r	r

r = read, s = special

Bits 7 to 0: Port Input x (PIx) (PIx.[7:0]). The PIx register always reflects the logical state of its pins when read.

6.5.3 Port Direction x Register (PDx)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	PDx.7	PDx.6	PDx.5	PDx.4	PDx.3	PDx.2	PDx.1	PDx.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Port Direction x (PDx) (PDx.[7:0]). This register is used to determine the direction of the Port x function. The port pins are independently controlled by their direction bit. When a bit is set to 1, its corresponding pin is used as an output, causing data in the respective POx register bit to be driven on the pin.

For Type A and Type B ports, when a bit is cleared to 0, its corresponding pin becomes a tri-stated input, allowing and external signal to drive the pin.

For Type C and Type D ports, when a bit is cleared to 0, its corresponding pin becomes an input that can be weakly pulled up (if the respective PO bit = 1) or can be tri-stated (if the respective PO bit = 0).

6.5.4 (Type A) External Interrupt Enable Register (EIEx)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	IT1	IT0	EX5	EX4	EX3	EX2	EX1	EX0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bit 7: Interrupt 2-5 Edge Select (IT1). This bit selects the edge detection mode for external interrupts 2-5.

0 = INT2, INT3, INT4 and INT5 are positive-edge triggered

1 = INT2, INT3, INT4 and INT5 are negative-edge triggered

Bit 6: Interrupt 0, 1 Edge Select (IT0). This bit selects the edge detection mode for external interrupts 0 and 1.

0 = INT0 and INT1 are positive-edge triggered

1 = INT0 and INT1 are negative-edge triggered

Bit 5: Enable External Interrupt 5 (EX5)

0 = external interrupt 5 function disabled

1 = external interrupt 5 function enabled

Bit 4: Enable External Interrupt 4 (EX4)

0 = external interrupt 4 function disabled

1 = external interrupt 4 function enabled

Bit 3: Enable External Interrupt 3 (EX3)

0 = external interrupt 3 function disabled

1 = external interrupt 3 function enabled

Bit 2: Enable External Interrupt 2 (EX2)

0 = external interrupt 2 function disabled

1 = external interrupt 2 function enabled

Bit 1: Enable External Interrupt 1 (EX1)

0 = external interrupt 1 function disabled

1 = external interrupt 1 function enabled

Bit 0: Enable External Interrupt 0 (EX0)

0 = external interrupt 0 function disabled

1 = external interrupt 0 function enabled

6.5.5 (Type A) External Interrupt Flag Register (EIFx)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	—	—	IE5	IE4	IE3	IE2	IE1	IE0
Reset	0	0	0	0	0	0	0	0
Access	r	r	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 and 6: Reserved

Bit 5: External Interrupt 5 Flag (IE5). This flag is set when a negative edge (IT1 = 1) or a positive edge (IT1 = 0) is detected on the INT5 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 4: External Interrupt 4 Flag (IE4). This flag is set when a negative edge (IT1 = 1) or a positive edge (IT1 = 0) is detected on the INT4 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 3: External Interrupt 3 Flag (IE3). This flag is set when a negative edge (IT1 = 1) or a positive edge (IT1 = 0) is detected on the INT3 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 2: External Interrupt 2 Flag (IE2). This flag is set when a negative edge (IT1 = 1) or a positive edge (IT1 = 0) is detected on the INT2 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 1: External Interrupt 1 Flag (IE1). This flag is set when a negative edge (IT0 = 1) or a positive edge (IT0 = 0) is detected on the INT1 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 0: External Interrupt 0 Flag (IE0). This flag is set when a negative edge (IT0 = 1) or a positive edge (IT0 = 0) is detected on the INT0 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

6.5.6 (Type D) External Interrupt Enable Register (EIEx)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	EX7	EX6	EX5	EX4	EX3	EX2	EX1	EX0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bit 7: Enable External Interrupt 7 (EX7)

0 = external interrupt 7 function disabled

1 = external interrupt 7 function enabled

Bit 6: Enable External Interrupt 6 (EX6)

0 = external interrupt 6 function disabled

1 = external interrupt 6 function enabled.

Bit 5: Enable External Interrupt 5 (EX5)

0 = external interrupt 5 function disabled

1 = external interrupt 5 function enabled

Bit 4: Enable External Interrupt 4 (EX4)

0 = external interrupt 4 function disabled

1 = external interrupt 4 function enabled

Bit 3: Enable External Interrupt 3 (EX3)

0 = external interrupt 3 function disabled

1 = external interrupt 3 function enabled

Bit 2: Enable External Interrupt 2 (EX2)

0 = external interrupt 2 function disabled

1 = external interrupt 2 function enabled

Bit 1: Enable External Interrupt 1 (EX1)

0 = external interrupt 1 function disabled

1 = external interrupt 1 function enabled

Bit 0: Enable External Interrupt 0 (EX0)

0 = external interrupt 0 function disabled

1 = external interrupt 0 function enabled

6.5.7 (Type D) External Interrupt Flag Register (EIFx)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	IE7	IE6	IE5	IE4	IE3	IE2	IE1	IE0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bit 7: External Interrupt 7 Flag (IE7). This flag is set when a negative edge (IT7 = 1) or a positive edge (IT7 = 0) is detected on the INT7 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 6: External Interrupt 6 Flag (IE6). This flag is set when a negative edge (IT6 = 1) or a positive edge (IT6 = 0) is detected on the INT6 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 5: External Interrupt 5 Flag (IE5). This flag will be set when a negative edge (IT5 = 1) or a positive edge (IT5 = 0) is detected on the INT5 pin. This bit will remain set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 4: External Interrupt 4 Flag (IE4). This flag is set when a negative edge (IT4 = 1) or a positive edge (IT4 = 0) is detected on the INT4 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 3: External Interrupt 3 Flag (IE3). This flag is set when a negative edge (IT3 = 1) or a positive edge (IT3 = 0) is detected on the INT3 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 2: External Interrupt 2 Flag (IE2). This flag is set when a negative edge (IT2 = 1) or a positive edge (IT2 = 0) is detected on the INT2 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 1: External Interrupt 1 Flag (IE1). This flag is set when a negative edge (IT1 = 1) or a positive edge (IT1 = 0) is detected on the INT1 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

Bit 0: External Interrupt 0 Flag (IE0). This flag is set when a negative edge (IT0 = 1) or a positive edge (IT0 = 0) is detected on the INT0 pin. This bit remains set until cleared in software. Setting this bit by software causes an interrupt if enabled.

6.5.8 (Type D) External Interrupt Edge Select Register (EIESx)

Bit #	7	6	5	4	3	2	1	0
Name	IT7	IT6	IT5	IT4	IT3	IT2	IT1	IT0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bit 7: Interrupt 7 Edge Select (IT7). This bit selects the edge detection mode for external interrupt 7.

0 = INT7 is positive-edge triggered

1 = INT7 is negative-edge triggered

Bit 6: Interrupt 6 Edge Select (IT6). This bit selects the edge detection mode for external interrupt 6.

0 = INT6 is positive-edge triggered

1 = INT6 is negative-edge triggered

Bit 5: Interrupt 5 Edge Select (IT5). This bit selects the edge detection mode for external interrupt 5.

0 = INT5 is positive-edge triggered

1 = INT5 is negative-edge triggered

Bit 4: Interrupt 4 Edge Select (IT4). This bit selects the edge detection mode for external interrupt 4.

0 = INT4 is positive-edge triggered

1 = INT4 is negative-edge triggered

Bit 3: Interrupt 3 Edge Select (IT3). This bit selects the edge detection mode for external interrupt 3.

0 = INT3 is positive-edge triggered

1 = INT3 is negative-edge triggered

Bit 2: Interrupt 2 Edge Select (IT2). This bit selects the edge detection mode for external interrupt 2.

0 = INT2 is positive-edge triggered

1 = INT2 is negative-edge triggered

Bit 1: Interrupt 1 Edge Select (IT1). This bit selects the edge detection mode for external interrupt 1.

0 = INT1 is positive-edge triggered

1 = INT1 is negative-edge triggered

Bit 0: Interrupt 0 Edge Select (IT0). This bit selects the edge detection mode for external interrupt 0.

0 = INT0 is positive-edge triggered

1 = INT0 is negative-edge triggered

SECTION 7: TIMER/COUNTER 0 MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 7: TIMER/COUNTER 0 MODULE

The Timer/Counter 0 Module allows the MAXQ to control a 16-bit programmable timer/counter. Whether and how many Timer/Counter 0 Modules are implemented in a given MAXQ-based microcontroller is product dependent.

7.1 Timer 0

Timer 0 is the first type of 16-bit timer/counter. Timer 0 consists of a 16-bit register in two bytes, T0H and T0L. Timer 0 is enabled by the Timer 0 Run Control (TR0) bit in the T0CN register. Timer 0 supports four basic modes of operations. The mode of operation is controlled by the Timer 0 Control (T0CN) register. Table 7-1 shows these four modes and the corresponding T0CN register bit settings. Each of the four Timer 0 operational modes can optionally select that an external pin serve as the Timer 0 input clock, and can also gate the input clock source (either internal or external) based upon an external pin state. The gating feature is useful in measuring the pulse width of external signals.

Table 7-1. Timer 0 Mode Summary

TIMER 0 OPERATIONAL MODE	T0CN REGISTER BIT SETTINGS	
	M1	M0
13-Bit Timer/Counter	0	0
16-Bit Timer/Counter	0	1
8-Bit Timer/Counter with Auto-Reload	1	0
Two 8-Bit Timer/Counters	1	1

Note 1: *When $C/\overline{T}$ = 1, the counter configuration is in effect, and the T0 pin signal provides the input clock.*
Note 2: *When GATE = 1, the gating control is in effect, and the T0G = 0 pin state causes gating of the Timer/Counter input clock.*

7.1.1 Timer 0 Mode: 13-Bit Timer/Counter

As referenced in Table 7-1, setting T0CN register bits M1:M0 = 00b selects the 13-bit Timer/Counter operating mode for Timer 0. T0H provides the 8 MSbs (most significant bits) of the 13-bit timer, while bits 4–0 of T0L serve as the 5 LSbs (least significant bits) of the 13-bit timer. Bit 4 of T0L is used as a ripple-out to T0H bit 0, thereby completely bypassing bits 5 to 7 of T0L. The upper three bits of T0L are indeterminate. When the 13-bit count reaches 1FFFh (all ones), the next count causes it to roll over to 0000h. The TF0 (T0CN.5) flag is set, and an interrupt occurs if enabled.

Once the timer is started using the TR0 (T0CN.4) timer enable, the timer counts as long as one of the following conditions is true:

1) GATE (T0CN.3) = 0

2) GATE (T0CN.3) = 1 and T0G (external pin) = 1

The Timer 0 input clock is normally a function of the system clock frequency as defined by the T0M (T0CN.6) bit. However, an external signal at the T0 pin can serve as the input clock if the $C/\overline{T}$ (T0CN.2) bit is set to 1. When using the T0 pin as an input clock, Timer/Counter 0 counts 1-to-0 transitions on the pin. To reliably detect external 1-to-0 transitions, the input signal high and low times each must be a minimum of one system clock in duration.

Note that when the Timer 0 input clock is derived from the system clock, changing the system clock divide ratio (via the CKCN register bit controls) consequently changes the input clock to the Timer.

7.1.2 Timer 0 Mode: 16-Bit Timer/Counter

Setting the T0CN register bits M1:M0 = 01b invokes the 16-bit Timer/Counter operating mode. This mode is identical to the 13-bit Timer/Counter mode, except that the T0H:T0L register pair hold a 16-bit value. T0H holds the MSB and T0L holds the LSB. Rollover occurs when the timer reaches FFFFh. An interrupt occurs if enabled and the TF0 (T0CN.5) flag is set. Time-base selection, counter/timer selection, and the gate function operate just as described for the 13-bit Timer/Counter mode.

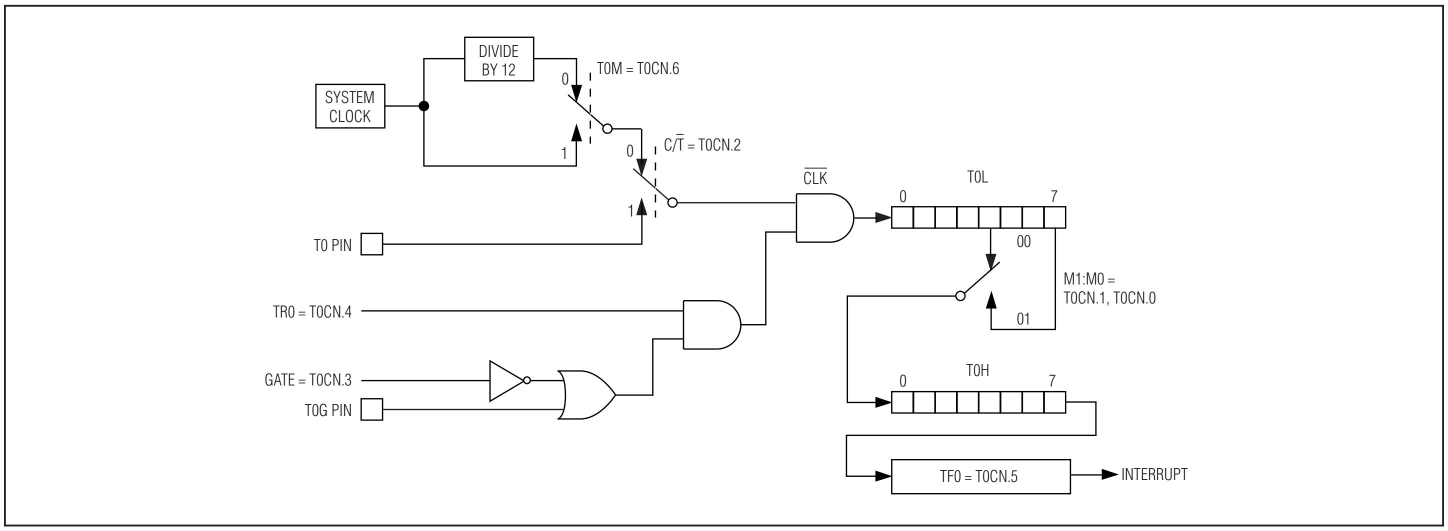


Figure 7-1. Timer/Counter 0 13-Bit/16-Bit Modes

7.1.3 Timer 0 Mode: 8-Bit Timer with Auto-Reload

When T0CN register bits M1:M0 = 10b, Timer 0 is configured as an 8-bit timer/counter with automatic reload of the start value. The timer uses T0L to count and T0H to store the reload value. Software must initialize both T0L and T0H with the same starting value for the first count to be correct. Once the T0L reaches FFh, it is automatically loaded with the value in T0H. The T0H value remains unchanged unless modified by software. Like the other Timer 0 modes, this mode allows counting of either clock cycles or pulses on the T0 pin (C/$\overline{T}$ = 1) and allows gating (GATE = 1) of the T0 pin input with the T0G pin.

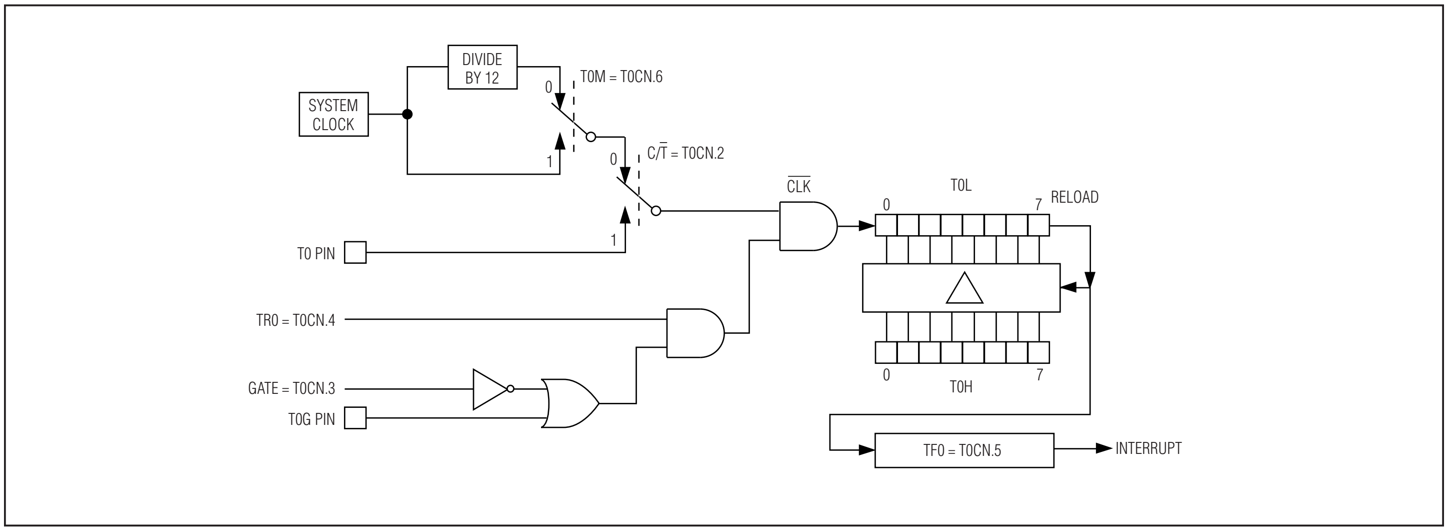


Figure 7-2. Timer/Counter 0 8-Bit Auto-Reload Mode

7.1.4 Timer 0 Mode: Two 8-Bit Timer/Counters

When T0CN register bits M1:M0 = 11b, Timer 0 provides two 8-bit timer/counters as shown in Figure 7-3. In this mode, T0L is an 8-bit timer/counter that can be used to count clock cycles or 1-to-0 transitions on pin T0 as determined by $C/\overline{T}$. (T0CN.2). As in the other modes, the GATE function can use T0G to give external run control of the timer to an outside signal.

T0H becomes an independent 8-bit timer that can only count clock cycles as shown in Figure 7-3. The clock-input enable for both timer/counters (T0L and T0H) is controlled by the Timer 0 Run (TR0) bit, while the Timer 0 interrupt flag bit (TF0) is associated only with rollover of the T0H register.

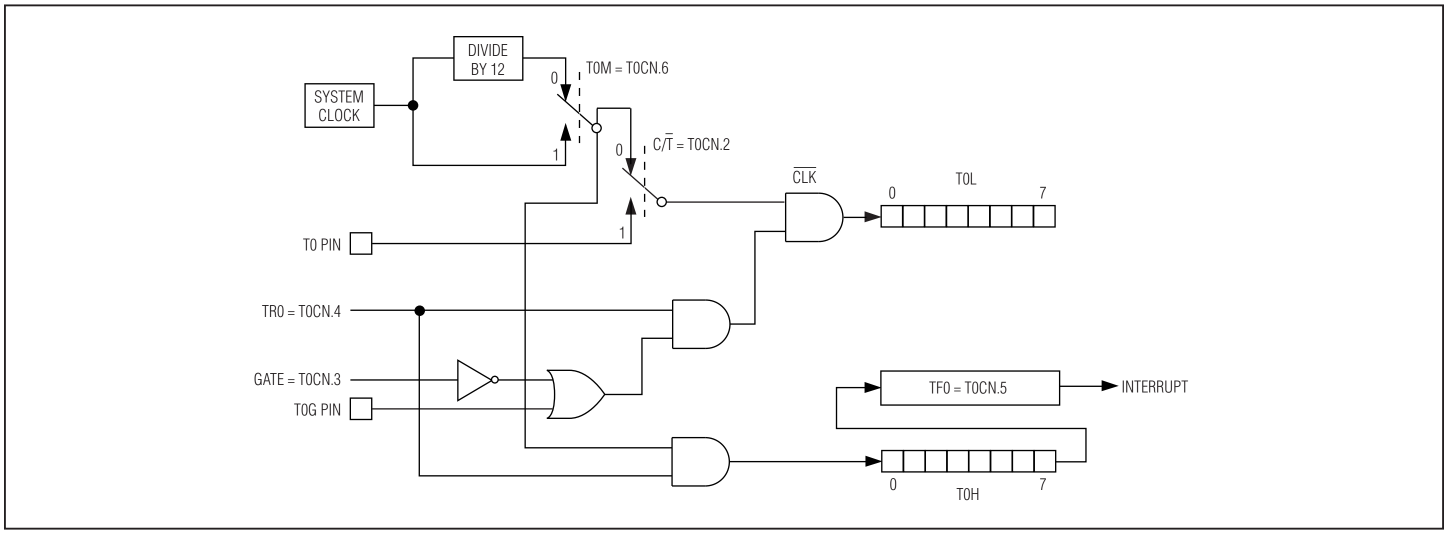


Figure 7-3. Timer/Counter 0 Dual 8-Bit Mode

7.2 Timer/Counter 0 Peripheral Registers

7.2.1 Timer/Counter 0 Control Register (T0CN)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	ET0	T0M	TF0	TR0	GATE	$C/\overline{T}$	M1	M0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bit 7: Enable Timer 0 Interrupt (ET0). Setting this bit to 1 enables interrupts from the Timer 0 TF0 flag. Clearing this bit to 0 disables the Timer 0 interrupt.

Bit 6: Timer 0 Clock Select (T0M). The T0M bit selects the clock frequency for Timer 0:

0 = Uses a divide by 12 of the system clock frequency as Timer 0 base clock.

1 = Uses a divide by 1 of the system clock frequency as Timer 0 base clock.

Bit 5: Timer 0 Overflow Flag (TF0). This bit is set to 1 when Timer 0 overflows its maximum count as defined by the current mode. It is cleared either by software or a reset. When this bit is 0, no Timer 0 overflow has been detected.

Bit 4: Timer 0 Run Control (TR0). Setting this bit enables Timer/Counter 0. Clearing this bit halts the Timer/Counter 0.

Bit 3: Timer 0 Gate Control (GATE)

0 = Timer 0 will clock when TR0 is 1, regardless of the logic state of the external T0G gating control pin

1 = Timer 0 will clock only when TR0 and the logic state of the external T0G gating control pin are 1

Bit 2: Counter/Timer 0 Select ($C/\overline{T}$)

0 = Selects timer function with internal clock when TR0 is 1

1 = Selects counter function with external T0 input when TR0 is 1

Bits 1 and 0: (M[1:0]). These mode select bits define the Timer/Counter mode of operation:

M1	**M0**	**FUNCTION**
0	0	Mode 0: 8-Bit with 5-Bit Prescale
0	1	Mode 1: 16-Bit with No Prescale
1	0	Mode 2: 8-Bit with Auto-Reload
1	1	Mode 3: Two 8-Bit Timers

7.2.2 Timer/Counter 0 High Register (T0H)

Bit #	7	6	5	4	3	2	1	0
Name	T0H.7	T0H.6	T0H.5	T0H.4	T0H.3	T0H.2	T0H.1	T0H.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Timer/Counter 0 High (T0H.[7:0]). The T0H register is used to load the most significant 8-bit value and least significant 8-bit value of Timer 0.

7.2.3 Timer/Counter 0 Low Register (T0L)

Bit #	7	6	5	4	3	2	1	0
Name	T0L.7	T0L.6	T0L.5	T0L.4	T0L.3	T0L.2	T0L.1	T0L.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Timer/Counter 0 Low (T0L.[7:0]). The T0L register is used to read the most significant 8-bit value and least significant 8-bit value of Timer 0.

SECTION 8: TIMER/COUNTER 1 MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 8: TIMER/COUNTER 1 MODULE

The Timer/Counter 1 Module allows the MAXQ to control a 16-bit programmable timer/counter. Whether and how many Timer/Counter 1 Modules are implemented in a given MAXQ-based microcontroller is product dependent.

8.1 Timer 1

Timer 1 is the second type of 16-bit timer/counter. Timer 1 consists of a 16-bit register in two bytes, T1H and T1L. Timer 1 is enabled by the Timer 1 Run Control (TR1) bit in the T1CN register. Unlike Timer 0, Timer 1 is operable only as a full 16-bit timer/counter. However, it supports many optional modes not available on Timer 0. These optional modes are enabled by T1CN register bits. To support the extended functionality of Timer 1, a 16-bit capture register composed of the T1CH, T1CL bytes and a second mode control register (T1MD) are implemented. Table 8-1 shows the Timer 1 operational modes and the corresponding T1CN register bit settings. With exception of the Timer 1 clock output mode, all Timer 1 modes can optionally select that an external pin serve as the Timer 1 input clock.

Table 8-1. Timer/Counter 1 Mode Summary

TIMER 1 OPERATIONAL MODE	T1CN REGISTER BIT SETTINGS					OPTIONAL CONTROL
	T1OE	DCEN	EXEN1	$C/\overline{T1}$	$CP/\overline{RL1}$	
Auto-Reload	0	0	0	x	0	—
Auto-Reload Using T1EX Pin	0	0	1	x	0	—
Capture Using T1EX Pin	0	0	1	x	1	—
Up/Down Count Using T1EX Pin	0	1	0	x	0	—
—	0	x	x	1	x	Input clock = T1 pin
Clock Output on T1 Pin	1	x	x	0	0	—

8.1.1 Timer 1 Mode: 16-Bit Timer/Counter with Auto-Reload

The Timer 1 auto-reload mode is configured by setting the $CP/\overline{RL1}$ (T1CN.0) bit to logic 0. In this mode, Timer 1 performs a simple timer or counter function where it behaves similarly to the 16-bit timer/counter mode offered on Timer 0, but adds a separate 16-bit reload value and the ability to trigger a reload with an external pin.

Timer 1 begins counting from the value supplied in T1H and T1L. When Timer 1 reaches an overflow state, i.e., rolls over from FFFFh to 0000, it sets the TF1 Flag. This flag can generate an interrupt if enabled. In addition, the timer restores its starting value and begins timing (or counting) again. The starting value is preloaded by software into the capture registers T1CH and T1CL. These registers cannot be used for capture functions while also performing auto-reload, so these modes are mutually exclusive.

When in auto-reload mode, Timer 1 can also be forced to reload with the T1EX pin. If EXEN1 (T1CN.3) is set to logic 1, then a 1-to-0 transition on T1EX causes a reload. Otherwise, the T1EX pin is ignored.

If the C/$\overline{\text{T1}}$ bit (T1CN.1) is logic 0, the timer's input clock is a function of the system clock. When C/$\overline{\text{T1}}$ = 1, pulses on the T1 pin are counted. Counting or timing is enabled or disabled using the with the Timer 1 Run Control bit = TR1 (T1CN.2). This mode, including the optional reload control, is illustrated in Figure 8-1.

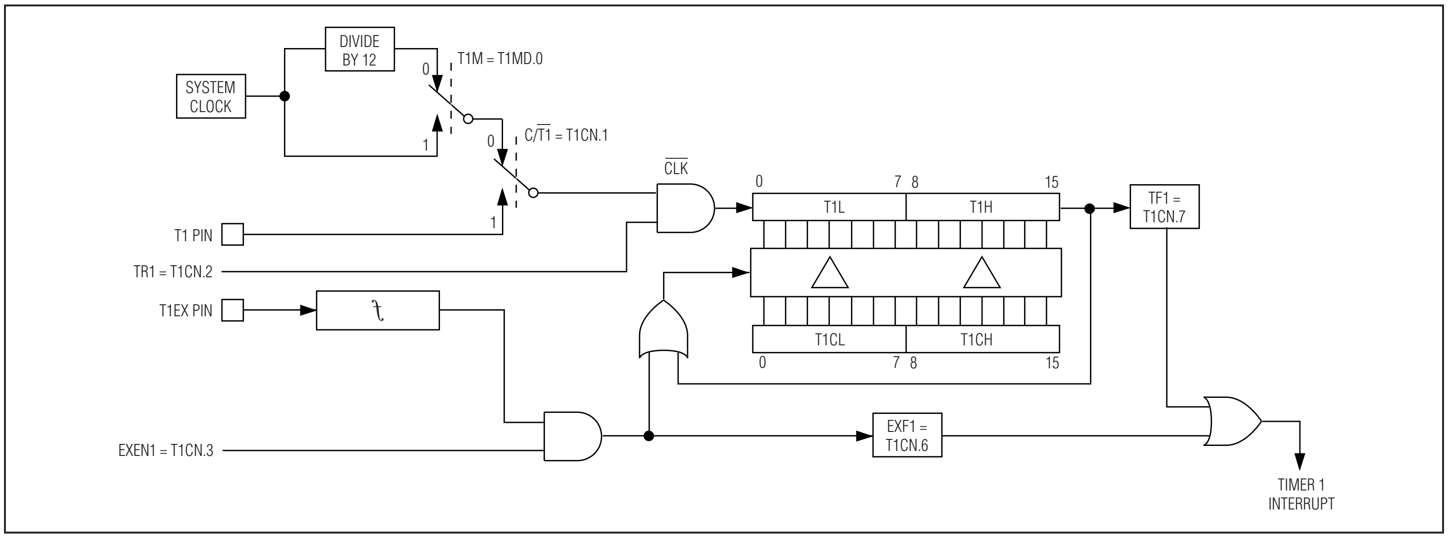


Figure 8-1. Timer/Counter 1 16-Bit Auto-Reload Mode

8.1.2 Timer 1 Mode: 16-Bit Event Capture

The 16-bit capture mode is invoked by setting the CP/$\overline{\text{RL1}}$ (T1CN.0) bit to logic 1. Timer 1 begins counting from the value supplied in T1H and T1L until reaching an overflow state, i.e., rolls over from FFFFh to 0000, at which point it sets the TF1 Flag. This flag can generate an interrupt if enabled. The optional capture function is enabled by setting the EXEN1 (T1CN.3) bit to logic 1. Once this has been done, a 1-to-0 transition on the T1EX pin causes the value in Timer 1 (T1H, T1L) to be transferred into the capture registers (T1CH, T1CL) and the EXF1 (T1CN.6) flag to be set. Setting of the EXF1 flag can generate an interrupt if enabled. If the EXEN1 bit is set to logic 0, 1-to-0 transitions on the T1EX pin do not automatically trigger a capture event.

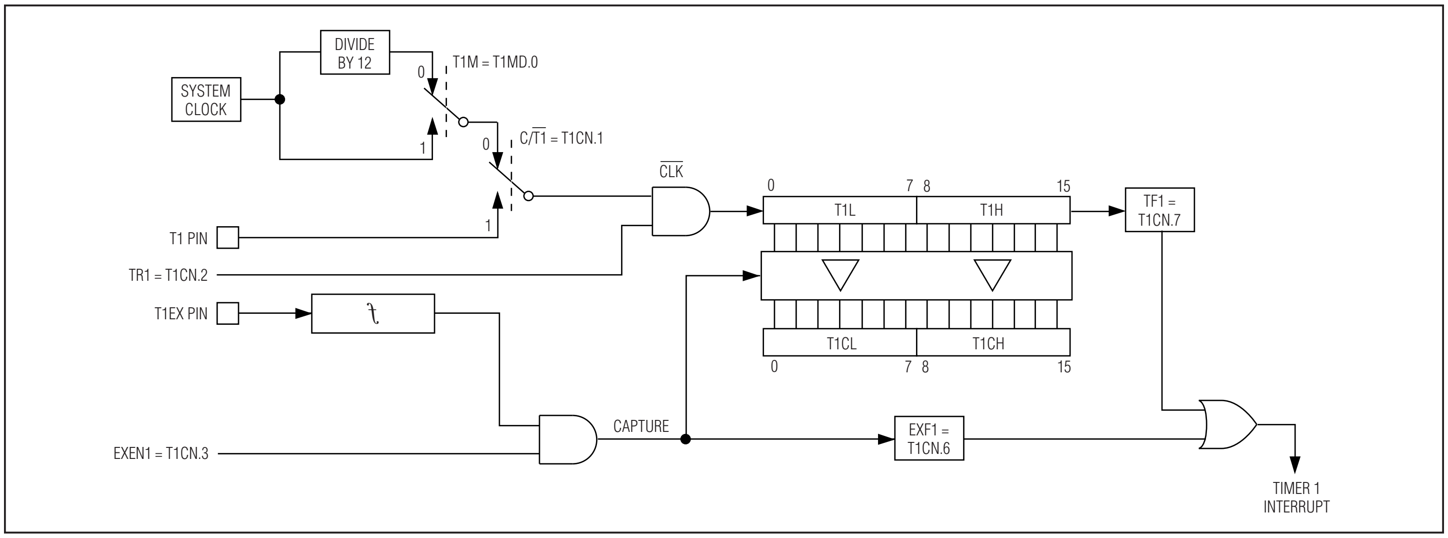


Figure 8-2. Timer/Counter 1 16-Bit Event Capture

8.1.3 Timer 1 Mode: Up/Down Count with Auto-Reload

The up/down count auto-reload option is enabled by the DCEN (T1CN.4) bit. When DCEN is set to logic 1, Timer 1 counts up or down as controlled by the state of T1EX pin. T1EX causes upward counting when a logic 1 is applied and down counting when a logic 0 is applied. When DCEN = 0, Timer 1 only counts up.

When an upward counting overflow occurs, the value in T1CH and T1CL loads into T1H and T1L. In the down count direction, an underflow occurs when T1H and T1L match the values in T1CH and T1CL, respectively. When an underflow occurs, FFFFh is loaded into T1H and T1L and counting continues.

Note that in this mode, the overflow/underflow output of the timer is provided to an edge-detection circuit as well as to the TF1 bit (T1CN.7). This edge-detection circuit toggles the EXF1 bit (T1CN.6) on every overflow or underflow. Therefore, the EXF1 bit behaves as a 17th bit of the counter, and may be used as such.

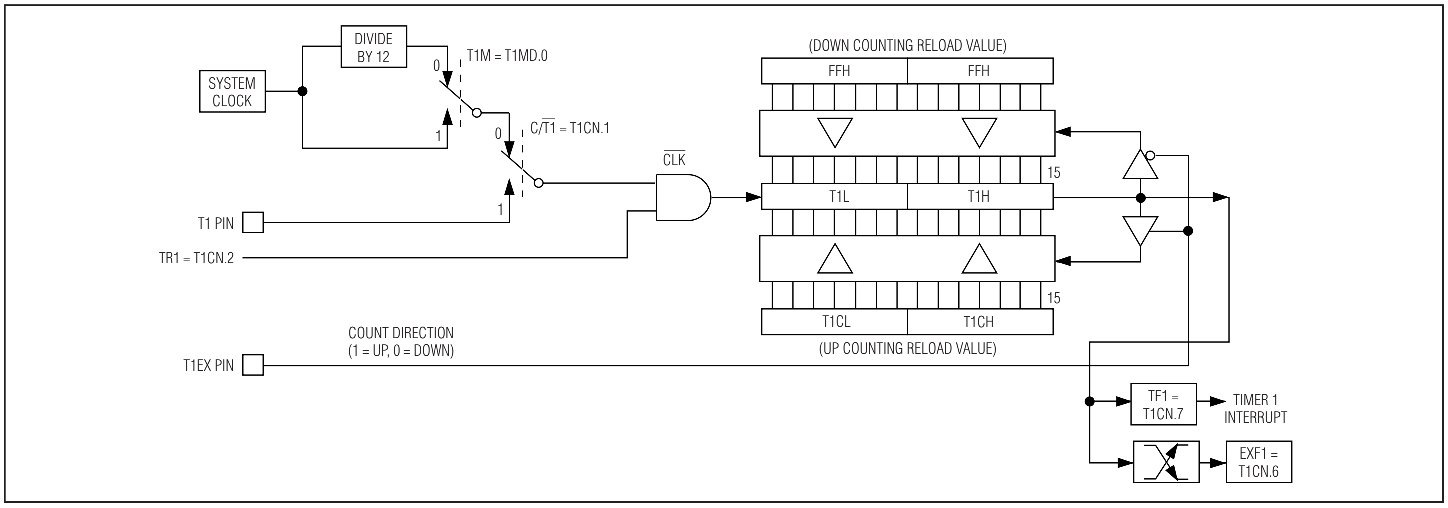


Figure 8-3. Timer/Counter 1 16-Bit Up/Down Count with Auto-Reload

8.1.4 Timer 1 Mode: Clock Output

Timer 1 can also be configured to drive a clock output on the T1 port pin, as shown in Figure 8-4. To configure Timer 1 for this mode, it must first be set to 16-bit auto-reload timer mode (CP/RL1 = 0, C/$\overline{T1}$ = 0). Next, the T1OE (T1CN.5) bit must be set to logic 1. TR1

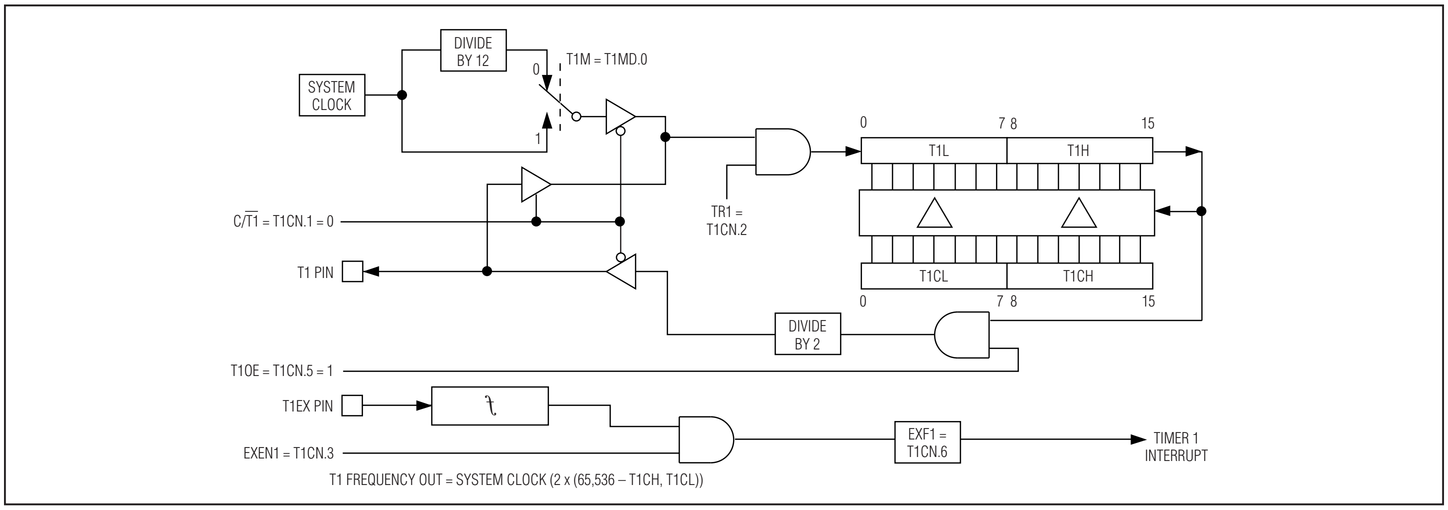


Figure 8-4. Timer 1 Clock Output Mode

(T1CN.2) must also be set to logic 1 to enable the timer. The DCEN bit has no effect in this mode. This mode produces a 50% duty cycle square-wave output. The frequency of the square wave is given by the formula in Figure 8-4. Each timer overflow causes an edge transition on the pin, i.e., the state of the pin toggles. Note that the timer itself does not generate an interrupt, but if needed, the Timer 1 external interrupt is still available for use when enabled (EXEN1 = 1).

8.2 Timer/Counter 1 Peripheral Registers

8.2.1 Timer/Counter 1 Control Register (T1CN)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	TF1	EXF1	T1OE	DCEN	EXEN1	TR1	$C/\overline{T1}$	$CP/\overline{RL1}$
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 1 Overflow Flag (TF1). This bit is set when Timer 1 overflows from FFFFh or the count is equal to the capture register in down count mode. It must be cleared by software.

Bit 6: External Timer 1 Trigger Flag (EXF1). A negative transition on the T1EX causes this flag to be set if ($CP/\overline{RL1}$ = EXEN1 = 1) or ($CP/\overline{RL1}$ = DCEN = 0 and EXEN1 = 1). When $CP/\overline{RL1}$ = 0 and DCEN = 1, EXF1 toggles whenever Timer 1 underflows or overflows. In this mode, EXF1 can be used as the 17th Timer bit and will not cause an interrupt. If set by a negative transition, this flag must be cleared by software. Setting this bit to 1 forces a Timer interrupt if enabled.

Bit 5: Timer 1 Output Enable (T1OE). Setting this bit to 1 enables the clock output function of T1 pin if $C/\overline{T1}$ = 0. Timer 1 rollovers will not cause interrupts. Clearing this bit to 0 causes the T1 pin to function as either a standard port pin or a counter input for Timer 1.

Bit 4: Down Count Enable (DCEN). This bit, in conjunction with the T1EX pin, controls the direction that Timer 1 counts in 16-bit auto-reload mode. Clearing this bit to 0 causes Timer 1 to count up. Setting this bit to 1 causes Timer 1 to count up if the T1EX pin is 1 and to count down if the T1EX pin is 0.

Bit 3: Timer 1 External Enable (EXEN1). Setting this bit to 1 enables the capture/reload function on the T1EX pin for a negative transition. Clearing this bit to 0 causes Timer 1 to ignore all external events on T1EX pin.

Bit 2: Timer 1 Run Control (TR1). Setting this bit enables Timer 1. Clearing this bit halts the Timer 1.

Bit 1: Counter/Timer Select ($C/\overline{T1}$). This bit determines whether Timer 1 functions as a Timer or counter. Setting this bit to 1 causes Timer 1 to count negative transitions on the T1 pin. Clearing this bit to 0 causes Timer 1 to function as a Timer. The speed of Timer 1 is determined by the T1M bit, either divide by 1 or divide by 12 of the system clock, including the clock output mode.

Bit 0: Capture/Reload Select ($CP/\overline{RL1}$). This bit determines whether the capture or reload function is used for Timer 1. Timer 1 functions in an auto-reload mode following each overflow. Setting this bit to 1 causes a Timer 1 capture to occur when a falling edge is detected on T1EX if EXEN1 = 1. Clearing this bit to 0 causes an auto-reload to occur when Timer 1 overflow or a falling edge is detected on T1EX if EXEN1 = 1.

8.2.2 Timer/Counter 1 High Register (T1H)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	T1H.7	T1H.6	T1H.5	T1H.4	T1H.3	T1H.2	T1H.1	T1H.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Timer/Counter 1 High (T1H.[7:0]). The T1H register is used to load the most significant 8-bit value and least significant 8-bit value of Timer 1.

8.2.3 Timer/Counter 1 Low Register (T1L)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	T1L.7	T1L.6	T1L.5	T1L.4	T1L.3	T1L.2	T1L.1	T1L.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Timer/Counter 1 Low (T1L.[7:0]). The T1L register is used to read the most significant 8-bit value and least significant 8-bit value of Timer 1.

8.2.4 Timer/Counter 1 High Register (T1CH)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	T1CH.7	T1CH.6	T1CH.5	T1CH.4	T1CH.3	T1CH.2	T1CH.1	T1CH.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Timer/Counter 1 High (T1CH.[7:0]). The T1CH register is used to capture the T1H values when Timer 1 is configured in capture mode. This register is also used as the MSB of a 16-bit reload value when Timer 1 is configured in auto-reload mode.

8.2.5 Timer/Counter 1 Low Register (T1CL)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	T1CL.7	T1CL.6	T1CL.5	T1CL.4	T1CL.3	T1CL.2	T1CL.1	T1CL.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Timer/Counter 1 Low (T1CL.[7:0]). The T1CL register is used to capture the T1L values when Timer 1 is configured in capture mode. This register is also used as the LSB of a 16-bit reload value when Timer 1 is configured in auto-reload mode.

8.2.6 Timer/Counter 1 Mode Register (T1MD)

Bit #	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	ET1	T1M
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	rw	rw

r = read, w = write

Bits 7 to 2: Reserved

Bit 1: Enable Timer 1 Interrupt (ET1). Setting this bit to 1 enables interrupts from the Timer 1 TF1 and EXF1 flags in T1CN. The EXF1 flag does not cause interrupts in the up/down count mode.

Bit 0: Timer 1 Clock Select (T1M). The T0M bit selects the clock frequency for Timer 1:

0 = Uses a divide by 12 of the system clock frequency as Timer 1 base clock

1 = Uses a divide by 1 of the system clock frequency as Timer 1 base clock

8.3 Time-Base Selection for Timers 0 and 1

The MAXQ allows selection of the time base for each timer independently. The input clock for each timer defaults to 12 system clocks per timer tick. The timer-input clock frequency can be increased by setting the respective TxM bit for the timer (T0M for Timer 0; T1M for Timer 1). Setting the TxM bit allows the system clock input to be used for the timer-input clock. Table 8-2 shows the resulting timer input clock for the various system clock modes according to timer control bit TxM setting.

Table 8-2. Input Clock Frequency Selection for Timer 0 and Timer 1

SYSTEM CLOCK MODE	SYSTEM CLOCK SELECT BITS PMME, CD1, CD0	TIMER 0, 1 INPUT CLOCK FREQUENCY	
		TxM = 0	**TxM = 1**
Divide by 1	000	CLK / 12	CLK / 1
Divide by 2	001	CLK / 24	CLK / 2
Divide by 4	010	CLK / 48	CLK / 4
Divide by 8	011	CLK / 96	CLK / 8
Power Management Mode (Divide by 256)	1xx	CLK / 3072	CLK / 256

SECTION 9: TIMER/COUNTER 2 MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 9: TIMER/COUNTER 2 MODULE

The Timer/Counter 2 Module provides a 16-bit programmable timer/counter with pulse-width modulation capability. Whether and how many Timer 2 Modules are implemented in a given MAXQ-based microcontroller is product dependent.

Timer 2 is an auto-reload, 16-bit timer/counter offering the following functions:

- 8-bit/16-bit timer/counter
- up/down auto-reload
- counter function of external pulse
- capture
- compare
- input/output enhancements

9.1 Timer 2

The 16-bit Timer 2 value is contained in the T2V register. The upper byte is always accessible through the T2H 8-bit register. When Timer 2 is operated in the dual 8-bit mode of operation, the high byte of T2V always reads x00h and is not write accessible. The low byte of the T2V will often be referenced as T2L. Similar registers and nomenclature are used for the Timer 2 auto-reload value resides in the T2R register. A separate 8-bit T2RH register allows read/write access to the high byte and the low byte of T2R is often referred to as T2RL. The Capture/Compare functionality is supported by Timer 2 through the 16-bit T2C capture register and the 8-bit T2CH high-byte access register. Timer 2 normally requires two pins to support the enhanced input/output functionality. Throughout this section the primary input/output pin will be referred to as T2P and the secondary pin, which may or may not be present for a given device, will be referred to as T2PB. Decision whether and/or where to implement the T2PB pin functionality is product and application dependent. Table 9-1 summarizes the modes supported by Timer 2 and the peripheral register bits associated with those modes.

Table 9-1. Timer/Counter 2 Functions and Control

MODE	T2MD	$C/\overline{T2}$	CCF[1:0]	CONTROL BITS
16-Bit Auto-Reload/Compare Timer	0	0	00	T2OE[1:0]—output enables (PWM out)
				T2POL[1:0]—input/output polarity select
				SS2—single-shot pulse control
				G2EN—gated PWM output
16-Bit Capture (CCF[1:0] bits define capture edge)	0	0	01, 10, or 11	T2OE[0] = 0
				T2POL[0]—gate level/reload edge select
				SS2—single-shot capture
				G2EN—gate timer clock (or gate reload)
				CPRL2—reload enable
16-Bit Counter (CCF[1:0] bits define count edge)	0	1	01, 10, or 11	T2OE[0] = 0
				T2OE[1]—pulse counter output
				T2POL[1]—output polarity select
Dual 8-Bit Auto-Reload Timers	1	0	00	T2OE[1:0]—output enables (PWM out)
				T2POL[1:0]—output polarity select
				T2H Only:
				SS2—single-shot pulse control
8-Bit Capture and 8-Bit Timer/PWM	1	0	01, 10, or 11	T2L Only:
				T2OE[1]—output enable
				T2POL[1]—output polarity select
				T2H Only:
				T2OE[0] = 0
				T2POL[0]—gate level/reload edge select
				SS2—single-shot capture
				G2EN—gate timer (or gate reload)
				CPRL2—reload enable
8-Bit Counter and 8-Bit Timer/PWM	1	1	01, 10, or 11	T2L Only:
				T2OE[1]—output enable
				T2POL[1]—output polarity select
				T2H Only:
				T2OE[0] = 0

9.2 Modes of Operation

As summarized in Table 9-1, Timer 2 can provide six timer functions. The Timer 2 operating mode selection is illustrated in Figure 9-1 and Figure 9-2 shows the PWM timer output possibilities.

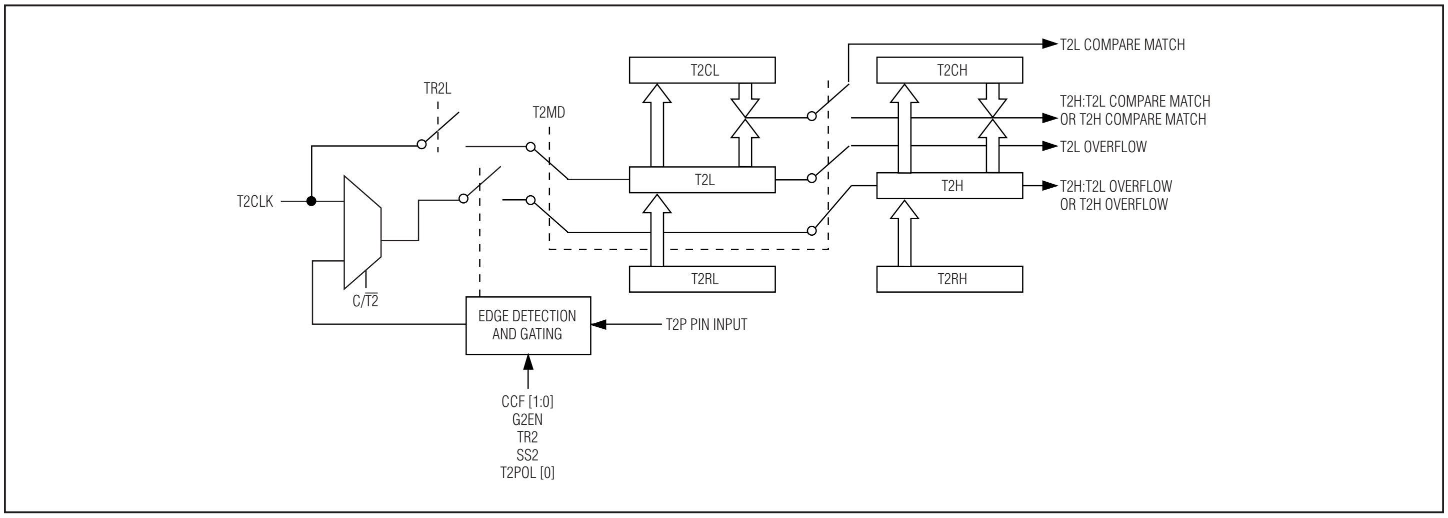


Figure 9-1. Timer 2 Mode Selection

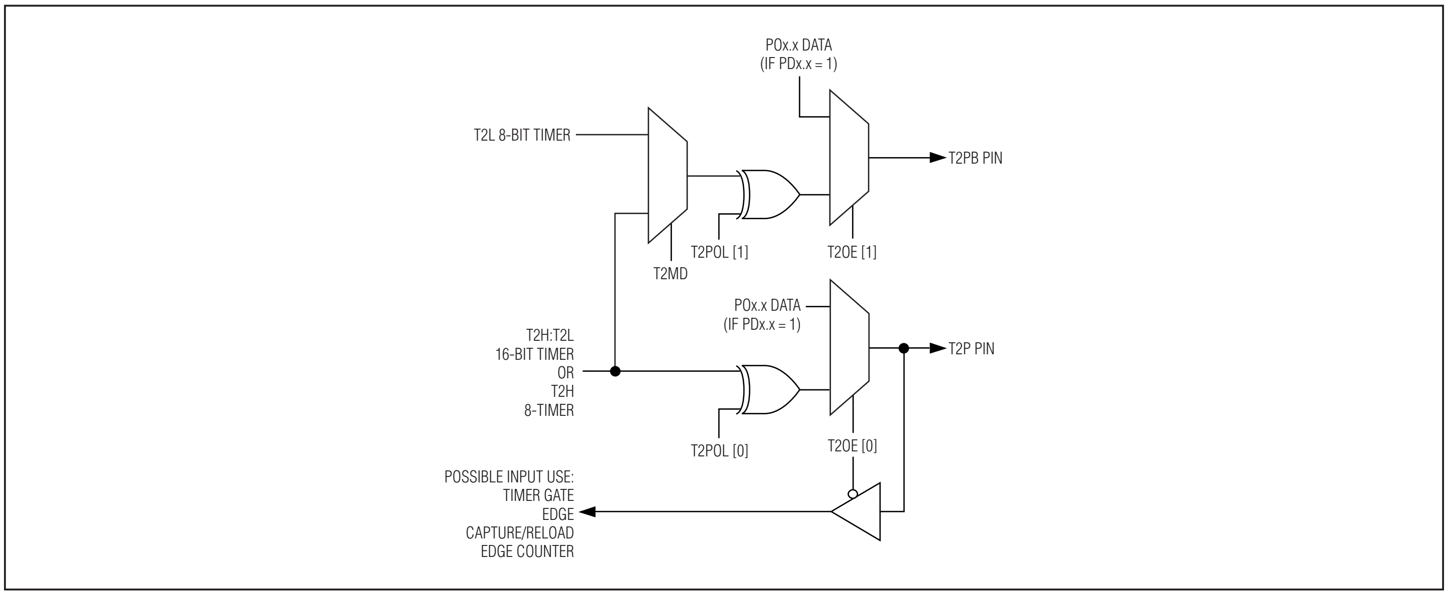


Figure 9-2. Output Enable and Polarity Control

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.

9.2.1.5 Capture/Reload Control

For the 16-bit compare operating mode, the CPRL2 bit is not used.

9.2.2 16-Bit Timer: Capture Mode

The 16-bit capture mode requires that some event trigger the capture. Normally this event is an external edge. The CCF[1:0] bits define which edge(s) cause a capture to occur. If CCF[1:0] = 01b, a rising edge causes a capture. If CCF[1:0] = 10b, a falling edge causes a capture. If CCF[1:0] = 11b, rising and falling edges both cause a capture to occur. The CPRL2 bit enables both capture and reload to occur on the specified edge(s).

9.2.2.1 Output Enables

In 16-bit capture mode, the output enables are meaningless. No output waveform is allowed since the capture/compare registers are being used for the purpose of capturing the Timer 2 value.

9.2.2.2 Polarity Control

The polarity control bits (T2POL[1:0]) have no specific meaning as related to the output function since there is no output function. The T2POL[0] bit is used to establish the gating condition for the single-edge capture mode when gating is enabled (G2EN = 1). If capture and reload are defined (CPRL2 = 1 and CCF[1:0] = 11b) for both edges, theT2POL[0] bit can be used to specify which edge does not have an associated edge reload when gating has also been enabled (G2EN bit = 1). When the SS2 bit is used to delay the timer run (for both edge capture), the T2POL[0] bit also defines which edge starts/ends the single-shot process.

9.2.2.3 Edge Detection

Edge detection was previously described (CCF[1:0] controlled).

9.2.2.4 Gated

If gating is specified, it uses the T2POL[0] bit to define when the input clock to Timer 2 is gated (just as described for the compare mode). This mode can easily be used to measure or incrementally capture high or low pulse durations. If a predefined high/low duration is required to generate an interrupt, the gated compare mode can also be used. Note that if capture is defined for both rising and falling edges, gating would serve no useful purpose as it would result in redundant capture data/interrupts. For this reason, when G2EN = 1 and CCF[1:0] = 11b, the T2POL[0] bit is used to specify which edge is a capture-only edge when CPRL2 = 1 (gating of the reload event).

9.2.2.5 Single Shot

The single-shot bit overrides the TR2 = 0 bit setting for a single edge-to-edge capture cycle (as defined by the CCF[1:0] bits). The single-shot takes effect (starting the timer) only when the edge defined by CCF[1:0] is detected or the defined gating condition is removed. While a capture and/or reload can occur on this starting edge, the interrupt flag is not set since a single-shot event has been requested. When rising or falling edge capture is defined, the single-shot mode is useful for measuring single periods. If gating is also specified for the single shot, the high/low pulse widths are easily measured. If rising and falling edges are defined, the T2POL[0] bit designates which edge starts/ends the single-shot cycle, but the starting edge does not cause the interrupt flag to set. If G2EN = 1 for the two-edge capture, the alternate edge (opposite of defined start/end edge can only be used for capture, not capture and reload). For T2POL[0] = 1, the falling edge starts and stops the single shot. This is important for combined duty cycle and period measurement.

9.2.2.6 Capture and Reload

The CPRL2 bit enables both capture and reload on the specified edge(s). The only exception to this rule is when the G2EN bit is set to logic 1. When G2EN is set to 1, a reload does not occur on the edge specified by T2POL[0]: when T2POL[0] = 0, the falling edge does not cause a reload; if T2POL[0] = 1, the rising edge does not cause a reload.

9.2.3 16-Bit Counter

The 16-bit counter mode is enabled by setting the $C/\overline{T2}$ bit to logic 1. When $C/\overline{T2}$ = 1, rising, falling, or both rising and falling edges are counted as determined by the CCF[1:0] bits. If CCF[1:0] = 00b, neither edge is defined as a counted edge, and the T2H:T2L counter holds its count since no edge is defined as the counting edge. When an overflow occurs, the reload value (T2R) is reloaded instead of the x0000h state. The Timer/Counter 2 overflow flag (TF2) is set every time that an overflow occurs. If Timer/Counter 2 interrupts have been enabled (ET2 = 1), the TF2 flag can generate an interrupt request. In counter mode, the capture/compare register (T2C) is compared versus the Timer/Counter 2 value register. Whenever a compare match occurs, the capture/compare status flag (TCC2) is set. If Timer/Counter 2 interrupts have been enabled (ET2 = 1), this event can generate 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.

9.2.3.1 Output Enable

For Timer 2 to serve as a counter, the T2P pin must be used as an input. Thus, when $C/\overline{T2}$ = 1, the T2OE[0] bit is ignored. The T2OE[1] bit can be used to output the generated waveform on T2PB resulting from compare match and overflow conditions for the counter. 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.

9.2.3.2 Polarity Control

Only the T2POL[1] bit is meaningful. It can define the starting state of the T2PB pin when the T2PB output has been enabled. The T2POL[1] bit can be changed at any time, however the assigned T2POL[1] state will take effect on the external pin only when the corresponding T2OE[1] bit is changed from 0 to 1.

9.2.3.3 Gating and Single Shot

Neither gating nor single-shot modes are supported when operating in 16-bit counter mode. The G2EN and SS2 bits should not be set to 1 when operating in the counter mode ($C/\overline{T2}$ = 1).

9.2.4 Dual 8-Bit Timers

The dual 8-bit timer mode of operation is initiated by setting the T2MD bit to logic 1. When T2MD = 1, each 16-bit register associated with Timer 2 is split into separate upper and lower 8-bit registers to support dual 8-bit timers. Thus, the primary 8-bit timer is composed of T2H (value), T2RH (reload), T2CH (capture/compare), and the secondary 8-bit timer is composed of T2L(value), T2RL(reload), and T2CH (capture/compare). There is still a single internal Timer 2 input clock that can be sourced by either of these two 8-bit timers. In the dual 8-bit mode of operation, both Timer 2 output clocks (from T2L and T2H) are available to internal peripherals as required by a given product. The secondary 8-bit timer/counter has its own run control bit (TR2L) and interrupt flags (TF2L, TC2L).

9.2.4.1 Output Enable (PWM Out)

The output enable bits (T2OE[1:0]) enable the respective 8-bit Timer 2 outputs to be presented on the pins associated with the respective bits. The T2H timer output onto the T2P pin is controlled by the T2OE[0] bit, and the T2L timer output onto the T2PB pin is controlled by the T2OE[1] bit. If Timer 2 has a single I/O pin, only the T2OE[0] bit is required as the secondary timer T2L cannot be output to a pin and can only serve as an internal timer.

9.2.4.2 Polarity Control

The polarity control bits (T2POL[1:0]) can be used to modify (invert) the enabled clock outputs to the pin(s). The starting state of the enabled clock outputs (defined by T2OE[1:0]) is the logic state of T2POL[1:0] and toggles on each compare match or overflow. 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. The T2POL[1:0] bits are logically XOR with the Timer 2 output signal, therefore setting a given T2POL[x] bit results in a high starting state. The T2POL[n] bit can be changed 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. T2POL[1] is not required for a single pin Timer 2 implementation.

9.2.4.3 Gated

To use the T2P pin as a G2EN, 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 input clock to T2H is gated any time the external signal matches the state of the T2POL[0] bit. This means that the default clock gating condition is associated with the T2P pin being low (T2POL[0] = 0). Note that the secondary 8-bit timer, T2L, cannot be gated. Also, since the output enables T2OE[1:0] apply to each individual 8-bit timer, there is no gated PWM mode available.

9.2.4.4 Single Shot

The single-shot bit and mode apply only to the primary 8-bit timer (T2H). The single-shot mode is used to automate the generation of single pulses under software control. To generate single-shot output pulses 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. Writing SS2 and TR2 = 1 at the same time still causes the SS2 bit to stay in effect until an overflow/reload occurs. However, the specified PWM output continues since TR2 was also written to 1.

9.2.5 8-Bit Timer/8-Bit Capture Mode

When the CCF[1:0] bits are configured to a state other than 00b, the edge-capture mode is enabled for the primary timer (T2H). The secondary timer (T2L) always remains in the timer/compare mode and does not support any capture functionality. The capture controls for the 8-bit mode are identical to those specified for the 16-bit mode, however they apply only to the upper timer, T2H.

One obvious difference is that the secondary timer (T2L), operable only in compare mode, can be used to generate a PWM output with valid T2OE[1] and T2POL[1] controls, while the primary timer is operating in capture mode.

9.2.6 8-Bit Timer/8-Bit Counter

Just as in the 16-bit mode, setting the C/$\overline{\text{T2}}$ bit to logic 1 enables the external T2P pin to function as a counter input. The edges that are counted are determined by the CCF[1:0] bits. The counter mode of operation applies only to the primary timer/counter (T2H). In a similar fashion to the 16-bit counter mode, when an overflow occurs, an auto-reload of T2RH occurs and the TF2 flag is set. The TCC2 flag is also set on a compare match between the T2H counter and the T2CH compare register (except for the case where T2CH is outside the T2RH to 0xFFh counting range. The secondary timer (T2L) always continues to operate in 8-bit compare mode. Just as in the above split 8-bit timer/8-bit capture mode, this allows the secondary timer (T2L) to function in the PWM output capacity if a T2PB pin is provided. The T2POL[1] control still applies to the 8-bit T2L PWM output when T2OE[1] = 1.

9.2.7 Timer 2 Input Clock Selection

Figure 9-3 shows the Timer 2 clock source. The Timer 2 input clock is selected by the T2CI bit while the clock prescale is determined by the T2DIV bits in the T2CFG register. Note that when T2CI is configured to 1, the alternate clock source (32kHz) is sampled by the current system clock selection. The maximum sampleable alternate clock frequency is (system clock/4).

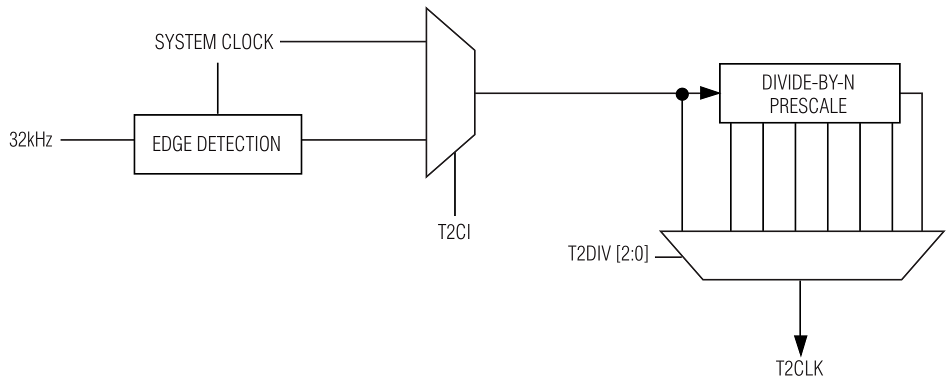


Figure 9-3. Timer 2 Clock

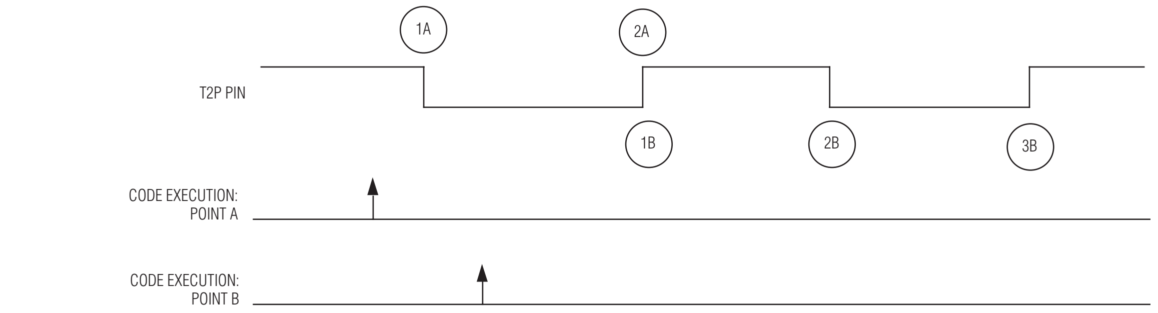
9.3 Timer 2 Capture Application Examples

The following examples and accompanying figures (Figures 9-4 through 9-8) are used to demonstrate some of the Timer 2 functions. All examples assume that pulse and/or period measurements do not exceed 2^{16} input clocks and that capture register holds the desired result.

9.3.1 Measure Low-Pulse Duration

To measure the duration of the first full low pulse seen on the T2P input pin, Timer 2 could be configured for a single shot capture, gating enabled for logic high, capture on the rising edge. The CPRL2 bit could optionally be set to generate a reload on the same rising edge as that which the capture occurs if the preconfigured T2R value is expected to be needed next.

```
; ------------------ Reset State:  T2R = T2V = T2C = 0000h ------------------------
MOVE T2CFG, #00000010b        ; T2CI          =0     (sysclk/N input)
                              ; T2DIV[2:0]    =000   (/1)
                              ; T2MD          =0     (16-bit)
                              ; CCF[1:0]      =01    (rising edge)
                              ; C/T2          =0     (timer/capture)
MOVE T2CNA, #10100111b        ; ET2           =1     (enable Timer 2 ints)
                              ; T2OE[0]       =0     (input)
                              ; T2POL[0]      =1     (gating level = '1')
                              ; TR2L:TR2      =00    (don't start timer)
                              ; CPRL2         =1     (reload on capture edge)
                              ; SS2           =1     (single-shot mode)
                              ; G2EN          =1     (gating enabled)
; ------------------ TCC2 Interrupt : DURATION = T2C
```



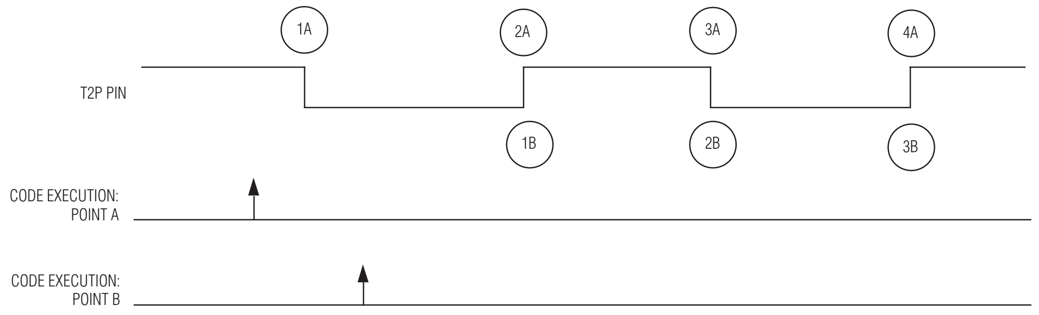
EVENTS:
1A: GATING CONDITION REMOVED; SINGLE-SHOT CAPTURE CYCLE BEGINS.
2A: RISING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE ENDS; DURATION = T2C.
1B: RISING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE BEGINS, TIMER CLOCK GATED SINCE T2P PIN = 1.
2B: GATING CONDITION REMOVED; TIMER RUNS.
3B: RISING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE ENDS; DURATION = T2C.

Figure 9-4. Timer 2 Application Example—Measure Low Pulse Width

9.3.2 Measure High-Pulse Duration Repeatedly

To measure the duration of high pulses seen on the T2P input pin repeatedly, Timer 2 could be configured for a single-shot delayed run, gating enabled for logic low, capture on the falling edge. The CPRL2 bit could be set to generate a reload on each falling edge.

```
; ----------------- Reset State:  T2R = T2V = T2C = 0000h -----------------------
MOVE T2CFG, #00000100b        ; T2CI          =0     (sysclk/N input)
                              ; T2DIV[2:0]    =000    (/1)
                              ; T2MD          =0     (16-bit)
                              ; CCF[1:0]      =10    (falling edge)
                              ; C/T2          =0     (timer/capture)
MOVE T2CNA, #10001111b        ; ET2           =1     (enable Timer 2 ints)
                              ; T2OE[0]       =0     (input)
                              ; T2POL[0]      =0     (gating level = '0')
                              ; TR2L:TR2      =01    (start timer 2 on single shot condition)
                              ; CPRL2         =1     (reload on capture edge)
                              ; SS2           =1     (single-shot mode)
                              ; G2EN          =1     (gating enabled)
; ----------------- TCC2 Interrupt : DURATION = T2C
```



EVENTS:
1A: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE BEGINS; TIMER CLOCK GATED SINCE T2P PIN = 0.
2A: GATING CONDITION REMOVED; TIMER RUNS.
3A: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE ENDS; DURATION = T2C. TIMER CONTINUES TO OPERATE SINCE TR2 = 1, BUT TIMER CLOCK GATED SINCE T2P PIN = 0.
4A: GATING CONDITION REMOVED; TIMER RUNS.

1B: GATING CONDITION REMOVED; SINGLE-SHOT CAPTURE CYCLE BEGINS.
2B: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE ENDS; DURATION = T2C. TIMER CONTINUES TO OPERATE SINCE TR2 = 1, BUT TIMER CLOCK GATED SINCE T2P PIN = 0.
3B: GATING CONDITION REMOVED; TIMER RUNS.

Figure 9-5. Timer 2 Application Example—Measure High Pulse Width

9.3.3 Measure Period

To measure the period of the signal seen on the T2P input pin, Timer 2 could be configured for a single-shot capture, no gating, either edge (selected by the CCF[1:0] bits). The CPRL2 bit could be set to generate a reload on each capture edge.

```
; ------------------ Reset State:  T2R = T2V = T2C = 0000h -----------------------
MOVE T2CFG, #00000100b        ; T2CI          =0     (sysclk/N input)
                              ; T2DIV[2:0]    =000   (/1)
                              ; T2MD          =0     (16-bit)
                              ; CCF[1:0]      =10    (falling edge)
                              ; C/T2          =0     (timer/capture)
MOVE T2CNA, #10000110b        ; ET2           =1     (enable Timer 2 ints)
                              ; T2OE[0]       =0     (input)
                              ; T2POL[0]      =0     (gating level = '0')
                              ; TR2L:TR2      =00    (don't start timer 2)
                              ; CPRL2         =1     (reload on capture edge)
                              ; SS2           =1     (single-shot mode)
                              ; G2EN          =0     (gating disabled)
; ------------------ TCC2 Interrupt : PERIOD = T2C
```

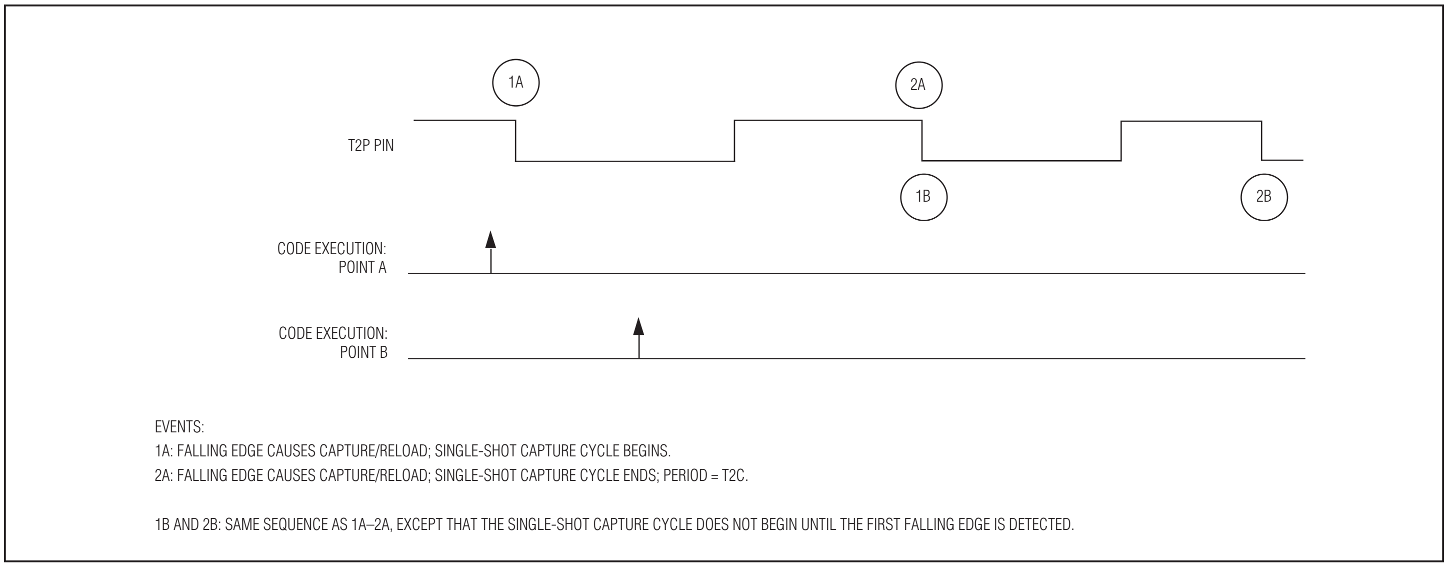
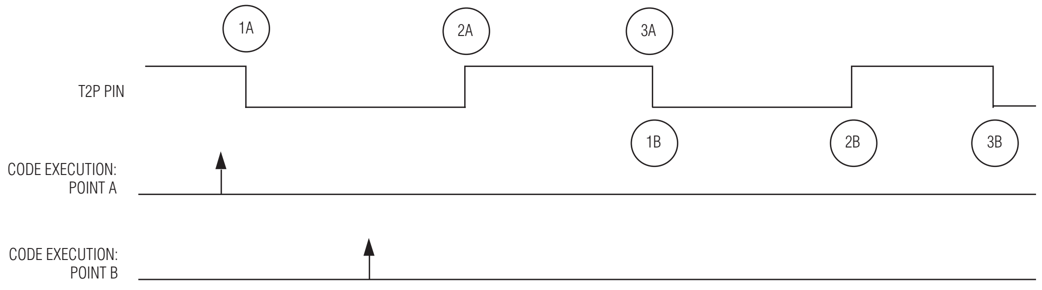

1A
2A
T2P PIN
1B
2B
CODE EXECUTION: POINT A
CODE EXECUTION: POINT B
EVENTS:
1A: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE BEGINS.
2A: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE ENDS; PERIOD = T2C.
1B AND 2B: SAME SEQUENCE AS 1A–2A, EXCEPT THAT THE SINGLE-SHOT CAPTURE CYCLE DOES NOT BEGIN UNTIL THE FIRST FALLING EDGE IS DETECTED.

Figure 9-6. Timer 2 Application Example—Measure Period

9.3.4 Measure Duty Cycle Repeatedly

To measure the duty cycle of the signal seen on the T2P input pin, Timer 2 could be configured for a single-shot delayed run with both edges defined for capture. The CPRL2 bits should be configured to 1 to request reloads on each edge. To prevent reloads on one of the edges, gating should be enabled. The T2POL[0] bit specifies which edge starts/ends the capture cycle and which edge does not have a reload associated with it.

```
; ------------------ Reset State:  T2R = T2V = T2C = 0000h -----------------------
MOVE T2CFG, #00000110b        ; T2CI          =0     (sysclk/N input)
                              ; T2DIV[ 2:0]   =000   (/1)
                              ; T2MD          =0     (16-bit)
                              ; CCF[ 1:0]     =11    (both edges)
                              ; C/T2          =0     (timer/capture)
MOVE T2CNA, #10101111b        ; ET2           =1     (enable Timer 2 ints)
                              ; T2OE[ 0]      =0     (input)
                              ; T2POL[ 0]     =1     (no reload on rising edge
;                                                    single-shot start/end on falling edge)
                              ; TR2L:TR2      =01    (start timer 2 on single shot condition)
                              ; CPRL2         =1     (reload on capture edge)
                              ; SS2           =1     (single-shot mode)
                              ; G2EN          =1     (gating enabled)
; ------------------ TCC2 Interrupt : LOW TIME=T2C
;------------------- TCC2 Interrupt : PERIOD = T2C
```



EVENTS:
1A: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE BEGINS.
2A: RISING EDGE CAUSES CAPTURE; LOW TIME = T2C.
3A: FALLING EDGE CAUSES CAPTURE/RELOAD; SINGLE-SHOT CAPTURE CYCLE ENDS; PERIOD = T2C. TIMER CONTINUES TO OPERATE SINCE TR2 = 1, ALLOWING THE NEXT LOW TIME/PERIOD TO BE MEASURED.
1B–3B: SAME SEQUENCE AS 1A–3A, EXCEPT THAT THE SINGLE-SHOT CAPTURE CYCLE DOES NOT BEGIN UNTIL THE FIRST FALLING EDGE IS DETECTED.

Figure 9-7. Timer 2 Application Example—Measure Duty Cycle

9.3.5 Overflow/Interrupt on Cumulative Time

To cause an overflow only when the T2P pin has been low for some cumulative duration, Timer 2 could be configured to the gated compare mode of operation with an initial starting value appropriate for the cumulative duration to be detected.

```
; ------------------ Reset State:  T2R = T2V = T2C = 0000h -----------------------
MOVE T2V, #1234h             ; Overflow after T2P input low for (10000h-01234h) T2CLKs
MOVE T2CFG, #01110000b       ; T2CI          =0     (sysclk/N input)
                             ; T2DIV[2:0]    =111   (/128)
                             ; T2MD          =0     (16-bit)
                             ; CCF[1:0]      =00    (no edges)
                             ; C/T2          =0     (timer/compare)
MOVE T2CNA, #10101001b       ; ET2           =1     (enable Timer 2 ints)
                             ; T2OE[0]       =0     (input)
                             ; T2POL[0]      =1     (gating level = '1')
                             ; TR2L:TR2      =01    (start timer 2)
                             ; CPRL2         =0     (no capture possible)
                             ; SS2           =0     (not single-shot mode)
                             ; G2EN          =1     (gating enabled)
; ------------------ TF2 Interrupt : Cumulative low duration reached
```

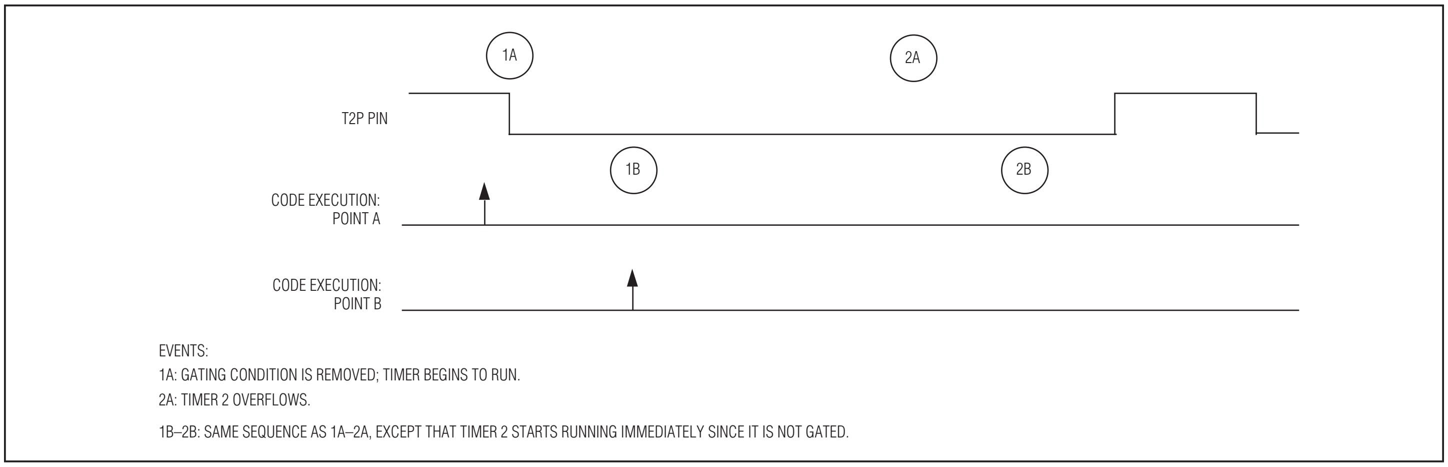

1A
2A
T2P PIN
1B
2B
CODE EXECUTION: POINT A
CODE EXECUTION: POINT B
EVENTS:
1A: GATING CONDITION IS REMOVED; TIMER BEGINS TO RUN.
2A: TIMER 2 OVERFLOWS.
1B–2B: SAME SEQUENCE AS 1A–2A, EXCEPT THAT TIMER 2 STARTS RUNNING IMMEDIATELY SINCE IT IS NOT GATED.

Figure 9-8. Timer 2 Application Example—Overflow/Interrupt on Cumulative Time

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

9.4.2 Timer/Counter 2 Control Register A (T2CNA)

Bit #	7	6	5	4	3	2	1	0
Name	ET2	T2OE0	T2POL0	TR2L	TR2	CPRL2	SS2	G2EN
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bit 7: Enable Timer 2 Interrupts (ET2). This bit serves as the local enable for Timer 2 interrupt sources that fall under the TF2 and TCC2 interrupt flags.

Bit 6: Timer 2 Output Enable 0 (T2OE0). This register bit enables the Timer 2 output function for the external T2P pin. The table below shows Timer 2 output possibilities for the T2P, T2PB pins.

T2OE[1:0]	T2MD	T2P PIN	T2PB PIN
00	X	Port Data	Port Data
01	0	16-Bit PWM Output	Port Data
10	0	Port Data	16-Bit PWM Output
11	0	16-Bit PWM Output	16-Bit PWM Output
01	1	8-Bit PWM Output (T2H)	Port Data
10	1	Port Data	8-Bit PWM Output (T2L)
11	1	8-Bit PWM Output (T2H)	8-Bit PWM Output (T2L)

Bit 5: Timer 2 Polarity Select 0 (T2POL0). When the Timer 2 output function has been enabled (T2OE0 = 1), the polarity select bit defines the starting logic level for the T2P output waveform. When T2POL0 = 0, the starting state for the T2P output will be logic low. When T2POL0 = 1, the starting state for the T2P output is logic high. The T2POL0 bit can only be modified when T2OE0 = 0 and takes effect on the external pin when T2OE0 is set to 1. When the Timer 2 pin is being used as an input (T2OE0 = 0), the polarity select bit defines which logic level can be used to gate the timer input clock (when CCF[1:0]<>11b). When CCF[1:0] = 11b, T2POL0 defines which edge can start/stop a single-shot capture and which edge reload can be skipped (if CPRL2 = 1 and G2EN = 1).

Bit 4: Timer 2 Low Run Enable (TR2L). This bit start/stops the low 8-bit Timer (T2L) when dual 8-bit mode (T2MD = 1) is in effect. This bit has no effect when T2MD = 0.

0 = Timer 2 Low stopped

1 = Timer 2 Low run

Bit 3: Timer 2 Run Enable (TR2). This bit starts/stop Timer 2. In the dual 8-bit mode of operation, this bit applies only to the T2H timer/counter. Otherwise, the bit applies to the full 16-bit T2H:T2L timer/counter. When the timer is stopped (TR2 = 0), the timer registers hold their count. The single-shot bit (SS2) can override and/or delay the effect of the TR2 bit.

0 = Timer 2 stopped

1 = Timer 2 run

Bit 2: Capture and Reload Enable (CPRL2). This bit enables a reload (in addition to a capture) on the edge specified by CCF[1:0] when operating in capture/reload mode ($C/\overline{T2}$ = 0). If both edges are defined for capture/reload (CCF[1:0] = 11b), enabling the gating control (G2EN = 1) allows the T2POL0 bit to be used to prevent a reload on one of the edges. If T2POL[0] is 0, no reload on the falling edge; if T2POL[0] is 1, no reload on the rising edge.

0 = capture on edge(s) specified by CCF[1:0] bits

1 = capture and reload on edge(s) specified by CCF[1:0] bits

Bit 1: Single Shot (SS2). This bit is used to automatically override or delay the effect of the TR2 bit setting. The single-shot bit is only useful in the timer mode of operation ($C/\overline{T2}$ = 0) and should not be set to 1 when the counter mode of operation is enabled ($C/\overline{T2}$ = 1).

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.

9.4.4 Timer 2 Value Register (T2V)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	T2V.15	T2V.14	T2V.13	T2V.12	T2V.11	T2V.10	T2V.9	T2V.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	T2V.7	T2V.6	T2V.5	T2V.4	T2V.3	T2V.2	T2V.1	T2V.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Timer 2 Value (T2V.[15:0]). The T2V register is a 16-bit register that holds the current Timer 2 value. When operating in 16-bit mode (T2MD = 0), the full 16 bits are read/write accessible. If the dual 8-bit mode of operation (T2MD = 1) is selected, the upper byte of T2V is inaccessible. T2V reads while in the dual 8-bit mode will return 00h as the high byte and writes to the upper byte of T2V will be blocked. A separate T2H register is provided to facilitate high byte access for dual 8-bit mode.

9.4.5 Timer 2 Value High Register (T2H)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	T2H.7	T2H.6	T2H.5	T2H.4	T2H.3	T2H.2	T2H.1	T2H.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Timer 2 Value High (T2H.[7:0]). This register is used to load and read the most significant 8-bit value in Timer 2.

9.4.6 Timer 2 Reload Register (T2R)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	T2R.15	T2R.14	T2R.13	T2R.12	T2R.11	T2R.10	T2R.9	T2R.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	T2R.7	T2R.6	T2R.5	T2R.4	T2R.3	T2R.2	T2R.1	T2R.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Timer 2 Reload (T2R.[15:0]). This 16-bit register holds the reload value for Timer 2. When operating in 16-bit mode (T2MD = 0), the full 16 bits are read/write accessible. If the dual 8-bit mode of operation is selected, the upper byte of T2R is inaccessible. T2R reads while in the dual 8-bit mode will return 00h as the high byte and writes to the upper byte of T2R will be blocked. A separate T2RH register is provided to facilitate high byte access for the dual 8-bit mode.

9.4.7 Timer 2 Reload High Register (T2RH)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	T2RH.7	T2RH.6	T2RH.5	T2RH.4	T2RH.3	T2RH.2	T2RH.1	T2RH.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Timer 2 Reload High (T2RH.[7:0]). This register is used to load and read the most significant 8-bit reload value in Timer 2.

9.4.8 Timer 2 Capture/Compare Register (T2C)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	T2C.15	T2C.14	T2C.13	T2C.12	T2C.11	T2C.10	T2C.9	T2C.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	T2C.7	T2C.6	T2C.5	T2C.4	T2C.3	T2C.2	T2C.1	T2C.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Timer 2 Capture/Compare (T2C.[15:0]). This 16-bit register that holds the compare value when operating in compare mode and gets the capture value when operating in capture mode. When operating in 16-bit mode (T2MD = 0), the full 16-bits are read/write accessible. If the dual 8-bit mode of operation is selected, the upper byte of T2C is inaccessible. T2C reads while in the dual 8-bit mode will return 00h as the high byte and writes to the upper byte of T2C will be blocked. A separate T2CH register is provided to facilitate high-byte access.

9.4.9 Timer 2 Capture/Compare High Register (T2CH)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	T2CH.7	T2CH.6	T2CH.5	T2CH.4	T2CH.3	T2CH.2	T2CH.1	T2CH.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Timer 2 Capture/Compare High (T2CH.[7:0]). This register is used to load and read the most significant 8-bit capture/compare value of Timer 2.

9.5 Low-Speed Infrared Transmit/Receive Support Using Timer 2

The MAXQ microcontroller can provide hardware to simplify support for low-speed infrared (IR) communication. To take advantage of the embedded hardware, the microcontroller device must be equipped with at least one Timer 2 module; that Timer must have at least two pins implemented and that Timer must be configured to a specific mode of operation.

The associated Timer 2 has to be configured into the dual 8-bit mode of operation. More specifically, it should be configured to the 8-bit Counter + 8-bit Timer/PWM mode. The T2OE[0] bit should be configured to logic 0 if it is intended that the T2 pin serve in the IRRX capacity. The T2OE[1] control bit should be configured to logic 0 if it is intended to serve in the IRTX capacity, since the internal IR hardware provides a separate control mechanism for enabling the carrier output to the T2PB pin. The 8-bit Timer/PWM is used to create the appropriate subcarrier waveform, while the 8-bit counter is used for modulation of the subcarrier when transmitting and for compare timing when decoding the IR receive waveforms. It is expected that the IR receive waveform will be coming directly from an external IR receiver module or circuitry, which can provide a filtered digital output indicating carrier presence by a logic 0 or logic 1.

9.5.1 Subcarrier Generation Using Timer 2 Low

Generation of the subcarrier frequency will always be performed by the 8-bit Timer/PWM (T2L). The period and duty cycle for the subcarrier is determined by the settings of reload and compare registers as described in the Timer 2 documentation. Figure 9-9 diagrams the basic subcarrier generation and its path to the T2PB/IRTX output pin. Notice that the T2POL[1] bit control still applies.

9.5.2 Transmit Baseband Modulator

Generation of the baseband modulator waveform is handled by the 8-bit counter (T2H). Normally, the 8-bit counter sources the external T2A pin signal and counts edges as defined by CCF[1:0]. However, when the IR hardware is enabled (IREN = 1), this counter sources the output of the T2L subcarrier waveform. This allows user software to define the number of subcarrier cycles through the T2RH register, which should be counted before T2H overflow. The T2POL[1] bit defines the starting (idle) state for the T2L output and the edge that are counted by the T2H counter. If T2POL[1] = 1, the T2L output idles high and only rising edges are counted by T2H. If T2POL[1] = 0, the T2L output idles low and only falling edges are counted by T2H. A separate register bit, IR bit-bang (IRBB), is used to determine whether the T2L output is gated or output to the pin for the next X subcarrier cycles. The value of X, as alluded to earlier, can be controlled by modifying the reload value. When IRBB = 1 and IRTX = 1, the T2L output is enabled onto the T2PB/IRTX pin. When IRBB = 0 and IRTX = 1, the gated (idle) condition, as defined by T2POL[1], is in effect on the pin.

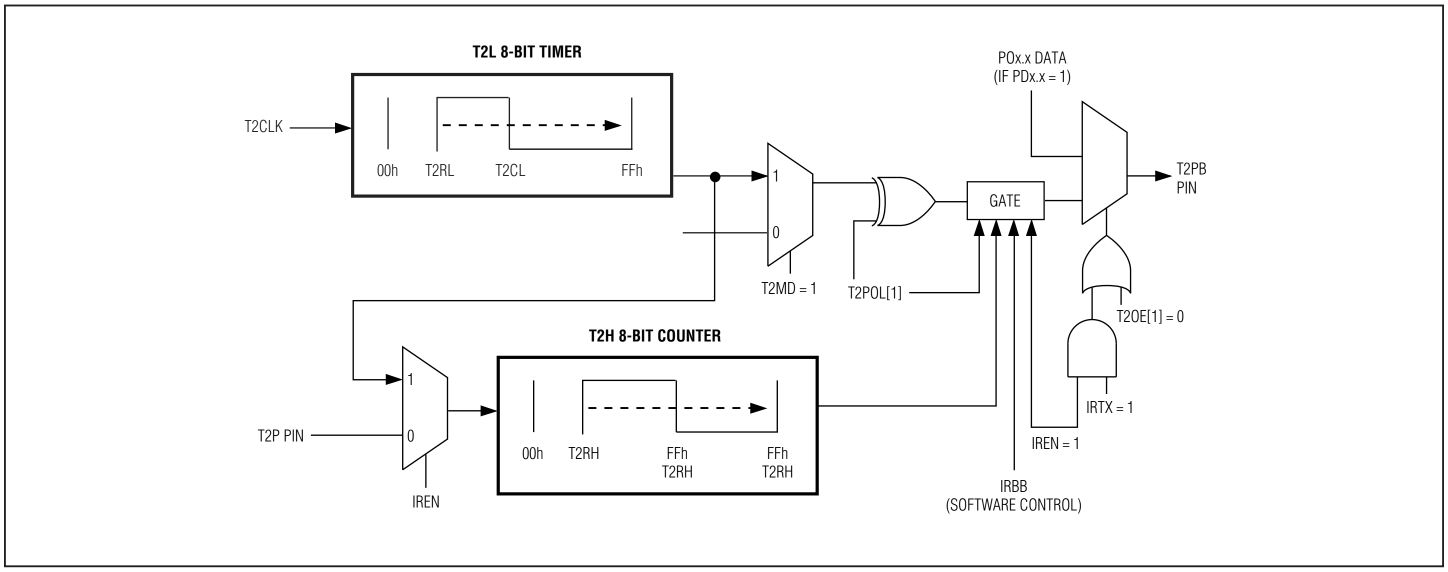


Figure 9-9. IR Transmit Subcarrier Generation and Baseband Modulator Control

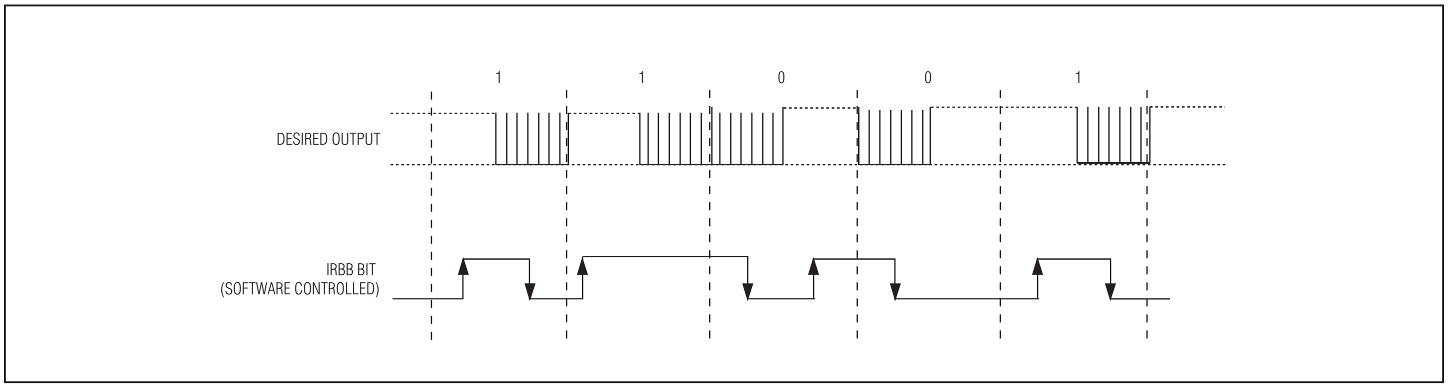


Figure 9-10. Biphase Encoding Example (T2RH Remains Fixed)

9.5.2.1 IR Encoding (Transmit) Example

For any encoding scheme, the proper T2L subcarrier generation settings should be established along with the desired T2POL[1] bit state. The T2POL[1] state takes effect once IREN = 1 and IRTX = 1.

For biphase encoding, the T2H reload value (T2RH) would be configured to count X subcarrier pulses in one half a bit time. The IRTX enable bit should be configured to logic 1 to block the functionality of the T2 pin used in receive mode. The IRBB bit would initially be configured to 0 to make the gated condition the starting state on the T2PB pin. At this point, the timer run bits for both the 8-bit timer/counters, which should be set. Once the timers are running, the software could then modify the IRBB control bit as needed for each half-bit time. On each T2H overflow, the same T2RH value is used as the reload and IRBB would be used to control whether the T2L subcarrier output is enabled to the output pin or gated. This process would continue until the desired number of bits are transmitted, at which point the user software would be responsible for placing the IR hardware back into receive mode or turning it off entirely.

For bit length encoding, the T2RH reload value could be modified selectively dependent upon whether a 0 or 1 subcarrier duration needed to be transmitted. In this case, the IRBB control function would remain the same and would typically be toggled on every overflow. Figure 9-11 illustrates an example of bit-length encoding.

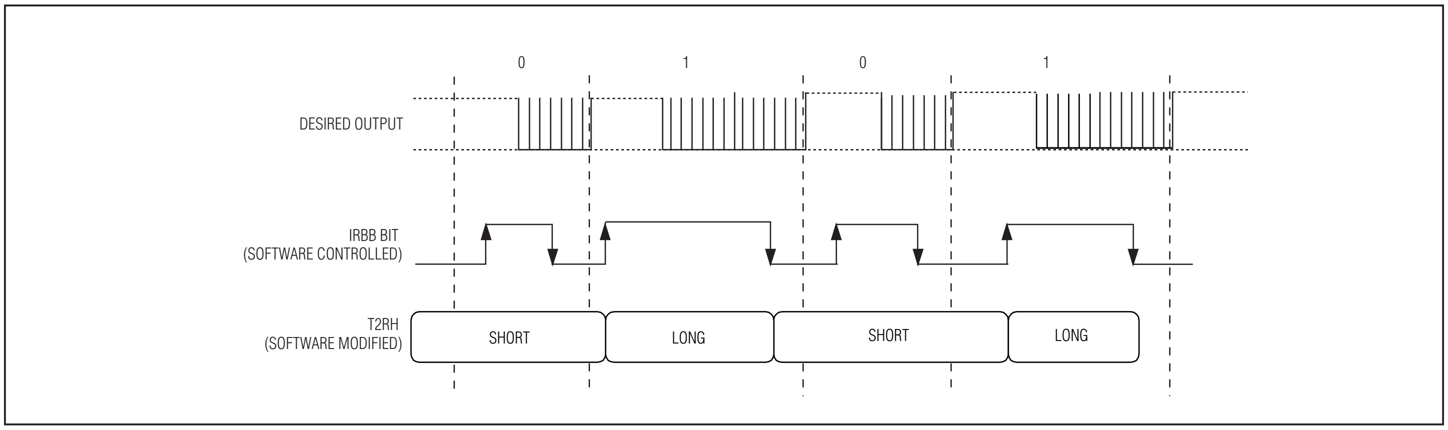


Figure 9-11. Bit-Length Encoding Example (T2RH Modified)

9.5.2.2 Receive Pin Sampling

When IREN = 1 and IRTX = 0, the IR hardware supports the T2H register counting of internal T2L edges just as described for the IR Transmit mode, but the function of the IRBB bit changes. The IRBB bit is used to store the state of the T2P input pin when a compare match occurs between the T2H and T2CH registers. Additionally, the CCF[1:0] bits define which edge(s) of the T2 pin should trigger reloads of the T2H counter to allow some form of synchronization when slightly different transmitter/receiver carrier frequencies and bit timing exist. The user software would be responsible for reading the IRBB sampled pin states and recreating, based upon the IR encoding format, the actual received data. The IRBB bit can be overwritten, thus the user software is responsible for reading the IRBB bit between compare matches to avoid loss of captured pin data.

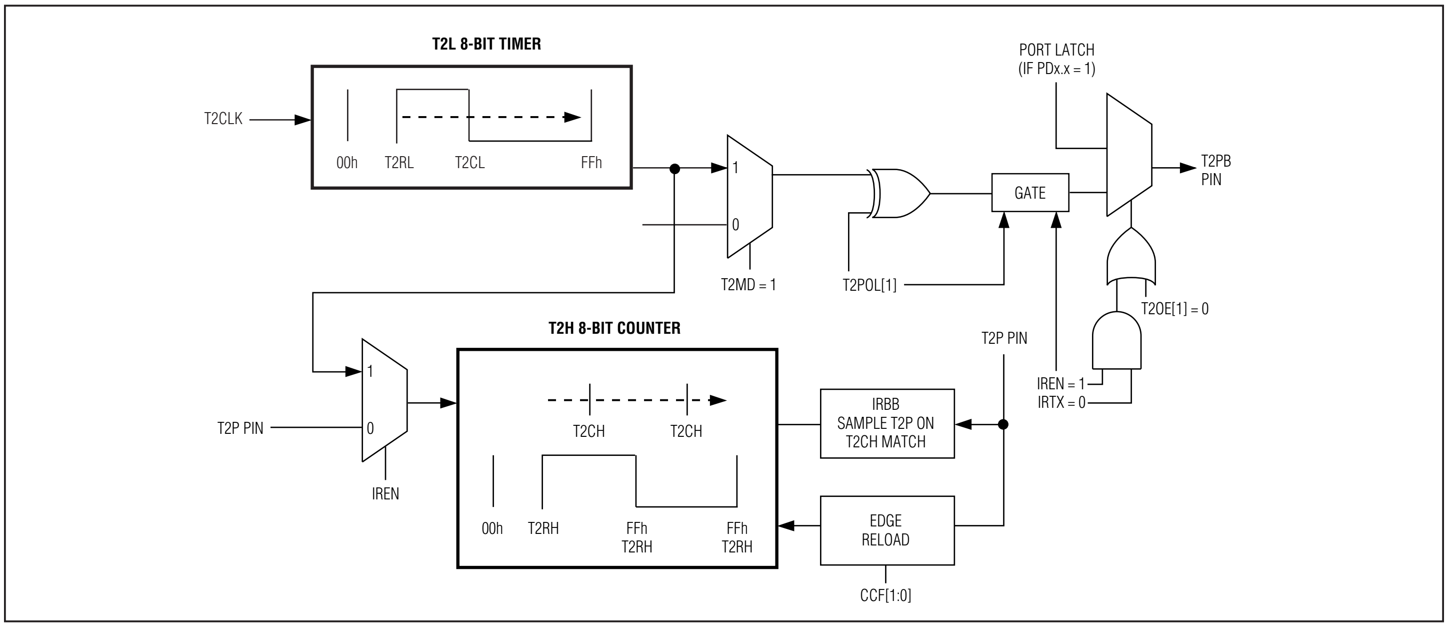


Figure 9-12. IR Receive Pin Sampling

9.5.2.3 IR Decoding (Receive) Example

One possible decoding configuration is shown in Figure 9-13 with the T2CH register configured to produce a match after approximately 1/4 of the subcarrier cycles present during a fixed bit time. Each of the two IRBB samples could be examined to determine one received bit.

Bit length decoding could use a similar strategy, possibly configuring the T2CH match register to 3/4 the number of subcarrier cycles present in a 0 bit time. The collected IRBB pattern could simply be inverted to produce the actual input stream.

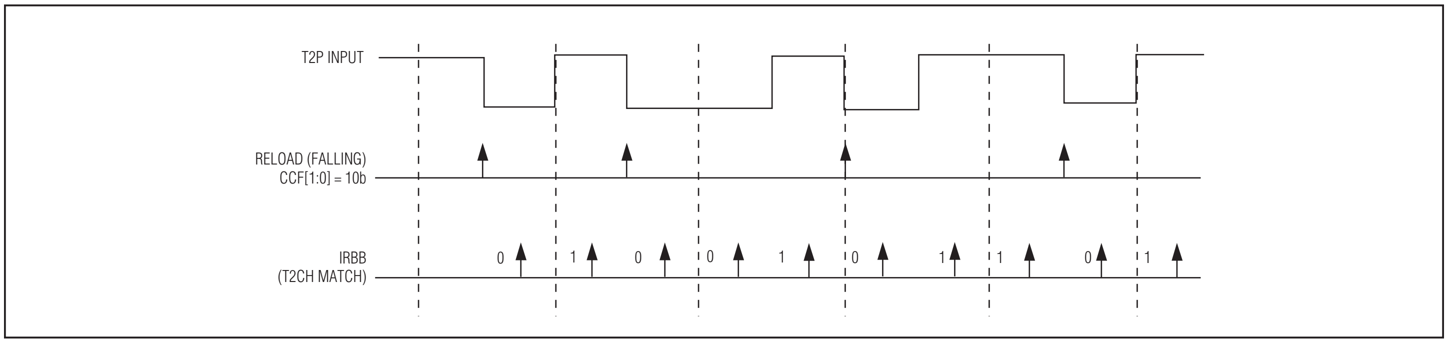


Figure 9-13. Biphase Decoding Example

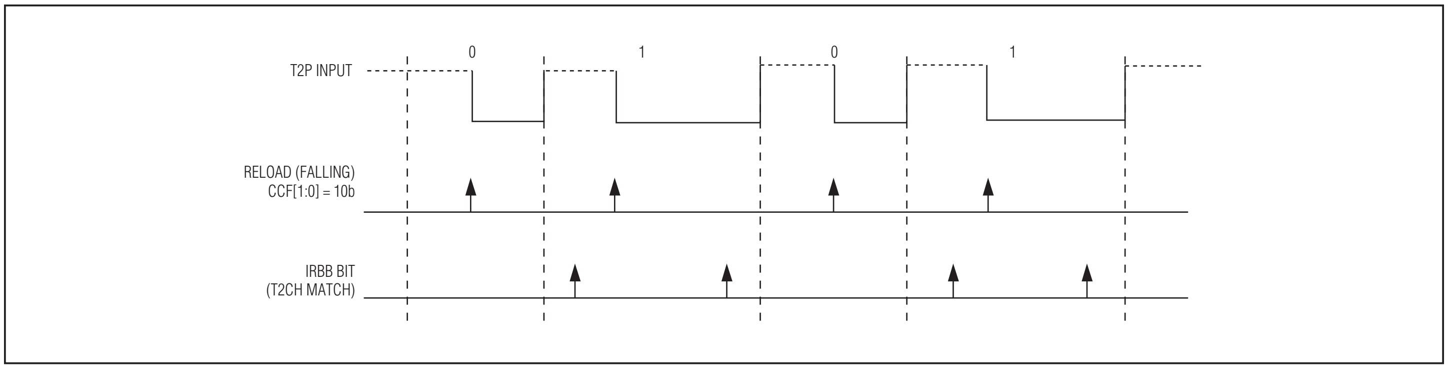


Figure 9-14. Bit Length Decoding Example

9.6 IR Peripheral Register

9.6.1 Infrared Control Register (IRCN)

Bit #	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	IREN	IRTX	IRBB
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	rw	rw	rw

r = read, w = write

Bits 7 to 3: Reserved

Bit 2: Infrared Subcarrier Enable (IREN). This register bit enables a special mode of operation for Timer 2. To use the IR hardware (i.e., before setting IREN = 1), Timer 2 should be configured properly to the dual 8-bit timer/8-bit counter mode. Setting IREN = 1 enables the IRRX mode if IRTX = 0, and enables the IRTX mode if IRTX = 1. In both cases, the T2L 8-bit timer output is fed as input to the T2H 8-bit counter.

Bit 1: Infrared Transmit Enable (IRTX). This register bit controls the contextual usage of the IRBB bit. When IRTX = 0, the IRBB bit captures the T2P(IRRX) pin state on compare matches. When IRTX = 1, the IRBB enables the T2L subcarrier output to the T2PB output pin on the next T2H counter overflow. This bit has no effect when IREN = 0.

Bit 0: Infrared Bit Bang Bit (IRBB). This register bit serves different purposes depending upon whether the IR Transmit Enable is configured as a 1 or 0. When IRTX = 1, the IR transmit mode is in effect and setting the IRBB bit to logic 1 enables the T2POL[1] modified T2L output starting at the next T2H counter overflow (allowing the subcarrier to be output to the pin). The user software is responsible for manually toggling IRBB and controlling the T2RH reload to achieve the desired protocol. For receive mode, IRTX = 0, the IRBB bit contains the latched state of the IRRX pin each time that a compare match (with T2CH) occurs. The user software is responsible for unloading IRBB and translating the recorded bit stream into the proper IR serial receive data.

SECTION 10: SERIAL I/O MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 10: SERIAL I/O MODULE

The Serial I/O Module provides the MAXQ access to a universal asynchronous receiver/transmitter (UART) for serial communication with framing error detection.

10.1 UART Modes

The UART supports four basic modes of operation, and is capable of both synchronous and asynchronous modes, with different protocols and baud rates. In the synchronous mode, the microcontroller supplies the clock and communication takes place in a half-duplex manner, while the asynchronous mode supports full-duplex operation. Table 10-1 shows the four serial operating modes.

The UART has a control register (SCON) and a transmit/receive buffer register (SBUF). Transmit or receive buffer access depends upon whether SBUF is used contextually as a source or destination. When SBUF is used as a source (read operation), the receive buffer will be accessed. When SBUF is used as a destination (write operation), the transmit buffer is accessed. The UART receiver incorporates a holding buffer so that it may receive an incoming word before software has read the previous one.

Please note that there is no single register bit that explicitly enables the UART for transmission. This means that the port pin(s) associated with UART transmission (i.e., TXD, and RXD for mode 0) will be controlled by the PDx and POx port control register bits when the UART is not actively transmitting a character.

Table 10-1. UART Mode Summary

MODE	SYNCHRONOUS/ ASYNCHRONOUS	BAUD CLOCK*	DATA BITS	START/STOP	9TH BIT FUNCTION
0	Synchronous	4 or 12 clocks	8	None	None
1	Asynchronous	Baud Clock Generator	8	1 start, 1 stop	None
2	Asynchronous	32 or 64 clocks	9	1 start, 1 stop	0, 1, parity
3	Asynchronous	Baud Clock Generator	9	1 start, 1 stop	0, 1, parity

**Use of any system clock-divide modes or power management mode affects the baud clock.*

10.1.1 UART Mode 0

This mode is used to communicate in synchronous, half-duplex format with devices that accept the MAXQ microcontroller as a master. Figure 10-1 shows a functional block diagram and basic timing of this mode. As can be seen, there is one bidirectional data line (RXD) and one shift clock line (TXD) used for communication. Mode 0 requires that the MAXQ microcontroller be the master since it generates the serial shift clock for data transfers that occur in either direction.

The RXD signal is used for both transmission and reception. Data bits enter and exit LSb first. TXD provides the shift clock. The baud rate is equal to the shift clock frequency. When not using Power Management Mode, the baud rate in Mode 0 is equivalent to the system clock divided by either 12 or 4, as selected by SM2 bit (SCON.5) for the UART.

The UART begins transmitting when any instruction writes to SBUF. The internal shift register then begins to shift data out. The clock is activated and transfers data until the 8-bit value is complete. Data is presented just prior to the falling edge of the shift clock (TXD) so that an external device can latch the data using the rising edge.

The UART begins to receive data when the REN bit in the SCON register (SCON.4) is set to logic 1 and the RI bit (SCON.0) is set to logic 0. This condition tells the UART that there is data to be shifted in. The shift clock (TXD) activates, and the UART latches incoming data on the rising edge. The external device should therefore present data on the falling edge. This process continues until 8 bits have been received. The RI bit is automatically set to logic 1 immediately following the last rising edge of the shift clock on TXD. This causes reception to stop until the SBUF has been read and the RI bit cleared. When RI is cleared, another byte can be shifted in.

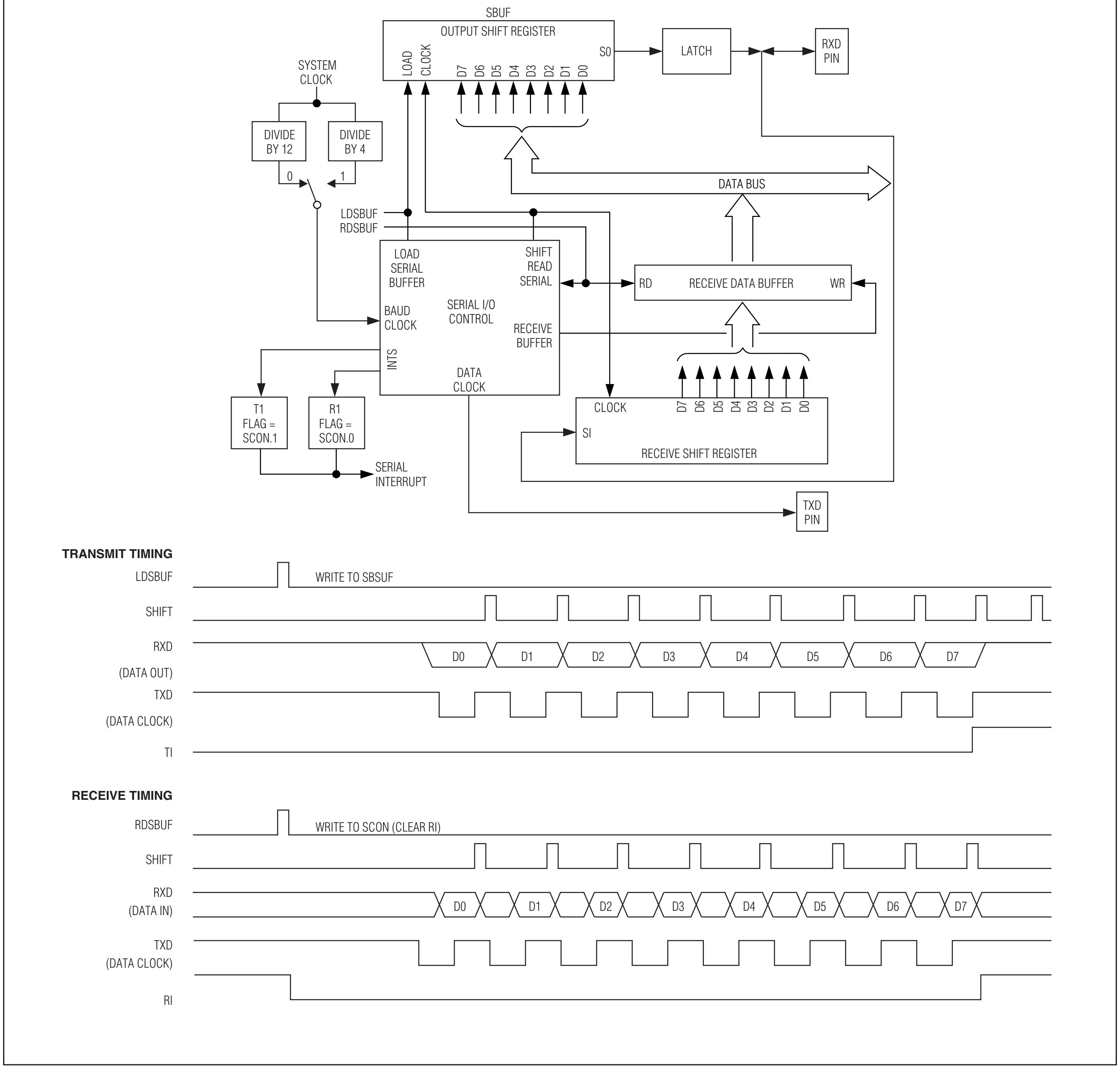


Figure 10-1. UART Mode 0

10.1.2 UART Mode 1

This mode provides asynchronous, full-duplex communication. A total of 10 bits is transmitted, consisting of a start bit (logic 0), 8 data bits, and 1 stop bit (logic 1), as illustrated in Figure 10-2. The data is transferred LSb first. The baud rate is programmable through the baud clock generator. Following a write to SBUF, the UART begins transmission five cycles after the first baud clock from the baud clock generator. Transmission takes place on the TXD pin. It begins with the start bit being placed on the pin. Data is then shifted out onto the pin, LSb first. The stop bit follows. The TI bit is set by hardware after the stop bit is placed on the pin. All bits are shifted out at the rate determined by the baud clock generator.

Once the baud clock generator is active, reception can begin at any time. The REN bit (SCON.4) must be set to logic 1 to allow reception. The detection of a falling edge on the RXD pin is interpreted as the beginning of a start bit, and will begin the reception process. Data is shifted in at the selected baud rate. At the middle of the stop bit time, certain conditions must be met to load SBUF with the received data:

RI must = 0, and either

If SM2 = 0, the state of the stop bit does not matter

or

If SM2 = 1, the state of the stop bit must = 1.

If these conditions are true, then SBUF (address) is loaded with the received byte, the RB8 bit (SCON.2) is loaded with the stop bit, and the RI bit (SCON.0) is set. If these conditions are false, then the received data will be lost (SBUF and RB8 not loaded) and RI will not be set. Regardless of the receive word status, after the middle of the stop bit time, the receiver goes back to looking for a 1-to-0 transition on the RXD pin.

Each data bit received is sampled on the 7th, 8th and 9th clock used by the divide-by-16 counter. Using majority voting, two equal samples out of the three determine the logic level for each received bit. If the start bit was determined to be invalid (= 1), then the receiver goes back to looking for a 1-to-0 transition on the RXD pin to start the reception of data.

10.1.3 UART Mode 2

This mode uses a total of 11 bits in asynchronous, full-duplex communication as illustrated in Figure 10-3. The 11 bits consist of one start bit (a logic 0), 8 data bits, a programmable 9th bit, and one stop bit (a logic 1). Like Mode 1, the transmissions occur on the TXD signal pin and receptions on RXD.

For transmission purposes, the 9th bit can be stuffed as a logic 0 or 1. The 9th bit is transferred from the TB8 bit position in the SCON register (SCON.3) following a write to SBUF to initiate a transmission. Transmission begins five clock cycles after the first rollover of the divide-by-16 counter following a software write to SBUF. It begins with the start bit being placed on the TXD pin. The data is then shifted out onto the pin, LSb first, followed by the 9th bit, and finally the stop bit. The TI bit (SCON.1) is set when the stop bit is placed on the pin.

Once the baud-rate generator is active and the REN bit (SCON.4) has been set to logic 1, reception can begin at any time. Reception begins when a falling edge is detected as part of the incoming start bit on the RXD pin. The RXD pin is then sampled according to the baud-rate speed. The 9th bit is placed in the RB8 bit location in SCON (SCON.2). At the middle of the 9th bit time, certain conditions must be met to load SBUF with the received data.

RI must = 0, and either

If SM2 = 0, the state of the 9th bit does not matter

or

If SM2 = 1, the state of the 9th bit must = 1.

If these conditions are true, then SBUF will be loaded with the received byte, RB8 will be loaded with the 9th bit, and RI will be set. If these conditions are false, then the received data will be lost (SBUF and RB8 not loaded) and RI will not be set. Regardless of the receive word status, after the middle of the stop bit time, the receiver goes back to looking for a 1-to-0 transition on RXD.

Data is sampled in a similar fashion to Mode 1 with the majority voting on three consecutive samples. Mode 2 uses the sample divide-by-16 counter with either the clock divided by 2 or 4, thus resulting in a baud clock of either system clock/32 or system clock/64.

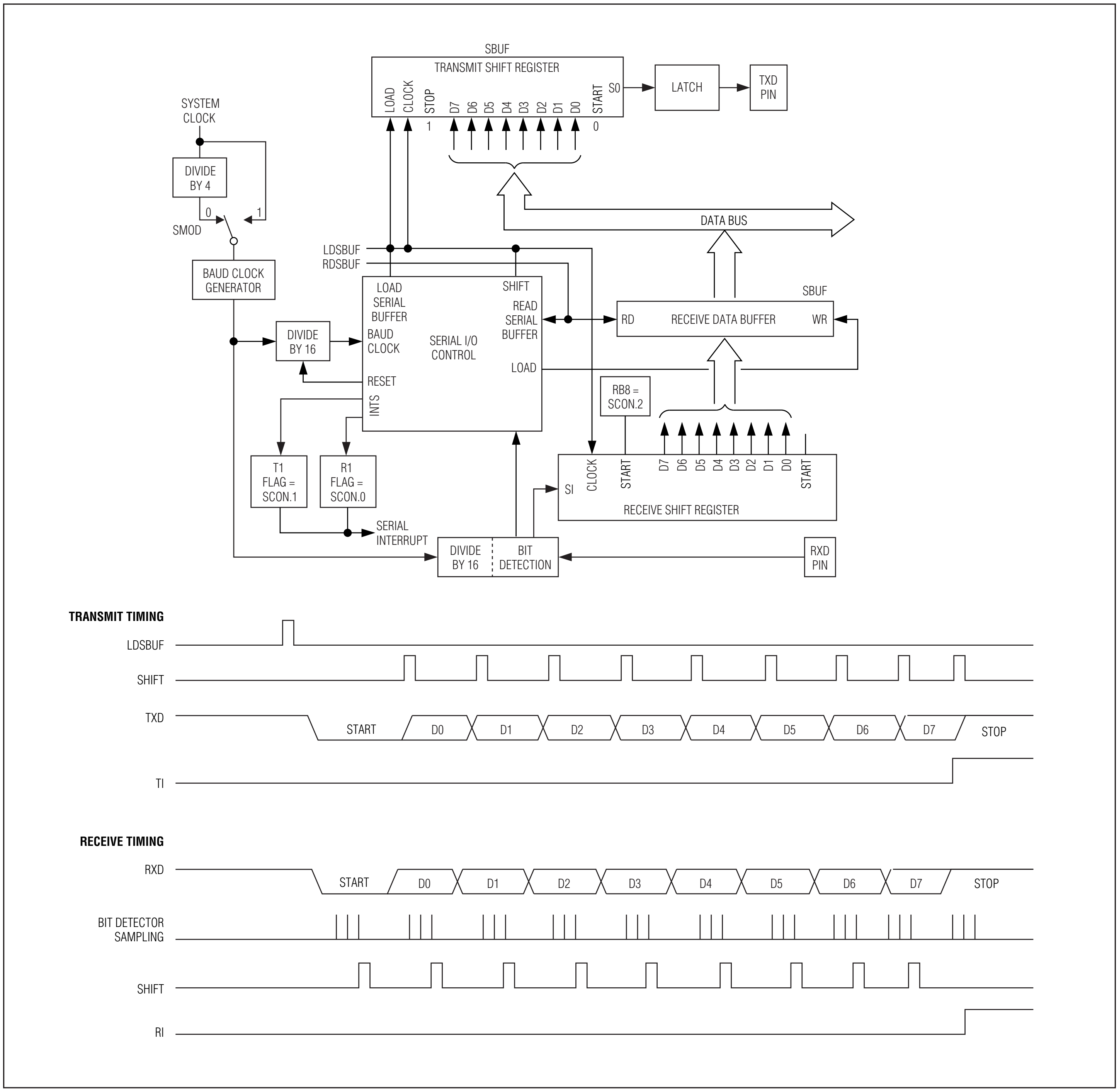


Figure 10-2. UART Mode 1

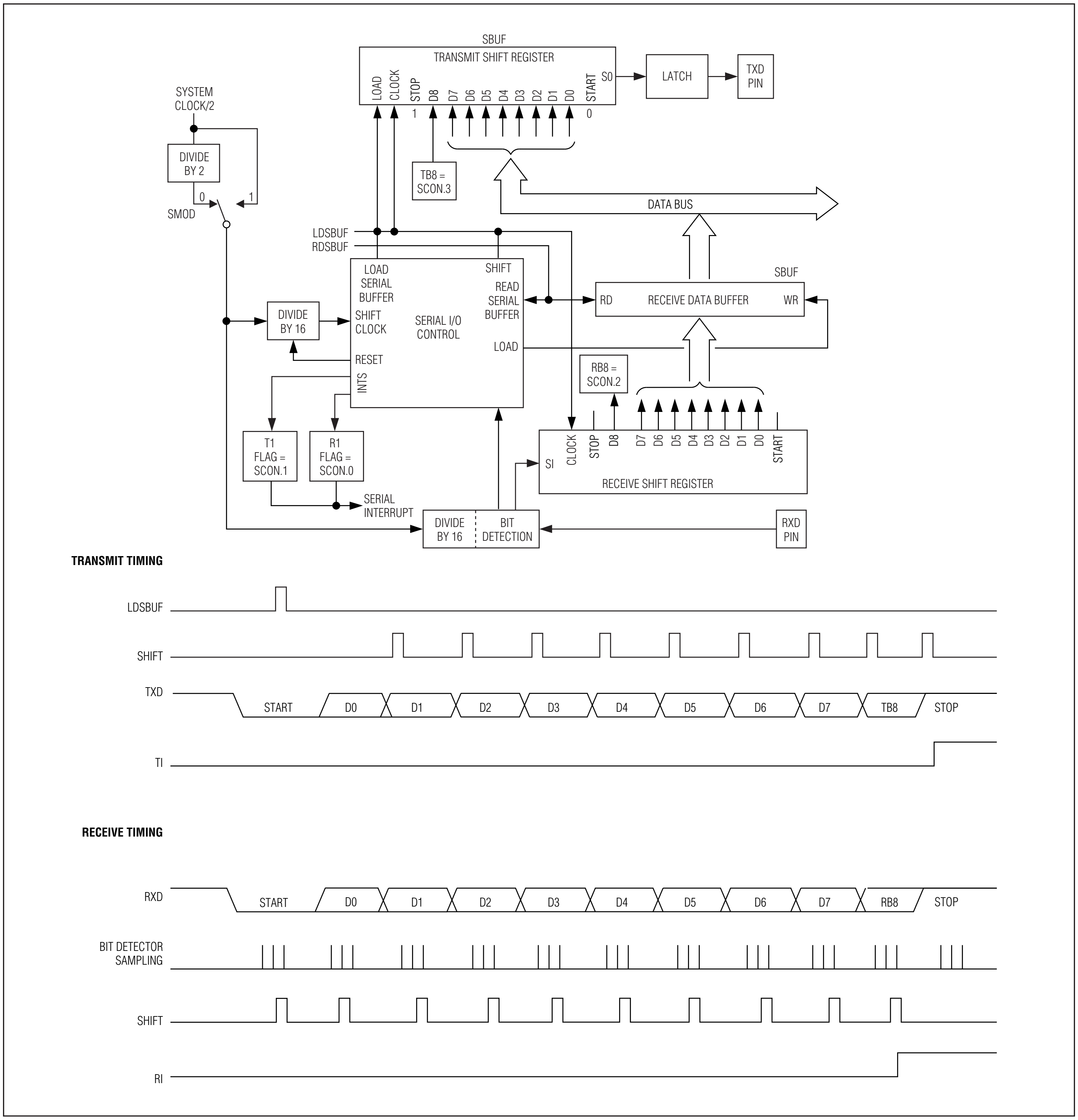


Figure 10-3. UART Mode 2

10.1.4 UART Mode 3

This mode has the same operation as Mode 2, except for the baud-rate source. As shown in Figure 10-4, Mode 3 generates baud rates through the baud clock generator. The bit shifting and protocol are the same.

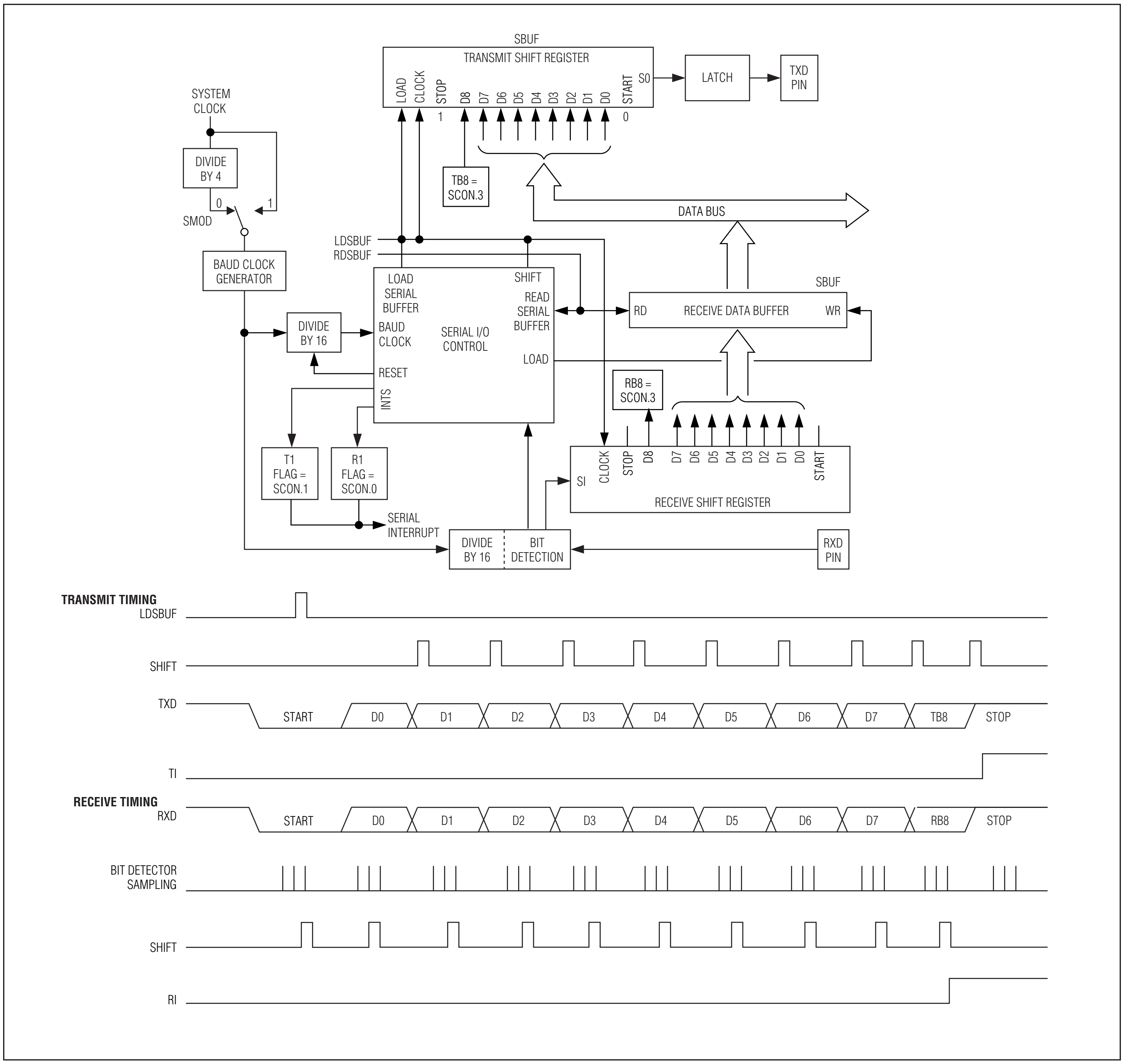


Figure 10-4. UART Mode 3

10.2 Baud-Rate Generation

Each mode of operation has a baud-rate generator associated with it. The baud-rate generation techniques are affected by certain user options such as the Power Management Mode Enable (PMME), Serial Mode 2 (SM2) select bit, and Baud-Rate Doubler (SMOD) bit. Table 10-2 summarizes the effects of the various user options on the UART baud clock.

Table 10-2. UART Baud Clock Summary

SYSTEM CLOCK MODE	BAUD CLOCK FREQUENCY					
	MODE 0		MODE 2		MODE 1, 3[†]	
	SM2 = 0	SM2 = 1	SMOD = 0	SMOD = 1	SMOD = 0	SMOD = 1
Divide by 1 (default)	CLK /12	CLK /4	CLK /64	CLK /32	BAUD /64	BAUD /16
Divide by 2	CLK /24	CLK /8	CLK /128	CLK /64	BAUD /64	BAUD /16
Divide by 4	CLK /48	CLK /16	CLK /256	CLK /128	BAUD /64	BAUD /16
Divide by 8	CLK /96	CLK /32	CLK /512	CLK /256	BAUD /64	BAUD /16
Power Management Mode (Divide by 256)	CLK /3072	CLK /1024	CLK /16384	CLK /8192	BAUD /64	BAUD /16

[†]*The BAUD frequency is determined by the baud-clock generator.*

10.2.1 Mode 0 Baud Rate

Baud rates for mode 0 are driven directly from the system clock source divided by either 12 or 4, with the default case being divide by 12. The user can select the shift clock frequency using the SM2 bit in the SCON register. When SM2 is set to logic 0, the baud rate is fixed at a divide by 12 of the system clock. When SM2 is set to logic 1, the baud rate is fixed at a divide by 4 of the system clock.

$$\text{Mode 0 Baud Rate} = \text{System Clock Frequency} \times 3^{SM2} / 12$$

10.2.2 Mode 2 Baud Rate

In this asynchronous mode, baud rates are also generated from the system clock source. The user can effectively double the UART baud clock frequency by setting the SMOD bit to a logic 1. The SMOD bit is set to a logic 0 on all resets, thus making 'divide by 64' the default setting. The baud rate is given by the following formula:

$$\text{Mode 2 Baud Rate} = \text{System Clock Frequency} \times 2^{SMOD} / 64$$

10.2.3 Mode 1 or 3 Baud Rate

These asynchronous modes are commonly used for communication with PCs, modems, and other similar interfaces. The baud rates are programmable using the baud clock generator in the UART module. The baud clock generator is basically a phase accumulator that generates a baud clock as the result of phase overflow into the most significant bit of the phase shifter. This baud-clock generator is driven by the system clock or system clock divided-by-4 source (depending upon the state of the SMOD bit). The baud-clock-generator output is always divided by 16 to generate the exact baud rate.

10.2.4 Baud-Clock Generator

The baud-clock generator is basically a phase accumulator that produces a baud clock as the result of phase overflow from the most significant bit of the phase shift circuitry. A 16-bit Phase Register (PR) is programmable by the user to select a suitable phase value for its baud clock. The phase value dictates the phase period of the accumulation process. The phase value is added to the current phase accumulator value on each system clock (SMOD = 1) or every 4th system clock (SMOD = 0). The baud clock is the result of addition overflow out of the most significant bit of the phase accumulator (bit 16). The baud-clock-generator output is always divided by 16 to produce the exact baud rate.

The following two formulas can be used to calculate the output of the baud-clock generator and the resultant Mode 1, 3 baud rates. Additionally, Table 10-3 gives example phase register (PR) settings needed to produce some more common baud rates at certain system clock frequencies (assuming SMOD = 1).

$$\text{Baud Clock Generator Output (BAUD)} = \text{System Clock Frequency} \times \text{PR} / 2^{17}$$

$$\text{Baud Rate for Modes 1 and 3} = \text{BAUD} \times 2^{(SMOD \times 2)} / 2^{6}$$

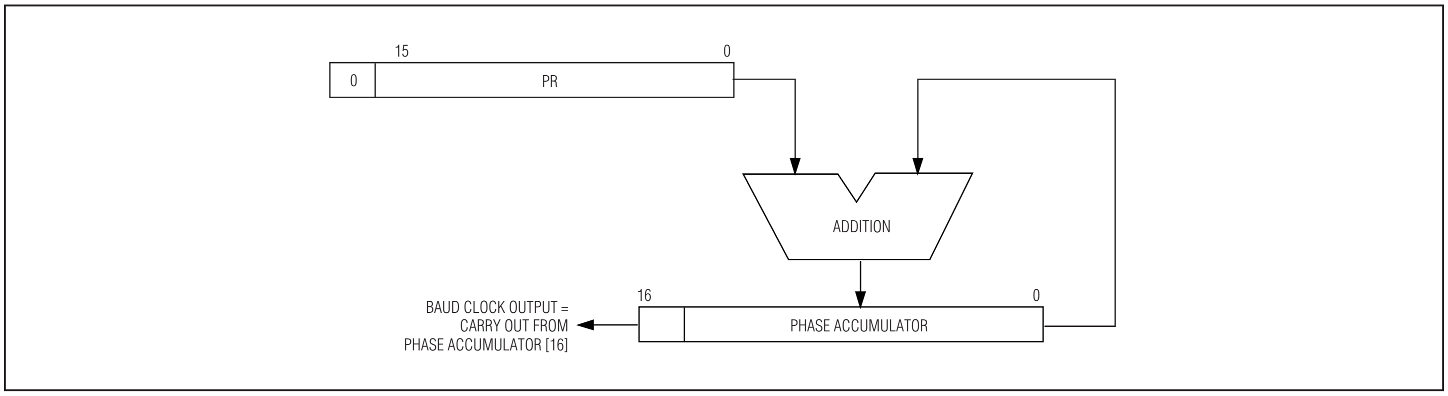


Figure 10-5. Baud-Clock Generator

Table 10-3. Example Baud-Clock Generator Settings (SMOD = 1)

SYSTEM CLOCK FREQUENCY (MHz)	BAUD RATE (PR SETTING)	SYSTEM CLOCK FREQUENCY (MHZ)	BAUD RATE (PR SETTING)
10	115,200 (5E5F); 57,600 (2F30); 19,200 (0FBB); 9600 (07DD); 2400 (01FF)	3.579545	57,600 (83D2); 19,200 (2BF1); 9600 (15F8); 2400 (057E)
8	115,200 (75F7); 57,600 (3AFB); 19,200 (13A9); 9600 (09D5); 2400 (0275)	2.4576	57,600 (C000); 19,200 (4000); 9600 (2000); 2400 (0800)
3.6864	115,200 (FFFF); 57,600 (8000); 19,200 (2AAB); 9600 (1555); 2400 (0555)	1	19,200 (9D49); 9600 (4EA5); 2400 (13A9)

10.3 Framing Error Detection

A framing error occurs when a valid stop bit is not detected. This results in the possible improper reception of the serial word. The UART can detect a framing error and notify the software. Typical causes of framing errors are noise and contention. The Framing Error condition is reported in the SCON register for the UART.

The Framing Error bit, FE, is located in SCON.7. Note that this bit normally serves as SM0 and is described as SM0/FE_0 in the register description. Framing Error information is made accessible by the FEDE (Framing Error Detection Enable) bit located at SMD.0. When FEDE is set to logic 1, the framing error information is shown in SM0/FE (SCON.7). When FEDE is set to logic 0, the SM0 function is accessible. The information for bits SM0 and FE is actually stored in different registers. Changing FEDE only modifies which register is accessed, not the contents of either.

The FE bit is set to 1 when a framing error occurs. It must be cleared by software. Note that the FEDE state must be 1 while reading or writing the FE bit. Also note that receiving a properly framed serial word does not clear the FE bit. This must be done in software.

10.4 UART Peripheral Registers

10.4.1 Serial Control Register (SCON)

Bit #	7	6	5	4	3	2	1	0
Name	FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bit 7: Framing Error Flag (FE). (FEDE = 1) This bit is set upon detection of an invalid stop bit. It must be cleared by software. Modification of this bit when FEDE is set has no effect on the serial mode setting.

Bit 7: Serial Port 0 Mode Bit 0 (SM0). (FEDE = 0) This bit is used in conjunction with the SM2 and SM1 bits to define the serial mode.

MODE	SM[2:0]	FUNCTION	LENGTH	PERIOD
0	0 0 0	Synchronous	8 Bits	12 System Clock
0	1 0 0	Synchronous	8 Bits	4 System Clock
1	x 1 0	Asynchronous	10 Bits	64/16 Baud Clock (SMOD = 0/1)
2	0 0 1	Asynchronous	11 Bits	64/32 System Clock (SMOD = 0/1)
2	1 0 1	Asynchronous (MP)	11 Bits	64/32 System Clock (SMOD = 0/1)
3	0 1 1	Asynchronous	11 Bits	64/16 Baud Clock (SMOD = 0/1)
3	1 1 1	Asynchronous (MP)	11 Bits	64/16 Baud Clock (SMOD = 0/1)

Bit 6: Serial Port 0 Mode Bit 1 (SM1). See the above table for more information.

Bit 5: Serial Port 0 Mode Bit 2 (SM2). Setting this bit in mode 1 ignores reception if an invalid stop bit is detected. Setting this bit in mode 2 or 3 enables multiprocessor communications, and prevents the RI bit from being set and the interrupt from being asserted if the 9th bit received is 0. See the above table for more information. This bit is also used to support mode 0 for clock selection:

0 = serial clock is system clock divided by 12

1 = serial clock is system clock divided by 4

Bit 4: Receive Enable (REN)

0 = serial port receiver disabled

1 = serial port receiver enabled for modes 1, 2, and 3; initiate synchronous reception for mode 0 (if RI = 0)

Bit 3: 9th Transmission Bit State (TB8). This bit defines the state of the 9th transmission bit in serial port modes 2 and 3.

Bit 2: 9th Received Bit State (RB8). This bit identifies the state of the 9th bit of received data in serial port modes 2 and 3. When SM2 is 0, it is the state of the stop bit in mode 1. This bit has no meaning in mode 0.

Bit 1: Transmit Interrupt Flag (TI). This bit indicates that the data in the serial port data buffer has been completely shifted out. It is set at the end of the last data bit for all modes of operation and must be cleared by software once set.

Bit 0: Receive Interrupt Flag (RI). This bit indicates that a data byte has been received in the serial port buffer. The bit is set at the end of the 8th bit for mode 0, after the last sample of the incoming stop bit for mode 1 subject to the value of the SM2 bit, or after the last sample of RB8 for modes 2 and 3. This bit must be cleared by software once set.

10.4.2 Serial Port Mode Register (SMD)

Bit #	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	ESI	SMOD	FEDE
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	rw	rw	rw

r = read, w = write

Bits 7 to 3: Reserved

Bit 2: Framing Error Detection Enable (FEDE). This bit selects the function of SM0 (SCON.7):

0 = SCON.7 functions as SM0 for serial port mode selection

1 = SCON.7 is converted to the Framing Error (FE) flag

Bit 1: Serial Port Baud Rate Select (SMOD). The SMOD selects the final baud rate for the asynchronous mode:

1 = 16 times the baud clock for mode 1 and 3, 32 times the system clock for mode 2

0 = 64 times the baud clock for mode 1 and 3, 64 times the system clock for mode 2

Bit 0: Enable Serial Port Interrupt (ESI). Setting this bit to 1 enables interrupt requests generated by the RI or TI flags in SCON. Clearing this bit to 0 disables the serial port interrupt.

10.4.3 Serial Port Data Buffer Register (SBUF)

Bit #	7	6	5	4	3	2	1	0
Name	SBUF.7	SBUF.6	SBUF.5	SBUF.4	SBUF.3	SBUF.2	SBUF.1	SBUF.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Serial Port Data Buffer (SBUF.[7:0]). Data for serial port is read from or written to this location. The serial transmit and receive buffers are separate but both are addressed at this location.

10.4.4 Serial Port Phase Register (PR)

Bit #	15	14	13	12	11	10	9	8
Name	PR.15	PR.14	PR.13	PR.12	PR.11	PR.10	PR.9	PR.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	PR.7	PR.6	PR.5	PR.4	PR.3	PR.2	PR.1	PR.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Serial Port Phase (PR.[15:0]). This register is used to load and read the value in the phase register.

SECTION 11: SERIAL PERIPHERAL INTERFACE (SPI) MODULE

This section contains the following information:

LIST OF FIGURES

SECTION 11: SERIAL PERIPHERAL INTERFACE (SPI) MODULE

The serial peripheral interface (SPI) module of the MAXQ microcontroller provides an independent serial communication channel to communicate synchronously with peripheral devices in a multiple master or multiple slave system. The interface allows access to a four-wire full-duplex serial bus that can be operated in either master mode or slave mode. The SPI functionality must be enabled by setting the SPI Enable (SPIEN) bit of the SPI Control register to logic 1. The maximum data rate of the SPI interface is 1/2 the system clock frequency for master mode operation and 1/8 the system clock frequency for slave mode operation. The four external interface signals used by the SPI module are MISO, MOSI, SPICK, and $\overline{\text{SSEL}}$. The function of each of these signals is as follows:

EXTERNAL PIN SIGNAL	MASTER MODE USE	SLAVE MODE USE
MISO: Master In, Slave Out	Input to serial shift register	Output from serial shift register when selected
MOSI: Master Out, Slave In	Ouput from serial shift register	Input to serial shift register when selected
SPICK: SPI Clock	Serial shift clock sourced to slave device(s)	Serial shift clock from an external master
$\overline{\text{SSEL}}$: Slave Select	(Optional) Mode fault-detection input if enabled (MODFE = 1)	Slave select input

The block diagram in Figure 11-1 shows the SPI external interface signals, control unit, read buffer, and single shift register common to the transmit and receive data path. Each time that an SPI transfer completes, the received character is transferred to the read buffer, giving double buffering on the receive side. The CPU has read/write access to the control unit and the SPI data buffer (SPIB). Writes to SPIB are always directed to the shift register while reads always come from the receive holding buffer.

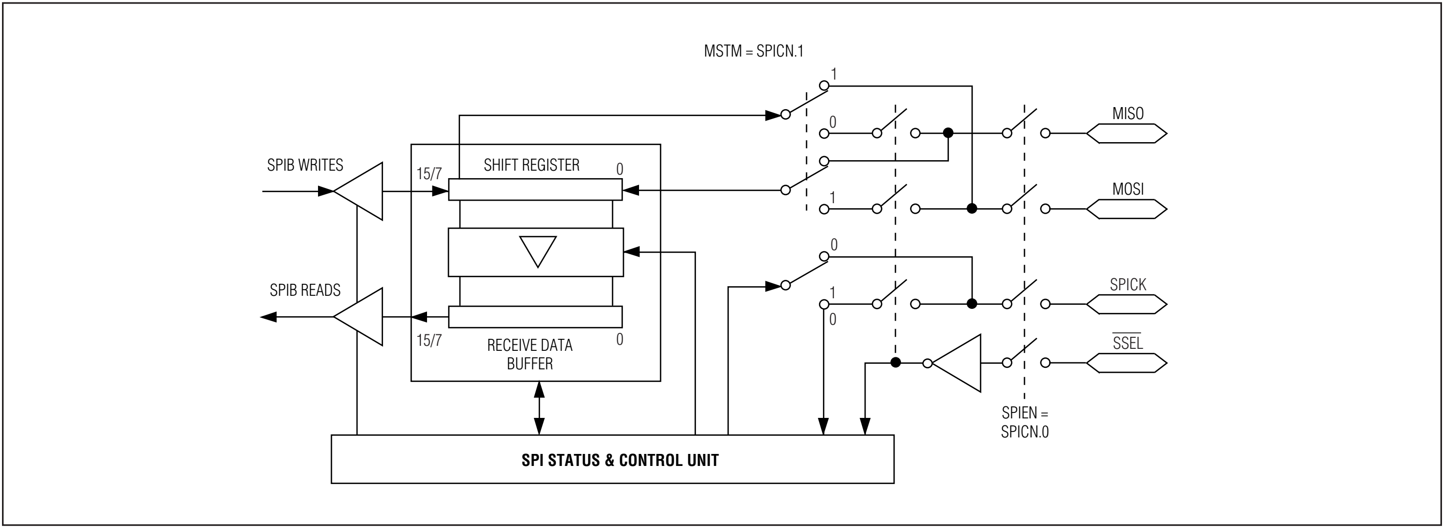


Figure 11-1. SPI Block Diagram

11.1 SPI Transfer Formats

During an SPI transfer, data is simultaneously transmitted and received over two serial data lines with respect to a single serial shift clock. The polarity and phase of the serial shift clock are the primary components in defining the SPI data transfer format. The polarity of the serial clock corresponds to the idle logic state of the clock line and therefore also defines which clock edge is the active edge. To define a serial shift clock signal that idles in a logic low state (active clock edge = rising), the Clock Polarity Select (CKPOL; SPICF.0) bit should be configured to a 0, while setting CKPOL = 1 will cause the shift clock to idle in a logic high state (active clock edge = falling). The phase of the serial clock selects which edge is used to sample the serial shift data. The Clock Phase Select (CKPHA; SPICF.1) bit controls whether the active or inactive clock edge is used to latch the data. When CKPHA is set to logic 1, data is sampled on the inactive clock edge (clock returning to the idle state). When CKPHA is set to logic 0, data is sampled on the active clock edge (clock transition to the active state). Together, the CKPOL and CKPHA bits allow the four possible SPI data transfer formats as illustrated in Figure 11-2.

Anytime that the active clock edge is used for sampling (CKPHA = 0), the transfer cycle must be started with assertion of the $\overline{\text{SSEL}}$ signal. This requirement necessitates that the $\overline{\text{SSEL}}$ signal be deasserted and reasserted between successive transfers. Conversely, when the inactive edge is used for sampling (CKPHA = 1), the $\overline{\text{SSEL}}$ signal may remain low through successive transfers allowing the active clock edge to signal the start of a new transfer.

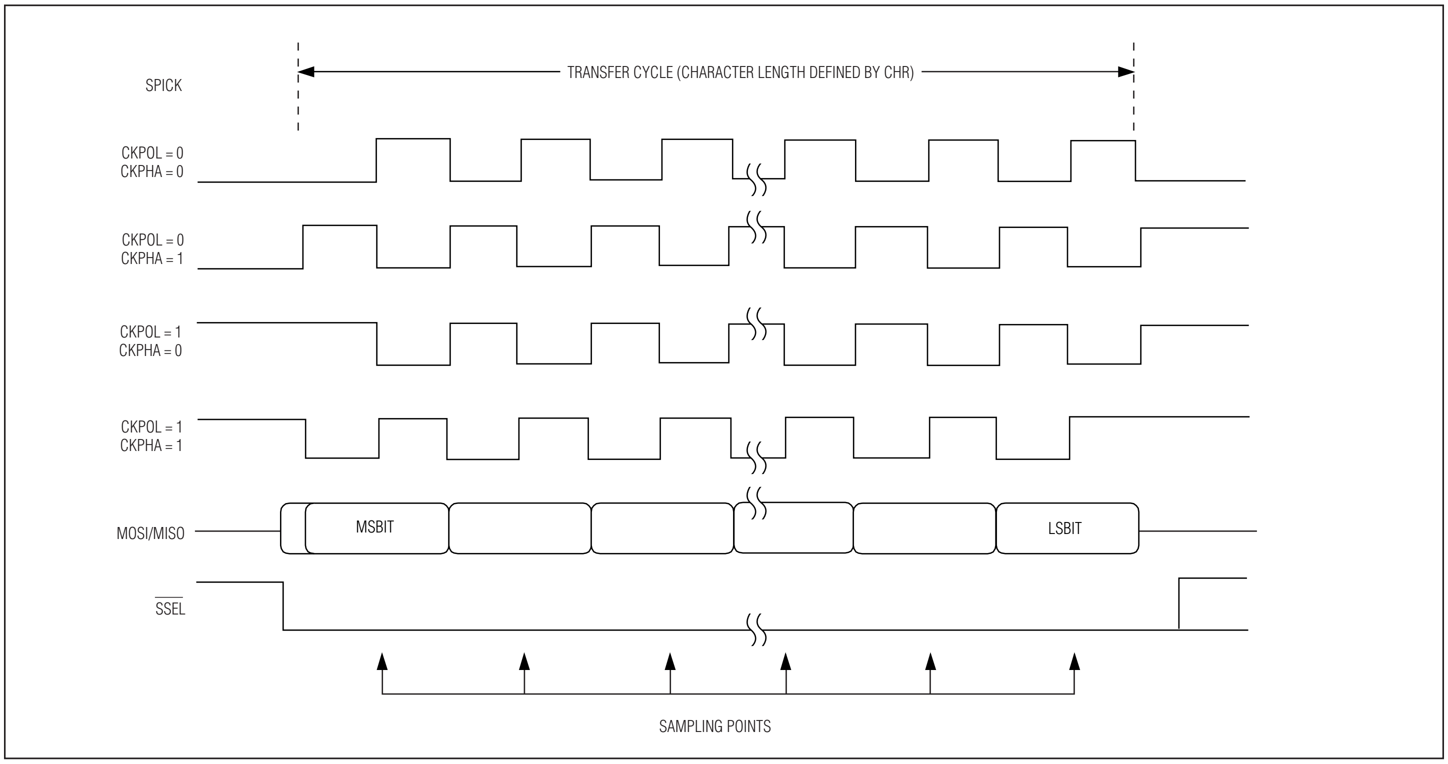


Figure 11-2. SPI Transfer Formats (CKPOL, CKPHA Control)

11.2 SPI Character Lengths

To flexibly accommodate different SPI transfer data lengths, the character length for any transfer is user configurable through the Character Length Bit (CHR) in the SPI Configuration Register. The CHR bit allows selection of either 8-bit or 16-bit transfers.

When loading 8-bit characters into the SPIB data buffer, the byte for transmission should be right-justified or placed in the least significant byte of the word. When a byte transfer completes, the received byte is right-justified and can be read from the least significant byte of the SPIB word. The MSB of the SPIB data buffer is not significant when transmitting and receiving 8-bit characters.

11.3 SPI Transfer Baud Rates

When operating as a slave device, an external master drives the SPI serial clock. For proper slave operation, the serial clock provided by the external master should not exceed the system clock frequency divided by 8.

When operating in the master mode, the SPI serial clock is sourced to the external slave device(s). The serial clock baud rate is determined by the clock-divide ratio specified in the SPI Clock Divider Ratio (SPICK) register. The SPI module supports 256 different clock-divide ratio selections for serial clock generation. The SPICK clock rate is determined by the following formula:

SPI Baud Rate = System Clock Frequency / (2 x Clock Divider Ratio)

where Clock Divider Ratio = (SPICK.7:0) + 1

Since the SPI baud rate is a function of the System Clock Frequency, using any of the system clock divide modes (including Power Management Mode) alters the baud rate. Attempts to invoke the Power Management Mode while an SPI transfer in is progress (STBY = 1) are ignored.

Note, however, that once in Power Management Mode (PMME = 1), writes to SPIB in master mode and assertion of the $\overline{\text{SSEL}}$ pin in slave mode both qualify as switchback sources if enabled (SWB = 1). The SPI module clocks are halted if the device is placed into Stop mode.

11.4 SPI System Errors

The SPI module can detect three types of SPI system errors. A mode fault error arises in a multiple master system when more than one SPI device simultaneously tries to be a master. A receive overrun error occurs when an SPI transfer completes before the previous character has been read from the receive-holding buffer. The third kind of error, write collision, indicates that an attempted write to SPIB was detected while a transfer was in progress (STBY = 1).

11.4.1 Mode Fault

When a SPI device is configured as a master and its Mode Fault Enable bit (SPICN.2: MODFE) is also set, a mode fault error occurs if $\overline{\text{SSEL}}$ input signal is driven low by an external device. This error is typically caused when a second SPI device attempts to function as a master in the system. In the condition where more than one device is configured as master concurrently, there is possibility of bus contention that can cause permanent damage to push-pull CMOS drivers. The mode fault error detection is to provide protection from such damage by disabling the bus drivers. When a mode fault is detected, the following actions are taken immediately:

1) The MSTM bit is forced to logic 0 to reconfigure the SPI device as a slave.

2) The SPIEN bit is forced to logic 0 to disable the SPI module.

3) The Mode Fault (SPICN.3: MODF) status flag is set. Setting the MODF bit can generate an interrupt if it is enabled.

The application software must correct the system conflict before resuming its normal operation. The MODF flag is set automatically by hardware but must be cleared by software or a reset once set. Setting the MODF bit to logic 1 by software causes an interrupt if enabled.

Mode fault detection is optional and can be disabled by clearing the MODFE bit to logic 0. Disabling the mode fault detection disables the function of the $\overline{\text{SSEL}}$ signal during master mode operation, allowing the associated port pin to be used as a general-purpose I/O.

Note that the mode fault mechanism does not provide full protection from bus contention in multiple master, multiple slave systems. For example, if two devices are configured as master at the same time, the mode fault-detect circuitry offers protection only when one of them selects the other as slave by asserting its $\overline{\text{SSEL}}$ signal. Also, if a master accidentally activates more than one slave and those devices try to simultaneously drive their output pins, bus contention can occur without and a mode fault error being generated.

11.4.2 Receive Overrun

Since the receive direction of SPI is double buffered, there is no overrun condition as long as the received character in the read buffer is read before the next character in the shift register ready to be transferred to the read buffer. However, if previous data in the read buffer has not been read out when a transfer cycle is completed and the new character is loaded into the read buffer, a receive overrun occurs and the Receive Overrun flag (SPICN.5: ROVR) is set. Setting the ROVR flag indicates that the oldest received character has been overwritten and is lost. Setting the ROVR bit to logic 1 causes an interrupt if enabled. Once set, the ROVR bit is cleared only by software or a reset.

11.4.3 Write Collision While Busy

A write collision occurs if an attempt to write the SPIB data buffer is made during a transfer cycle (STBY = 1). Since the shift register is single buffered in the transmit direction, writes to SPIB are made directly into the shift register. Allowing the write to SPIB while another transfer is in progress could easily corrupt the transmit/receive data. When such a write attempt is made, the current transfer continues undisturbed, the attempted write data is not transferred to the shift register, and the control unit sets the Write Collision flag (SPICN.4: WCOL). Setting the WCOL bit to logic 1 causes an interrupt if SPI interrupt sources are enabled. Once set, the WCOL bit is cleared only by software or a reset.

Normally, write collisions are associated solely with slave devices since they do not control initiation of transfers and do not have access to as much information about the SPICK clock as the master. As a master, write collisions are completely avoidable, however, the control unit detects write collisions for both master and slave modes.

11.5 SPI Master Operation

The SPI module is placed in master mode by setting the Master Mode Enable (MSTM) bit in the SPI Control register to logic 1. Only an SPI master device can initiate a data transfer. The master is responsible for manually selecting/deselecting the desired slave devices. This can be done using a general-purpose output pin. Writing a data character to the SPI shift register (SPIB) while in master mode starts a data transfer. The SPI master immediately shifts out the data serially on the MOSI pin, most significant bit first, while providing the serial clock on SPICK output. New data is simultaneously received on the MISO pin into the least significant bit of the shift register. The data transfer format (clock polarity and phase), character length, and baud rate are configurable as described earlier in the section. During the transfer, the SPI Transfer Busy (SPICN.7:STBY) flag is set to indicate that a transfer is in process. At the end of the transfer, the data contained in the shift register is moved into the receive data buffer, the STBY bit is cleared by hardware, and the SPI Transfer Complete flag (SPICN.6: SPIC) is set. Setting the SPIC bit generates an interrupt request if SPI interrupt sources are enabled (ESPII = 1).

11.6 SPI Slave Operation

The SPI module operates in slave mode when the MSTM bit is cleared to logic 0. In slave mode, the SPI is dependent on the SPICK sourced from the master to control the data transfer. The SPICK input frequency should be no greater than the system clock of the slave device frequency divided by 8.

The Slave Select $\overline{\text{SSEL}}$ input must be externally asserted by a master before data exchange can take place. $\overline{\text{SSEL}}$ must be low before data transaction begins and must remain low for the duration of the transaction. If data is to be transmitted by the slave device, it must be written to its shift register before the beginning of a transfer cycle, otherwise the character already in the shift register will be transferred. The slave device considers a transfer to begin with the first clock edge or the falling edge of the $\overline{\text{SSEL}}$, dependent on the data transfer format.

The SPI slave receives data from the external master MOSI pin, most significant bit first, while simultaneously transferring the contents of its shift register to the master on the MISO pin, also most significant bit first. Data received from the external master replaces data in the internal shift register until the transfer completes. Just like in the master mode of operation, received data is loaded into the read buffer and the SPI Transfer Complete flag is set at the end of transfer. The setting of the Transfer Complete flag generates an interrupt request if enabled.

When $\overline{\text{SSEL}}$ is not asserted, the slave device ignores the SPICK clock and the shift register is disabled. Under this condition, the device is basically idle, no data is shifted out from the shift register, and no data is sampled from the MOSI pin. The MISO pin is placed in an input mode and is weakly pulled high to allow other devices on the bus to drive the bus. Deassertion of the $\overline{\text{SSEL}}$ signal by the master during a transfer (before a full character, as defined by CHR, is received) aborts the current transfer. When the transfer is aborted, no data is loaded into the read buffer, the SPIC flag is not set, and the slave logic and bit counter are reset.

In slave mode, the Clock Divider Ratio bits (CKR7:0) have no function since an external master supplies the serial clock. The transfer format (CKPOL, CKPHA settings) and the character length selection (CHR) for the slave device, however, should match the master for a proper communication.

11.7 SPI Peripheral Registers

11.7.1 SPI Control Register (SPICN)

Bit #	7	6	5	4	3	2	1	0
Name	STBY	SPIC	ROVR	WCOL	MODF	MODFE	MSTM	SPIEN
Reset	0	0	0	0	0	0	0	0
Access	r	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bit 7: SPI Transfer Busy Flag (STBY). This bit is used to indicate the current transmit/receive activity of the SPI module. STBY is set to 1 when an SPI transfer cycle starts and is cleared to 0 when the transfer cycle is completed. This bit is controlled by hardware and is read-only for user software.

0 = SPI module is idle—no transfer in process

1 = SPI transfer in process

Bit 6: SPI Transfer Complete Flag (SPIC). This bit signals the completion of an SPI transfer cycle. This bit must be cleared to 0 by software once set. Setting this bit to logic 1 causes an interrupt if enabled.

0 = No SPI transfers have completed since the bit was last cleared

1 = SPI transfer complete

Bit 5: Receive Overrun Flag (ROVR). This bit indicates when a receive overrun has occurred. A receive overrun results when a received character is ready to be transferred to the SPI receive data buffer before the previous character in the data buffer is read. The most recent receive data is lost. This bit must be cleared to 0 by software once set. Setting this bit to logic 1 causes an interrupt if enabled.

0 = No receive overrun has occurred

1 = Receive overrun occurred

Bit 4: Write Collision Flag (WCOL). This bit signifies that an attempt was made by software to write the SPI Buffer (SPIB) while a transfer was in progress (STBY = 1). Such attempts will always be blocked. This bit must be cleared to 0 by software once set. Setting this bit to logic 1 causes an interrupt if enabled.

0 = No write collision has been detected

1 = Write collision detected

Bit 3: Mode Fault Flag (MODF). This bit is the mode fault flag for SPI master mode operation. When mode fault detection is enabled (MODFE = 1) in master mode, detection of high-to-low transition on the $\overline{\text{SSEL}}$ pin signifies a mode fault causes MODF to be set to 1. This bit must be cleared to 0 by software once set. Setting this bit to logic 1 causes an interrupt if enabled. This flag has no meaning in slave mode.

0 = No mode fault has been detected

1 = Mode fault detected while operating as a master (MSTM = 1)

Bit 2: Mode Fault Enable (MODFE). When set to logic 1, the $\overline{\text{SSEL}}$ input pin is used for mode fault detection during SPI master mode operation. When cleared to 0, the $\overline{\text{SSEL}}$ input has no function and its pin can be used for general-purposes I/O. In slave mode, the $\overline{\text{SSEL}}$ pin always functions as a slave-select input signal to the SPI module, independent of the setting of the MODFE bit.

Bit 1: Master Mode Enable (MSTM). The MSTM bit functions as a master mode enable bit for the SPI module.

0 = SPI module operates in slave mode when enabled (SPIEN = 1)

1 = SPI module operates in master mode when enabled (SPIEN = 1)

Note that this bit can be set from 0 to 1 only when the $\overline{\text{SSEL}}$ signal is deasserted. This bit can be automatically cleared to 0 by hardware if a mode fault is detected.

Bit 0: SPI Enable (SPIEN)

0 = SPI module and its baud-rate generator are disabled

1 = SPI module and its baud-rate generator are enabled

11.7.2 SPI Configuration Register (SPICF)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	ESPII	—	—	—	—	CHR	CKPHA	CKPOL
Reset	0	0	0	0	0	0	0	0
Access	rw	r	r	r	r	rw	rw	rw

r = read, w = write

Bit 7: SPI Interrupt Enable (ESPII). This bit enables any of the SPI interrupt source flags (MODF, WCOL, ROVR, SPIC) to generate interrupt requests.

0 = SPI interrupt sources disabled

1 = SPI interrupt sources enabled

Bits 6 to 3: Reserved

Bit 2: Character Length Bit (CHR). This bit determines the character length for a SPI transfer cycle. A character can be 8 bits in length or 16 bits in length.

0 = 8-bit character length specified

1 = 16-bit character length specified

Bit 1: Clock Phase Select (CKPHA). This bit selects the clock phase and is used with the CKPOL bit to define the SPI data transfer format.

0 = Data sampled on the active clock edge

1 = Data sampled on the inactive clock edge

Bit 0: Clock Polarity Select (CKPOL). This bit selects the clock polarity and is used with the CKPHA bit to define the SPI data transfer format.

0 = Clock idles in the logic 0 state (rising = active clock edge)

1 = Clock idles in the logic 1 state (falling = active clock edge)

11.7.3 SPI Clock Register (SPICK)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	CKR.7	CKR.6	CKR.5	CKR.4	CKR.3	CKR.2	CKR.1	CKR.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: Clock Divider Ratio (CKR.[7:0]). This 8-bit value determines the system clock-divide ratio to be used for SPI master mode baud-clock generation. This register has no function when operating in slave mode as the SPI clock generation circuitry is disabled. The frequency of the SPI master mode baud rate is calculated using the following equation:

SPI Baud Rate = (0.5 x System Clock Frequency) / (CKR[7:0] + 1)

11.7.4 SPI Data Buffer Register (SPIB)

Bit #	15	14	13	12	11	10	9	8
Name	SPIB.15	SPIB.14	SPIB.13	SPIB.12	SPIB.11	SPIB.10	SPIB.9	SPIB.8
Reset	0	0	0	0	0	0	0	0
Access	rs	rs	rs	rs	rs	rs	rs	rs

Bit #	7	6	5	4	3	2	1	0
Name	SPIB.7	SPIB.6	SPIB.5	SPIB.4	SPIB.3	SPIB.2	SPIB.1	SPIB.0
Reset	0	0	0	0	0	0	0	0
Access	rs	rs	rs	rs	rs	rs	rs	rs

r = read, s = special

Bits 15 to 0: SPI Data Buffer (SPIB.[15:0]). Data for SPI is read from or written to this location. The serial transmit and receive buffers are separate but both are addressed at this location. Write access is allowed only outside of the transfer cycle. When the STBY bit is set, write attempts are blocked and cause a write collision error.

SECTION 12: HARDWARE MULTIPLIER MODULE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 12: HARDWARE MULTIPLIER MODULE

The hardware multiplier module can be used by the MAXQ microcontroller to support high-speed multiplications. The hardware multiplier module is equipped with two 16-bit operand registers, a 32-bit read-only result register, and an accumulator of width between 32 bits and 48 bits, depending on the specific MAXQ device. The multiplier can complete a 16-bit x 16-bit multiply-and-accumulate/subtract operation in a single cycle. The hardware multiplier module supports the following operations without interfering with the normal core functions:

- Signed or unsigned Multiply (16 bit x 16 bit)
- Signed or unsigned Multiply-Accumulate (16 bit x 16 bit)
- Signed or unsigned Multiply-Subtract (16 bit x 16 bit)
- Signed Multiply and Negate (16 bit x 16 bit)

12.1 Hardware Multiplier Organization

The hardware multiplier consists of two 16-bit, parallel-load operand registers (MA, MB); a read-only result register formed by two parallel 16-bit registers (MC1R and MC0R); an accumulator, which is formed by up to three 16-bit parallel registers (MC2, MC1, and MC0); and a status/control register (MCNT). Note that the width and/or presence of the MC2 register depend on the specified accumulator size for the given MAXQ device. Figure 12-1 shows a block diagram of the hardware multiplier.

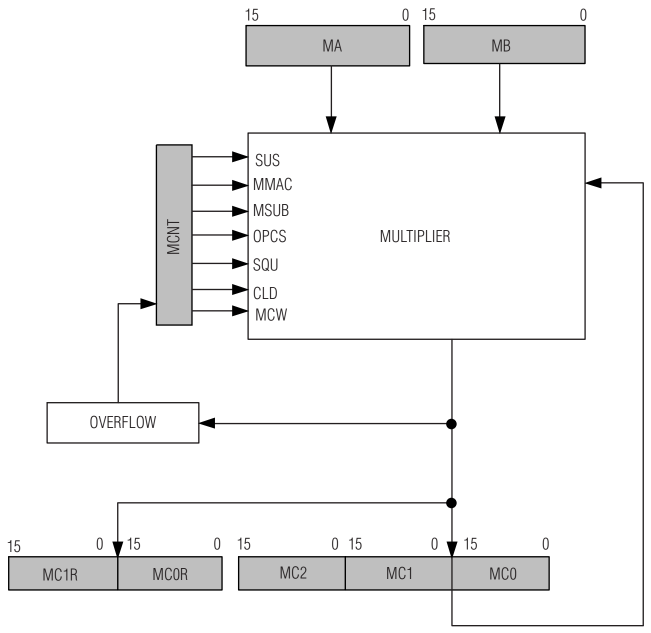


Figure 12-1. Multiplier Organization

12.2 Hardware Multiplier Controls

The selection of operation to be performed by the multiplier is determined by four control bits in the MCNT register: SUS, MSUB, MMAC, and SQU. The number of operands that must be loaded to trigger the specified operation is dictated by the OPCS bit setting, except when the square function is enabled (SQU = 1). Enabling the square function implicitly defines that only a single operand (either MA or MB) needs to be loaded to trigger the square operation, independent of the OPCS bit setting. The MCNT register bits must be configured to select the desired operation and operand count prior to loading the operand(s) to trigger the multiplier operation. Any write to MCNT automatically resets the operand load counter of the multiplier, but does not affect the operand registers, unless such action is requested using the Clear Data Registers (CLD) control bit. Once the desired operation has been specified via the MCNT register bits, loading the prescribed number of operands triggers the respective multiply, multiply-accumulate/subtract or multiply-negate operation.

12.3 Register Output Selection

The Hardware Multiplier implements the MC Register Write Select (MCW) control bit so that writing of the result to the MC2:MC0 registers can be blocked to preserve the MC registers (accumulator). When the MCW bit is configured to logic 1, the result for the given operation is not written to the MC registers. When the MCW bit is configured to logic 0, the MC registers are updated with the result of the operation. The MC1R, MC0R read-only register pair are updated independent of the MCW bit setting. This register pair always reflect the output that would normally be placed in MC1:MC0, given that MCW = 1 or MMAC = 0. When MCW = 0 and MMAC = 1, the MC1R:MC0R content may not match the MC1:MC0 register content, but it will be predictable and may be useful in certain situations. See Table 12-1 for details.

12.3.1 Signed-Unsigned Operand Selection

The operands can be either signed or unsigned numbers, but the data type must be defined by the user software via the Signed-Unsigned (SUS) bit prior to triggering the operation. For an unsigned operation, the Signed-Unsigned bit (SUS) in the MCNT register must be set to 1; for a signed operation, the SUS bit must be cleared to 0. The multiplier treats unsigned numbers as absolute magnitude. For a 16-bit positional binary number, this represents a value in the range 0 to 2^{16} - 1 (xFFFFh). The signed number representation is a two's-complement value, where the most significant bit is defined as a sign bit. The range of a 16-bit two's-complement number is $-2^{(16-1)}$ (x8000h) to $+2^{(16-1)}$ - 1 (x7FFF). The product of any signed operation will be sign extended before being stored or accumulated/subtracted into the MC registers. The SUS bit should always be configured to logic 0 (i.e., signed operands) for the multiply-negate operation. Attempting an unsigned multiply-negate operation results in incorrect results and setting of the OF bit. Modifying the operand data type selection via the SUS bit does not alter the contents of the MC registers. The MC registers are read/write accessible and can be modified by user code when necessary.

12.3.2 Operand Count Selection

The OPCS bit allows selection of single operand or two operands operation for the multiply and multiply-accumulate/subtract operations. When the OPCS bit is cleared to 0, the multiply or multiply-accumulate/subtract operation established by the SUS, MSUB, and MMAC bits is triggered once two operands are loaded, one to each of the MA and MB registers. When OPCS is set to 1, the operation commences once data is loaded to either MA or MB. The OPCS bit is ignored when the square operation is enabled (SQU), since loading of data to the MA or MB register actually writes to both registers.

12.4 Hardware Multiplier Operations

The control bits, which specify data type (SUS), operand count (OPCS or SQU), and destination control (MCW), have already been described. However, there are two additional MCNT register bits that serve to define the Hardware Multiplier operation. The multiply-accumulate/subtract and multiply-negate operations are enabled by the Multiply-Accumulate Enable (MMAC) and Multiply Negate (MSUB) bits in the MCNT register. When the MMAC bit is set to 1, the multiplier performs a multiply-accumulate (if MSUB = 0) or a multiply-subtract (if MSUB = 1). If MMAC is configured to 0, the multiplier result is not accumulated or subtracted, but can be stored directly (if MSUB = 0) or negated (if MSUB = 1) before storage. The multiply-negate operation (MMAC = 0, MSUB = 1) is only allowable for signed data operands (SUS = 0). For unsigned multiply-accumulate/subtract operations, the OF bit is set when a carry-out/borrow-in from the most significant bit of the MC register occurs. For a signed two's-complement multiply-accumulate/subtract operations, the OF bit is set when the carry-out/borrow-in from the most significant magnitude position of the MC register is different from the carry-out/borrow-in of the sign position of the MC register. Since there is no overflow condition for multiply and multiply-negate operations, the OF bit is always cleared for these operations with one exception. The OF bit will be set to logic 1 if an unsigned multiply-negate (invalid operation) is requested. Table 12-1 shows the operations supported by the multiplier and associated MCNT control bit settings.

12.4.1 Accessing the Multiplier

There are no restrictions on how quickly data is entered into the operand registers or the order of data entry. The only requirement to do a calculation is to perform the loading of MA and/or MB registers having specified data type and operation in the MCNT register. The multiplier keeps track of the writes to the MA and MB registers, and starts calculation immediately after the prescribed number of operands is loaded. If two operands are specified for the operation, the multiplier waits for the second operand to be loaded into the other operand register before starting the actual calculation. If for any reason software needs to reload the first operand, it should either reload that same operand register or use the CLD bit in the MCNT register to reinitialize the multiplier; otherwise, loading data to another operand register triggers the calculation. The CLD bit is a self-clearing bit that can be used for multiplier initialization. When it is set, it clears all data registers and the OF bit to zero and resets the multiplier operand write counter.

The specified hardware multiplier operation begins when the final operand(s) is loaded and will complete in a single cycle. The read-only MC1R, MC0R result registers can be accessed in the very next cycle unless accumulation/subtraction with MC2:0 is requested (MCW = 0 and MMAC = 1), in which case, one cycle is required so that stable data can be read. When MCW = 0, the MC2:0 registers always require one wait cycle before the operation result is accessible. The single wait cycle needed for updating the MC2:0 registers with a calculated result does not prevent initiating another calculation. Back-to-back operations can be triggered (independent of data type and operand count) without the need of wait state between loading of operands.

Table 12-1. Hardware Multiplier Operations

MCW:MSUB:MMAC	OPERATION	MC2	MC1	MC0	MC1R:MC0R	OF STATUS
000	Multiply	MA*MB			MA*MB	No
001	Multiply-Accumulate	MC+(MA*MB)			32lsbits of (MC+2*(MA*MB))	Yes
010	Multiply-Negate (SUS = 0 only)	-(MA*MB)			-(MA*MB)	No
011	Multiply-Subtract	MC-(MA*MB)			32lsbits of (MC-2*(MA*MB))	Yes
100	Multiply	MC2	MC1	MC0	MA*MB	No
101	Multiply-Accumulate	MC2	MC1	MC0	32lsbits of (MC+(MA*MB))	No
110	Multiply-Negate (SUS = 0 only)	MC2	MC1	MC0	-(MA*MB)	No
111	Multiply-Subtract	MC2	MC1	MC0	32lsbits of (MC-(MA*MB))	No

12.5 Hardware Multiplier Peripheral Registers

12.5.1 Hardware Multiplier Control Register (MCNT)

Bit #	7	6	5	4	3	2	1	0
Name	OF	MCW	CLD	SQU	OPCS	MSUB	MMAC	SUS
Reset	0	0	0	0	0	0	0	0
Access	r	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bit 7: Overflow Flag (OF). This bit is set to logic 1 when an overflow occurred for the last operation. This bit can be set for accumulation/subtraction operations or unsigned multiply-negate attempts. This bit is automatically cleared to 0 following a reset, starting a multiplier operation, or setting of the CLD bit to 0.

Bit 6: MC Register Write Select (MCW). The state of the MCW bit determines if an operation result will be placed into the accumulator registers (MC).

0 = The result will be written to the MC registers.

1 = The result is not written to the MC registers (MC register content is unchanged).

Bit 5: Clear Data Registers (CLD). This bit initializes the operand registers and the accumulator of the multiplier. When it is set to 1, the contents of all data registers and the OF bit are cleared to 0 and the operand load counter is reset immediately. This bit is cleared by hardware automatically. Writing a 0 to this bit has no effect.

Bit 4: Square Function Enable (SQU). This bit supports the hardware square function. When this bit is set to logic 1, a square operation is initiated after an operand is written to either the MA or the MB register. Writing data to either of the operand registers writes to both registers and triggers the specified square or square-accumulate/subtract operation. Setting this bit to 1 also overrides the OPCS bit setting. When SQU is cleared to logic 0, the hardware square function is disabled.

0 = Square function disabled

1 = Square function enabled

Bit 3: Operand Count Select (OPCS). This bit defines how many operands must be loaded to trigger a multiply or multiply-accumulate/subtract operation (except when SQU = 1 since this implicitly specifies a single operand). When this bit is cleared to logic 0, both operands (MA and MB) must be written to trigger the operation. When this bit is set to 1, the specified operation is triggered once either operand is written.

0 = Both operands (MA and MB) must be written to trigger the multiplier operation.

1 = Loading one operand (MA or MB) triggers the multiplier operation.

Bit 2: Multiply Negate (MSUB). Configuring this bit to logic 1 enables negation of the product for signed multiply operations and subtraction of the product from the accumulator (MC[2:0]) when MMAC = 1. When MSUB is configured to logic 0, the product of multiply operations will not be negated and accumulation is selected when MMAC = 1.

Bit 1: Multiply-Accumulate Enable (MMAC). This bit enables the accumulate or subtract operation (as per MSUB) for the hardware multiplier. When this bit is cleared to logic 0, the multiplier will perform only multiply operations. When this bit is set to logic 1, the multiplier will perform a multiply-accumulate or multiply-subtract operation based upon the MSUB bit.

0 = Accumulate/subtract operation disabled

1 = Accumulate/subtract operation enabled

Bit 0: Signed-Unsigned Select (SUS). This bit determines the data type of the operands. When this bit is cleared to logic 0, the operands are treated as two's-complement values and the multiplier performs a signed operation. When this bit is set to logic 1, the operands are treated as absolute magnitudes and the multiplier performs an unsigned operation.

0 = Signed operands

1 = Unsigned operands

12.5.2 Multiplier Operand A Register (MA)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	MA.15	MA.14	MA.13	MA.12	MA.11	MA.10	MA.9	MA.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	MA.7	MA.6	MA.5	MA.4	MA.3	MA.2	MA.1	MA.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Multiplier Operand A Register (MA.[15:0]). This operand A register is used by the application code to load 16-bit values for multiplier operations.

12.5.3 Multiplier Operand B Register (MB)

Bit #	15	14	13	12	11	10	9	8
Name	MB.15	MB.14	MB.13	MB.12	MB.11	MB.10	MB.9	MB.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	MB.7	MB.6	MB.5	MB.4	MB.3	MB.2	MB.1	MB.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Multiplier Operand B Register (MB.[15:0]). This operand B register is used by the application code to load 16-bit values for multiplier operations.

12.5.4 Multiplier Accumulator 2 Register (MC2)

Bit #	15	14	13	12	11	10	9	8
Name	MC2.15	MC2.14	MC2.13	MC2.12	MC2.11	MC2.10	MC2.9	MC2.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	MC2.7	MC2.6	MC2.5	MC2.4	MC2.3	MC2.2	MC2.1	MC2.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Multiplier Accumulator 2 Register (MC2.[15:0]). The MC2 register represents the two most significant bytes of the accumulator register. The 48-bit accumulator is formed by MC2, MC1 and MC0. For a signed operation, the most significant bit of this register is the sign bit. The MC2 register width is dependent upon the hardware multiplier accumulator width of the given MAXQ device. For a MAXQ device having only a 32-bit accumulator, the MC2 register will not be present.

12.5.5 Multiplier Accumulator 1 Register (MC1)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	MC1.15	MC1.14	MC1.13	MC1.12	MC1.11	MC1.10	MC1.9	MC1.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	MC1.7	MC1.6	MC1.5	MC1.4	MC1.3	MC1.2	MC1.1	MC1.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Multiplier Accumulator 1 Register (MC1.[15:0]). The MC1 register represents bytes 3 and 2 of the accumulator register. The 48-bit accumulator is formed by MC2, MC1, and MC0.

12.5.6 Multiplier Accumulator 0 Register (MC0)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	MC0.15	MC0.14	MC0.13	MC0.12	MC0.11	MC0.10	MC0.9	MC0.8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	MC0.7	MC0.6	MC0.5	MC0.4	MC0.3	MC0.2	MC0.1	MC0.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 15 to 0: Multiplier Accumulator 0 Register (MC0.[15:0]). The MC0 register represents the two least significant bytes of the accumulator register. The 48-bit accumulator is formed by MC2, MC1, and MC0.

12.5.7 Multiplier Read Register 1 (MC1R)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	MC1R.15	MC1R.14	MC1R.13	MC1R.12	MC1R.11	MC1R.10	MC1R.9	MC1R.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	MC1R.7	MC1R.6	MC1R.5	MC1R.4	MC1R.3	MC1R.2	MC1R.1	MC1R.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read

Bits 15 to 0: Multiplier Read Register 1 (MC1R.[15:0]). The MC1R register represents bytes 3 and 2 result from the last operation when MCW = 1 or the last operation was a multiply or multiply-negate. When MCW = 0 and the last operation was a multiply-accumulate/subtract, the contents of this register may or may not agree with the contents of MC1 due to the combinatorial nature of the adder. The content of this register may change if MCNT, MA, MB, or MC[2:0] is changed.

12.5.8 Multiplier Read Register 0 (MC0R)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	MC0R.15	MC0R.14	MC0R.13	MC0R.12	MC0R.11	MC0R.10	MC0R.9	MC0R.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	MC0R.7	MC0R.6	MC0R.5	MC0R.4	MC0R.3	MC0R.2	MC0R.1	MC0R.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read

Bits 15 to 0: Multiplier Read Register 0 (MC0R.[15:0]). The MC1R register represents bytes 1 and 0 result from the last operation when MCW = 1 or the last operation was a multiply or multiply-negate. When MCW = 0 and the last operation was a multiply-accumulate/subtract, the contents of this register may or may not agree with the contents of MC0 due to the combinatorial nature of the adder. The content of this register may change if MCNT, MA, MB or MC[2:0] is changed.

12.6 Hardware Multiplier Examples

The following are code examples of multiplier operations.

```
;Unsigned Multiply 16-bit x 16-bit
   move  MCNT, #21h           ; CLD=1, SUS=1 (unsigned)
   move  MA, #0FFFh           ; MC2:0=0000_0000_0000h
   move  MB, #1001h           ; MC1R:MC0R= 00FF_FFFFh
                              ; MC2:0=0000_00FF_FFFFh

;Signed Multiply 16-bit x 16-bit
   move  MCNT, #20h           ; CLD=1, SUS=0 (signed)
   move  MA, #F001h           ; MC2:0=0000_0000_0000h
   move  MB, #1001h           ; MC1R:MC0R= FF00_0001h
                              ; MC2:0=FFFF_FF00_0001h

;Unsigned Multiply-Accumulate 16-bit x 16-bit
                              ; MC2:0=0000_0100_0001h
   move  MCNT, #03h           ; MMAC=1, SUS=1 (unsigned)
   move  MA, #0FFFh           ;
   move  MB, #1001h           ;
                              ; MC1R:MC0R=02FF_FFFFh
                              ; MC2:0=0000_0200_0000h

;Signed Multiply-Accumulate 16-bit x 16-bit
                              ; MC2:0=0000_0100_0001h
   move  MCNT, #02h           ; SUS=0 (signed)
   move  MA, #F001h           ;
   move  MB, #1001h           ;
                              ; MC1R:MC0R= FF00_0003h
                              ; MC2:0=0000_0000_0002h

;Unsigned Multiply-Subtract 16-bit x 16-bit
                              ; MC2:0=0000_0100_0001h
   move  MCNT, #07h           ; MMAC=1, MSUB=1, SUS=1 (unsigned)
   move  MA, #0FFFh           ;
   move  MB, #1001h           ;
                              ; MC1R:MC0R=FF00_0003h
                              ; MC2:0=0000_0000_0002h

;Signed Multiply-Subtract 16-bit x 16-bit
                              ; MC2:0=0000_0100_0001h
   move  MCNT, #06h           ; MMAC=1, MSUB=1, SUS=0 (signed)
   move  MA, #F001h           ;
   move  MB, #1001h           ;
                              ; MC1R:MC0R= 02FF_FFFFh
                              ; MC2:0=0000_0200_0000h

;Signed Multiply Negate 16-bit x 16-bit
   move  MCNT, #24h           ; CLD=1, MSUB=1, SUS=0 (signed)
   move  MA, #F001h           ; MC2:0=0000_0000_0000h
   move  MB, #1001h           ; MC1R:MC0R =00FF_FFFFh
                              ; MC2:0=0000_00FF_FFFFh
```

SECTION 13: 1-Wire BUS MASTER

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 13: 1-Wire BUS MASTER

The 1-Wire Bus Master can be used by the MAXQ microcontroller to support 1-Wire communication to external 1-Wire devices without tying up valuable CPU resources. The Bus Master provides complete control of the 1-Wire bus, and transmit and receive activities. All timing and control sequences of the 1-Wire bus are generated within the Bus Master. Communication between the CPU and the Bus Master is through read/write access of 1-Wire Master Address (OWA) and 1-Wire Master Data (OWD) peripheral registers. When bus activity has generated a condition that requires CPU service, the Bus Master sets a status bit, allowing an interrupt to be generated if enabled. The 1-Wire Bus Master is operable for any system clock frequency between 4MHz and 25MHz, and supports the Bit Banging and Search ROM Accelerator modes. Detailed operation of the 1-Wire bus is described in the *Book of iButton Standards*, available on the Maxim/Dallas Semiconductor website at www.maxim-ic.com/iButtonbook. Figure 13-1 shows a functional block diagram of the 1-Wire Bus Master.

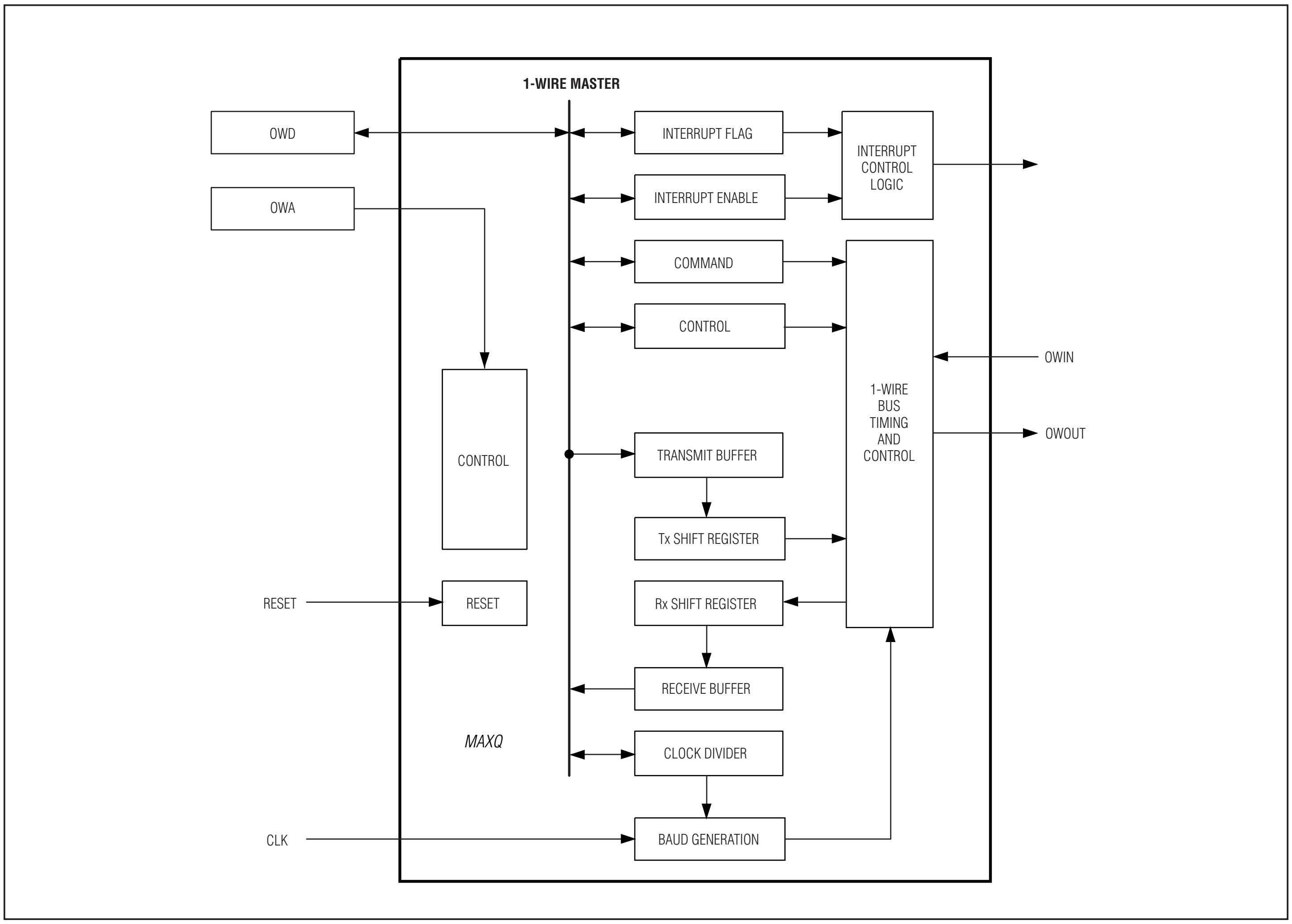


Figure 13-1. 1-Wire Bus Master Functional Diagram

13.1 1-Wire Peripheral Registers

The MAXQ microcontroller interfaces to the 1-Wire Bus Master through two peripheral registers: 1-Wire Master Address (OWA) and 1-Wire Master Data (OWD). These two registers allow read/write access of the six internal registers of the 1-Wire Bus Master. The internal registers provide a means for the CPU to configure and control transmit/receive activity through the Bus Master.

The three least significant bits (A[2:0]) of the OWA peripheral register specify the address of the internal register to be accessed. The OWD SFR is used for read/write access to the implemented bits of the specified internal register. To access an internal 1-Wire register, a valid address must be specified in the OWA peripheral register prior to performing a read/write operation to the OWD peripheral register. As long as a valid address is presented in OWA, read accesses of OWD will return data content from the internal target register and writes to OWD will update the internal target register with the data provided via OWD (with exception of the interrupt flag Register, which is read only). The following details the OWA and OWD registers.

13.1.1 1-Wire Address Register (OWA)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	—	—	—	—	—	A2	A1	A0
Reset	0	0	0	0	0	1	1	1
Access	r	r	r	r	r	rw	rw	rw

r = read, w = write

Bits 7 to 3: Reserved

Bits 2 to 0: 1-Wire Internal Register Address Bits (A[2:0]). These bits are used to select one of the 1-Wire Master internal registers to be accessed via the OWD register. Prior to accessing any of the 1-Wire Master internal registers, the address for the target internal register must be specified. Addresses where A[2:0] = 11xb are considered reserved addresses and are not supported by the Bus Master. Read access to these addresses will return invalid data in OWD and write accesses will not change the content of any writable registers.

A2	A1	A0	INTERNAL REGISTER (READ/WRITE ACCESSIBILITY)
0	0	0	Command (Read/Write)
0	0	1	Transmit/Receive Buffer (Read/Write)
0	1	0	Interrupt Flag (Read)
0	1	1	Interrupt Enable (Read/Write)
1	0	0	Clock Divisor (Read/Write)
1	0	1	Control (Read/Write)
1	1	X	Reserved

13.1.2 1-Wire Data Register (OWD)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	OWD.7	OWD.6	OWD.5	OWD.4	OWD.3	OWD.2	OWD.1	OWD.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: 1-Wire Data Register (OWD.[7:0]). This register contains the data value read from the target internal register as selected by the A[2:0] bits in the OWA register when read. A write to the OWD causes the data to be written to the target internal register selected by the A[2:0] bits of the OWA register (with exception of the interrupt flag register, which is read-only).

13.2 1-Wire Clock Control

All 1-Wire timing patterns are generated using a base clock of 1.0MHz. To create this base clock frequency, the 1-Wire Bus Master must internally divide down the microcontroller system clock. The Clock Divisor internal register implements bits to control this clock division. The prescaler bits (PRE[1:0]) divide the microcontroller system clock by 1, 3, 5, or 7 for settings of 00b, 01b, 10b, and 11b, respectively. The divider bits (DIV[2:0]) control circuitry to then divide the prescaler output clock by 1, 2, 4, 8, 16, 32, 64, or 128. The CLK_EN bit (bit 7 of the Clock Divisor register) enables or disables the clock generation circuitry. Setting CLK_EN to logic 1 enables the clock generation circuitry, while clearing the bit disables the clock generation circuitry. When cleared to 0, this bit essentially puts the Bus Master into a power-saving mode that disables the clock divisor circuitry while not in use. Note that without the clock, the Bus Master functionality is basically disabled. The following documents the internal clock divisor register.

13.2.1 1-Wire Clock Divisor Register (OWA = 100b)

Bit #	7	6	5	4	3	2	1	0
Name	CLK_EN	—	—	DIV2	DIV1	DIV0	PRE1	PRE0
Reset	0	0	0	0	0	0	0	0
Access	rw	r	r	rw	rw	rw	rw	rw

r = read, w = write

Bit 7: Clock Enable (CLK_EN). The CLK_EN bit of the Clock Divisor register is used to control the clock generation circuitry in the Bus Master. Setting the CLK_EN bit to logic 1 enables the clock generation circuitry according to the PRE[1:0] and DIV[2:0] settings.

Bits 6 and 5: Reserved

Bits 4, 3, 2: Divider Bits 2:0 (DIV[2:0]). These bits allow further division of the prescaled clock for generating a 1-Wire base clock:

$$\text{Base Clock} = \text{Prescaled Clock} / (2^{\text{DIV2:0}})$$

Bits 1 and 0: Clock Prescaler Bits 1:0 (PRE[1:0]). These prescaler bits define the initial clock division applied to the reference clock input. The prescaled clock output will be according to the following equation:

$$\text{Prescaled Clock} = \text{Reference Clock} / (2 \times \text{PRE[1:0]} + 1)$$

The Clock Divisor register must be configured properly before any 1-Wire communication can take place. The Bus Master clock divisor settings currently allow reference clock input frequencies between 4MHz and 25MHz with ~50% duty cycle to be supported. Table 13-1 summarizes the proper division values, based upon the reference input clock range. Settings not listed in the table are reserved and can result in improper operation if used. Note that providing a system clock frequency nearer the minimum of a given reference clock frequency range yields base-clock frequencies closer to 1MHz and better timing margin.

Table 13-1. Clock Divisor Register Setting for Reference Clock Rates

REFERENCE CLOCK FREQUENCY (MHz)		DIVIDER RATIO	DIV2	DIV1	DIV0	PRE1	PRE0
MIN	MAX						
4.0	< 5.0	4	0	1	0	0	0
5.0	< 6.0	5	0	0	0	1	0
6.0	< 7.0	6	0	0	1	0	1
7.0	< 8.0	7	0	0	0	1	1
8.0	< 10.0	8	0	1	1	0	0
10.0	< 12.0	10	0	0	1	1	0
12.0	< 14.0	12	0	1	0	0	1
14.0	< 16.0	14	0	0	1	1	1
16.0	< 20.0	16	1	0	0	0	0
20.0	< 24.0	20	0	1	0	1	0
24.0	≤ 25.0	24	0	1	1	0	1

13.3 1-Wire Bus Master Control

The 1-Wire Bus Master can perform certain special functions to support OW line operation. These special functions can be set up through the Control register that is documented below. The Control Register defaults to 00h on a reset, which disables all special functions.

13.3.1 1-Wire Control Register (OWA = 101b)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	EOWMI	—	BIT_CTL	—	—	EN_FOW	PPM	LLM
Reset	0	0	0	0	0	0	0	0
Access	rw	r	rw	r	r	rw	rw	rw

r = read, w = write

Bit 7: Enable 1-Wire Master Interrupts (EOWMI). Setting this bit to logic 1 enables the 1-Wire interrupt request to the CPU if any of the interrupt flags in the Interrupt Flag register is set and its corresponding enable bit in the Interrupt Enable register is also set. Clearing this bit to logic 0 disables 1-Wire interrupt request to the CPU.

Bits 6, 4, and 3: Reserved

Bit 5: Bit-Banging Mode Enable (BIT_CTL). Setting this bit to logic 1 places the master into bit-banging mode of operation, where only the least significant bit of the transmit/receive register would be sent/received before enabling the interrupt that signals the end of the transmission. Clearing this bit to logic 0 leaves the master operating in full byte boundaries. The Search ROM Accelerator Mode (SRA = 1) overrides the bit-banging mode.

Bit 2: Enable Force OW (EN_FOW). To enable the force OW line command, the EN_FOW bit must be set to logic 1. Clearing this bit to logic 0 disables the force OW line command in the command register.

Bit 1: Presence Pulse Masking (PPM). This bit is used to enable presence pulse masking function. Setting this bit to logic 1 causes the master to initiate the beginning of a presence pulse during a 1-Wire reset. This enables the master to prevent the larger amount of ringing caused by the slave devices when initiating a low on the OW line. If the PPM bit is set, the PDR result bit in the Interrupt Flag Register is always set, which shows that a slave device was on the line even if there were none. Clearing this bit to logic 0 disables the presence pulse-masking function.

Bit 0: Long Line Mode (LLM). This bit is used to enable the long line mode timing. Setting this bit to logic 1 effectively moves the write one release and the data sample timing during standard mode communication out to 8µs and 22µs, respectively. The recovery time will also be extended to 14ms. This provides a less strict environment for long line transmissions. Clearing this bit to logic 0 leaves the write one release, data sampling, and recovery time during standard mode communication at 5ms, 15ms, and 10ms, respectively.

13.4 1-Wire Bus Master Commands

The 1-Wire Bus Master can generate special commands on the 1-Wire bus in addition to transmitting and receiving data. The commands are generated via the setting of a corresponding bit in the Command Register (A[2:0] = 000b), which is documented below. These operational modes are defined in the Book of iButton Standards.

13.4.1 1-Wire Command Register (OWA = 000b)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	—	—	—	—	OW_IN	FOW	SRA	1WR
Reset	0	0	0	0	s	0	0	0
Access	r	r	r	r	r	rw	rw	rw

r = read, w = write, s = special

Bits 7 to 4: Reserved

Bit 3: 1-Wire Input (OW_IN). This bit always reflects the current logic state of the OW_IN line.

Bit 2: Force 1-Wire (FOW). Setting this bit to logic 1 forces OW line to a low value if the EN_FOW bit in the 1-Wire internal control register is also set to logic 1. The FOW bit has no effect on the OW line when the EN_FOW bit is cleared to logic 0.

Bit 1: Search ROM Accelerator (SRA). Setting this bit to logic 1 places the Bus Master into Search ROM Accelerator mode to expedite the Search ROM process and prevent the CPU from having to perform single-bit manipulations of the bus during a Search ROM operation. Note that the receive buffer must be empty before invoking SRA mode. This mode of operation is used to get either the addresses of all devices connected to the 1-Wire bus or the serial number of one device and simultaneously address the device. Clearing this bit to logic 0 disables the Search ROM accelerator.

Bit 0: 1-Wire Reset (1WR). Setting this bit to logic 1 causes a reset on the 1-Wire bus, which must precede any command given on the bus. Setting this bit also automatically clears the SRA bit. The 1WR bit is automatically cleared as soon as the 1-Wire bus reset completes. The Bus Master sets the presence-detect interrupt flag (PD) when the reset is completed and sufficient time for a 1-Wire reset to occur has passed. The result of the 1-Wire reset is placed in the Interrupt Register bit PDR. If a presence-detect pulse was received, PDR is cleared; otherwise, it is set. This bit is cleared to logic 0 when no reset action is required.

13.5 Search Operation Using Search ROM Accelerator

The 1-Wire Bus Master supports a Search ROM Accelerator Mode to expedite learning of ROM IDs for those devices connected to the bus. The bus master must determine the ROM IDs of the slave devices on the 1-Wire bus before it can address each slave device individually.

The Search ROM command (F0h) is used by the Bus Master to signal external 1-Wire devices that a ROM ID search will be conducted. The Search ROM command can be issued immediately following a Reset sequence initiated by the master. Once the search ROM command has been issued by the bus master, slave devices simultaneously transmit, bit-by-bit, their unique ROM IDs. There are three 1-Wire bus time slots associated with each ROM ID bit acquisition. These three time slots are as follows:

1) Read Time Slot 1: each slave transmits a single bit of its ROM ID (lsb first).

2) Read Time Slot 2: each slave transmits a complementary bit to that transmitted in 1.

3) Write Time Slot: bus master transmits discrepancy decision bit if needed.

The ROM ID acquisition and selection process listed starts with the least significant bit of each slave device. If the ROM ID bits match for all currently selected slave devices, the two read time slots will reflect complementary data, and the bus master will not need to deselect or remove any slave devices from the selection process. The bus master simply repeats the Read Time Slot 1 data as its Write Time Slot data in the third time slot, and continues to the next higher ROM ID bit acquisition period. Since it is expected that all 1-Wire devices have unique ROM IDs, the first two read time slots inevitably result in conflicting data being driven on the bus for at least one bit position when multiple slaves are connected. When this occurs, the wired-AND line state yields a 0 for both read time slots. At this point, the master has to send a bit value 1 or 0 to select the devices that remain in the search process. All deselected devices are idle until they receive a Reset Pulse. Table 13-2 shows the four possible scenarios for slave ROM ID read time slots.

Table 13-2. ROM ID Read Time Slot Possibilities

READ TIME SLOT 1 (SLAVE)	READ TIME SLOT 2 (SLAVE)	WRITE TIME SLOT (MASTER)	DESCRIPTION
0	1	0	All slave devices remaining in the selection process have a 0 in this ROM ID bit position.
1	0	1	All slave devices remaining in the selection process have a 1 in this ROM ID bit position.
0	0	0 or 1	ID Discrepancy—Slave devices remaining in the selection process have both 0 and 1 in this ROM ID bit position. The Bus Master write time slot dictates which devices remain in the selection process.
1	1	1	Error—No slave devices responded during the read time slots.

The general principle of this search process is to deselect slave devices at every conflicting bit position. At the end of each ROM Search process, the master has learned another ROM ID. A pass of search process takes 64 reading/selection cycles for the master to learn one device's ROM ID. Each reading/selection cycle, as noted above, consists of two Read time slots and a Write time slot. Subsequent search passes are performed identically to the last up until the point of the last decision. For details of Search ROM algorithm in the 1-Wire system, refer to the *Book of iButton Standards*.

To speed up this ROM ID Search process, the 1-Wire Bus Master incorporates a Search ROM Accelerator. To enable the Search ROM accelerator, the SRA bit in the Command Register must be set immediately following the Reset sequence and issuance of the Search ROM command. Note that the receive buffer must empty before invoking SRA mode. After the bus master is placed in Search ROM Accelerator mode, each byte loaded into the transmit buffer contains one nibble (4 bits) worth of discrepancy decision data. The two slave read time slots are automatically generated by the bus master as a part of the transmit sequence. After four reading/selection cycles, the receive buffer data will reflect four newly acquired bits of the ROM ID and four corresponding bits flagging whether a discrepancy existed in a given bit position. Table 13-3 details the format for the transmit and receive data (when in Search ROM Accelerator mode).

Table 13-3. Search ROM Accelerator Transmit/Receive Byte Sequence

BYTE SEQUENCE	BUFFER	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Byte 1	Transmit	r_3	x	r_2	x	r_1	x	r_0	x
	Receive	ID_3	d_3	ID_2	d_2	ID_1	d_1	ID_0	d_0
Byte 2	Transmit	r_7	x	r_6	x	r_5	x	r_4	x
	Receive	ID_7	d_7	ID_6	d_6	ID_5	d_5	ID_4	d_4
•••									
Byte 16	Transmit	r_{63}	x	r_{62}	x	r_{61}	x	r_{60}	x
	Receive	ID_{63}	d_{63}	ID_{62}	d_{62}	ID_{61}	d_{61}	ID_{60}	d_{60}

r_n = decision discrepancy data (write time slot selection data if ID discrepancy)
ID_n = selected ROM ID bit (r_n if discrepancy occurred, otherwise read time slot 1)
d_n = discrepancy detected flag (ID discrepancy or no response)
x = don't care data

The CPU must send and receive 16 bytes of data to complete a single Search ROM pass on the 1-Wire bus. To perform a Search ROM sequence one starts with all decision discrepancy bits (r_n) being 0. In case of bus error, all subsequent response bits IDn are 1s until the Search Accelerator is deactivated by clearing the SRA bit in the Command Register. Thus if ID_{63} and d_{63} are both 1, an error has occurred during the search process and the last sequence has to be repeated. Otherwise, $ID_{63:0}$ is the ROM code of the device that has been found and addressed. For the next Search ROM sequence one reuses the previous set r_n (for n = 0:63), changing to 1 only that bit position where the highest discrepancy was detected (d_n flags). This process is repeated until the highest discrepancy occurs in the same bit position for two passes, then the next lower discrepancy flag is used for next search. When the Search ROM process is completed, the SRA bit should be cleared in order to release the 1-Wire Master from Search ROM Accelerator mode.

13.5.1 Accelerated ROM Search Example

The following example should provide a better understanding of how the Search ROM Accelerator functionality allows the 1-Wire Master to identify four different devices on the 1-Wire bus with ROM IDs as shown (least significant bit first):

ROM1 = 00110101....

ROM2 = 10101010....

ROM3 = 11110101....

ROM4 = 00010001....

1) The host issues a reset pulse by writing 01h to the Command Register. All slave devices respond simultaneously with a presence detect.
2) The host issues a Search ROM command by writing F0h to the Transmit Buffer. The host must wait for RBF flag and read (empty) the Receive Buffer.
3) The host places the 1-Wire Master in Search ROM Accelerator mode by writing 02h to the Command Register.
4) The host writes 00h to Transmit Buffer and reads the return data from the Receive Buffer. This process is repeated for total of 16 bytes. The data read contains ROM4 in the ID bit locations and the discrepancy flags d_0 and d_2 are set. This can easily be seen by examining the ROM IDs bit by bit. The first discrepancy occurs in bit position 0 (d_0). The bus master write time slot contains a 0, thus deselecting ROM2 and ROM3. A discrepancy between ROM1 and ROM4 then occurs in bit position 2 (d_2), leaving only ROM4 in the search. The receive data is as follows (d_0ID_0 d_1ID_1 d_2ID_2 d_3ID_3 d_4ID_4 d_5ID_5 d_6ID_6 d_7ID_7 ....):

   Receive Data = **1**0 **0**0 **1**0 **0**1 **0**0 **0**0 **0**0 **0**1....
5) The host then deinterleaves the data to arrive at a ROM ID of 00010001... and discrepancy data **(bold)** of 10100000....with the last discrepancy at location d_2.
6) The host writes 0x00h to the Command Register to exit accelerator mode. The host is now free to send a command or read data directly from this device.
7) Steps 1 to 6 are now repeated to find the next device on the bus. The 16 bytes of data transmitted this time are identical to ROM4 up until the last discrepancy flag (d_2 in this case), which is inverted, and all higher order decision discrepancy data bits are set to 0 as shown: $r_0r_1r_2r_3r_4r_5$....= 001000..... For this search iteration, the receive data contains ROM1 in the ID bit locations, again with discrepancy flags d_0 and d_2 set.

   Receive Data = **1**0 **0**0 **1**1 **0**1 **0**0 **0**1 **0**0 **0**1....
8) Since the most significant discrepancy (d_2) did not change, the next highest discrepancy (d_0) is used for the next search $r_0r_1r_2r_3r_4r_5$....=100000.....

   Receive Data = **1**1 **1**0 **0**1 **0**0 **0**1 **0**0 **0**1 **0**0....

Deinterleaving yields a ROM ID of 10101010.. (ROM2) and discrepancy flags of 11000000.. (d_1 is the most significant flag).

9) The next search uses the ROM ID acquired in the previous search up until the most significant discrepancy: $r_0r_1r_2r_3r_4r_5$....=110000...

   Receive Data = **1**1 **1**1 **0**1 **0**1 **0**0 **0**1 **0**0 **0**1....

   Deinterleaving yields a ROM ID of 11110101.. (ROM3) and discrepancy flags of 11000000.. (d_1 is the most significant flag).
10) At this point, the most significant discrepancy (d_1) did not change so the next highest discrepancy (d_0) should be used. However, d_0 has now been reached for the second time, and since there are no lesser significant discrepancies possible, the search is completed and all four devices are identified.

13.6 1-Wire Transmit and Receive Operations

All data transmitted and received by the 1-Wire Bus Master passes through the transmit/receive data buffer (internal register address A[2:0] = 001b).

The data buffer combination for the transmit interface is composed of the Transmit Buffer and Transmit Shift Register. Each of these registers has a flag that can be used as an interrupt source. The Transmit Buffer Empty (TBE) flag is set when the Transmit Buffer is empty and ready to accept a new byte of data from the user. As soon as the data byte is written into the Transmit Buffer, TBE is cleared. The Transmit Shift Register Empty (TEMT) flag is set when the shift register has no data and is ready to load a new data byte from the Transmit Buffer. When a byte of data is transferred into the Transmit Shift Register, TEMT is cleared and TBE becomes set.

To send a byte of data on the 1-Wire bus, the user writes the desired data to the Transmit Buffer. The data is moved to the Transmit Shift Register, where it is shifted serially onto the 1-Wire bus, least significant bit first. When the Transmit Shift Register is empty, new data will be transferred from the Transmit Buffer (if available) and the serial process repeats. Note that the 1-Wire protocol requires a reset before any bus communication.

The data buffer combination for the receive interface is composed of the Receive Buffer and the Receive Shift Register. The receive registers can also generate interrupts. The Receive Shift Register Full (RSRF) flag is set at the start of data being shifted into the register, and is cleared when the Receive Shift Register is empty. The Receive Buffer Full (RBF) flag is set when data is transferred from the Receive Shift Register into the Receive Buffer and is cleared after the CPU reads the register. If RBF is set, and another byte of data is received in the Receive Shift Register, the Receive Shift Register will hold the new byte and wait until the user reads the Receive Buffer, clearing the RBF flag. Thus, if both RSRF and RBF are set, no further transmissions should be made on the 1-Wire bus, or else data may be lost, as the byte in the Receive Shift Register will be overwritten by the next received data.

To read data from a slave device, the Bus Master must first be ready to transmit data depending on commands in the Command Register already set up by the CPU. Data is retrieved from the bus in a similar fashion to a write operation. The CPU initiates a read operation by writing FFh data to the Transmit buffer. The data that will then be shifted into the Receive Shift Register is the wired-AND of the Bus Master write data (FFh) and the data from the slave device. When the Receive Shift Register is full, the data is transferred to the Receive Buffer (if RBF = 0), where the CPU can read it. Additional bytes can be read by sending FFh again. If the slave device is not ready to respond to read request, the data received the by the Bus Master will be identical to that which was transmitted (FFh).

13.6.1 1-Wire Transmit/Receive Buffer (OWA = 001b)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	Input/Output Buffer (8 Bits)							
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: 1-Wire Transmit/Receive Buffer (OWA = 001b.[7:0]). All data transmit and receive activity of the 1-Wire Bus Master passes through the transmit/receive data buffer. The data buffer is double buffered with separate transmit and receive buffers. Double buffering of the transmit buffer and receive buffer does not work when SRA (Search ROM Accelerator) mode is active. Writing to the data buffer connects the Transmit Buffer to the data bus while reading connects the Receive Buffer to the data bus.

13.7 1-Wire Bus Master Interrupts

The 1-Wire Bus Master can be configured to generate an interrupt request to the CPU on the occurrence of a number of 1-Wire related events or conditions. These include the following: Presence Detect, Transmit Buffer Empty, Transmit Shift Register Empty, Receive Buffer Full, Receive Shift Register Full, 1-Wire Short, and 1-Wire Low. Each of these potential 1-Wire interrupt sources has a corresponding enable bit and flag bit. Each flag bit in the Interrupt Flag register (A2:A0 = 010b) is set, independent of the interrupt enable bit, when the associated event or condition occurs. For the interrupt flag to generate an interrupt request to the CPU, the individual enable bit for the source along with the 1-Wire Bus Master interrupt enable bit (EOWMI; Control Register bit 7) must be set to a logic 1 as well as having interrupts enabled modularly and globally. To clear the 1-Wire Bus Master Interrupt, a read of the Interrupt Flag Register must always be performed by software. The Interrupt Enable and Interrupt Flag registers are documented below.

13.7.1 1-Wire Interrupt Flag Register (OWA = 010b)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	OW_LOW	OW_SHORT	RSRF	RBF	TEMT	TBE	PDR	PD
Reset	0	0	0	0	1	1	1	0
Access	r	r	r	r	r	r	r	r

r = read, w = write

Bit 7: 1-Wire Low Interrupt (OW_LOW). This flag is set to 1 when the OW line is low while the master is in idle, signaling that a slave device has issued a presence pulse on the OW line. When this bit is 0, it shows that the OW line is high while the master is in idle. A read to the Interrupt Flag register clears this bit if OW is no longer a low while the master is idle.

Bit 6: 1-Wire Short Interrupt (OW_SHORT). This flag is set to 1 when the OW line was low before the master was able to send out the beginning of a reset or a time slot. When this flag is 0, it indicates that the OW line was high prior to all resets and time slots. A read to the Interrupt Flag register clears this bit.

Bit 5: Receive Shift Register Full (RSRF). This flag is set to 1 when there is a byte of data waiting in the Receive Shift register. When this bit is 0, it indicates that the Receive Shift register either empty or currently receiving data. This bit is cleared by the hardware when data in the Receive Shift register is transferred to the Receive Buffer. A read to the Interrupt Flag register has no effect on this bit.

Bit 4: Receive Buffer Full (RBF). This flag is set to 1 when there is a byte of data waiting to be read in the Receive Buffer. When this bit is 0, it indicates that the Receive Buffer has no new data to be read. This bit is cleared when the byte is read from the Receive Buffer. A read to the Interrupt Flag register has no effect on this bit. However, following a read of the Interrupt Flag register while Enable Receive Buffer Full Interrupt (ERBF) is set to 1, if the ERBF is not cleared and the value is not read from the Receive Buffer, the interrupt will fire again.

Bit 3: Transmit Shift Register Empty (TEMT). This flag is set to 1 when there is nothing in the Transmit Shift register and is ready to receive the next byte of data to be transmitted from the Transmit Buffer.

When this bit is 0, it indicates that the Transmit Shift Register is busy sending out data. This bit is cleared when data is transferred from the Transmit Buffer to the Transmit Shift register. A read to the Interrupt Flag Register has no effect on this bit.

Bit 2: Transmit Buffer Empty (TBE). This flag is set to 1 when there is nothing in the Transmit Buffer and is ready to receive the next byte of data. When it is 0, it indicates that the transmit buffer is waiting for the transmit shift register to finish sending its current data before updating it. This bit is cleared when data is written to the Transmit Buffer. A read to the Interrupt Flag register has no effect on this bit.

Bit 1: Presence-Detect Result (PDR). When a presence-detect interrupt occurs, this bit reflects the result of the presence detect read. The bit is 0 if a slave device was found, or 1 if no device was found. Reading the Interrupt Flag register does not affect the state of this bit.

Bit 0: Presence Detect (PD). After a 1-Wire reset has been issued, this flag is set to 1 after the appropriate time for a presence-detect pulse to have occurred. This flag is 0 when the master has not recently issued a presence detect. This bit is cleared when the Interrupt Flag register is read.

13.7.2 1-Wire Interrupt Enable Register (OWA = 011b)

Bit #	7	6	5	4	3	2	1	0
Name	EOWL	EOWSH	ERSF	ERBF	ETMT	ETBE	—	EPD
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	r	rw

r = read, w = write

Bit 7: Enable 1-Wire Low Interrupt (EOWL). Setting this bit to logic 1 enables the 1-Wire low interrupt. If both EOWMI and EOWL are set, OWMI is asserted when OW_LOW flag is set. Clearing this bit disables OW_LOW as an active interrupt source.

Bit 6: Enable 1-Wire Short Interrupt (EOWSH). Setting this bit to logic 1 enables the 1-Wire short interrupt. If both EOWMI and EOWSH are set, OWMI is asserted when OW_SHORT flag is set. Clearing this bit disables OW_SHORT as an active interrupt source.

Bit 5: Enable Receive Shift Register Full Interrupt (ERSF). Setting this bit to logic 1 enables the receive shift register full interrupt. If both EOWMI and ERSF are set, OWMI is asserted when RSRF flag is set. Clearing this bit disables RSRF as an active interrupt source.

Bit 4: Enable Receive Buffer Full Interrupt (ERBF). Setting this bit to logic 1 enables the receive buffer full interrupt. If both EOWMI and ERBF are set, OWMI is asserted when RBF flag is set. Clearing this bit disables RBF as an active interrupt source.

Bit 3: Enable Transmit Shift Register Empty Interrupt (ETMT). Setting this bit to logic 1 enables the transmit shift register empty interrupt. If both EOWMI and ETMT are set, OWMI is asserted when TEMT flag is set. Clearing this bit disables TEMT as an active interrupt source.

Bit 2: Enable Transmit Buffer Empty Interrupt (ETBE). Setting this bit to logic 1 enables the transmit buffer empty interrupt. If both EOWMI and ETBE are set, OWMI is asserted when TBE flag is set. Clearing this bit disables TBE as an active interrupt source.

Bit 1: Reserved

Bit 0: Enable Presence Detect Interrupt (EPD). Setting this bit to a logic 1 enables the presence detect interrupt. If both EOWMI and EPD are set, OWMI will be asserted after an appropriate amount of time has passed for a presence-detect pulse to have occurred, whenever a 1-Wire Reset is sent while the presence-detect flag (PD) is also set. Clearing this bit disables the presence detect as an active interrupt source.

13.8 I/O Signaling

The 1-Wire bus requires strict signaling protocols to ensure integrity. The five protocols used by the 1-Wire Bus Master are initialization sequence (Reset Pulse followed by Presence Pulse), Write 0, Write 1, Read 0, and Read 1. The Bus Master initiates all of these types of signaling except the presence pulse. Figure 13-2 illustrates the details of these signaling protocols.

The initialization sequence is required to begin any communication with the bus slave devices. The 1-Wire Bus Master transmits a reset pulse for t_{RSTL}. The 1-Wire bus line is then pulled high by the pullup resistor. After detecting the rising edge on the OWOUT pin, the slave device waits for t_{PDH} and then transmits the Presence Pulse for t_{PDL}. A Presence Pulse following a Reset Pulse indicates the slave device is ready to accept a ROM command. The Bus Master samples the bus at tPDS after the slave device responds to test for a valid presence pulse. The result of this sample is stored in the PDR bit of the Interrupt Flag Register. The reset time slot ends tRSTH after the Bus Master releases the bus.

A write time slot is initiated when the 1-Wire Bus Master pulls the 1-Wire bus line from a logic high (inactive) level to a logic low level. The Bus Master generates a Write 1 time slot by releasing the line at t_{LOW1} and allowing the line to pullup to logic high level. On the other hand, the line is held low for t_{LOW0} to generate a Write 0 time slot. A slave device samples the 1-Wire bus line between 15µs and 60µs after the line falls. If the line is high when sampled, a Write 1 occurs. If the line is low when sampled, a Write 0 occurs.

A read time slot is initiated when the 1-Wire Master pulls the bus low for at least 1(s and then releases it. The slave device continues to hold the line low for up to 60µs if it is responding with a 0, otherwise it releases it immediately. The Bus Master samples the data t_{RDV} from the start of the read time slot. If the line is high when sampled, a Read 1 occurs. If the line is low when sampled, a Read 0 occurs. The Bus Master ends the read slot after t_{SLOT}.

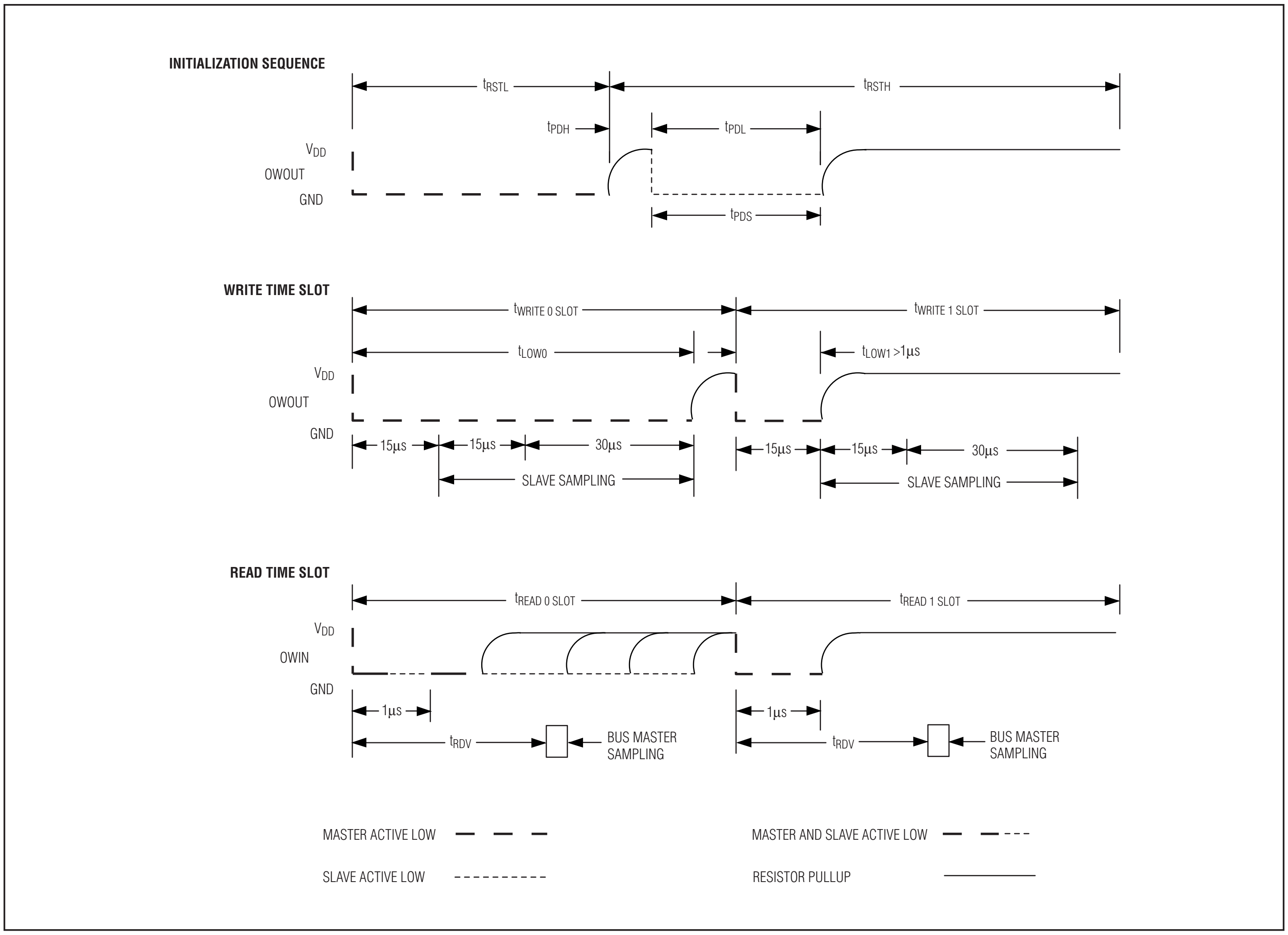


Figure 13-2. 1-Wire Bus Signaling in Standard Mode

SECTION 14: REAL-TIME CLOCK MODULE

This section contains the following information:

LIST OF FIGURES

SECTION 14: REAL-TIME CLOCK MODULE

The real-time clock (RTC) is a binary timer that keeps the time of day and provides time-of-day and sub-second alarm functionality in the form of system interrupts. The RTC consists of cascaded 32-bit and 8-bit ripple counters that respectively represent absolute seconds (~136 years) and sub-seconds (in 1/256 second resolution). The 8-bit sub-second counter increments with each 256Hz clock tick derived from the 32.768kHz oscillator. A separate auto-reload sub-second alarm counter can be used to generate interval alarms with granularity of 1/256 seconds. The 32-bit seconds counter increments with each rollover of the 8-bit sub-second counter. The 32-bit counter can be used with the programmable time-of-day comparison alarm to provide a single event timer. The RTC must be stopped for the counter registers to initially be written, but once enabled, the RTC counts continuously as long as it is enabled and does not stop for reads of the counter registers. The RTC also supports a digital trim facility for those applications requiring high accuracy.

User-application code accesses the RTC via eight peripheral registers. The 16-bit RTC Control register (RCNT) provides the control and status for RTC functions, including RTC read/write access controls, clock selection/control, RTC enable, square-wave output enable, alarm enables, and their interrupt flags. The seconds count information is set or initialized only by writing to the 16-bit RTC Second High and the 16-bit RTC Second Low register (RTSH and RTSL registers). Similarly, the 8-bit RTC Sub-Second register (RTSS) can be used to establish and access the sub-second counter. The Time-of-Day alarm is composed of the RASH:RASL register pair and the interval alarm is programmable via the RSSA register. The digital trim value is held in the RTRM register.

Figure 14-1 shows a functional diagram of a full-featured RTC. Certain MAXQ microcontroller devices may be equipped with a subset of the RTC functionality described. Refer to the individual device data sheet or supplemental user guide for more information.

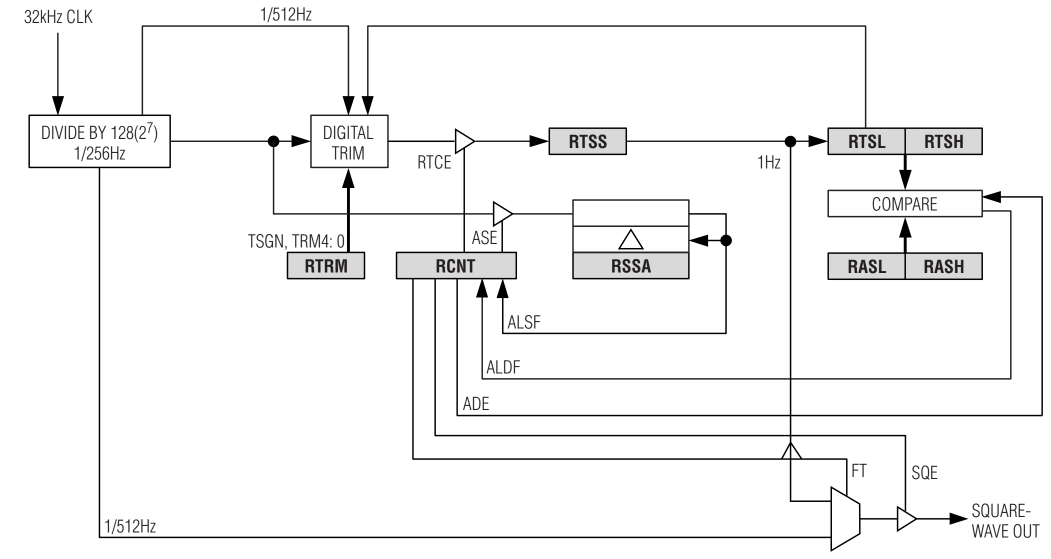


Figure 14-1. RTC Functional Block Diagram

14.1 RTC Alarm Functions

The RTC provides time-of-day and sub-second interval alarm functions. The time-of-day alarm, when enabled, occurs based upon matching of the least significant 20 bits of the RTC seconds counter information (RTSH:RTSL) with the least significant 20 bits of the alarm register values (RASH:RASL) defined by the user. The sub-second interval alarm provides an auto-reload timer that is driven by the untrimmed 256Hz clock source.

14.1.1 Time-of-Day Alarm

The RTC Alarm Second High and RTC Alarm Second Low registers (RASH and RASL) provide a programmable, 20-bit time-of-day alarm function. The 20 bits of the time-of-day alarm should be programmed with the desired value to match with the least significant 20 bits of the RTC seconds counter for the purpose of triggering an alarm. The least significant 16 bits of the time-of-day alarm are programmable in the RASL register, while the most significant 4 bits of the alarm value are programmable in the lowest 4 bits of the RASH register. The time-of-day alarm can be programmed to any future value between 1 second and 12 days relative to the current time with a resolution of 1 second. The time-of-day alarm must be disabled before the changing the time-of-day alarm registers. The time-of-day alarm is a single event alarm that sets the ALDF flag to 1 when an RTSS rollover occurs and the contents of RTSH and RTSL counter registers match the 20-bit value set in the RASH and RASL alarm registers. Setting the ALDF bit causes an interrupt request to the processor if the ADE bit and the system interrupt enable are set.

14.1.2 Sub-Second Alarm

The RTC Sub-Second Alarm (RSSA) register is used to store the sub-second interval alarm value for the sub-second alarm function. The RSSA register is independent of the RTC counter value and is configurable per MAXQ device as necessary to 16 bits to achieve sub-second interval alarms needed by various applications. The default RSSA register size is 8 bits wide, allowing a maximum interval alarm of 1 second and a programming resolution of ~3.9 milliseconds (1/256Hz). The sub-second interval alarm must be disabled (ASE = 0 and BUSY = 0) before changing the interval alarm value.

The delay (uncertainty) associated with the enabling of the interval sub-second alarm is up to one period of the sub-second clock (1/256Hz = ~3.9ms). Thus, the same uncertainty is associated with the first interval alarm. Thereafter, if the interval alarm remains enabled, the alarm triggers after each RSSA defined sub-second interval. This is due to the fact that the sub-second alarm is constructed as an auto-reload counter such that the RSSA alarm value is reloaded to the counter only on a rollover. Note that enabling the sub-second alarm (ASE = 1) with sub-second interval alarm register programmed to 0's results in the maximum sub-second alarm interval (1 second if RSSA = 8 bits wide).

The sub-second interval counter sources its clock from the 32kHz/128 counter before the possible insertion of pulses by the digital-trim facility. This is done to keep the alarm interval consistent and avoid deviations in the interval that would automatically be created each time the digital-trim facility were to add/subtract clock pulses.

14.1.3 System Wakeup by Time-of-Day or Sub-Second Interval Alarm

The time-of-day alarm or interval alarm can wake up the system from Stop mode if not already awake. The wakeup function is allowed only when these interrupts have not been masked at all levels. The time-of-day and interval alarms also qualify as valid Power Management Mode switchback sources.

14.2 RTC Trim Function

The uncompensated accuracy of the RTC is a function of the attached crystal (and its respective temperature drift characteristics within the end system). To accommodate those applications requiring high accuracy, a digital-trim facility is made accessible to the user. The trim facility allows extra clocks to be inserted or removed at the 256Hz stage of the divider chain. Five trim bits (TRM4:0) are used to control the trim adjustment, and the sign bit (TSGN) designates whether pulses should be inserted (TSGN = 0) or deleted (TSGN = 1). Every 16 seconds, the five trim bits are added to the previous phase accumulator value. Whenever the phase accumulator generates a carry-out from the addition, the output of the 512Hz stage is selected instead of the 256Hz stage if positive trim is selected, effectively adding 128 extra 32kHz clocks. If negative trim is selected, the 256Hz stage output pulse is masked in order to effectively subtract 128 32kHz clocks. The 512Hz or 1Hz output can be made accessible on an external pin as controlled by the FT and SQE bits. Figure 14-2 shows a block diagram illustrating the digital-trim facility. Figure 14-3 shows a representative timing diagram.

The minimum adjustment (00001h) would result in a phase accumulator carry-out every 32 x 16 seconds = 512 seconds or adding/subtracting 1 pulse (=128 cycles of 32.768kHz) every 512 seconds. This would be an adjustment of 128 / (32,768Hz x 512s) = ±7.63ppm. The maximum adjustment would be achieved by programming TRM4:0 = 11111b. This would result in an adjustment of 31 extra pulses per 512-second interval or (31 x 128) / (32,768Hz x 512s) = ±236.5ppm. This range of adjustment should be satisfactory to cover the temperature drift characteristics of most 32kHz crystals over the industrial temperature range.

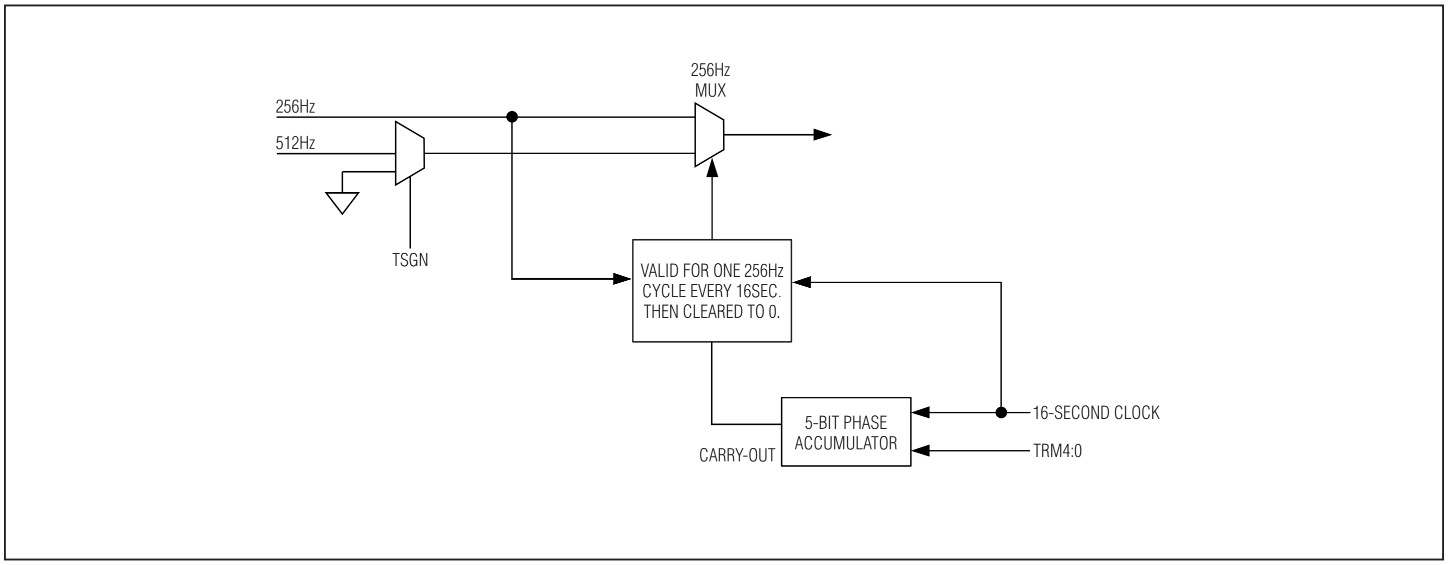


Figure 14-2. RTC Digital-Trim Facility Block Diagram

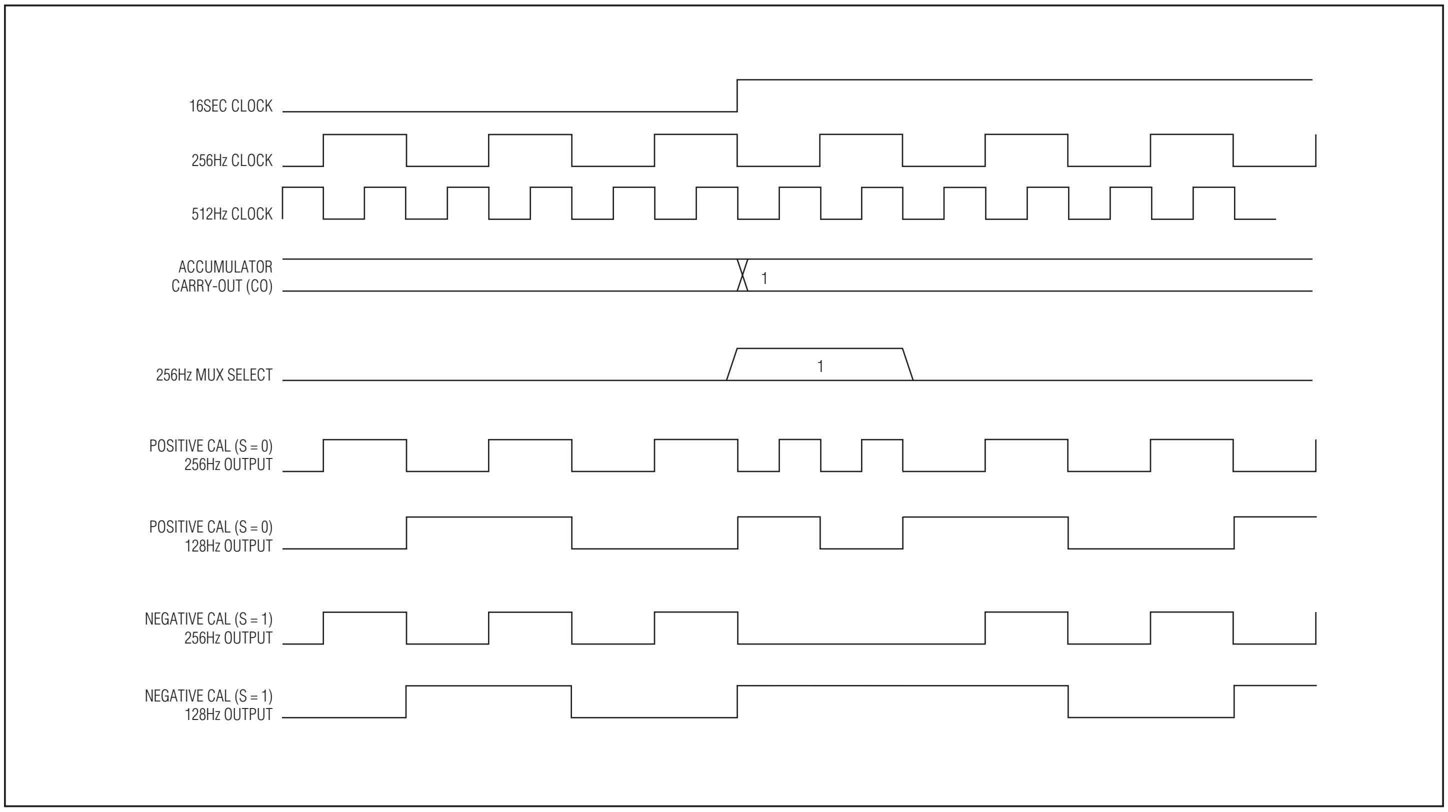

16SEC CLOCK
256Hz CLOCK
512Hz CLOCK
ACCUMULATOR CARRY-OUT (CO)
1
256Hz MUX SELECT
1
POSITIVE CAL (S = 0) 256Hz OUTPUT
POSITIVE CAL (S = 0) 128Hz OUTPUT
NEGATIVE CAL (S = 1) 256Hz OUTPUT
NEGATIVE CAL (S = 1) 128Hz OUTPUT

Figure 14-3. Digital Trim Pulse Calibration Diagram

14.3 RTC Register Access

Since RTC registers and register bits must be used in the 32kHz clock domain and also be accessible in the system clock domain, a handshaking or signaling protocol is implemented to simplify user access.

14.3.1 Busy Bit Write Signaling

The BUSY bit of the RTC Control (RCNT) register is a read-only status bit. Hardware sets the BUSY bit when any of the following conditions occur: 1) system reset; 2) software changes the state of any the following RTC bits (RTCE, ASE, ADE); 3) software writes to any RTC count register (RTSS, RTSL, RTSH). When the BUSY bit is set by hardware, writes to the referenced RTC control bits and count registers are blocked by hardware. The BUSY bit remains active until a synchronized 32kHz version of the register (or bit) is in place. This takes place when the next rising edge of the 32kHz clock occurs, which means that the BUSY bit is set for a duration no longer than one 32kHz clock = ~30.5µs. Once the BUSY bit is cleared to 0, additional writes can be performed as permitted by individual count or alarm-enable bits.

14.3.2 Ready Bit Read Signaling

The Ready (RDY) bit of the RTC Control (RCNT) register provides a mechanism for determining when the RTC count registers are stable and may be reliably read. The RDY bit is cleared by hardware approximately one 32kHz clock before the ripple occurs through the RTC counter chain (RTSS, RTSL, RTSH) and is set once again immediately after the ripple occurs. The period of the RDY bit set/clear activity (as controlled by hardware) is therefore 1/256Hz = 3.9ms, providing a very large window during which the RTC count registers may be read. The RDY bit can be cleared by software at any time and remains clear until set by hardware again. A separate Ready Enable (RDYE) bit is provided in the RCNT register for the purpose of generating an interrupt whenever the RDY bit is set by hardware. This interrupt can be used to signal the start of a new RTC read window. When RDYE is set to a 1 and RDY becomes set, an interrupt request is generated if enabled globally and modularly. When RDYE is cleared to 0, the setting of the RDY bit does not generate an interrupt.

14.3.3 RTC Count Register Access

The RTC Count registers (RTSS, RTSL, RTSH) should only be read when RDY = 1. Data read from these registers when RDY = 0 should be considered as invalid. To write the RTC count registers, the RTC Enable (RTCE) bit must be cleared to 0. Clearing of the RTCE bit is permitted only when the Write Enable (WE) bit is set to 1 and is governed by the BUSY bit signaling process (i.e., the BUSY bit is deasserted once a synchronized 32kHz version of the bit is in place). Writes to each RTC count register should also obey the rules associated with BUSY bit signaling.

14.3.4 RTC Alarm Register Access

The RTC Alarm registers (RSSA, RASL, RASH) are read-accessible at any time. To write the RTC alarm register, the respective alarm enable (ASE or ADE) bit must be cleared to 0. Clearing these bits requires monitoring the BUSY bit to assess completion of the write. Once the respective alarm enable is cleared, the associated RTC alarm register(s) can be freely written by user code.

14.3.5 RTC Trim Register Access

These RTC Trim bits (TSGN, TRM4:0) are read-accessible at any time. Write access to these bits requires that the Write Enable (WE) bit be set to logic 1 and is governed by the BUSY bit signaling process.

14.4 RTC Peripheral Registers

14.4.1 RTC Control Register (RCNT)

Bit #	15	14	13	12	11	10	9	8
Name	WE	X32D	ACS	—	—	—	FT	SQE
Power-On Reset	0	1	0	0	0	0	0	0
System Reset	0	u	u	0	0	0	0	0
Access	rw	rs	rs	r	r	r	rw	rw

Bit #	7	6	5	4	3	2	1	0
Name	ALSF	ALDF	RDYE	RDY	BUSY	ASE	ADE	RTCE
Power-On Reset	0	0	0	0	1	0	0	0
System Reset	u	u	0	0	1	u	u	u
Access	rw	rw	rw	rw	r	rw	rw	rs

r = read, w = write, s = special, u = unaffected

Bit 15: RTC Write Enable (WE). This register bit serves as an additional protection mechanism against unintentional writes to the RTC enable bit (RTCE). This bit must be set to logic 1 to gain write access to the RTCE bit. When this bit is configured to logic 0, the RTCE bit is read-only.

Bit 14: 32kHz Crystal Oscillator Disable (X32D). Setting this bit to logic 1 disables the internal oscillator circuitry connected between the 32kHz crystal pins. In this configuration, the RTC can be driven directly by an external clock signal provided on 32KIN or CX1 pin. Clearing this bit to logic 0 enables the internal crystal oscillator circuitry. When the internal oscillator circuitry is enabled, a warmup delay could be required before the 32kHz oscillator begins running. The exact length of this warmup delay can vary from among devices. Refer to the data sheet and user's guide supplement for the specific MAXQ device for details.

Bit 13: Alternate Clock Select (ACS). This bit enables the HFClk/128 clock to drive the RTC in place of the 32kHz clock. This bit is provided for those applications where a 32kHz clock may not be present. This bit may only be changed when RTCE = 0. When ACS = 1, the RTC is effectively halted anytime the high-frequency oscillator is disabled (e.g., stop mode).

Bits 12, 11, 10: Reserved

Bit 9: RTC Frequency Test (FT). This register bit selects the frequency output that is possible on the SQW pin if the square-wave output is enabled. Setting FT = 1 selects the 512Hz output (when SQE = 1), while FT = 0 selects the 1Hz output (when SQE = 1). This bit has no function if the square-wave output is disabled.

Bit 8: RTC Square-Wave Output Enable (SQE). Setting this bit to logic 1 enables either the 1Hz tap or the 512Hz tap of the RTC to the SQW pin. When cleared to 0, the SQW pin is not driven by the RTC.

Bit 7: Alarm Sub-Second Flag (ALSF). This bit is set when the subsecond timer has been reloaded by the RSSA register. Setting the ALSF causes an interrupt request to the CPU if ASE = 1 and interrupts are enabled at the system level.

Bit 6: Alarm Time-of-Day Flag (ALDF). This bit is set when the contents of RTSH and RTSL counter registers match the 20-bit value in the RASH and RASL alarm registers. Setting the ALDF will cause an interrupt request to the CPU if the ADE is set and interrupts are allowed at the system level. This alarm is qualified as a stop mode wakeup source and a potential switchback function if the interrupt has not been masked.

Bit 5: RTC Ready Enable (RDYE). Setting this bit to 1 allows a system interrupt to be generated when RDY becomes active (if interrupts are enabled globally and modularly). Clearing this bit to 0 disables the RDY interrupt.

Bit 4: RTC Ready (RDY). This bit is set to 1 by hardware when the RTC count registers have updated. It can be cleared to 0 by software at any time. It is also cleared to 0 by hardware just prior to an update of the RTC count register. This bit can generate an interrupt if the RDYE bit is set to 1.

Bit 3: RTC Busy (BUSY). This bit is set to 1 by hardware when any of the following conditions occur: 1) system reset, 2) software writes to RTC count registers, or 3) software changes RTCE, ASE, or ADE. For conditions 2 and 3, the write or change should not be considered complete until hardware clears the BUSY bit. This is an indication that 32kHz synchronized version of the register bit(s) is in place.

Bit 2: Alarm Sub-Second Enable (ASE). The ASE bit is the RTC's subsecond timer enable and must be set to logic 1 for the sub-second alarm to generate a system interrupt request. When the ASE is cleared to logic 0, the sub-second alarm is disabled, and no interrupt is generated even if the alarm is set.

Bit 1: Alarm Time-of-Day Enable (ADE). The ADE bit is the RTC's time-of-day alarm enable and must be set to logic 1 for the alarm to generate a system interrupt request. When the ADE is cleared to logic 0, the time-of-day alarm is disabled and no interrupt is generated on a time-of-day alarm (RASH:RASL) match.

Bit 0: Real-Time Clock Enable (RTCE). Setting this bit to logic 1 activates the RTC by allowing the 256Hz clock to the ripple counters. Clearing this bit to logic 0 disables the clock. This bit is writable only when WE (RCNT.15) = 1.

14.4.2 RTC Seconds High Register (RTSH)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	RTSH.15	RTSH.14	RTSH.13	RTSH.12	RTSH.11	RTSH.10	RTSH.9	RTSH.8
Power-On Reset	0	0	0	0	0	0	0	0
System Reset	u	u	u	u	u	u	u	u
Access	s	s	s	s	s	s	s	s

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	RTSH.7	RTSH.6	RTSH.5	RTSH.4	RTSH.3	RTSH.2	RTSH.1	RTSH.0
Power-On Reset	0	0	0	0	0	0	0	0
System Reset	u	u	u	u	u	u	u	u
Access	s	s	s	s	s	s	s	s

s = special, u = unaffected

Bits 15 to 0: RTC Seconds High (RTSH.[15:0]). This register contains the most significant bits for the 32-bit second counter. The RTC is a ripple counter that consists of a cascaded 32-bit second counter (RTSH, RTSL) and an 8-bit sub-second counter (RTSS). This register is write-accessible when RTCE = 0 and BUSY = 0, and should be read-only when RDY = 1.

14.4.3 RTC Seconds Low Register (RTSL)

Bit #	15	14	13	12	11	10	9	8
Name	RTSL.15	RTSL.14	RTSL.13	RTSL.12	RTSL.11	RTSL.10	RTSL.9	RTSL.8
Power-On Reset	0	0	0	0	0	0	0	0
Reset	u	u	u	u	u	u	u	u
Access	s	s	s	s	s	s	s	s

Bit #	7	6	5	4	3	2	1	0
Name	RTSL.7	RTSL.6	RTSL.5	RTSL.4	RTSL.3	RTSL.2	RTSL.1	RTSL.0
Power-On Reset	0	0	0	0	0	0	0	0
Reset	u	u	u	u	u	u	u	u
Access	s	s	s	s	s	s	s	s

s = special, u = unaffected

Bits 15 to 0: RTC Seconds Low (RTSL.[15:0]). This register contains the least significant bits for the 32-bit second counter. The RTC is a ripple counter that consists of a cascaded 32-bit second counter (RTSH, RTSL) and an 8-bit sub-second counter (RTSS). This register is write-accessible when RTCE = 0 and BUSY = 0, and should be read-only when RDY = 1.

14.4.4 RTC Sub-Seconds Register (RTSS)

Bit #	7	6	5	4	3	2	1	0
Name	RTSS.7	RTSS.6	RTSS.5	RTSS.4	RTSS.3	RTSS.2	RTSS.1	RTSS.0
Power-On Reset	0	0	0	0	0	0	0	0
Reset	u	u	u	u	u	u	u	u
Access	s	s	s	s	s	s	s	s

s = special, u = unaffected

Bits 7 to 0: RTC Sub-Seconds (RTSS.[7:0]). This ripple counter represents 1/256 second resolution for the RTC, and its content is incremented with each 256Hz clock tick derived from the 32.768 kHz oscillator (or alternate clock source if ACS = 1). When the RTSS counter rolls over, its output is used to drive the 32-bit seconds counter. This register is write-accessible when RTCE = 0 and BUSY = 0, and should be read-only when RDY = 1.

14.4.5 RTC Alarm Seconds High Register (RASH)

Bit #	7	6	5	4	3	2	1	0
Name	—	—	—	—	RASH.3	RASH.2	RASH.1	RASH.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	rs	rs	rs	rs

r = read, s = special

Bits 7 to 4: Reserved

Bits 3 to 0: RTC Alarm Seconds High (RASH.[3:0]). This register contains the most significant bits for the 20-bit time-of-day alarm. The time-of-day alarm is formed by the RASH and the RASL registers. The 20 bits of the RASH:RASL value are compared against the least significant 20 bits of the RTSH:RTSL seconds value for generating a time-of-day alarm. This register is write-accessible only when ADE = 0 and BUSY = 0.

14.4.6 RTC Alarm Seconds Low Register (RASL)

Bit #	15	14	13	12	11	10	9	8
Name	RASL.15	RASL.14	RASL.13	RASL.12	RASL.11	RASL.10	RASL.9	RASL.8
Reset	0	0	0	0	0	0	0	0
Access	rs	rs	rs	rs	rs	rs	rs	rs

Bit #	7	6	5	4	3	2	1	0
Name	RASL.7	RASL.6	RASL.5	RASL.4	RASL.3	RASL.2	RASL.1	RASL.0
Reset	0	0	0	0	0	0	0	0
Access	rs	rs	rs	rs	rs	rs	rs	rs

r = read, s = special

Bits 15 to 0: RTC Alarm Seconds Low (RASL.[15:0]). This register contains the least significant bits for the 24-bit time-of-day alarm. The time-of-day alarm is formed by the RASH and the RASL registers, and only the lower 20 bits are meaningful for the alarm function. The lower 20 bits of the 24-bit RASH:RASL value are compared against the least significant 20 bits of the RTSH:RTSL seconds value for generating a time-of-day alarm. This register is write-accessible only when ADE = 0 and BUSY = 0.

14.4.7 RTC Sub-Second Alarm Register (RSSA)

Bit #	15	14	13	12	11	10	9	8
Name	RSSA.15	RSSA.14	RSSA.13	RSSA.12	RSSA.11	RSSA.10	RSSA.9	RSSA.8
Reset	0	0	0	0	0	0	0	0
Access	rs	rs	rs	rs	rs	rs	rs	rs

Bit #	7	6	5	4	3	2	1	0
Name	RSSA.7	RSSA.6	RSSA.5	RSSA.4	RSSA.3	RSSA.2	RSSA.1	RSSA.0
Reset	0	0	0	0	0	0	0	0
Access	rs	rs	rs	rs	rs	rs	rs	rs

r = read, s = special

Bits 15 to 0: RTC Sub-Second Alarm (RSSA.[15:0]). This register contains the reload value for the sub-second alarm. The ALSF bit is set when an auto-reload occurs. The width of the RSSA register for any given MAXQ microcontroller and hence, the longest programmable interval, is device dependent. This register is write-accessible only when ASE = 0 and BUSY = 0.

14.4.8 RTC Trim Register (RTRM)

Bit #	7	6	5	4	3	2	1	0
Name	—	—	TSGN	TRM4	TRM3	TRM2	TRM1	TRM0
Power-On Reset	0	0	0	0	0	0	0	0
Reset	0	0	u	u	u	u	u	u
Access	r	r	rs	rs	rs	rs	rs	rs

r = read, s = special, u = unaffected

Bits 7 and 6: Reserved

Bit 5: Trim Sign Bit (TSGN). This register bit selects whether the trim calibration for the RTC is positive (TSGN = 0) or negative (TSGN = 1). The trim bits are always readable, but are write-accessible only when WE = 1.

Bits 4 to 0: Trim Calibration Bits (TRM[4:0]). These register bits provide a binary value between 0b–31b, which is used for accumulation every 16 seconds. The carry-out of the accumulation determines when an additional 128 32kHz-input clock cycles are added/subtracted to the RTC counter chain. The trim bits are always readable, but are write-accessible only when WE = 1.

SECTION 15: TEST ACCESS PORT (TAP)

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 15: TEST ACCESS PORT (TAP)

The MAXQ microcontroller incorporates a Test Access Port (TAP) and TAP controller for communication with a host device across a 4-wire synchronous serial interface. The TAP can be used by MAXQ microcontrollers to support in-system programming and/or in-circuit debug. The TAP is compatible with the JTAG IEEE standard 1149, and is formed by four interface signals, as described in the following table. For detailed information on the TAP and TAP controller, refer to IEEE STD 1149.1 "IEEE Standard Test Access Port and Boundary-Scan Architecture."

EXTERNAL PIN SIGNAL	FUNCTION
TDO (Test Data Output)	Serial-Data Output. This signal is used to serially transfer internal data to the external host. Data is transferred least significant bit first. Data is driven out only on the falling edge of TCK, only during TAP Shift-IR or Shift-DR states and is otherwise inactive.
TDI (Test Data Input)	Serial-Data Input. This signal is used to receive data serially transferred by the host. Data is received least significant bit first and is sampled on the rising edge of TCK. TDI is weakly pulled high internally when TAP = 1.
TCK (Test Clock Input)	Serial Shift Clock Provided by Host. When this signal is stopped at 0, storage elements in the TAP logic must retain their data indefinitely. TCK is weakly pulled high internally when TAP = 1.
TMS (Test Mode Select Input)	Mode Select Input. This signal is sampled at the rising edge of TCK and controls movement between TAP states. TMS is weakly pulled high internally when TAP = 1.

15.1 TAP Controller

The TAP controller is a synchronous state machine that responds to changes at the TMS and TCK signals. Based on its state transition, the controller provides the clock and control sequence for TAP operation. The performance of the TAP is dependent on the TCK clock frequency. The maximum TCK clock frequency should be limited to 1/8 the system clock frequency. This section provides a brief description of the state machine and its state transitions. The state diagram in Figure 15-1 summarizes the transitions caused by the TMS signal sampling on the rising edge at TCK. The TMS signal value is presented adjacent to each state transition in the figure.

15.2 TAP State Control

The TAP provides an independent serial channel to communicate synchronously with the host system. The TAP state control is achieved through host manipulation of the Test Mode Select (TMS) and Test Clock (TCK) signals. The TMS signal is sampled at the rising edge of TCK and decoded by the TAP controller to control movement between the TAP states. The TDI input and TDO output are meaningful once the TAP is in a serial shift state (i.e., Shift-IR or Shift-DR).

15.2.1 Test-Logic-Reset

On a power-on reset, the TAP controller is initialized to the Test-Logic-Reset state and the instruction register (IR2:0) is initialized to the By-Pass instruction so that it does not affect normal system operation. No matter what the state of the controller, it enters Test-Logic-Reset when TMS is held high for at least five rising edges of TCK. The controller remains in the Test-Logic-Reset state if TMS remains high. An erroneous low signal on the TMS can cause the controller to move into the Run-Test-Idle state, but no disturbance is caused to system operation if the TMS signal is returned and kept at the intended logic high for three rising edges of TCK since this returns the controller to the Test-Logic-Reset state.

15.2.2 Run-Test-Idle

As illustrated in Figure15-1, the Run-Test-Idle state is simply an intermediate state for getting to one of the two state sequences in which the controller performs meaningful operations:

- Controller state sequence (IR-Scan), or
- Data register state sequence (DR-Scan)

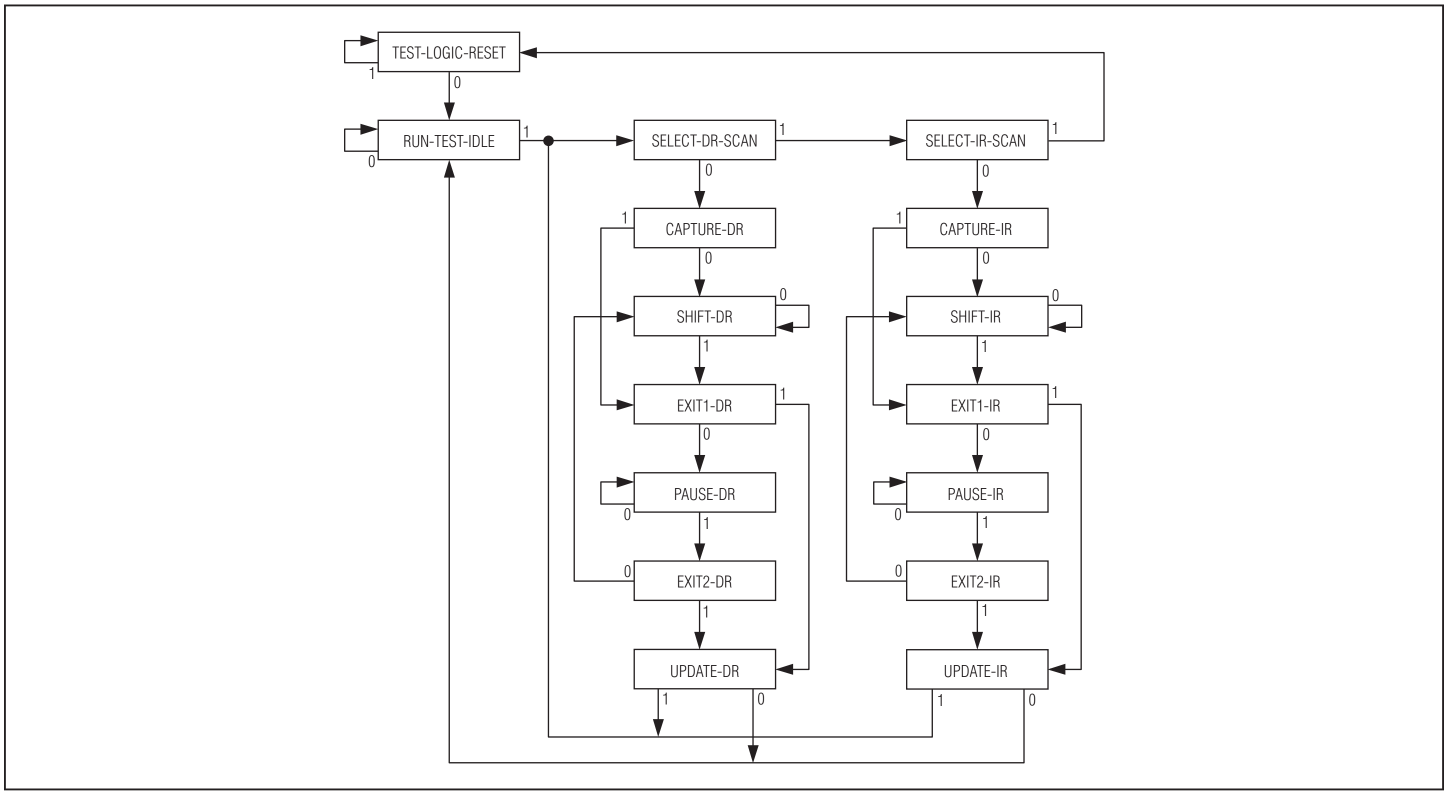


Figure 15-1. TAP Controller State Diagram

15.2.3 IR-Scan Sequence

The controller state sequence allows instructions (e.g., 'Debug' and 'System Programming') to be shifted into the instruction register starting from the Select-IR-Scan state. In the TAP, the instruction register is connected between the TDI input and the TDO output. Inside the IR-Scan Sequence, the Capture-IR state loads a fixed binary pattern (001b) into the 3-bit shift register and the Shift-IR state causes shifting of TDI data into the shift register and serial output to TDO, least significant bit first. Once the desired instruction is in the shift register, the instruction can be latched into the parallel instruction register (IR2:0) on the falling edge of TCK in the Update-IR state. The contents of the 3-bit instruction shift register and parallel instruction register (IR2:0) are summarized with respect to the TAP controller states in Table 15-1.

Table 15-1. Instruction Register Content vs. TAP Controller State

TAP CONTROLLER STATE	INSTRUCTION SHIFT REGISTER	PARALLEL (3-BIT) INSTRUCTION REGISTER (IR2:0)
Test-Logic-Reset	Undefined	Set to By-pass (011b) Instruction
Capture-IR	Load 001b at the rising edge of TCK	Retain last state
Shift-IR	Input data via TDI and Shift towards TDO at the rising edge of TCK	Retain last state
Exit1-IR, Exit2-IR, Pause-IR	Retain last state	Retain last state
Update-IR	Retain last state	Load from shift register at the falling edge of TCK
All other states	Undefined	Retain last state

When the parallel instruction register (IR2:0) is updated, the TAP controller decodes the instruction and performs any necessary operations, including activation of the data shift register to be used for the particular instruction during data register shift sequences (DR-Scan). The length of the activated shift register depends upon the value loaded to the instruction register (IR2:0). The supported instruction-register encodings and associated data-register selections are shown in Table 15-2.

Table 15-2. Instruction Register (IR2:0) Encodings

IR2:0	INSTRUCTION	FUNCTION	SERIAL DATA SHIFT REGISTER SELECTION
000	Extest	No operation	Unchanged (retain previous selection)
001	Sample/Preload	No operation	Unchanged (retain previous selection)
010	Debug	In-circuit debug mode	10-bit shift register
011	By-pass	No operation (default)	1-bit shift register
100	System Programming	Bootstrap function	3-bit shift register
101	By-pass	No operation (default)	1-bit shift register
110	Reserved		
111	By-pass	No operation (default)	1-bit shift register

The Extest (IR2:0 = 000b) and Sample/Preload (IR2:0 = 001b) instructions are mandated by the JTAG standard, however, the MAXQ microcontroller does not intend to make practical use of these instructions. Hence, these instructions are treated as no operations but may be entered into the instruction register without affecting the on-chip system logic or pins and does not change the existing serial data register selection between TDI and TDO.

The By-pass (IR2:0 = 011b, 101b, or 111b) instruction is also mandated by the JTAG standard. The By-pass instruction is fully implemented by the MAXQ microcontroller to provide a minimum length serial data path between the TDI and the TDO pins. This is accomplished by providing a single cell bypass shift register. When the instruction register is updated with the By-pass instruction, a single bypass register bit is connected serially between TDI and TDO in the Shift-DR state. The instruction register automatically defaults to the By-pass instruction when the TAP is in the Test-Logic-Reset state. The By-pass instruction has no effect on the operation of the on-chip system logic.

The Debug (IR2:0 = 010b) and System Programming (IR2:0 = 100b) instructions are private instructions that are intended solely for in-circuit debug and in-system programming operations respectively. If the instruction register is updated with the Debug instruction, a 10-bit serial shift register is formed between the TDI and TDO pins in the Shift-DR state. If the System Programming instruction is entered into the instruction register (IR2:0), a 3-bit serial data shift register is formed between the TDI and TDO pins in the Shift-DR state.

Instruction register (IR2:0) settings other than those listed and described above are reserved for internal use. As can be seen in Figure 15-2, the instruction register serves to select the length of the serial data register between TDI and TDO during the Shift-DR state.

15.2.4 DR-Scan Sequence

Once the instruction register has been configured to a desired state (mode), transactions are performed via a data buffer register associated with that mode. These data transactions are executed serially in a manner analogous to the process used to load the instruction register and are grouped in the TAP Controller state sequence starting from the Select-DR-Scan state. In the TAP controller state sequence, the Shift-DR state allows internal data to be shifted out through the TDO pin while the external data is shifted in simultaneously via the TDI pin. Once a complete data pattern is shifted in, input data can be latched into the parallel buffer of the selected register on the falling edge of TCK in the Update-DR state. On the same TCK falling edge, in the Update-DR state, the internal parallel buffer is loaded to the data shift register for output. This Shift-DR/Update-DR process serves as the basis for passing information between the external host and the MAXQ microcontroller. These data register transactions occur in the data register portion of the TAP controller state sequence diagram and have no effect on the instruction register.

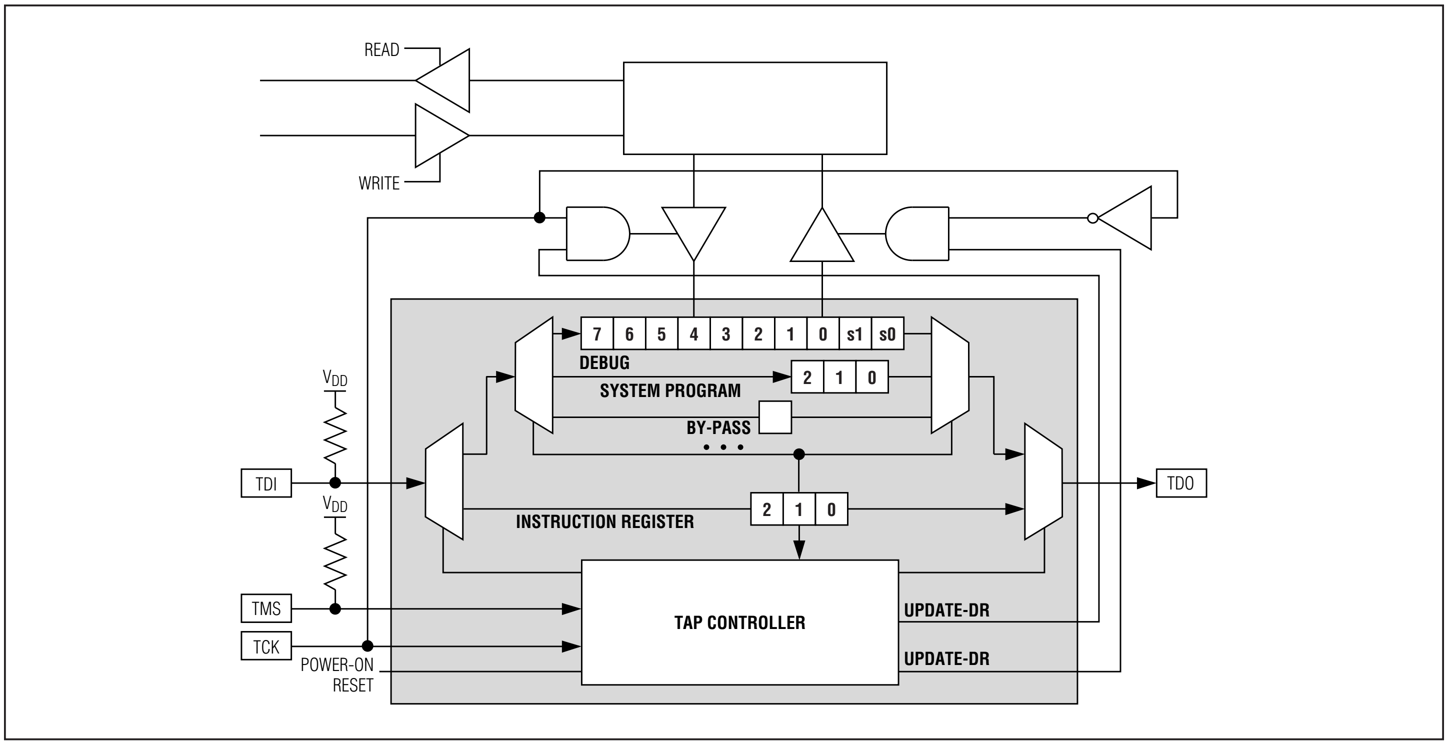


Figure 15-2. TAP and TAP Controller

15.3 Communication via TAP

The TAP controller is in Test-Logic-Reset state after a power-on-reset. During this initial state, the instruction register contains By-pass instruction and the serial path defined between the TDI and TDO pins for the Shift-DR state is the 1-bit bypass register. All TAP signals (TCK, TMS, TDI, and TDO) default to being weakly pulled high internally on any reset. The TAP controller remains in the Test-Logic-Reset state as long as TMS is held high. The TCK and TMS signals can be manipulated by the host to transition to other TAP states. The TAP controller remains in a given state whenever TCK is held low.

For the host to establish a specific data communication link, a private instruction must be loaded into the IR2:0 register. Once the instruction is latched in the instruction parallel buffer at the Update-IR state, it is recognized by the TAP controller and the communication channel is established. In-Circuit Debug or In-System Programming commands and data can be exchanged between the host and the MAXQ microcontroller by operating in the data register portion of the state sequence (i.e., DR-Scan). The TAP retains the private instruction that was loaded into IR2:0 until a new instruction is shifted in or until the TAP controller returns to the Test-Logic-Reset state.

15.3.1 TAP Communication Examples—IR-Scan and DR-Scan

Figures 15-3 and 15-4 illustrate examples of communication between the host JTAG controller and the Test Access Port (TAP) of the MAXQ microcontroller. The host controls the TCK and TMS signals to move through the desired TAP states while accessing the selected shift register through the TDI input and TDO output pair.

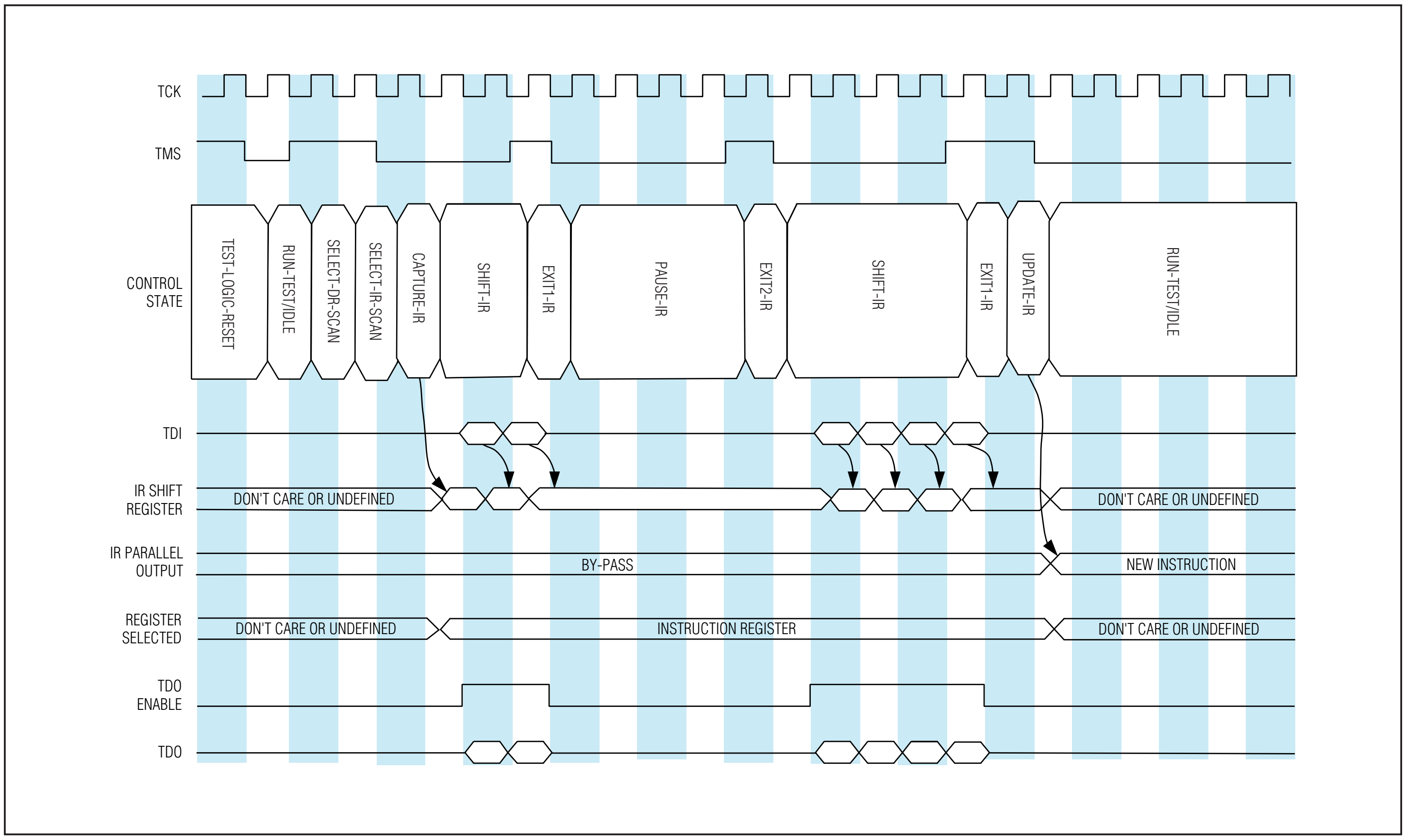


Figure 15-3. TAP Controller Debug Mode IR-Scan Example

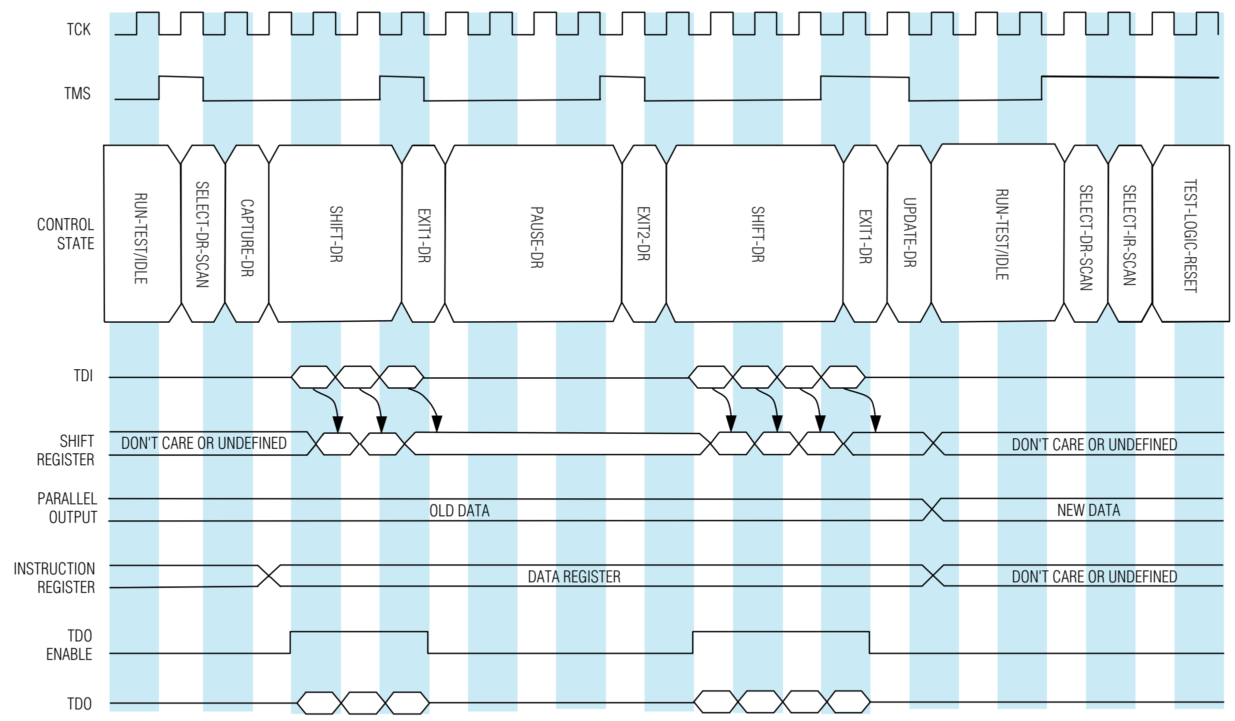


Figure 15-4. TAP Controller Debug Mode DR-Scan Example

SECTION 16: IN-CIRCUIT DEBUG MODE

This section contains the following information:

LIST OF FIGURES

LIST OF TABLES

SECTION 16: IN-CIRCUIT DEBUG MODE

Most MAXQ microcontroller devices are equipped with embedded debug hardware and embedded ROM firmware developed for the purpose of providing in-circuit debugging capability to the user application. The in-circuit debug mode uses the JTAG-compatible TAP as its means of communication between the host and MAXQ microcontroller. Figure 16-1 shows a block diagram of the in-circuit debugger. The in-circuit debug hardware and software features include:

- a debug engine
- a set of registers providing the ability to set breakpoints on register, code, or data
- a set of debug service routines stored in a ROM

Collectively, these hardware and software features allow two basic modes of in-circuit debugging:

- Background mode allows the host to configure and set up the in-circuit debugger while the CPU continues to execute the normal program. Debug mode can be invoked from Background mode.
- Debug mode allows the debug engine to take control of the CPU, providing read-write access to internal registers and memory, and single-step trace operation.

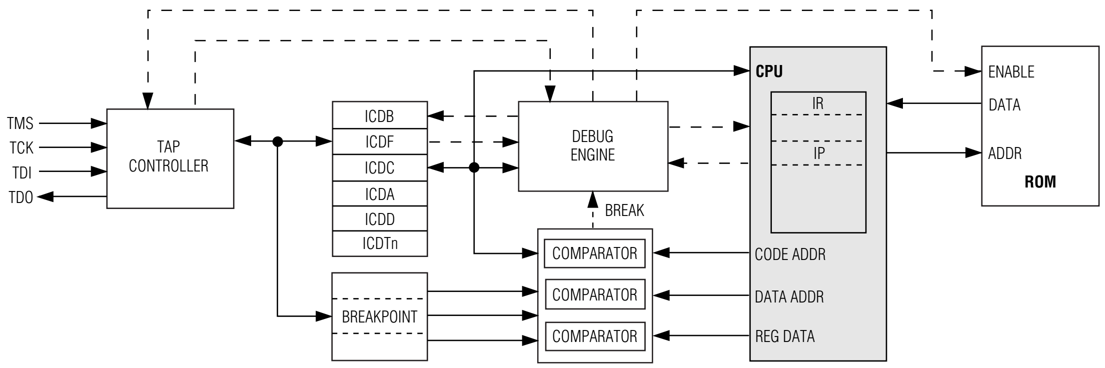
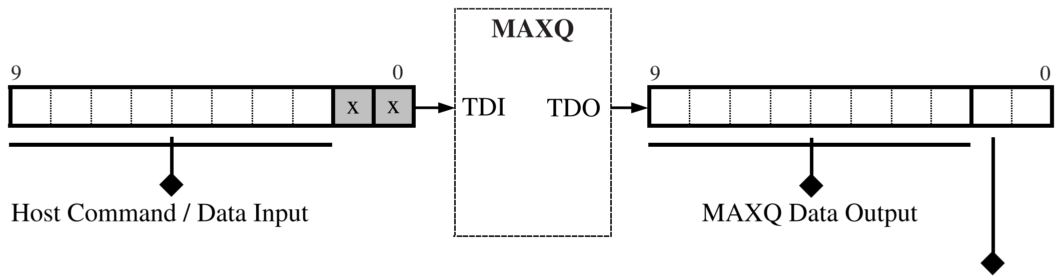


Figure 16-1. In-Circuit Debugger

The embedded hardware debug engine is implemented as a stand-alone hardware block in the MAXQ microcontroller. The debug engine can be enabled for monitoring internal activities and interacting with selected internal registers while the CPU is executing user code. This capability allows the user to employ the embedded debug engine to debug the actual system, in place of the in-circuit emulator that uses external hardware to duplicate operation of the microcontroller outside of the real application environment.

To enable a communication link between the host and the microcontroller debug engine, the Debug instruction (010b) must be loaded into the TAP instruction register using the IR-Scan sequence. Once the instruction is latched in the instruction parallel buffer (IR2:0) and is recognized by the TAP controller in the Update-IR state, the 10-bit data shift register is activated as the communication channel for DR-Scan sequences. The TAP instruction register retains the Debug instruction until a new instruction is shifted via an IR-Scan or the TAP controller returns to the Test-Logic-Reset state.

The host now can transmit and receive serial data through the 10-bit data shift register that exists between the TDI input and TDO output during DR-Scan sequences. All background and debug mode communication (commands, data input/output, and status) occurs via this serial channel. Each 10-bit exchange of data between the host and the MAXQ internal hardware is composed of two status bits and a single byte of command or data. The 10-bit word is always transmitted least significant bit first with the format shown below.



s1:s0	Status/Condition
00	**Non-Debug**. Default condition, Background mode, or debug engine inactive.
01	**Debug Idle**. Debug engine is ready to receive data from the host (command, data).
10	**Debug Busy**. Debug engine is busy without valid data (i.e. ROM code execution, trace operations).
11	**Debug Valid**. Debug engine is busy with valid data

The data byte portion of the 10-bit shift register is interfaced directly to the ICDB parallel register. The ICDB register functions as the holding data register for both transmit and receive operations. On the falling edge of TCK in the Update-DR state, the outgoing data is loaded from the ICDB parallel register to the debug shift register and the incoming shift register data is latched in the ICDB parallel register.

16.1 Background Mode Operation

When the instruction register is loaded with the Debug instruction (IR2:0 = 010b), the host can communicate with the MAXQ microcontroller in a background mode using TAP DR-Scan sequences without disturbing CPU operation. Note, however, that JTAG in-system programming also requires use of the 10-bit debug shift register and, if enabled (SPE, PSS1:0 = 100b), takes precedence over background mode communication. When operating in background mode, the status bits are always cleared to 00b (non-debug), which indicates that the MAXQ microcontroller is ready to receive background mode commands.

The host can perform the following operations from background mode:

- read/write internal breakpoint registers (BP0-BP5)
- read/write internal in-circuit debug registers (ICDC, ICDF, ICDA, ICDD)
- monitor to determine when a breakpoint match has occurred
- directly invoke debug mode

Table 16-1 shows the background mode commands supported by the MAXQ microcontroller. Encodings not listed in this table are not supported in background mode and are treated as no operations.

Table 16-1. Background Mode Commands

OP CODE	COMMAND	OPERATION
0000-0000	No Operation	**No Operation.** Default state for Debug Shift register.
0000-0001	Read ICDC	**Read Control Data from the ICDC.** The contents of the ICDC register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires one follow-on transfer cycle.
0000-0010	Read ICDF	**Read Flags from the ICDF.** The contents of the ICDF register (one byte) are loaded into the Debug Shift Register via the ICDB register for host read. This command requires one follow-on transfer cycle.
0000-0011	Read ICDA	**Read Data from the ICDA.** The contents of the ICDA register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0100	Read ICDD	**Read Data from the ICDD.** The contents of the ICDD register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0101	Read BP0	**Read Data from the BP0.** The contents of the BP0 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0110	Read BP1	**Read Data from the BP1.** The contents of the BP1 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0111	Read BP2	**Read Data from the BP2.** The contents of the BP2 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1000	Read BP3	**Read Data from the BP3.** The contents of the BP3 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1001	Read BP4	**Read Data from the BP4.** The contents of the BP4 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1010	Read BP5	**Read Data from the BP5.** The contents of the BP5 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0001-0001	Write ICDC	**Write Control Data to the ICDC.** The contents of ICDB are loaded into the ICDC register by the debug engine at the end of the data transfer cycle.
0001-0011	Write ICDA	**Write Data to the ICDA.** The contents of ICDB are loaded into the ICDA register by the debug engine at the end of the data transfer cycles. Data is transferred with the least significant byte first.
0001-0100	Write ICDD	**Write Data to the ICDD.** The contents of ICDB are loaded into the ICDD register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0101	Write BP0	**Write Data to the BP0.** The contents of ICDB are loaded into the BP0 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0110	Write BP1	**Write Data to the BP1.** The contents of ICDB are loaded into the BP1 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0111	Write BP2	**Write Data to the BP2.** The contents of ICDB are loaded into the BP2 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1000	Write BP3	**Write Data to the BP3.** The contents of ICDB are loaded into the BP3 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1001	Write BP4	**Write Data to the BP4.** The contents of ICDB are loaded into the BP4 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1010	Write BP5	**Write Data to the BP5.** The contents of ICDB are loaded into the BP5 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1111	Debug	**Debug Command.** This command forces the debug engine into debug mode and halts the CPU operation at the completion of the current instruction after the debug engine recognizes the debug command.

16.1.1 Breakpoint Registers

The MAXQ microcontroller incorporates six breakpoint registers (BP0-BP5) that are configurable by the host for establishing different types of breakpoint mechanisms. The first four breakpoint registers (BP0-BP3) are 16-bit registers that are configurable as program memory address breakpoints. When enabled, the debug engine will force a break when a match between the breakpoint register and the program memory execution address occurs. The final two 16-bit breakpoint registers (BP4, BP5) are configurable in one of two possible capacities. They may be configured as data memory address breakpoints or may be configured to support register access breakpoints. In either case, if breakpoints are enabled and the defined breakpoint match occurs, the debug engine will generate a break condition. The six breakpoint registers are documented below.

16.1.1.1 Breakpoint 0 Register (BP0)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	BP0.15	BP0.14	BP0.13	BP0.12	BP0.11	BP0.10	BP0.9	BP0.8
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	BP0.7	BP0.6	BP0.5	BP0.4	BP0.3	BP0.2	BP0.1	BP0.0
Reset	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	S

s = special

Bits 15 to 0: Breakpoint 0 (BP0.[15:0]). This register is accessible only via background mode read/write commands. Breakpoint registers BP0, BP1, BP2, and BP3 serve as program memory address breakpoints. When DME bit is set in background mode, the debug engine monitors the program-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 control of the CPU and enter debug mode.

16.1.1.2 Breakpoint 1 Register (BP1)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	BP1.15	BP1.14	BP1.13	BP1.12	BP1.11	BP1.10	BP1.9	BP1.8
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	BP1.7	BP1.6	BP1.5	BP1.4	BP1.3	BP1.2	BP1.1	BP1.0
Reset	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	S

s = special

Bits 15 to 0: Breakpoint 1 (BP1.[15:0]). This register is accessible only via background mode read/write commands. Breakpoint registers BP0, BP1, BP2, and BP3 serve as program memory address breakpoints. When DME bit is set in background mode, the debug engine monitors the program-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 control of the CPU and enter debug mode.

16.1.1.3 Breakpoint 2 Register (BP2)

Bit #	15	14	13	12	11	10	9	8
Name	BP2.15	BP2.14	BP2.13	BP2.12	BP2.11	BP2.10	BP2.9	BP2.8
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	BP2.7	BP2.6	BP2.5	BP2.4	BP2.3	BP2.2	BP2.1	BP2.0
Reset	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	S

s = special

Bits 15 to 0: Breakpoint 2 (BP2.[15:0]). This register is accessible only via background mode read/write commands. Breakpoint registers BP0, BP1, BP2, and BP3 serve as program memory address breakpoints. When DME bit is set in background mode, the debug engine monitors the program-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 control of the CPU and enter debug mode.

16.1.1.4 Breakpoint 3 Register (BP3)

Bit #	15	14	13	12	11	10	9	8
Name	BP3.15	BP3.14	BP3.13	BP3.12	BP3.11	BP3.10	BP3.9	BP3.8
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	BP3.7	BP3.6	BP3.5	BP3.4	BP3.3	BP3.2	BP3.1	BP3.0
Reset	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	S

s = special

Bits 15 to 0: Breakpoint 3 (BP3.[15:0]). This register is accessible only via background mode read/write commands. Breakpoint registers BP0, BP1, BP2, and BP3 serve as program memory address breakpoints. When DME bit is set in background mode, the debug engine monitors the program-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 control of the CPU and enter debug mode.

16.1.1.5 Breakpoint 4 Register (BP4) (REGE = 0)

Bit #	15	14	13	12	11	10	9	8
Name (REGE = 0)	BP4.15	BP4.14	BP4.13	BP4.12	BP4.11	BP4.10	BP4.9	BP4.8
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)	BP4.7	BP4.6	BP4.5	BP4.4	BP4.3	BP4.2	BP4.1	BP4.0
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 4 (BP4.[15:0]). This register is accessible only via background mode read/write commands.

When (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.

16.1.1.6 Breakpoint 4 Register (BP4) (REGE = 1)

Bit #	15	14	13	12	11	10	9	8
Name (REGE = 1)	—	—	—	—	—	—	—	BP4.8
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)	BP4.7	BP4.6	BP4.5	BP4.4	BP4.3	BP4.2	BP4.1	BP4.0
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 4 (BP4.[8:0]). This register is accessible only via background mode read/write commands.

When (REGE = 1): This register serves as one of the two register breakpoints. A break occurs when the destination register address for the executed instruction matches with the specified module and index.

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.

16.2.1 Debug Mode Commands

The debug engine sets the data shift register status bits to 01b (debug-idle) to indicate that it is ready to accept debug commands from the host.

The host can perform the following operations from debug mode:

- read register map
- read program stack
- read/write register
- read/write data memory
- single step of CPU (trace)
- return to background mode
- unlock password

The only operations directly controlled by the debug engine are single step and return. All other operations are assisted by debug service routines contained in the Utility ROM. These operations require that multiple bytes be transmitted and/or received by the host, however each operation always begins with host transmission of a command byte. This command byte is decoded by the debug engine in order to determine the quantity, sequence, and destination for follow-on bytes received from the host. Even though there is no timing window specified for receiving the complete command and follow-on data, the debug engine must receive the correct number of bytes for a particular command before executing that command. If command and follow-on data are transmitted out of byte order or proper sequence, the only way to resolve this situation is to disable the debug engine by changing the instruction register (IR2:0) and reloading the Debug instruction. Once the debug engine has received the proper number of command and follow-on bytes for a given ROM assisted operation, it will respond with the following actions:

- update the Command bits (CMD3:0) in the ICDC register to reflect the host request,
- enable the ROM if it is not been enabled,
- force a jump to ROM address x8010h, and
- set the data shift register status bits to 10b (debug-busy)

The ROM code performs a read to the ICDC register CMD3:0 bits to determine its course of action. Some commands can be processed by the ROM without receiving data from the host beyond the initially supplied follow-on bytes, while others (e.g., Unlock Password) require additional data from the host. Some commands need only to provide an indication of completion to the host, while others (Read register map) need to supply multiple bytes of output data. To accomplish data flow control between the host and ROM, the status bits should be used by the host to assess when the ROM is ready for additional data and/or when the ROM is providing valid data output. Internally, the ROM can ascertain when new data is available or when it may output the next data byte via the TXC flag. The TXC flag is an important indicator between the debug engine and the Utility ROM debug routines. The Utility ROM firmware sets the TXC flag to 1 to indicate that valid data has been loaded to the ICDB register. The debug engine clears the TXC flag to 0 to indicate completion of a data shift cycle, thus allowing the ROM to continue execution of a requested task that is still in progress. The Utility ROM signals that it has completed a requested task by setting the ROM Operation Done (ROD) bit of the SC register to logic 1. The ROD bit is reset by the debug engine when it recognizes the done condition.

Table 16-2 shows the debug mode commands supported by the MAXQ microcontroller. Note that background mode commands are supported inside debug mode, however the documentation of these commands can be found in the Background mode section of the document. Encodings not listed in this table are not supported in debug mode and are treated as no operations.

Table 16-2. Debug Mode Commands

OP CODE	COMMAND	OPERATION
0010-0000	No Operation	No Operation
0010-0001	Read Register Map	Read Data from Internal Registers. This command forces the debug engine to update the CMD3:0 bits in the ICDC to 0001b and perform a jump to ROM code at x8010h. The ROM debug service routine will load register data to ICDB for host capture/read, starting at the lowest register location in module 0, one byte at a time in a successive order until all internal registers are read and output to the host.
0010-0010	Read Data Memory	Read Data from Data Memory. This command requires four follow-on transfer cycles, two for the starting address and two for the word read count, starting with the LSB address and ending with the MSB read count. The address is moved to the ICDA register and the word read count is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:0 bits to 0010b and performs a jump to ROM code at x8010h. The ROM debug service routine will load ICDB from data memory according to address and count information provided by the host.
0010-0011	Read Program Stack	Read Data from Program Stack. This command requires four follow-on transfer cycles, two for the starting address and two for the read count, starting with the LSB address and ending with the MSB read count. The address is moved to the ICDA register and the read count is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:0 bits to 0011b and performs a jump to ROM code at x8010h. The ROM Debug service routine will pop data out from the stack according to the information received in the ICDA and ICDD register. The stack pointer is pre-decremented for each pop operation.
0010-0100	Write Register	Write Data to a Selected Register. This command requires four follow-on transfer cycles, two for the register address and two for the data, starting with the LSB address and ending with the MSB data. The address is moved to the ICDA register and the data is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:0 bits to 0100b and performs a jump to ROM code at x8010h. The ROM Debug service routine will update the select register according to the information received in the ICDA and ICDD registers.
0010-0101	Write Data Memory	Write Data to a Selected Data Memory Location. This command requires four follow-on transfer cycles, two for the memory address and two for the data, starting with the LSB address and ending with the MSB data. The address is moved to the ICDA register and the data is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:0 bits to 0101b and performs a jump to ROM code at x8010h. The ROM Debug service routine will update the selected data memory location according to the information received in the IICDA and ICDD registers.
0010-0110	Trace	Trace Command. This command allows single stepping the CPU and requires no follow-on transfer cycle. The trace operation is a 'debug mode exit, one cycle CPU execution, debug mode entry' sequence.
0010-0111	Return	Return Command. This command terminates the debug mode and returns the debug engine to background mode. This allows the CPU to resume its normal operation at the point where it has been last interrupted.
0010-1000	Unlock Password	Unlock the Password Lock. This command requires 32 follow-on transfer cycles each containing a byte value to be compared with the program memory password for the purpose of clearing the PWL bit and granting access to protected debug and loader functions. When this command is received, the debug engine updates the CMD3:0 bit to 1000b and performs a jump to ROM code at x8010h. Data is loaded to the ICDB register when each byte of data is received, beginning with the LSB of the least significant word first and end with the MSB of the most significant word.
0010-1001	Read Register	Read from a Selected Internal Register. This command requires two follow-on transfer cycles, starting with the LSB address and ending with the MSB address. The address is moved to ICDA register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:0 bits to 1001b and performs a jump to ROM code at x8010h. The ROM Debug service routine will always assume a 16-bit register length and return the requested data LSB first.

16.2.2 Read Register Map Command Host-ROM Interaction

A read register map command reads out data contents for all implemented system and peripheral registers. The host does not specify a target register but instead should expect register data output in successive order, starting with the lowest order register in register module 0. Data is loaded by the ROM to the 8-bit ICDB register and is output one byte per transfer cycle. Thus, for a 16-bit register, two transfer cycles are necessary. The host initiates each transfer cycle to shift out the data bytes and will find valid data output tagged with a debug-valid (status = 11b). At the end of each transfer cycle, the debug engine clears the TXC flag to signal the ROM service routine that another byte may be loaded to ICDB. The ROM service routine sets the TXC flag each time after loading data to the ICDB register. This process is repeated until all registers have been read and output to the host. The host system recognizes the completion of the register read when the status debug-idle is presented. This indicates that the debug engine is ready for another operation.

16.2.3 Single-Step Operation (Trace)

The debug engine supports single step operation in debug mode by executing a trace command from the host. The debug engine allows the CPU to return to its normal program execution for one cycle and then forces a debug mode re-entry:

1) Set status to 10b (debug-busy).
2) Pop the return address from the stack.
3) Set the IGE bit to logic 1 if debug mode was activated when IGE = 1.
4) Supply the CPU with an instruction addressed by the return address.
5) Stall the CPU at the end of the instruction execution.
6) Block the next instruction fetch from program memory.
7) Push the return address onto the stack.
8) Set the contents of IP to x8010h.
9) Clear the IGE bit to 0 to disable the interrupt handler.
10) Halt CPU operation.
11) Set the status to debug-idle.

Note that the trace operation uses a return address from the stack as a legitimate address for program fetching. The host must maintain consistency of program flow during the debug process. The Instruction Pointer is automatically incremented after each trace operation, thus a new return address will be pushed onto the stack before returning the control to the debug engine. Also, note that the interrupt handler is an essential part of the CPU and a pending interrupt could be granted during single step operation since the IGE bit state present on debug mode entry is restored for the single step.

16.2.4 Return

To terminate the debug mode and return the debug engine to background mode, the host must issue a Return command to the debug engine. This command causes the following actions:

1) Pop the return address from the stack.
2) Set the IGE bit to logic 1 if debug mode was activated when IGE = 1.
3) Supply the CPU with an instruction addressed by the return address.
4) Allow the CPU to execute the normal user program.
5) Set the status to 00b (non-debug).

To prevent a possible endless-breakpoint matching loop, no break occurs for a breakpoint match on the first instruction after returning from debug mode to background mode. Returning to background mode also enables all internal timer functions.

16.2.5 Debug Mode Special Considerations

The following are special considerations when using Debug Mode.

- The debug engine cannot be operated reliably when the CPU is configured in the Power Management Mode (divide-by-256 system clock mode). To allow for proper execution of debug mode commands when invoked during PMM, the Switchback enable (SWB) bit should be configured to a logic 1. With SWB = 1, entering active debug mode (whether by breakpoint match or issuance of the debug command) forces a switchback to the divide-by-1 system clock mode and allow the debug engine to function correctly. This allows user code to configure breakpoints that occur inside PMM, thus providing reliable use of debug commands. However, it does not allow a good means for re-entering PMM.

- Special caution should be exercised when using the Write Register command on register bits that globally affect system operation (e.g., IGE, STOP). If the write register command is used to invoke stop mode (setting STOP = 1), the $\overline{\text{RST}}$ pin may be asserted to reset the debug engine and return to the background mode of operation.
- Single stepping ('Trace') through any IGE bit change operation results in the debug engine overriding the bit change since it retains the IGE bit setting captured when active debug mode was entered.
- Single stepping ('Trace') into an operation that sets STOP = 1 when IGE = 1 effectively allows enabled interrupts normally capable of causing exit from stop mode to do so.
- Single stepping ('Trace') through any memory read instruction that reads from the utility ROM (such as 'move Acc,' @DP[0] with DP[0] set to 8000h) will cause the memory read to return an incorrect value.
- Single stepping ('Trace') cannot be used when executing code from the utility ROM.
- Data memory allocation is important during system development if in-circuit debug is planned. The top 32-byte memory location may be used by the debug service routine during debug mode. The data contents in these locations may be altered and cannot be recovered.
- One available stack location is needed for debug mode. If the stack is full when entering debug mode, the oldest data in the stack will be overwritten.
- The crystal warmup counter is the only counter not disabled when active debug mode is entered. If the crystal warmup counter completes while in active debug mode, a glitchless switch will be made to selected clock source (which was being counted). It is important that the user recognize that this action will occur since the TAP clock should be run no faster than 1/8 the system clock frequency.
- Any signal sampling that relies upon the internal system clock (e.g., counter inputs) can be unreliable since the system clock is turned off inside active debug mode between debug mode commands.
- Power Management Mode cannot be invoked in the first instruction executed when returning from active debug mode. The PMME bit will not be set if such an attempt is made.

16.3 In-Circuit Debug Peripheral Registers

16.3.1 In-Circuit Debug Temp 0 Register (ICDT0)

Bit #	15	14	13	12	11	10	9	8
Name	ICDT0.15	ICDT0.14	ICDT0.13	ICDT0.12	ICDT0.11	ICDT0.10	ICDT0.9	ICDT0.8
Reset	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s

Bit #	7	6	5	4	3	2	1	0
Name	ICDT0.7	ICDT0.6	ICDT0.5	ICDT0.4	ICDT0.3	ICDT0.2	ICDT0.1	ICDT0.0
Reset	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s

s = special

Bits 15 to 0: In-Circuit Debug Temp 0 (ICDT0.[15:0]). This register is read/write accessible by the CPU only in background mode or debug mode. This register is intended for use by the utility ROM routines as temporary storage to save registers that might otherwise have to be placed in the stack.

16.3.2 In-Circuit Debug Temp 1 Register (ICDT1)

Bit #	15	14	13	12	11	10	9	8
Name	ICDT1.15	ICDT1.14	ICDT1.13	ICDT1.12	ICDT1.11	ICDT1.10	ICDT1.9	ICDT1.8
Reset	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s

Bit #	7	6	5	4	3	2	1	0
Name	ICDT1.7	ICDT1.6	ICDT1.5	ICDT1.4	ICDT1.3	ICDT1.2	ICDT1.1	ICDT1.0
Reset	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s

s = special

Bits 15 to 0: In-Circuit Debug Temp 1 (ICDT1.[15:0]). This register is read/write accessible by the CPU only in background mode or debug mode. This register is intended for use by the utility ROM routines as temporary storage to save registers that might otherwise have to be placed in the stack.

16.3.3 In-Circuit Debug Control Register (ICDC)

Bit #	7	6	5	4	3	2	1	0
Name	DME	—	REGE	—	CMD3	CMD2	CMD1	CMD0
Reset	0	0	0	0	0	0	0	0
Access	rs	r	rs	r	rs	rs	rs	rs

r = read, s = special

Bit 7: Debug Mode Enable (DME). When this bit is cleared to 0, background mode commands may be executed, but breakpoints are disabled. When this bit is set to 1, breakpoints are enabled while background mode commands still may be entered. This bit may only be set or cleared from background debug mode. This bit has no meaning for the ROM code.

Bits 6 and 4: Reserved

Bit 5: Break-On Register Enable (REGE). The REGE bit is used to enable the break-on register function. When REGE bit is set to 1, BP4 and BP5 are used as register breakpoints. A break occurs when the content of BP4 is matched with the destination address of the current instruction. For BP5, a break occurs only on a selected data pattern for a selected destination register addressed by BP5. The data pattern is determined by the contents in the ICDA and ICDD register. The REGE bit alone does not enable register breakpoints, but simply changes the manner in which BP4, BP5 are used. The DME bit still must be set to a logic 1 for any breakpoint to occur. This bit has no meaning for the ROM code.

Bits 3 to 0: Command Bits (CMD[3:0]). These bits reflect the current host command in debug mode. These bits are set by the debug engine and allow the ROM code to determine the course of action.

CMD[3:0]	ACTION
0000	No Operation
0001	Read Register Map
0010	Read Data Memory
0011	Read Stack Memory
0100	Write Register
0101	Write Data Memory
1000	Unlock Password
1001	Read Register
Other	Reserved

16.3.4 In-Circuit Debug Flag Register (ICDF)

Bit #	7	6	5	4	3	2	1	0
Name	—	—	—	—	PSS1	PSS0	SPE	TXC
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw

r = read, w = write

Bits 7 to 4: Reserved

Bits 3 to 2: Programming Source Select Bits 1:0 (PSS[1:0]). These bits are used to select a programming interface during In-System programming when SPE is set to logic 1. Otherwise, the logic values of these bits have no meaning. The logical states of these bits, when read by the CPU, reflect the logical-OR of the PSS bits that are write accessible by the CPU and those in the System Programming Buffer (SPB) register of the TAP module (which are accessible via JTAG). These bits are read/write accessible for the CPU and are cleared to 0 by a power-on reset or Test-Logic-Reset. CPU writes to the PSS bits result in clearing of the JTAG PSS[1:0] bits.

PSS1	PSS0	SOURCE SELECTION
0	0	JTAG
0	1	UART
1	0	SPI
1	1	Reserved

Bit 1: System Program Enable (SPE). The SPE bit is used for in-system programming support and its logical state, when by the CPU, always reflects the logical-OR of the SPE bit that is write accessible by the CPU and SPR bit of the System Programming Buffer (SPB) Register in the TAP Module (which is accessible via JTAG.) The logical state of this bit determines the program flow after a reset. When it is set to logic 1, in-system programming is executed by the Utility ROM. When it is cleared to 0, execution is transferred to user code. This but allows read/write access by the SPU and is cleared to 0 only on a power-on reset or Test-Logic-Reset. The JTAG SPE bit is cleared by hardware when the ROD bit is set. CPU writes to the SPE bit result in clearing the JTAG PSS[1:0] bits.

Bit 0: Serial Transfer Complete (TXC). This bit is set by hardware at the end of a transfer cycle at the TAP communication link. The TXC bit helps the debug engine to recognize host requests, either command or data. This bit is normally set by ROM code to signify or request the sending or receiving of data. The TXC bit is cleared by the debug engine once set. CPU writes to the TXC bit results in clearing of the JTAG PSS[1:0] bits.

16.3.5 In-Circuit Debug Buffer Register (ICDB)

Bit #	7	6	5	4	3	2	1	0
Name	ICDB.7	ICDB.6	ICDB.5	ICDB.4	ICDB.3	ICDB.2	ICDB.1	ICDB.0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bits 7 to 0: In-Circuit Debug Buffer Register (ICDB.[7:0]). This register serves as the parallel holding buffer for the debug shift register of the TAP. Data is read from or written to ICDB for serial communication between the debug routines and the external host.

16.3.6 In-Circuit Debug Data Register (ICDD)

Bit #	15	14	13	12	11	10	9	8
Name	ICDD.15	ICDD.14	ICDD.13	ICDD.12	ICDD.11	ICDD.10	ICDD.9	ICDD.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	ICDD.7	ICDD.6	ICDD.5	ICDD.4	ICDD.3	ICDD.2	ICDD.1	ICDD.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read

Bits 15 to 0: In-Circuit Debug Data (ICDD.[15:0]). This register is used by the debug engine to store data/read count so that ROM code can view that information. This register is also used by the debug engine as a data register for content matching when BP5 is used as a register breakpoint. In this case, only data bits in this register with their corresponding mask bits in the ICDA register set will be compared with the updated destination data to determine if a break should be generated.

16.3.7 In-Circuit Debug Address Register (ICDA)

Bit #	15	14	13	12	11	10	9	8
Name	ICDA.15	ICDA.14	ICDA.13	ICDA.12	ICDA.11	ICDA.10	ICDA.9	ICDA.8
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

Bit #	7	6	5	4	3	2	1	0
Name	ICDA.7	ICDA.6	ICDA.5	ICDA.4	ICDA.3	ICDA.2	ICDA.1	ICDA.0
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r

r = read

Bits 15 to 0: In-Circuit Debug Address (ICDA.[15:0]). This register is used by the debug engine to addresses so that ROM code can view that information. This register is also used by the debug engine as a mask register to mask out don't care bits in the ICDD register when BP5 is used as a register breakpoint. When a bit in this register is set to 1, the corresponding bit location in the ICDD register will be compared to the data being written to the destination register to determine if a break should be generated. When a bit in this register is cleared, the corresponding bit in the ICDD register becomes a don't care and is not compared against the data being written. When all bits in this register are cleared, any updated data pattern will cause a break when the BP5 register matches the destination register address of the current instruction.

SECTION 17: IN-SYSTEM PROGRAMMING (JTAG)

This section contains the following information:

SECTION 17: IN-SYSTEM PROGRAMMING (JTAG)

Internal nonvolatile memory of MAXQ microcontrollers can be initialized via Bootstrap Loader mode. To enable the Bootstrap loader and establish a desired communication channel, the System Programming instruction (100b) must be loaded into the TAP instruction register using the IR-Scan sequence. Once the instruction is latched in the instruction parallel buffer (IR2:0) and is recognized by the TAP controller in the Update-IR state, a 3-bit data shift register is activated as the communication channel for DR-Scan sequences. The TAP retains the System Programming instruction until a new instruction is shifted in or the TAP controller returns to the Test-Logic-Reset state. This 3-bit shift register formed between the TDI and TDO pins is directly interfaced to the 3-bit Serial Programming Buffer (SPB). The System Programming Buffer (SPB) contains three bits with the following functions:

- SPB.0: System Programming Enable (SPE). Setting this bit to logic 1 denotes that system programming is desired upon exiting reset. When it is cleared to logic 0, no system programming is needed. The logic state of SPE is examined by the reset vector in the Utility ROM to determine the program flow after a reset. When SPE = 1, the Bootstrap Loader selected by the PSS1:0 bits are activated to perform a Bootstrap Loader function. When SPE = 0, the Utility ROM transfers execution control to the normal user program.
- SPB.2:1: Programming Source Select (PSS1:PSS0). These bits allow the host to select programming interface sources. The PSS bits have no functions when the SPE bit is cleared.

PSS1	PSS0	PROGRAMMING SOURCE
0	0	JTAG
0	1	UART
1	0	SPI
1	1	Reserved

The DR-Scan sequence is used to configure the SPB bits. The data content of the SPB register is reflected in the ICDF register and allows read/write access by the CPU. These bits are cleared by power-on reset or Test-Logic-Reset of the TAP controller.

17.1 JTAG Bootloader Operation

Devices that support a JTAG bootloader have the benefit of using the same status bit handshaking hardware as is used for in-circuit debugging. When the SPE bit of the System Programming Buffer (SPB) is set to 1 and JTAG is selected as the programming source (PSS1:0 = 00b), the background and active debug mode state machines are disabled. Once the host loads the Debug instruction into the TAP instruction register (IR2:0), the 10-bit shift register interface to ICDB and the status bits becomes available for host-to-ROM bootloader communication. The status bits should be interpreted as follows for JTAG bootloader operation:

BITS 1:0	STATUS	CONDITION
00	Reserved	Invalid condition.
01	Reserved	Invalid condition
10	Loader-Busy	ROM Loader is busy executing code or processing the current command.
11	Loader-Valid	ROM Loader is supplying valid output data to the host in current shift operation.

When the using the JTAG bootloader option (SPE = 1, PSS1:0 = 00b), the sole purpose of the debug hardware is to simultaneously transfer the data byte shifted in from the host into the ICDB register and transfer the contents of an internal holding register (loaded by ROM code writes of ICDB) into the shift register for output to the host. This transfer takes place on the falling edge of TCK at the Update-DR state. The debug hardware additionally clears the TXC bit at this point in the state diagram. The ROM loader code controls the status bit output to the host by asserting TXC = 1 when it has valid data to be shifted out. The ROM code may flexibly implement whatever communication protocol and command set it wishes within the data byte portion of the shifted 10-bit word.

17.2 Password-Protected Access

Some applications require preventative measures to protect against simple access and viewing of program code memory. To address this need for code protection, any MAXQ microcontroller equipped with a Utility ROM that permits in-system programming, in-application programming, or in-circuit debugging grants full access to those utilities only after a password has been supplied. The password is defined as the 16 words of physical program memory at addresses x0010h to x001Fh. Note that using these memory locations as a password does not exclude their usage for general code space if a unique password is not needed.

A single Password Lock bit (PWL) is implemented in the SC register. When the PWL is set to 1, a password is required to access the in-circuit debug and in-system programming ROM routines that allow reading or writing of internal memory. When PWL is cleared to 0, these utilities are fully accessible through the Utility ROM without a password.

The PWL bit defaults to 1 by a power-on reset. To access the ROM utilities, a correct password is needed; otherwise access to the ROM utilities is denied. Once the correct password has been supplied by the user, the ROM clears the password lock. The PWL remains clear until one of the following occurs:

- a power-on reset, or
- set to logic 1 by user software.

For devices with ROM program memory, the end user supplies the ROM code, thus the user always knows the password if needed. It is expected that the password will rarely be needed since the utility of memory programming and/or in-circuit debug to the end user will be minimal once the decision has been made to freeze the code in program ROM.

For devices with reprogrammable nonvolatile memory, the password is always known for a fully erased device since the unprogrammed state of these memories will be fixed. Once the memory has been programmed, a password is established and can be used for access protection. The Utility ROM code denies access to the protected routines when PWL indicates a locked state.

17.2.1 Entering Password

A password can be entered in one of two ways:

- Via the in-system programming interface established by the PSS1:PSS0 bits when SPE bit is set to logic 1; the ROM Bootstrap Loader dictates the protocol for entering the password over the specified serial communication interface
- Via the TAP interface directly by issuing the Unlock Password debug mode command. The Unlock Password command requires 32 follow-on transfer cycles each containing a byte value to be compared with the program memory password.

SECTION 18: MAXQ FAMILY INSTRUCTION SET SUMMARY

This section contains the following information:

LIST OF TABLES

SECTION 18: MAXQ FAMILY INSTRUCTION SET SUMMARY

Table 18-1. Instruction Set Summary

	MNEMONIC	DESCRIPTION	16-BIT INSTRUCTION WORD	STATUS BITS AFFECTED	AP INC/DEC	NOTES
LOGICAL OPERATIONS	AND src	Acc ⟵ Acc AND src	f001 1010 ssss ssss	S, Z	Y	1
	OR src	Acc ⟵ Acc OR src	f010 1010 ssss ssss	S, Z	Y	1
	XOR src	Acc ⟵ Acc XOR src	f011 1010 ssss ssss	S, Z	Y	1
	CPL	Acc ⟵ ~Acc	1000 1010 0001 1010	S, Z	Y	
	NEG	Acc ⟵ ~Acc + 1	1000 1010 1001 1010	S, Z	Y	
	SLA	Shift Acc left arithmetically	1000 1010 0010 1010	C, S, Z	Y	
	SLA2	Shift Acc left arithmetically twice	1000 1010 0011 1010	C, S, Z	Y	
	SLA4	Shift Acc left arithmetically four times	1000 1010 0110 1010	C, S, Z	Y	
	RL	Rotate Acc left (w/o C)	1000 1010 0100 1010	S	Y	
	RLC	Rotate Acc left (through C)	1000 1010 0101 1010	C, S, Z	Y	
	SRA	Shift Acc right arithmetically	1000 1010 1111 1010	C, Z	Y	
	SRA2	Shift Acc right arithmetically twice	1000 1010 1110 1010	C, Z	Y	
	SRA4	Shift Acc right arithmetically four times	1000 1010 1011 1010	C, Z	Y	
	SR	Shift Acc right (0 ⟶ msbit)	1000 1010 1010 1010	C, S, Z	Y	
	RR	Rotate Acc right (w/o C)	1000 1010 1100 1010	S	Y	
	RRC	Rotate Acc right (though C)	1000 1010 1101 1010	C, S, Z	Y	
BIT OPERATIONS	MOVE C, Acc.<b>	C ⟵ Acc.<b>	1110 1010 bbbb 1010	C		
	MOVE C, #0	C ⟵ 0	1101 1010 0000 1010	C		
	MOVE C, #1	C ⟵ 1	1101 1010 0001 1010	C		
	CPL C	C ⟵ ~C	1101 1010 0010 1010	C		
	MOVE Acc.<b>, C	Acc.<b> ⟵ C	1111 1010 bbbb 1010	S, Z		
	AND Acc.<b>	C ⟵ C AND Acc.<b>	1001 1010 bbbb 1010	C		
	OR Acc.<b>	C ⟵ C OR Acc.<b>	1010 1010 bbbb 1010	C		
	XOR Acc.<b>	C ⟵ C XOR Acc.<b>	1011 1010 bbbb 1010	C		
	MOVE dst.<b>, #1	dst.<b> ⟵ 1	1ddd dddd 1bbb 0111	C, S, E, Z		2
	MOVE dst.<b>, #0	dst.<b> ⟵ 0	1ddd dddd 0bbb 0111	C, S, E, Z		2
	MOVE C, src.<b>	C ⟵ src.<b>	fbbb 0111 ssss ssss	C		
MATH	ADD src	Acc ⟵ Acc + src	f100 1010 ssss ssss	C, S, Z, OV	Y	1
	ADDC src	Acc ⟵ Acc + (src + C)	f110 1010 ssss ssss	C, S, Z, OV	Y	1
	SUB src	Acc ⟵ Acc – src	f101 1010 ssss ssss	C, S, Z, OV	Y	1
	SUBB src	Acc ⟵ Acc – (src + C)	f111 1010 ssss ssss	C, S, Z, OV	Y	1

Table 18-1. Instruction Set Summary (continued)

	MNEMONIC	DESCRIPTION	16-BIT INSTRUCTION WORD	STATUS BITS AFFECTED	AP INC/DEC	NOTES
BRANCHING	{L/S}JUMP src	IP ⟵ IP + src or src	f000 1100 ssss ssss			6
	{L/S}JUMP C, src	If C=1, IP ⟵ (IP + src) or src	f010 1100 ssss ssss			6
	{L/S}JUMP NC, src	If C=0, IP ⟵ (IP + src) or src	f110 1100 ssss ssss			6
	{L/S}JUMP Z, src	If Z=1, IP ⟵ (IP + src) or src	f001 1100 ssss ssss			6
	{L/S}JUMP NZ, src	If Z=0, IP ⟵ (IP + src) or src	f101 1100 ssss ssss			6
	{L/S}JUMP E, src	If E=1, IP ⟵ (IP + src) or src	0011 1100 ssss ssss			6
	{L/S}JUMP NE, src	If E=0, IP ⟵ (IP + src) or src	0111 1100 ssss ssss			6
	{L/S}JUMP S, src	If S=1, IP ⟵ (IP + src) or src	f100 1100 ssss ssss			6
	{L/S}DJNZ LC[n], src	If --LC[n] <> 0, IP⟵ (IP + src) or src	f10n 1101 ssss ssss			6
	{L/S}CALL src	@++SP ⟵ IP+1; IP ⟵ (IP+src) or src	f011 1101 ssss ssss			6,7
	RET	IP ⟵ @SP--	1000 1100 0000 1101			
	RET C	If C=1, IP ⟵ @SP--	1010 1100 0000 1101			
	RET NC	If C=0, IP ⟵ @SP--	1110 1100 0000 1101			
	RET Z	If Z=1, IP ⟵ @SP--	1001 1100 0000 1101			
	RET NZ	If Z=0, IP ⟵ @SP--	1101 1100 0000 1101			
	RET S	If S=1, IP ⟵ @SP--	1100 1100 0000 1101			
	RETI	IP ⟵ @SP-- ; INS⟵ 0	1000 1100 1000 1101			
	RETI C	If C=1, IP ⟵ @SP-- ; INS⟵ 0	1010 1100 1000 1101			
	RETI NC	If C=0, IP ⟵ @SP-- ; INS⟵ 0	1110 1100 1000 1101			
	RETI Z	If Z=1, IP ⟵ @SP-- ; INS⟵ 0	1001 1100 1000 1101			
	RETI NZ	If Z=0, IP ⟵ @SP-- ; INS⟵ 0	1101 1100 1000 1101			
	RETI S	If S=1, IP ⟵ @SP-- ; INS⟵ 0	1100 1100 1000 1101			
DATA TRANSFER	XCH (MAXQ20 only)	Swap Acc bytes	1000 1010 1000 1010	S	Y	
	XCHN	Swap nibbles in each Acc byte	1000 1010 0111 1010	S	Y	
	MOVE dst, src	dst ⟵ src	fddd dddd ssss ssss	C, S, Z, E	(Note 8)	7, 8
	PUSH src	@++SP ⟵ src	f000 1101 ssss ssss			7
	POP dst	dst ⟵ @SP--	1ddd dddd 0000 1101	C, S, Z, E		7
	POPI dst	dst ⟵ @SP-- ; INS ⟵ 0	1ddd dddd 1000 1101	C, S, Z, E		7
	CMP src	E ⟵ (Acc = src)	f111 1000 ssss ssss	E		
	NOP	No operation	1101 1010 0011 1010			

Note 1: The active accumulator (Acc) is not allowed as the src in operations where it is the implicit destination.

Note 2: Only module 8 and modules 0-5 (when implemented for a given product) are supported by these single-cycle bit operations. Potentially affects C or E if PSF register is the destination. Potentially affects S and/or Z if AP or APC is the destination.

Note 3: The terms Acc and A[AP] can be used interchangeably to denote the active accumulator.

Note 4: Any index represented by <b> or found inside [ ] brackets is considered variable, but required.

Note 5: The active accumulator (Acc) is not allowed as the dst if A[AP] is specified as the src.

Note 6: The '{L/S}' prefix is optional.

Note 7: Instructions that attempt to simultaneously push/pop the stack (e.g. PUSH @SP--, PUSH @SPI--, POP @++SP, POPI @++SP) or modify SP in a conflicting manner (e.g., MOVE SP, @SP--) are invalid.

Note 8: Special cases: If 'MOVE APC, Acc' sets the APC.CLR bit, AP will be cleared, overriding any auto-inc/dec/modulo operation specified for AP. If 'MOVE AP, Acc' causes an auto-inc/dec/modulo operation on AP, this overrides the specified data transfer (i.e., Acc will not be transferred to AP).

ADD/ADDC src **Add/Add with Carry**

Description: The ADD instruction sums the active accumulator (Acc or A[AP]) and the specified src data and stores the result back to the active accumulator. The ADDC instruction additionally includes the Carry (C) Status Flag in the summation. For the complete list of src specifiers, reference the MOVE instruction. The MAXQ20 may use the PFX[n] register to supply the high byte of data for 8-bit sources.

Status Flags: C, S, Z, OV

ADD Operation: Acc ← Acc + src

Encoding: 15 … 0

f100	1010	ssss	ssss

MAXQ10 Example(s):

```
                  ;Acc = 45h for each example
ADD A[3]          ; A[3]=0Fh
                  ; →Acc =54h,C=0, Z=0, S=0;
ADD #0C0h         ; → Acc =05h,C=1, Z=0, S=0;
ADD A[4]          ; A[4]=40h
                  ; → Acc = 85h, C=0, Z=0, S=1, OV=1
```

MAXQ20 Example(s):

```
                  ;Acc = 2345h for each example
ADD A[3]          ; A[3]=FF0Fh
                  ; → Acc =2254h,C=1, Z=0, S=0, OV=0
ADD #0C0h         ; → Acc =2405h,C=0, Z=0, S=0, OV=0
ADD A[4]          ; A[4]=C000h
                  ; → Acc = E345h, C=0, Z=0, S=1, OV=0
ADD A[5]          ; A[5]=6789h
                  ; → Acc = 8ACEh, C=0, Z=0, S=1, OV=1
```

ADDC Operation: Acc ← Acc + C + src

Encoding: 15 … 0

f110	1010	ssss	ssss

MAXQ10 Example(s):

```
                  ; Acc = 45h for each example
ADDC A[3]         ; A[3] = BAh, C=1
                  ; → Acc = 00h, C=1, Z=1, S=0
ADDC @DP[0]--     ; @DP[0] = 0Eh, C=1
                  ; → Acc = 54h, C=0, Z=0, S=0
```

MAXQ20
Example(s):

```
                        ; Acc = 2345h for each example
          ADDC A[3]     ; A[3] = DCBAh, C=1
                        ; → Acc = 0000h, C=1, Z=1, S=0, OV=0
          ADDC @DP[0]-- ; @DP[0] = 00EEh, C=1
                        ; → Acc = 2434h, C=0, Z=0, S=0, OV=0
```

Special Notes: The active accumulator (Acc) is not allowed as the src for these operations.

AND src — Logical AND

Description: Performs a logical-AND between the active accumulator (Acc) and the specified src data. For the complete list of src specifiers, reference the MOVE instruction. The MAXQ20 may use the PFX[n] register to supply the high byte of data for 8-bit sources.

Status Flags: S, Z

Operation: Acc ← Acc AND src

Encoding: 15 … 0

f001	1010	ssss	ssss

MAXQ10
Example(s):

```
                        ; Acc = 45h for each example
          AND A[3]      ; A[3]=0Fh
                        ; → Acc = 05h, S=0, Z=0
          AND  #33h     ; → Acc = 01h, S=0, Z=0
```

MAXQ20
Example(s):

```
                        ; Acc = 2345h for each example
          AND A[3]      ; A[3]=0F0Fh
                        ; → Acc = 0305h, S=0, Z=0
          AND  #33h     ; → Acc = 0001h
          AND #2233h    ; generates object code below
                        ; MOVE PFX[0], #22h (smart-prefixing)
                        ; AND #33h
                        ; → Acc = 2201h
          MOVE PFX[0], #0Fh
          AND M0[8]     ; M0[8]=0Fh (assume M0[8] is an 8-bit register)
                        ; → Acc = 0305h
```

Special Notes: The active accumulator (Acc) is not allowed as the src for this operation.

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

Description: Performs a logical-AND 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 AND Acc. <b>

Encoding: 15 0

1001	1010	bbbb	1010

MAXQ10 Example(s):

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

MAXQ20 Example(s):

```
                ; Acc = 2345h, C=1 at start
AND Acc.0       ; Acc.0=1   → C=1
AND Acc.1       ; Acc.1=0   → C=0
AND C, Acc.8    ; Acc.8=1   → C=0
```

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.

{L/S}CALL src **{Long/Short} Call to Subroutine**

Description: Performs a call to the subroutine destination specified by src. The CALL instruction uses an 8-bit immediate src to perform a relative short call (IP +127/-128 words). The CALL instruction uses a 16-bit immediate src to perform an absolute long CALL to the specified 16-bit address. The PFX[0] register is used to supply the high byte of a 16-bit immediate address for the absolute long CALL. Using the optional 'L' prefix (i.e., LCALL) results in an absolute long call and use of the PFX[0] register. Using the optional 'S' prefix (i.e., SCALL) attempts to generate a relative short call, but is flagged by the assembler if the destination is out of range. Specifying an internal register src (no matter whether 8-bit or 16-bit) always produces an absolute CALL to a 16-bit address, thus the 'L' and 'S' prefixes should not be used. The PFX[n] register value is used to supply the high address byte when an 8-bit register src is specified.

Status Flags: None

Operation:

@++SP ← IP + 1	PUSH
IP ← src	Absolute CALL
IP ← IP + src	Relative CALL

Encoding: 15 0

f011	1101	ssss	ssss

Example(s):

```
CALL label1           ; relative call to label1 (must be within IP +127/ -
                      ; 128 address range)
CALL label1           ; absolute call to label1 = 0120h
                      ; MOVE PFX[0], #01h
                      ; CALL #20h.
CALL  DP[0]           ; DP[0] holds 16-bit address of subroutine
CALL  M0[0]           ; assume M0[0] is an 8-bit register
                      ; absolute call to addr16
                      ; high(addr16)=00h   (PFX[0])
                      ; low (addr16)=M0[0]
MOVE PFX[0], #22h     ;
CALL  M0[0]           ; assume M0[0] is an 8-bit register
                      ; high(addr16)=22h   (PFX[0])
                      ; low (addr16)=M0[0]
LCALL label1          ; label=0120h and is relative to this instruction
                      ; absolute call is forced by use of 'L' prefix
                      ; MOVE PFX[0], #01h
                      ; CALL #20h
SCALL label1          ; relative offset for label1 calculated and used
                      ; if label1 is not relative, assembler will generate an error
SCALL #10h            ; relative offset of #10h is used directly by the CALL
```

CMP src — Compare Accumulator

Description: Compare for equality between the active accumulator and the least significant byte of the specified src. The MAXQ20 may use the PFX[n] register to supply the high byte of data for 8-bit sources.

Status Flags: E

Operation: Acc = src: E ← 1

Acc <> src: E ← 0

Encoding: 15 … 0

f111	1000	ssss	ssss

MAXQ10 Example(s):

```
CMP A[1]       ; Acc = 45h, A[1] = 10h, E=0
CMP  #45h      ; Acc = 45h, E=1
CMP DP[0]      ; Acc = 45h, DP[0]=0345h, E=1
```

MAXQ20 Example(s):

```
CMP  #45h      ; Acc = 0145h, E=0
CMP  #145h     ; PFX[0] register used
               ; MOVE PFX[0], #01h (smart-prefixing)
               ; CMP  #45h  E=1
```

CPL — Complement Acc

Description: Performs a logical bitwise complement (1's complement) on the active accumulator (Acc or A[AP]) and returns the result to the active accumulator.

Status Flags: S, Z

Operation: Acc ← ~Acc

Encoding: 15 … 0

1000	1010	0001	1010

MAXQ10 Example(s):

```
               ; Acc = FFh, S=1, Z=0
CPL            ; Acc ← 00h, S=0, Z=1
               ; Acc = 09h, S=0, Z=0
CPL            ; Acc ← F6h, S=1, Z=0
```

MAXQ20 Example(s):

```
               ; Acc = FFFFh, S=1, Z=0
CPL            ; Acc ← 0000h, S=0, Z=1
               ; Acc = 0990h, S=0, Z=0
CPL            ; Acc ← F66Fh, S=1, Z=0
```

CPL C | **Complement Carry Flag**

Description:	Logically complements the Carry (C) Flag.
Status Flags:	C
Operation:	C ← ~C
Encoding:	15 … 0

1101	1010	0010	1010

Example(s):

```
                ; C = 0
CPL C           ; C ← 1
```

{L/S}DJNZ LC[n], src | **Decrement Counter, {Long/Short} Jump Not Zero**

Description: The DJNZ LC[n], src instruction performs a conditional branch based upon the associated Loop Counter (LC[n]) register. The DJNZ LC[n], src instruction decrements the LC[n] loop counter and branches to the address defined by src if the decremented counter has not reached 0000h. Program branches can be relative or absolute depending upon the src specifier and may be qualified by using the 'L' or 'S' prefixes as documented in the JUMP src op code.

Status Flags: None

Operation: LC[n] ← LC[n] -1

LC[n] <> 0: IP ← IP + src (relative) -or- src (absolute)

LC[n] = 0: IP ← IP + 1

Encoding: 15 … 0

f10n	1101	ssss	ssss

Example(s):

```
MOVE LC[1], #10h        ; counter = 10h
Loop:
ADD     @DP[0]++        ; add data memory contents to Acc, post-inc DP[0]
DJNZ  LC[1],  Loop      ; 16 times before falling through
```

{L/S} JUMP src **Unconditional {Long/Short} Jump**

Description: Performs an unconditional jump as determined by the src specifier. The JUMP instruction uses an 8-bit immediate src to perform a relative jump (IP +127/-128 words). The JUMP instruction uses a 16-bit immediate src to perform an absolute JUMP to the specified 16-bit address. The PFX[0] register is used to supply the high byte of a 16-bit immediate address for the absolute JUMP. Using the optional 'L' prefix (i.e., LJUMP) results in an absolute long jump and use of the PFX[0] register. Using the optional 'S' prefix (i.e., SJUMP) attempts to generate a relative short jump, but is flagged by the assembler if the destination is out of range. Specifying an internal register src (no matter whether 8-bit or 16-bit) always produces an absolute JUMP to a 16-bit address, thus the 'L' and 'S' prefixes should not be used. The PFX[n] register value is used to supply the high address byte when an 8-bit register src is specified.

Status Flags: None

Operation:

IP ← src	Absolute JUMP
IP ← IP + src	Relative JUMP

Encoding: 15 … 0

f000	1100	ssss	ssss

Example(s):

```
JUMP label1          ; relative jump to label1 (must be within range
                     ; IP +127/-128 words)
JUMP label1          ; absolute jump to label1= 0400h
                     ; MOVE PFX[0], #04h
                     ; JUMP #00h
JUMP   DP[0]         ; absolute jump to addr16 DP[0]
JUMP M0[0]           ; assume M0[0] is an 8-bit register
                     ; absolute jump to addr16
                     ; high(addr16)=00h   (PFX[0])
                     ; low (addr16)=M0[0]
LJUMP label1         ; label=0120h and is relative to this instruction
                     ; absolute jump is forced by use of 'L' prefix
                     ; MOVE PFX[0], #01h
                     ; JUMP #20h
SJUMP label1         ; relative offset for label1 calculated and used
                     ; if label1 is not relative, assembler will generate an error
SJUMP #10h           ; relative offset of #10h is used directly by the JUMP
```

{L/S}JUMP C/{L/S}JUMP NC, src,
L/S}JUMP Z/{L/S}JUMP NZ, src,
{{L/S}JUMP E/{L/S}JUMP NE, src,
{L/S}JUMP S, src

Conditional {Long/Short} Jump on Status Flag

Description: Performs conditional branching based upon the state of a specific processor status flag. JUMP C results in a branch if the Carry flag is set while JUMP NC branches if the Carry flag is clear. JUMP Z results in a branch if the Zero flag is set while JUMP NZ branches if the Zero flag is clear. JUMP E results in a branch if the Equal flag is set while JUMP NE branches if the Equal flag is clear. JUMP S results in a branch if the Sign flag is set. Program branches can be relative or absolute depending upon the src specifier and may be qualified by using the 'L' or 'S' prefixes as documented in the JUMP src op code. Special src restrictions apply to JUMP E and JUMP NE.

Status Flags: None

JUMP C C=1: IP ← IP + src (relative) -or- src (absolute)

Operation: C=0: IP ← IP + 1

Encoding: 15 0

f010	1100	ssss	ssss

Example(s): JUMP C, label1 ; C=0, branch not taken

JUMP NC C=0: IP ← IP + src (relative) -or- src (absolute)

Operation: C=1: IP ← IP +1

Encoding: 15 0

f110	1100	ssss	ssss

Example(s): JUMP NC, label1 ; C=0, branch taken

JUMP Z Z=1: IP ← IP + src

Operation: Z=0: IP ← IP + 1

Encoding: 15 0

f001	1100	ssss	ssss

Example(s): JUMP Z, label1 ; Z=1, branch taken

JUMP NZ Z=0: IP ← IP + src (relative) -or- src (absolute)

Operation: Z=1: IP ← IP + 1

Encoding: 15 … 0

f101	1100	ssss	ssss

Example(s): JUMP NZ, label1 ; Z=1, branch taken

JUMP E E=1: IP ← IP + src (relative) -or- src (absolute)

Operation: E=0: IP ← IP + 1

Encoding: 15 … 0

0011	1100	ssss	ssss

Example(s): JUMP E, label1 ; E=1, branch taken

Special Notes: The src specifier must be immediate data.

JUMP NE

Operation: E=0: IP ← IP + src (relative) -or- src (absolute)

E=1: IP ← IP + 1

Encoding: 15 … 0

0111	1100	ssss	ssss

Example(s): JUMP NE, label1 ; E=0, branch taken

Special Notes: The src specifier must be immediate data.

JUMP S S=1: IP ← IP + src (relative) -or- src (absolute)

Operation: S=0: IP ← IP + 1

Encoding: 15 … 0

f100	1100	ssss	ssss

Example(s): JUMP S, label1 ; S=0, branch not taken

MOVE dst, src **Move Data**

Description: Moves data from a specified source (src) to a specified destination (dst). A list of defined source, destination specifiers is given in the table below. Also, since src can be either 8-bit (byte) or 16-bit (word) data, the rules governing data transfer are also explained below in the encoding section.

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

C, E (if dst is PSF)

Operation: dst ← src

Encoding: 15 0

fddd	dddd	ssss	ssss

Table 18-2. Source Specifier Codes

src	src Bit Encoding (f sssssssss)	WIDTH (16 or 8)	DESCRIPTION
#k	0 kkkk kkkk	8	kkkkkkkk = Immediate (Literal) Data
MN[n]	1 nnnn 0NNN	8/16	nnnn Selects One of First 16 Registers in Module NNN; where NNN= 0 to 5. Access to Second 16 Using PFX[n].
AP	1 0000 1000	8	Accumulator Pointer
APC	1 0001 1000	8	Accumulator Pointer Control
PSF	1 0100 1000	8	Processor Status Flag Register
IC	1 0101 1000	8	Interrupt and Control Register
IMR	1 0110 1000	8	Interrupt Mask Register
SC	1 1000 1000	8	System Control Register
IIR	1 1011 1000	8	Interrupt Identification Register
CKCN	1 1110 1000	8	Clock Control Register
WDCN	1 1111 1000	8	Watchdog Control Register
A[n]	1 nnnn 1001	8/16	nnnn Selects One of 16 Accumulators
Acc	1 0000 1010	8/16	Active Accumulator = A[AP]. Update AP per APC
A[AP]	1 0001 1010	8/16	Active Accumulator = A[AP]. No change to AP
IP	1 0000 1100	16	Instruction Pointer
@SP--	1 0000 1101	16	16-Bit Word @SP, Post-Decrement SP
SP	1 0001 1101	16	Stack Pointer
IV	1 0010 1101	16	Interrupt Vector
LC[n]	1 011n 1101	16	n Selects 1 of 2 Loop Counter Registers
@SPI--	1 1000 1101	16	16-bit word @SP, Post-Decrement SP, INS=0
@BP[Offs]	1 0000 1110	8/16	Data Memory @BP[Offs]
@BP[Offs++]	1 0001 1110	8/16	Data memory @BP[Offs]; Post Increment OFFS
@BP[Offs--]	1 0010 1110	8/16	Data Memory @BP[Offs]; Post Decrement OFFS
OFFS	1 0011 1110	8	Frame Pointer Offset from Base Pointer (BP)
DPC	1 0100 1110	16	Data Pointer Control Register
GR	1 0101 1110	16	General Register
GRL	1 0110 1110	8	Low Byte of GR Register
BP	1 0111 1110	16	Frame Pointer Base Pointer (BP)
GRS	1 1000 1110	16	Byte-Swapped GR Register
GRH	1 1001 1110	8	High Byte of GR Register
GRXL	1 1010 1110	16	Sign Extended Low Byte of GR Register
FP	1 1011 1110	16	Frame Pointer (BP[Offs])
@DP[n]	1 0n00 1111	8/16	Data Memory @DP[n]
@DP[n]++	1 0n01 1111	8/16	Data Memory @DP[n], Post-Increment DP[n]
@DP[n]--	1 0n10 1111	8/16	Data Memory @DP[n], Post-Decrement DP[n]
DP[n]	1 0n11 1111	16	n Selects 1 of 2 Data Pointers

MOVE dst, *src* (continued) **Move Data**

Table 18-3. Destination Specifier Codes

dst	dst Bit Encoding (ddd dddd)	WIDTH (16 OR 8)	DESCRIPTION
NUL	111 0110	8/16	Null (Virtual) Destination. Intended as a bit bucket to assist software with pointer increments/decrements.
MN[n]	nnn 0NNN	8/16	nnnn Selects One of First 8 Registers in Module NNN; where NNN= 0 to 5. Access to Next 24 Using PFX[n].
AP	000 1000	8	Accumulator Pointer
APC	001 1000	8	Accumulator Pointer Control
PSF	100 1000	8	Processor Status Flag Register
IC	101 1000	8	Interrupt and Control Register
IMR	110 1000	8	Interrupt Mask Register
A[n]	nnn 1001	8/16	nnn Selects 1 of First 8 Accumulators: A[0]..A[7]
Acc	000 1010	8/16	Active Accumulator = A[AP]
PFX[n]	nnn 1011	8	nnn Selects One of 8 Prefix Registers
@++SP	000 1101	16	16-Bit Word @SP, Pre-Increment SP
SP	001 1101	16	Stack Pointer
IV	010 1101	16	Interrupt Vector
LC[n]	11n 1101	16	n Selects 1 of 2 Loop Counter Registers
@BP[Offs]	000 1110	8/16	Data Memory @BP[Offs]
@BP[++Offs]	001 1110	8/16	Data Memory @BP[Offs]; Pre-Increment OFFS
@BP[--Offs]	010 1110	8/16	Data Memory @BP[Offs]; Pre-Decrement OFFS
OFFS	011 1110	8	Frame Pointer Offset from Base Pointer (BP)
DPC	100 1110	16	Data Pointer Control Register
GR	101 1110	16	General Register
GRL	110 1110	8	Low Byte of GR Register
BP	111 1110	16	Frame Pointer Base Pointer (BP)
@DP[n]	n00 1111	8/16	Data Memory @DP[n]
@++DP[n]	n01 1111	8/16	Data Memory @DP[n], Pre-Increment DP[n]
@--DP[n]	n10 1111	8/16	Data Memory @DP[n], Pre-Decrement DP[n]
DP[n]	n11 1111	16	n Selects 1 of 2 Data Pointers
2-CYCLE DESTINATION ACCESS USING PFX[n] REGISTER (See Special Notes)			
SC	000 1000	8	System Control Register
CKCN	110 1000	8	Clock Control Register
WDCN	111 1000	8	Watchdog Control Register
A[n]	nnn 1001	16	nnn Selects 1 of Second 8 Accumulators A[8]..A[15]
GRH	001 1110	8	High Byte of GR Register

Data Transfer Rules	dst (16-bit) ← src (16-bit):	dst[15:0] ← src[15:0]
	dst (8-bit) ← src (8-bit):	dst[7:0] ← src[7:0]
	dst (16-bit) ← src (8-bit):	dst[15:8] ← 00h *
		dst[7:0] ← src[7:0]
	dst (8-bit) ← src (16-bit):	dst[7:0] ← src[7:0]

**Note:* The PFX[0] register may be used to supply a separate high-order data byte for this type of transfer.*

Example(s):

```
MOVE  A[0], A[3]        ; A[0] ← A[3]
MOVE DP[0], #110h       ; DP[0] ← #0110h  (PFX[0] register used)
                        ; MOVE PFX[0], #01h (smart-prefixing)
                        ; MOVE DP[0], #10h
MOVE DP[0], #80h        ; DP[0] ← #0080h (PFX[0] register not needed)
```

Special Notes: Proper loading of the PFX[n] registers, when for the purpose of supplying 16-bit immediate data or accessing 2-cycle destinations, is handled automatically by the assembler and is therefore an optional step for the user when writing assembly source code. Examples of the automatic PFX[n] code insertion by the assembler are demonstrated below.

Initial Assembly Code	**Assembler Output**
MOVE DP[0], #0100h	MOVE PFX[0], #01h
MOVE A[15], A[7]	MOVE PFX[2], anysrc
	MOVE A[7], A[7]
MOVE A[8], #3040h	
MOVE PFX[2], #30h	MOVE A[0], #40h

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

Description: Replaces the specified bit of the active accumulator with the Carry bit.

Status Flags: S, Z

Operation: Acc.<b> ← C

Encoding: 15 … 0

1111	1010	bbbb	1010

MAXQ010 Example(s):

```
                        ; Acc = 80h, S=1, Z=0, C=0
MOVE Acc.7, C           ; Acc = 00h, S=0, Z=1
```

MAXQ020 Example(s):

```
                        ; Acc = 8000h, S=1, Z=0, C=0
MOVE Acc.15, C          ; Acc = 0000h, S=0, Z=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, 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
```

MOVE C, #1 **Set Carry Flag**

Description: Sets the Carry (C) processor status flag.

Status Flag: C ← 1

Operation: C ← 1

Encoding: 15 0

1101	1010	0001	1010

Example(s):

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

MOVE dst.<b>, #0 **Clear Bit**

Description: Clears the bit specified by dst.<b>.

Status Flags: C, E (if dst is PSF), S, Z

Operation: dst.<b> ← 0

Encoding: 15 0

1ddd	dddd	0bbb	0111

Example(s):

```
                    ; M0[0] = FEh
MOVE M0[0].1, #0    ; M0[0] = FCh
MOVE M0[0].7, #0    ; M0[0] = 7Ch
```

Special Notes: Only system module 8 and peripheral modules (0-5) are supported by MOVE dst.<b>, #0.

MOVE dst.<b>, #1 **Set Bit**

Description: Sets the bit specified by dst.<b>.

Status Flags: C, E (if dst is PSF), S, Z

Operation: dst.<b> ← 1

Encoding: 15 0

1ddd	dddd	1bbb	0111

Example(s):

```
                    ; M0[0] = 00h
MOVE  M0[0].1, #1   ; M0[0] = 02h
MOVE  M0[0].7, #1   ; M0[0] = 82h
```

Special Notes: Only system module 8 and peripheral modules (0-5) are supported by MOVE dst.<b>, #1.

NEG **Negate Accumulator**

Description: Performs a negation (two's complement) of the active accumulator and returns the result back to the active accumulator.

Status Flags: S, Z

Operation: Acc ← ~Acc + 1

Encoding: 15 … 0

1000	1010	1001	1010

MAXQ10 Example(s):

```
                        ; Acc = FEh, S=1, Z=0
NEG                     ; Acc = 02h, S=0, Z=0
```

MAXQ20 Example(s):

```
                        ; Acc = FEEDh, S=1, Z=0
NEG                     ; Acc = 0113h,  S=0, Z=0
```

OR src **Logical OR**

Description: Performs a logical-OR between the active accumulator (Acc or A[AP]) and the specified src data. For the complete list of src specifiers, reference the MOVE instruction. The MAXQ20 may use the PFX[n] register to supply the high byte of data for 8-bit sources.

Status Flags: S, Z

Operation: Acc ← Acc OR src

Encoding: 15 … 0

f010	1010	ssss	ssss

MAXQ10 Example(s):

```
                        ; Acc = 45h for each example
OR A[3]                 ; A[3]= 0Fh → Acc = 4Fh
OR #33h                 ; → Acc = 77h
```

MAXQ20 Example(s):

```
                        ; Acc = 2345h for each example
OR A[3]                 ; A[3]= 0F0Fh → Acc = 2F4Fh
OR #1133h               ; MOVE PFX[0], #11h (smart-prefixing)
                        ; OR #33h → Acc = 3377h
```

Special Notes: The active accumulator (Acc) is not allowed as the src for this operation.

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	← SP (initial)
0876h	← SP (after POP GR)
xxxxh	← SP (after POP @DP[0])
xxxxh	

POPI dst **Pop Word from the Stack Enable Interrupts**

Description: Pops a single word from the stack (@SP) to the specified dst and decrements the stack pointer (SP). Additionally, POPI returns the interrupt logic to a state in which it can acknowledge additional interrupts.

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

C, E (if dst = PSF)

Operation: dst ← @ SP--

INS ← 0

Encoding: 15 0

1ddd	dddd	1000	1101

Example(s): See POP

PUSH src **Push Word to the Stack**

Description: Increments the stack pointer (SP) and pushes a single word specified by src to the stack (@SP).

Status Flags: None

Operation: SP ← ++SP

Encoding: 15 0

f000	1101	ssss	ssss

Example(s):

```
PUSH GR                 ; GR=0F3Fh
PUSH #40h
```

Stack Data:

xxxxh	
0040h	← SP (after PUSH #40h)
0F3Fh	← SP (after PUSH GR)
xxxxh	← SP (initial)
xxxxh	

RET **Return from Subroutine**

Description: RET pops a single word from the stack (@SP) into the Instruction Pointer (IP) and decrements the stack pointer (SP). The decremented SP is saved as the new stack pointer (SP).

Status Flags: None

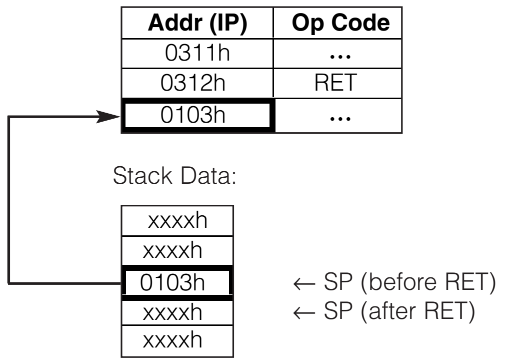
Operation: IP ← @ SP--

Encoding: 15 0

1000	1100	0000	1101

Example(s): RET

Code Execution:



Addr (IP)	Op Code
0311h	...
0312h	RET
0103h	...

Stack Data:

xxxxh
xxxxh
0103h
xxxxh
xxxxh

RET C/RET NC, RET Z/RET NZ, RET S **Conditional Return on Status Flag**

Description: Performs conditional return (RET) based upon the state of a specific processor status flag. RET C returns if the Carry flag is set while RET NC returns if the Carry flag is clear. RET Z returns if the Zero flag is set while RET NZ returns if the Zero flag is clear. RET S returns if the Sign flag is set. See RET for additional information on the return operation.

Status Flags: None

RET C C=1: IP ← @SP--

Operation: C=0: IP ← IP + 1

Encoding: 15 0

1010	1100	0000	1101

Example(s): RET C ; C=1, return (RET) is performed

RET NC

Operation: C=0: IP ← @SP--
C=1: IP ← IP +1

Encoding:

15			0
1110	1100	0000	1101

Example(s): RET NC ; C=1, return (RET) does not occur

RET Z

Operation: Z=1: IP ← @SP--
Z=0: IP ← IP + 1

Encoding:

15			0
1001	1100	0000	1101

Example(s): RET Z ; Z=0, return (RET) does not occur

RET NZ

Operation: Z=0: IP ← @SP--
Z=1: IP ← IP +1

Encoding:

15			0
1101	1100	0000	1101

Example(s): RET NZ ; Z=0, return (RET) is performed

RET S

Operation: S=1: IP ← @SP--
S=0: IP ← IP + 1

Encoding:

15			0
1100	1100	0000	1101

Example(s): RET S ; S=0, return (RET) does not occur

RETI **Return from Interrupt**

Description: RETI pops a single word from the stack (@SP) into the Instruction Pointer (IP) and decrements the stack pointer (SP). Additionally, RETI returns the interrupt logic to a state in which it can acknowledge additional interrupts.

Status Flags: None

Operation: IP ← @SP--

INS ← 0

Encoding: 15 0

1000	1100	1000	1101

Example(s): See RETI

RETI C/RETI NC, RETI Z/RETI NZ, RETI S **Conditional Return from Interrupt on Status Flag**

Description: Performs conditional return (RETI) based upon the state of a specific processor status flag. RETI C returns if the Carry flag is set while RETI NC returns if the Carry flag is clear. RETI Z returns if the Zero flag is set while RETI NZ returns if the Zero flag is clear. RETI S returns if the Sign flag is set. See RETI for additional information on the return operation.

Status Flags: None

RETI C

Operation: C=1: IP ← @SP--

INS ← 0

C=0: IP ← IP + 1

Encoding: 15 0

1010	1100	1000	1101

Example(s): RETI C ; C=1, return from interrupt (RETI) is performed

RETI NC

Operation: C=0: IP ← @SP--

INS ← 0

C=1: IP ← IP +1

Encoding: 15 0

1110	1100	1000	1101

Example(s): RETI NC ; C=1, return from interrupt (RETI) does not occur

RETI Z

Operation: Z=1: IP ← @SP--
INS ← 0
Z=0: IP ← IP + 1

Encoding: 15 0

1001	1100	1000	1101

Example(s): RETI Z ; Z=0, return from interrupt (RETI) does not occur

---

RETI NZ

Operation: Z=0: IP ← @SP--
INS ←0
Z=1: IP ← IP +1

Encoding: 15 0

1101	1100	1000	1101

Example(s): RETI NZ ; Z=0, return from interrupt (RETI) is performed

---

RETI S

Operation: S=1: IP ← @SP--
INS ← 0
S=0: IP ← IP + 1

Encoding: 15 0

1100	1100	1000	1101

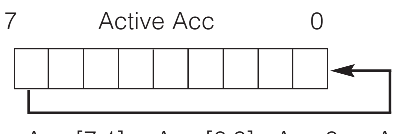
Example(s): RETI S ; S=0, return from interrupt (RETI) does not occur

(MAXQ10 Version) **Rotate Left Accumulator**
RL/RLC **Carry Flag (Ex/In)clusive**

Description: Rotates the active accumulator left by a single bit position. The RL instruction circulates the msb of the accumulator (bit 7) back to the lsb (bit 0) while the RLC instruction includes the Carry (C) flag in the circular left shift.

Status Flags: C (for RLC only), S, Z (for RLC only)

RL Operation:



Acc.[7:1]← Acc.[6:0]; Acc.0 ← Acc.7

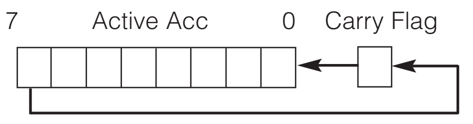
Encoding:

15			0
1000	1010	0100	1010

Example(s):

```
                        ; Acc = A3h, S=1, Z=0
RL                      ; Acc = 47h, S=0, Z=0
RL                      ; Acc = 8Eh, S=1, Z=0
```

RLC Operation:



Acc.[7:1]← Acc.[6:0]; Acc.0 ← C; C ← Acc.7

Encoding:

15			0
1000	1010	0101	1010

Example(s):

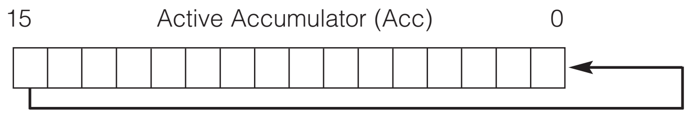
```
                        ; Acc = A3h, C=1, S=1, Z=0
RLC                     ; Acc = 47h, C=1, S=0, Z=0
RLC                     ; Acc = 8Fh, C=0, S=1, Z=0
```

(MAXQ20 Version) **Rotate Left Accumulator**
RL/RLC **Carry Flag (Ex/In)clusive**

Description: Rotates the active accumulator left by a single bit position. The RL instruction circulates the msb of the accumulator (bit 15) back to the lsb (bit 0) while the RLC instruction includes the Carry (C) flag in the circular left shift.

Status Flags: C (for RLC only), S, Z (for RLC only)

RL Operation:



Acc.[15:1]← Acc.[14:0]; Acc.0 ← Acc.15

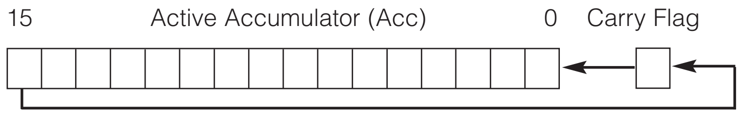
Encoding:

15			0
1000	1010	0100	1010

Example(s):

```
                        ; Acc = A345h, S=1, Z=0
RL                      ; Acc = 468Bh, S=0, Z=0
RL                      ; Acc = 8D16h, S=1, Z=0
```

RLC Operation:



Acc.[15:1]← Acc.[14:0]; Acc.0 ← C; C ← Acc.15

Encoding:

15			0
1000	1010	0101	1010

Example(s):

```
                        ; Acc = A345h, C=1, S=1, Z=0
RLC                     ; Acc = 468Bh, C=1, S=0, Z=0
RLC                     ; Acc = 8D17h, C=0, S=1, Z=0
```

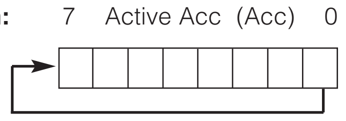
(MAXQ10 Version) **Rotate Right Accumulator**

RR/RRC **Carry Flag (Ex/In)clusive**

Description: Rotates the active accumulator right by a single bit position. The RR instruction circulates the lsb of the accumulator (bit 0) back to the msb (bit 7) while the RRC instruction includes the Carry (C) flag in the circular right shift.

Status Flags: C (for RRC only), S, Z (for RRC only)

RR Operation:



Acc.[6:0] ← Acc.[7:1]; Acc.7 ← Acc.0

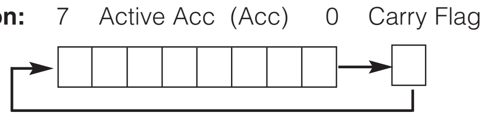
Encoding: 15 … 0

1000	1010	1100	1010

Example(s):

```
                        ; Acc = 45h, S=1, Z=0
RR                      ; Acc = A2h, S=1, Z=0
RR                      ; Acc = 51h, S=0, Z=0
```

RRC Operation:



Acc.7 ← C; C ← Acc.0

Encoding: 15 … 0

1000	1010	1101	1010

Example(s):

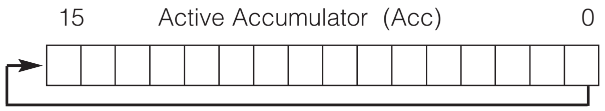
```
                        ; Acc = 45h, C=1, S=1, Z=0
RRC                     ; Acc = A2h, C=1, S=1, Z=0
RRC                     ; Acc = D1h, C=0, S=1, Z=0
```

(MAXQ20 Version) **Rotate Right Accumulator**
RR/RRC **Carry Flag (Ex/In)clusive**

Description: Rotates the active accumulator right by a single bit position. The RR instruction circulates the lsb of the accumulator (bit 0) back to the msb (bit 15) while the RRC instruction includes the Carry (C) flag in the circular right shift.

Status Flags: C (for RRC only), S, Z (for RRC only)

RR Operation:



Acc.[14:0]← Acc.[15:1]; Acc.15 ← Acc.0

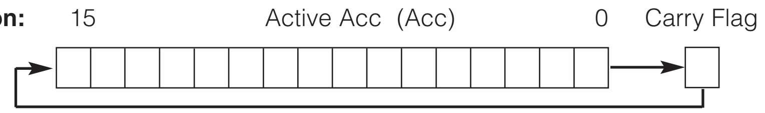
Encoding:

15			0
1000	1010	1100	1010

Example(s):

```
                    ;Acc = A345h, S=1, Z=0
RR                  ; Acc = D1A2h, S=1, Z=0
RR                  ; Acc = 68D1h, S=0, Z=0
```

RRC Operation:



Acc.[14:0]← Acc.[15:1]; Acc.15 ← C; C ← Acc.0

Encoding:

15			0
1000	1010	1101	1010

Example(s):

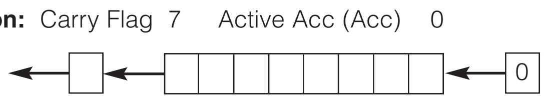
```
                    ; Acc = A345h, C=1, S=1, Z=0
RRC                 ; Acc = D1A2h, C=1, S=1, Z=0
RRC                 ; Acc = E8D1h, C=0, S=1, Z=0
```

(MAXQ10 Version) **Shift Accumulator Left Arithmetically**
SLA/SLA2/SLA4 **One, Two, or Four Times**

Description: Shifts the active accumulator left once, twice, or four times respectively for SLA, SLA2, and SLA4. For each shift iteration, a 0 is shifted into the lsb, and the msb is shifted into the Carry (C) flag. For signed data, this shifting process effectively retains the sign orientation of the data to the point at which overflow/underflow would occur.

Status Flags: C, S, Z

SLA Operation:



C ← Acc.7 ; Acc.[7:1]← Acc.[6:0] ; Acc.0 ← 0

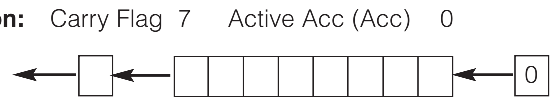
Encoding: 15 … 0

1000	1010	0010	1010

Example(s):

```
                        ; Acc = E3h, C=0, S=1, Z=0
SLA                     ; Acc = C6h, C=1, S=1, Z=0
SLA                     ; Acc = 8Ch, C=1, S=1, Z=0
```

SLA2 Operation:



C ← Acc.6 ; Acc.[7:2]← Acc.[5:0] ; Acc.[1:0] ← 0

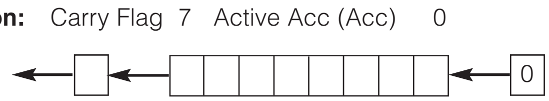
Encoding: 15 … 0

1000	1010	0011	1010

Example(s):

```
                        ; Acc = E3h, C=0, S=1, Z=0
SLA2                    ; Acc = 8Ch, C=1, S=1, Z=0
```

SLA4 Operation:



C ← Acc.4 ; Acc.[7:4]← Acc.[3:0] ; Acc.[3:0] ← 0

Encoding: 15 … 0

1000	1010	0110	1010

Example(s):

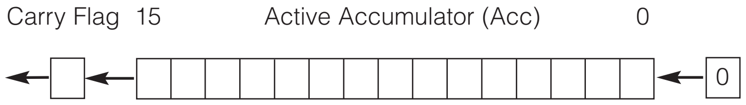
```
                        ; Acc = E3h, C=0, S=1, Z=0
SLA4                    ; Acc = 30h, C=0, S=0, Z=0
```

(MAXQ20 Version) **Shift Accumulator Left Arithmetically**
SLA/SLA2/SLA4 **One, Two, or Four Times**

Description: Shifts the active accumulator left once, twice, or four times respectively for SLA, SLA2, and SLA4. For each shift iteration, a 0 is shifted into the lsb, and the msb is shifted into the Carry (C) flag. For signed data, this shifting process effectively retains the sign orientation of the data to the point at which overflow/underflow would occur.

Status Flags: C, S, Z

SLA Operation:



C ← Acc.15; Acc.[15:1]← Acc.[14:0]; Acc.0 ← 0

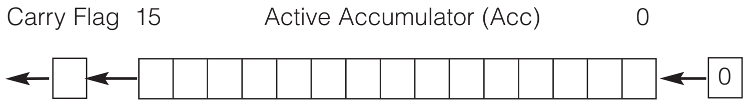
Encoding:

15			0
1000	1010	0010	1010

Example(s):

```
                       ; Acc = E345h, C=0, S=1, Z=0
SLA                    ; Acc = C68h, C=1, S=1, Z=0
SLA                    ; Acc = 8D14h, C=1, S=1, Z=0
```

SLA2 Operation:



C ← Acc.14 ; Acc.[15:2]← Acc.[13:0] ; Acc.[1:0] ← 0

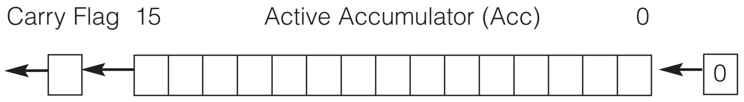
Encoding:

15			0
1000	1010	0011	1010

Example(s):

```
                       ; Acc = E345h, C=0, S=1, Z=0
SLA2                   ; Acc = 8D14h, C=1, S=1, Z=0
```

SLA4 Operation:



C ← Acc.12; Acc.[15:4]← Acc.[11:0]; Acc.[3:0] ← 0

Encoding:

15			0
1000	1010	0110	1010

Example(s):

```
                       ; Acc = E345h, C=0, S=1, Z=0
SLA4                   ; Acc = 3450h, C=0, S=0, Z=0
```

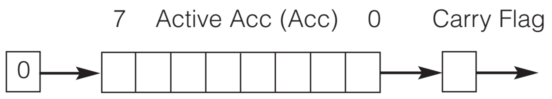
(MAXQ10 Version)
SR/SRA/SRA2/SRA4

Shift Accumulator Right/
Shift Accumulator Right Arithmetically
One, Two, or Four Times

Description: Shifts the active accumulator right once for the SR, SRA instructions and two or four times, respectively, for the SRA2, SRA4 instructions. The SR instruction shifts a 0 into the accumulator msb, while the SRA, SRA2, and SRA4 instructions effectively shift a copy of the current msb into the accumulator, thereby preserving any sign orientation. For each shift iteration, the accumulator lsb is shifted into the Carry (C) flag.

Status Flags: C, S (changes for SR only), Z

SR Operation:



Acc.[6:0]← Acc.[7:1]

Acc.7 ← 0

C ← Acc.0

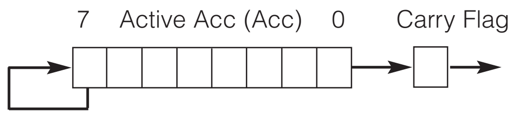
Encoding:

15			0
1000	1010	1010	1010

Example(s):

```
                        ; Acc = 45h, C=1, S=0, Z=0
SR                      ; Acc = 22h, C=1, S=0, Z=0
SR                      ; Acc = 11h, C=0, S=0, Z=0
```

SRA Operation:



Acc.[6:0]← Acc.[7:1]

Acc.7 ← Acc.7

C ← Acc.0

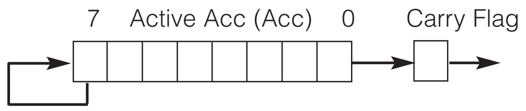
Encoding:

15			0
1000	1010	1111	1010

Example(s):

```
                        ; Acc = 03h, C=0, Z=0
SRA                     ; Acc = 01h, C=1, Z=0
SRA                     ; Acc = 00h, C=1, Z=1
```

SRA2 Operation:



Acc.[5:0] ← Acc.[7:2]

Acc.[7:6] ← Acc.7

C ← Acc.1

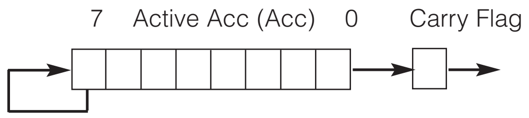
Encoding: 15 0

1000	1010	1110	1010

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: 15 0

1000	1010	1011	1010

Example(s):

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

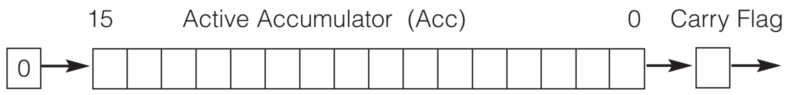
(MAXQ20 Version)
SR/SRA/SRA2/SRA4

Shift Accumulator Right/
Shift Accumulator Right Arithmetically
One, Two, or Four Times

Description: Shifts the active accumulator right once for the SR, SRA instructions and 2 or 4 times, respectively, for the SRA2, SRA4 instructions. The SR instruction shifts a 0 into the accumulator msb while the SRA, SRA2, and SRA4 instructions effectively shift a copy of the current msb into the accumulator, thereby preserving any sign orientation. For each shift iteration, the accumulator lsb is shifted into the Carry (C) flag.

Status Flags: C, S (changes for SR only), Z

SR Operation:



Acc.15 ← 0; Acc.[14:0]← Acc.[15:1]; C ← Acc.0

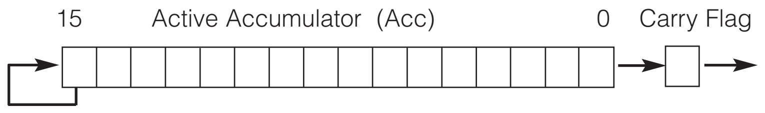
Encoding:

15			0
1000	1010	1010	1010

Example(s):

```
                       ; Acc = A345h, C=1, S=1, Z=0
SR                     ; Acc = 51A2h, C=1, S=0, Z=0
SR                     ; Acc = 28D1h, C=0, S=0, Z=0
```

SRA Operation:



Acc.[14:0]← Acc.[15:1]

Acc.15 ← Acc.15

C ← Acc.0

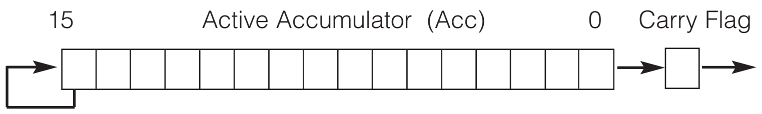
Encoding:

15			0
1000	1010	1111	1010

Example(s):

```
                       ; Acc = 0003h, C=0, Z=0
SRA                    ; Acc = 0001h, C=1, Z=0
SRA                    ; Acc = 0000h, C=1, Z=1
```

SRA2 Operation:



Acc.[13:0] ← Acc.[15:2]

Acc.[15:14] ← Acc.15

C ← Acc.1

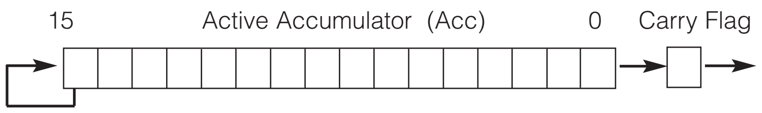
Encoding:

15			0
1000	1010	1110	1010

Example(s):

```
                        ; Acc = 0003h, C=0, Z=0
SRA2                    ; Acc = 0000h, C=1, Z=1
```

SRA4 Operation:



Acc.[11:0] ← Acc.[15:4]

Acc.[15:12] ← Acc.15

C ← Acc.3

Encoding:

15			0
1000	1010	1011	1010

Example(s):

```
                        ; Acc = 9878h, C=0, Z=0
SRA4                    ; Acc = F987h, C=1, Z=0
SRA4                    ; Acc = FF98h, C=0, Z=0
```

SUB/SUBB src **Subtract /Subtract with Borrow**

Description: Subtracts the specified src from the active accumulator (Acc) and returns the result back to the active accumulator. The SUBB additionally subtracts the borrow (Carry Flag), which may have resulted from previous subtraction. For the complete list of src specifiers, reference the MOVE instruction. The MAXQ20 may use the PFX[n] register to supply the high byte of data for 8-bit sources.

Status Flags: C, S, Z, OV

SUB Operation: Acc ← Acc - src

Encoding: 15 0

f101	1010	ssss	ssss

MAXQ10 Example(s):

```
                ; Acc = 23h to start, A[1]= 12h
SUB    A[1]     ; Acc = 11h, C=0, S=0, Z=0
SUB    A[1]     ; Acc = FFh, C=1, S=1, Z=0
```

MAXQ20 Example(s):

```
                ; Acc = 2345h to start, A[1]= 1250h
SUB    A[1]     ; Acc = 10F5h, C=0, S=0, Z=0, OV=0
SUB    A[1]     ; Acc = FEA5h, C=1, S=1, Z=0, OV=0
SUB    A[2]     ; A[2] =7FFFh
                ; → Acc = 7EA6h; C=0, S=0, Z=0, OV=1
```

SUBB Operation: Acc ← Acc - (src + C)

Encoding: 15 0

f111	1010	ssss	ssss

MAXQ10 Example(s):

```
                ; Acc = 23h, A[1]= 12h, C=1
SUBB   A[1]     ; Acc = 10h, C=0, S=0, Z=0
SUBB   A[1]     ; Acc = FEh, C=1, S=1, Z=0
SUBB   #0Dh     ; Acc = F0h, C=0, S=1, Z=0
```

MAXQ20 Example(s):

```
                ; Acc = 2345h, A[1]= 1250h, C=1
SUBB   A[1]     ; Acc = 10F4h, C=0, S=0, Z=0
SUBB   A[1]     ; Acc = FEA4h, C=1, S=1, Z=0
```

Special Notes: The active accumulator (Acc) is not allowed as the src for these operations.

(MAXQ20 Only) **Exchange Accumulator Bytes**
XCH

Description: Exchanges the upper and lower bytes of the active accumulator.

Status Flags: S

Operation: Acc.[15:8] ← Acc.[7:0]
Acc.[7:0] ← Acc.[15:8]

Encoding: 15 … 0

1000	1010	1000	1010

Example(s):

```
                ; Acc = 2345h
XCHN            ; Acc = 4523h
```

XCHN **Exchange Accumulator Nibbles**

Description: Exchanges the upper and lower nibbles in the active accumulator byte(s).

Status Flags: S

Operation: Acc.[7:4] ← Acc.[3:0]
Acc.[3:0] ← Acc.[7:4]
Acc.[15:12] ← Acc.[11:8]
Acc.[11:8] ← Acc.[15:12]

Encoding: 15 … 0

1000	1010	0111	1010

MAXQ10
Example(s):

```
                ; Acc = 23h
XCHN            ; Acc = 32h
```

MAXQ20
Example(s):

```
                ; Acc = 2345h
XCHN            ; Acc = 3254h
```

XOR src **Logical XOR**

Description: Performs a logical-XOR between the active accumulator (Acc or A[AP]) and the specified src data. For the complete list of src specifiers, reference the MOVE instruction. The MAXQ20 may use the PFX[n] register to supply the high byte of data for 8-bit sources.

Status Flags: S, Z

Operation: Acc ← Acc XOR src

Encoding: 15 0

f011	1010	ssss	ssss

MAXQ10 Example(s):

```
                    ; Acc = 23h
XOR A[2]            ; A[2]=0Fh; Acc ← 2Ch
XOR #28h            ; Acc ← 04h
```

MAXQ20 Example(s):

```
                    ; Acc = 2345h
XOR A[2]            ; A[2]=0F0Fh; Acc ← 2C4Ah
```

Special Notes: The active accumulator (Acc) is not allowed as the src for this operation.

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

Description: Performs a logical-XOR 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 XOR Acc.<b>

Encoding: 15 0

1011	1010	bbbb	1010

MAXQ10 Example(s):

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

MAXQ20 Example(s):

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

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.

REVISION HISTORY

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	9/04	Original release.	—
1	1/05	Updated *Loading a 16-bit register with a 16-bit immediate value:* Changed *…PFX[2]…to…PFX[0].*	28
		Updated *I/O Port: Type B:* Changed *alternate function to special function.*	54
		Replaced *Table 10.*	62
		Updated *Timer 0 Mode: 16-Bit Timer/Counter:* Removed *also* from *An interrupt* ***also*** *occurs if enabled…*	63
		Updated *RTC Trim (RTRM) Register, (TRM4:0) Trim Calibration Bits:* Changed *32 seconds* to *16 seconds*	134
		Updated *JUMP NE:* Changed *E = 1, branch taken* to *E = 0, branch taken.*	165
2	4/05	Updated *Debug Mode Special Considerations:* Added two bullets regarding single stepping because the debug engine step mechanism (that forces the IP to 8010h) does not allow a memory read from the utility ROM to work properly when single stepping.	149
3	9/05	Updated *Polarity Control* and *Output Enable, Polarity Control:* Added the following sentence: *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.*	73, 75
4	10/05	Updated *External Reset:* Added *watchdog* to list of causes for a reset condition.	25
		Updated *Watchdog Timer Reset:* Added *and holds the $\overline{RST}$ pin low* to statement for a watchdog resetting the processor.	
5	2/06	Updated *Accessing the Multiplier:* Added *and the OF bit* to last sentence of first paragraph to clarify that the CLD bit, when set, clears all data registers and the OF bit to zero.	111
		Updated *Hardware Multiplier Control (MCNT) Register Description, Bits CLD and OF:* Updated bit descriptions to clarify that the OF bit is cleared to 0 when the CLD bit is set to 1.	112
6	9/08	Created newer template-style document. Updated layout for peripheral registers based on new template style.	All
		Added *Figure 9-14: Bit Length Decoding Example* (omitted in error.)	9-23