Rev 2; 7/10

MAXQ610 USER'S GUIDE

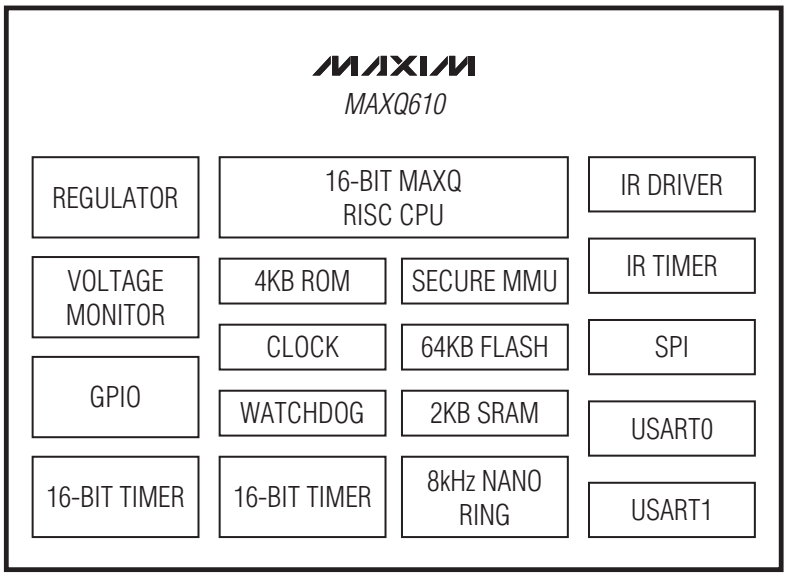


TABLE OF CONTENTS

SECTION 1: Overview 1-1

SECTION 2: Architecture 2-1

SECTION 3: Programming 3-1

SECTION 4: System Register Description 4-1

SECTION 5: Peripheral Register Modules 5-1

SECTION 6: General-Purpose I/O Module 6-1

SECTION 7: Timer/Counter Type B 7-1

SECTION 8: IR Timer 8-1

SECTION 9: Serial I/O Module 9-1

SECTION 10: Serial Peripheral Interface (SPI) Module 10-1

SECTION 11: Test Access Port (TAP) 11-1

SECTION 12: In-Circuit Debug Mode 12-1

SECTION 13: In-System Programming (JTAG) 13-1

SECTION 14: MAXQ610 Instruction Set Summary 14-1

SECTION 15: Utility ROM 15-1

REVISION HISTORY R-1

SECTION 1: OVERVIEW

The MAXQ® family of 16-bit reduced instruction set computing (RISC) microcontrollers is targeted towards 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.

1.1 Instruction Set

The MAXQ610 microcontroller uses an instruction set where all instructions are fixed in length (16 bits). 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. Because 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 the MAXQ610 are separate from one another and are each accessed by a separate bus. This type of memory architecture (known as 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. Because data memory is not required to store program code, its width can also vary and could conceivably be targeted for a specific application.

Also, because 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

Because all functions in the MAXQ610 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 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 the MAXQ610 family of microcontrollers.

The common register set, also known as the system registers, includes the following:

- Arithmetic logic unit (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
- Autodecrementing 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 soft stack
- Interrupt vector table and priority registers
- One code pointer for quick program memory access as data

Peripheral registers (module 0 to module 5) on the MAXQ610 contain registers that are used to access the peripherals, including:

- General-purpose I/O ports
- External interrupts
- Timers/counters
- USART ports
- Serial peripheral interface (SPI™) port

SPI is a trademark of Motorola, Inc.

SECTION 2: ARCHITECTURE

This section contains the following information:

2.1 Instruction Decoding 2-4
2.2 Register Space 2-5
2.3 Memory Organization 2-6
2.3.1 Program Memory 2-6
2.3.2 Utility ROM 2-6
2.3.3 Data Memory 2-7
2.3.4 Stack Memory 2-8
2.4 Memory Management Unit 2-8
2.5 Memory Mapping 2-8
2.5.1 Memory Mapping Into Data Space 2-9
2.5.2 Memory Mapping into Code Space 2-12
2.5.3 Memory Mapping Rules 2-12
2.6 Memory Protection 2-13
2.6.1 Rules for System Software 2-14
2.6.2 Privilege Exception Interrupt 2-15
2.6.3 Memory Access Protection Impact on Data Pointers (and Code Pointer) 2-15
2.6.4 Debugging 2-17
2.6.5 Enabling Memory Protection 2-17
2.6.6 Reset Procedure and Setup of Memory Protection 2-17
2.6.7 Loader Access Control 2-18
2.6.8 Disabling MAXQ610-Specific Memory Access Features 2-20
2.6.9 No User-Loader Segment 2-21
2.7 Clock Generation 2-22
2.7.1 External Clock (Crystal/Resonator) 2-23
2.7.2 External Clock (Direct Input) 2-23
2.7.3 Internal System Clock Generation 2-24
2.8 Wake-Up Timer 2-24
2.8.1 Using the Wake-Up Timer to Exit Stop Mode 2-24
2.9 Interrupts 2-24
2.9.1 Servicing Interrupts 2-24
2.9.2 Interrupt System Operation 2-25
2.9.3 Synchronous vs. Asynchronous Interrupt Sources 2-25
2.9.4 Interrupt Prioritization by Software 2-26
2.9.5 Interrupt Exception Window 2-27
2.10 Operating Modes 2-27
2.11 Reset Mode 2-27
2.11.1 Power-On/Power-Fail Reset 2-28

2.11.2 External Reset 2-29
2.11.3 Watchdog Timer Reset 2-29
2.11.4 Internal System Reset 2-29
2.12 Power-Management Mode 2-29
2.12.1 Switchback 2-30
2.13 Stop Mode 2-30

LIST OF FIGURES

Figure 2-1. MAXQ610 Transport-Triggered Architecture 2-3
Figure 2-2. Instruction Word Format 2-4
Figure 2-3. MAXQ610 Memory Map (64KB Program Space) 2-9
Figure 2-4. CDA Functions in Word Mode 2-10
Figure 2-5. CDA Functions in Byte Mode 2-11
Figure 2-6. MAXQ610 Memory Map and UPA 2-12
Figure 2-7. Overview of Memory Regions 2-18
Figure 2-8. Program Memory Segmentation (Only Two Segments) 2-21
Figure 2-9. MAXQ610 Clock Sources 2-22
Figure 2-10. On-Chip Crystal Oscillator 2-23

LIST OF TABLES

Table 2-1. Register-to-Register Transfer Operations 2-6
Table 2-2. CDA Bits to Access Program Space as Data 2-9
Table 2-3. Memory Areas and Associated Maximum Privilege Levels 2-13
Table 2-4. PRIV Register Bit Definitions 2-13
Table 2-5. Privilege Level Constants 2-13
Table 2-6. System Clock Rate Control Settings 2-24
Table 2-7. Interrupt Priority 2-26
Table 2-8. Power-Fail Reset Check Interval 2-28

SECTION 2: ARCHITECTURE

The MAXQ610 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.

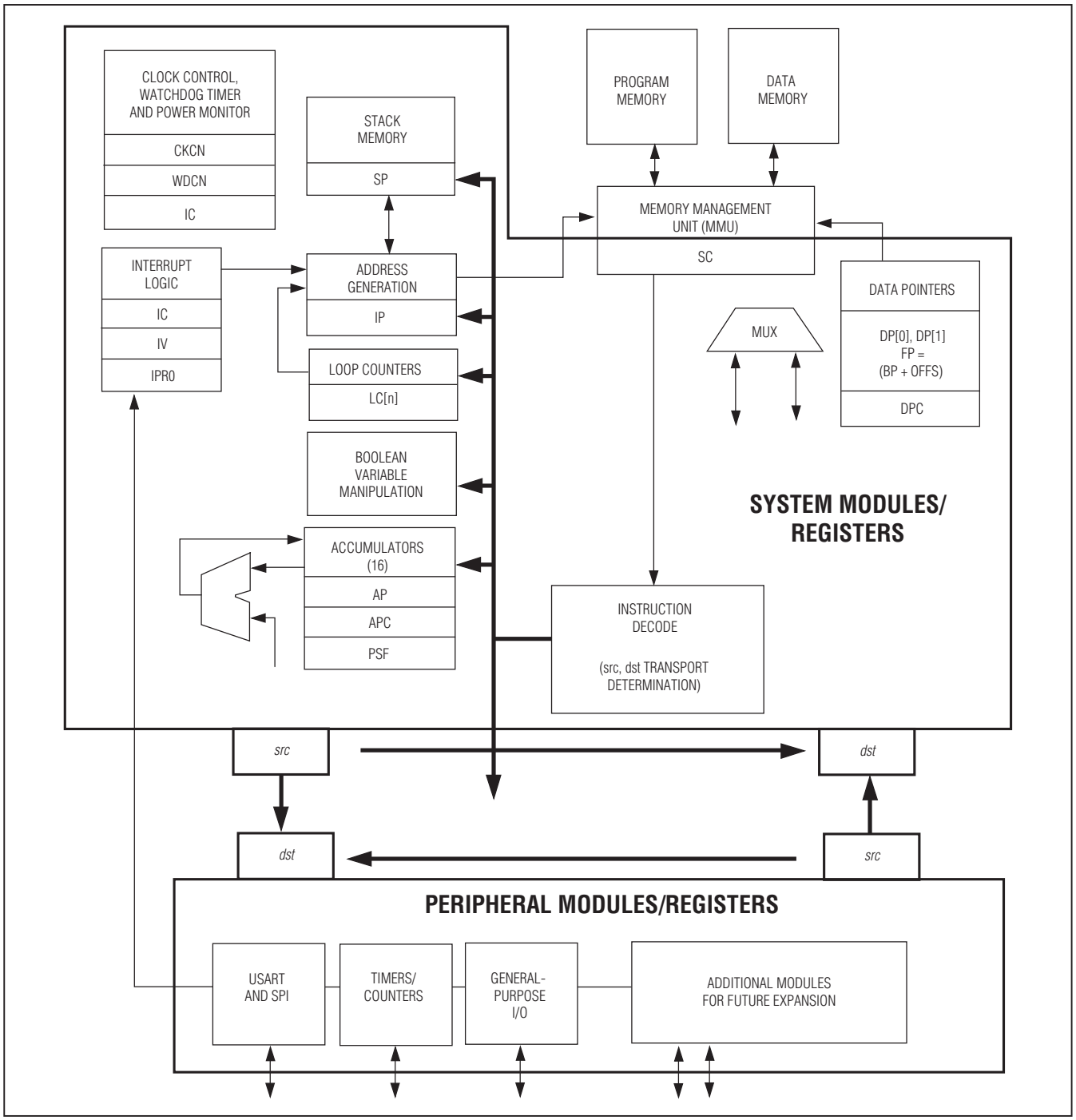


Figure 2-1. MAXQ610 Transport-Triggered Architecture

Memory access from the MAXQ610 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 MAXQ610 to decode and execute nearly all instructions 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 offset register (OFFS). All three pointers support postincrement/decrement functionality for read operations and preincrement/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 accessible through the stack pointer (SP). Program memory is read accessible through the code pointer (CP), which supports postincrement/decrement functionality.

2.1 Instruction Decoding

Every MAXQ instruction is encoded as a single 16-bit word according to the format shown 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 from which the source value is read.

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 4 bits contain the module specifier and the upper 4 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 4 bits containing the module specifier and the upper 3 bits containing the register subindex within that module.

Because the source field is 8 bits wide and 4 bits are required to specify the module, any one of 16 registers in that module can 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 can initially be considered a limitation, this space can be used effectively. First, since read-only registers can never be specified as destinations, they can be placed in the second eight locations in a module to give single-cycle read access. Second, 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/compiler and requires one additional execution cycle.**

The MAXQ architecture is transport-triggered. This means that writing to or reading from certain register locations also causes side effects. These side effects form the basis for the higher level op codes defined by the assembler, such as ADDC, OR, JUMP, and so on. These op codes are actually implemented as MOVE instructions between certain register locations, **while the encoding is handled by the assembler/compiler 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 MAXQ610 provides a total of 16 register modules. Each of these modules contains 32 registers. The first eight registers in each module can be read from or written to in a single cycle; the second eight registers can be read from in a single cycle and written to in two cycles (by using the prefix register, PFX); the last 16 registers can be read or written in two cycles (always requiring use of the prefix register, PFX).

Registers can 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 would 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.

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 can 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 depends 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 is 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 to5h) are known as the peripheral register modules, while the high 10 modules (specifiers 6h to 0Fh) 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 MAXQ610 architecture. This includes all register locations that are used to implement higher level op codes as well as the following common system features:

- ALU (16 bits) and associated status flags (zero, equals, carry, sign, overflow)
- 16 working accumulator registers (16-bit width), along with associated control registers
- Instruction pointer
- Registers for interrupt control and handling
- Autodecrementing loop counters for fast, compact looping
- Two data pointer registers, a frame pointer, and a stack pointer for data memory/stack access
- One code pointer register for program memory access

The peripheral registers define additional functionality included in the MAXQ610. This functionality is broken up into discrete modules so that only the features that are required for a given product need to be included. Because the peripheral registers add functionality outside the common MAXQ system architecture, they are not used to implement op codes.

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]

2.3 Memory Organization

Beyond the internal register space, memory on the MAXQ610 microcontroller is organized according to a Harvard architecture, with a separate address space and bus for program memory and data memory.

To provide additional memory map flexibility, 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 0000h and is contiguous through 7FFFh (64KB). 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 forces program flow to the new address on the next cycle following the write. The content of the instruction pointer is incremented by one 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 is stored in the lowest bytes of the program memory.

ROM-only versions of the MAXQ610 require program code to be masked into the program ROM during chip fabrication; no write access to program memory is available. Program flash memory provides in-system programming capability, but requires that the memory targeted for the write operation be programmed (erased). The utility ROM provides routines to carry out the necessary operations (erase, write, verify) on flash memory.

2.3.2 Utility ROM

A utility ROM is placed in the start of the upper half of the program memory space starting at address 8000h. This utility ROM provides the following system utility functions:

- Reset vector
- Bootstrap function for system initialization
- Utility functions to match/query customer specific secrets to prevent loading and/or operation on generic MAXQ610 parts
- In-application programming (flash versions only)
- In-circuit debug (flash versions only)

Following each reset, the processor automatically starts execution at address 8000h in the utility ROM, allowing utility ROM code to perform any necessary system support functions. Next, the 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 1, the processor executes 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 by 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 SRAM data memory begins at address 0000h and is contiguous through 03FFh (2KB) in word mode. Data memory is accessed by indirect register addressing through a data pointer (@DP), frame pointer (@BP[OFFS]), or stack pointer (PUSH/POP). The data pointer is used as one of the operands in a MOVE instruction. If the data pointer is used as source, the CPU 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 CPU executes a store operation that writes data to the data memory location addressed by the data pointer. The data pointer itself can be directly accessed by the user software.

The MAXQ610 incorporates two 16-bit data pointers (DP[0] and DP[1]) to support data block transfers. All data pointers support indirect addressing mode and indirect addressing with autoincrement or autodecrement. Data pointers DP[0] and DP[1] can be used as postincrement/decrement source pointers by a MOVE instruction or preincrement/decrement destination pointers by a MOVE instruction. Using a data pointer indirectly with "++" automatically increases 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 "--" decreases 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 postincrement/decrement source pointer by a MOVE instruction or as a preincrement/decrement destination pointer. Using the frame pointer indirectly with "++" (@BP[++OFFS] for a write or @BP[OFFS++] for a read) automatically increases 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) decreases 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/carryout from the OFFS register.

In addition, the MAXQ610 has a code pointer (CP) to support data block transfer from flash memory (or masked ROM on a ROM-only part). This allows the user to access the program flash memory as data, even when executing from the flash. In addition, there are some restrictions on use of the code pointer due to memory access protection. See *Section 2.6.3: Memory Access Protection Impact on Data Pointers (and Code Pointer)* for details. The code pointer, like the normal data pointers, supports indirect addressing mode and indirect addressing with autoincrement or autodecrement. The code pointer can be used as postincrement/decrement source pointer by MOVE instructions. Using the code pointer indirectly with "++" automatically increases the content of the active code pointer by 1 immediately following the execution of the read operation (e.g., MOVE dst, @CP++). Using code pointer indirectly with "--" decreases the content of the active code pointer by 1 immediately following the execution of the read operation (e.g., MOVE dst, @CP--).

A normal data memory cycle using DP[0], DP[1], and FP to access SRAM takes 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. To read program memory as data using CP requires two system clocks. Data memory mapping and access control are handled by the memory management unit (MMU). Read/write access to the data memory can be in word or in byte.

2.3.4 Stack Memory

The MAXQ610 implements a soft stack that uses the on-chip data memory (SRAM) 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 PUSH, POP, and POPI instructions. The POPI instruction acts identically to the POP instruction, except that it additionally set the IPS bits.

The width of the stack is 16 bits to accommodate the instruction pointer size. 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 MAXQ610 is the SRAM data memory size.

On reset, the stack pointer SP initializes to the top of the stack (03F0h). The CALL, PUSH, and interrupt vectoring operations increase the stack depth (decrement SP) and then store a value at the memory location pointed to by SP. The RET, RETI, POP, and POPI operations retrieve the value at @SP and then decrease the stack depth (increment SP).

2.4 Memory Management Unit

Memory allocation and access control for program and data memory is managed by the MMU.

The MAXQ610 MMU supports the following:

- Flash or masked ROM code memory of up to 64KB; utility ROM of 4KB and data memory SRAM of 2KB.
- In-system and in-application programming of embedded flash (flash versions only).
- Access to any of the three memory areas (SRAM, code memory, utility ROM) using the data memory pointers and the code pointer.
- Execution from any of the program memory areas (code memory, factory written and tested utility ROM routines) and from data memory.

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

- Program memory:

  Physical program memory pages (P0, P1) are logically mapped into data space based upon selection of byte/word access mode and CDA[1:0] bit settings.

- Data memory:

  Access can be either word or byte.

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

The MAXQ610 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.

2.5 Memory Mapping

Figure 2-3 summarizes the MAXQ610 default memory maps.

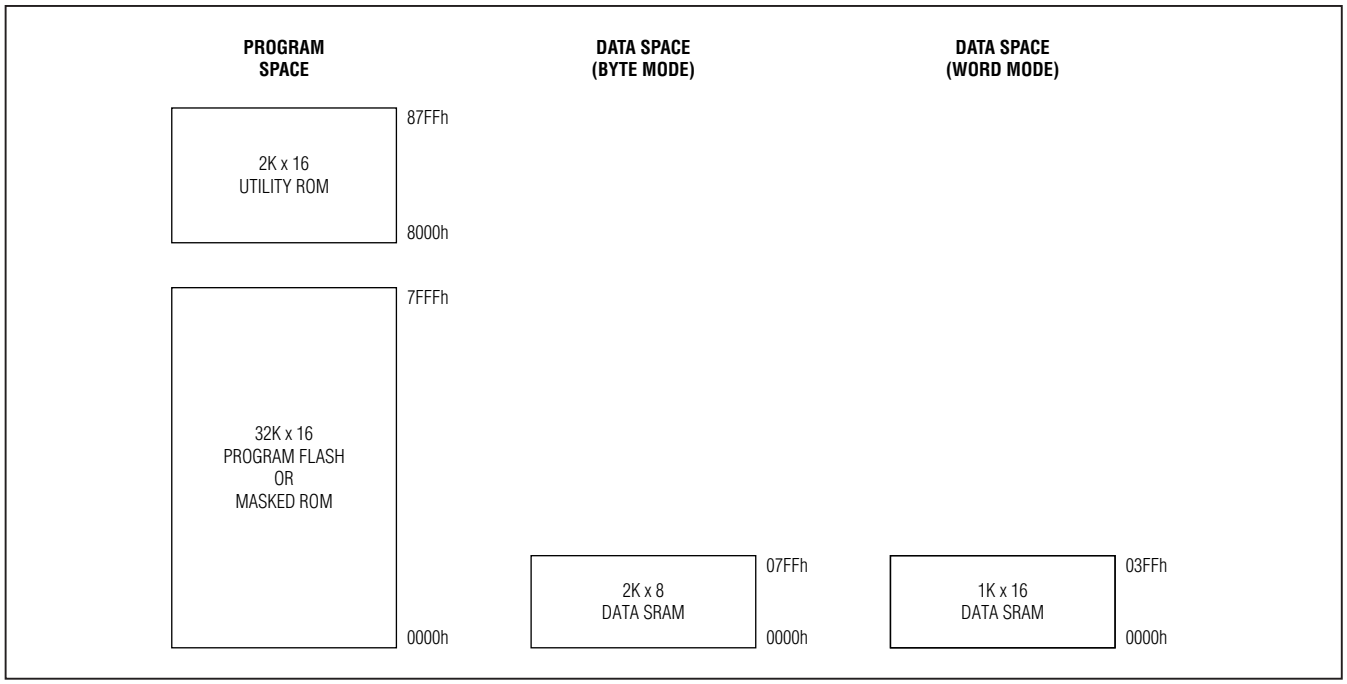


Figure 2-3. MAXQ610 Memory Map (64KB Program Space)

Table 2-2. CDA Bits to Access Program Space as Data

CDA[1:0]	SELECTED PAGE IN BYTE MODE	SELECTED PAGE IN WORD MODE
00	P0	P0 and P1
01	P1	P0 and P1

2.5.1 Memory Mapping Into Data Space

The MAXQ610 maps program memory into data space from 0000h to 7FFFh. The selection of physical program memory page or pages to be logically mapped to data space is determined by the CDA1 and CDA0 bits, as shown in Table 2-2. Note that CDA1 is fixed at 0.

Figure 2-3 summarize the default memory maps for this memory structure. The WBSn bits of the MAXQ610 default to word access mode (WBSn = 1).

The upper half of the data memory map (8000h to FFFFh) is the logical area for the utility ROM when accessed as data. Executing code from the utility ROM allows the user to map the program memory to 8000h to FFFFh by properly selecting the CDA bits.

Figure 2-4 and 2-5 illustrate the effects of the CDA bits.

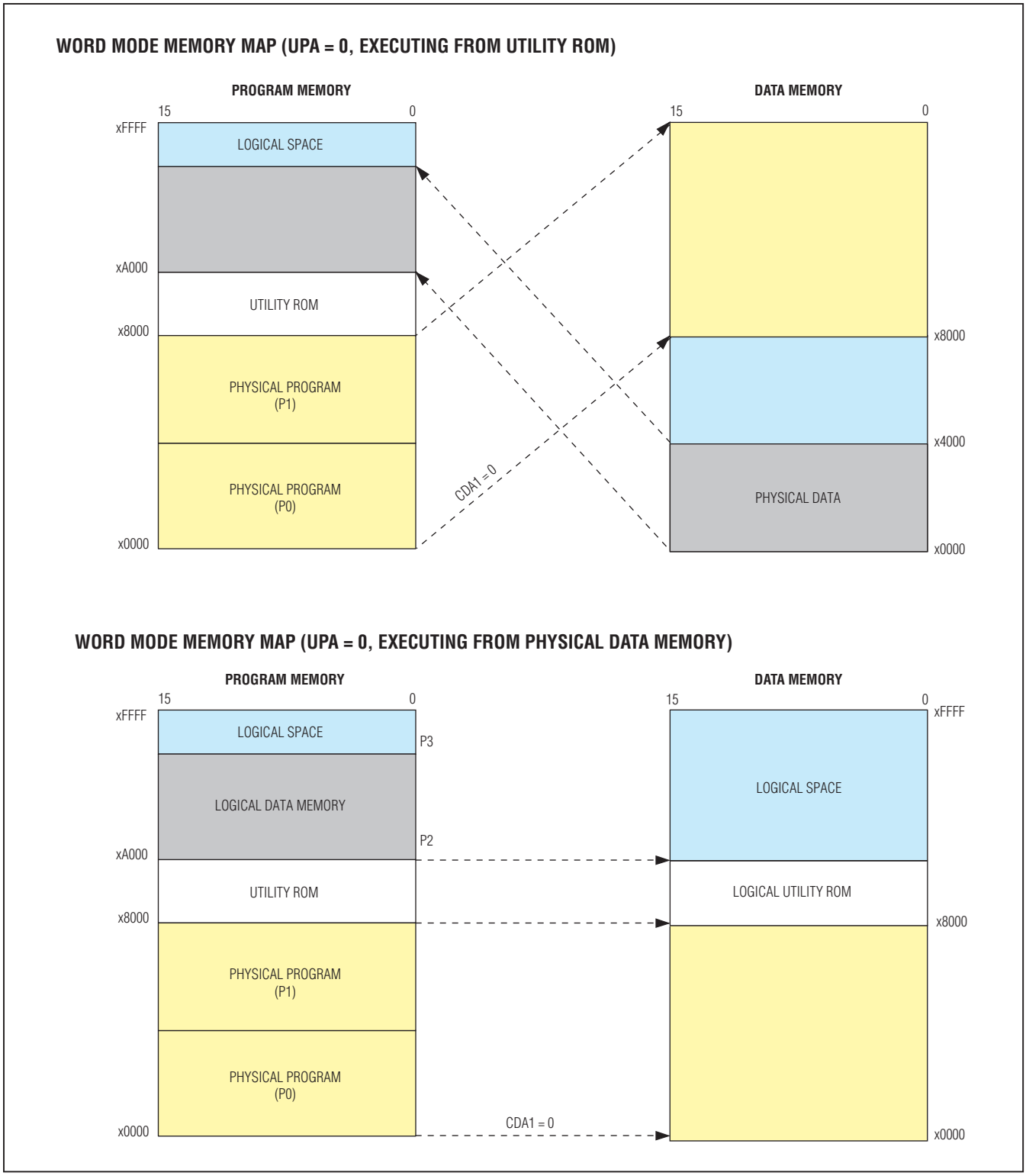


Figure 2-4. CDA Functions in Word Mode

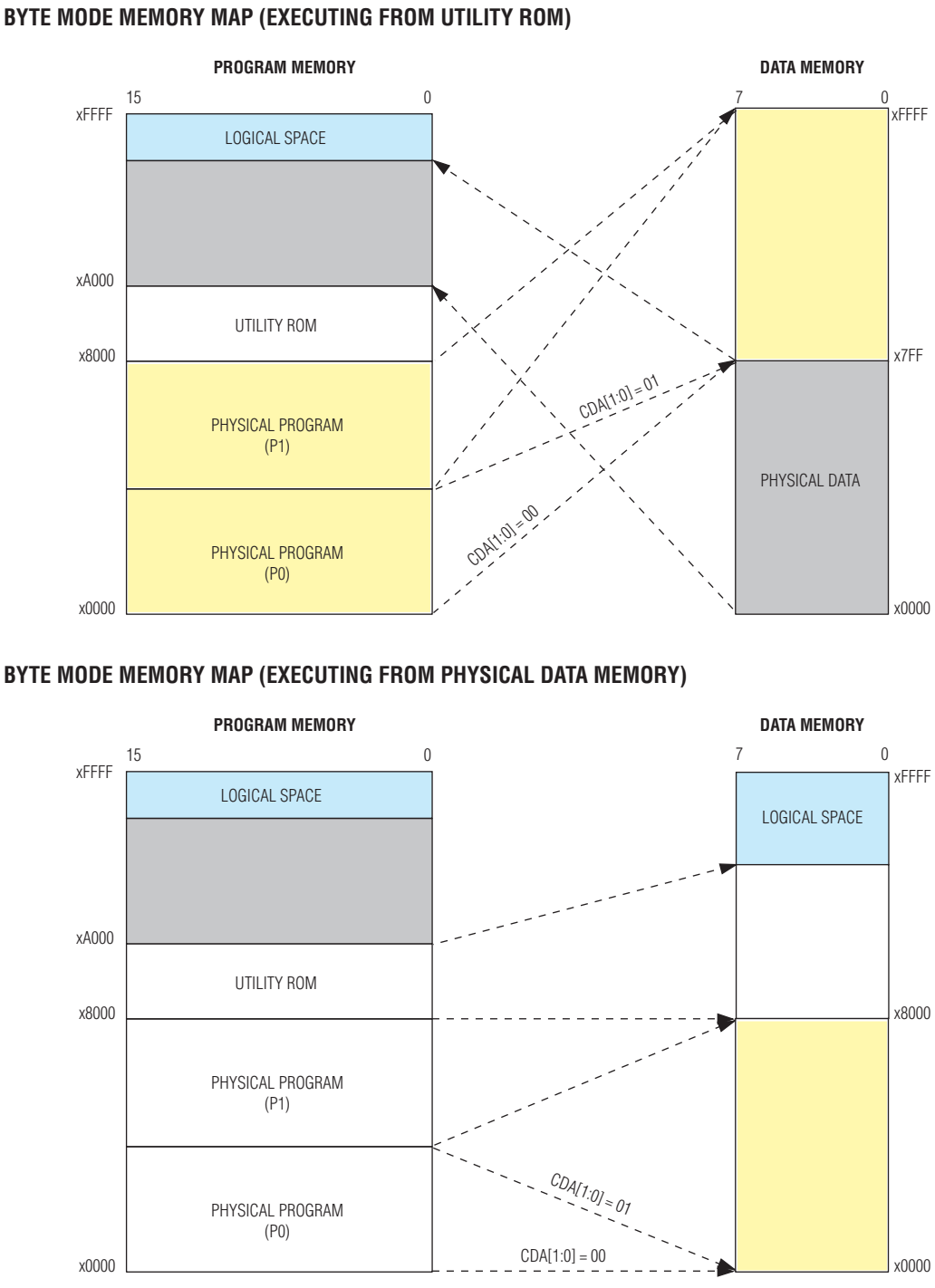


Figure 2-5. CDA Functions in Byte Mode

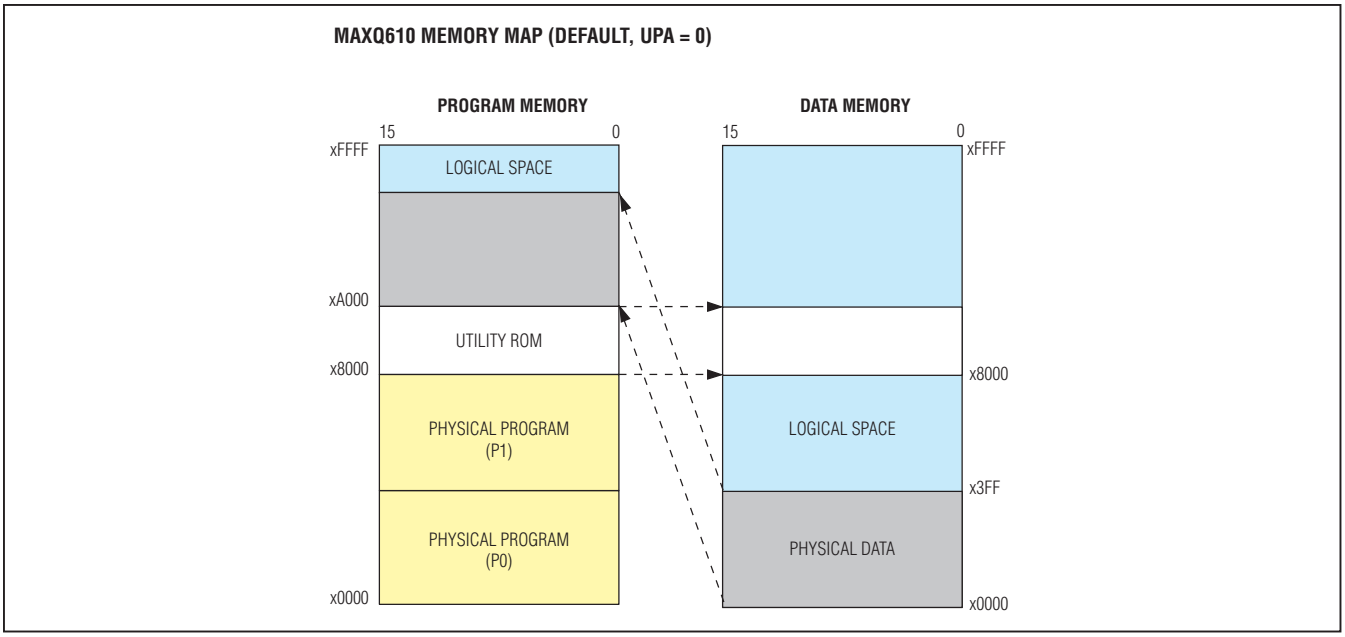


Figure 2-6. MAXQ610 Memory Map and UPA

2.5.2 Memory Mapping into Code Space

The effective program address can be anywhere in the full 64KB memory space. Program memory from 0000h to 7FFFh is the normal user code segment, followed by the utility ROM. The top of the memory is the logical area for data memory when accessed as a code segment.

2.5.3 Memory Mapping Rules

When executing program code in a particular memory segment, the same memory segment cannot be simultaneously accessed as data.

The following is a summary of the memory mapping rules.

- When executing from the normal user code segment:

  The lower 32KWords program space (P0 and P1) is always executable as program.

  The utility ROM is an extension of the program space if the UPA bit is 0.

  The physical data memory is available for access as a code segment with offset at 0A000h if the UPA bit is 0.

  Load and store operations to data memory are executed normally when addressed to the physical data memory.

  The utility ROM can be read as data, starting at 08000h of the data space.

- When executing from the utility ROM (only when UPA bit is 0):

  The lower 32KWords program space (P0 and P1) functions as normal program memory.

  Data memory is available for access as a code segment at the upper half of the program memory map, immediately following the utility ROM segment.

  Load and store operations to data memory are executed normally when addressed to the physical data memory.

  P0 can be accessed as data with offset at 08000h when CDA[1:0] = 00b in byte mode or CDA1 = 0 in word mode.

  P1 can be accessed as data with offset at 08000h when CDA[1:0] = 01b in byte mode or at offset 0C000h when CDA1 = 0 in word mode.

- When executing from the data memory (only when UPA is 0):

Program flows freely between the lower 32KWords user code (P0 and P1) and the utility ROM segment.

The utility ROM can be accessed as data with offset at 08000h.

P0 can be accessed as data with offset at 0000h when CDA[1:0] = 00b in byte mode or CDA1 = 0 in word mode.

P1 can be accessed as data with offset at 0000h when CDA[1:0] = 01b in byte mode or at offset 04000h when CDA1 = 0 in word mode.

2.6 Memory Protection

The MAXQ610 supports privilege levels for code. When enabled, code memory is separated into three areas. Each area has an associated privilege level. RAM/utility ROM are assigned privilege levels as well:

- Code in the system area can be confidential. Code in the user areas can be prevented from reading and writing system code.
- The user loader can be protected from user application code.

Table 2-3. Memory Areas and Associated Maximum Privilege Levels

AREA	PAGE ADDRESS	MAXIMUM PRIVILEGE LEVEL
System	0 to ULDR-1	High
User Loader	ULDR to UAPP-1	Medium
User Application	UAPP to top	Low
Utility ROM	N/A	High
Other (RAM)	N/A	Low

The PRIV register reflects the current execution privilege. Hardware guarantees that the contents of PRIV are never higher than the maximum privilege level of the memory area the code is running from. For example, if user code were trying to set PRIV to high, this would be prevented by hardware. However, any code can decide to lower the privilege level at any time (see Equation 1).

PRIV = min(maxprivilege(IP), PRIV) (Equation 1)

The bit contents of the PRIV register are shown in Table 2-4. The convenient constants high/medium/low are defined in Table 2-5, but all values from 00b to 11b can be used.

In addition to the PRIV register, the privilege level can also be set by writing to PRIVT0 and PRIVT1 in sequence. Again, hardware guarantees that the contents of PRIVT0 are never higher than the maximum privilege level of the memory area the code is running from.

When writing to PRIVT1, hardware modifies the PRIV register based on Equation 2.

PRIV = min(PRIVT0, argument, maxprivilege(IP)) (Equation 2)

This means that, when using PRIVT[1:0], the privilege level cannot be raised unless all code between the writes to PRIVT0 and PRIVT1 executes. Writing to PRIV automatically resets PRIVT0 to low.

Table 2-4. PRIV Register Bit Definitions

BIT	3	2	1	0
MEANING	System Write	System Read	User Loader Write	User Loader Read

Table 2-5. Privilege Level Constants

BIT	3	2	1	0
HIGH	1	1	1	1
MEDIUM	0	0	1	1
LOW	0	0	0	0

2.6.1 Rules for System Software

While privilege levels are implemented in hardware, there are two ways user code could try to circumvent the memory access protection:

- Manipulation of shared, common stack or registers
- Jumping or calling to code in system memory that is not an official entry point

To ensure a safe system and prevent these attacks, the system code programmer must follow the following rules:

- System code must not save and restore the privilege level. Instead, every interrupt and every system library **function that raises the privilege must also unconditionally lower the privilege before exiting**. If there are interrupts that lower the privilege level, or interrupt code running outside of system space, any **code that raises the privilege must disable interrupts** for the duration of the privileged operation.

  Example:

```
interrupt:
     move IGE, #0
     move PRIV, #HIGH
     … ; action
     move PRIV, #LOW
     move IGE, #1
     reti

system_code:
     move IGE, #0
     move PRIV, #HIGH
     ... ; action
     move PRIV, #LOW
     move IGE, #1
     ret
```

- An operation that requires high privilege levels **must not call subroutines to raise the privilege level**.

  Example:

```
incorrect:
     call raise_priv
     … ; action
     move PRIV, #LOW
correct:
     move PRIV, #HIGH
     … ; action
     move PRIV, #LOW
```

- A system library **function that checks arguments before raising the privilege level must do so in an atomic fashion** using PRIVT0 and PRIVT1 to prevent short-circuiting the check *(the rule about disabling interrupts also applies).*

  Example:

```
system_library:
     move IGE, #0
     move PRIVT0, #HIGH
     … ; check
     jump ne, exit            move PRIVT1, #HIGH
     ; … action
exit:
     move PRIV, #LOW
     move IGE, #1
     ret
```

2.6.2 Privilege Exception Interrupt

Any attempt to exceed the current privilege level causes a privilege exception interrupt that can be handled by system code. Examples that cause an interrupt are writing high to PRIV from user code, or trying to read system code while PRIV is low. The intent of the interrupt is to notify low priority code when an operation was denied by hardware.

2.6.3 Memory Access Protection Impact on Data Pointers (and Code Pointer)

Memory access protection complicates the use of the data and code pointers. In the MAXQ architecture, code pointers must be activated before use in order for memory data to be available on the same cycle it is needed using synchronous RAMs. This means that data is essentially prefetched into the physical data pointer when the pointer is activated (e.g., by loading an address to DP[0]). This can have some unintended consequences with respect to the memory protection function.

Specifically, when MPE is enabled, and when executing from RAM, any write to the traditional MAXQ data pointers, DP[0], DP[1], and BP, OFFS, or DPC, has the potential to generate a memory fault.

For example, a scenario in which code is executed from RAM is presented. In this particular case, the code is stored in a serial EEPROM. The code is loaded dynamically into RAM when needed. It is assumed this code has to have access to RAM variables, and remember we are executing from RAM.

To accomplish this without memory access protection, the customer would configure DPC and load DP[0] and then call the utility ROM function UROM_moveDP0. The code would look like the following:

```
MOVE  DPC, #REQUIRED_DP0_MODE       ; (1)
MOVE  DP[0], #REQUESTED_RAM_ADDRESS ; (2)
LCALL UROM_MOVEDP0                  ; (3)
     ; actual ROM function
     MOVE DP[0], DP[0]              ; (3a)
     MOVE GR, @DP[0]                ; (3b)
     RET                            ; (3c)
```

In the above example, (1) and (2) are both considered valid pointer activation instructions. In the MAXQ transfer-triggered architecture every standard instruction represents a MOVE from a source (SRC) to a destination (DST). The POP ACC instruction is equivalent to MOVE ACC, @SP--, JUMP LABEL is equivalent to MOVE IP, #LABEL, and so on. With the exception of a handful of arithmetic and logical instructions, every instruction is interpreted as a MOVE DST, SRC operation.

This is no different for instructions that operate on data pointers. For example, a pointer to pointer move such as MOVE @DP[1], @DP[0] first requires the read pointer to be activated. Architecturally, this strobes the chip enable and read signals on the memory mapped to the location in DP[0]. This value is latched internally so that it is available when @ DP[0] is used as the source operand. At that time, the internally latched data is transferred to the destination register.

This functions normally when memory protection is not enabled. However if MPE is set the same code can cause a memory protection fault. For this example let us assume the following:

1) The code is executing from RAM
2) REQUESTED_RAM_ADDRESS is defined as #0000h
3) Flash memory is located from 0000h–7FFFh

```
MOVE   DPC, #REQUIRED_DP0_MODE          ; Activates DP[0]
                                        ; In this MMU mapping,
                                        ; addresses 0-7FFFh are in Flash
                                        ; and *if* the previous contents
                                        ; of DP[0], modified by DPC, are
                                        ; in System space, we will generate
                                        ; a memory fault
MOVE   DP[0], #REQUESTED_RAM_ADDRESS    ; Again, activates DP[0]
                                        ; Now we know that DP[0]
                                        ; points to address 0000h
                                        ; and in the current MMU
                                        ; mapping, we are
                                        ; definitely pointing to
                                        ; *and reading from*
                                        ; System space in flash.
                                        ; MEMORY FAULT GUARANTEED
LCALL UROM_MOVEDP0                      ; Changes MMU mapping. In
                                        ; this case, addresses
                                        ; 0-7FFFh point to RAM
   ; actual ROM function
   MOVE DP[0], DP[0]                    ; ACTIVATE DP[0] in RAM
                                        ; space. If we studied
                                        ; the above discussion
                                        ; carefully, we know that
                                        ; *activate* means *read*
   MOVE GR, @DP[0]                      ; Transfer the latched
                                        ; DP[0] value to GR
   RET                                  ;
```

So, if MPE is enabled and the memory fault interrupt is enabled, the first two instructions generate a memory fault and the corresponding interrupt is executed. To avoid a memory fault under these circumstances, a function must be written in flash. This function has to take as an input, the address to be accessed, but it must be passed using a nonpointer register (such as an accumulator register). The RAM code routine would write the address into this register (e.g., A[0]).

Next, the RAM routine calls into the flash function. Once we are executing out of flash, we can activate the DP[0] pointer without causing a memory fault because the MMU now maps RAM into address range 0–7FFFh and ROM to higher addresses. None of this space is MPE protected. That flash routine would look similar to this:

```
  // this routine must be implemented in flash
ReadRAM:
   push DPC
   move DPC, #18h
   move DP[0], A[0]
   move A[0] @DP[0]
   pop  DPC
   ret
```

The corresponding RAM routine looks like:

```
;
; No pointer activation from RAM code
;
MOVE  A[0], #REQUESTED_RAM_ADDRESS
LCALL ReadRAM
```

2.6.4 Debugging

Note that debugging system code (including trace, break, memory dump, etc.) is disabled once memory protection is enabled.

2.6.5 Enabling Memory Protection

Memory protection is always enabled unless the system password is empty. Utility ROM initialization code is responsible for checking the password and clearing the memory protection enable (MPE) bit.

2.6.6 Reset Procedure and Setup of Memory Protection

Utility ROM code as well as system and user loader code is responsible for setting up the memory protection boundaries.

Both passwords and memory area boundary definitions are loaded from code memory. These values are part of the system, user loader, and user application image files, and are defined when assembling or compiling the code image files.

Example for the System Image:

```
org 0000h
      ; Reset
      move CP, #usr_ldr_page
      move ULDR, @CP
      jump sys_init
org 000Fh
user_ldr_page:
      ; Starting page address of user loader
      dw 0020h ; Page 32
org 0010h
      ; System password
      dw …, …, …, …
org 0020h
interrupt0:
```

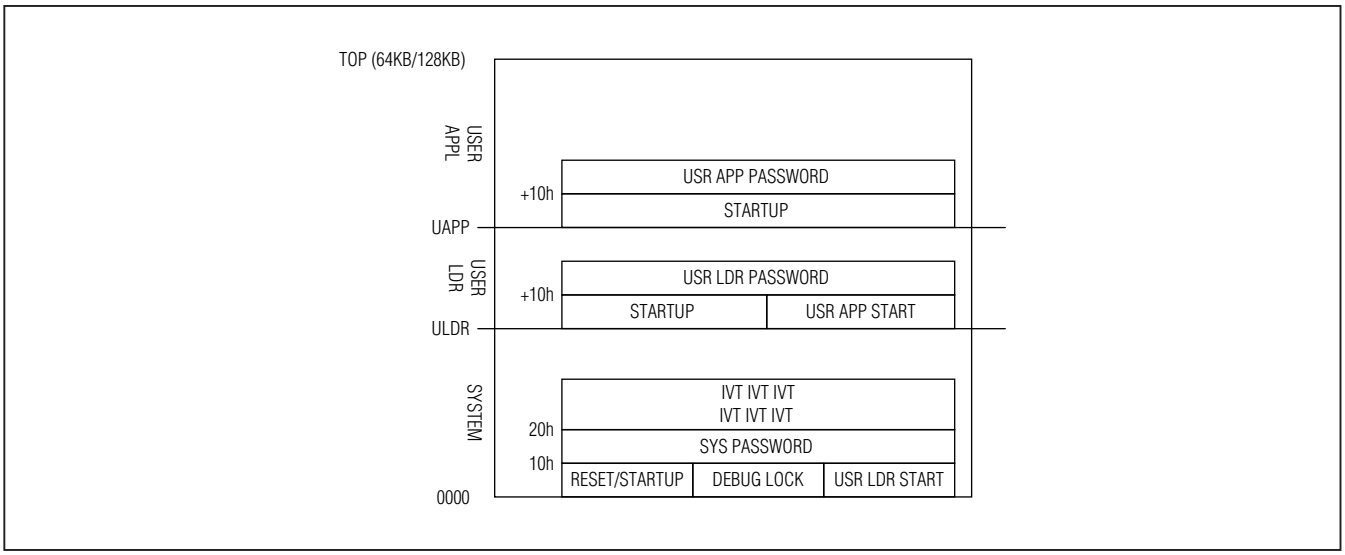


Figure 2-7. Overview of Memory Regions

Figure 2-7 shows the code memory with passwords and the location of the values that are programmed into the ULDR/UAPP registers.

The user loader starting page address is located at 0Fh, one word before the system password. The user application starting page address is stored one word before the user loader password (i.e., ULDR*Flash page size + 0Fh).

The startup sequence is as follows:

1) The system resets at 8000h and starts running utility ROM code. On a 64KB part with flash pages of 512 bytes, ULDR and UAPP are at their reset values of 80h (end of flash memory). The PRIV register is at its reset value of high. The MPE (memory protection enable bit) is at its reset value of 1 (enable).
2) Utility ROM initialization code checks the system password and disables MPE if the password is empty.
3) After utility ROM initialization is complete, the utility ROM passes execution to system code memory at address 0000h.
4) System code starts executing and uses a CP of 0Fh to read the user loader starting page address and writes it into the ULDR register.
5) After system initialization is complete, system code jumps to address ULDR*Flash page size + 0000h. This jump automatically drops PRIV to medium.
6) The user-loader code starts executing and uses a CP of ULDR*Flash page size + 0Fh to read the user application starting page address and writes it into the UAPP register.
7) After user loader initialization is complete, user-loader code jumps to address UAPP*Flash page size + 0000h. This jump automatically drops PRIV to low.

2.6.7 Loader Access Control

As stated previously, the MAXQ610 has three memory regions: system, user loader, and application. The loader maintains a context register to determine which of the regions is to be the target of the loader commands. Family 0 and Family F commands have no context. They are global in scope. For details on the nonparty-specific loader commands, refer to Application Note 4012: *Implementing a JTAG Bootloader Master for the MAXQ2000 Microcontroller*.

There are two Family F loader commands specific to the MAXQ610:

Command 0xF0: GetContext

Input : None

Output : Context Byte – 00x00, SystemContext; 0x01, LoaderContext; 0x2, ApplicationContext

Command 0xF1" SetContext

Input : Context Byte – 0x00, SystemContext; 0x01, LoaderContext; 0x02, ApplicationContext

Output : Sets Error Code (retrieved using Getstatus bootloader command)

The bootloader sets a default context based on the lowest privileged region that exists (see Figure 2-8). The default context is selected according to the following rules:

If all three regions exist:

The user application context (UAPP_CONTEXT).

If only system and user application regions exist:

The user application context.

If only system and user loader regions exist:

The user loader context (ULDR_CONTEXT).

If only the system region exists:

The system context (SYSTEM)CONTEXT).

Only the default context will have its password tested and corresponding PWL bit cleared. The context can be changed through the Family F commands shown above, but the password for the new region is not tested after a context change and, therefore, a password match loader command must be sent to clear the password lock bit of the associated region even if the password for that region is clear.

If the system password has not been set, memory protection is disabled by the ROM. If word address 000Eh in the system code region is programmed (any value other than 0xFFFF), the debug lockout condition is set by setting SC.DBGLCK to 1 (all debug functions are disabled).

The "current context" is used by the loader to determine how to apply master erase and password-protected loader commands. The master erase command erases pages starting at the base address of the current context and all pages with addresses greater than the base address. Password-protected commands check the password lock bit of the current region. The unlock password command uses the password from the current region (indicated by the current context) to determine the state of the current region password lock.

The loader provides several commands that require a password and a master erase command that does not.

All password-protected commands check the following:

- System password match: Access to full memory
- Loader password match: Access to user memory
- Application password match: Access to user application memory
- No match: No access

Three PWL bits allow the loader to find out whether a password match was successful. The PWL bits for system and user loader can only be written by utility ROM code (see *Section 13.2: Password-Protected Access*).

Master erase does not require a password and defaults to erasing the user application only. Two Family F commands are added that allow master erase of user loader and system code:

- Master erase system: Complete system erase.
- Master erase user loader: Erases user loader and user application.

2.6.8 Disabling MAXQ610-Specific Memory Access Features

The MAXQ610 memory-protection features are specific to the MAXQ610/69 family of parts and can cause some confusion in the way that they impact debugging and bootloader commands when compared to MAXQ parts. To enable users to develop initial firmware as quickly as possible, the following code can be added to your application code to disable the memory protection features and allow code loading and debugging in the same manner as previous parts:

```
ORG 0
 Jump Start

ORG 000eh
Debug_Lockout:
 DW 0ffffh ; disable debug lockout

ORG 000fh
ULDR_PageNumber:
 DW 0ffffh ; do not define a user loader page

ORG 0010h
System_PassworD:
 DW offffh,offffh, offffh,offffh, offffh,offffh, offffh,offffh
 DW offffh,offffh, offffh,offffh, offffh,offffh, offffh,offffh

ORG 0020h
; interrupt vectors go here

ORG 0100h
Start:
; Your application code here
;…
END
```

Once the memory-protection features are fully understood, this code can be removed from the user's application code to enable memory access control.

For ROM-only versions, the customer provides ULDR and UAPP with the ROM contents. This information is stored during manufacture.

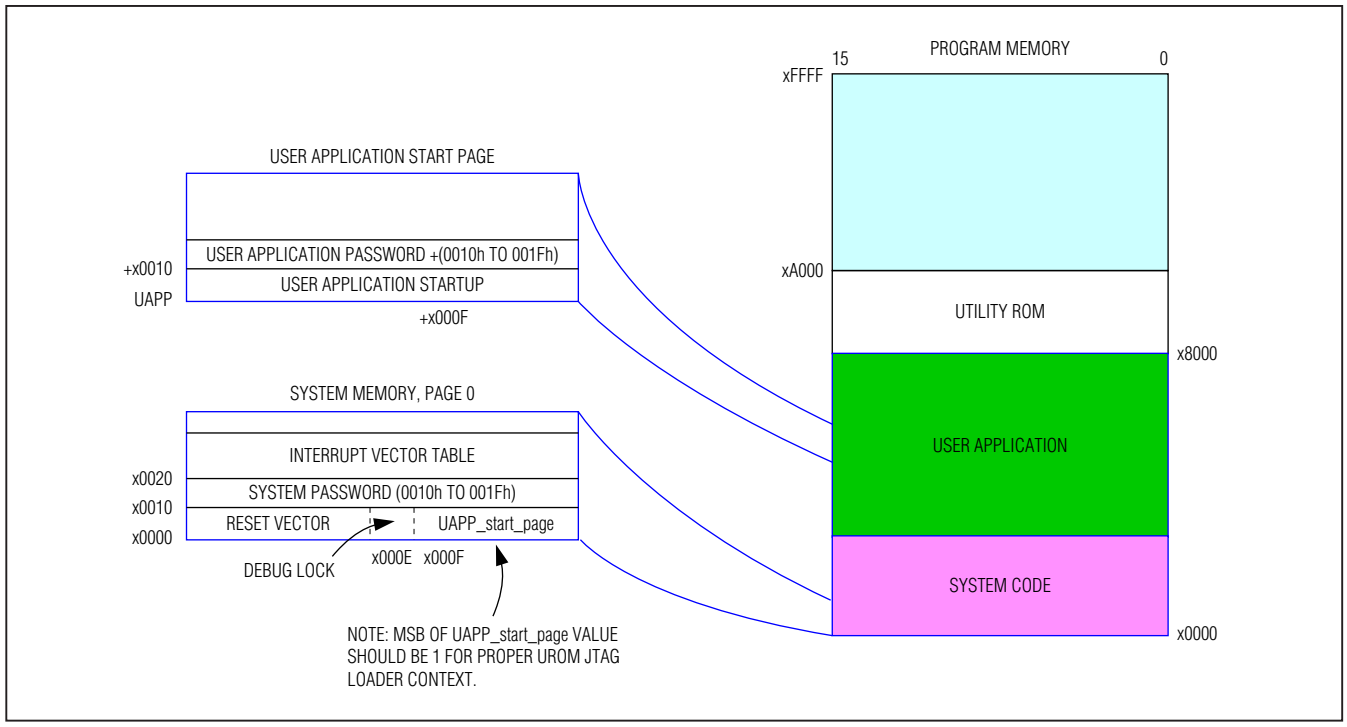


Figure 2-8. Program Memory Segmentation (Only Two Segments)

2.6.9 No User-Loader Segment

For devices with only two memory segments, the user-loader memory region is excluded, leaving only the system memory region and the user-application memory region. To support the two-segment memory configuration, the flash word at address 000Fh that is normally used to supply the value of the starting page for the user loader (for writing to ULDR) is used instead to supply a value for the starting page of the user application. For the utility ROM to contextually know that the aforementioned flash word is meant to supply information about the UAPP starting page instead of the ULDR starting page, the most significant bit in this word must be set to 1. Thus, if system code is to exclude the user loader, the following code would be used and the program memory segmentation map would change accordingly.

```
org 0000h
   ; Reset
   move CP, #usr_app_page
   move UAPP, @CP
   move ULDR, UAPP    ; set ULDR=UAPP
   jump sys_init
org 000Fh
user_app_page:
   ; Starting page address of user application (no user loader)
   dw 8020h ; Page 32, msbit=1
org 0010h
   ; System password
```

2.7 Clock Generation

All functional modules in the MAXQ610 are synchronized to a single system clock with the exception of the wakeup timer. The internal clock circuitry generates the system clock from one of two possible sources:

- Internal oscillator, using an external crystal or resonator
- External clock signal

The external clock and crystal are mutually exclusive since they are input through the same clock pin. Each time code execution must start or restart (as can be the case when exiting stop mode) using the external clock source, the following sequence occurs:

- Reset the crystal warmup counter.
- Allow the required warmup delay: 8192 external clock cycles if exiting from stop mode.
- Code execution starts after the crystal warmup sequence.

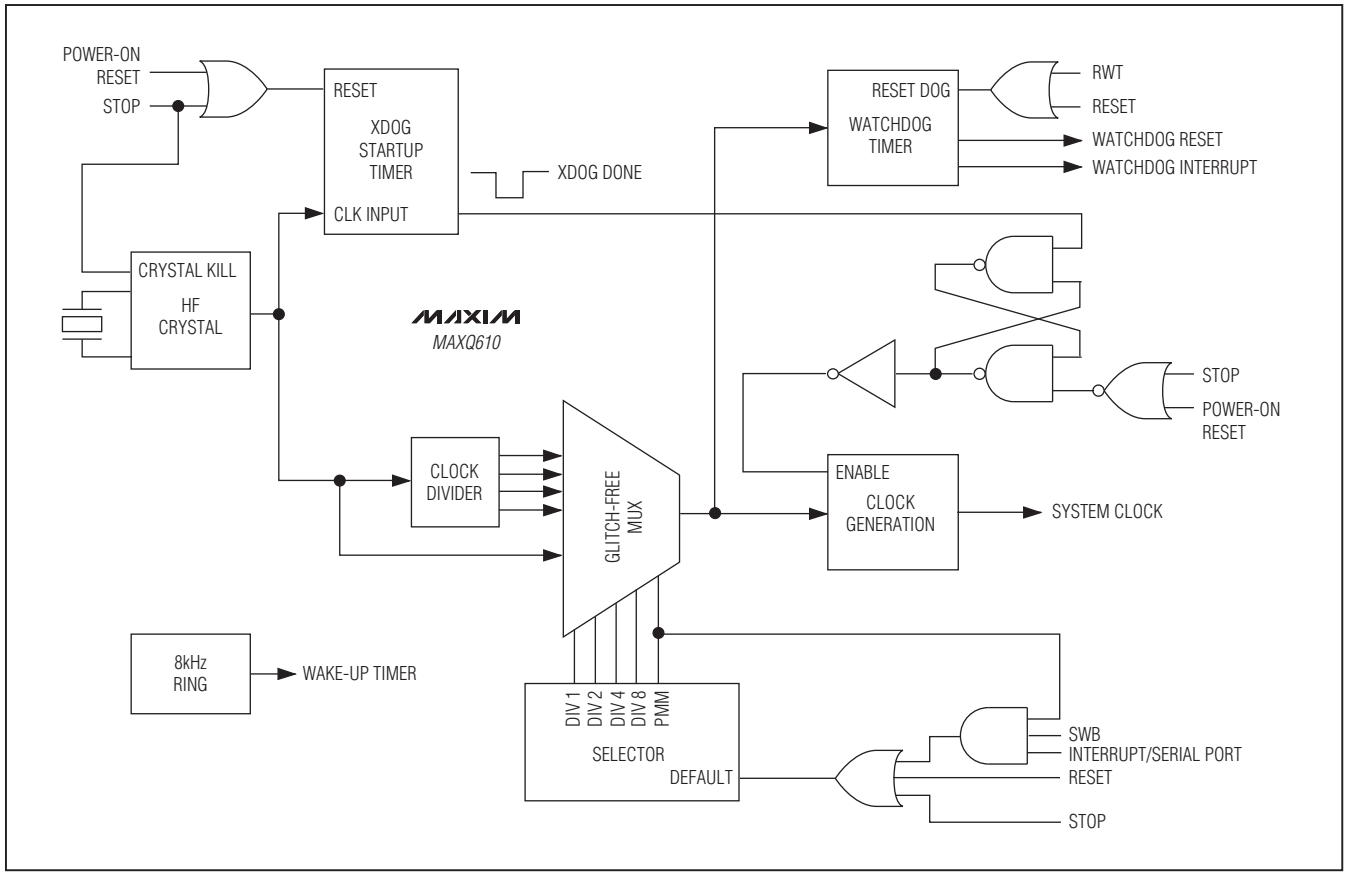


Figure 2-9. MAXQ610 Clock Sources

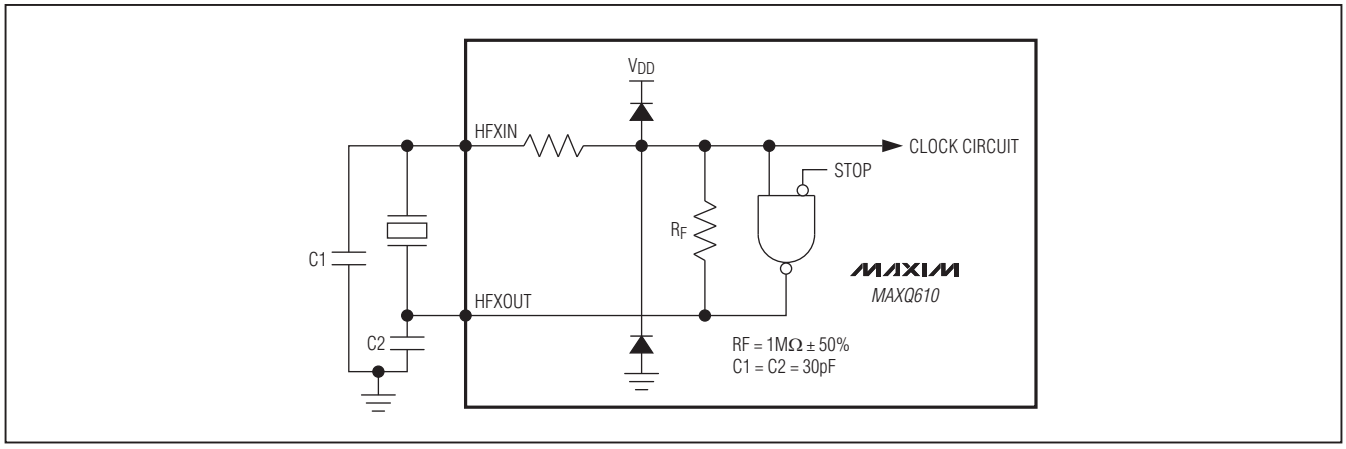


Figure 2-10. On-Chip Crystal Oscillator

2.7.1 External Clock (Crystal/Resonator)

An external quartz crystal or a ceramic resonator can be connected from HFXIN to HFXOUT determining the frequency, as illustrated in Figure 2-10. 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. The MAXQ610 is designed to operate at a 12MHz maximum frequency. To further reduce the effects of external noise, a guard ring can be placed around the oscillator circuitry.

Pins HFXIN and HFXOUT are protected by clamping devices against on-chip electrostatic discharge. These clamping devices are diodes parasitic to the feedback resistor R_F in the oscillator's inverter circuit. The inverter circuit is presented as a NAND gate that can disable clock generation in stop mode.

Noise at HFXIN and HFXOUT 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 HFXIN, HFXOUT, and ground with a 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 can require a different circuit configuration and capacitance value.

2.7.2 External Clock (Direct Input)

The MAXQ610 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 MAXQ610 from an external clock, connect the clock source to HFXIN and leave HFXOUT unconnected. The clock source should be driven through a CMOS driver. If the clock driver is a TTL gate, its output must be connected to V_{DD} 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.

2.7.3 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 according to Table 2-6.

Table 2-6. 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 Wake-Up Timer

The MAXQ610 provides a simple wake-up timer that can trigger an interrupt after a user-definable number of internal 8kHz ring cycles. Since the wake-up timer is running off the internal ring and keeps running even during stop mode, it can be used to wake the MAXQ610 up from stop mode at periodic intervals.

To use the wake-up timer, the WUT register should be written first (before the wake-up timer is started) to define the countdown interval. Once the time interval has been defined, the wake-up timer can be started by setting the WTE bit to 1. The time interval until the wake-up timer counts down to zero is defined by:

$$f_{NANO} \times WUT[15:0]$$

With the maximum possible time interval being:

$$f_{NANO} \times (2^{16} - 1)$$

2.8.1 Using the Wake-Up Timer to Exit Stop Mode

To use the wake-up timer to exit stop mode after a predefined period of time, the following conditions must be met before entering stop mode:

- The WUT register must be written to define the countdown interval value.
- The WTE bit must be written to 1 to start the wake-up timer.
- The IGE (IC.0) bit must be set to 1 to enable global interrupts. The wake-up timer cannot wake the MAXQ610 up from stop mode if its interrupt does not fire.

2.9 Interrupts

The MAXQ610 provides a hardware interrupt handler with interrupt vector (IV) table base address register and the interrupt control (IC) register. The IV register is fixed at 0020h and acts as the vector table base location. Interrupts can be generated from system level sources (e.g., watchdog timer) or by sources associated the peripheral modules. The interrupt vectors are preset at eight fixed memory address offsets from IV with hardware priority control that can be programmed through the interrupt priority register zero (IPR0).

2.9.1 Servicing Interrupts

For the MAXQ610 to service an interrupt, interrupt handling must be enabled globally and locally. The IGE bit located in the IC register acts as a global interrupt mask that affects all interrupts, with the exception of the power-fail warning interrupt. This bit defaults to 0, and it must be set to 1 before any interrupt handling 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. When an interrupt condition occurs, its individual flag

is set, even if the interrupt source is disabled at the local or global level. Interrupt flags must be cleared within the user interrupt routine to avoid repeated interrupts from the same source.

The handler uses three levels of interrupt priorities that allow the user software to select a suitable priority for an interrupt vector source. The interrupt handler (hardware) modifies the interrupt priority status bits (IPSn) when it is servicing an interrupt. These bits are set to 11b by the interrupt handler when executing a RETI instruction.

2.9.2 Interrupt System Operation

The interrupt handler 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 served 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 could also be accessing the stack, or that the current instruction could be a prefix (PFX) write. For this reason, the CPU must allow the current instruction to complete before pushing the stack and vectoring to the proper interrupt vector table address. If an interrupt exception window is generated by the currently executing instruction, the following instruction must be executed, thus the interrupt service routine is delayed an additional cycle.

Interrupt operation in the MAXQ610 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 IPS bits are set to the current interrupt level to prevent recursive interrupt calls from interrupts of lower priority.
4) The instruction pointer is set to the location of the interrupt service routine as defined by the interrupt source.
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 IPS bits are set to 11b 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 do not interrupt a RETI instruction; a new interrupt is serviced after first being acknowledged in the execution cycle that follows the RETI instruction and then after the standard one stall cycle of interrupt latency. This means there are at least two cycles between back-to-back interrupts.

2.9.3 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. Asynchronous interrupt sources are passed through a three-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, CD[1:0] controlled clock divide) such that the number of system clocks required to recognize an asynchronous interrupt request depend upon the system clock divide ratio:

- If the system clock-divide ratio is 1, the interrupt request is recognized after three system clocks.
- If the system clock-divide ratio is 2, the interrupt request is recognized after two system clocks.
- If the system clock divide ratio is 4 or greater, the interrupt request is recognized after one system clock.

An interrupt request with 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.

Table 2-7. Interrupt Priority

INTERRUPT	VECTOR ADDRESS (hex)	NATURAL PRIORITY	FLAG	ENABLE*	PRIORITY CONTROL
Power Fail	20h	0	PFI (PWCN.2)	PFIE (PWCN.1)	IPV0[1:0] (IPR0[1:0])
Memory Fault	28h	1	PULRF (IC.4), PULWF (IC.5), PSYRF (IC.6), PSYWF (IC.7)	MPE (SC.10)	IPV1[1:0] (IPR0[3:2])
External INT[7:0]	30h	2	IE[7:0] (EIF0)	EX[7:0] (EIE0)	IPV2[1:0] (IPR0[5:4])
IR Timer	38h	3	IROV (IRCNB.0), IRIF (IRCNB.1)	IRIE (IRCNB.2)	IPV3[1:0] (IPR0[7:6])
Serial Port 0	40h	4	RI (SCON0.0), TI (SCON0.1)	ESI (SMD0.2)	IPV4[1:0] (IPR0[9:8])
Serial Port 1			RI (SCON1.0), TI (SCON1.1)	ESI (SMD1.2)	
SPI	48h	5	MODF (SPICN.3), WCOL (SPICN.4), ROVR (SPICN.5), SPIC (SPICN.6)	ESPII (SPICF.7)	IPV5[1:0] (IPR0[11:10])
External INT[15:8]			IE[15:8] (EIF1)	EX15[7:8] (EIE1)	
Timer B0	50h	6	TFB (TBOCN.7), EXFB (TB0CN.6)	ETB (TB0CN.1)	IPV6[1:0] (IPR0[13:12])
Timer B1			TFB (TB1CN.7), EXFB (TB1CN.6)	ETB (TB1CN.1)	
Wake-Up Timer	58h	7	WTF (WUTC.1)	WTE (WUTC.0)	IPV7[1:0] (IPR0[15:14])
Watchdog Timer			WDIF (WDCN.3)	EWDI (WDCN.6)	
Unhandled Interrupt	98h	8	None	None	None

**With the exception of the power-fail interrupt, all interrupts require that the IGE bit be set to 1 to generate an interrupt request, regardless of the individual interrupt enable listed. The power-fail interrupt is not governed by IGE (i.e., interrupt request generation is controlled solely by the PFIE enable bit).*

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 vs. 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.

2.9.4 Interrupt Prioritization by Software

There are three levels of interrupt priorities: level 0 to 2. Level 0 is the highest priority and level 2 is the lowest. All interrupts have individual priority bits in the IPR0 register to allow each interrupt to be assigned a priority level. All interrupts have a natural priority or hierarchy. In this manner, when a set of interrupts has been assigned the same priority, this natural priority hierarchy determines which interrupt is allowed to take precedence if multiple interrupts occur simultaneously. The natural hierarchy is determined by analyzing potential interrupts in a sequential manner with the preferred order as listed in Table 2-7. Once an interrupt is being processed, only an interrupt with higher priority level can preempt it. Therefore, the MAXQ610 supports a maximum of two levels of interrupt nesting.

For example, suppose three interrupts occur simultaneously and the assigned priorities (IPV bits) for each of the interrupt sources are as follows:

- IR Timer: assigned priority level 1

- Serial Port 0: assigned priority level 2
- Timer B0: assigned priority level 2

Because simultaneous interrupts are first evaluated according to assigned priority level, the IR timer interrupt is serviced first. Once the IR timer interrupt source has been cleared, the serial port 0 and timer B0 interrupt sources are evaluated. Both of these interrupt sources have been assigned to the same priority level (level 2), so the natural priority of each source is used to determine which is serviced first. The serial port 0 interrupt is serviced first as its natural priority is 4, whereas timer B0 has natural priority 6. If two interrupts that are grouped under the same natural priority occur simultaneously, the order in which handling of the interrupts occurs is left to the discretion of user code (i.e., user code must decide what order to check the associated interrupt flags).

For an unhandled interrupt, the interrupt handler vectors to flash address 0x98 if the user disables any of the interrupts when an interrupt is triggered or when a medium priority interrupt occurs while in stop mode. A simple "RETI" is required to be placed at 0x98.

2.9.5 Interrupt Exception Window

An interrupt exception window is a noninterruptible 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. There are two conditions in the MAXQ610 microcontroller that cause an interrupt exception window:

- Activation of the prefix register (PFX)
- Code memory access using the code pointer (CP)

When the prefix register (PFX) 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.

The one-cycle stall when using the code pointer is due to the fact that the current instruction could also be accessing the stack.

If an interrupt occurs during an interrupt exception window, an additional latency of one cycle in the interrupt handling is caused as the interrupt is not serviced until the next cycle.

2.10 Operating Modes

In addition to the standard program execution mode, the MAXQ610 can also be in three other operating modes. 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 in order to decrease power consumption. Finally, stop mode halts execution and all internal clocks (with the exception of the wake-up timer if enabled) to save power until an external stimulus indicates that processing should be resumed.

2.11 Reset Mode

When the MAXQ610 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 from utility ROM at address 8000h.

There are four different sources that can cause the MAXQ610 to enter reset mode:

- Power-on/power-fail reset
- External reset
- Watchdog timer reset
- Internal system reset

2.11.1 Power-On/Power-Fail Reset

An on-chip power-on reset (POR) circuit is provided to ensure proper initialization on internal device states. The POR circuit provides a minimum POR delay sufficient to accomplish this initialization. For fast V_{DD} supply rise times, the MAXQ610 is, at a minimum, held in reset for the POR delay when initially powered up. For slow V_{DD} supply rise times, the MAXQ610 is held in reset until V_{DD} is above the POR voltage threshold.

The MAXQ610 supports power-fail detection where an on-chip bandgap and reference comparator constantly monitor the supply voltage V_{DD} to ensure that it is within acceptable limits. If V_{DD} is below the power-fail level warning level, an interrupt is generated to the CPU if enabled. If V_{DD} falls further to below the operating condition, the power monitor initiates a reset condition. This can occur either when the MAXQ610 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 ($V_{DD} > V_{RST}$), there is a delay of 8192 oscillator cycles until execution resumes to ensure that the clock source has stabilized.

Rather than leaving the power-fail reset monitoring circuit always on once the V_{RST} condition has occurred, it can be advantageous to the application to conserve battery capacity during power-fail reset in order to extend the time until POR is reached (and possibly retaining SRAM contents). While there is still no single bit indicator that can be used to guarantee SRAM retention once power-fail reset has occurred, one possibility is that the user can perform a checksum over the area for which retention is questioned to make this assessment. So, in order to reduce current consumption during the power-fail reset state, two power-fail reset check time configuration bits (PFRCK[1:0]) are provided for the user. These bits are used to enable duty cycling of the V_{RST} power-monitoring circuitry during the time when V_{DD} is below the V_{RST} threshold but has not reached the POR threshold. These bits are reset only by POR (not even V_{RST}). Table 2-8 provides the bit settings and corresponding duty cycling of the power monitor check when $V_{POR} < V_{DD} < V_{RST}$. Note that the V_{POR} state for the bits is 00b, which results in the monitor being on always.

Table 2-8. Power-Fail Reset Check Interval

PFRCK[1:0]	POWER-FAIL MONITOR CHECK INTERVAL (NANOPOWER RING OSCILLATOR CYCLES)
00	No interval defined (Monitor on always as normal)
01	2^{10} (~ 128ms for 8kHz nanopower ring oscillator frequency)
10	2^{11} (~ 256ms for 8kHz nanopower ring oscillator frequency)
11	2^{12} (~ 512ms for 8kHz nanopower ring oscillator frequency)

During the power-fail reset condition, duty cycling of the V_{RST} power-monitoring circuitry results in reduced current that can be approximated by the following equation:

$$I_{POWERFAIL} = (3 \times I_{S2} + (\text{Check Interval Cycles} - 3) \times (I_{S1} + I_{NANO}))/\text{Check Interval Cycles}$$

where:

I_{S1} = stop-mode current with power-fail monitor off

I_{S2} = stop-mode current with power-fail monitor on

I_{NANO} = nanopower ring oscillator current

When the processor exits from the power-on/power-fail 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.

The power-fail monitor is always on during normal operation. However, it can be selectively disabled during stop mode using the power-fail monitor disable (PFD) bit in the PWCN register if the regulator is also selectively disabled (REGEN = 0) during stop mode. If the user opts to leave the regulator on during stop mode, the power-fail monitor is automatically left enabled as well, regardless of the state of the PFD bit. The reset default state for the PFD bit is 0, which enables the power-fail monitor function during stop mode. If power-fail monitoring is disabled (PFD = 1) during stop mode, the circuitry responsible for generating a power-fail warning or reset is shut down and neither condition is detected. Thus, the $V_{DD} < V_{RST}$ condition does not generate a reset. However, in the event that V_{DD} falls below the

POR level, a POR is generated. The power-fail monitor is enabled prior to the stop mode exit and before code execution begins. If a power-fail warning condition ($V_{DD} < V_{PFW}$) is then detected, the power-fail interrupt flag is set on stop mode exit. If a power-fail reset condition is detected ($V_{DD} < V_{RST}$), the CPU goes into reset.

2.11.2 External Reset

During normal operation, the MAXQ610 device is placed into external reset mode by holding the $\overline{\text{RESET}}$ pin low for at least four clock cycles. If the MAXQ610 is in the low-power stop mode (i.e., system clock is not active), the $\overline{\text{RESET}}$ pin becomes an asynchronous source, forcing the reset state immediately after being taken low. Once the MAXQ610 enters reset mode, it remains in reset as long as the $\overline{\text{RESET}}$ pin is held low. After the $\overline{\text{RESET}}$ pin returns to high, the processor exits the reset state within four clock cycles and begins program execution from utility ROM at address 8000h.

The $\overline{\text{RESET}}$ pin is an output as well as an input. If a reset condition is caused by another source (such as a power-fail reset or internal reset), an output reset pulse is generated at the $\overline{\text{RESET}}$ pin for as long as the MAXQ610 remains in reset. If the $\overline{\text{RESET}}$ 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.

2.11.3 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 initiates a reset state.

If the watchdog resets the processor, it remains in reset for four clock cycles. Once the reset condition is removed, the processor begins executing program code from utility ROM 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.

2.11.4 Internal System Reset

The MAXQ610 can incorporate functions that logically warrant the ability to generate an internal system reset. This reset generation capability is assessed by MAXQ610 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 will want the ability to reset the system. This internal (software-triggered) reset generation capability is possible following in-system programming.

2.12 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 can 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 could be the case that a single application can 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 occurs 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 is also cleared automatically to 0 by any reset condition.

To avoid data loss, the PMME bit cannot be set while the USART or SPI ports are either transmitting or receiving, or while an external interrupt is waiting to be serviced. Attempts to set the PMME bit under these conditions result in a no-op.

2.12.1 Switchback

When power-management mode is active, the MAXQ610 operates at a reduced clock rate. Although execution continues as normal, peripherals that base their timing on the system clock such as the USART module and the SPI module might 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 INTn pin and the corresponding external interrupt is enabled.
- An active-low transition occurs on the USART 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 USART 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.
- Active debug mode is entered either by breakpoint match or issuance of the debug command from background mode.
- Power-fail interrupt if enabled (PFIE = 1).

2.13 Stop Mode

When the MAXQ610 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 the lowest power stop mode 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.

Note: It is necessary to include a 'nop' immediately following the instruction to invoke stop mode for proper interrupt operation. Example code is as follows:

```
move ckcn, #010h     ; enter stop mode
nop                  ; No operation to cause a one cycle delay
```

The MAXQ610 exits stop mode when any of the following conditions occur:

- An external interrupt condition occurs on one of the INTn 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 $\overline{\text{RESET}}$ pin. After the reset signal is removed, execution resumes from utility ROM at 8000h as it would after any reset state.
- A power-fail interrupt occurs, if enabled (PFIE = 1).
- A wake-up timer interrupt occurs, if enabled (WTE = 1).

Note that the voltage monitor and bandgap reference can be disabled during stop mode to conserve current consumption. In this case, a power-fail condition does not cause a reset as it would under normal conditions. However, the POR monitor remains enabled, and any voltage drop on V_{DD} that goes below the POR level causes a POR to occur. To continue to monitor supply voltage during stop mode, the power-fail monitor is left on if the regulator is left on (REGEN = 1), or it can be explicitly enabled (if the regulator is disabled; REGEN = 0) by clearing the PWCN.PFD bit to 0. The power-fail monitor is always enabled prior to stop mode exit and resumption of code execution.

Once the processor exits stop mode, it resumes execution as follows:

- If the crystal oscillator is selected as the system clock source, the crystal oscillator is started and execution resumes following an 8192-clock-cycle delay to allow the oscillator frequency to stabilize.

SECTION 3: PROGRAMMING

This section contains the following information:

3.1 Addressing Modes 3-3
3.2 Prefix Operations 3-3
3.3 Reading and Writing Registers 3-4
3.3.1 Loading an 8-Bit Register with an Immediate Value 3-4
3.3.2 Loading a 16-Bit Register with a 16-Bit Immediate Value 3-4
3.3.3 Moving Values Between Registers of the Same Size 3-4
3.3.4 Moving Values Between Registers of Different Sizes 3-5
3.3.5 8-Bit Destination ← Low Byte (16-Bit Source) 3-5
3.3.6 8-Bit Destination ← High Byte (16-Bit Source) 3-5
3.3.7 16-Bit Destination ← Concatenation (8-Bit Source, 8-Bit Source) 3-5
3.3.8 Low (16-Bit Destination) ← 8-Bit Source 3-6
3.3.9 High (16-Bit Destination) ← 8-Bit Source 3-6
3.4 Reading and Writing Register Bits 3-6
3.5 Using the Arithmetic and Logic Unit 3-7
3.5.1 Selecting the Active Accumulator 3-7
3.5.2 Enabling Autoincrement and Autodecrement 3-7
3.5.3 ALU Operations Using the Active Accumulator and a Source 3-9
3.5.4 ALU Operations Using Only the Active Accumulator 3-9
3.5.5 ALU Bit Operations Using Only the Active Accumulator 3-10
3.5.6 Example: Adding Two 4-Byte Numbers Using Autoincrement 3-10
3.6 Processor Status Flag Operations 3-10
3.6.1 Sign Flag 3-10
3.6.2 Zero Flag 3-11
3.6.3 Equals Flag 3-11
3.6.4 Carry Flag 3-11
3.6.5 Overflow Flag 3-12
3.7 Controlling Program Flow 3-12
3.7.1 Obtaining the Next Execution Address 3-12
3.7.2 Unconditional jumps 3-12
3.7.3 Conditional Jumps 3-13
3.7.4 Calling Subroutines 3-13
3.7.5 Loop Operations 3-13
3.7.6 Conditional Returns 3-14
3.7.7 Conditional Return from Interrupt 3-15
3.8 Accessing the Stack 3-15
3.9 Accessing Data Memory 3-16
3.10 Using the Watchdog Timer 3-18

LIST OF FIGURES

Figure 3-1. Watchdog Timer Block Diagram 3-19

LIST OF TABLES

Table 3-1. Accumulator Pointer Control Register Settings 3-8
Table 3-2. Watchdog Timer Register Control Bits 3-18
Table 3-3. Watchdog Timeout Period Selection 3-20

SECTION 3: PROGRAMMING

This section provides a programming overview of the MAXQ610. For full details on the instruction set as well as the system register and peripheral register detailed bit descriptions, see the appropriate sections later in this document.

3.1 Addressing Modes

The instruction set for the MAXQ610 provides three different addressing modes: direct, indirect, and immediate.

System and peripheral registers are referenced by direct addressing only. This addressing mode is used to specify both source and 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 Prefix Operations

All instructions on the MAXQ610 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 are 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 bits. This is because the prefix value defaults to zero, so the following line is not required:

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

3.3 Reading and Writing Registers

All functions in the MAXQ610 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 writable 8-bit register with a subindex 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
```

Writable 8-bit registers with subindexes 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 writable 16-bit register with a subindex from 0h to 7h 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 subindex 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[2], #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 0Fh.

```
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 0Fh, 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 is 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.

3.3.5 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 the single-cycle write or read regions, 0 to 7h and 0 to 0Fh, respectively.

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

3.3.6 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
```

3.3.7 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 could 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], hig      ; PFX=00:high
move dst, low         ; dst=high:low
```

3.3.8 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
```

3.3.9 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 modules 0h to 5h or a system register in module 8h. The set or clear operation does 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 is written with the prefix data and the low byte is written with the bit mask (i.e., all zeros with a single one for the set bit operation or all ones with a single zero 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 can also be used to transfer any one of the lowest 8 bits from a register source or any active accumulator (Acc) bit to the carry flag. There is no restriction on the source register module for the "MOVE C, src.bit" instruction.

```
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 MAXQ610 provides a 16-bit arithmetic and logic unit (ALU) that allows operations to be performed between the active accumulator and any other register. The default ALU configuration provides 16 accumulator registers that are 16-bit wide, of which any one can be selected as the active accumulator.

3.5.1 Selecting the Active Accumulator

Any of the 16 accumulator registers A[0] to A[15] can be selected as the active accumulator by setting the low 4 bits of the accumulator pointer register (AP) to the index of the accumulator register the users wants 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] = 0055 hex
move AP, #00h         ; select A[0] as active accumulator
move Acc, #55h        ; set A[0] = 0055 hex
```

3.5.2 Enabling Autoincrement and Autodecrement

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 autoincrement/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 (Bitwise complement active accumulator)
- NEG (Negate active accumulator)
- SLA (Arithmetic shift left on active accumulator)
- SLA2 (Arithmetic shift left active accumulator 2 bit positions)
- SLA4 (Arithmetic shift left active accumulator 4 bit positions)
- SRA (Arithmetic shift right on active accumulator)
- SRA2 (Arithmetic shift right active accumulator 2 bit positions)

- SRA4 (Arithmetic shift right active accumulator 4 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)
- XCH (Exchange active accumulator bytes)

The active accumulator cannot 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 autoincrement or autodecrement function takes place, even if this function is enabled. Note that the active accumulator cannot be the destination for the MOVE dst, A[AP] instruction (i.e., MOVE Acc, A[AP] is prohibited).

So, the following two instructions are equivalent, except that the first instruction triggers autoincrement/decrement (if it is enabled), while the second one would never do so.

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

The accumulator pointer control register (APC) is used to control the automatic increment/decrement mode as well as select the range of bits (modulo) in the AP register that are to be incremented or decremented. There are 10 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	AUTOINCREMENT/DECREMENT SETTING
0	0	0	0	00h	No autoincrement/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
mov       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
```

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 cannot be used as the source for any of these operations.

```
add       A[4]         ; Acc = Acc + A[4]
addc      #32h         ; Acc = Acc + 0032h + Carry
sub       A[15]        ; Acc = Acc – A[15]
subb      A[1]         ; Acc = Acc – A[1] - Carry
cmp       #00h         ; If (Acc == 0000h), set Equals flag
and       A[0]         ; Acc = Acc AND A[0]
or        #55h         ; Acc = Acc OR #0055h
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 4 times
sr                     ; Shift accumulator right, set Carry to Acc.0,
                       ; set Acc.15 to zero
```

```
sra                 ; Shift accumulator right arithmetically once
sra2                ; Shift accumulator right arithmetically twice
sra4                ; Shift accumulator right arithmetically 4 times
xchn                ; Swap low and high nibbles of each Acc byte
xch                 ; 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 can use an Acc bit from 0 to 7.

```
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 autoincrement, autodecrement, or modulo operations defined by the accumulator pointer control (APC) register.

3.5.6 Example: Adding Two 4-Byte Numbers Using Autoincrement

```
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 four flags that are used to indicate and store the results of arithmetic and logical operations as well as control 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.15. If signed arithmetic is being used, this flag indicates whether the value in the accumulator is positive or negative.

Because 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 autoincrement/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.

Because 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. Any instruction that changes which accumulator is the active one (including AP autoincrement/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. 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)
- 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 MAXQ610 provides several options to control program flow and branching. Jumps can be unconditional, conditional, relative, or absolute. Subroutine calls store the return address on the soft 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 IP register. This can be particularly useful for initializing loops, as shown in the following sections. Note that the value returned is actually the address of the current instruction plus 1, so this is 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 either 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 can 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 the jump is made. 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 Loop Operations

Looping over a section of code can, of course, 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 desired count 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
   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 does not need 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 MAXQ610 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
```

3.7.7 Conditional Return from Interrupt

Similar to the conditional returns, the MAXQ610 microcontroller also supports a set of conditional return from interrupt operation. Based upon the value of one of the status flags, the CPU can conditionally pop the stack, set the IPS bits to 11b, and begin execution at the address popped from the stack. If the condition is not true, the conditional return from interrupt instruction leaves the IPS bits 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
```

3.8 Accessing the Stack

The soft 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 increases the stack depth (by decrementing 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 decreases the stack depth (by incrementing 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 sets the IPS bits to 11b'. Thus, the following two instructions would be equivalent:

```
popi IP
reti
```

The @SP-- mnemonic can be used by the MAXQ610 so that stack values can 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
add  @SP--            ; with Acc, disregarding overflow
add  @SP--
```

The stack pointer SP can be set explicitly, however, only the least significant bits needed to represent the stack depth are used. The MAXQ610 has a stack depth constrained only by the size of the SRAM, and the lowest 10 bits of SP are used. Setting SP to 03F0h returns it to its reset state.

On the MAXQ610, the stack naturally grows downward from the top of the SRAM. A push operation (move @++SP, …) increases the depth of the stack, but decreases the numeric value in the stack pointer. A pop (move …, @SP--) decreases the depth of the stack, but decreases the numeric value in the stack pointer.

Because 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
```

3.9 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 can be postincremented or postdecremented following any read, or can be preincremented or predecremented 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 can be preincremented/decremented when writing to data memory, and can be postincremented/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 register (DPC). 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 64KB of memory.

Each data pointer 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 is 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, autoincrement/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 used 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 (SDPS[1:]0; DPC[1:0]) or implicitly by writing to the DP[n], BP, or OFFS register, as shown below. Additionally, any write operation sets the SDPS bits, thereby activating the write pointer as the active source pointer.

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

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

- The source data-pointer select bits are changed through 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 data-memory write operation sets the SDPS and activates the pointer used for writing as the 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, subindex 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++]
```

3.10 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 shown in Figure 3-1, the main system clock drives the timer, which 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 (through the WD[1:0] bits) and the 512-clock reset counter are reset if either the interrupt or reset function is enabled. When the watchdog timer is initially enabled, there is a one- to three-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.

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 resets/restarts 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 1) before the timeout period expires, the timer starts over. Hardware automatically clears RWT after software sets it.

Table 3-2. Watchdog Timer Register Control Bits

BIT NAME	DESCRIPTION	REGISTER LOCATION	BIT POSITION
EWDI	Enable Watchdog Timer Interrupt	WDCN (0Fh, 8h)	WDCN.6
WD[1:0]	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

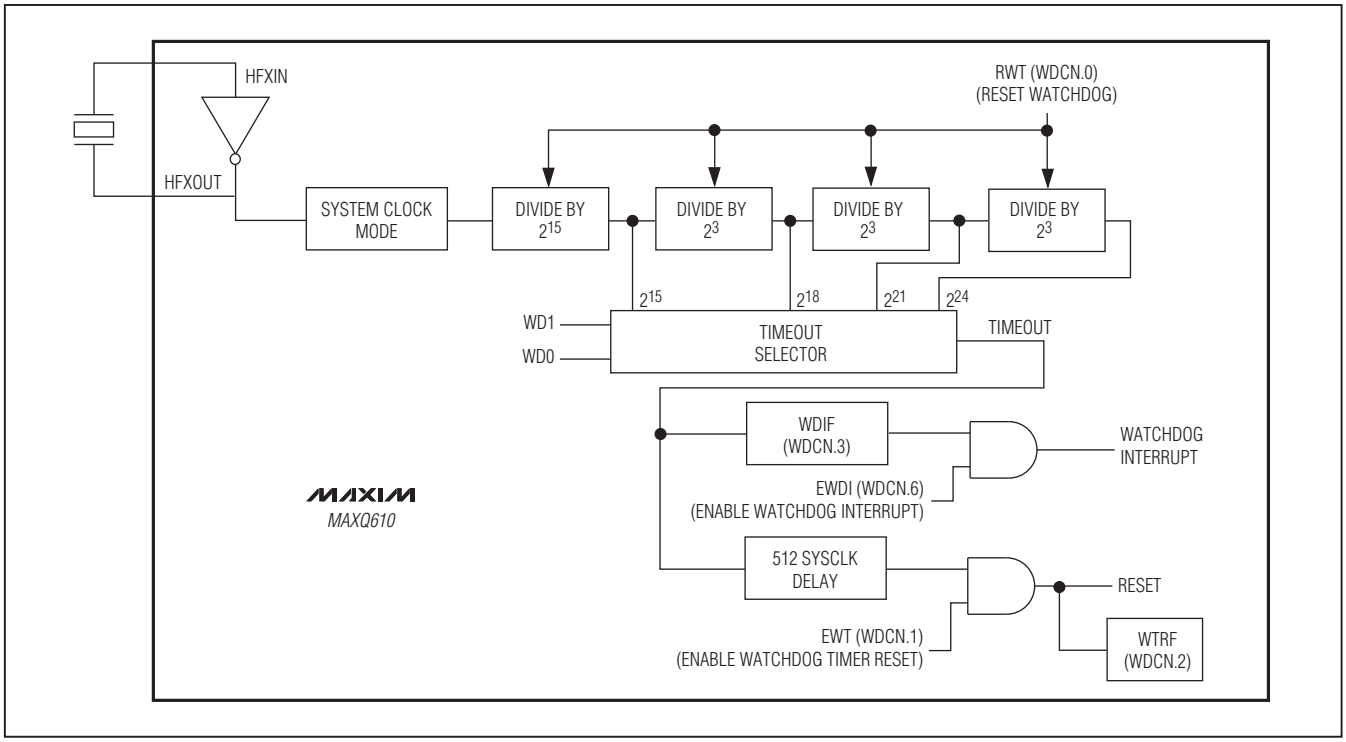


Figure 3-1. Watchdog Timer Block Diagram

If the timeout is reached without RWT being set, hardware generates 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 can reset the CPU if EWT = 1. When the reset occurs, the watchdog timer reset flag (WTRF = WDCN.2) is automatically 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 sets the WDIF bit (WDCN.3), and an interrupt occurs if the interrupt global enable (IGE = IC.0) is set and either 1) the interrupt priority status (IPS[1:0]) bits are set to 11b (idle), or 2) the watchdog timer interrupt is configured to higher priority than an interrupt currently being serviced. Note that WDIF is set 512 system clocks before a potential watchdog reset. The watchdog interrupt flag indicates 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) allows 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 can 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 solely 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 Figure 3-1. Because 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. Table 3-3 shows a summary of the selectable watchdog timeout intervals for the various system clock modes and WD[1:0] control bit settings. If enabled, the watchdog reset is always scheduled to occur 512 system clocks following the timeout. Watchdog-generated resets last for eight system clock cycles.

Table 3-3. Watchdog Timeout Period Selection

SYSTEM CLOCK MODE	SYSTEM CLOCK SELECT BITS			WATCHDOG TIMEOUT (IN NUMBER OF OSCILLATOR CLOCKS)			
	PMME	CD1	CD0	WD[1:0] = 00b	WD[1:0] = 01b	WD[1:0] = 10b	WD[1:0] = 11b
Divide by 1 (default)	0	0	0	2^{15}	2^{18}	2^{21}	2^{24}
Divide by 2	0	0	1	2^{16}	2^{19}	2^{22}	2^{25}
Divide by 4	0	1	0	2^{17}	2^{20}	2^{23}	2^{26}
Divide by 8	0	1	1	2^{18}	2^{21}	2^{24}	2^{27}
Power-Management Mode (Divide by 256)	1	x	x	2^{23}	2^{26}	2^{29}	2^{32}

SECTION 4: SYSTEM REGISTER DESCRIPTION

This section contains the following information:

4.1 System Register Descriptions 4-5

LIST OF TABLES

Table 4-1. System Register Map 4-2
Table 4-2. System Register Bit Map 4-3
Table 4-3. System Register Bit Reset Values 4-4

SECTION 4: SYSTEM REGISTER DESCRIPTION

Registers currently defined in the MAXQ610 system register map are described in Tables 4-1, 4-2, and 4-3.

Table 4-1. System Register Map

	MODULE SPECIFIER									
REGISTER INDEX WITHIN MODULE	**06h**	**07h**	**08h**	**09h**	**0Ah**	**0Bh**	**0Ch**	**0Dh**	**0Eh**	**0Fh**
00h	—	—	AP	**A[0]**	**Acc**	**PFX[0]**	**IP**	—	—	—
01h	—	—	APC	**A[1]**	***A[AP]***	**PFX[1]**	—	**SP**	—	—
02h	—	—	PRIV	**A[2]**	—	**PFX[2]**	—	***IV***	—	—
03h	—	—	PRIVT0	**A[3]**	—	**PFX[3]**	—	—	OFFS	**DP[0]**
04h	—	—	PSF	**A[4]**	—	**PFX[4]**	—	—	**DPC**	—
05h	—	—	IC	**A[5]**	—	**PFX[5]**	—	—	**GR**	—
06h	—	—	PRIVT1	**A[6]**	—	**PFX[6]**	—	**LC[0]**	GRL	—
07h	—	—	—	**A[7]**	—	**PFX[7]**	—	**LC[1]**	**BP**	**DP[1]**
08h	—	—	**SC**	**A[8]**	—	—	—	—	***GRS***	—
09h	—	—	**IPR0**	**A[9]**	—	—	—	—	GRH	—
0Ah	—	—	—	**A[10]**	—	—	—	—	*GRXL*	—
0Bh	—	—	PRIVF	**A[11]**	—	—	—	—	***FP***	**CP**
0Ch	—	—	ULDR	**A[12]**	—	—	—	—	—	—
0Dh	—	—	UAPP	**A[13]**	—	—	—	—	—	—
0Eh	—	—	CKCN	**A[14]**	—	—	—	—	—	—
0Fh	—	—	WDCN	**A[15]**	—	—	—	—	—	—
10h	—	—	—	—	—	—	—	—	—	—
11h	—	—	—	—	—	—	—	—	—	—
12h	—	—	—	—	—	—	—	—	—	—
13h	—	—	—	—	—	—	—	—	—	—
14h	—	—	—	—	—	—	—	—	—	—
15h	—	—	—	—	—	—	—	—	—	—
16h	—	—	—	—	—	—	—	—	—	—
17h	—	—	—	—	—	—	—	—	—	—
18h	—	—	—	—	—	—	—	—	—	—
19h	—	—	—	—	—	—	—	—	—	—
1Ah	—	—	—	—	—	—	—	—	—	—
1Bh	—	—	—	—	—	—	—	—	—	—
1Ch	—	—	—	—	—	—	—	—	—	—
1Dh	—	—	—	—	—	—	—	—	—	—
1Eh	—	—	—	—	—	—	—	—	—	—
1Fh	—	—	—	—	—	—	—	—	—	—

Note 1: *Register names that appear in italics indicate read-only registers. Register names that appear in bold indicate 16-bit registers.*

Note 2: *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.*

Note 3: *All undefined or unused indexes (indicated by a "—") 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 are probably not the intended ones.*

Table 4-2. System Register Bit Map

REG	BIT															
	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
PRIV									—	—	—	—	PSYW	PSYR	PULW	PULR
PRIVT0									—	—	—	—	PRIVT0 (4 bits)			
PSF									Z	S	—	GPF1	GPF0	OV	C	E
IC									—	—	—	—	IPS1	IPS0	—	IGE
PRIVT1									—	—	—	—	PRIVT1 (4 bits)			
SC	—	—	—	—	—	MPE	PWLL	PWLS	TAP	—	CDA1	CDA0	UPA	ROD	PWL	—
IPR0	IVP7[1:0]		IVP6[1:0]		IVP5[1:0]		IVP4[1:0]		IVP3[1:0]		IVP2[1:0]		IVP1[1:0]		IVP0[1:0]	
PRIVF									PSYWF	PSYRF	PULWF	PULRF	—	—	—	—
ULDR	—	—	—	—	—	—	—	ULDR (9 bits)								
UAPP	—	—	—	—	—	—	—	UAPP (9 bits)								
CKCN									—	—	—	STOP	SWB	PMME	CD1	CD0
WDCN									POR	EWDI	WD1	WD0	WDIF	WTRF	EWT	RWT
A[0:15]	A[0:15] (16 bits)															
PFX[0:7]	PFX[0:7] (16 bits)															
IP	IP (16 bits)															
SP	—	—	—	—	—	—	SP (10 bits)									
IV	IV (16 bits)															
LC[0]	LC[0] (16 bits)															
LC[1]	LC[1] (16 bits)															
OFFS									OFFS (8 bits)							
DPC	—	—	—	—	—	—	—	—	—	CWBS	—	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, 8 bits):(GRL, 8 bits)															
FP	FP = BP[OFFS] (16 bits)															
DP[0]	DP[0] (16 bits)															
DP[1]	DP[1] (16 bits)															
CP	CP (16 bits)															

Table 4-3. System Register Bit Reset Values

REG	BIT															
	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
PRIV									0	0	0	0	1	1	1	1
PRIVT0									0	0	0	0	0	0	0	0
PSF									1	0	0	0	0	0	0	0
IC									0	0	0	0	1	1	0	0
PRIVT1									0	0	0	0	0	0	0	0
SC	0	0	0	0	0	1	s	s	1	0	0	0	0	0	s	0
IPR0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PRIVF									0	0	0	0	0	0	0	0
ULDR	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
UAPP	0	0	0	0	0	0	0	0	1	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[0:15]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PFX[0:7]	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	1	1	1	1	1	1	0	0	0	0
IV	0	0	0	0	0	0	0	0	0	0	1	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	0	0	0	0	0	0	0	0	0	1	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
CP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note 1: *Bits marked as "s" are static across some or all resets.*

Note 2: *ULDR/UAPP reset values shown are for parts with 64KB/512B per page of program space. The reset value is the first page address past the available program memory on all resets.*

4.1 System Register Descriptions

The addresses for each register are given in the format *module[index]*, where *module* is the module specifier from 08h to 0Fh and *index* is the register subindex from 00h to 0Fh.

<table>
<tr><th>REGISTER</th><th colspan="2">DESCRIPTION</th></tr>
<tr><td>AP, 08h[00h]</td><td colspan="2">Accumulator Pointer Register (8 bits)</td></tr>
<tr><td>Initialization</td><td colspan="2">This register is cleared to 00h on all forms of reset.</td></tr>
<tr><td>Access
AP.3 to AP.0</td><td colspan="2">Unrestricted direct read/write access.
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 is automatically changed after each arithmetic or logical operation. If a 'MOVE AP, Acc' instruction is executed, any enabled AP inc/dec/modulo control takes precedence over the transfer of Acc data into AP.</td></tr>
<tr><td>AP.7 to AP.4</td><td colspan="2">Reserved. All reads return 0.</td></tr>
<tr><td>APC, 08h[01h]</td><td colspan="2">Accumulator Pointer Control Register (8 bits)</td></tr>
<tr><td>Initialization</td><td colspan="2">This register is cleared to 00h on all forms of reset.</td></tr>
<tr><td>Access
APC.2 to APC.0
(MOD2 to MOD0)</td><td colspan="2">Unrestricted direct read/write access.
Accumulator Pointer Auto Increment/Decrement Modulus. If these bits are set to a non-zero value, the accumulator pointer (AP[3:0]) is automatically incremented or decremented following each arithmetic or logical operation.
The mode for the autoincrement/decrement is determined as follows:</td></tr>
<tr><td></td><td>MOD[2:0]</td><td>AUTOINCREMENT/DECREMENT MODE</td></tr>
<tr><td></td><td>000</td><td>No autoincrement/decrement (default)</td></tr>
<tr><td></td><td>001</td><td>Increment/decrement AP[0] modulo 2</td></tr>
<tr><td></td><td>010</td><td>Increment/decrement AP[1:0] modulo 4</td></tr>
<tr><td></td><td>011</td><td>Increment/decrement AP[2:0] modulo 8</td></tr>
<tr><td></td><td>100</td><td>Increment/decrement AP modulo 16</td></tr>
<tr><td></td><td>101 to 111</td><td>Reserved (modulo 16 when set)</td></tr>
<tr><td>APC.5 to APC.3
APC.6 (IDS)</td><td colspan="2">Reserved. All reads return 0.
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.</td></tr>
<tr><td>APC.7 (CLR)</td><td colspan="2">AP Clear. Writing this bit to 1 clears the accumulator pointer, AP, to zero. If a 'MOVE APC, Acc' instruction is executed requesting that AP be set to zero (i.e., CLR = 1), the AP clear function overrides any enabled inc/dec/modulo control. All reads from this bit return 0.</td></tr>
</table>

REGISTER	DESCRIPTION
PRIV, 08h[02h] *Initialization*	**Privilege Register (8 bits)** This register is reset to 00001111b on all resets. Bits 3 and 2 are cleared by hardware when the current IP is not in utility ROM code, nor system code. Bits 1 and 0 are cleared by hardware when the current IP is not in utility ROM, system, nor user loader code.
Access	Bits 3 and 2 can only be modified by utility ROM code, or system code. Bits 1 and 0 can only be modified by utility ROM code, system code, or user loader code. Unrestricted read access. Writing this register clears the PRIVT0 register.
PRIV.0 (PULR)	**User Loader Read Privilege.** This bit defaults to 1 on a power-on reset. When this bit is 1, code can read the user loader memory area. Clearing this bit to 0 disables reading from user loader memory and any read attempt generates a protection-fault interrupt. Note that this bit is automatically cleared when the current IP is not in utility ROM code, system memory, or user loader memory.
PRIV.1 (PULW)	**User Loader Write Privilege.** This bit defaults to 1 on a power-on reset. This bit defaults to 1 on a power-on reset. When this bit is 1, code can write (program) the user loader memory area. Clearing this bit to 0 disables writing to user loader memory and any write attempt generates a protection-fault interrupt. Note that this bit is automatically cleared when the current IP is not in utility ROM code, system memory, or user loader memory.
PRIV.2 (PSYR)	**System Read Privilege.** This bit defaults to 1 on a power-on reset. When this bit is 1, code can read the system memory area. Clearing this bit to 0 disables reading from system memory and any read attempt generates a protection-fault interrupt. Note that this bit is automatically cleared when the current IP is not in utility ROM code or system memory.
PRIV.3 (PSYW)	**System Write Privilege.** This bit defaults to 1 on a power-on reset. This bit defaults to 1 on a power-on reset. When this bit is 1, code can write (program) the system memory area. Clearing this bit to 0 disables writing to system memory and any write attempt generates a protection-fault interrupt. Note that this bit is automatically cleared when the current IP is not in utility ROM code or system memory.
PRIV.7 to PRIV.4	Reserved. Reads return 0.
PRIVT0, 08h[03h] *Initialization*	**Privilege Register Atomic 0 (8 bits)** This register is reset to 00h on all resets, and on any write to the PRIV register, or the PRIVT1 destination. Bits 3 and 2 are cleared by hardware when the current IP is not in utility ROM code, nor system code. Bits 1 and 0 are cleared by hardware when the current IP is not in utility ROM, system, nor user loader code.
Access	Bits 3 and 2 can only be modified by utility ROM code, or system code. Bits 1 and 0 can only be modified by utility ROM code, system code, or user loader code. Unrestricted read access.
PRIVT0.3 to PRIVT0.0	**Privilege Atomic 0 Bits.** These bits default to 0 on a power-on reset. The bits are used as a logical AND bit mask when writing to PRIVT1.
PRIVT0.7 to PRIVT0.4	Reserved. Reads return 0.

REGISTER	DESCRIPTION
PSF, 08h[04h]	**Processor Status Flags Register (8 bits)**
Initialization *Access*	This register is cleared to 80h on all forms of reset. Bit 7 (Z) and bit 6 (S) are read-only. Bits 4, 3 (GPF1, GPF0), bit 2 (OV), bit 1 (C) and bit 0 (E) are unrestricted read/write.
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 addition or subtraction operation (ADD, ADDC, SUB, SUBB) returns a carry or borrow. This bit flag is cleared to 0 whenever an addition or subtraction operation does not return a carry or borrow.
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.** General-purpose flag bits are provided for user software control.
PSF.5	Reserved. 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.

<table>
<tr><th>REGISTER</th><th>DESCRIPTION</th></tr>
<tr><td>IC, 8h[5h]
Initialization
Access</td><td>Interrupt and Control Register (8 bits)
This register is cleared to 0Ch on all forms of reset.
Unrestricted direct read. Write access to bits 0, 4, 5, 6, 7 only. See bit descriptions for details.</td></tr>
<tr><td>IC.0 (IGE)</td><td>Interrupt Global Enable
If this bit is set to 1, interrupts can be enabled individually.
If this bit is set to 0, all interrupts are disabled (except the power-fail warning interrupt, which is enabled solely by its interrupt enable (PFIE)).</td></tr>
<tr><td>IC.1</td><td>Reserved. Reads return 0.</td></tr>
<tr><td>IC.2 (IPS0)</td><td>Interrupt Priority Status 0</td></tr>
<tr><td>IC.3 (IPS1)</td><td>Interrupt Priority Status 1. These read-only bits are set to 11b if the processor is not serving an interrupt. These bits are updated by the interrupt handler in response to an interrupt request. Any value other than 11b indicates that the processor is currently executing an interrupt service routine with the specified priority. These bits are set to 11b when the processor executes the corresponding RETI instruction.
<table>
<tr><th>IPS1</th><th>IPS0</th><th>FUNCTION</th></tr>
<tr><td>0</td><td>0</td><td>Serving a level 0 (highest priority) interrupt</td></tr>
<tr><td>0</td><td>1</td><td>Serving a level 1 interrupt</td></tr>
<tr><td>1</td><td>0</td><td>Serving a level 2 (lowest priority) interrupt</td></tr>
<tr><td>1</td><td>1</td><td>Not serving any interrupt</td></tr>
</table></td></tr>
<tr><td>IC.7 to IC.4</td><td>Reserved. Reads return 0.</td></tr>
<tr><td>PRIVT1, 08h[06h]
Initialization</td><td>Privilege Register Atomic 1 (8 bits)
This register is reset to 00h on all resets.
Bits 3 and 2 are cleared by hardware when the current IP is not in utility ROM code, nor system code.
Bits 1 and 0 are cleared by hardware when the current IP is not in utility ROM, system, nor user loader code.</td></tr>
<tr><td>Access</td><td>Bits 3 and 2 can only be written by utility ROM code, or system code. Bits 1 and 0 can only be written by utility ROM code, system code, or user loader code. No read access.</td></tr>
<tr><td>PRIVT1.3 to PRIVT1.0</td><td>Privilege Atomic 1 Bits. These bits default to 0 on a power-on reset. The bits are used as a logical AND bit mask. Writing these bits sets the corresponding bits in the PRIV register using the PRIVT0 register as a logical AND bit mask:
PRIV = (PRIVT0) AND (PRIVT1).
Writing to PRIVT1 clears the PRIVT0 register.</td></tr>
<tr><td>PRIVT1.7 to PRIVT1.4</td><td>Reserved. Reads return 0.</td></tr>
</table>

<table>
<tr><th>REGISTER</th><th>DESCRIPTION</th></tr>
<tr><td>SC, 08h[08h]
Initialization</td><td>System Control Register (16 bits)
This register is reset to 000001ss100000s0b on all resets. Bits 1, 8, and 9 (PWL, PWLS, PWLL) are set to 1 on power-fail and power-on reset only.</td></tr>
<tr><td>Access</td><td>Bits 8, 9, and 10 have write restrictions (see bit descriptions). All other bits: unrestricted read/write access.</td></tr>
<tr><td>SC.0</td><td>Reserved. All reads return 0.</td></tr>
<tr><td>SC.1 (PWL)</td><td>Password Lock Application. This bit defaults to 1 on power-fail and power-on reset. When this bit is 1, it requires a 32-byte password to be matched with the password in the user application program space before allowing access to the user-application password protected in-circuit debug or bootstrap loader utility ROM routines. Clearing this bit to 0 disables the password protection for these utility ROM routines. ROM-assisted active debug commands are always disallowed if the value at flash word address 000Eh is programmed (i.e., ≠FFFFh).</td></tr>
<tr><td>SC.2 (ROD)</td><td>Utility ROM Operation Done. This bit is used to signify completion of a utility ROM operation sequence to the control units. This allows the debug engine to determine the status of a utility ROM sequence. Setting this bit to 1 causes an internal system reset if the JTAG SPE bit is also set. Setting the ROD bit clears the JTAG SPE bit if it is set, and the ROD bit is automatically cleared by hardware once the control unit acknowledges the done indication.</td></tr>
<tr><td>SC.3 (UPA)</td><td>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. However, P2 and P3 must be implicitly activated in the upper half of the program space by setting the UPA bit to 1 for normal program execution. When UPA bit is cleared to 0, the upper program memory space is occupied by the utility ROM and the physical data to be accessible as program memory. This bit is reserved and reads return 0 on all parts with 64KB program memory or less.</td></tr>
<tr><td>SC.5 to SC.4
(CDA1, CDA0)</td><td>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:
<table>
<tr><th>CDA[1:0]</th><th>BYTE MODE
ACTIVE PAGE</th><th>WORD MODE
ACTIVE PAGE</th></tr>
<tr><td>00</td><td>P0</td><td>P0 and P1</td></tr>
<tr><td>01</td><td>P1</td><td>P0 and P1</td></tr>
<tr><td>10</td><td>P2</td><td>P2 and P3</td></tr>
<tr><td>11</td><td>P3</td><td>P2 and P3</td></tr>
</table>
The logical addresses are depending on which memory segment is executing.
CDA1 is reserved and reads return 0 on all parts with 64KB program memory or less.
CDA0 is reserved and reads return 0 on all parts with 32KB program memory or less.</td></tr>
<tr><td>SC.6</td><td>Reserved. All reads return 0.</td></tr>
<tr><td>SC.7 (TAP)</td><td>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.</td></tr>
<tr><td>SC.8 (PWLS)</td><td>Password Lock System. This bit defaults to 1 on power-fail and power-on reset. When this bit is 1, it requires a 32-byte password to be matched with the password in the system program space before allowing access to the system password-protected in-circuit debug or bootstrap loader utility ROM routines. Clearing this bit to 0 disables the password protection for these utility ROM routines. This register bit can only be written by utility ROM code when PRIV = HIGH. ROM assisted active debug commands are always disallowed if the value at flash word address 000Eh is programmed (i.e., ≠FFFFh).</td></tr>
</table>

<table>
<tr><th>REGISTER</th><th>DESCRIPTION</th></tr>
<tr><td>SC.9 (PWLL)</td><td>Password Lock User Loader. This bit defaults to 1 on power-fail and power-on reset. When this bit is 1, it requires a 32-byte password to be matched with the password in the user loader program space before allowing access to the user loader password-protected in-circuit debug or bootstrap loader utility ROM routines. Clearing this bit to 0 disables the password protection for these utility ROM routines. This register bit can only be written by utility ROM code when PRIV ≥ MEDIUM. ROM-assisted active debug commands are always disallowed if the value at flash word address 000Eh is programmed (i.e., ≠FFFFh).</td></tr>
<tr><td>SC.10 (MPE)</td><td>Memory Protection Enable. This bit defaults to 1 on any reset. When this bit is 1, it enables memory protection and access control. When this bit is 0, no protection-fault interrupts are generated and any code can access the protected resources. This register bit can only be changed from 1 to 0 (thereby disabling memory protection) when PRIV = HIGH. Note that the ability to read utility ROM is always allowed (independent of the MPE bit state).</td></tr>
<tr><td>SC.15 to SC.11</td><td>Reserved. Reads return 0.</td></tr>
<tr><td>IPR0, 08h[09h]</td><td>Interrupt Priority Register Zero (16 bits)</td></tr>
<tr><td>Initialization</td><td>This register is cleared to 0000h on all forms of reset.</td></tr>
<tr><td>Access</td><td>Unrestricted direct read/write.</td></tr>
<tr><td>IPR0[1:0] (IVP0[1:0])</td><td>Interrupt Vector 0 Priority Bits 1:0. These bits are used to specify the priority level of interrupt vector 0.
<table>
<tr><th>IVP[1:0]</th><th>PRIORITY</th></tr>
<tr><td>00</td><td>Level 0 (the highest)</td></tr>
<tr><td>01</td><td>Level 1</td></tr>
<tr><td>10</td><td>Level 2 (the lowest)</td></tr>
<tr><td>11</td><td>Reserved (interrupt disabled)</td></tr>
</table></td></tr>
<tr><td>IPR0[3:2] (IVP1[1:0])</td><td>Interrupt Vector 1 Priority Bits 1:0. These bits are used to specify the priority level of interrupt vector 1.</td></tr>
<tr><td>IPR0[5:4] (IVP2[1:0])</td><td>Interrupt Vector 2 Priority Bits 1:0. These bits are used to specify the priority level of interrupt vector 2.</td></tr>
<tr><td>IPR0[7:6] (IVP3[1:0])</td><td>Interrupt Vector 3 Priority Bits 1:0. These bits are used to specify the priority level of interrupt vector 3.</td></tr>
<tr><td>IPR0[9:8] (IVP4[1:0])</td><td>Interrupt Vector 4 Priority Bits 1:0. These bits are used to specify the priority level of interrupt vector 4.</td></tr>
<tr><td>IPR0[11:10] (IVP5[1:0])</td><td>Interrupt Vector 5 Priority Bits 1:0. These bits are used to specify the priority level of interrupt vector 5.</td></tr>
<tr><td>IPR0[13:12] (IVP6[1:0])</td><td>Interrupt Vector 6 Priority Bits 1:0. These bits are used to specify the priority level of interrupt vector 6.</td></tr>
<tr><td>IPR0[15:14] (IVP7[1:0])</td><td>Interrupt Vector 7 Priority Bits 1:0. These bits are used to specify the priority level of interrupt vector 7.</td></tr>
</table>

REGISTER	DESCRIPTION
PRIVF, 08h[0Bh]	**Privilege Flag Register (8 bits)**
Initialization	This register is cleared to 00h on all forms of reset.
Access	Unrestricted direct read/write.
PRIVF.3 to PRIVF.0	Reserved. All reads return 0.
PRIVF.4 (PULRF)	**Protected User Loader Read Interrupt Flag.** The interrupt flag is set to 1 if code attempts/requests to read user loader memory when PULR = 0. Once set, this flag can only be cleared by software or by reset.
PRIVF.5 (PULWF)	**Protected User Loader Write Interrupt Flag.** The interrupt flag is set to 1 if code attempts/requests to write user loader memory when PULW = 0. Once set, this flag can only be cleared by software or by reset.
PRIVF.6 (PSYRF)	**Protected System Read Interrupt Flag.** The interrupt flag is set to 1 if code attempts/requests to read system memory when PSYR = 0. Once set, this flag can only be cleared by software or by reset.
PRIVF.7 (PSYWF)	**Protected System Write Interrupt Flag.** The interrupt flag is set to 1 if code attempts/requests to write system memory when PSYW = 0. Once set, this flag can only be cleared by software or by reset.
ULDR, 08h[0Ch]	**User Loader Starting Page Address (16 bits)**
Initialization	This register is reset to the first page address past the available flash program memory on all resets. On a part with 64KB of program memory with 512-byte pages, this register is reset to 0080h.
Access	This register can only be modified when PRIV = HIGH. Unrestricted read access.
ULDR.8 to ULDR.0	**User Loader Starting Page Address.** These bits define the starting page address of the user loader memory area.
ULDR.15 to ULDR.9	Reserved. Reads return 0.
UAPP, 08h[0Dh]	**User Application Starting Page Address (16 bits)**
Initialization	This register is reset to the first page address past the available flash program memory on all resets. On a part with 64KB of program memory with 512-byte pages, this register is reset to 0080h.
Access	This register can only be modified when PRIV ≥ MEDIUM. Unrestricted read access.
UAPP.8 to UAPP.0	**User Application Starting Page Address.** These bits define the starting page address of the user application memory area.
UAPP.15 to UAPP.9	Reserved. Reads return 0.

<table>
<tr><th>REGISTER</th><th>DESCRIPTION</th></tr>
<tr><td>CKCN, 08h[0Eh]</td><td>System Clock Control Register (8 bits)</td></tr>
<tr><td>Initialization
Access</td><td>Bits 4:0 are cleared to zero on all forms of reset. See bit description for bits 7:5.
Unrestricted read/write, except there is a locking mechanism for the PMME, CD1, and CD0 bits when changing their bits values; bit 5 is read-only.</td></tr>
<tr><td>CKCN.0 (CD0)</td><td>Clock Divide Bit 0</td></tr>
<tr><td>CKCN.1 (CD1)</td><td>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:
<table>
<tr><th>CD1</th><th>CD0</th><th>OSCILLATOR CLOCK CYCLES PER SYSTEM CLOCK CYCLE</th></tr>
<tr><td>0</td><td>0</td><td>1 (default)</td></tr>
<tr><td>0</td><td>1</td><td>2</td></tr>
<tr><td>1</td><td>0</td><td>4</td></tr>
<tr><td>1</td><td>1</td><td>8</td></tr>
</table>
If the PMME bit is set to 1, the values of CD0 and CD1 cannot be altered and do not affect the system clock frequency.</td></tr>
<tr><td>CKCN.2 (PMME)</td><td>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, switchback mode has been enabled by setting the SWB bit and a switchback source (such as an enabled external interrupt) is currently active, PMME is cleared to 0 and cannot be set to 1 unless all switchback sources are inactive.</td></tr>
<tr><td>CKCN.3 (SWB)</td><td>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 is cleared to 0 when one of the qualifying events occurs. For details, refer to the switchback description.
When any of these conditions cause switchback to clear PMME to 0, the system clock rate is then determined by the settings of CD0 and CD1. After PMME is cleared to 0 by switchback, it cannot be set back to 1 as long as any of the above conditions are true.</td></tr>
<tr><td>CKCN.4 (STOP)</td><td>Stop Mode Select. Setting this bit to 1 causes the MAXQ610 to enter stop mode. This does not change the currently selected clock divide ratio (CD0, CD1, PMME).</td></tr>
<tr><td>CKCN.7 to CKCN.5</td><td>Reserved. Reads return 0.</td></tr>
</table>

REGISTER	DESCRIPTION
WDCN, 08h[0Fh] *Initialization*	**Watchdog Control Register (8 bits)** 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.
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 is stopped. If the watchdog timer is stopped (EWT = 0 and EWDI = 0), setting the EWT bit resets the watchdog interval and resets counter, and enables the watchdog timer. This bit is cleared on power-fail and 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 does not cause a watchdog reset. This bit is cleared by power-fail and 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 is not set.
WDCN.3 (WDIF)	**Watchdog Interrupt Flag.** This bit is 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 occurs if the watchdog interrupt has been enabled (EWDI = 1) and not otherwise masked or prevented by a higher priority interrupt already in service (i.e., IGE = 1, and IPS = 11b or lower priority interrupt in service in order 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 through the WD[1:0] bits automatically resets the watchdog timer unless the 512 system clock reset counter is already in progress, in which case, changing the WD[1:0] bits does not affect the watchdog timer or reset counter.

WD1	WD0	CLOCKS UNTIL INTERRUPT	CLOCKS UNTIL RESET
0	0	2^{15}	2^{15} + 512
0	1	2^{18}	2^{18} + 512
1	0	2^{21}	2^{21} + 512
1	1	2^{24}	2^{24} + 512

REGISTER	DESCRIPTION
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 occurs 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 is stopped. If the watchdog timer is stopped (EWT = 0 and EWDI = 0), setting the EWDI bit resets the watchdog interval and reset counter, and enables the watchdog timer. This bit is cleared to 0 by power-fail and power-on reset and is unaffected by other forms of reset.
WDCN.7 (POR)	**Power-on Reset Flag.** This bit is set to 1 anytime when V_{DD} is below the V_{POR} threshold. This bit must be cleared by software. This bit is unaffected by resets and is set to 1 by hardware only by POR ($V_{DD} < V_{POR}$).
A[n], 09h[nh] *Initialization* *Access* *A[n].15 to A[n].0*	**Accumulator n Register (16 bits)** This register is cleared to 0000h on all forms of reset. Unrestricted direct read/write access. 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.
PFX[n], 0Bh[nh] *Initialization* *Access* *PFX[n].15 to PFX[n].0*	**Prefix Register (16 bits)** This register is cleared to 0000h on all forms of reset. Unrestricted direct read/write access. 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 only holds any data written to it for one execution cycle, after which it reverts 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 selects 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.

SOURCE, DESTINATION INDEX SELECTION

WRITE TO	SOURCE REGISTER RANGE	DESTINATION REGISTER RANGE
PFX[0]	00h to 0Fh	00h to 07h
PFX[1]	10h to 1Fh	00h to 07h
PFX[2]	00h to 0Fh	08h to 0Fh
PFX[3]	10h to 1Fh	08h to 0Fh
PFX[4]	00h to 0Fh	10h to 17h
PFX[5]	10h to 1Fh	10h to 17h
PFX[6]	00h to 0Fh	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.

REGISTER	DESCRIPTION
IP, 0Ch[00h]	**Instruction Pointer Register (16 bits)**
Initialization	This register is cleared to 8000h on all forms of reset.
Access *IP.15 to IP.0*	Unrestricted direct read/write access. 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 causes program flow to jump to that address. Reading from this register does not affect program flow.
SP, 0Dh[01h]	**Stack Pointer Register (16 bits)**
Initialization	This register is cleared to 03F0h on all forms of reset.
Access *SP.9 to SP.0*	Unrestricted direct read/write access. These 10 bits indicate the current top (equals the lowest address used) of the soft stack. This pointer is decremented before a value is pushed on the stack (increasing the stack depth, MOVE @++SP, …) and incremented after a value is popped from the stack (decreasing the stack depth, MOVE …, @SP--).
SP.15 to SP.10	Reserved. Reads return 0.
IV, 0Dh[02h]	**Interrupt Vector Register (16 bits)**
Initialization	This register is cleared to 0020h on all forms of reset.
Access *IV.15 to IV.0*	Unrestricted direct read-only. This register contains the address of the interrupt service routine. The interrupt handler generates a CALL to an offset from this address whenever the corresponding interrupt is acknowledged.
LC[0], 0Dh[06h]	**Loop Counter 0 Register (16 bits)**
Initialization	This register is cleared to 0000h on all forms of reset.
Access *LC[0].15 to LC[0].0*	Unrestricted direct read/write access. 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*.
LC[1], 0Dh[07h]	**Loop Counter 1 Register (16 bits)**
Initialization	This register is cleared to 0000h on all forms of reset.
Access *LC[1].15 to LC[1].0*	Unrestricted direct read/write access. 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*.
OFFS, 0Eh[03h]	**Frame Pointer Offset Register (8 bits)**
Initialization	This register is cleared to 00h on all forms of reset.
Access *OFFS.7 to OFFS.0*	Unrestricted direct read/write access. 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 postincremented or postdecremented when using the frame pointer for read operations and can be preincremented or predecremented 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).

<table>
<tr><th>REGISTER</th><th>DESCRIPTION</th></tr>
<tr><td>DPC, 0Eh[04h]
Initialization
Access
DPC.1 to DPC.0
(SDPS1, SDPS0)</td><td>Data Pointer Control Register (16 bits)
This register is cleared to 005Ch on all forms of reset.
Unrestricted direct read/write access.
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:
<table>
<tr><th>SDPS1</th><th>SDPS0</th><th>SOURCE POINTER SELECTION</th></tr>
<tr><td>0</td><td>0</td><td>DP[0]</td></tr>
<tr><td>0</td><td>1</td><td>DP[1]</td></tr>
<tr><td>1</td><td>0</td><td>FP (BP[OFFS])</td></tr>
<tr><td>1</td><td>1</td><td>Reserved (select FP if set)</td></tr>
</table>
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) changes the setting of the SDPS bits to reflect the active source pointer selection.</td></tr>
<tr><td>DPC.2 (WBS0)</td><td>Word/Byte Select 0. This bit selects access mode for DP[0]. When WBS0 is set to 1, the DP[0] is operated in word mode for data memory access; when WBS0 is cleared to 0, DP[0] is operated in byte mode for data memory access.</td></tr>
<tr><td>DPC.3 (WBS1)</td><td>Word/Byte Select 1. This bit selects access mode for DP[1]. When WBS1 is set to 1, the DP[1] is operated in word mode for data memory access; when WBS1 is cleared to 0, DP[1] is operated in byte mode for data memory access.</td></tr>
<tr><td>DPC.4 (WBS2)</td><td>Word/Byte Select 2. This bit selects access mode for BP[OFFS]. When WBS2 is set to 1, the BP[OFFS] is operated in word mode for data memory access; when WBS2 is cleared to 0, BP[OFFS] is operated in byte mode for data memory access.</td></tr>
<tr><td>DPC.5</td><td>Reserved. Reads return 0.</td></tr>
<tr><td>DPC.6 (CWBS)</td><td>Code Pointer Word/Byte Select. This bit selects access mode for the code pointer, CP. When CWBS is set to 1, the CP is operated in word mode for data memory access; when CWBS is cleared to 0, CP is operated in byte mode for data memory access.</td></tr>
<tr><td>DPC.15 to DPC.7</td><td>Reserved. Read returns 0.</td></tr>
<tr><td>GR, 0Eh[05h]
Initialization
Access
GR.15 to GR.0</td><td>General Register (16 bits)
This register is cleared to 0000h on all forms of reset.
Unrestricted direct read/write access.
This register is intended primarily for supporting byte operations on 16-bit data. The 16-bit register is byte-readable, byte-writable through the corresponding GRL and GRH 8-bit registers and byte-swappable through the GRS 16-bit register</td></tr>
<tr><td>GRL, 0Eh[06h]
Initialization
Access
GRL.7 to GRL.0</td><td>General Register Low Byte (8 bits)
This register is cleared to 00h on all forms of reset.
Unrestricted direct read/write access.
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 is also stored in the low byte of the GR register.</td></tr>
<tr><td>BP, 0Eh[07h]
Initialization
Access
BP.15 to BP.0</td><td>Frame Pointer Base Register (16 bits)
This register is cleared to 0000h on all forms of reset.
Unrestricted direct read/write access.
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.</td></tr>
</table>

REGISTER	DESCRIPTION
GRS, 0Eh[08h]	**General Register Byte-Swapped (16 bits)**
Initialization	This register is cleared to 0000h on all forms of reset
Access *GRS.15 to GRS.0*	Unrestricted read-only access. 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.
GRH, 0Eh[09h]	**General Register High Byte (8 bits)**
Initialization	This register is cleared to 00h on all forms of reset.
Access *GRH.7 to GRH.0*	Unrestricted direct read/write access. 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 is also stored in the high byte of the GR register.
GRXL, 0Eh[0Ah]	**General Register Sign Extended Low Byte (16 bits)**
Initialization	This register is cleared to 0000h on all forms of reset.
Access	Unrestricted direct read-only access.
GRXL.15 to GRXL.0	This register provides the sign extended low byte of GR as a 16-bit source.
FP, 0Eh[0Bh]	**Frame Pointer Register (16 bits)**
Initialization	This register is cleared to 0000h on all forms of reset.
Access	Unrestricted direct read-only access.
FP.15 to FP.0	This register provides the current value of the frame pointer (BP[OFFS]).
DP[0], 0Fh[03h]	**Data Pointer 0 Register (16 bits)**
Initialization	This register is cleared to 0000h on all forms of reset.
Access *DP[0].15 to DP[0].0*	Unrestricted direct read/write access. 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.
DP[1], 0Fh[07h]	**Data Pointer 1 Register (16 bits)**
Initialization	This register is cleared to 0000h on all forms of reset.
Access *DP[1].15 to DP[1].0*	Unrestricted direct read/write access. 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.
CP, 0Fh[0Bh]	**Code Pointer Address Register (16 bits)**
Initialization	This register is cleared to 0000h on all forms of reset.
Access *CP.15 to CP.0*	Unrestricted direct read/write access. This register is used as a pointer to access program code memory. CP can be automatically incremented or decremented following each read operation.

SECTION 5: PERIPHERAL REGISTER MODULES

This section contains the following information:

5.1 Peripheral Register Bit Descriptions 5-5

LIST OF TABLES

Table 5-1. Peripheral Register Map 5-2
Table 5-2. Peripheral Register Bit Function 5-2
Table 5-3. Peripheral Register Reset Values 5-3

SECTION 5: PERIPHERAL REGISTER MODULES

The MAXQ610 microcontroller uses peripheral registers to control and monitor peripheral modules. These registers reside in modules 0h to 3h, with subindex values 0h to 1Fh.

Table 5-1. Peripheral Register Map

MODULE		INDEX OF SPECIAL FUNCTION REGISTER (SECTIONS I AND II)															
MODULE	**SPECIFIER**	**00000**	**00001**	**00010**	**00011**	**00100**	**00101**	**00110**	**00111**	**01000**	**01001**	**01010**	**01011**	**01100**	**01101**	**01110**	**01111**
M0	00000	PO0	PO1	PO2	PO3	EIF0	EIE0	EIF1	EIE1	PI0	PI1	PI2	PI3	EIES0	EIES1		
M1	00001	PO4				WUTC	WUT			PI4				PWCN			
M2	00010	TB0R	TB0CN	TB1R	TB1CN	IRCN	IRCA	IRMT	IRCNB	TB0C	TB0V	TB1C	TB1V	IRV			
M3	00011	SCON0	SBUF0	SCON1	SBUF1	SPIB	SPICN			PR0	SMD0	PR1	SMD1	SPICF	SPICK		
M4	00100																
M5	00101																

MODULE		INDEX OF SPECIAL FUNCTION REGISTER (SECTION III)															
MODULE	**SPECIFIER**	**10000**	**10001**	**10010**	**10011**	**10100**	**10101**	**10110**	**10111**	**11000**	**11001**	**11010**	**11011**	**11100**	**11101**	**11110**	**11111**
M0	00000	PD0	PD1	PD2	PD3				CHPREV								
M1	00001	PD4															
M2	00010																
M3	00011																
M4	00100																
M5	00101																

Table 5-2. Peripheral Register Bit Function

REG	BIT															
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PO0									PO0[7:0]							
PO1									PO1[7:0]							
PO2									PO2[7:0]							
PO3									PO3[7:0]							
EIF0									IE[7:0]							
EIE0									EX[7:0]							
EIF1									IE[15:8]							
EIE1									EX[15:8]							
PI0									PI0[7:0]							
PI1									PI1[7:0]							
PI2									PI2[7:0]							
PI3									PI3[7:0]							
EIES0									IT[7:0]							
EIES1									IT[15:8]							
PD0									PD0[7:0]							
PD1									PD1[7:0]							
PD2									PD2[7:0]							
PD3									PD3[7:0]							
CHPREV									CHPREV[7:0]							

Table 5-2. Peripheral Register Bit Function (continued)

REG	BIT															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PO4									—	—	PO4[5:0]					
WUTC									—	—	—	—	—	—	WTF	WTE
WUT	WUT[15:0]															
PI4									—	—	PI4[5:0]					
PWCN	—	—	—	—	—	—	PFRCK1	PFRCK0	PFRST	IRRXWP	IRTXOUT	IRTXOE	REGEN	PFI	PFIE	PFD
PD4									—	—	PD4[5:0]					
TB0R	TB0R[15:0]															
TB0CN	C/TB	—	—	TBCS	TBCR	TBPS2	TBPS1	TBPS0	TFB	EXFB	TBOE	DCEN	EXENB	TRB	ETB	CP/RLB
TB1R	TB1R[15:0]															
TB1CN	C/TB	—	—	TBCS	TBCR	TBPS2	TBPS1	TBPS0	TFB	EXFB	TBOE	DCEN	EXENB	TRB	ETB	CP/RLB
IRCN	—	—	—	IRDIV[2:0]			IRENV[1:0]		IRXRL	IRCFME	IRRXSEL[1:0]		IRDATA	IRTXPOL	IRMODE	IREN
IRCA	IRCAH[7:0]								IRCAL[7:0]							
IRMT	IRMT[15:0]															
IRCNB									—	—	—	—	RXBCNT	IRIE	IRIF	IROV
TB0C	TB0C[15:0]															
TB0V	TB0V[15:0]															
TB1C	TB1C[15:0]															
TB1V	TB1V[15:0]															
IRV	IRV[15:0]															
SCON0									SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
SBUF0									SBUF0[7:0]							
SCON1									SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
SBUF1									SBUF1[7:0]							
SPIB	SPIB[15:0]															
SPICN									STBY	SPIC	ROVR	WCOL	MODF	MODFE	MSTM	SPIEN
PR0	PR0[15:0]															
SMD0									—	—	—	—	—	ESI0	SMOD0	FEDE0
PR1	PR1[15:0]															
SMD1									—	—	—	—	—	ESI1	SMOD1	FEDE1
SPICF									ESPII	SAS	—	—	—	CHR	CKPHA	CKPOL
SPICK									CKR[7:0]							

Table 5-3. Peripheral Register Reset Values

REG	BIT															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PO0									0	0	0	0	0	0	0	0
PO1									0	0	0	0	0	0	0	0
PO2									0	0	0	0	0	0	0	0
PO3									0	0	0	0	0	0	0	0
EIF0									0	0	0	0	0	0	0	0
EIE0									0	0	0	0	0	0	0	0

Table 5-3. Peripheral Register Reset Values (continued)

REG	BIT															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EIF1									0	0	0	0	0	0	0	0
EIE1									0	0	0	0	0	0	0	0
PI0									s	s	s	s	s	s	s	s
PI1									s	s	s	s	s	s	s	s
PI2									s	s	s	s	s	s	s	s
PI3									s	s	s	s	s	s	s	s
EIES0									0	0	0	0	0	0	0	0
EIES1									0	0	0	0	0	0	0	0
PD0									s	s	s	s	s	s	s	s
PD1									s	s	s	s	s	s	s	s
PD2									s	s	s	s	s	s	s	s
PD3									s	s	s	s	s	s	s	s
CHPREV									s	s	s	s	s	s	s	s
PO4									0	0	0	0	0	0	0	0
WUTC									0	0	0	0	0	0	0	0
WUT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PI4									0	0	s	s	s	s	s	s
PWCN	0	0	0	0	0	0	s	s	s	1	1	0	0	0	0	0
PD4									0	0	s	s	s	s	s	s
TB0R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TB0CN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TB1R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TB1CN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IRCN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IRCA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IRMT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IRCNB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TB0C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TB0V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TB1C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TB1V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IRV	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SCON0									0	0	0	0	0	0	0	0
SBUF0									0	0	0	0	0	0	0	0
SCON1									0	0	0	0	0	0	0	0
SBUF1									0	0	0	0	0	0	0	0
SPIB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPICN									0	0	0	0	0	0	0	0
PR0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SMD0									0	0	0	0	0	0	0	0
PR1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SMD1									0	0	0	0	0	0	0	0
SPICF									0	0	0	0	0	0	0	0
SPICK									0	0	0	0	0	0	0	0

5.1 Peripheral Register Bit Descriptions

REGISTER	DESCRIPTION
PO0 (00h, 00h) *Initialization:* *Read/Write Access:* *PO0.7 to PO0.0*	**Port 0 Output Register (8-bit register)** This register is set to 00h on all forms of reset. Unrestricted read/write. **Port 0 Output Register Bits 7:0.** The PO0 register stores output data for port 0 when it is defined as an output port and controls whether the internal weak p-channel pullup transistor is enabled/disabled if a port pin is defined as an input. The contents of this register can be modified by a write access. Reading from the register returns the contents of the register. Changing the direction of port 0 does not change the data contents of the register.
PO1 (01h, 00h) *Initialization:* *Read/Write Access:* *PO1.7 to PO1.0*	**Port 1 Output Register (8-bit register)** This register is set to 00h on all forms of reset. Unrestricted read/write. **Port 1 Output Register Bits 7:0.** The PO1 register stores output data for port 1 when it is defined as an output port and controls whether the internal weak p-channel pullup transistor is enabled/disabled if a port pin is defined as an input. The contents of this register can be modified by a write access. Reading from the register returns the contents of the register. Changing the direction of port 1 does not change the data contents of the register.
PO2 (02h, 00h) *Initialization:* *Read/Write Access:* *PO2.7 to PO2.0*	**Port 2 Output Register (8-bit register)** This register is set to 00h on all forms of reset. Unrestricted read/write. **Port 2 Output Register Bits 7:0.** The PO2 register stores output data for port 2 when it is defined as an output port and controls whether the internal weak p-channel pullup transistor is enabled/disabled if a port pin is defined as an input. The contents of this register can be modified by a write access. Reading from the register returns the contents of the register. Changing the direction of port 2 does not change the data contents of the register.
PO3 (03h, 00h) *Initialization:* *Read/Write Access:* *PO3.7 to PO3.0*	**Port 3 Output Register (8-bit register)** This register is set to 00h on all forms of reset. Unrestricted read/write. **Port 3 Output Register Bits 7:0.** The PO3 register stores output data for port 3 when it is defined as an output port and controls whether the internal weak p-channel pullup transistor is enabled/disabled if a port pin is defined as an input. The contents of this register can be modified by a write access. Reading from the register returns the contents of the register. Changing the direction of port 3 does not change the data contents of the register.
EIF0 (04h, 00h) *Initialization:* *Read/Write Access:* *EIF0.7 to EIF0.0 (IE[7:0])*	**External Interrupt Flag 0 Register** EIF0 is cleared to 00h on all forms of reset. Unrestricted read/write. **Interrupt Edge Detect Bits 7:0.** These bits are set when a negative edge (ITn = 1) or a positive edge (ITn = 0) is detected on the interrupt pin n. Setting any of the bits to 1 generates an interrupt to the CPU if the corresponding interrupt is enabled. The bit remains set until cleared by software or a reset. It must be cleared by software before exiting the interrupt source routine or another interrupt is generated as long as the bit remains set.
EIE0 (05h, 00h) *Initialization:* *Read/Write Access:* *EIE0.7 to EIE0.0 (EX[7:0])*	**External Interrupt Enable 0 Register** EIE0 is cleared to 00h on all forms of reset. Unrestricted read/write. **Enable External Interrupt Bits 7:0.** Setting any of these bits to 1 enables the corresponding external interrupt. Clearing any of the bits to 0 disables the corresponding interrupt function.

REGISTER	DESCRIPTION
EIF1 (06h, 00h) *Initialization:* *Read/Write Access:* *EIF1.7 to EIF1.0 (IE[15:8])*	**External Interrupt Flag 1 Register** EIF1 is cleared to 00h on all forms of reset. Unrestricted read/write. **Interrupt Edge Detect Bits 15:8.** These bits are set when a negative edge (ITn = 1) or a positive edge (ITn = 0) is detected on the interrupt n pin. Setting any of the bits to 1 generates an interrupt to the CPU if the corresponding interrupt is enabled. The bit remains set until cleared by software or a reset. It must be cleared by software before exiting the interrupt source routine or another interrupt is generated as long as the bit remains set. **Note:** For the 32-pin package, the INT8 to INT15 functions are not present on external pins, however, the associated interrupt registers (EIE1, EIF1, EIES1) are still present. Software should not write to the EIF1 register as this could trigger an unplanned interrupt condition if EIE1 and EIES1 are used for general purpose.
EIE1 (07h, 00h) *Initialization:* *Read/Write Access:* *EIE1.7 to EIE1.0 (EX[15:8])*	**External Interrupt Enable 1 Register** EIE1 is cleared to 00h on all forms of reset. Unrestricted read/write. **Enable External Interrupt Bits 15:8.** Setting any of these bits to 1 enables the corresponding external interrupt. Clearing any of the bits to 0 disables the corresponding interrupt function. **Note:** For the 32-pin package, the INT8 to INT15 functions are not present on external pins. This register can be used as a general-purpose register as long as the user software does not write to the EIF1 flag register since this could trigger an unplanned interrupt condition.
PI0 (08h, 00h) *Initialization:* *Read/Write Access:* *PI0.7 to PI0.0*	**Port 0 Input Register** The reset value for this register is dependent on the logical states of the pins. Unrestricted read-only. **Port 0 Input Register Bits 7:0.** The PI0 register always reflects the logic state of its pins when read. Note that each port pin has a weak pullup circuit when functioning as an input and the p-channel pullup transistor is controlled by its respective PO bits. If the PO bit is set to 1, the weak pullup is on, if the PO bit is cleared to 0, the weak pullup is off and forces the port pin into three-state.
PI1 (09h, 00h) *Initialization:* *Read/Write Access:* *PI1.7 to PI1.0*	**Port 1 Input Register** The reset value for this register is dependent on the logical states of the pins. Unrestricted read. **Port 1 Input Register Bits 7:0.** The PI1 register always reflects the logic state of its pins when read. Note that each port pin has a weak pullup circuit when functioning as an input and the p-channel pullup transistor is controlled by its respective PO bits. If the PO bit is set to 1, the weak pullup is on, if the PO bit is cleared to 0, the weak pullup is off and forces the port pin into three-state.
PI2 (0Ah, 00h) *Initialization:* *Read/Write Access:* *PI2.7 to PI2.0*	**Port 2 Input Register** The reset value for this register is dependent on the logical states of the pins. Unrestricted read. **Port 2 Input Register Bits 7:0.** The PI2 register always reflects the logic state of its pins when read. Note that each port pin has a weak pullup circuit when functioning as an input and the p-channel pullup transistor is controlled by its respective PO bits. If the PO bit is set to 1, the weak pullup is on, if the PO bit is cleared to 0, the weak pullup is off and forces the port pin into three-state.

REGISTER	DESCRIPTION
PI3 (0Bh, 00h) *Initialization:* *Read/Write Access:* *PI3.7 to PI3.0*	**Port 3 Input Register** The reset value for this register is dependent on the logical states of the pins. Unrestricted read. **Port 3 Input Register Bits 7:0.** The PI3 register always reflects the logic state of its pins when read. Note that each port pin has a weak pullup circuit when functioning as an input and the p-channel pullup transistor is controlled by its respective PO bits. If the PO bit is set to 1, the weak pullup is on, if the PO bit is cleared to 0, the weak pullup is off and forces the port pin into three-state.
EIES0 (0Ch, 00h) *Initialization:* *Read/Write Access:* *EIES0.7 to EIES0.0 (IT[7:0])*	**External Interrupt Edge Select 0 Register** EIES0 is cleared to 00h on all forms of reset. Unrestricted read/write. **Edge Select for External Interrupt Bits 7:0** ITn = 0: External Interrupt n is positive edge triggered. ITn = 1: External Interrupt n is negative edge triggered.
EIES1 (0Dh, 00h) *Initialization:* *Read/Write Access:* *EIES1.7 to EIES1.0 (IT[15:8])*	**External Interrupt Edge Select 1 Register** EIES1 is cleared to 00h on all forms of reset. Unrestricted read/write. **External Interrupt Edge Select Bits 15:8** ITx = 0: External interrupt x is positive edge triggered. ITx = 1: External interrupt x is negative edge triggered. **Note:** For the 32-pin package, the INT8 to INT15 functions are not present on external pins. This register can be used as a general-purpose register as long as the user software does not write to the EIF1 flag register since this could trigger an unplanned interrupt condition.
PD0 (10h, 00h) *Initialization:* *Read/Write Access:* *PD0.7 to PD0.0*	**Port 0 Direction Register** This register is cleared to 00h on all resets except power-fail reset. This register is unaffected by power-fail reset. Unrestricted read/write. **Port 0 Direction Register Bits 7:0.** PD0 is used to determine the direction of the port 0 function. The port pins are independently controlled by their direction bits. When a bit is set to 1, its corresponding pin is used as an output; data in the PO register is driven on the pin. When a bit is cleared to 0, its corresponding pin is used as an input, and allows an external signal to drive the pin. Note that each port pins has a weak pullup circuit when functioning as an input and the p-channel pullup transistor is controlled by its respective PO bits. If the PO bit is set to 1, the weak pullup is on, if the PO bit is cleared to 0, the weak pullup is off and forces the port pin into three-state.
PD1 (11h, 00h) *Initialization:* *Read/Write Access:* *PD1.7 to PD1.0*	**Port 1 Direction Register** This register is cleared to 00h on all resets except power-fail reset. This register is unaffected by power-fail reset. Unrestricted read/write. **Port 1 Direction Register Bits 7:0.** PD1 is used to determine the direction of the port 1 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; data in the PO register is driven on the pin. When a bit is cleared to 0, its corresponding pin is used as an input, and allows an external signal to drive the pin. Note that each port pin has a weak pullup circuit when functioning as an input and the p-channel pullup transistor is controlled by its respective PO bits. If the PO bit is set to 1, the weak pullup is on, if the PO bit is cleared to 0, the weak pullup is off and forces the port pin into three-state.

REGISTER	DESCRIPTION
PD2 (12h, 00h) *Initialization:* *Read/Write Access:* *PD2.7 to PD2.0*	**Port 2 Direction Register** This register is cleared to 00h on all resets except power-fail reset. This register is unaffected by power-fail reset. Unrestricted read/write. **Port 2 Direction Register Bits 7:0.** PD2 is used to determine the direction of the port 2 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; data in the PO register is driven on the pin. When a bit is cleared to 0, its corresponding pin is used as an input, and allows an external signal to drive the pin. Note that each port pin has a weak pullup circuit when functioning as an input and the p-channel pullup transistor is controlled by its respective PO bits. If the PO bit is set to 1, the weak pullup is on, if the PO bit is cleared to 0, the weak pullup is off and forces the port pin into three-state.
PD3 (13h, 00h) *Initialization:* *Read/Write Access:* *PD3.7 to PD3.0*	**Port 3 Direction Register** This register is cleared to 00h on all resets except power-fail reset. This register is unaffected by power-fail reset. Unrestricted read/write. **Port 3 Direction Register Bits 7:0.** PD3 is used to determine the direction of the port 3 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; data in the PO register is driven on the pin. When a bit is cleared to 0, its corresponding pin is used as an input, and allows an external signal to drive the pin. Note that each port pin has a weak pullup circuit when functioning as an input and the p-channel pullup transistor is controlled by its respective PO bits. If the PO bit is set to 1, the weak pullup is on, if the PO bit is cleared to 0, the weak pullup is off and forces the port pin into three-state.
CHPREV (13h, 00h) *Initialization:* *Read/Write Access:* *CHPREV.7 to CHPREV.0*	**Chip Revision Register (16-bit register)** The reset value of this register is dependent on the revision of the chip. Unrestricted read-only. **Chip Revision ID Register Bits 7:0.** The register is used to provide chip revision information. Read accesses return the chip revision in the lower byte and 00h in the upper byte (e.g., 00A1h).
PO4 (00h, 01h) *Initialization:* *Read/Write Access:* *PO4.5 to PO4.0* *PO4.7 to PO4.6*	**Port 4 Output Register (8-bit register)** This register is set to 00h on all forms of reset. Unrestricted read/write. **Port 4 Output Register Bits 5:0.** The PO4 register stores output data for port 4 when it is defined as an output port and controls whether the internal weak p-channel pullup transistor is enabled/disabled if a port pin is defined as an input. The contents of this register can be modified by a write access. Reading from the register returns the contents of the register. Changing the direction of port 4 does not change the data contents of the register. Reserved. Reads return 0.

REGISTER	DESCRIPTION
WUTC (04h, 01h) *Initialization:*	**Wake-Up Timer Control Register (8-bit register)** This register is cleared to 00h on all resets.
Read/Write Access:	Unrestricted read/write access except that bit 1 is read-only.
WUTC.0 (WTE)	**Wake-Up Timer Enable.** This control bit enables down counting of the 16-bit wake-up timer. Clearing this bit resets the internal wake-up timer down counter and resets WTF = 0. When WTE = 0, the initial down-counter starting value written into the WUT register is accessed on WUT register reads. Setting this bit from 0 to 1 loads the internal down counter with the initial value written to the WUT register, and enables down counting of the wake-up timer using the ring oscillator. When WTE = 1, the internal down counter value is accessed on WUT register reads. When WTE = 1, hardware setting of the WTF bit can generate an interrupt request to the CPU if also enabled globally.
WUTC.1 (WTF)	**Wake-Up Timer Flag.** This bit serves as a status bit/interrupt flag to denote when the wake-up timer down count has reached 0h. Hardware sets this bit whenever the wake-up down counter reaches 0h. The WTF bit is cleared by hardware any time the WTE bit is changed from 1 to 0.
WUTC.7 to WUT.2	Reserved. Reads return 0.
WUT (05h, 01h) *Initialization:*	**Wake-Up Timer Register (16-bit register)** This register is cleared to 0000h on all resets.
Read/Write Access:	Unrestricted write access. When WTE = 0, reads access the initial starting value written to WUT. When WTE = 1, reads access the internal down counter, thus multiple reads should be made to attain a stable value
WUT.15 to WUT.0	**Wake-Up Timer Value Register Bits 15:0.** These bits reflect the 16 bit value of the Wake-Up Timer. When WTE = 0, the initial wake-up timer starting value may be accessed by reads and writes of the WUT register. This initial starting value is retained internally so that triggering another wake-up timer interval requires only toggling of the WTE bit $1 \geq 0 \geq 1$. When WTE = 1, the internal down-counter value is accessed by reads of WUT, however, write access is still directed to the initial starting value (that is loaded to the down counter each time WTE is changed $0 \geq 1$). The 16-bit wake-up timer counts downward until reaching 0h unless disabled. The internal down counter is asynchronously reset to 0 anytime the wake-up timer is disabled by clearing WTE = 0. Once started, the WTF flag is set by hardware when the down count reaches 0h. The 0FFFFh starting state for the WUT[15:0] bits yield the maximum possible down-count range. Writing the WUT[15:0] bits establishes the down-count starting values shown below:

WUT[15:0]	DOWN-COUNT START VALUE
0001h	1
0002h	2
0003h	3
0004h	4
--Other--	(WUT[15:0])
0FFFEh	$(2^{16} - 2) = 65,534$
0FFFFh	$(2^{16} - 1) = 65,535$

PI4 (08h, 01h) *Initialization:*	**Port 4 Input Register** The reset value for this register is dependent on the logical states of the pins.
Read/Write Access:	Unrestricted read.
PI4.5 to PI4.0	**Port 4 Input Register Bits 5:0.** The PI4 register always reflects the logic state of its pins when read. Note that each port pin has a weak pullup circuit when functioning as an input and the p-channel pullup transistor is controlled by its respective PO bits. If the PO bit is set to 1, the weak pullup is on, if the PO bit is cleared to 0, the weak pullup is off and forces the port pin into three-state.
PI4.7 to PI4.6	Reserved. Reads return 0.

<table>
<tr><th>REGISTER</th><th>DESCRIPTION</th></tr>
<tr><td>PWCN (0Ch, 01h)
Initialization:</td><td>Power Control Register (16-bit register)
This register is set to 000000sss1100000b on all forms of reset.</td></tr>
<tr><td>Read/Write Access:</td><td>Unrestricted read/write.</td></tr>
<tr><td>PWCN.0 (PFD)</td><td>Power-Fail Monitor Disable. This bit determines whether the power-fail monitoring is enabled in stop mode when the regulator is off (REGEN = 0). When the regulator is enabled (as in normal operation or when REGEN = 1 in stop mode), the power-fail monitoring is always enabled, independent of the PFD bit setting. Otherwise, when set to 1, the power-fail reset detection for DVDD is disabled when the device is placed into stop mode. When placed into stop mode with PFD = 1 and REGEN = 0, the power-fail reset comparator shuts down. When configured to 0 with REGEN = 0, the power-fail monitoring function is enabled for detecting the condition DVDD < V_{RST} during stop mode.</td></tr>
<tr><td>PWCN.1 (PFIE)</td><td>Power-Fail Monitor Interrupt Enable. Setting this bit to 1 generates an interrupt to the CPU when PFI is set to 1. Clearing this bit to 0 disables the interrupt from generating. The power-fail monitor interrupt is not masked by the global interrupt enable (IGE) and is controlled solely by the PFIE bit.</td></tr>
<tr><td>PWCN.2 (PFI)</td><td>Power-Fail Monitor Interrupt. This bit is set to 1 when the supply voltage falls below the power-fail warning threshold. Clearing this bit to 0 clears the interrupt flag. However, if the supply voltage is still below the threshold, this flag is set again. Setting this bit to 1 causes an interrupt to the CPU when PFIE = 1. The power-fail monitor interrupt is not masked by the global interrupt enable (IGE) and is controlled solely by the PFIE bit.
It is not recommended to write to flash when the supply voltage drops below the power-fail warning level as there is uncertainty in the duration of continuous power supply. The user application should check the status of the PFI flag before initiating a flash program/erase operation.</td></tr>
<tr><td>PWCN.3 (REGEN)</td><td>Regulator Enable. When set to 1, the internal regulator remains powered on when the device is placed in stop mode. When cleared to 0, the internal regulator is shut down to conserve power. The regulator is always enabled outside of stop mode, independent of the REGEN bit setting.</td></tr>
<tr><td>PWCN.4 (IRTXOE)</td><td>IRTX Output Enable. The IRTXOE bit is used in conjunction with the IRTXOUT bit to determine the state of the IRTX pin when the IR timer is not enabled (i.e., IREN = 0). When the bit is set to 1, the IRTX pin is used as an output; data in the IRTXOUT bit is driven on the pin. When the bit is cleared to 0, the IRTX pin is three-stated (if IRTXOUT = 0) or weakly pulled up (if IRTXOUT = 1).
<table>
<tr><th>IRTXOE</th><th>IRTXOUT</th><th>IREN</th><th>IRTX PIN STATE</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>High-Z</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Weak Pullup</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Strong 0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Strong 1</td></tr>
<tr><td>X</td><td>X</td><td>1</td><td>IR Timer Tx Control</td></tr>
</table></td></tr>
</table>

REGISTER	DESCRIPTION
PWCN.5 (IRTXOUT)	**IRTX Output Pin Control.** This bit controls the output drive state for the IRTX pin when the IR timer is not enabled (i.e., IREN = 0) and when the IRTX pin has been enabled for output by IRTXOE = 1. When IREN = 0 and IRTXOE = 1, setting this bit to 1 enables a strong output high drive on the IRTX pin. Clearing this bit to 0 enables a strong output low drive on the IRTX pin. When IRTXOE = 0 and the IR timer is not enabled (IREN = 0), this bit controls the input mode for the IRTX pin. When IRTXOE = 0, the IRTX pin is three-state. When IRTXOE = 1, the pin is weakly pulled up.
PWCN.6 (IRRXWP)	**IRRX Weak Pullup Enable.** This bit controls the input mode of the IRRX pin. When this bit is set to 1, the internal weak pullup is enabled. When this bit is cleared to 0, the internal weak pullup is turned off, resulting in the three-state input mode.
PWCN.7 (PFRST)	**Power-Fail Reset Flag.** This bit is set to 1 whenever a power-fail reset occurs. It is unaffected by other forms of reset. This bit can be checked by software following a reset to determine if it was a power-fail reset that occurred. It should always be cleared by software following a reset to ensure that the source of any future reset can be determined correctly. Note that this bit is set anytime $V_{DD} < V_{RST}$. The WDCN.POR bit can be examined to determine whether V_{DD} was below the V_{P}OR threshold.
PWCN.9 to PWCN.8 (PFRCK[1:0])	**Power-Fail Reset Check Time Bits 1:0.** These bits are used to enable duty cycling of the V_{RST} power-monitoring circuitry during the time when V_{DD} is below the V_{RST} threshold, but has not reached the POR threshold. The duty cycling of the power-fail monitor during the V_{RST} condition is provided to reduce the time-averaged current consumption and extend the SRAM data-retention time when the battery voltage is low, but still provide adequate response time to exit the V_{RST} state if the battery source is replaced. These bits are reset only by POR (not even V_{RST}). The table below provides the bit settings and corresponding duty cycling of the power monitor check when $V_{POR} < V_{DD} < V_{RST}$.
	PFRCK[1:0] / **POWER-FAIL MONITOR CHECK INTERVAL (NANOPOWER RING OSCILLATOR CYCLES)** 00 / No interval defined (Monitor on always as normal) 01 / 2^{10} (~128ms for 8kHz nanopower ring oscillator frequency) 10 / 2^{11} (~256ms for 8kHz nanopower ring oscillator frequency) 11 / 2^{12} (~512ms for 8kHz nanopower ring oscillator frequency)
PWCN.15 to PWCN.10	Reserved. Read returns 0.
PD4 (10h, 01h) *Initialization:* *Read/Write Access:*	**Port 4 Direction Register** This register is cleared to 00h on all resets except power-fail reset. This register is unaffected by power-fail reset. Unrestricted read/write.
PD4.5 to PD4.0	**Port 4 Direction Register Bits 5:0.** PD4 is used to determine the direction of the port 4 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; data in the PO register is driven on the pin. When a bit is cleared to 0, its corresponding pin is used as an input, and allows an external signal to drive the pin. Note that each port pin has a weak pullup circuit when functioning as an input and the p-channel pullup transistor is controlled by its respective PO bits. If the PO bit is set to 1, the weak pullup is on, if the PO bit is cleared to 0, the weak pullup is off and forces the port pin into three-state.
PD4.7 to PD4.6	Reserved. Reads return 0.
TB0R (00h, 02h) *Initialization:* *Read/Write Access:*	**Timer B 0 Capture/Reload Value Register (16-bit register)** This register is cleared to 0000h on all forms of reset. Unrestricted read/write.
TB0R.15 to TB0R.0	**Timer B Capture/Reload Bits 15:0.** This register is used to capture the TBV value when Timer B is configured in capture mode. This register is also used as the 16-bit reload value when Timer B is configured in autoreload mode.

REGISTER	DESCRIPTION
TB0CN (01h, 02h) *Initialization:*	**Timer B 0 Control Register (16-bit register)** This register is cleared to 0000h on all forms of reset.
Read/Write Access:	Unrestricted read/write.
TB0CN.0 (CP/RLB)	**Capture/Reload Select.** This bit determines whether the capture or reload function is used for Timer B. Timer B functions in an autoreload mode following each overflow/underflow. See the TFB bit description for overflow/underflow condition. Setting this bit to 1 causes a Timer B capture to occur when a falling edge is detected on TBB if EXENB is 1. Clearing this bit to 0 causes an autoreload to occur when Timer B overflow or a falling edge is detected on TBB if EXENB is 1. It is not intended that the Timer B compare functionality should be used when operating in capture mode.
TB0CN.1 (ETB)	**Enable Timer B Interrupt.** Setting this bit to 1 enables the interrupt from the Timer B TFB and EXFB flags in TBCN. In Timer B clock output mode (TBOE = 1), the timer overflow flag (TFB) is still set on an overflow, however, the TBOE = 1 condition prevents this flag from causing an interrupt when ETB = 1.
TB0CN.2 (TRB)	**Timer B Run Control.** This bit enables Timer B operation when set to 1. Clearing this bit to 0 halts Timer B operation and preserves the current count in TBV.
TB0CN.3 (EXENB)	**Timer B External Enable.** Setting this bit to 1 enables the capture/reload function on the TBB pin for a negative transition (in up-counting mode). A reload results in TBV being reset to 0000h. Clearing this bit to 0 causes Timer B to ignore all external events on TBB pin. When operating in autoreload mode (CP/RLB = 0) with the PWM output functionality enabled, enabling the TBB input function (EXENB = 1) allows PWM output negative transitions to set the EXFB flag, however, no reload occurs as a result of the external negative edge detection.
TB0CN.4 (DCEN)	**Down-Count Enable.** This bit in conjunction with the TBB pin controls the direction that Timer B counts in 16-bit autoreload mode. Clearing this bit to 0 causes Timer B to count up only. Setting this bit to 1 enables the up/down-counting mode (i.e., it causes Timer B to count up if the TBB pin is 1 and to count down if the TBB pin is 0). When Timer B PWM output mode functionality is enabled along with up/down counting (DCEN = 1), the up/down-count control of Timer B is controlled internally based upon the count in relation to the register settings. In the compare modes, the DCEN bit controls whether the timer counts up and resets (DCEN = 0), or counts up and down (DCEN = 1).

<table>
<tr><th>REGISTER</th><th>DESCRIPTION</th></tr>
<tr><td>TB0CN.5 (TBOE)</td><td>Timer B Output Enable. Setting this bit to 1 enables the clock output function on the TBA pin if C/TB = 0. Timer B rollovers do not cause interrupts. Clearing this bit to 0 allows the TBA pin to function as either a standard port pin or a counter input for Timer B. Timer B 0 and Timer B 1 share the TBA pin. If both timers are configured to generate clock output, the Timer B 0 clock output special function takes priority over the Timer B 1 clock output.</td></tr>
<tr><td>TB0CN.6 (EXFB)</td><td>External Timer B Trigger Flag. When configured as a Timer (C/TB = 0), a negative transition on the TBB pin causes this flag to be set if (CP/RLB = EXENB = 1) or (CP/RLB = DCEN = 0 and EXENB = 1) or (CP/RLB = 0 and DCEN = EXENB = 1 and TBCS:TBCR ≠ 00b). When CP/RLB = 0 and DCEN = 1 and TBCS:TBCR = 00b, EXFB toggles whenever Timer B underflows or overflows. Overflow/underflow condition is the same as described for the TFB bit. In this mode, EXFB can be used as the 17th timer bit and does 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.</td></tr>
<tr><td>TB0CN.7 (TFB)</td><td>Timer B Overflow Flag. This bit is set when Timer B overflows from TBR or the count is equal to 0000h in down count mode. It must be cleared by software.</td></tr>
<tr><td>TB0CN.10 to TB0CN.8 (TBPS[2:0])</td><td>Timer B Clock Prescaler Bits 2:0. The TBPS[2:0] bits select the clock prescaler applied to the system clock input to Timer B. The TBPS[2:0] bits should be configured by the user when the timer is stopped (TRB = 0). While hardware does not prevent changing the TBPS[2:0] bits when the timer is running, the resulting behavior is indeterministic.
Timer B Clock = System Clock/$2^{(2xTBPS[2:0])}$
<table>
<tr><th>TBPS[2:0]</th><th>TIMER B INPUT CLOCK</th></tr>
<tr><td>000</td><td>Sysclk/1</td></tr>
<tr><td>001</td><td>Sysclk/4</td></tr>
<tr><td>010</td><td>Sysclk/16</td></tr>
<tr><td>011</td><td>Sysclk/64</td></tr>
<tr><td>100</td><td>Sysclk/256</td></tr>
<tr><td>101</td><td>Sysclk/1024</td></tr>
<tr><td>11x</td><td>Sysclk/1</td></tr>
</table></td></tr>
<tr><td>TB0CN.11 (TBCR)</td><td>TBB Pin Output Reset Mode</td></tr>
<tr><td>TB0CN.12 (TBCS)</td><td>TBB Pin Output Set Mode. These mode bits define whether the PWM mode output function is enabled on the TBB pin, the initial output starting state, and what compare mode output function is in effect. Note that the TBB pin still has certain input functionality when the PWM output function is enabled. Reference the PWM Output Function section for details on this mode.</td></tr>
<tr><td>TB0CN.14 to TBCN.13</td><td>Reserved. Reads return 0.</td></tr>
<tr><td>TB0CN.15 (C/TB)</td><td>Counter/Timer Select. This bit determines whether Timer B functions as a timer or counter. Setting this bit to 1 causes Timer B to count negative transitions on the TBA pin. Clearing this bit to 0 causes Timer B to function as a timer. The speed of Timer B is determined by the TBPS[2:0] bits of TBCN.</td></tr>
<tr><td>TB1R (02h, 02h)</td><td>Timer B Capture/Reload Value Register (see the TB0R register bit description)</td></tr>
<tr><td>TB1CN (03h, 02h)</td><td>Timer B Control Register (see the TB0CN register bit description)</td></tr>
</table>

<table>
<tr><th>REGISTER</th><th>DESCRIPTION</th></tr>
<tr><td>IRCN (04h, 02h)
Initialization:</td><td>Infrared Control Register (16-bit register)
This register is cleared to 0000h on all forms of reset.</td></tr>
<tr><td>Read/Write Access:</td><td>Unrestricted read/write.</td></tr>
<tr><td>IRCN.0 (IREN)</td><td>IR Enable. This register bit enables the IR module. Setting this bit to 1 starts the operating mode as defined by IRMODE bit. Clearing this bit to 0 terminates IR operation.</td></tr>
<tr><td>IRCN.1 (IRMODE)</td><td>IR Mode. This register bit controls the IR module operation mode.
<table>
<tr><th>IRMODE</th><th>IR OPERATION MODE</th></tr>
<tr><td>0</td><td>Receive Mode</td></tr>
<tr><td>1</td><td>Transmit Mode</td></tr>
</table></td></tr>
<tr><td>IRCN.2 (IRTXPOL)</td><td>IR TX Polarity Select. When the IR timer is enabled (IREN = 1), this bit selects the starting/idle logic state, and the carrier polarity for the IRTX transmit output. This bit also impacts the polarity of the IRTXM envelope when the independent modulator transmit output mode is enabled (IRENV[1:0] = 01b or 10b). When IRENV[1:0] = 01b or 10b, the latched IRDATA bit is directly output to the IRTXM pin as the envelope when IRTXPOL = 0. When IRTXPOL = 1, the complement of the latched IRDATA bit is output.</td></tr>
<tr><td>IRCN.3 (IRDATA)</td><td>IR Data. This register bit defines how the carrier is modulated in transmit mode and in receive mode, it contains the state of IRRX when a qualified capture event happens. When IR transmit mode is in effect, setting IRDATA = 1 enables the output of the carrier module (as affected by IRTXPOL) to be visible on the IRTX pin. When IRDATA = 0, the IR module is put in the idle state and IRTXPOL is output onto IRTX. In receive mode, the IRDATA bit contains the latched state of the IRRX pin each time a capture event occurs.</td></tr>
<tr><td>IRCN.5 to IRCN.4 (IRRXSEL[1:0])</td><td>IR Receive Edge Select Bits. These bits define which edge of the input signal trigger a receive capture function when enabled.
<table>
<tr><th>IRRXSEL[1:0]</th><th>IR RECEIVE MODE</th></tr>
<tr><td>00</td><td>Trigger on falling edge</td></tr>
<tr><td>01</td><td>Trigger on rising edge</td></tr>
<tr><td>10</td><td>Trigger on both rising and falling edge</td></tr>
<tr><td>11</td><td>Reserved (disables edge detection)</td></tr>
</table></td></tr>
<tr><td>IRCN.6 (IRCFME)</td><td>IR Clock Frequency Mux Enable. In receive mode, setting this bit to 1 enables direct clocking of the IRV register using the defined IRCLK during the IR receive operation. Clearing this bit to 0 results in IRV counting of the IRCA-defined carrier during the receive operation. Using IRCFME = 1 allows IRCLK clock resolution when capturing whereas IRCFME = 0 allows only (IRCLK/2) resolution when IRCA = 0000h. In transmit mode, setting this bit to 1 enables direct clocking of the IRV register down counter with IRCLK so that intervals can be generated with IRCLK resolution. When this bit is cleared to 0, the IRV down counter is clocked with the IRCA-defined carrier clock, resulting in IRV interval generation according to the defined carrier frequency.</td></tr>
<tr><td>IRCN.7 (IRXRL)</td><td>IR Receive Reload Enable. Setting this bit to 1 enables automatic reload of the IRV register with 0000h whenever a qualified edge event capture occurs during the IR receive operation. If IRXRL = 0, the IRV register is not reloaded with 0000h, but continues running during the IR receive operation.</td></tr>
<tr><td>IRCN.9 to IRCN.8 (IRENV[1:0])</td><td>IR Envelope Mode Bits 1:0. Setting either of these bits (but not both) to 1 enables the envelope modulation signal (based upon the IRDATA and IRTXPOL bits) to be output separately to the IRTXM pin during transmit mode. When the bits are both cleared to 0 or set to 1, the standard internal modulation is performed during IR transmit mode and the envelope signal is not output to the IRTXM pin. When the envelope mode is enabled, it is possible to output either the modulated or unmodulated carrier to the IRTX pin (see table).
<table>
<tr><th>IRENV[1:0]</th><th>IRTX OUTPUT</th></tr>
<tr><td>00 or 11</td><td>Envelope mode disabled.
Standard IRTX modulation (default).</td></tr>
<tr><td>01</td><td>Standard IRTX modulation.</td></tr>
<tr><td>10</td><td>Constant IRTX carrier (unmodulated).</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>REGISTER</th><th>DESCRIPTION</th></tr>
<tr><td>IRCN.12 to IRCN.10 (IRDIV[2:0])</td><td>IR Clock Divide Bits. These two bits select the divide ratio for the IR input clock.
<table>
<tr><th>IRDIV[2:0]</th><th>IR INPUT CLOCK-DIVIDE RATIO</th></tr>
<tr><td>000</td><td>f_{SYSCLK}/1</td></tr>
<tr><td>001</td><td>f_{SYSCLK}/2</td></tr>
<tr><td>010</td><td>f_{SYSCLK}/4</td></tr>
<tr><td>011</td><td>f_{SYSCLK}/8</td></tr>
<tr><td>100</td><td>f_{SYSCLK}/16</td></tr>
<tr><td>101</td><td>f_{SYSCLK}/32</td></tr>
<tr><td>110</td><td>f_{SYSCLK}/64</td></tr>
<tr><td>111</td><td>f_{SYSCLK}/128</td></tr>
</table></td></tr>
<tr><td>IRCN.15 to IRCN.13</td><td>Reserved. Reads return 0.</td></tr>
<tr><td>IRCA (05h, 02h)
Initialization:</td><td>IR Carrier Register (16-bit register)
This register is cleared to 0000h on all forms of reset.</td></tr>
<tr><td>Read/Write Access:</td><td>Unrestricted read/write.</td></tr>
<tr><td>IRCA.7 to IRCA.0 (IRCAL[7:0])</td><td>IR Carrier Low Byte Bits 7:0. The IRCAL byte defines the number of IR input clocks during carrier low time. The carrier low time = IRCAL[7:0] + 1.</td></tr>
<tr><td>IRCA.15 to IRCA.8 (IRCAH[7:0])</td><td>IR Carrier High Byte Bits 7:0. The IRCAH byte defines the number of IR input clocks during carrier high time. The carrier high time = IRCAH[7:0] + 1.</td></tr>
<tr><td>IRMT (06h, 02h)
Initialization:</td><td>IR Modulator Time (16-bit register)
This register is cleared to 0000h on all forms of reset.</td></tr>
<tr><td>Read/Write Access:</td><td>Unrestricted read/write.</td></tr>
<tr><td>IRMT.15 to IRMT.0</td><td>IR Modulator Time Bits 15:0. The IRMT register is a 16-bit register that defines the IRDATA active time during transmit mode. In receive mode (when RXBCNT = 0), it is used to capture the IRV value on qualified IRRXSEL edges. In receive mode (when RXBCNT = 1), the IRMT register increments on detection of selected IRRXSEL edge(s). When RXBCNT is changed from 0 to 1, the IRMT register is set to 0001h by hardware.</td></tr>
<tr><td>IRCNB (07h, 02h)
Initialization:</td><td>Infrared Control Register B (8-bit register)
This register is cleared to 00h on all forms of reset.</td></tr>
<tr><td>Read/Write Access:</td><td>Unrestricted read/write.</td></tr>
<tr><td>IRCNB.0 (IROV)</td><td>IR Timer Overflow Flag. This flag is set to 1 when the IR timer overflows from 0FFFFh to 0000h in receive mode. This bit must be cleared to 0 by software once it is set.</td></tr>
<tr><td>IRCNB.1 (IRIF)</td><td>IR Interrupt Flag. This flag is set to 1 during transmit when the IR timer reloads its value and in receive mode (if RXBCNT = 0), when a capture occurs. In receive mode (when RXBCNT = 1), this flag is set whenever the IRCA*2 interval timer expires. This bit must be cleared to 0 by software once it is set.</td></tr>
<tr><td>IRCNB.2 (IRIE)</td><td>IR Interrupt Enable. Setting this bit to 1 enables an interrupt be generated to the CPU when the IR timer overflow (IROV) or IR interrupt flag is set (IRIF). Clearing this bit to 0 disables IR timer interrupt generation.</td></tr>
<tr><td>IRCNB.3 (RXBCNT)</td><td>Receive Carrier Burst-Count Enable. Setting this bit to 1 enables the carrier burst counting mode for the IR timer when operating in receive mode. This bit is not meaningful for the transmit mode. Whenever software changes RXBCNT from 0 to 1, the IRMT register is set to 0001h by hardware. When RXBCNT = 1, the IR timer receive mode is modified in the following ways: 1) The IRV register is not captured to the IRMT register on detection of the IRRXSEL[1:0] selected edge(s); 2) The IRMT register is incremented on detection of the IRRXSEL[1:0] selected edge(s); 3) The IRIF flag is no longer set on capture edge detection; 4) An IRCA x 2 interval timer is enabled and upon expiration the IRIF flag is set. When RXBCNT = 0, the receive carrier burst-count mode is disabled and normal receive capture functionality can be used.</td></tr>
<tr><td>IRCNB.7 to IRCNB.4</td><td>Reserved. Reads return 0.</td></tr>
</table>

REGISTER	DESCRIPTION
TB0C (08h, 02h) *Initialization:* *Read/Write Access:* *TB0C.15 to TB0C.0*	**Timer B 0 Compare Register (16-bit register)** This register is cleared to 0000h on all forms of reset. Unrestricted read/write. **Timer B Compare Bits 15:0.** This register is used for comparison versus the TBV value when Timer B is operated in compare mode.
TB0V (09h, 02h) *Initialization:* *Read/Write Access:* *TB0V.15 to TB0V.0*	**Timer B 0 Value Register (16-bit register)** The Timer B Value is cleared to 0000h on all forms of reset. Unrestricted read/write. **Timer B Value Bits 15:0.** This register is used to load and read the 16-bit Timer B value.
TB1C (0Ah, 02h)	**Timer B Compare Register** (see the TB0C register bit description)
TB1V (0Bh, 02h)	**Timer B Value Register** (see the TB0V register bit description)
IRV (0Ch, 02h) *Initialization:* *Read/Write Access:* *IRV.15 to IRV.0*	**IR Value Register (16-bit register)** This register is cleared to 0000h on all forms of reset. Unrestricted read/write. **IR Value Register Bits 15:0.** The IRV register is a 16-bit register that holds the current IR timer value. The IR timer value starts counting when the IREN bit is set to 1. It stops counting when the IREN bit is cleared to 0 and retains the current timer value.
SCON0 (00h, 03h) *Initialization:* *Read/Write Access:* *SCON0.0 (RI)* *SCON0.1 (TI)* *SCON0.2 (RB8)* *SCON0.3 (TB8)* *SCON0.4 (REN)* *SCON0.5 (SM2)*	**Serial Port 0 Control Register** The serial port control is cleared to 00h on all forms of reset. Unrestricted read/write. **Receive Interrupt Flag.** 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. **Transmit Interrupt Flag.** 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. **9th Received Bit State.** 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. **9th Transmission Bit State.** This bit defines the state of the 9th transmission bit in serial port modes 2 and 3. **Receive Enable**. REN_0 = 0: Serial port 0 receiver disabled. REN_0 = 1: Serial port 0 receiver enabled for modes 1, 2, and 3. Initiate synchronous reception for mode 0. **Serial Port Mode Bit 2.** 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. This bit also used to support mode 0 for clock selection: SM2 = 0: Clock is divided by 12. SM2 = 1: Clock is divided by 4.

<table>
<tr><th>REGISTER</th><th colspan="7">DESCRIPTION</th></tr>
<tr><td>SCON0.6 (SM1)
SCON0.7 (SM0/FE)</td><td colspan="7">Serial Port 0 Mode Bits 1:0 (when FEDE is 0). When FEDE is set to 1, this bit is the Framing Error Flag that 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.</td></tr>
<tr><td></td><td>MODE</td><td>SM2</td><td>SM1</td><td>SM0</td><td>FUNCTION</td><td>LENGTH (BITS)</td><td>PERIOD</td></tr>
<tr><td></td><td>0</td><td>0</td><td>0</td><td>0</td><td>Synchronous</td><td>8</td><td>12 system clocks</td></tr>
<tr><td></td><td>0</td><td>1</td><td>0</td><td>0</td><td>Synchronous</td><td>8</td><td>4 system clocks</td></tr>
<tr><td></td><td>1</td><td>x</td><td>1</td><td>0</td><td>Asynchronous</td><td>10</td><td>64/16 baud clocks (SMOD = 0/1)</td></tr>
<tr><td></td><td>2</td><td>0</td><td>0</td><td>1</td><td>Asynchronous</td><td>11</td><td>64/32 system clocks (SMOD = 0/1)</td></tr>
<tr><td></td><td>2</td><td>1</td><td>0</td><td>1</td><td>Asynchronous (MP)</td><td>11</td><td>64/32 system clocks (SMOD = 0/1)</td></tr>
<tr><td></td><td>3</td><td>0</td><td>1</td><td>1</td><td>Asynchronous</td><td>11</td><td>64/16 baud clocks (SMOD = 0/1)</td></tr>
<tr><td></td><td>3</td><td>1</td><td>1</td><td>1</td><td>Asynchronous (MP)</td><td>11</td><td>64/16 baud clocks (SMOD = 0/1)</td></tr>
<tr><td>SBUF0 (01h, 03h)
Initialization:
Read/Write Access:
SBUF0.7 to SBUF0.0</td><td colspan="7">Serial Data Buffer 0
This buffer is cleared to 00h on all forms of reset.
Unrestricted read/write.
Serial Data Buffer 0 Bits 7:0. Data for serial port 0 is read from or written to this location. The serial transmit and receive buffers are separate but both are addressed at this location.</td></tr>
</table>

<table>
<tr><th>REGISTER</th><th>DESCRIPTION</th></tr>
<tr><td>SCON1 (02h, 03h)
Initialization:</td><td>Serial Port 1 Control Register
The serial port control is cleared to 00h on all forms of reset.</td></tr>
<tr><td>Read/Write Access:</td><td>Unrestricted read/write.</td></tr>
<tr><td>SCON1.0 (RI)</td><td>Receive Interrupt Flag. 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.</td></tr>
<tr><td>SCON1.1 (TI)</td><td>Transmit Interrupt Flag. 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.</td></tr>
<tr><td>SCON1.2 (RB8)</td><td>9th Received Bit State. 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.</td></tr>
<tr><td>SCON1.3 (TB8)</td><td>9th Transmission Bit State. This bit defines the state of the 9th transmission bit in serial port modes 2 and 3.</td></tr>
<tr><td>SCON1.4 (REN)</td><td>Receive Enable
REN_0 = 0: Serial port 0 receiver disabled.
REN_0 = 1: Serial port 0 receiver enabled for modes 1, 2, and 3. Initiate synchronous reception for mode 0.</td></tr>
<tr><td>SCON1.5 (SM2)</td><td>Serial Port 1 Mode Bit 2. 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. This bit also used to support mode 0 for clock selection:
SM2 = 0: Clock is divided by 12.
SM2 = 1: Clock is divided by 4.</td></tr>
<tr><td>SCON1.6 (SM1)
SCON1.7 (SM0/FE)</td><td>Serial Port 1 Mode Bits 1:0 (when FEDE is 0). When FEDE is set to 1, this bit is the Framing Error Flag that 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.
<table>
<tr><th>MODE</th><th>SM2</th><th>SM1</th><th>SM0</th><th>FUNCTION</th><th>LENGTH (BITS)</th><th>PERIOD</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Synchronous</td><td>8</td><td>12 system clocks</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>Synchronous</td><td>8</td><td>4 system clocks</td></tr>
<tr><td>1</td><td>x</td><td>1</td><td>0</td><td>Asynchronous</td><td>10</td><td>64/16 baud clocks (SMOD = 0/1)</td></tr>
<tr><td>2</td><td>0</td><td>0</td><td>1</td><td>Asynchronous</td><td>11</td><td>64/32 system clocks (SMOD = 0/1)</td></tr>
<tr><td>2</td><td>1</td><td>0</td><td>1</td><td>Asynchronous (MP)</td><td>11</td><td>64/32 system clocks (SMOD = 0/1)</td></tr>
<tr><td>3</td><td>0</td><td>1</td><td>1</td><td>Asynchronous</td><td>11</td><td>64/16 baud clocks (SMOD = 0/1)</td></tr>
<tr><td>3</td><td>1</td><td>1</td><td>1</td><td>Asynchronous (MP)</td><td>11</td><td>64/16 baud clocks (SMOD = 0/1)</td></tr>
</table></td></tr>
</table>

REGISTER	DESCRIPTION
SBUF1 (03h, 03h) *Initialization:*	**Serial Data Buffer 1** This buffer is cleared to 00h on all forms of reset.
Read/Write Access:	Unrestricted read/write.
SBUF1.7 to SBUF1.0	**Serial Data Buffer 1 Bit 7:0.** Data for serial port 0 is read from or written to this location. The serial transmit and receive buffers are separate but both are addressed at this location.
SPIB (04h, 03h) *Initialization:*	**SPI Data Buffer (16-bit register)** This buffer is cleared to 0000h on all forms of reset.
Read/Write Access:	Unrestricted read, write is allowed outside of a transfer cycle; when the STBY bit is set, write is blocked and causes write collision error.
SPIB.15 to SPIB.0	**SPI Data Buffer Bits 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.
SPICN (05h, 03h) *Initialization:*	**SPI Control Register** This buffer is cleared to 00h on all forms of reset.
Read/Write Access:	Unrestricted read/write except bit 7 is read-only.
SPICN.0 (SPIEN)	**SPI Enable.** Setting this bit to 1 enables the SPI module and its baud-rate generator for SPI operation. Clearing this bit to 0 disables the SPI module and its baud-rate generator.
SPICN.1 (MSTM)	**Master Mode Enable.** MSTM functions as a master mode enable bit for the SPI module. When MSTM is set to 1, the SPI operates as a master. When MSTM is cleared to 0, the SPI module operates in slave mode. Note that this bit can be set from 0 to 1 only when the $\overline{\text{SSEL}}$ signal is deasserted.
SPICN.2 (MODFE)	**Mode Fault Enable.** When set to 1 in master mode, this bit enables the use of $\overline{\text{SSEL}}$ input as a mode fault signal; when cleared to 0, the $\overline{\text{SSEL}}$ has no function and its port pin can be used for other purposes. 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.
SPICN.3 (MODF)	**Mode Fault Flag.** This bit is the mode fault flag when the SPI is operating as a master. When mode fault detection is enabled as MODFE = 1 in master mode, a detection of a high to low transition on the $\overline{\text{SSEL}}$ pin signifies a mode fault and sets the MODF to 1. This bit must be cleared to 0 by software once set. Setting this bit to 1 by software causes an interrupt if enabled. This flag has no meaning in slave mode.
SPICN.4 (WCOL)	**Write Collision Flag.** This bit indicates a write collision when set to 1. This is caused by attempting to write to the SPIB while a transfer cycle is in progress. This bit must be cleared to 0 by software once set. Setting this bit to 1 by software causes an interrupt if enabled.
SPICN.5 (ROVR)	**Receive Overrun Flag.** This bit indicates a receive overrun when set to 1. This is caused by two or more characters have been received since the last read by the processor. The newer data is lost. This bit must be cleared to 0 by software once set. Setting this bit to 1 by software causes an interrupt if enabled.
SPICN.6 (SPIC)	**SPI Transfer Complete Flag.** This bit indicates the completion of a transfer cycle when set to 1. This bit must be cleared to 0 by software once set. Setting this bit to 1 by software causes an interrupt if enabled.
SPICN.7 (STBY)	**SPI Transfer Busy Flag.** This bit is used to indicate the current status of the SPI module. STBY is set to 1 when starting a SPI transfer cycle and is cleared to 0 when the transfer cycle is completed. This bit is controlled by hardware and is read-only for user software.
PR0 (08h, 03h) *Initialization:*	**Phase Register 0** The phase register is cleared to 0000h on all forms of reset.
Read/Write Access:	Unrestricted read/write.
PR0.15 to PR0.0	**Phase Register Bits 15:0.** This register is used to load and read the 16-bit value in the phase register that determines the baud rate for the serial port 0.

REGISTER	DESCRIPTION
SMD0 (09h, 03h) *Initialization:*	**Serial Port Mode Register 0** This register is cleared to 00h on all forms of reset.
Read/Write Access:	Unrestricted read/write.
SMD0.0 (FEDE0)	**Framing Error-Detection Enable.** This bit selects the function of SM0 (SCON0.7): FEDE = 0: SCON0.7 functions as SM0 for serial port mode selection. FEDE = 1: SCON0.7 is converted to the framing error (FE) flag.
SMD0.1 (SMOD0)	**Serial Port 0 Baud-Rate Select.** The SMOD selects the final baud rate for the asynchronous mode: SMOD = 1: 16 times the baud clock for mode 1 and 3, 32 times the system clock for mode 2. SMOD = 0: 64 times the baud clock for mode 1 and 3, 64 times the system clock for mode 2.
SMD0.2 (ESI0)	**Enable Serial Port 0 Interrupt.** Setting this bit to 1 enables interrupt requests generated by the RI or TI flags in SCON0. Clearing this bit to 0 disables the serial port interrupt.
SMD0.7 to SMD0.3	Reserved. Reads return 0.
PR1 (0Ah, 03h) *Initialization:*	**Phase Register 1** The phase register is cleared to 0000h on all forms of reset.
Read/Write Access:	Unrestricted read/write.
PR1.15 to PR1.0	**Phase Register 1 Bits 15:0.** This register is used to load and read the 16-bit value in the phase register that determines the baud rate for the serial port 1.
SMD1 (0Bh, 03h) *Initialization:*	**Serial Port Mode Register 1** This register is cleared to 00h on all forms of reset.
Read/Write Access:	Unrestricted read/write.
SMD1.0 (FEDE1)	**Framing Error-Detection Enable.** This bit selects the function of SM0 (SCON1.7): FEDE = 0: SCON1.7 functions as SM0 for serial port mode selection. FEDE = 1: SCON1.7 is converted to the framing error (FE) flag.
SMD1.1 (SMOD1)	**Serial Port 1 Baud-Rate Select**. The SMOD selects the final baud rate for the asynchronous mode: SMOD = 1: 16 times the baud clock for mode 1 and 3, 32 times the system clock for mode 2. SMOD = 0: 64 times the baud clock for mode 1 and 3, 64 times the system clock for mode 2.
SMD1.2 (ESI1)	**Enable Serial Port 1 Interrupt.** Setting this bit to 1 enables interrupt requests generated by the RI or TI flags in SCON1. Clearing this bit to 0 disables the serial port interrupt.
SMD1.7 to SMD1.3	**Reserved, read returns 0.**

REGISTER	DESCRIPTION
SPICF (0Ch, 03h) *Initialization:*	**SPI Configuration Register** This buffer is cleared to 00h on all forms of reset.
Read/Write Access:	Unrestricted read/write.
SPICF.0 (CKPOL)	**Clock Polarity Select.** This bit is used with the CKPHA bit to determine the SPI transfer format. When the CKPOL is set to 1, the SPI uses the clock falling edge as an active edge. When the CKPOL is cleared to 0, the SPI selects the clock rising edge as an active edge.
SPICF.1 (CKPHA)	**Clock Phase Select**. This bit is used with the CKPOL bit to determine the SPI transfer format. When the CKPHA is set to 1, the SPI samples input data at an inactive edge. When the CKPHA is cleared to 0, the SPI samples input data at an active edge.
SPICF.2 (CHR)	**Character Length Bit.** The CHR bit determines the character length for an SPI transfer cycle. A character can consist 8 or 16 bits in length. When CHR bit is 0, the character is 8 bits; when CHR is set to 1, the character is 16 bits.
SPICF.5 to SPICF.3	Reserved. Reads return 0.
SPICF.6 (SAS)	**Slave Active Select.** This bit is used to determine the $\overline{SSEL}$ active state. When the SAS is cleared to 0, the $\overline{SSEL}$ is active low and responds to an external low signal. When the SAS is set to 1, the $\overline{SSEL}$ is active high.
SPICF.7 (ESPII)	**SPI Interrupt Enable**. Setting this bit to 1 enables the SPI interrupt when MODF, WCOL, ROVR, or SPIC flags are set. Clearing this bit to 0 disables the SPI interrupt.
SPICK (0Dh, 03h) *Initialization:*	**SPI Clock Register** This buffer is cleared to 00h on all forms of reset.
Read/Write Access:	Unrestricted read/write.
SPICK.7 to SPICK.0 (CKR[7:0])	**Clock-Divide Ratio Bits 7:0.** These bits select one of the 256 divide ratios (0 to 255) used for the baud-rate generator, with bit 7 as the most significant bit. The frequency of the SPI baud rate is calculated using the following equation: SPI Baud Rate = 0.5 x System Clock/(divide ratio + 1) This register has no function when operating in slave mode and the clock generation circuitry should be disabled.

SECTION 6: GENERAL-PURPOSE I/O MODULE

This section contains the following information:

6.1 Port Pin Register Descriptions 6-3
6.1.1 Port Pin Example 1: Driving Outputs on Port 0 6-7
6.1.2 Port Pin Example 2: Receiving Inputs on Port 1 6-7
6.2 External Interrupt Register Descriptions 6-7

LIST OF TABLES

Table 6-1. Port Pin Special Functions 6-2
Table 6-2. MAXQ610 Port Pin Input/Output States 6-3

SECTION 6: GENERAL-PURPOSE I/O MODULE

The MAXQ610 provides 38 port pins for general-purpose I/O that are grouped into eight port pins on port 0 to port 3 and six port pins on port 4. Each of these port pins has the following features:

- CMOS output drivers
- Schmitt trigger inputs
- Optional weak pullup to V_{DD} when operating in input mode

From a software perspective, each port appears as a group of peripheral registers with unique addresses. Special function pins can also be used as general-purpose IO pins when the special functions are disabled (ports 2 and 3).

Table 6-1. Port Pin Special Functions

PORT PIN	DIRECTION	SPECIAL FUNCTION	ENABLED WHEN
P0.0	Input/Output	IR modulator/envelope output (IRTXM)	IRENV[1:0] = 01b or 10b
P0.1	Input/Output	Serial USART 0 Receive (RX0)	REN = 1
P0.2	Input/Output	Serial USART 0 Transmit (TX0)	SBUF0 written
P0.3	Input/Output	Serial USART 1 Receive (RX1)	REN = 1
P0.4	Input/Output	Serial USART 1 Transmit (TX1)	SBUF1 written
P0.5	Input/Output	Timer 0 Input (TBA0)/Timer 1 Input (TBA1)	C/TB = 1 or TBOE = 1
P0.6	Input/Output	Timer 0 PWM Output (TBB0)	EXENB = 1 or TBCR:TBCS ≠ 00b
P0.7	Input/Output	Timer 1 PWM Output (TBB1)	EXENB = 1 or TBCR:TBCS ≠ 00b
P1.0	Input/Output	External Interrupt 0 (INT0)	EX0 = 1
P1.1	Input/Output	External Interrupt 1 (INT1)	EX1 = 1
P1.2	Input/Output	External Interrupt 2 (INT2)	EX2 = 1
P1.3	Input/Output	External Interrupt 3 (INT3)	EX3 = 1
P1.4	Input/Output	External Interrupt 4 (INT4)	EX4 = 1
P1.5	Input/Output	External Interrupt 5 (INT5)	EX5 = 1
P1.6	Input/Output	External Interrupt 6 (INT6)	EX6 = 1
P1.7	Input/Output	External Interrupt 7 (INT7)	EX7 = 1
P2.0	Input/Output	SPI Master Out-Slave In (MOSI)	SPIEN = 1
P2.1	Input/Output	SPI Master In-Slave Out (MISO)	SPIEN = 1
P2.2	Input/Output	SPI Slave Clock (SCLK)	SPIEN = 1
P2.3	Input/Output	SPI Slave Select ($\overline{SSEL}$)	SPIEN = 1
P2.4	Input/Output	JTAG interface—TAP Clock (TCK)	(SC.7) TAP =
P2.5	Input/Output	JTAG interface—TAP Data Input (TDI)	(SC.7) TAP= 1
P2.6	Input/Output	JTAG interface—TAP Mode Select (TMS)	(SC.7) TAP = 1
P2.7	Input/Output	JTAG interface—TAP Data Output (TDO)	(SC.7) TAP = 1
P3.0	Input/Output	External Interrupt 8 (INT8)	EX8 = 1
P3.1	Input/Output	External Interrupt 9 (INT9)	EX9 = 1
P3.2	Input/Output	External Interrupt 10 (INT10)	EX10 = 1
P3.3	Input/Output	External Interrupt 11 (INT11)	EX11 = 1
P3.4	Input/Output	External Interrupt 12 (INT12)	EX12 = 1
P3.5	Input/Output	External Interrupt 13 (INT13)	EX13 = 1
P3.6	Input/Output	External Interrupt 14 (INT14)	EX14 = 1
P3.7	Input/Output	External Interrupt 15 (INT15)	EX15 = 1
P4.0	Input/Output	—	—
P4.1	Input/Output	—	—

Table 6-1. Port Pin Special Functions (continued)

PORT PIN	DIRECTION	SPECIAL FUNCTION	ENABLED WHEN
P4.2	Input/Output	—	—
P4.3	Input/Output	—	—
P4.4	Input/Output	—	—
P4.5	Input/Output	—	—

All these special functions are disabled by default with the exception of the JTAG interface pins, which are enabled by default following any reset.

The port pin input/output states can be defined as shown in Table 6-2.

Table 6-2. MAXQ610 Port Pin Input/Output States

PDx.y	POx.y	PORT PIN MODE	PORT PIN (Px.y) STATE
0	0	Input	Three-state
0	1	Input	Weak pullup HIGH
1	0	Output	Strong drive LOW
1	1	Output	Strong drive HIGH

6.1 Port Pin Register Descriptions

The following peripheral registers are used to control the general-purpose I/O and external interrupt features specific to the MAXQ610.

Register Name **PO0**
Register Description **Port 0 Output Register**
Register Address **M0[00h]**

Bit #	7	6	5	4	3	2	1	0
Name	PO0.7	PO0.6	PO0.5	PO0.4	PO0.3	PO0.2	PO0.1	PO0.0
Reset	s	s	s	s	s	s	s	s
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0: Port 0 Output. This register stores the data that is output on any of the pins of port 0 that have been defined as output pins. If the port pins are in input mode, this register controls the weak pullup enable for each pin. Changing the data direction of any pins for this port (through register PD0) does not affect the value in this register.

Register Name **PO1**
Register Description **Port 1 Output Register**
Register Address **M0[01h]**

Bit #	7	6	5	4	3	2	1	0
Name	PO1.7	PO1.6	PO1.5	PO1.4	PO1.3	PO1.2	PO1.1	PO1.0
Reset	s	s	s	s	s	s	s	s
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0: Port 1 Output. This register stores the data that is output on any of the pins of port 1 that have been defined as output pins. If the port pins are in input mode, this register controls the weak pullup enable for each pin. Changing the data direction of any pins for this port (through register PD0) does not affect the value in this register.

Register Name **PO2**
Register Description **Port 2 Output Register**
Register Address **M0[02h]**

Bit #	7	6	5	4	3	2	1	0
Name	PO2.7	PO2.6	PO2.5	PO2.4	PO2.3	PO2.2	PO2.1	PO2.0
Reset	s	s	s	s	s	s	s	s
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0: Port 2 Output. This register stores the data that is output on any of the pins of port 2 that have been defined as output pins. If the port pins are in input mode, this register controls the weak pullup enable for each pin. Changing the data direction of any pins for this port (through register PD2) does not affect the value in this register.

Register Name **PO3**
Register Description **Port 3 Output Register**
Register Address **M0[03h]**

Bit #	7	6	5	4	3	2	1	0
Name	PO3.7	PO3.6	PO3.5	PO3.4	PO3.3	PO3.2	PO3.1	PO3.0
Reset	s	s	s	s	s	s	s	s
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0: Port 3 Output. This register stores the data that is output on any of the pins of port 3 that have been defined as output pins. If the port pins are in input mode, this register controls the weak pullup enable for each pin. Changing the data direction of any pins for this port (through register PD3) does not affect the value in this register.

Register Name **PO4**
Register Description **Port 4 Output Register**
Register Address **M1[04h]**

Bit #	7	6	5	4	3	2	1	0
Name	—	—	PO4.5	PO4.4	PO4.3	PO4.2	PO4.1	PO4.0
Reset	0	0	s	s	s	s	s	s
Access	r	r	rw	rw	rw	rw	rw	rw

Bits 5:0: Port 4 Output. This register stores the data that is output on any of the pins of port 4 that have been defined as output pins. If the port pins are in input mode, this register controls the weak pullup enable for each pin. Changing the data direction of any pins for this port (through register PD4) does not affect the value in this register.

Register Name **PI0**
Register Description **Port 0 Input Register**
Register Address **M0[08h]**

Bit #	7	6	5	4	3	2	1	0
Name	PI0.7	PI0.6	PI0.5	PI0.4	PI0.3	PI0.2	PI0.1	PI0.0
Reset	s	s	s	s	s	s	s	s
Access	r	r	r	r	r	r	r	r

Bits 7:0: Port 0 Input Bits. The read values of these bits reflect the logic states present at port 0 pins P0.0 to P0.7.

Register Name **PI1**
Register Description **Port 1 Input Register**
Register Address **M0[09h]**

Bit #	7	6	5	4	3	2	1	0
Name	PI1.7	PI1.6	PI1.5	PI1.4	PI1.3	PI1.2	PI1.1	PI1.0
Reset	s	s	s	s	s	s	s	s
Access	r	r	r	r	r	r	r	r

Bits 7:0: Port 1 Input Bits. The read values of these bits reflect the logic states present at port 1 pins P1.0 to P1.7.

Register Name **PI2**
Register Description **Port 2 Input Register**
Register Address **M0[0Ah]**

Bit #	7	6	5	4	3	2	1	0
Name	PI2.7	PI2.6	PI2.5	PI2.4	PI2.3	PI2.2	PI2.1	PI2.0
Reset	s	s	s	s	s	s	s	s
Access	r	r	r	r	r	r	r	r

Bits 7:0: Port 2 Input Bits. The read values of these bits reflect the logic states present at port 2 pins P2.0 to P2.7.

Register Name **PI3**
Register Description **Port 3 Input Register**
Register Address **M0[0Bh]**

Bit #	7	6	5	4	3	2	1	0
Name	PI3.7	PI3.6	PI3.5	PI3.4	PI3.3	PI3.2	PI3.1	PI3.0
Reset	s	s	s	s	s	s	s	s
Access	r	r	r	r	r	r	r	r

Bits 7:0: Port 3 Input Bits. The read values of these bits reflect the logic states present at port 3 pins P3.0 to P4.7.

Register Name **PI4**
Register Description **Port 4 Input Register**
Register Address **M1[08h]**

Bit #	7	6	5	4	3	2	1	0
Name	—	—	PI4.5	PI4.4	PI4.3	PI4.2	PI4.1	PI4.0
Reset	0	0	s	s	s	s	s	s
Access	r	r	r	r	r	r	r	r

Bits 5:0: Port 4 Input Bits. The read values of these bits reflect the logic states present at port 4 pins P4.0 to P4.5.

Register Name **PD0**
Register Description **Port 0 Direction Register**
Register Address **M0[10h]**

Bit #	7	6	5	4	3	2	1	0
Name	PD0.7	PD0.6	PD0.5	PD0.4	PD0.3	PD0.2	PD0.1	PD0.0
Reset	s	s	s	s	s	s	s	s
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0: Input/Output Direction for Port 0. The bits in this register control the input/output direction for port pins P0.0 to P0.7. When PD0.n is set to 0, the corresponding port pin (P0.n) acts as an input with characteristics determined by PO0.n. When PD0.n is set to 1, the port pin acts as an output, driving the output state given by PO0.n.

Register Name **PD1**
Register Description **Port 1 Direction Register**
Register Address **M0[11h]**

Bit #	7	6	5	4	3	2	1	0
Name	PD1.7	PD1.6	PD1.5	PD1.4	PD1.3	PD1.2	PD1.1	PD1.0
Reset	s	s	s	s	s	s	s	s
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0: Input/Output Direction for Port 1. The bits in this register control the input/output direction for port pins P1.0 to P1.7. When PD1.n is set to 0, the corresponding port pin (P1.n) acts as an input with characteristics determined by PO1.n. When PD1.n is set to 1, the port pin acts as an output, driving the output state given by PO1.n.

Register Name **PD2**
Register Description **Port 2 Direction Register**
Register Address **M0[12h]**

Bit #	7	6	5	4	3	2	1	0
Name	PD2.7	PD2.6	PD2.5	PD2.4	PD2.3	PD2.2	PD2.1	PD2.0
Reset	s	s	s	s	s	s	s	s
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0: Input/Output Direction for Port 2. The bits in this register control the input/output direction for port pins P2.0 to P2.7. When PD2.n is set to 0, the corresponding port pin (P2.n) acts as an input with characteristics determined by PO2.n. When PD2.n is set to 1, the port pin acts as an output, driving the output state given by PO2.n.

Register Name **PD3**
Register Description **Port 3 Direction Register**
Register Address **M0[13h]**

Bit #	7	6	5	4	3	2	1	0
Name	PD3.7	PD3.6	PD3.5	PD3.4	PD3.3	PD3.2	PD3.1	PD3.0
Reset	s	s	s	s	s	s	s	s
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0: Input/Output Direction for Port 3. The bits in this register control the input/output direction for port pins P3.0 to P3.7. When PD3.n is set to 0, the corresponding port pin (P3.n) acts as an input with characteristics determined by PO3.n. When PD3.n is set to 1, the port pin acts as an output, driving the output state given by PO3.n.

Register Name	**PD4**
Register Description	**Port 4 Direction Register**
Register Address	**M2[10h]**

Bit #	7	6	5	4	3	2	1	0
Name	—	—	PD4.5	PD4.4	PD4.3	PD4.2	PD4.1	PD4.0
Reset	0	0	s	s	s	s	s	s
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bits 5:0: Input/Output Direction for Port 4. The bits in this register control the input/output direction for port pins P4.0 to P4.5. When PD4.n is set to 0, the corresponding port pin (P4.n) acts as an input with characteristics determined by PO4.n. When PD4.n is set to 1, the port pin acts as an output, driving the output state given by PO4.n.

6.1.1 Port Pin Example 1: Driving Outputs on Port 0

```
move PO0, #000h ; Set all outputs low
move PD0, #0FFh ; Set all P0 pins to output mode
```

6.1.2 Port Pin Example 2: Receiving Inputs on Port 1

```
move PO1, #0FFh    ; Set weak pullups ON on all pins
move PD1, #000h    ; Set all P1 pins to input mode
nop                ; Wait for external source to drive P1 pins

move Acc, PI1      ; Get input values from P1 (will return FF if
                   ; no other source drives the pins low)
```

6.2 External Interrupt Register Descriptions

Register Name	**EIF0**
Register Description	**External Interrupt Flag 0 Register**
Register Address	**M0[06h]**

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

Each bit in this register is set when a negative edge or a positive edge (depending on the ITn bit setting) is detected on the corresponding interrupt pin. Once an external interrupt has been detected, the interrupt flag bit will remain set until cleared by software or a reset. Setting any of these bits causes the corresponding interrupt to trigger if it is enabled to do so.

Bit 7: External Interrupt 7 Edge Detect (IE7)

Bit 6: External Interrupt 6 Edge Detect (IE6)

Bit 5: External Interrupt 5 Edge Detect (IE5)

Bit 4: External Interrupt 4 Edge Detect (IE4)

Bit 3: External Interrupt 3 Edge Detect (IE3)

Bit 2: External Interrupt 2 Edge Detect (IE2)

Bit 1: External Interrupt 1 Edge Detect (IE1)

Bit 0: External Interrupt 0 Edge Detect (IE0)

Register Name **EIF1**
Register Description **External Interrupt Flag 1 Register**
Register Address **M0[07h]**

Bit #	7	6	5	4	3	2	1	0
Name	IE15	IE14	IE13	IE12	IE11	IE10	IE9	IE8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Each bit in this register is set when a negative edge or a positive edge (depending on the ITn bit setting) is detected on the corresponding interrupt pin. Once an external interrupt has been detected, the interrupt flag bit remains set until cleared by software or a reset. Setting any of these bits causes the corresponding interrupt to trigger if it is enabled to do so.

Bit 7: External Interrupt 15 Edge Detect (IE15)

Bit 6: External Interrupt 14 Edge Detect (IE14)

Bit 5: External Interrupt 13 Edge Detect (IE13)

Bit 4: External Interrupt 12 Edge Detect (IE12)

Bit 3: External Interrupt 11 Edge Detect (IE11)

Bit 2: External Interrupt 10 Edge Detect (IE10)

Bit 1: External Interrupt 9 Edge Detect (IE9)

Bit 0: External Interrupt 8 Edge Detect (IE8)

Register Name **EIE0**
Register Description **External Interrupt Enable 0 Register**
Register Address **M0[08h]**

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

Each bit in this register controls the enable for one external interrupt. If a bit is set to 1, the corresponding interrupt is enabled (if it is not otherwise masked). If a bit is set to 0, its corresponding interrupt is disabled.

Bit 7: External Interrupt 7 Enable (EX7)

Bit 6: External Interrupt 6 Enable (EX6)

Bit 5: External Interrupt 5 Enable (EX5)

Bit 4: External Interrupt 4 Enable (EX4)

Bit 3: External Interrupt 3 Enable (EX3)

Bit 2: External Interrupt 2 Enable (EX2)

Bit 1: External Interrupt 1 Enable (EX1)

Bit 0: External Interrupt 0 Enable (EX0)

Register Name **EIE1**
Register Description **External Interrupt Enable 1 Register**
Register Address **M0[09h]**

Bit #	7	6	5	4	3	2	1	0
Name	EX15	EX14	EX13	EX12	EX11	EX10	EX9	EX8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Each bit in this register controls the enable for one external interrupt. If a bit is set to 1, the corresponding interrupt is enabled (if it is not otherwise masked). If a bit is set to 0, its corresponding interrupt is disabled.

Bit 7: External Interrupt 15 Enable (EX15)

Bit 6: External Interrupt 14 Enable (EX14)

Bit 5: External Interrupt 13 Enable (EX13)

Bit 4: External Interrupt 12 Enable (EX12)

Bit 3: External Interrupt 11 Enable (EX11)

Bit 2: External Interrupt 10 Enable (EX10)

Bit 1: External Interrupt 9 Enable (EX9)

Bit 0: External Interrupt 8 Enable (EX8)

Register Name **EIES0**
Register Description **External Interrupt Edge Select 0 Register**
Register Address **M0[0Ch]**

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

Each bit in this register controls the edge select mode for an external interrupt, as follows:

0 = The internal interrupt triggers on a rising (positive) edge.

1 = The external interrupt triggers on a negative (falling) edge.

Bit 7: Edge Select for External Interrupt 7 (IT7)

Bit 6: Edge Select for External Interrupt 6 (IT6)

Bit 5: Edge Select for External Interrupt 5 (IT5)

Bit 4: Edge Select for External Interrupt 4 (IT4)

Bit 3: Edge Select for External Interrupt 3 (IT3)

Bit 2: Edge Select for External Interrupt 2 (IT2)

Bit 1: Edge Select for External Interrupt 1 (IT1)

Bit 0: Edge Select for External Interrupt 0 (IT0)

Register Name **EIES1**
Register Description **External Interrupt Edge Select 1 Register**
Register Address **M0[0Dh]**

Bit #	7	6	5	4	3	2	1	0
Name	IT15	IT14	IT13	IT12	IT11	IT10	IT9	IT8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Each bit in this register controls the edge select mode for an external interrupt, as follows:

0 = The internal interrupt triggers on a rising (positive) edge.

1 = The external interrupt triggers on a negative (falling) edge.

Bit 7: Edge Select for External Interrupt 15 (IT15)

Bit 6: Edge Select for External Interrupt 14 (IT14)

Bit 5: Edge Select for External Interrupt 13 (IT13)

Bit 4: Edge Select for External Interrupt 12 (IT12)

Bit 3: Edge Select for External Interrupt 11 (IT11)

Bit 2: Edge Select for External Interrupt 10 (IT10)

Bit 1: Edge Select for External Interrupt 9 (IT9)

Bit 0: Edge Select for External Interrupt 8 (IT8)

SECTION 7: TIMER/COUNTER TYPE B

This section contains the following information:

7.1 Timer B 7-2
7.1.1 Timer B Mode: Autoreload Mode 7-2
7.1.2 Timer B Mode: Capture Mode 7-3
7.1.3 Timer B Mode: Up/Down Autoreload Mode 7-4
7.1.4 Timer B Mode: Clock Output Mode 7-4
7.1.5 Timer B Mode: PWM Output Function 7-5
7.1.5.1 Timer B Mode: Up-Counting PWM Output Mode 7-6
7.1.5.2 Timer B Mode: Up/Down-Count PWM Output Mode 7-7
7.1.6 Timer B Input Clock 7-8
7.2 Timer/Counter B Peripheral Registers 7-8
7.2.1 Timer B Control Register (TBCN) 7-8
7.2.2 Timer B Value Register (TBV) 7-10
7.2.3 Timer B Capture/Reload Value Register (TBR) 7-10
7.2.4 Timer B Compare Register (TBC) 7-10

LIST OF FIGURES

Figure 7-1. Timer B Autoreload Mode 7-3
Figure 7-2. Timer B Capture Mode 7-3
Figure 7-3. Timer B Up/Down Autoreload Mode 7-4
Figure 7-4. Timer B Clock Output Mode 7-5
Figure 7-5. Timer B PWM Output Waveforms (Up Count, DCEN = 0) 7-6
Figure 7-6. Timer B PWM Output Up/Down-Count Examples 7-7

LIST OF TABLES

Table 7-1. Timer/Counter B Mode Summary 7-2
Table 7-2. Timer B PWM Output Function 7-5
Table 7-3. Timer B Input Clock Prescaler Selection 7-8

SECTION 7: TIMER/COUNTER TYPE B

The timer/counter module allows the MAXQ610 to control a 16-bit programmable timer/counter. The MAXQ610 implements two timer type B modules ("Timer B"): TB0 and TB1.

7.1 Timer B

"Timer B" is an enhanced version of the MAXQ timer type 1 with modifications to support different input clock prescaling and set/reset/compare output functionality. The new timer also counts in the range 0000h to TBR instead of TBR to 0FFFFh.

The Timer B value that increments or decrements (depending on mode of operation) is contained in the 16-bit register, TBV. Timer B is enabled by the Timer B run control (TRB) bit in the TBCN register. To support the basic functionality of Timer B, a 16-bit capture/reload register (TBR) is provided. The basic Timer B operational modes and corresponding TBCN register bit settings are shown in Table 7-1. Following the table, each operational mode is described. The TBA pin can be used as a counter input for any mode except for the Timer B clock output mode since this mode uses TBA for clock output. The Timer B PWM output functionality is described in the sections that follow the basic modes.

7.1.1 Timer B Mode: Autoreload Mode

The Timer B autoreload mode is configured by setting the CP/RLB (TBCN.0) bit to 0. In this mode, Timer B performs a simple timer or counter function, but adds a separate 16-bit reload value and the ability to trigger a reload with an external pin.

When initially enabled, Timer B starts counting from the TBV value. On overflow, TBV is reset and counting continues from 0000h. When Timer B reaches an overflow state, i.e., the TBR value is reached, the TFB flag is set in the following system clock cycle. This flag can generate an interrupt if enabled. In addition, the timer restores its starting 0000h value and begins timing (or counting) again. The overflow value is preloaded by software into the capture/reload register, TBR. This register cannot be used for capture functionality while also performing autoreload, so these modes are mutually exclusive.

When in autoreload mode, Timer B can also be forced to reload with the TBB pin. If EXENB (TBCN.3) is set to 1, then a 1-to-0-transition on TBB causes a reload and the EXFB (TBCN.6) flag to be set. Note that the EXFB flag can be set independent of the state of the TRB bit (e.g., EXFB can still be set on detection of a negative edge when TRB = 0). Otherwise, the TBB pin is ignored.

If the C/TB bit (TBCN.15) is 0, the timer's input clock is a function of the system clock. When C/TB = 1, pulses on the TBA pin are counted. Counting or timing is enabled or disabled using the Timer B run control bit = TRB (TBCN.2). This mode, including the optional reload control, is illustrated in Figure 7-1.

Table 7-1. Timer/Counter B Mode Summary

TIMER B OPERATIONAL MODE	TBCN REGISTER BIT SETTINGS*				
	TBOE	DCEN	EXENB	C/TB	CP/RLB
Autoreload	0	0	0	x	0
Autoreload using TBB pin	0	0	1	x	0
Capture using TBB pin	0	0	1	x	1
Up/down count using TBB pin	0	1	0	x	0
Clock output on TBA pin	1	x	x	0	0

**For modes where the C/TB bit is x: When C/TB = 0, the timer input clock is a prescaled version of the system clock. When C/TB = 1, counter mode is enabled and the external TBA pin is counted.*

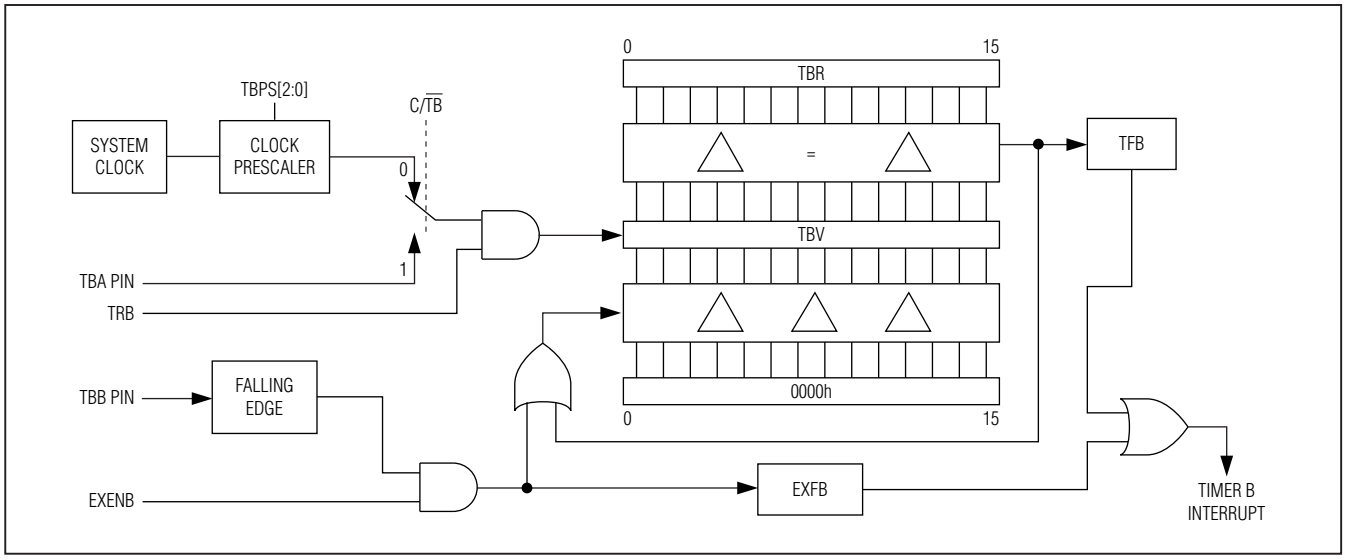


Figure 7-1. Timer B Autoreload Mode

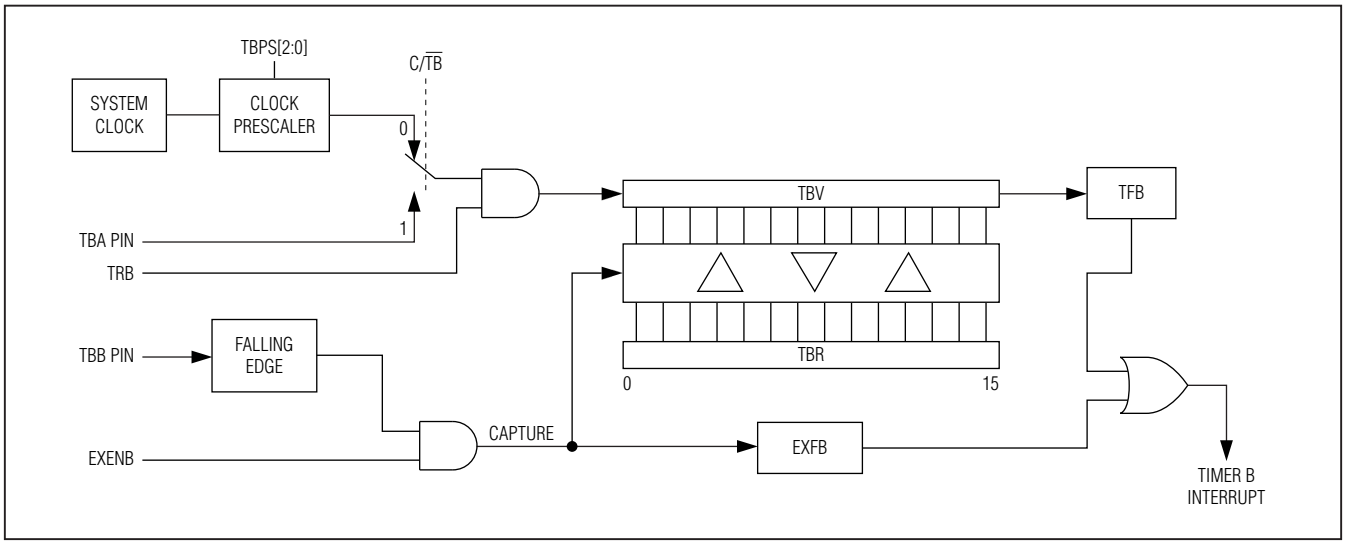


Figure 7-2. Timer B Capture Mode

7.1.2 Timer B Mode: Capture Mode

The 16-bit capture mode is invoked by setting the CP/RLB (TBCN.0) bit to 1. Timer B, when initially enabled, begins counting from the TBV value and upon overflow, subsequently continues counting from 0000h to the 0FFFFh overflow, i.e., rolls over from 0FFFFh to 0000h if left enabled and running. When an overflow occurs, it sets the TFB Flag. This flag can generate an interrupt if enabled. The optional capture function is enabled by setting the EXENB (TBCN.3) bit to 1. Once this has been done, a 1-to-0-transition on the TBB pin causes the value in Timer B (TBV) to be transferred into the capture register (TBR) and the EXFB (TBCN.6) flag to be set. Note that the EXFB flag can be set independent of the state of the TRB bit (e.g., EXFB can still be set on detection of a negative edge when TRB = 0). Setting of the EXFB flag can generate an interrupt if enabled. If the EXENB bit is set to 0, then 1-to-0-transitions on the TBB pin do not automatically trigger a capture event.

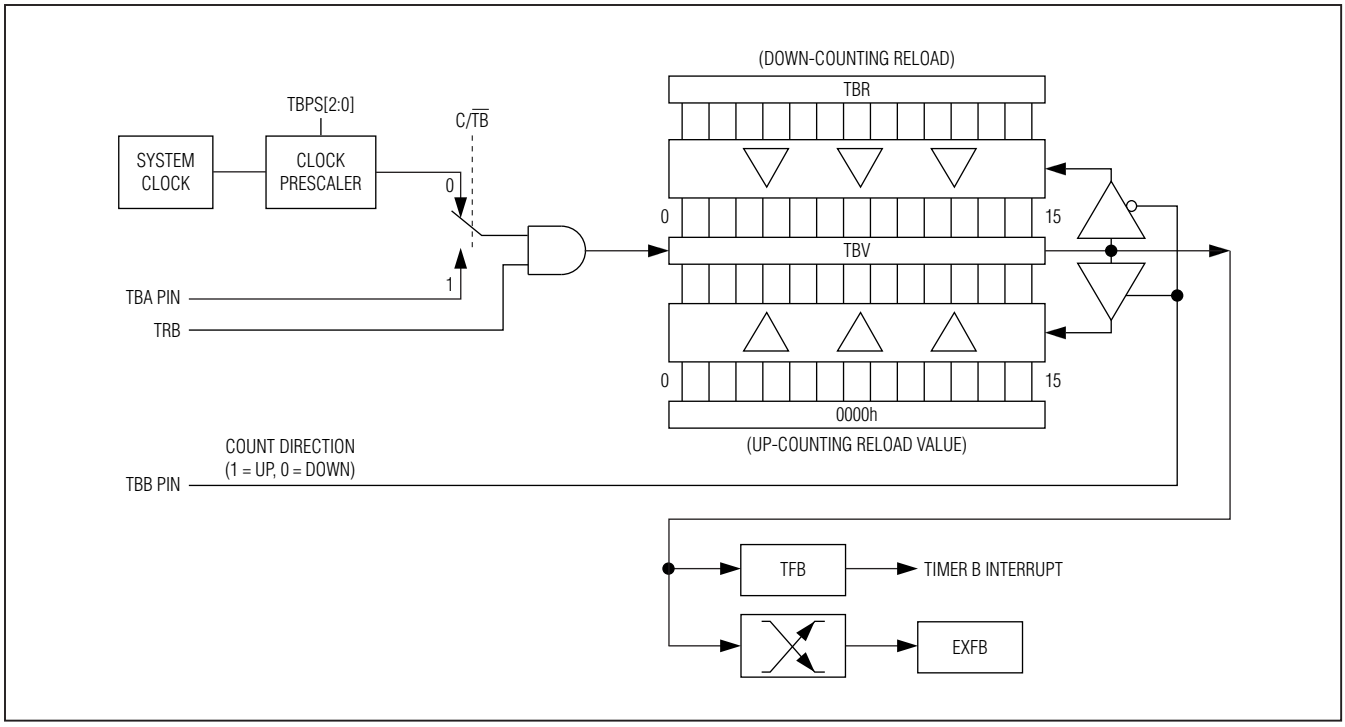


Figure 7-3. Timer B Up/Down Autoreload Mode

7.1.3 Timer B Mode: Up/Down Autoreload Mode

The up/down-count autoreload option is enabled by the DCEN (TBCN.4) bit. When DCEN is set to 1, Timer B counts up or down as controlled by the state of TBB pin. TBB causes upward counting when a high is applied and down counting when a low is applied. When DCEN = 0, Timer B only counts up.

When an upward counting overflow occurs (TBV overflow occurs after reaching TBR), a 0000h value loads into TBV. In the down-count direction, an underflow occurs when TBV reaches 0000h. When an underflow occurs, the TBR value is loaded into TBV 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 TFB bit (TBCN.7). This edge-detection circuit toggles the EXFB bit (TBCN.6) on every overflow or underflow. Therefore, the EXFB bit behaves as a 17th bit of the counter, and can be used as such.

7.1.4 Timer B Mode: Clock Output Mode

Timer B can also be configured to drive a clock output on the TBA port pin as shown in Figure 7-4. To configure Timer B for this mode, first it must be set to 16-bit autoreload timer mode (CP/RLB = 0, C/TB = 0). Next, the TBOE (TBCN.5) bit must be set to 1. The output state for this mode is always set to 1 each time the TBOE bit is changed from 0 to 1. TRB (TBCN.2) must also be set to 1 to enable the timer and the corresponding output. If the timer is stopped (TRB = 0) and subsequently restarted (TRB = 1) while leaving TBOE = 1, the previous timer clock output state is restored on the TBA pin. 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 the figure. Each timer overflow causes an edge transition on the pin, i.e., the state of the pin toggles. The timer overflow flag (TFB) is still set on an overflow in clock output mode, however, the TBOE = 1 condition prevents this flag from causing an interrupt. The Timer B external interrupt is still available for use when enabled (EXENB = 1). Note that the EXFB flag can be set independent of the state of the TRB bit (e.g., EXFB can still be set on detection of a negative edge when TRB = 0).

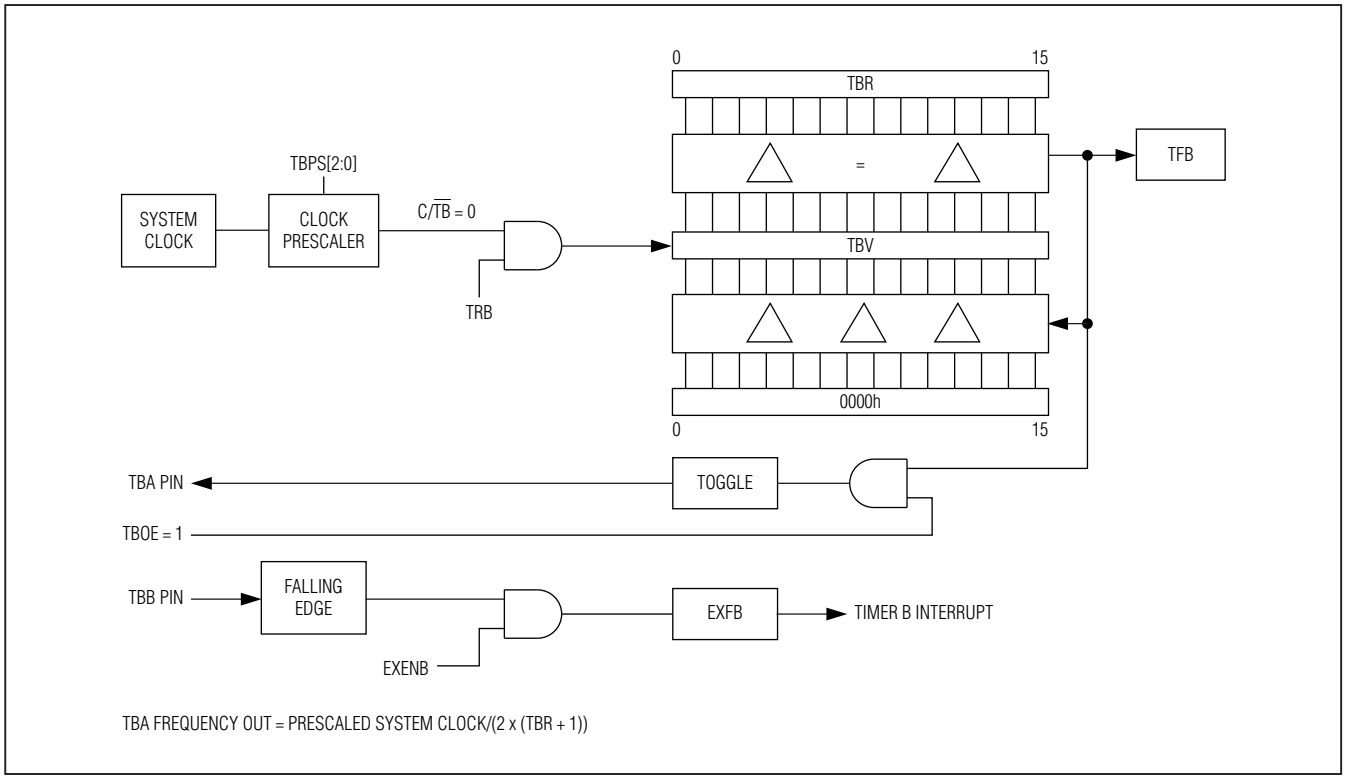


Figure 7-4. Timer B Clock Output Mode

Table 7-2. Timer B PWM Output Function

TBCS:TBCR	FUNCTION	INITIAL STATE (TRB = 0)
00	None (Compare Disable)	No change
01 (Reset)	Reset on TBC Match, Set on 0000h	Low
10 (Set)	Set on TBC Match, Reset on TBR Match	High
11 (Toggle)	Toggle on TBC Match (except TBR or 0h)	No change*

**The initial state for the toggle mode depends on the previous set or reset mode. This means that the TBCS:TBCR bits should be configured to set or reset momentarily when changing from "Compare Disable" to "Compare Toggle Mode" to establish a specific starting state.*

7.1.5 Timer B Mode: PWM Output Function

The PWM output function is enabled whenever the TBCS:TBCR bit pair is nonzero. Table 7-2 shows how these bits define a certain output function.

When the PWM output function is configured to the reset mode, configuring TBC = 0000h disables the TBC compare match reset operation. The timer will do one set on 0000h and never reset. When the PWM output function is configured to the set mode, configuring TBC = TBR disables the TBC compare match set operation. The timer will do one reset on TBR match and never set. When the PWM output function is configured to toggle, configuring TBC = 0000h or TBR disables the toggle function.

When the timer is not running (TRB = 0), the initial output starting state of the TBB output is established as low or high, respectively, if the reset function (TBCR = 1,TBCS = 0) or set function (TBCR = 0, TBCS = 1) is established. Invoking the toggle function does not change the already defined starting state for TBB, thus a fixed high or low starting state may be defined for the toggle mode by first passing through the set or reset mode. The initial starting state takes effect on the pin when the timer is started (TRB = 1). Changing the output function to set or reset while the timer is running does not affect the current output.

7.1.5.1 Timer B Mode: Up-Counting PWM Output Mode

The 16-bit timer/counter with autoreload mode is used for the up-counting PWM output mode to produce edge-aligned PWM output. In the 16-bit autoreload timer mode, the Timer B allows an optional external pin (TBB) triggered reload event when the EXENB bit is configured to 1. The external input special function and the PWM output function can be enabled at the same time, however the input special function changes slightly when the PWM output is enabled. When the PWM output mode is enabled (TBCS:TBCR ≠ 00b) and the external pin input is enabled (EXENB = 1), the detection of a output falling edge on TBB should still result in setting of the EXFB interrupt flag, but should **not** force an autoreload. Note that the EXFB flag can be set independent of the state of the TRB bit (e.g., EXFB can still be set on detection of a negative edge when TRB = 0). While it is most likely that TRB = 1 when EXFB is set, since TRB = 1 is required to enable the PWM output, a negative edge on the TBB pin while TRB = 0 can still result in setting of EXFB. Using the standard GPI/O port controls to generate a negative edge when the PWM is not running, for instance, can set EXFB. Example TBB output waveforms for the autoreload up-counting mode are shown in Figure 7-5.

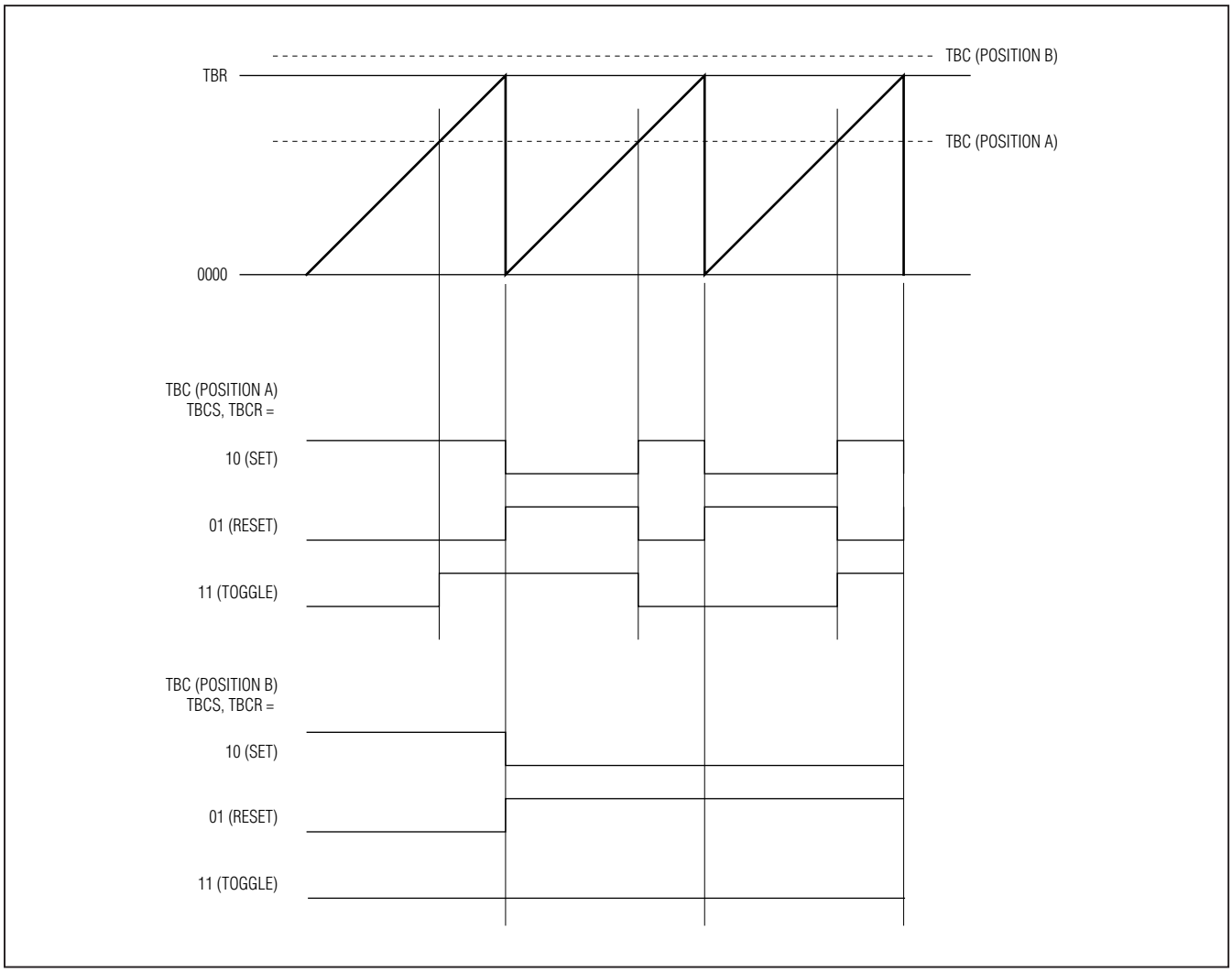


Figure 7-5. Timer B PWM Output Waveforms (Up Count, DCEN = 0)

The set and reset functions for the autoreload up-counting mode essentially provide the same functionality. They provide a 16-bit PWM with the ability to change the frequency using the TBR reload value. The toggle mode allows a 50% duty cycle waveform to be created (when the TBC register is configured to a value inside the counting range, i.e., 0 < TBC < TBR, with Timer B running). With the TBC register outside of the count range, the set and reset functions allow a timed clear or set of a given pin without need of polling or interrupting the CPU such that it can manually be performed.

Up-count set, reset PWM duty cycle can be calculated as follows (where period = TBR + 1):

Set mode = (TBR - TBC)/(TBR + 1)

Reset mode = TBC/(TBR + 1)

7.1.5.2 Timer B Mode: Up/Down-Count PWM Output Mode

The 16-bit up/down-count timer is utilized for the up/down-count PWM output mode to produce center-aligned PWM output. When the Timer B is configured in the up/down-count mode, the external pin (TBB) is used to control the direction of the timer count. When the Timer B PWM output functionality is enabled at the same time as the up/down-count autoreload mode, the TBB pin no longer controls the direction of counting. Instead, the up/count count is controlled internally. When the timer is up counting upward and reaches TBR, in the next cycle, it reverses its direction of counting. When the timer is down counting and reaches 0000h, it reverses direction so as to begin up counting (as illustrated in Figure 7-6). When the up/down autoreload and PWM output modes are both enabled, the TBB input function can still be enabled by the EXENB = 1 configuration. Enabling the TBB input function during this mode allows detection of PWM output negative edges to set the EXFB interrupt flag. Note that the EXFB flag can be set independent of the state of the TRB bit (e.g., EXFB can still be set on detection of a negative edge when TRB = 0). While it is most likely that TRB = 1 when EXFB is set, since TRB = 1 is required to enable the PWM output, a negative edge on the TBB pin while TRB = 0 can still result in setting of EXFB. Using the standard GPI/O port controls to generate a negative edge when the PWM is not running, for instance, can set EXFB.

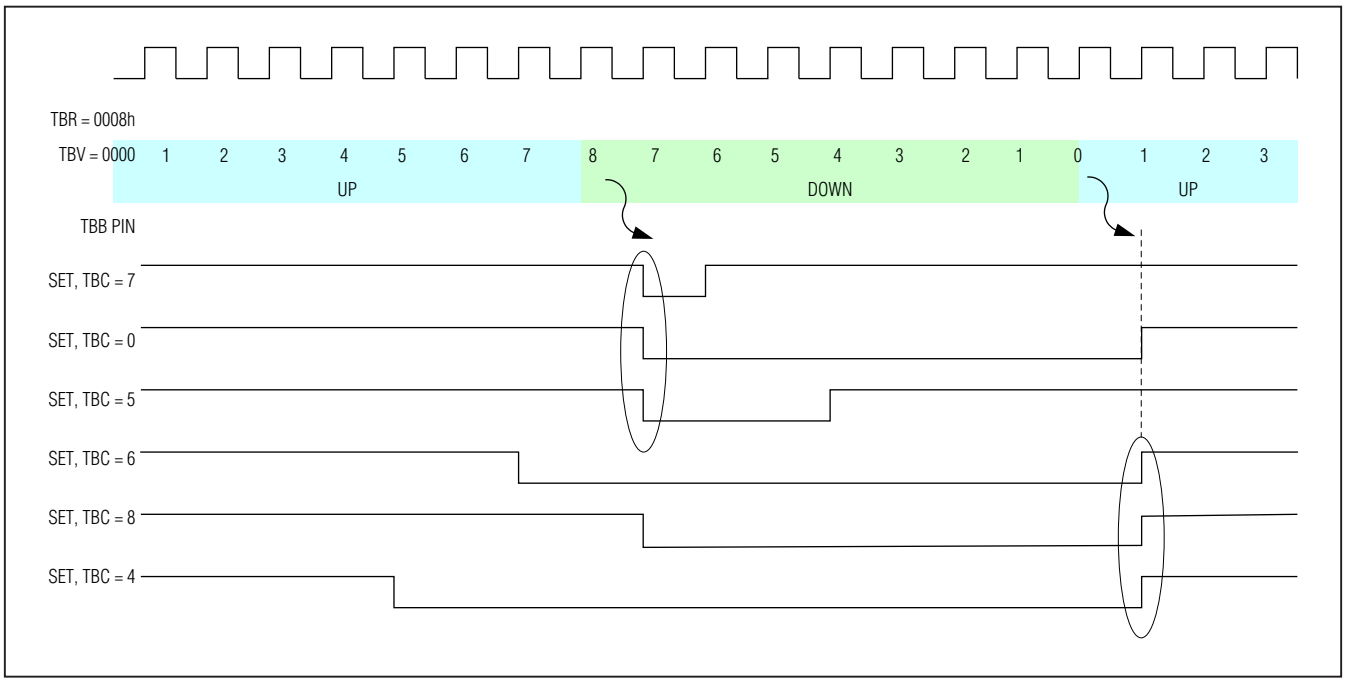


Figure 7-6. Timer B PWM Output Up/Down-Count Examples

Example TBB output waveforms for the autoreload up/down-counting modes are shown below.

Up/down-count PWM duty cycle can be calculated as follows (where period = 2 x TBR):

Set mode = (TBR + TBC)/(2 x TBR)

Reset mode = TBC/(2 x TBR)

Toggle mode = TBC/TBR or (TBR - TBC)/TBR

Note that the toggle mode has two possible duty-cycle calculations and depends upon the initial pin state and starting TBV and TBC values. For example, the TBC/TBR equation would be used if the starting pin state were 1, TBV = 0, and 0 < TBC < TBR. If the starting pin state were 0, and all other initial conditions were the same, the (TBR - TBC)/TBR equation would apply.

The set and reset up/down-count PWM modes effectively allow 17-bit resolution since the set mode allows duty-cycle variation > 50% with 50% of the period always being high and the reset mode allows duty-cycle variation < 50% with 50% of the period always being low.

The toggle mode still effectively provides 16-bit PWM resolution with twice the period of the pure up-counting autoreload mode.

7.1.6 Timer B Input Clock

The Timer B Input Clock can be prescaled using the TBPS[2:0] bits of the TBCN register. The Timer B input clock is a divided version of the system clock as per the equation below (which also appears in the register bit descriptions).

$$\text{Timer B Clock} = \text{System Clock}/2^{(2\times TBPS[2:0])}$$

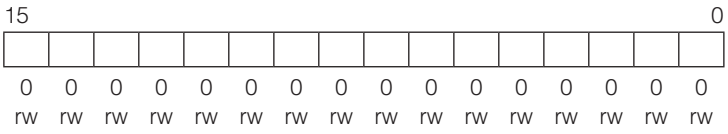
Table 7-3. Timer B Input Clock Prescaler Selection

TBPS[2:0]	TIMER B INPUT CLOCK
000	Sysclk/1
001	Sysclk/4
010	Sysclk/16
011	Sysclk/64
100	Sysclk/256
101	Sysclk/1024
11x	Sysclk/1

The TBPS[2:0] bits should be configured by the user when the timer is stopped (TRB = 0). While hardware does not prevent changing the TBPS[2:0] bits when the timer is running, the resultant behavior is indeterministic.

7.2 Timer/Counter B Peripheral Registers

7.2.1 Timer B Control Register (TBCN)



Timer B Control Register (TBCN)

Power-On Reset and System Resets

Read (r), Write (w), or Special (s) access

Bit 15: (TBCN.15) Counter/Timer Select (C/TB). This bit determines whether Timer B functions as a timer or counter. Setting this bit to 1 causes Timer B to count negative transitions on the TBA pin. If this bit is cleared to 0 Timer B functions as a Timer. The speed of Timer B is determined by the TBPS[2:0] bits of TBCN.

Bits 14 and 13: Reserved. Reads return 0.

Bits 12 and 11: TBB Pin Output Reset Mode, Set Mode (TBCS:TBCR). These mode bits define whether the PWM Mode output function is enabled on the TBB pin, the initial output starting state, and what compare mode output function is in effect. Note that the TBB pin still has certain input functionality when the PWM/output function is enabled. See the *7.1.5: Timer B Mode: PWM Output Function* section for details on this mode.

Bits 10 to 8: Timer B Clock Prescaler Bits 2:0 (TBPS[2:0]. The TBPS[2:0] bits select the clock prescaler applied to the system clock input to Timer B. The TBPS[2:0] bits should be configured by the user when the timer is stopped (TRB = 0). While hardware does not prevent changing the TBPS[2:0] bits when the timer is running, the resultant behavior is indeterministic. See Timer B input clocks for supported prescaler values.

Bit 7: Timer B Overflow Flag (TFB). This bit is set when Timer B overflows from TBR or the count is equal to 0000h in down-count mode. It must be cleared by software.

Bit 6: External Timer B Trigger Flag (EXFB). When configured as a timer (C/TB = 0), a negative transition on the TBB pin causes this flag to be set if (CP/RLB = EXENB = 1) or (CP/RLB = DCEN = 0 and EXENB = 1) or (CP/RLB = 0 and EXENB = 1 and TBCS:TBCR ≠ 00b). When configured in any of these ways, this flag can be set independent of the state of the TRB bit (e.g., EXFB can still be set on detection of a negative edge when TRB = 0).When CP/RLB = 0 and DCEN = 1 and TBCS:TBCR = 00b, EXFB toggles whenever Timer B underflows or overflows. Overflow/underflow condition is the same as described in TFB bit description. In this mode, EXFB can be used as the 17th timer bit and does not cause an interrupt. This flag must be cleared by software if set by a negative transition. Setting this bit to 1 forces a timer interrupt if enabled.

Bit 5: Timer B Output Enable (TBOE). Setting this bit to 1 enables the clock output function on the TBA pin if C/TB = 0. Timer B rollovers do not cause interrupts. Clearing this bit to 0 allows the TBA pin to function as either a standard port pin or a counter input for Timer B.

Bit 4: Down-Count Enable (DCEN). This bit, in conjunction with the TBB pin, controls the direction that Timer B counts in 16-bit autoreload mode. Clearing this bit to 0 causes Timer B to count up only. Setting this bit to 1 enables the up/down-counting mode (i.e., it causes Timer B to count up if the TBB pin is 1 and to count down if the TBB pin is 0). When Timer B PWM output mode functionality is enabled along with up/down counting (DCEN = 1), the up/down-count control of Timer B is controlled internally based upon the count in relation to the register settings. In the compare modes, the DCEN bit controls whether the timer counts up and resets (DCEN = 0), or counts up and down (DCEN = 1).

Bit 3: Timer B External Enable (EXENB). Setting this bit to 1 enables the capture/reload function on the TBB pin for a negative transition (in up-counting mode). A reload results in TBV being reset to 0000h. Clearing this bit to 0 causes Timer B to ignore all external events on the TBB pin. When operating in autoreload mode (CP/RLB = 0) with the PWM output functionality enabled, enabling the TBB input function (EXENB = 1) allows PWM output negative transitions to set the EXFB flag, however, no reload occurs as a result of the external negative edge detection.

Bit 2: Timer B Run Control (TRB). This bit enables Timer B operation when set to 1. Clearing this bit to 0 halts Timer B operation and preserves the current count in TBV.

Bit 1: Enable Timer B Interrupt (ETB). Setting this bit to 1 enables the interrupt from the Timer B TFB and EXFB flags in TBCN. In Timer B clock output mode (TBOE = 1), the timer overflow flag (TFB) is still set on an overflow, however, the TBOE = 1 condition prevents this flag from causing an interrupt when ETB = 1.

Bit 0: Capture/Reload Select (CP/RLB). This bit determines whether the capture or reload function is used for Timer B. Timer B functions in an autoreload mode following each overflow/underflow. See TFB bit description for overflow/underflow condition. Setting this bit to 1 causes a Timer B capture to occur when a falling edge is detected on TBB if EXENB is 1. Clearing this bit to 0 causes an autoreload to occur when Timer B overflow or a falling edge is detected on TBB if EXENB is 1. It is not intended that the Timer B compare functionality should be used when operating in capture mode.

7.2.2 Timer B Value Register (TBV)

Timer B Value Register (TBV)

Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

The TBV register is a 16-bit register that holds the current Timer B value.

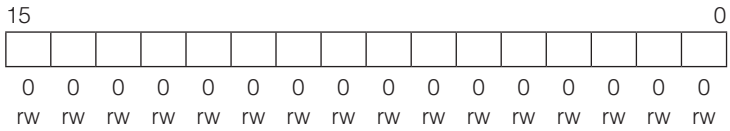
7.2.3 Timer B Capture/Reload Value Register (TBR)

Timer B Capture/Reload Value Register (TBR)

Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

This 16-bit register is used to capture the TBV value when Timer B is configured in capture mode and holds the reload value when Timer B is configured in autoreload mode.

7.2.4 Timer B Compare Register (TBC)



Timer B Compare Register (TBC)

Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

This register is used to compare against TBV when Timer B operates in compare mode.

SECTION 8: IR TIMER

This section contains the following information:

8.1 Carrier Generation Module 8-2
8.2 IR Transmission 8-2
8.3 IR Transmit—Independent External Carrier and Modulator Outputs 8-4
8.4 IR Receive 8-5
8.5 Carrier Burst-Count Mode 8-6
8.6 IRV Stand-Alone Count Mode 8-6
8.7 IR Timer Peripheral Registers 8-10
8.7.1 IR Control Register (IRCN) 8-10
8.7.2 IR Control Register B (IRCNB) 8-11
8.7.3 IR Value Register (IRV) 8-12
8.7.4 IR Carrier Register (IRCA) 8-12
8.7.5 IR Modulator Time Register (IRMT) 8-12

LIST OF FIGURES

Figure 8-1. IR Transmit Frequency Shifting Example (IRCFME = 0) 8-3
Figure 8-2. IR Transmit Carrier Generation and Carrier Modulator Control 8-3
Figure 8-3. IR Transmission Waveform (IRCFME = 0) 8-4
Figure 8-4. External IRTXM (Modulator) Output 8-5
Figure 8-5. IR Capture 8-5
Figure 8-6. Receive Burst Count Example 8-7
Figure 8-7. Philips Remote Encoding Example 8-8
Figure 8-8. Sony Remote Encoding Example 8-9

SECTION 8: IR TIMER

The MAXQ610 microcontroller provides a dedicated IR timer/counter module to simplify support for low-speed infrared (IR) communication. The IR timer implements two pins (IRTX and IRRX) for supporting IR transmit and receive, respectively. The IRTX pin has no corresponding port pin designation, so the standard PD, PO, and PI port control status bits are not present. However, the IRTX pin output can be manipulated high or low using the PWCN.IRTXOUT bit when the IRTX function is not enabled (i.e., IREN = 0 or both IREN = 1 and IRMODE = 0).

The IR timer is composed of two separate timing entities: a carrier generator and a carrier modulator. The carrier generation module uses the 16-bit IR carrier register (IRCA) to define the high and low time of the carrier through the IR carrier high byte (IRCAH) and IR carrier low byte (IRCAL). The carrier modulator uses the IR data bit (IRDATA) and IR modulator time register (IRMT) to determine whether the carrier or the idle condition is present on IRTX.

The IR timer is enabled when the IR enable bit (IREN) is set to 1.

The IR value register (IRV) defines the beginning value for the carrier modulator. During transmission, the IRV register is initially loaded with the IRMT value and begins down counting towards 0000h, whereas in receive mode it counts upward from the initial IRV value. During the receive operation, the IRV register can be configured to reload with 0000h when capture occurs on detection of selected edges or can be allowed to continue running free throughout the receive operation. An overflow occurs when the IR timer value rolls over from 0FFFFh to 0000h. The IR overflow flag (IROV) is set to 1 and an interrupt is generated if enabled (IRIE = 1).

8.1 Carrier Generation Module

The IRCAH byte defines the carrier high time in terms of the number of IR input clock, whereas the IRCAL byte defines the carrier low time.

IR Input Clock (f_{IRCLK}) = $f_{SYS}/2^{IRDIV[1:0]}$

Carrier Frequency ($f_{CARRIER}$) = $f_{IRCLK}/(IRCAH + IRCAL + 2)$

Carrier High Time = IRCAH +1

Carrier Low Time = IRCAL+1

Carrier Duty Cycle = (IRCAH +1)/(IRCAH + IRCAL + 2)

During transmission, the IRCA register is latched for each IRV down-count interval and is sampled along with the IRTXPOL and IRDATA bits at the beginning of each new IRV down-count interval so that duty-cycle variation and frequency shifting is possible from one interval to the next. This is illustrated in Figure 8-1.

Figure 8-2 illustrates the basic carrier generation and its path to the IRTX output pin. The IR transmit polarity bit (IRTXPOL) defines the starting/idle state and the carrier polarity of the IRTX pin when the IR timer is enabled.

8.2 IR Transmission

During IR transmission (IRMODE = 1), the carrier generator is used to create the appropriate carrier waveform, while the necessary modulation is performed by the carrier modulator.

The carrier modulation can be performed as a function of carrier cycles or as a function of IRCLK cycles dependent on the setting of the IRCFME bit. When IRCFME = 0, the IRV down counter is clocked by the carrier frequency and, thus, the modulation is a function of carrier cycles. When IRCFME = 1, the IRV down counter is clocked by IRCLK, allowing carrier modulation timing with IRCLK resolution.

The IRTXPOL bit defines the starting/idle state as well as the carrier polarity for the IRTX pin. If IRTXPOL = 1, the IRTX pin is set to a logic-high when the IR timer module is enabled. If IRTXPOL = 0, the IRTX pin is set to a logic-low when the IR timer is enabled.

A separate register bit, IR data (IRDATA), is used to determine whether the carrier generator output is output to the IRTX pin for the next IRMT carrier cycles. When IRDATA = 1, the carrier waveform (or inversion of this waveform if IRTXPOL = 1) is output on the IRTX pin during the next IRMT cycles. When IRDATA = 0, the idle condition, as defined by IRTXPOL, is output on the IRTX pin during the next IRMT cycles.

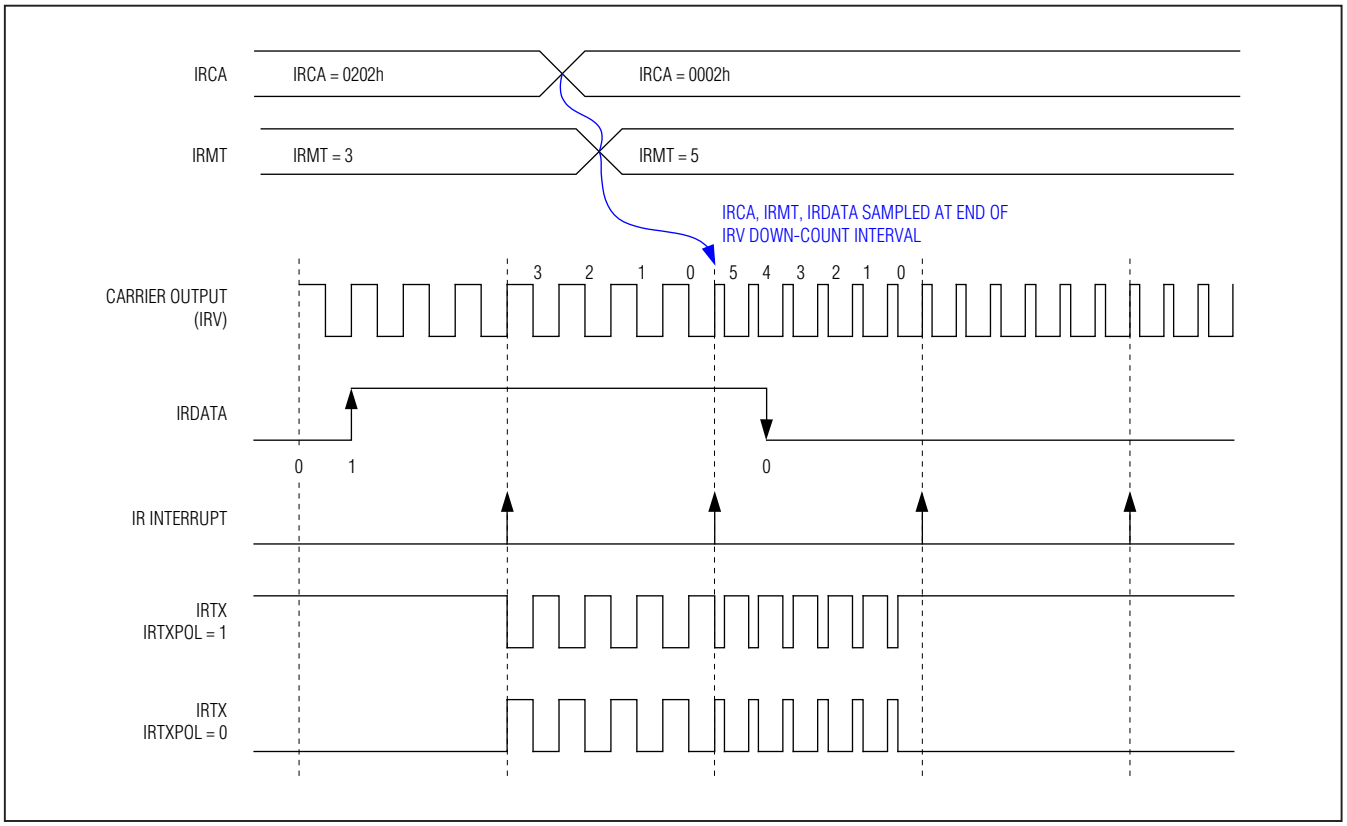


Figure 8-1. IR Transmit Frequency Shifting Example (IRCFME = 0)

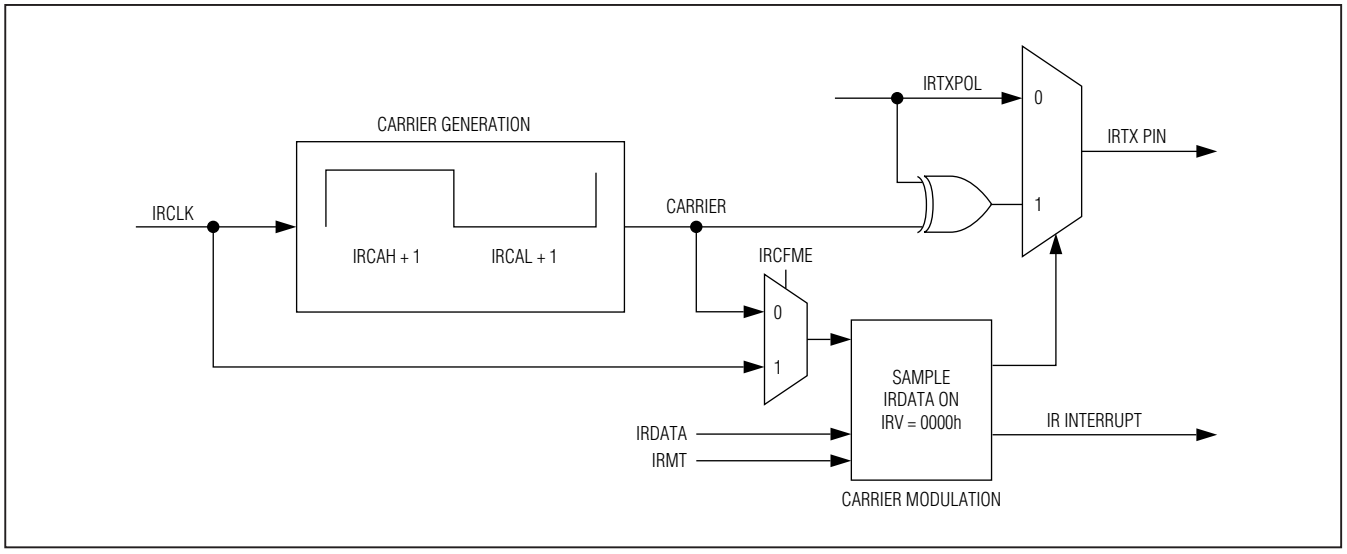


Figure 8-2. IR Transmit Carrier Generation and Carrier Modulator Control

The IR timer acts as a down counter in transmit mode. An IR transmission starts when the IREN bit is set to 1 when IRMODE = 1; when the IRMODE bit is set to 1 when IREN = 1; or when IREN and IRMODE are both set to 1 in the same instruction. The IRMT and IRCA registers, along with the IRDATA and IRTXPOL bits, are sampled at the beginning of the transmit process and every time the IR timer value reloads its value. When the IRV reaches 0000h value, on the next carrier clock, it does the following:

1) Reloads IRV with IRMT.
2) Samples IRCA, IRDATA, and IRTXPOL.
3) Generates IRTX accordingly.
4) Sets IRIF to 1.
5) Generates an interrupt to the CPU if enabled (IRIE = 1).

To terminate the current transmission, the user can switch to receive mode (IRMODE = 0) or clear IREN to 0.

Carrier Modulation Time = IRMT + 1 cycles

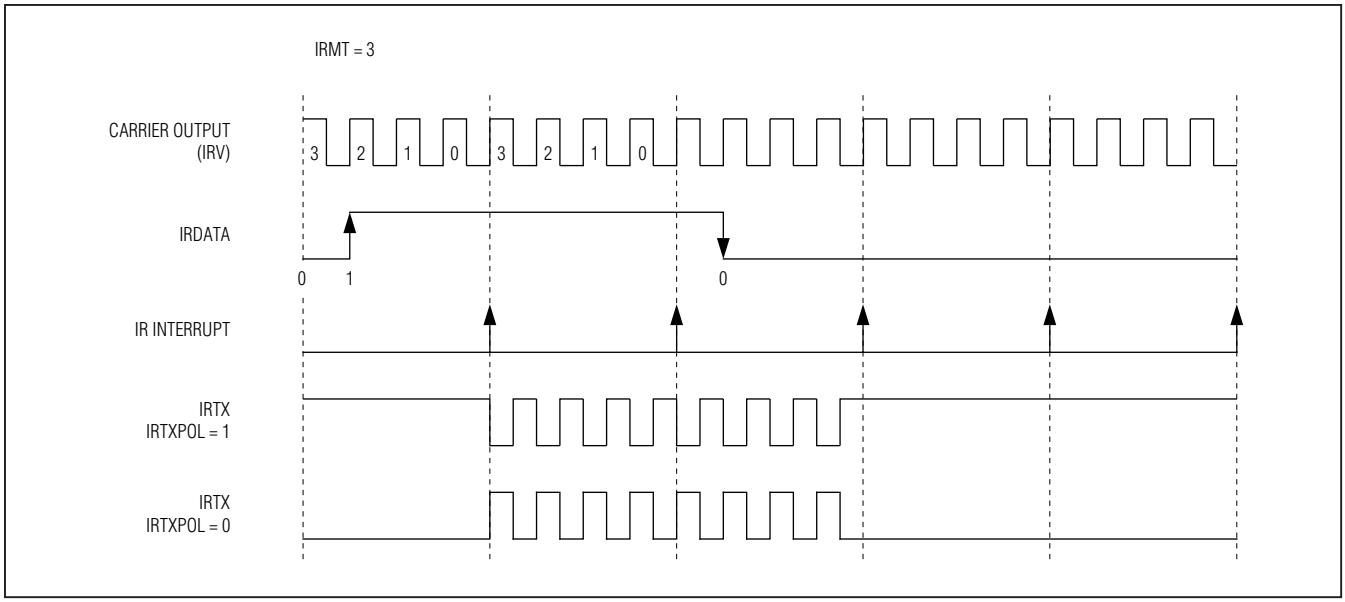


Figure 8-3. IR Transmission Waveform (IRCFME = 0)

8.3 IR Transmit—Independent External Carrier and Modulator Outputs

The normal transmit mode performs internal modulation of the carrier based upon the IRDATA bit.

However, the user has the option to discretely provide the modulator (envelope) on an external pin if desired. If the IRENV[1:0] bits are configured to 01b or 10b, the modulator/envelope is output to the IRTXM pin. The IRDATA bit is output directly to the IRTXM pin (if IRTXPOL = 0) on each IRV down-count interval boundary just as if it were being used to internally modulate the carrier frequency. If IRTXPOL = 1, the inverse of the IRDATA bit is output to the IRTXM pin on the IRV interval down-count boundaries. The envelope output is illustrated in Figure 8-4. When the envelope mode is enabled, it is possible to output either the modulated (IRENV[1:0] = 01b) or unmodulated (INENV[1:0] = 10b) carrier to the IRTX pin.

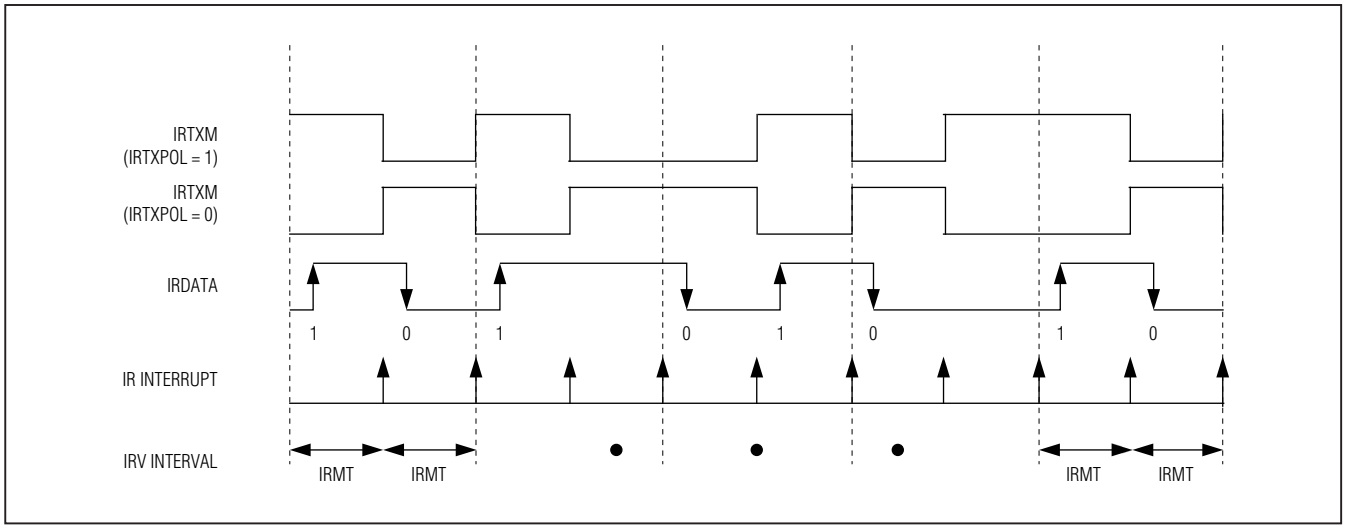


Figure 8-4. External IRTXM (Modulator) Output

8.4 IR Receive

When configured in receive mode (IRMODE = 0), the IR hardware supports the IRRX capture function. The IRRXSEL[1:0] bits define which edge(s) of the IRRX pin should trigger IR timer capture function.

The IR module starts operating in the receive mode when IRMODE = 0 and IREN = 1. Once started, the IR timer (IRV) starts up counting from 0000h when a qualified capture event as defined by IRRXSEL happens. The IRV register will, by default, be counting carrier cycles as defined by the IRCA register. However, the IR clock frequency mux enable (IRCFME) bit can be set to 1 to allow clocking of the IRV register directly with the IRCLK for finer resolution. When IRCFME = 0, the IRCA-defined carrier is counted by IRV. When IRCFME = 1, the IRCLK clocks the IRV register.

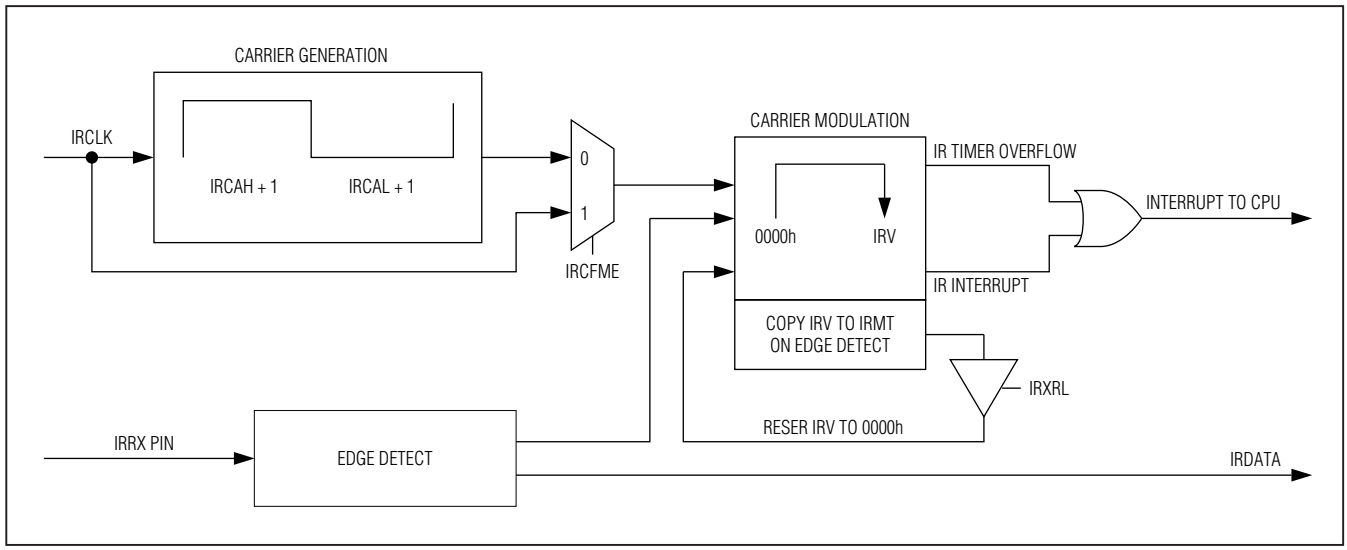


Figure 8-5. IR Capture

On the first qualified event, it does the following:

1) Captures the IRRX pin state and transfers its value to IRDATA. If a falling edge occurs, IRDATA = 0. If a rising edge occurs, IRDATA = 1.
2) Transfers its current IRV value to the IRMT.
3) Resets IRV content to 0000h (if IRXRL = 1).
4) Continues counting again until the next qualified event.

If the IR timer value rolls over from 0FFFFh to 0000h before a qualified event happens, the IR timer overflow (IROV) flag is set to 1 and an interrupt is generated, if enabled. The IR module continues to operate in receive mode until it is stopped by switching into transmit mode (IRMODE = 1) or clearing IREN = 0.

8.5 Carrier Burst-Count Mode

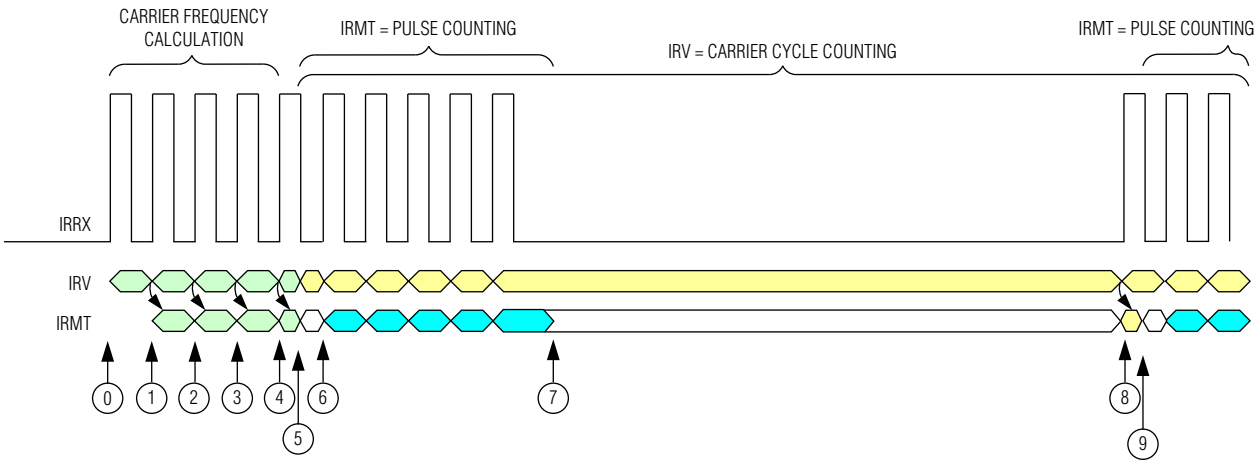
A special mode has been implemented to reduce the CPU processing burden when performing IR learning functions. Typically, when operating in an IR learning capacity, some number of carrier cycles is examined for frequency determination. Once the frequency has been satisfactorily determined, the IR receive function can be reduced to counting the number of carrier pulses in the burst and the duration of the combined mark-space time within the burst. To simplify this process, the receive burst-count mode (as enabled by the RXBCNT bit) can be used. When RXBCNT = 0, the standard IR receive capture functionality is in place. When RXBCNT = 1, the IRV capture operation is disabled and the interrupt flag associated with the capture no longer denotes a capture. In the carrier burst-count mode, the IRMT register is now used only to count qualified edges. The IRIF interrupt flag (normally used to signal a capture when RXBCNT = 0) now becomes set if ever two IRCA cycles elapse without getting a qualified edge. The IRIF interrupt flag thus denotes absence of the carrier and the beginning of a space in the receive signal. When the RXBCNT bit is changed from 0 to 1, the IRMT register is set to 0001h. The IRCFME bit is still used to define whether the IRV register is counting IRCLKs or IRCA-defined carrier cycles. The IRXRL bit is still used to define whether the IRV register is reloaded with 0000h on detection of a qualified edge (per the IRXSEL[1:0] bits).

Figure 8-6 and the descriptive sequence embedded in the figure illustrate the expected usage of the receive burst-count mode.

8.6 IRV Stand-Alone Count Mode

A special mode has been implemented to allow using the IRV as a simple counter. When IRVCEN = 1 and IRMODE = 0, the IRV acts as an up counter counting IRCLK edges (IRCFME = 1) or carrier-generated clock edges (IRCFME = 0). If IREN = 1 and IRXRL = 1, a qualifying edge resets the IRV counter and generates an interrupt (if enabled). Using this feature when IREN = 0 allows controlling the IRTX pin with the PWCN IRTX control bits.

This mode should not be used with RXBCNT set and is not operational if IRMODE = 1.



0 TO 4
CAPTURE INTERRUPT (IRIF = 1)
IRV ≥ IRMT
IRV = 0 (IF IRXRL = 1)

5
SOFTWARE SET IRCA = CARRIER FREQUENCY.
SOFTWARE SETS RXBCNT = 1 (WHICH SETS IRMT = 0001 IN HARDWARE).
SOFTWARE CLEARS IRCFME = 0 SO THAT IRV COUNTS CARRIER CYCLES. IRV IS RESET TO 0 ON QUALIFIED EDGE DETECTION IF IRXRL = 1.
SOFTWARE ADDS TO IRMT THE NUMBER OF PULSES USED FOR CARRIER MEASUREMENT.
IRCA x 2X COUNTER FOR SPACE CAN BEGIN IMMEDIATELY (QUALIFIED EDGE WILL RESET) .

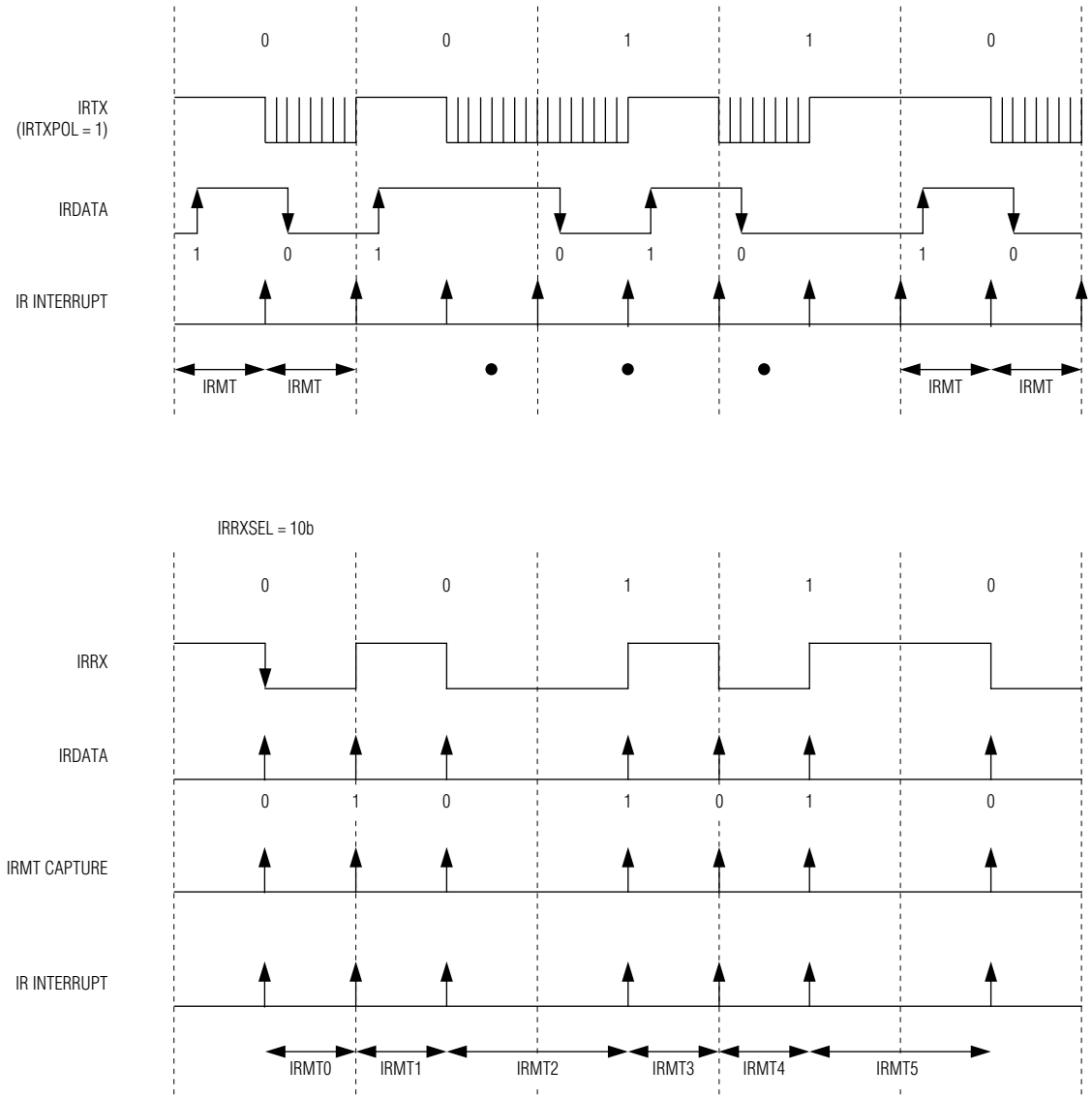
6
QUALIFIED EDGE DETECTED: IRMT++
IRV RESET TO 0 IF IRXRL = 1.

7
IRCA x 2 PERIOD ELAPSES: IRIF = 1; CARRIER ABSENCE = SPACE.
BURST MARK = IRMT PULSES.
SOFTWARE CLEARS RXBCNT = 0 SO THAT WE CAPTURE ON THE NEXT QUALIFIED EDGE.

8
QUALIFIED EDGE DETECTED: IRIF = 1, CAPTURE ≥ IRMT AS THE BURST SPACE (PLUS UP TO ONE CARRIER CYCLE).

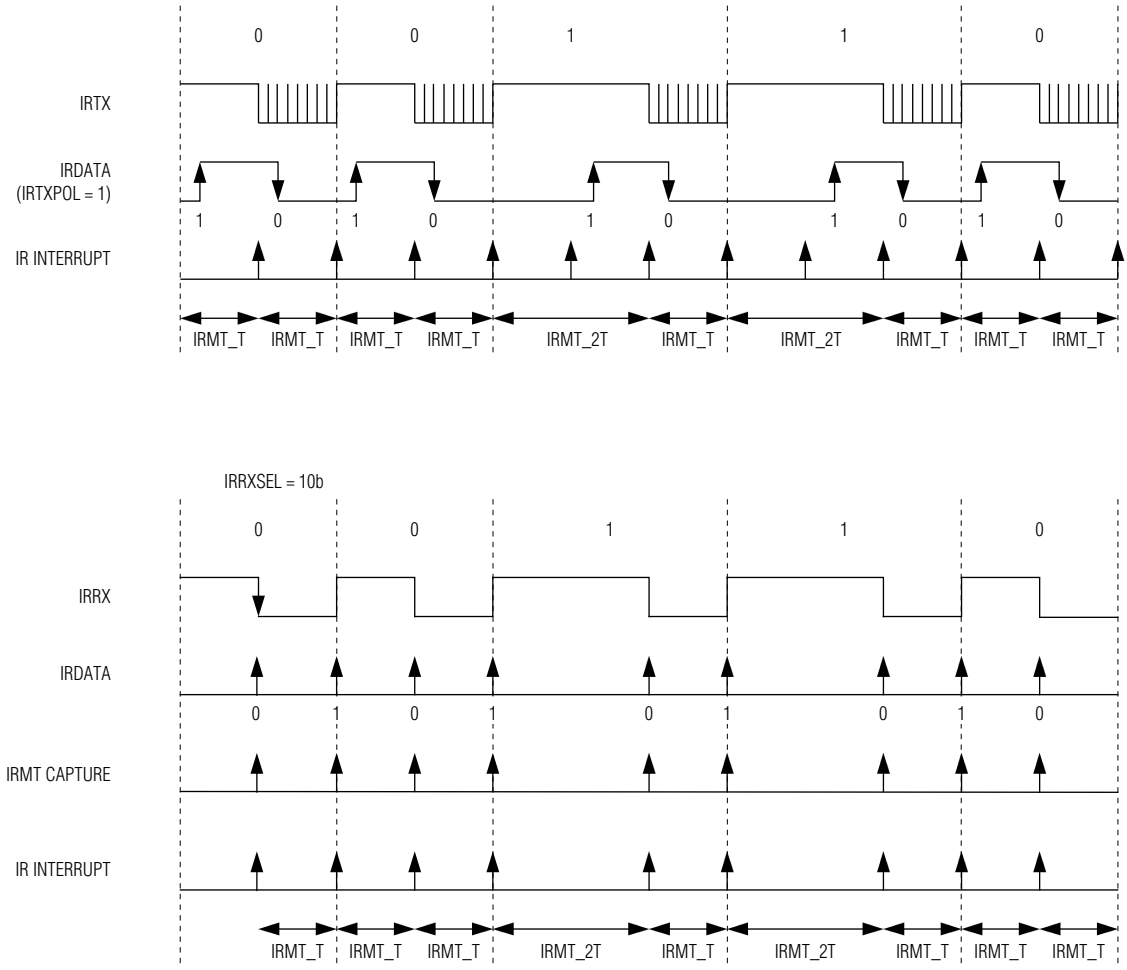
9
SOFTWARE SET RXBCNT = 1 AS IN (5).
CONTINUE (5) TO (8) UNTIL LEARNING SPACE EXCEEDS SOME DURATION. IRV ROLLOVERS CAN BE USED.

Figure 8-6. Receive Burst-Count Example



NOTE: 0 = HIGH-TO-LOW TRANSITION, 1 = LOW-TO-HIGH TRANSITION, BIT LENGTH FIXED.

Figure 8-7. Philips Remote Encoding Example

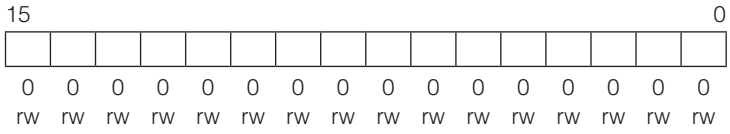


NOTE: 0 = 1T HIGH, 1T LOW, 1 = 2T HIGH, 1T LOW, VARIED BIT LENGTH.

Figure 8-8. Sony Remote Encoding Example

8.7 IR Timer Peripheral Registers

8.7.1 IR Control Register (IRCN)



IR Control Register (IRCN)

Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

Bit 13: IRV Count Enable (IRVCEN). Setting this bit to 1, while IRMODE = 0 (receive mode), enables IRV up counting. IRCFME is used to select the clock source of the IRV in this mode. To use this mode without affecting the IRTX pin, keep IREN = 0.

Bits 12 to 10: IR Clock Divide Bits [1:0] (IRDIV[2:0]). These two bits select the divide ratio for the IR input clock.

IRDIV[2:0]	IR INPUT CLOCK-DIVIDE RATIO
000	$f_{SYSCLK}/1$
001	$f_{SYSCLK}/2$
010	$f_{SYSCLK}/4$
011	$f_{SYSCLK}/8$
100	$f_{SYSCLK}/16$
101	$f_{SYSCLK}/32$
110	$f_{SYSCLK}/64$
111	$f_{SYSCLK}/128$

Bits 9 and 8: IR Envelope Mode Bits [1:0] (IRENV[1:0]). Setting either of these bits (but not both) to 1 enables the envelope modulation signal (based upon the IRDATA and IRTXPOL bits) to be output separately to the IRTXM pin during transmit mode. When these bits are both cleared to 0 or set to 1, the standard internal modulation is performed during IR transmit mode and the envelope signal is not output to the IRTXM pin. When the envelope mode is enabled, it is possible to output either the modulated or unmodulated carrier to the IRTX pin (see the following table).

IRENV[1:0]	IRTX OUTPUT
00 or 11	Envelope mode disabled. Standard IRTX modulation (default).
01	Standard IRTX modulation.
10	Constant IRTX carrier (unmodulated).

Bit 7: IR Receive Reload Enable (IRXRL). Setting this bit to 1 enables automatic reload of the IRV register with 0000h whenever a qualified edge event capture occurs during the IR receive operation. If IRXRL = 0, the IRV register is not reloaded with 0000h, but continues running during the IR receive operation.

Bit 6: IR Carrier Frequency Measure Enable (IRCFME). Setting this bit to 1 enables direct clocking of the IRV register using the defined IRCLK during the IR receive operation. Clearing this bit to 0 results in IRV counting of the IRCA-defined carrier during the receive operation. Using IRCFME = 1 allows system clock resolution when capturing, whereas IRCFME = 0 allows only (Sysclk/2) resolution when IRCA = 0000h.

Bits 5 and 4: IR Receive Edge Select Bits (IRRXSEL[1:0]) These bits define which edge of the input signal triggers a receive capture function when enabled.

IRRXSEL[1:0]	IR RECEIVE MODE
00	Trigger on falling edge.
01	Trigger on rising edge.
10	Trigger on both rising and falling edge.
11	Reserved.

Bit 3: IR Data (IRDATA). This register bit defines how the carrier is modulated in transmit mode, and in receive mode, it contains the state of IRRX when a qualified capture event happens. When IR transmit mode is in effect, setting IRDATA = 1 enables the output of the carrier module (as affected by IRTXPOL) to be visible on the IRTX pin. When IRDATA = 0, the IR module is put in the idle state and IRTXPOL is output onto IRTX. In receive mode, the IRDATA bit contains the latched state of the IRRX pin each time a capture event occurs.

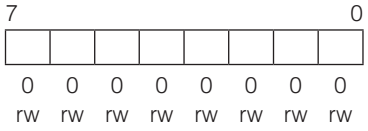
Bit 2: IR TX Polarity Select (IRTXPOL). When the IR timer is enabled (IREN = 1), this bit selects the starting/idle logic state and the carrier polarity for the transmit output. This bit also impacts the polarity of the IRTXM envelope when the independent modulator transmit output mode is enabled (IRENV[1:0] =01b or 10b). When IRENV[1:0] =01b or 10b, the latched IRDATA bit is directly output to the IRTXM pin as the envelope when IRTXPOL = 0. When IRTXPOL = 1, the complement of the latched IRDATA bit is output.

Bit 1: IR Mode (IRMODE). This register bit controls the IR module operation mode.

IRMODE	IR OPERATION MODE
0	Receive Mode
1	Transmit Mode

Bit 0: IR Enable (IREN). This register bit enables the IR module. Setting this bit to 1 starts the operating mode as defined by IRMODE bit. Clearing this bit to 0 terminates IR operation.

8.7.2 IR Control Register B (IRCNB)



IR Control Register B (IRCNB)

Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

Bit 3: Receive Carrier Burst-Count Enable (RXBCNT). Setting this bit to 1 enables the carrier burst-counting mode for the IR timer when operating in receive mode. This bit is not meaningful for the transmit mode. Whenever software changes RXBCNT from 0 to 1, the IRMT register is set to 0001h by hardware. When RXBCNT = 1, the IR timer receive mode is modified in the following ways:

1) The IRV register is not captured to the IRMT register on detection of the IRRXSEL[1:0] selected edge(s).
2) The IRMT register is incremented on detection of the IRRXSEL[1:0] selected edge(s).
3) The IRIF flag is no longer set on capture-edge detection.
4) An IRCA x 2 interval timer is enabled and upon expiration, the IRIF flag is set.

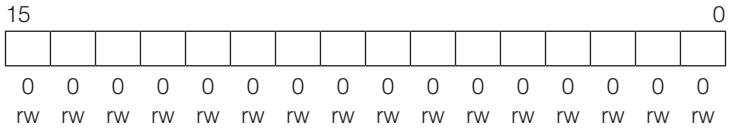
When RXBCNT = 0, the receive carrier burst-count mode is disabled and normal receive capture functionality can be used.

Bit 2: IR Interrupt Enable (IRIE). Setting this bit to 1 enables an interrupt to be generated to the CPU when the IR timer overflow (IROV) or IR interrupt flag is set (IRIF). Clearing this bit to 0 disables IR timer interrupt generation.

Bit 1: IR Interrupt Flag (IRIF). This flag is set to 1 during transmit when the IR timer reloads its value and in receive mode (if RXBCNT = 0), when a capture occurs. In receive mode (when RXBCNT = 1), this flag is set whenever the IRCA x 2 interval timer expires. This bit must be cleared to 0 by software once it is set.

Bit 0: IR Timer Overflow Flag (IROV). This flag is set to 1 when the IR timer overflows from 0FFFFh to 0000h in receive mode. This bit must be cleared to 0 by software once it is set.

8.7.3 IR Value Register (IRV)

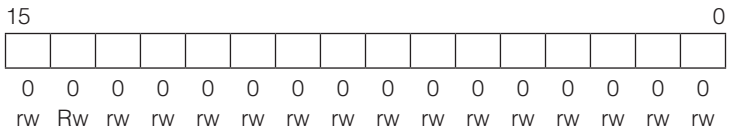


IR Value Register (IRV)

Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

The IRV register is a 16-bit register that holds the current IR timer value. The IR timer value starts counting when the IREN bit is set to 1. It stops counting when the IREN bit is cleared to 0 and retains the current timer value.

8.7.4 IR Carrier Register (IRCA)



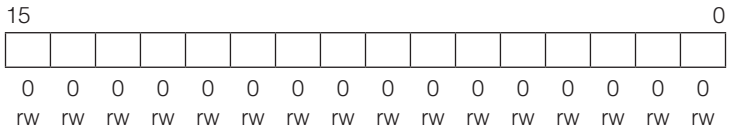
IR Carrier Register (IRCA)

Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

Bits 15 to 8: IR Carrier High Byte Bits [7:0] (IRCAH[7:0]). The IRCAH byte defines the number of IR input clocks during carrier high time. The carrier high time = IRCAH[7:0] + 1.

Bits 7 to 0: IR Carrier Low Byte Bits [7:0] (IRCAL[7:0]). The IRCAL byte defines the number of IR input clocks during carrier low time. The carrier low time = IRCAL[7:0] + 1.

8.7.5 IR Modulator Time Register (IRMT)



IR Modulator Time Register (IRMT)

Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

The IRMT register is a 16-bit register that defines the IRDATA active time during transmit mode. In receive mode (when RXBCNT = 0), it is used to capture the IRV value on qualified IRRXSEL edges. In receive mode (when RXBCNT = 1), the IRMT register increments on detection of selected IRRXSEL edge(s). When RXBCNT is changed from 0 to 1, the IRMT register is set to 0001h by hardware.

SECTION 9: SERIAL I/O MODULE

This section contains the following information:

9.1 USART Modes 9-2
9.1.1 USART Mode 0 9-2
9.1.2 USART Mode 1 9-4
9.1.3 USART Mode 2 9-4
9.1.4 USART Mode 3 9-7
9.2 Baud-Rate Generation 9-8
9.2.1 Mode 0 Baud Rate 9-8
9.2.2 Mode 2 Baud Rate 9-8
9.2.3 Mode 1 or 3 Baud Rate 9-8
9.2.4 Baud-Clock Generator 9-8
9.3 Framing Error Detection 9-9
9.4 USART Peripheral Registers 9-10
9.4.1 Serial Control Register (SCON) 9-10
9.4.2 Serial Port Mode Register (SMD) 9-11
9.4.3 Serial Port Data Buffer Register (SBUF) 9-11
9.4.4 Serial Port Phase Register (PR) 9-11

LIST OF FIGURES

Figure 9-1. USART Mode 0 9-3
Figure 9-2. USART Mode 1 9-5
Figure 9-3. USART Mode 2 9-6
Figure 9-4. USART Mode 3 9-7

LIST OF TABLES

Table 9-1. USART Mode Summary 9-2
Table 9-2. USART Baud-Clock Summary 9-8
Table 9-3. Example Baud-Clock Generator Settings (SMOD = 1) 9-9

SECTION 9: SERIAL I/O MODULE

The serial I/O module provides the MAXQ610 access to a universal synchronous/asynchronous receiver-transmitter (USART) for serial communication with framing error detection.

9.1 USART Modes

The USART 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. The four serial operating modes are shown in Table 9-1, followed by detailed descriptions of each mode.

Table 9-1. USART 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.*

The USART 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 is accessed. When SBUF is used as a destination (write operation), the transmit buffer is accessed. The USART receiver incorporates a holding buffer so that it can receive an incoming word before software has read the previous one.

Note that there is no single register bit that explicitly enables the USART for transmission. This means that the port pin(s) associated with USART transmission (i.e., TXD and RXD for mode 0) is controlled by the PDn and POn port control register bits when the USART is not actively transmitting a character.

9.1.1 USART Mode 0

This mode is used to communicate in synchronous, half-duplex format with devices that accept the MAXQ610 microcontroller as a master. A functional diagram and basic timing of this mode is shown in Figure 9-1. 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 MAXQ610 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 clock input divided by either 12 or 4, as selected by SM2 bit (SCON.5) for the USART.

The USART 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 USART begins to receive data when the REN bit in the SCON register (SCON.4) is set to 1 and the RI bit (SCON.0) is set to 0. This condition tells the USART that there is data to be shifted in. The shift clock (TXD) activates, and the USART 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 automatically is set to 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.

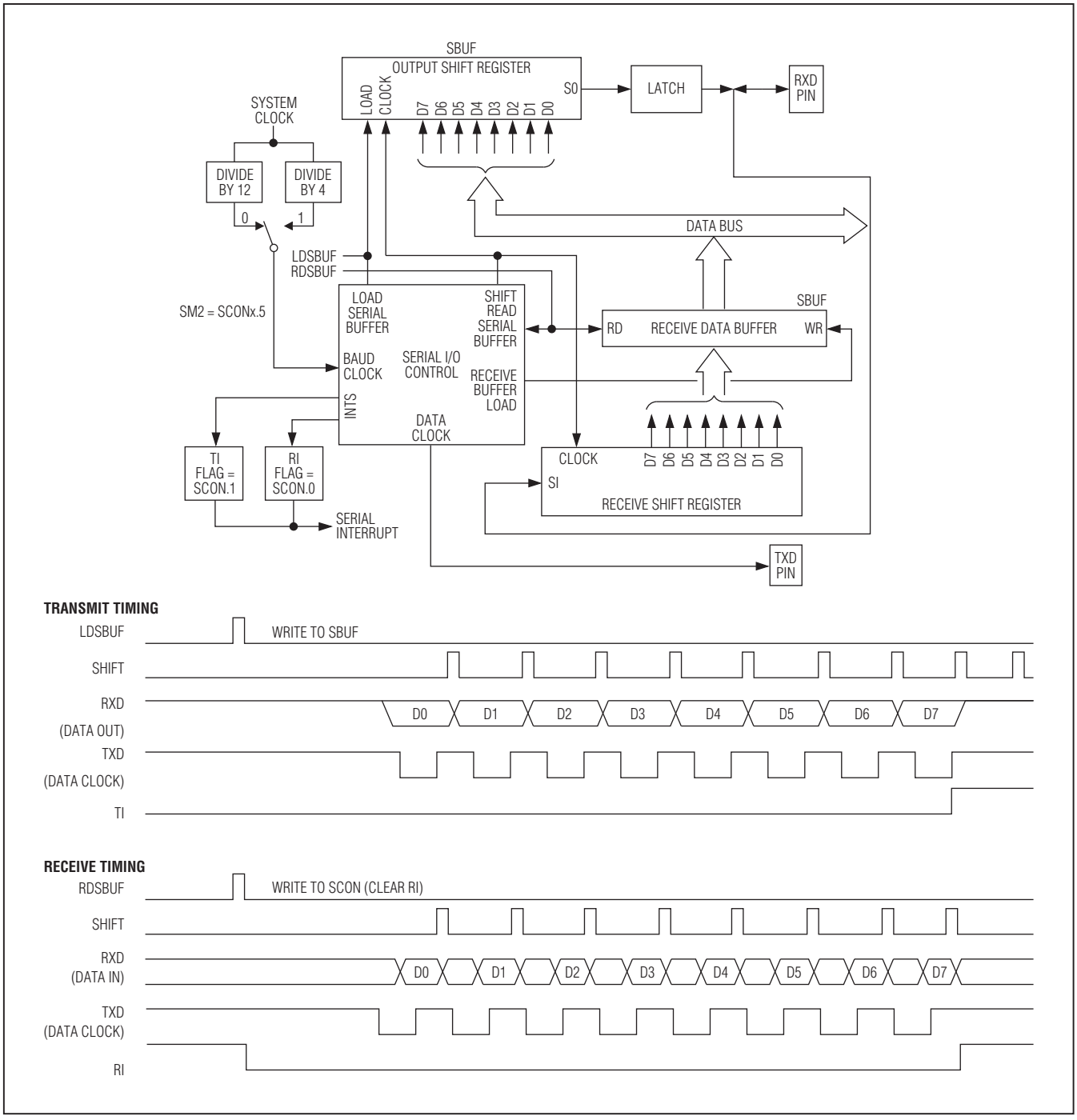


Figure 9-1. USART Mode 0

9.1.2 USART 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 9-2. The data is transferred LSB first. The baud rate is programmable through the baud-clock generator. Following a write to SBUF, the USART 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 1 to allow reception. The detection of a falling edge on the RXD pin is interpreted as the beginning of a start bit and begins 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 be 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 be 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 is lost (SBUF and RB8 not loaded) and RI is not 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, determines 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 in order to start the reception of data.

9.1.3 USART Mode 2

This mode uses a total of 11 bits in asynchronous full-duplex communication as illustrated in Figure 9-3. 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 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 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 be 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 be 1.

If these conditions are true, then SBUF is loaded with the received byte, RB8 is loaded with the 9th bit, and RI is set. If these conditions are false, then the received data is lost (SBUF and RB8 not loaded) and RI is not 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.

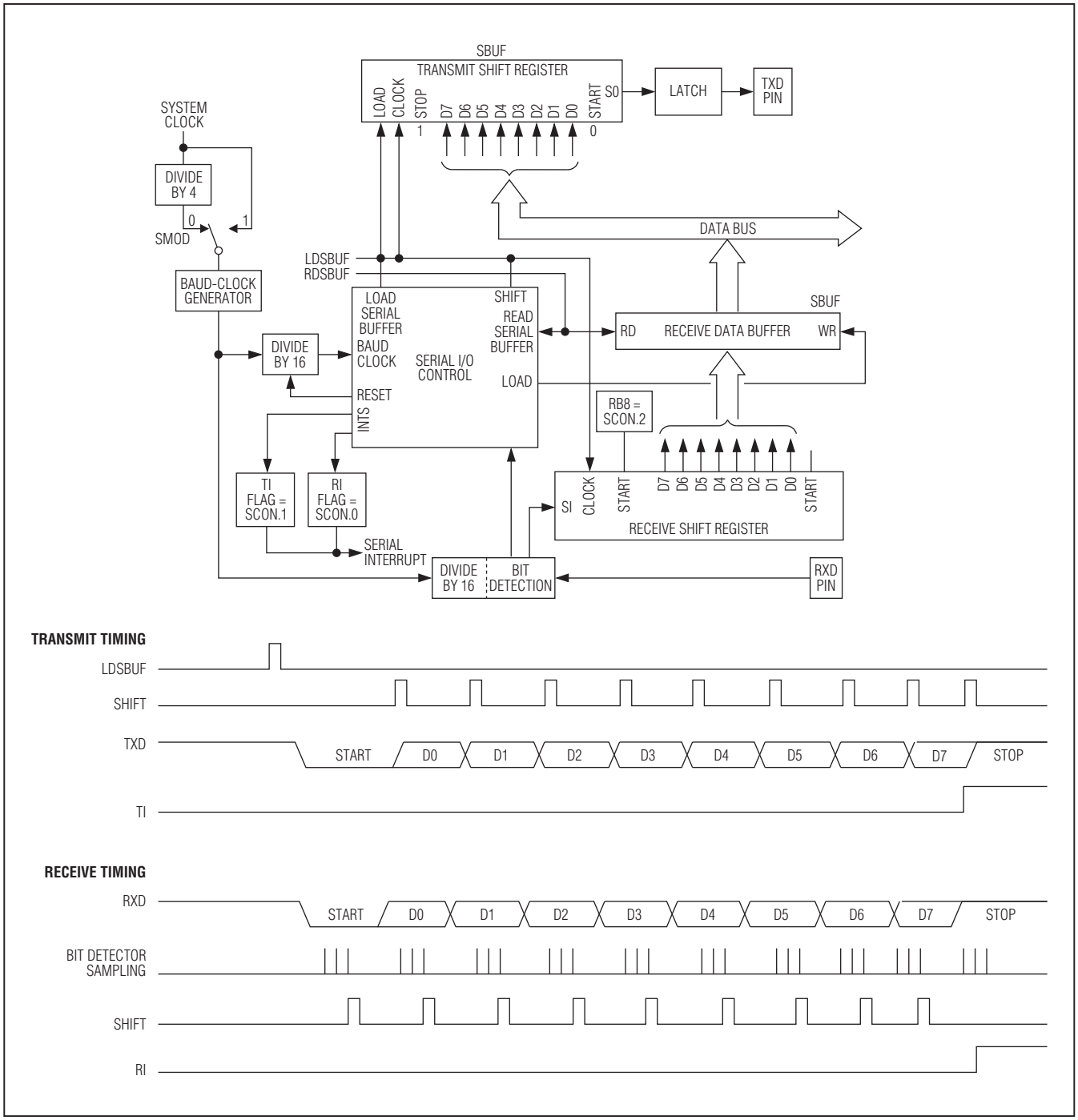


Figure 9-2. USART Mode 1

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.

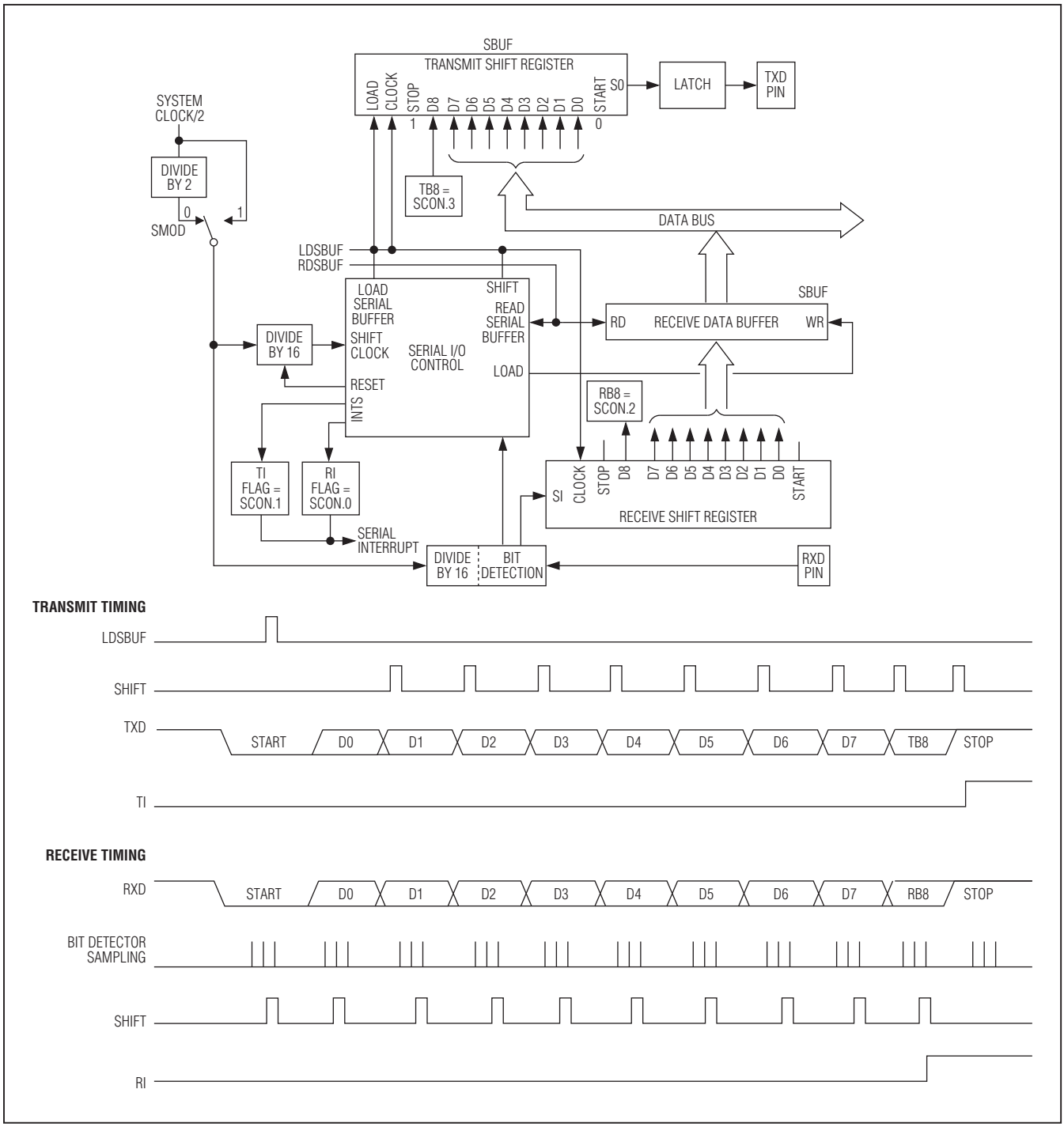


Figure 9-3. USART Mode 2

9.1.4 USART Mode 3

This mode has the same operation as mode 2, except for the baud rate source. As shown in Figure 9-4, mode 3 generates baud rates through the baud-clock generator. The bit shifting and protocol are the same.

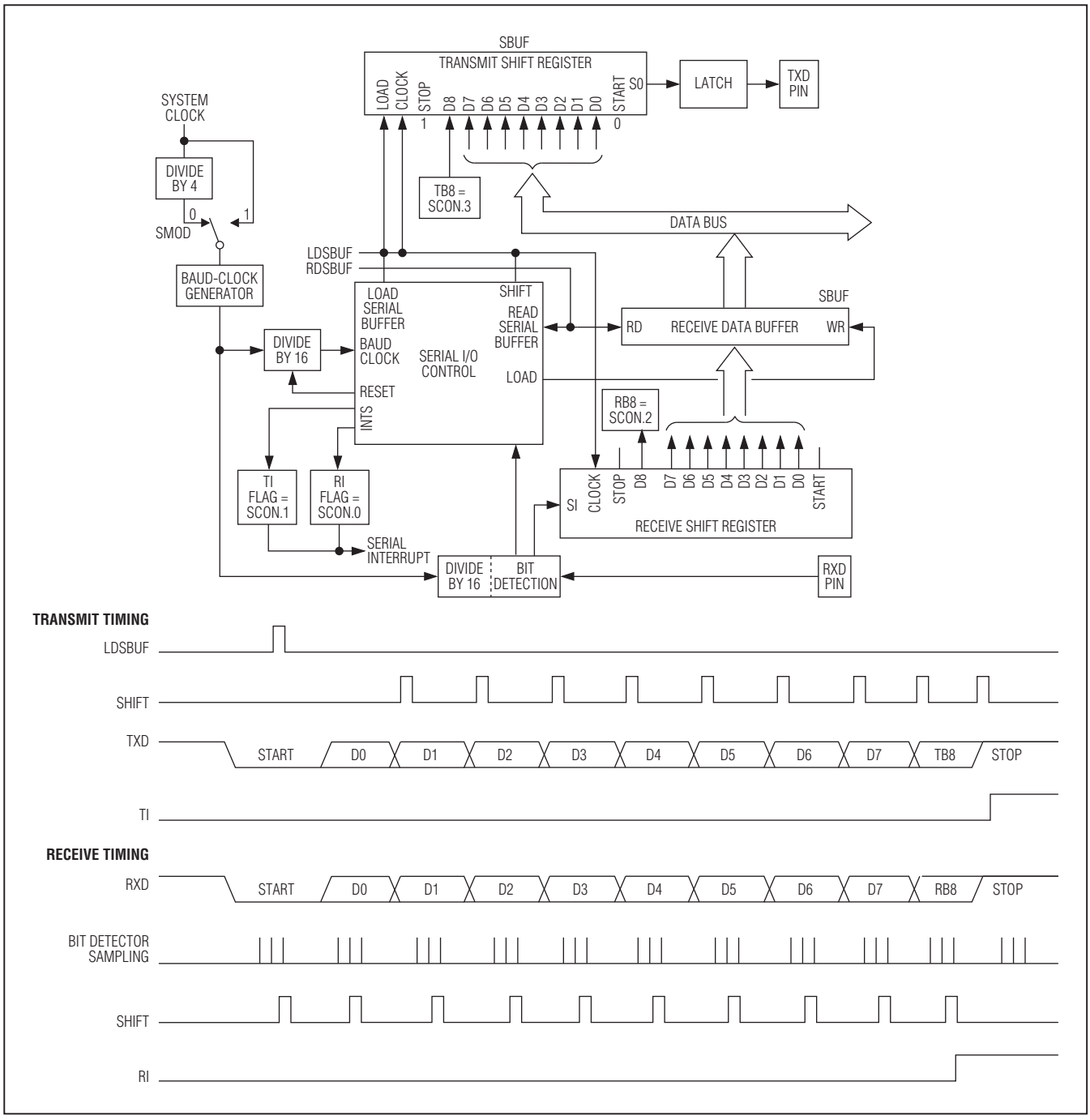


Figure 9-4. USART Mode 3

9.2 Baud-Rate Generation

Each mode of operation has a baud-rate generator associated with it. The baud-rate generation techniques are impacted 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 9-2 summarizes the effects of the various user options on the USART baud clock.

Table 9-2. USART 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 (described later in this section).*

9.2.1 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 divided by 12. The user can select the shift clock frequency using the SM2 bit in the SCON register. When SM2 is set to 0, the baud rate is fixed at a divide by 12 of the system clock. When SM2 is set to 1, the baud rate is fixed at a divide by 4 of the system clock.

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

9.2.2 Mode 2 Baud Rate

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

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

9.2.3 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 USART 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.

9.2.4 Baud-Clock Generator

The baud-clock generator is essentially 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 fourth 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.

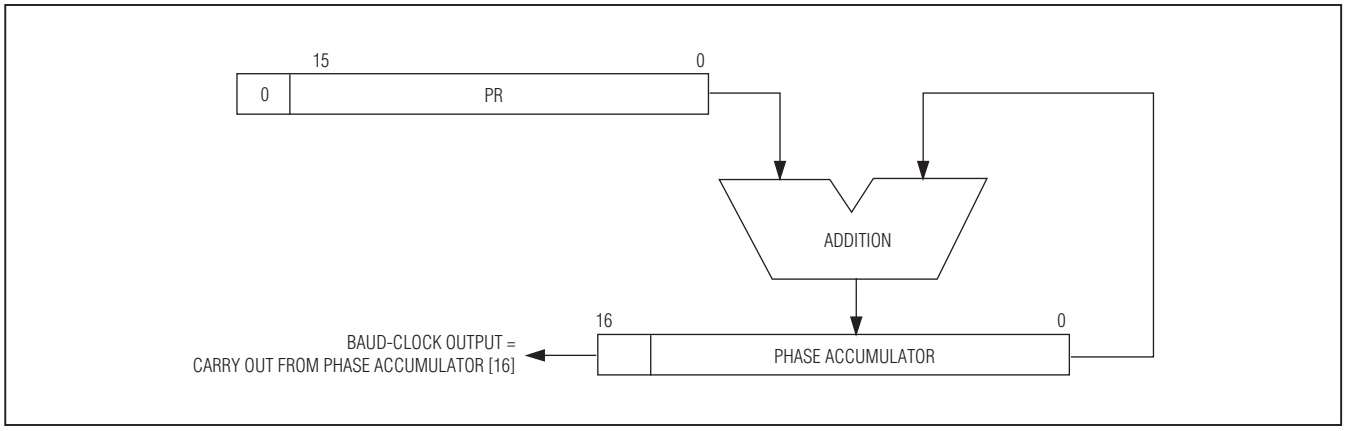


Figure 9-5. Baud-Clock Generator

The below formulas can be used to calculate the output of the baud-clock generator and the resultant mode 1, 3 baud rates. Additionally, a table has been provided giving example phase register (PR) settings needed to produce some more common baud rates at certain system clock frequencies (assuming SMOD = 1).

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

$$\text{Baud Rate for Modes 1 and 3} = BAUD \times 2^{(SMODx2)}/2^{6}$$

Table 9-3. Example Baud-Clock Generator Settings (SMOD = 1)

SYSTEM CLOCK FREQUENCY (MHz)	BAUD RATE (PR SETTING)	SYSTEM CLOCK FREQUENCy (MHz)	BAUD RATE (PR SETTING)
10	115200 (5E5F) 57600 (2F30) 19200 (0FBB) 9600 (07DD) 2400 (01FF)	3.579545	57600 (83D2) 19200 (2BF1) 9600 (15F8) 2400 (057E)
8	115200 (75F7) 57600 (3AFB) 19200 (13A9) 9600 (09D5) 2400 (0275)	2.4576	57600 (C000) 19200 (4000) 9600 (2000) 2400 (0800)
3.6864	115200 (FFFF) 57600 (8000) 19200 (2AAB) 9600 (1555) 2400 (0555)	1	19200 (9D49) 9600 (4EA5) 2400 (13A9)

9.3 Framing Error Detection

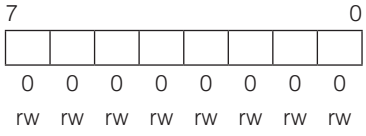
A framing error occurs when a valid stop bit is not detected. This results in the possible improper reception of the serial word. The USART can detect a framing error and notify the software. Typical causes of framing errors are noise and contention. The framing error condition is reported in the SCON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

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 1, the framing error information is shown in SM0/FE (SCON.7). When FEDE is set to 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 a 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.

9.4 USART Peripheral Registers

9.4.1 Serial Control Register (SCON)



Serial Control Register (SCON)
Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

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	SM2	SM1	SM0	FUNCTION	LENGTH (BITS)	PERIOD
0	0	0	0	Synchronous	8	12 system clock
0	1	0	0	Synchronous	8	4 system clock
1	X	1	0	Asynchronous	10	64/16 baud clock (SMOD = 0/1)
2	0	0	1	Asynchronous	11	64/32 system clock (SMOD = 0/1)
2	1	0	1	Asynchronous (MP)	11	64/32 system clock (SMOD = 0/1)
3	0	1	1	Asynchronous	11	64/16 baud clock (SMOD = 0/1)
3	1	1	1	Asynchronous (MP)	11	64/16 baud clock (SMOD = 0/1)

Bits 6:5: Serial Port 0 Mode Bits 2:1 (SM[2:1]). 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.

This bit 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.

9.4.2 Serial Port Mode Register (SMD)

7							0
—	—	—	—	—			
0	0	0	0	0	0	0	0
r	r	r	r	r	rw	rw	rw

Serial Port Mode Register (SMD)
Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

Bit 3: 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.

Bit 2: 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: 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

9.4.3 Serial Port Data Buffer Register (SBUF)

7							0
0	0	0	0	0	0	0	0
rw	rw	rw	rw	rw	rw	rw	rw

Serial Port Data Buffer Register (SBUF)
Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

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.

9.4.4 Serial Port Phase Register (PR)

15															0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Serial Port Phase Register (PR)
Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

This register is used to load and read the value in the phase register.

SECTION 10: SERIAL PERIPHERAL INTERFACE (SPI) MODULE

This section contains the following information:

10.1 SPI Transfer Formats 10-2
10.2 SPI Slave Select 10-4
10.3 SPI Character Lengths 10-4
10.4 SPI Transfer Baud Rates 10-4
10.5 SPI System Errors 10-4
10.5.1 Mode Fault 10-4
10.5.2 Receive Overrun 10-5
10.5.3 Write Collision While Busy 10-5
10.6 SPI Master Operation 10-5
10.7 SPI Slave Operation 10-5
10.8 SPI Peripheral Registers 10-6
10.8.1 SPI Control Register (SPICN) 10-6
10.8.2 SPI Configuration Register (SPICF) 10-7
10.8.3 SPI Clock Register (SPICK) 10-8
10.8.4 SPI Data Buffer Register (SPIB) 10-8

LIST OF FIGURES

Figure 10-1. SPI Block Diagram 10-2
Figure 10-2. SPI Transfer Formats (CKPHA = 1) 10-3
Figure 10-3. SPI Transfer Formats (CKPHA = 0) 10-3

LIST OF TABLES

Table 10-1. SPI Module Signal Functions 10-2

SECTION 10: SERIAL PERIPHERAL INTERFACE (SPI) MODULE

The serial peripheral interface (SPI) module of the MAXQ610 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 4-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 1. The maximum data rate of the SPI interface is 1/2 the system clock frequency for master mode operation and 1/4 the system clock frequency for slave mode operation.

The four external interface signals used by the SPI module are MISO, MOSI, SPICK, and $\overline{SSEL}$. Table 10-1 shows the function of each of these signals.

Figure 10-1 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.

Table 10-1. SPI Module Signal Functions

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	Output 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
$\overline{SSEL}$: Slave Select	(Optional) Mode fault-detection input if enabled (MODFE = 1)	Slave-select input

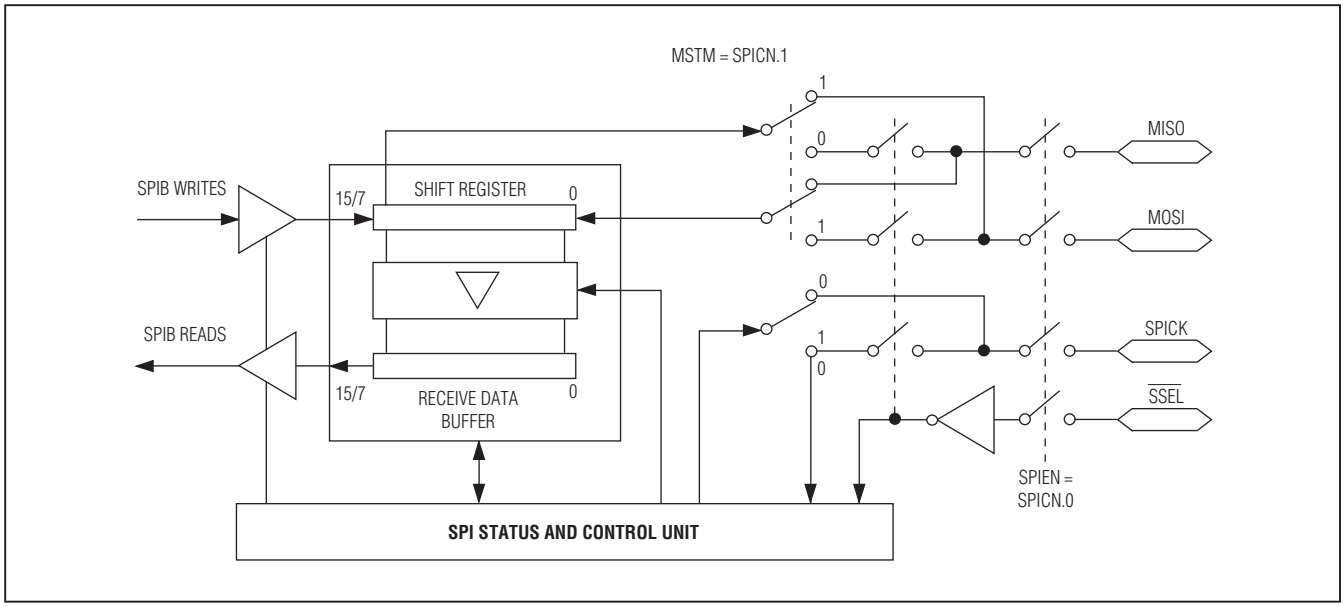


Figure 10-1. SPI Block Diagram

10.1 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 bit (CKPOL; SPICF.0) should be configured to a 0, while setting CKPOL = 1 causes 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 bit (CKPHA; SPICF.1) controls whether the active or inactive clock edge is used to latch the data. When CKPHA is set to 1, data is sampled on the inactive clock edge (clock returning to the idle state). When CKPHA is set to 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 illustrated in Figure 10-2 and Figure 10-3. The $\overline{SSEL}$ signal can remain asserted between successive transfers.

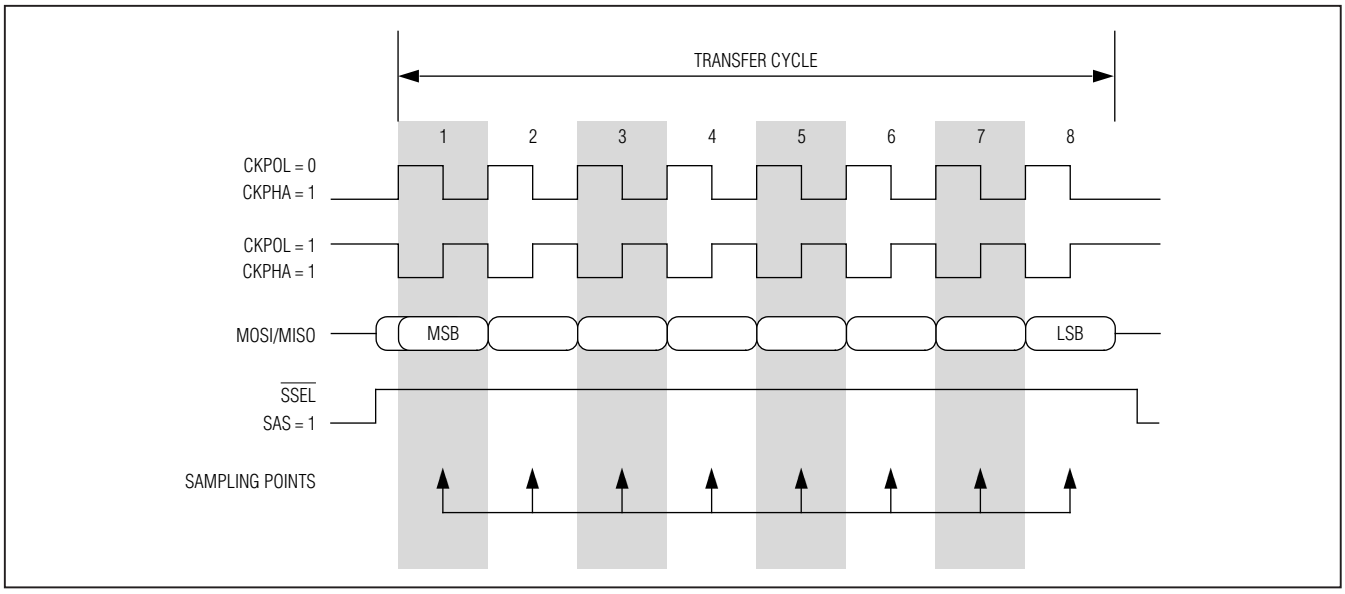


Figure 10-2. SPI Transfer Formats (CKPHA = 1)

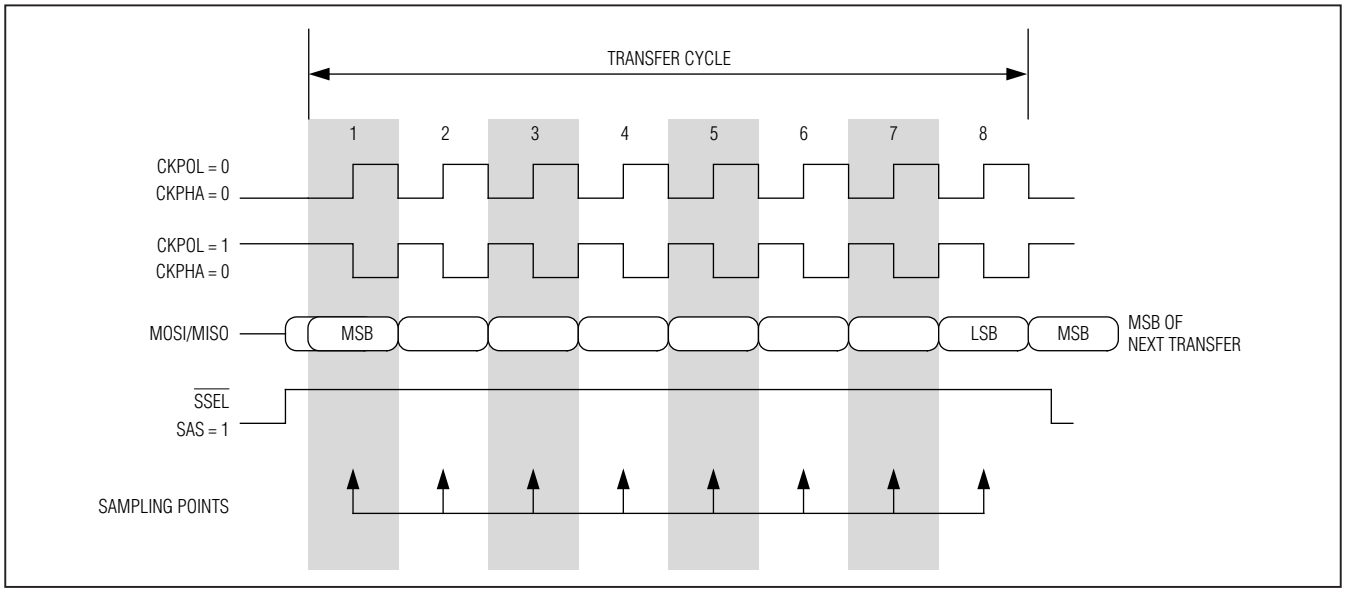


Figure 10-3. SPI Transfer Formats (CKPHA = 0)

10.2 SPI Slave Select

The SPI slave-select $\overline{SSEL}$ can be configured to accept either an active-low or active-high $\overline{SSEL}$ signal through the slave active select bit (SAS) in the SPI configuration register. The SAS bit allows the selection of the $\overline{SSEL}$ asserted state.

When SAS is cleared to 0, $\overline{SSEL}$ is configured to be asserted low. When SAS is set to 1, $\overline{SSEL}$ is configured to be asserted high.

10.3 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 most significant byte of the SPIB data buffer is not used when transmitting and receiving 8-bit characters.

10.4 SPI Transfer Baud Rates

When operating as a slave device, the SPI serial clock is driven by an external master. For proper slave operation, the serial clock provided by the external master should not exceed the system clock frequency divided by 4.

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:

$$\text{SPI Baud Rate} = \text{System Clock Frequency}/2 \times \text{Clock Divider Ratio}$$

$$\text{where clock divider ratio} = (\text{SPICK}[7:0]) + 1$$

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 $\overline{SSEL}$ 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.

10.5 SPI System Errors

Three types of SPI system errors can be detected by the SPI module. 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).

10.5.1 Mode Fault

When an 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 the $\overline{SSEL}$ input signal is asserted by an external device. The asserted state of $\overline{SSEL}$ is defined by slave active select bit (SPICN.6: SAS). If SAS is cleared to 0 and a low $\overline{SSEL}$ input signal is detected while MODFE is set, a mode fault error has occurred. If SAS is set to 1, a high $\overline{SSEL}$ signal indicates that a mode fault error has occurred. 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 0 to reconfigure the SPI device as a slave.
2) The SPIEN bit is forced to 0 to disable the SPI module.
3) The mode fault status flag (SPICN.3: MODF) 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 1 by software causes an interrupt if enabled.

Mode fault detection is optional and can be disabled by clearing the MODFE bit to 0. Disabling the mode fault detection disables the function of the $\overline{SSEL}$ 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 $\overline{SSEL}$ 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.

10.5.2 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 1 causes an interrupt if enabled. Once set, the ROVR bit is cleared only by software or a reset.

10.5.3 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 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.

10.6 SPI Master Operation

The SPI module is placed in master mode by setting the master mode enable bit (MSTM) in the SPI control register to 1. Only an SPI master device can initiate a data transfer. The master is responsible for manually selecting/deselecting slave(s) through the $\overline{SSEL}$ slave input signals. 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 all configurable as described earlier in the section. During the transfer, the SPI transfer busy flag (SPICN.7: STBY) 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 of the SPIC bit generates an interrupt request if SPI interrupt sources are enabled (ESPII = 1).

10.7 SPI Slave Operation

The SPI module operates in slave mode when the MSTM bit is cleared to 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 4.

The slave-select $\overline{SSEL}$ input must be externally asserted by a master before data exchange can take place. $\overline{SSEL}$ must be asserted before data transaction begin and must remain asserted 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 is transferred. The slave device considers a transfer to begin with

the first clock edge or the active $\overline{SSEL}$ edge, dependent on the data transfer format. When SAS is cleared to 0, the active $\overline{SSEL}$ edge is the falling edge of $\overline{SSEL}$, while if SAS is set to 1, the active $\overline{SSEL}$ edge is the rising edge of $\overline{SSEL}$.

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. Note also that when CKPHA = 0, the most significant bit of the SPI data buffer is shifted out on the 8th shift clock edge.

When $\overline{SSEL}$ 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 $\overline{SSEL}$ 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 the bit counter are reset.

In slave mode, the clock divider ratio bits (CKR[7:0]) have no function since the serial clock is supplied by an external master. 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.

10.8 SPI Peripheral Registers

10.8.1 SPI Control Register (SPICN)

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.

7							0
0	0	0	0	0	0	0	0
r	rw	rw	rw	rw	rw	rw	rw

SPI Control Register (SPICN)
Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

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 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 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 are always blocked. This bit must be cleared to 0 by software once set. Setting this bit to 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 $\overline{SSEL}$ 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 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 1, the $\overline{SSEL}$ input pin is used for mode fault detection during SPI master mode operation. When cleared to 0, the $\overline{SSEL}$ input has no function and its pin can be used for general-purpose I/O. In slave mode, the $\overline{SSEL}$ 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 $\overline{SSEL}$ 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

10.8.2 SPI Configuration Register (SPICF)

7							0
		—	—	—			
0	0	0	0	0	0	0	0
rw	rw	r	r	r	rw	rw	rw

SPI Configuration Register (SPICF)
Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

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

Bit 6: SPI Slave Active Select (SAS). This bit selects $\overline{SSEL}$ asserted state.

0 = $\overline{SSEL}$ is active low

1 = $\overline{SSEL}$ is active high

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 in conjunction with the CKPOL bit to define the SPI data transfer format.

0 = data sampled on the active clock edge

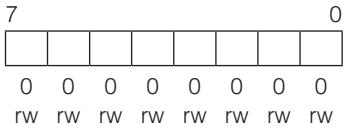
1 = data sampled on the inactive clock edge

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

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

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

10.8.3 SPI Clock Register (SPICK)



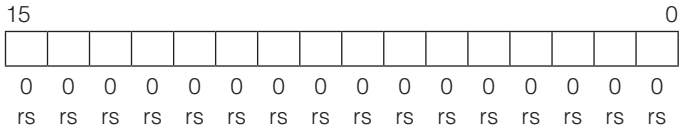
SPI Clock Register (SPICK)

Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

Bits 7:0: Clock Divider Ratio Bits 7:0 (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:

$$\text{SPI Baud Rate} = (0.5 \times \text{System Clock Frequency})/(\text{CKR}[7:0] + 1)$$

10.8.4 SPI Data Buffer Register (SPIB)



SPI Data Buffer Register (SPIB)

Power-On Reset and System Resets
Read (r), Write (w), or Special (s) access

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 the transfer cycle. When the STBY bit is set, write attempts are blocked and cause a write collision error.

SECTION 11: TEST ACCESS PORT (TAP)

This section contains the following information:

11.1 TAP Controller 11-2
11.2 TAP State Control 11-2
11.2.1 Test-Logic-Reset 11-2
11.2.2 Run-Test-Idle 11-3
11.2.3 IR-Scan Sequence 11-3
11.2.4 DR-Scan Sequence 11-5
11.3 Communication Through TAP 11-5
11.3.1 TAP Communication Examples—IR-Scan and DR-Scan 11-6

LIST OF FIGURES

Figure 11-1. TAP Controller State Diagram 11-3
Figure 11-2. TAP and TAP Controller 11-5
Figure 11-3. TAP Controller Debug Mode IR-Scan Example 11-6
Figure 11-4. TAP Controller Debug Mode DR-Scan Example 11-7

LIST OF TABLES

Table 11-1. TAP Signals 11-2
Table 11-2. Instruction Register Content vs. TAP Controller State 11-4
Table 11-3. Instruction Register (IR[2:0]) Encodings 11-4

SECTION 11: TEST ACCESS PORT (TAP)

The MAXQ610 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 MAXQ610 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 Table 11-1. For detailed information on the TAP and TAP controller, refer to IEEE Std 1149.1 "IEEE Standard Test Access Port and Boundary-Scan Architecture."

Table 11-1. TAP Signals

EXTERNAL PIN SIGNAL	DESCRIPTION
TDO: Test Data Output	Serial Data Output Pin. 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 Pin. 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 the 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 Pin. 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.

11.1 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 1/8th the system clock frequency. This section provides a brief description of the state machine and its state transitions. The state diagram in Figure 11-1 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.

11.2 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).

11.2.1 Test-Logic-Reset

On a power-on reset, the TAP controller is initialized to the test-logic-reset state and the instruction register (IR[2:0]) is initialized to the bypass 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 may 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.

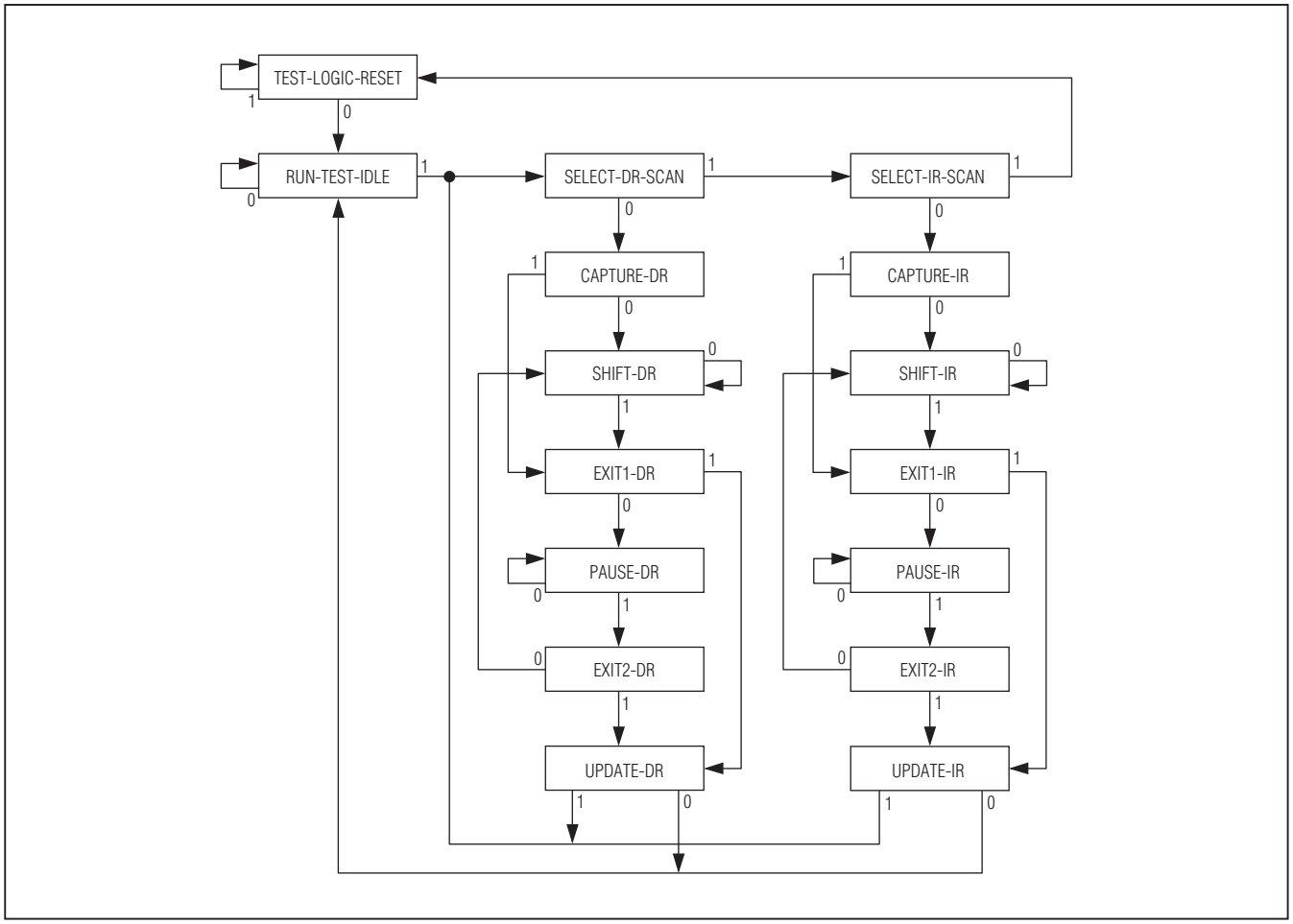


Figure 11-1. TAP Controller State Diagram

11.2.2 Run-Test-Idle

As illustrated in Figure 11-1, 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)

11.2.3 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 (IR[2:0]) on the falling edge of TCK in the update-IR state. The contents of the 3-bit instruction shift register and parallel instruction register (IR[2:0]) are summarized with respect to the TAP controller states in Table 11-2.

Table 11-2. Instruction Register Content vs. TAP Controller State

TAP CONTROLLER STATE	INSTRUCTION SHIFT REGISTER	PARALLEL (3-BIT) INSTRUCTION REGISTER (IR[2:0])
Test-Logic-Reset	Undefined	Set to bypass (011b) instruction
Capture-IR	Load 001b at the rising edge of TCK	Retain last state
Shift-IR	Input data through 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 (IR[2:0]) 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 (IR[2:0]). The supported instruction register encodings and associated data register selections are shown in Table 11-3.

Table 11-3. Instruction Register (IR[2:0]) Encodings

IR[2:0]	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	Bypass	No operation (default)	1-bit shift register
100	System Programming	Bootstrap function	3-bit shift register
101	Bypass	No operation (default)	1-bit shift register
110	Reserved	Reserved	Reserved
111	Bypass	No operation (default)	1-bit shift register

The extest (IR[2:0] = 000b) and sample/preload (IR[2:0] = 001b) instructions are mandated by the JTAG standard, however, the MAXQ610 microcontroller does not intend to make practical use of these instructions. Hence, these instructions are treated as no operations but can 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 bypass (IR[2:0] = 011b, 101b, or 111b) instruction is also mandated by the JTAG standard. The bypass instruction is fully implemented by the MAXQ610 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 bypass 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 bypass instruction when the TAP is in the test-logic-reset state. The bypass instruction has no effect on the operation of the on-chip system logic.

The debug (IR[2:0] = 010b) and system programming (IR[2:0] = 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 (IR[2:0]), a 3-bit serial data shift register is formed between the TDI and TDO pins in the shift-DR state.

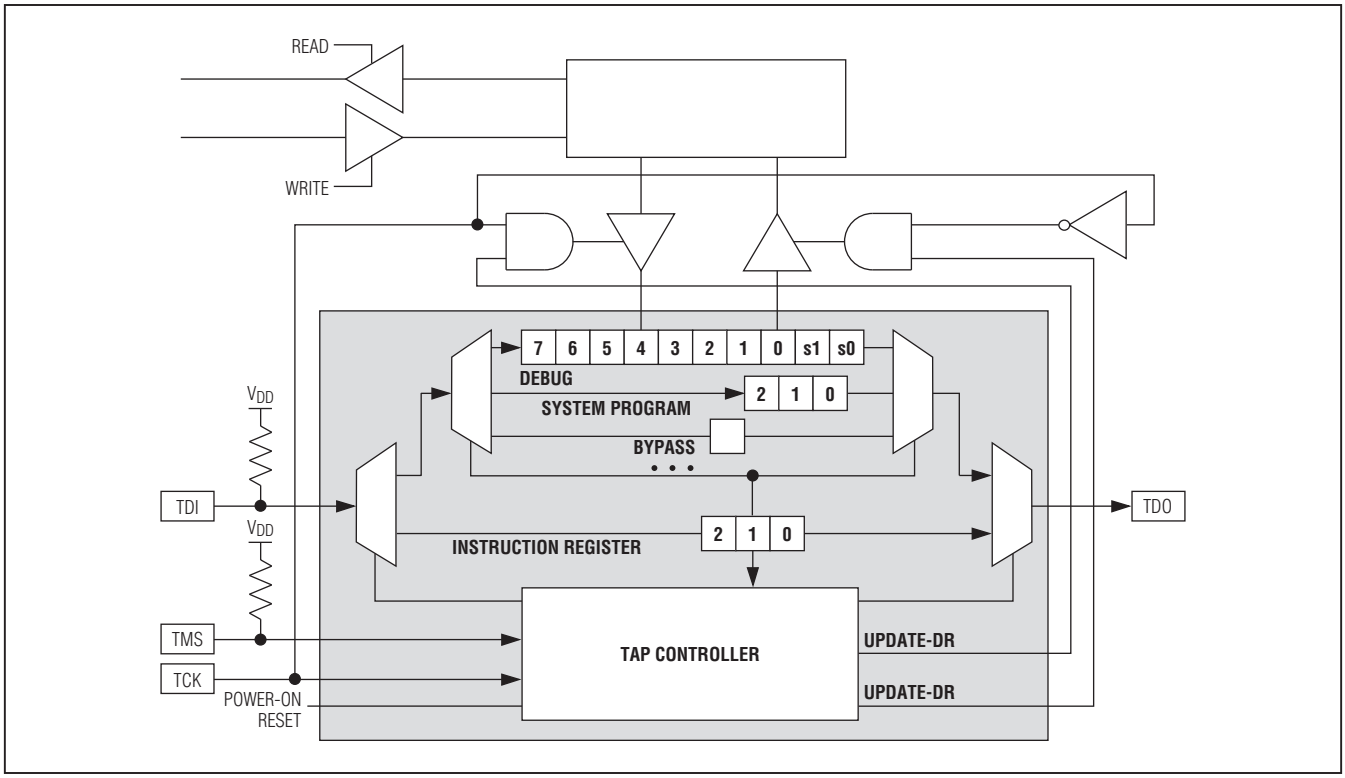


Figure 11-2. TAP and TAP Controller

Instruction register (IR[2:0]) settings other than those listed and described above are reserved for internal use. As can be seen in Figure 11-2, the instruction register serves to select the length of the serial data register between TDI and TDO during the shift-DR state.

11.2.4 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 through 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 MAXQ610 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.

11.3 Communication Through TAP

The TAP controller is in test-logic-reset state after a power-on-reset. During this initial state, the instruction register contains bypass 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 IR[2:0] 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 MAXQ610 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 IR[2:0] until a new instruction is shifted in or until the TAP controller returns to the test-logic-reset state.

11.3.1 TAP Communication Examples—IR-Scan and DR-Scan

Figure 11-3 and Figure 11-4 illustrate examples of communication between the host JTAG controller and the TAP of the MAXQ610 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.

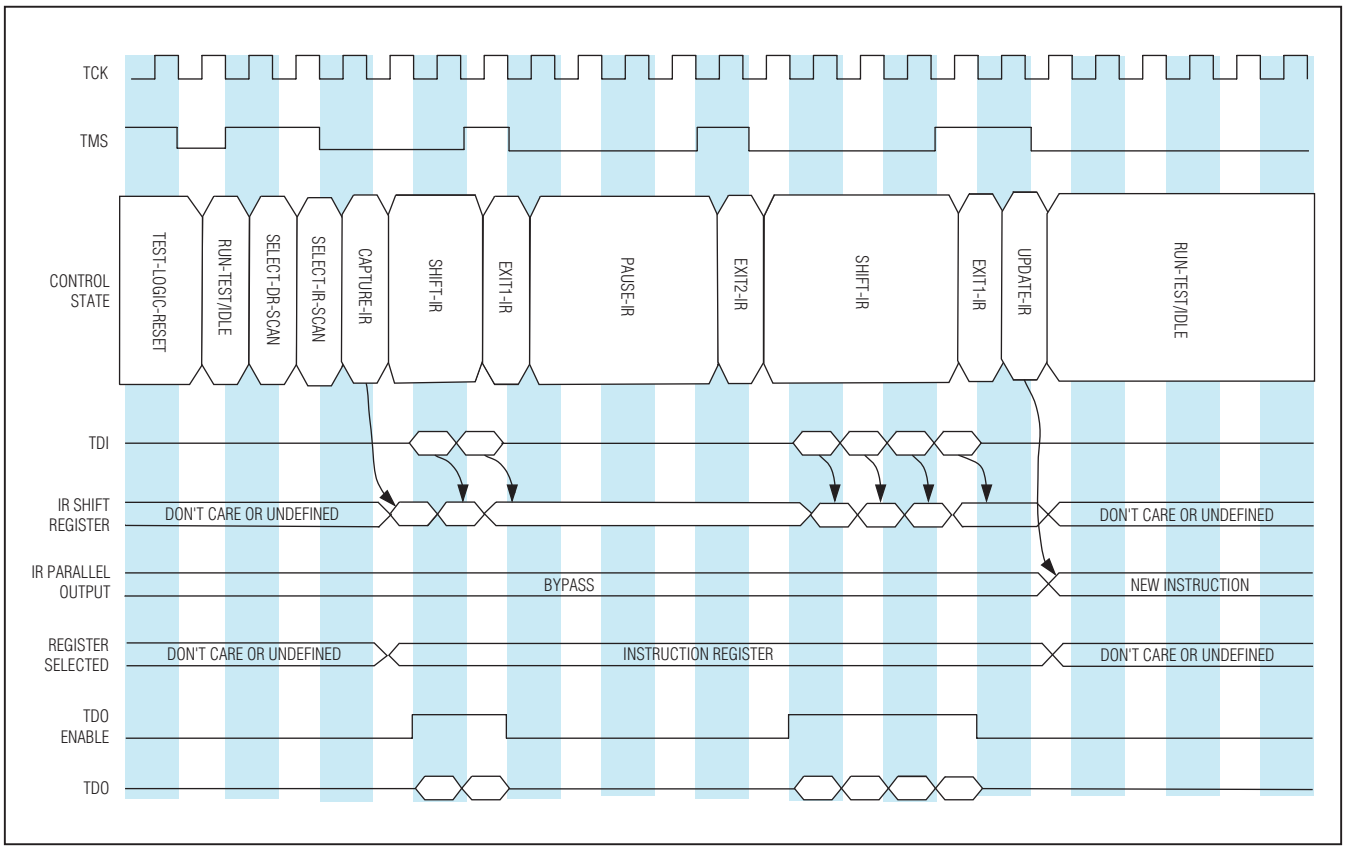


Figure 11-3. TAP Controller Debug Mode IR-Scan Example

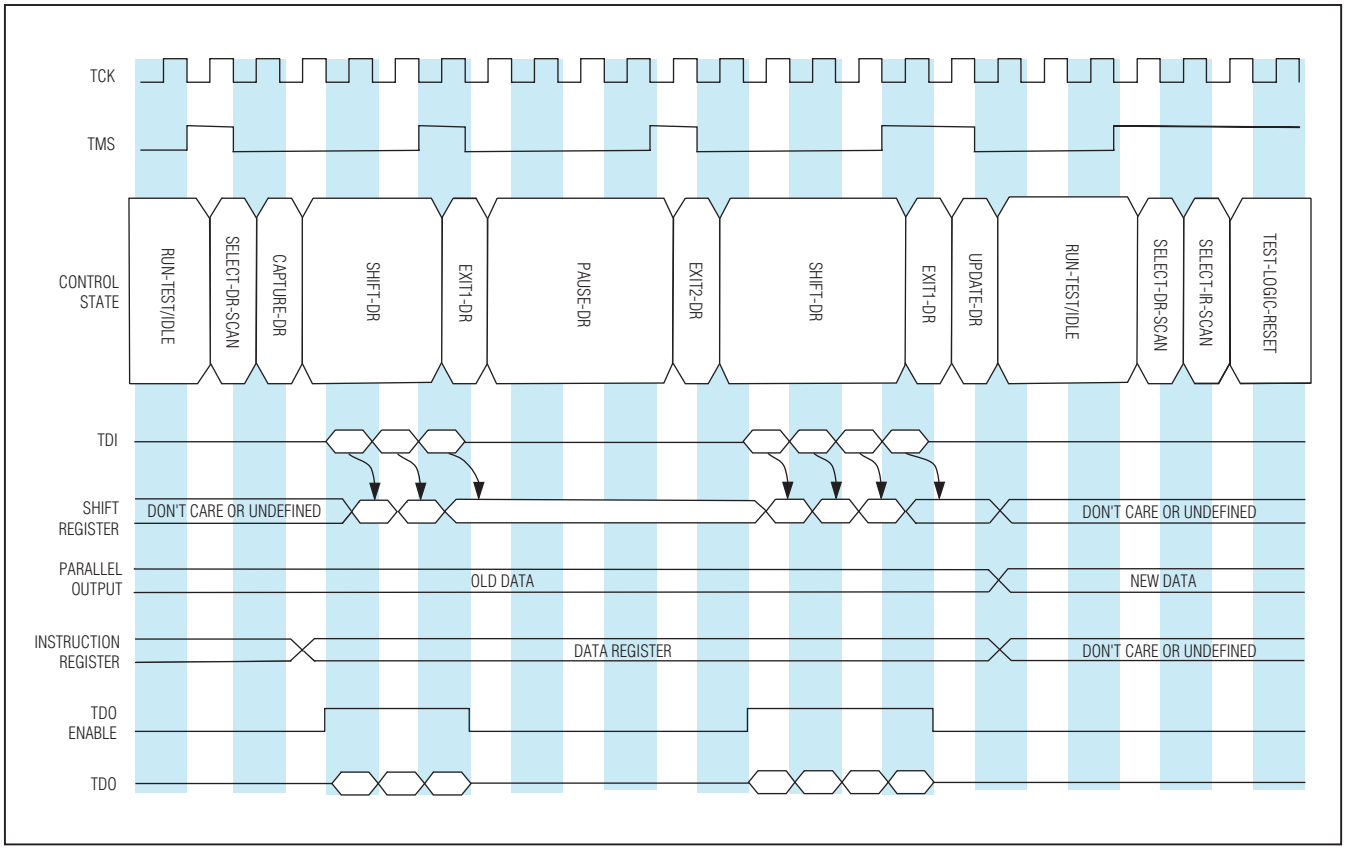


Figure 11-4. TAP Controller Debug Mode DR-Scan Example

SECTION 12: IN-CIRCUIT DEBUG MODE

This section contains the following information:

12.1 Background Mode Operation 12-3
12.2 Breakpoint Registers 12-5
12.2.1 Breakpoint n Register (BPn, n = 0 to 3) 12-5
12.2.2 Breakpoint 4 Register (BP4) 12-5
12.2.3 Breakpoint 5 Register (BP5) 12-6
12.2.4 Using Breakpoints 12-6
12.3 Debug Mode 12-7
12.3.1 Debug Mode Commands 12-7
12.3.2 Read Register Map Command Host-Utility ROM Interaction 12-9
12.3.3 Single-Step Operation (Trace) 12-10
12.3.4 Return 12-10
12.3.5 Debug Mode Special Considerations 12-10
12.4 In-Circuit Debug Peripheral Registers 12-11
12.4.1 In Circuit Debug Temp 0/1 Register (ICDT0/ICDT1) 12-11
12.4.2 In-Circuit Debug Control Register (ICDC) 12-11
12.4.3 In-Circuit Debug Flag Register (ICDF) 12-12
12.4.4 In-Circuit Debug Buffer Register (ICDB) 12-13
12.4.5 In-Circuit Debug Data Register (ICDD) 12-13
12.4.6 In-Circuit Debug Address Register (ICDA) 12-13

LIST OF FIGURES

Figure 12-1. In-Circuit Debugger 12-2

LIST OF TABLES

Table 12-1. Background Mode Commands 12-4
Table 12-2. Debug Mode Commands 12-8

SECTION 12: IN-CIRCUIT DEBUG MODE

Flash-based MAXQ610 microcontroller devices are equipped with embedded debug hardware and embedded utility 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 MAXQ610 microcontroller. Figure 12-1 shows a block diagram of the in-circuit debugger. The in-circuit debug hardware and software features include the following:

- Debug engine
- Set of registers providing the ability to set breakpoints on register, code, or data
- Set of debug service routines stored in a utility 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.

Note: The in-circuit debug peripheral registers ICDTn, ICDA, ICDB, ICDD, ICDC, and ICDF are used only by the utility ROM. The user does not have access to these registers.

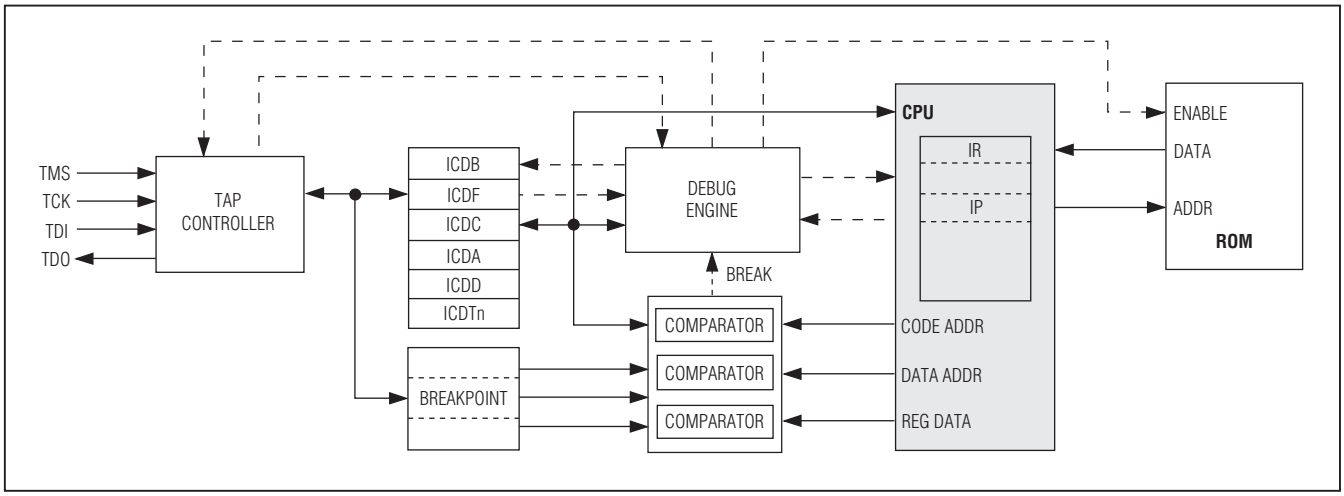


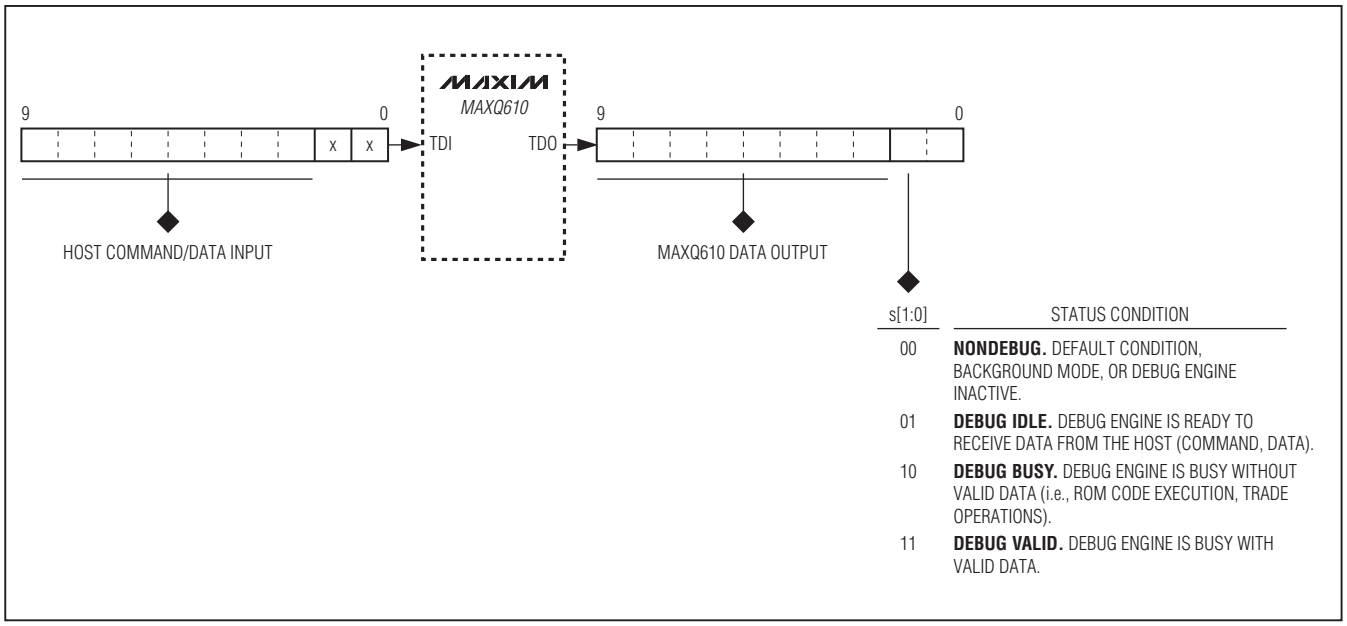
Figure 12-1. In-Circuit Debugger

The embedded hardware debug engine is implemented as a stand-alone hardware block in the MAXQ610 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, which 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 (IR[2:0]) 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 through 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 through this serial channel. Each 10-bit exchange of data between the host and the MAXQ610 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.



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.

12.1 Background Mode Operation

When the instruction register is loaded with the debug instruction (IR[2:0] = 010b), the host can communicate with the MAXQ610 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, PSS[1:0] = 100b), takes precedence over background mode communication. When operating in background mode, the status bits are always cleared to 00b (nondebug), which indicates that the MAXQ610 microcontroller is ready to receive background mode commands.

The host can perform the following operations from background mode:

- Read/write internal breakpoint registers (BP0 to 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

The background mode commands supported by the MAXQ610 microcontroller are shown in Table 12-1. Encodings not listed in this table are not supported in background mode and are treated as no operations.

Table 12-1. 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 through 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 (1 byte) are loaded into the debug shift register through 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 through 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 through 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 through 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 through 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 through 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 through 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 through 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 through 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.

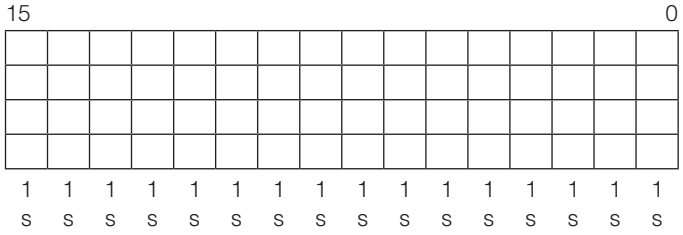
Table 12-1. Background Mode Commands (continued)

OP CODE	COMMAND	OPERATION
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 command is recognized by the debug engine.

12.2 Breakpoint Registers

The MAXQ610 microcontroller incorporates six breakpoint registers (BP0 to BP5) that are configurable by the host for establishing different types of breakpoint mechanisms. The first four breakpoint registers (BP0 to BP3) are 16-bit registers that are configurable as program memory address breakpoints. When enabled, the debug engine forces 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 can be configured as data memory address breakpoints or can be configured to support register access breakpoints. In either case, if breakpoints are enabled and the defined breakpoint match occurs, the debug engine generates a break condition.

12.2.1 Breakpoint n Register (BPn, n = 0 to 3)

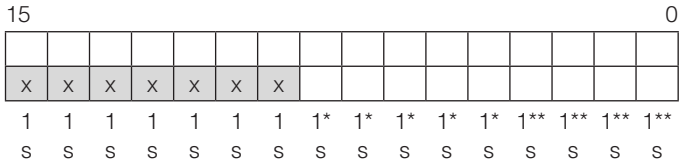


Breakpoint 0 Register (BP0)
Breakpoint 1 Register (BP1)
Breakpoint 2 Register (BP2)
Breakpoint 3 Register (BP3)

Power-On Reset and Test-Logic-Reset
Read (r), Write (w), or Special (s) access

These registers are accessible only through background mode read/write commands. These four registers 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.

12.2.2 Breakpoint 4 Register (BP4)



Breakpoint 4 Register (BP4) (REGE = 0)
Breakpoint 4 Register (BP4) (REGE = 1)

Power-On Reset and Test-Logic-Reset
Read (r), Write (w), or Special (s) access

***Module Specifier 3:0 {0 to 15}*

**Register Index within Module {0 to 31}*

This register is accessible only through 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 monitors 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.

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.

12.2.3 Breakpoint 5 Register (BP5)

15																0
x	x	x	x	x	x	x										
1	1	1	1	1	1	1	1*	1*	1*	1*	1*	1**	1**	1**	1**	
s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	

Breakpoint 5 Register (BP5) (REGE = 0)

Breakpoint 5 Register (BP5) (REGE = 1)

Power-On Reset and Test-Logic-Reset

Read (r), Write (w), or Special (s) access

***Module Specifier 3:0 {0 to 15}*

**Register Index within Module {0 to 31}*

This register is accessible only through 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 monitors 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.

When (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 are compared. When all bits in the ICDA register are cleared, Condition 2 becomes a don't care.

12.2.4 Using Breakpoints

All breakpoint registers (BP0 to BP5) default to the 0FFFFh 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 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 MAXQ610 microcontroller utility 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 MAXQ610 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 change from the nondebug (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.

12.3 Debug Mode

There are two ways to enter the debug mode from background mode:

- Issuance of the debug command directly by the host through 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 8010h in 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) Save the state of the UPA bit and clear it.
4) Set the contents of IP to 8010h.
5) Clear the IGE bit to 0 to disable interrupt handler if it is not already clear.
6) Halt CPU operation.

Once in debug mode, further breakpoint matches or host issuance of the debug command are treated as no operations and do 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.

12.3.1 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 regsiter (IR[2:0]) and

reloading the debug instruction. Once the debug engine has received the proper number of command and follow-on bytes for a given utility ROM assisted operation, it responds with the following actions:

- Update the command bits (CMD[3:0]) in the ICDC register to reflect the host request
- Enable the utility ROM if it is not enabled
- Force a jump to utility ROM address 8010h
- Set the data shift register status bits to 10b (debug-busy)

The utility ROM code performs a read to the ICDC register CMD[3:0] bits to determine its course of action. Some commands can be processed by the utility 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. In order to accomplish data flow control between the host and utility ROM, the status bits should be used by the host to assess when the utility ROM is ready for additional data and/or when the utility ROM is providing valid data output. Internally, the utility ROM can ascertain when new data is available or when it may output the next data byte through 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 utility 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 utility ROM operation done (ROD) bit of the SC register to 1. The ROD bit is reset by the debug engine when it recognizes the done condition.

The debug mode commands supported by the MAXQ610 microcontroller are shown in Table 12-2. 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 12-2. 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 CMD[3:0] bits in the ICDC to 0001b and perform a jump to utility ROM code at 8010h. The utility ROM debug service routine loads 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 utility ROM code. At the completion of this command period, the debug engine updates the CMD[3:0] bits to 0010b and performs a jump to utility ROM code at 8010h. The utility ROM debug service routine loads 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 utility ROM code. At the completion of this command period, the debug engine updates the CMD[3:0] bits to 0011b and performs a jump to utility ROM code at 8010h. The utility ROM debug service routine pops data out from the stack according to the information received in the ICDA and ICDD register. The stack pointer is postincremented for each pop operation.

Table 12-2. Debug Mode Commands (continued)

OP CODE	COMMAND	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 utility ROM code. At the completion of this command period, the debug engine updates the CMD[3:0] bits to 0100b and performs a jump to utility ROM code at 8010h. The utility ROM debug service routine updates 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 utility ROM code. At the completion of this command period, the debug engine updates the CMD[3:0] bits to 0101b and performs a jump to utility ROM code at 8010h. The utility ROM debug service routine updates the selected data memory location according to the information received in the ICDA 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/PWLL/PWLS bits and granting access to protected debug and loader functions. When this command is received, the debug engine updates the CMD[3:0] bit to 1000b and performs a jump to utility ROM code at 8010h. 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 utility ROM code. At the completion of this command period, the debug engine updates the CMD[3:0] bits to 1001b and performs a jump to utility ROM code at 8010h. The utility ROM debug service routine always assumes a 16-bit register length and returns the requested data LSB first.

12.3.2 Read Register Map Command Host-Utility 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 utility ROM to the 8-bit ICDB register and is output 1 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 finds 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 utility ROM service routine that another byte can be loaded to ICDB. The utility 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.

12.3.3 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 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) Save and clear the UPA bit.
9) Set the contents of IP to 8010h.
10) Clear the IGE bit to 0 to disable the interrupt handler.
11) Halt CPU operation.
12) 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 is 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.

However, single tracing through program in system memory is prohibited by hardware if multiple memory regions are defined.

12.3.4 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) Restore the state of the UPA bit.
3) Set the IGE bit to 1 if debug mode was activated when IGE = 1.
4) Supply the CPU with an instruction addressed by the return address.
5) Allow the CPU to execute the normal user program.
6) Set the status to 00b (nondebug).

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.

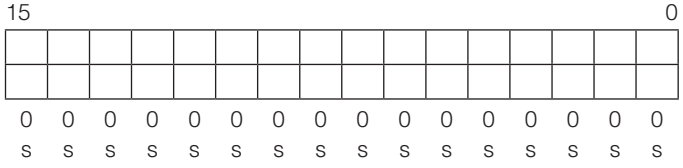
12.3.5 Debug Mode Special Considerations

- The debug engine does not operate reliably when the CPU is in 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 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 allows 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 $\overline{\text{RESET}}$ pin can be asserted to reset the debug engine and return to the background mode of operation.
- Single stepping (trace) through any operation that changes the state of the IGE bit 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.
- Data memory allocation is important during system development if in-circuit debug is planned. The top 32 bytes (16 words) of data memory are used by the debug service routine during debug mode. The data contents in these locations can 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 is 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 is made to selected clock source (which was being counted). It is important that the user recognize that this action occurs since the TAP clock should be run no faster than 1/8th 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.
- To debug UAPP/ULDR regions their passwords should not be cleared. The following methods can be used for this purpose:
  1) Place the application startup code at the start of the UAPP segment (or)
  2) Explicitly add a password to be located at word address 0x2010 (byte address 0x4020). Only one of the 32 bytes of the password field needs to be programmed to something other than 0xFF or 0x00.
- The stack plug-in should be disabled for debug commands to be effective.

12.4 In-Circuit Debug Peripheral Registers

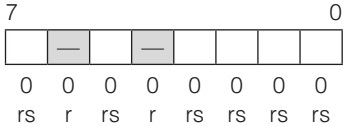
12.4.1 In Circuit Debug Temp 0/1 Register (ICDT0/ICDT1)



In Circuit Debug Temp 0 Register (ICDT0)
In Circuit Debug Temp 1 Register (ICDT1)
Power-On Reset or Test-Logic-Reset
Read (r), Write (w), or Special (s) access

These registers are read/write accessible by the CPU only in background mode or debug mode. These registers are intended for use by the utility ROM routines as temporary storage to save registers that might otherwise have to be placed in the stack.

12.4.2 In-Circuit Debug Control Register (ICDC)



In-Circuit Debug Control Register (ICDC)
Power-On Reset or Test-Logic-Reset
Read (r), Write (w), or Special (s) access

Bit 7: Debug Mode Enable (DME). When this bit is cleared to 0, background mode commands can be executed, but breakpoints are disabled. When this bit is set to 1, breakpoints are enabled while background mode commands still can be entered. This bit cany only be set or cleared from background debug mode. This bit has no meaning for the utility ROM code.

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 BP 5 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 utility ROM code.

Bits 3:0: 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 utility 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

12.4.3 In-Circuit Debug Flag Register (ICDF)

In-Circuit Debug Flag Register (ICDF)

Power-On Reset and Test-Logic-Reset

Read (r), Write (w), or Special (s) access

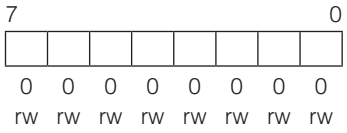
Bits 3:2: Programming Source Select Bits 1:0 (PSS[1:0]). These bits are used to select a programming interface during in-system programming when SPE is set to 1. Otherwise, the logic values of these bits have no meaning. The logical states of these bits, when read by the CPU, reflect the logical-OR of 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 through 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
Others		Reserved

Bit 1: System Program Enable (SPE). The SPE bit is used for in-system programming support and its logical state, when read by the CPU, always reflects the logical-OR of the SPE bit that is write accessible by the CPU and the SPE bit of the system programming buffer (SPB) register in the TAP module (which is accessible through JTAG). The logical state of this bit determines the program flow after a reset. When it is set to 1, in-system programming is executed by the utility ROM. When it is cleared to 0, execution is transferred to user code. This bit allows read/write access by the CPU 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 the clearing of 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 utility 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 the clearing of the JTAG PSS[1:0] bits.

12.4.4 In-Circuit Debug Buffer Register (ICDB)

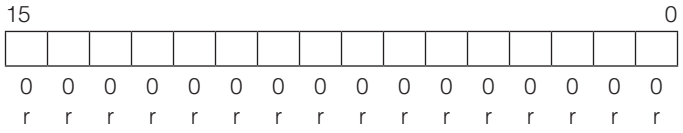


In-Circuit Debug Buffer Register (ICDB)

Power-On Reset and Test-Logic-Reset
Read (r), Write (w), or Special (s) access

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.

12.4.5 In-Circuit Debug Data Register (ICDD)

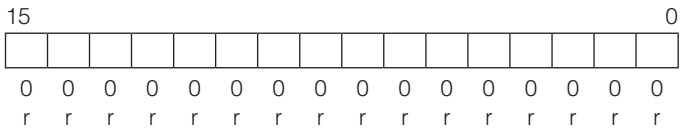


In-Circuit Debug Data Register (ICDD)

Power-On Reset or Test-Logic-Reset
Read (r), Write (w), or Special (s) access

This register is used by the debug engine to store data/read count so that utility 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 are compared with the updated destination data to determine if a break should be generated.

12.4.6 In-Circuit Debug Address Register (ICDA)



In-Circuit Debug Address Register (ICDA)

Power-On Reset or Test-Logic-Reset
Read (r), Write (w), or Special (s) access

This register is used by the debug engine to addresses so that utility 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 is 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 causes a break when the BP5 register matches the destination register address of the current instruction.

SECTION 13: IN-SYSTEM PROGRAMMING (JTAG)

This section contains the following information:

13.1 JTAG Bootloader Operation. 13-2
13.2 Password-Protected Access 13-3
13.2.1 Entering Passwords 13-3

LIST OF TABLES

Table 13-1. Status Bits for Bootloader Operation 13-2

SECTION 13: IN-SYSTEM PROGRAMMING (JTAG)

Internal nonvolatile (flash) memory of MAXQ610 microcontrollers can be initialized through 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 (IR[2:0]) 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 a 1 denotes that system programming is desired upon exiting reset. When it is cleared to 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 PSS[1:0] bits is 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 (PSS[1:0]). 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	Reserved
1	0	Reserved
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.

13.1 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 (PSS[1:0] = 00b), the background and active debug mode state machines are disabled. Once the host loads the debug instruction into the TAP instruction register (IR[2:0]), the 10-bit shift register interface to ICDB and the status bits become available for host-to-utility ROM bootloader communication. The status bits should be interpreted as shown in Table 13-1 for JTAG bootloader operation:

Table 13-1. Status Bits for Bootloader Operation

BITS 1:0	STATUS	CONDITION
00	Reserved	Invalid condition
01	Reserved	Invalid condition
10	Loader-Busy	Utility ROM loader is busy executing code or processing the current command
11	Loader-Valid	Utility ROM loader is supplying valid output data to the host in current shift operation

When the using the JTAG bootloader option (SPE = 1, PSS[1:0] = 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 utility 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 utility 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 utility ROM code can flexibly implement whatever communication protocol and command set it wishes within the data byte portion of the shifted 10-bit word.

13.2 Password-Protected Access

Some applications require preventive measures to protect against simple access and viewing of program code memory. To address this need for code protection, any MAXQ610 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 0010h to 001Fh of each memory area (system, user loader, user application, see Figure 2-7). 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.

Multiple password lock bits (PWL/PWLS/PWLL) are implemented in the SC register. When a PWL bit is set to 1, a password is required to access the in-circuit debug and in-system programming utility ROM routines that allow reading or writing of internal memory. When a PWL is cleared to 0, these utilities are fully accessible through the utility ROM without password.

The PWL bits default to 1 after a power-on reset. To access the ROM utilities, a correct password is needed; otherwise, access to the utility ROM utilities is denied. Once the correct password has been supplied by the user, the utility ROM clears the password lock. The PWLs remain clear until one of the following occurs:

- Power-on reset

or

- Set to 1 by user software

For flash-less 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 is rarely needed since the utility of memory programming and/or in-circuit debug to the end user is 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 is 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.

13.2.1 Entering Passwords

A password can be entered in one of two ways:

- Through the in-system programming interface established by the PSS[1:0] bits when SPE bit is set to 1; the utility ROM bootstrap loader dictates the protocol for entering the password over the specified serial communication interface.
- Through 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 14: MAXQ610 INSTRUCTION SET SUMMARY

Table 14-1. MAXQ610 Instruction Set Summary

	MNEMONIC	DESCRIPTION	16-BIT INSTRUCTION WORD	STATUS BITS AFFECTED	AP INC/DEC	EXECUTION CYCLES	NOTES
LOGICAL OPERATIONS	AND src	Acc ← Acc AND src	f001 1010 ssss ssss	S, Z	Y	1	1
	OR src	Acc ← Acc OR src	f010 1010 ssss ssss	S, Z	Y	1	1
	XOR src	Acc ← Acc XOR src	f011 1010 ssss ssss	S, Z	Y	1	1
	CPL	Acc ← ~Acc	1000 1010 0001 1010	S, Z	Y	1	—
	NEG	Acc ← ~Acc + 1	1000 1010 1001 1010	S, Z	Y	1	—
	SLA	Shift Acc left arithmetically	1000 1010 0010 1010	C, S, Z	Y	1	—
	SLA2	Shift Acc left arithmetically twice	1000 1010 0011 1010	C, S, Z	Y	1	—
	SLA4	Shift Acc left arithmetically four times	1000 1010 0110 1010	C, S, Z	Y	1	—
	RL	Rotate Acc left (w/o C)	1000 1010 0100 1010	S	Y	1	—
	RLC	Rotate Acc left (through C)	1000 1010 0101 1010	C, S, Z	Y	1	—
	SRA	Shift Acc right arithmetically	1000 1010 1111 1010	C, Z	Y	1	—
	SRA2	Shift Acc right arithmetically twice	1000 1010 1110 1010	C, Z	Y	1	—
	SRA4	Shift Acc right arithmetically four times	1000 1010 1011 1010	C, Z	Y	1	—
	SR	Shift Acc right (0 → msbit)	1000 1010 1010 1010	C, S, Z	Y	1	—
	RR	Rotate Acc right (w/o C)	1000 1010 1100 1010	S	Y	1	—
	RRC	Rotate Acc right (though C)	1000 1010 1101 1010	C, S, Z	Y	1	—
BIT OPERATIONS	MOVE C, Acc.<b>	C ← Acc.<b>	1110 1010 bbbb 1010	C	—	1	—
	MOVE C, #0	C ← 0	1101 1010 0000 1010	C	—	1	—
	MOVE C, #1	C ← 1	1101 1010 0001 1010	C	—	1	—
	CPL C	C ← ~C	1101 1010 0010 1010	C	—	1	—
	MOVE Acc.<b>, C	Acc.<b> ← C	1111 1010 bbbb 1010	S, Z	—	1	—
	AND Acc.<b>	C ← C AND Acc.<b>	1001 1010 bbbb 1010	C	—	1	—
	OR Acc.<b>	C ← C OR Acc.<b>	1010 1010 bbbb 1010	C	—	1	—
	XOR Acc.<b>	C ← C XOR Acc.<b>	1011 1010 bbbb 1010	C	—	1	—
	MOVE dst.<b>, #1	dst.<b> ← 1	1ddd dddd 1bbb 0111	C, S, E, Z	—	(Note 2)	3
	MOVE dst.<b>, #0	dst.<b> ← 0	1ddd dddd 0bbb 0111	C, S, E, Z	—	(Note 2)	3
	MOVE C, src.<b>	C ← src.<b>	fbbb 0111 ssss ssss	C	—	1	—
MATH	ADD src	Acc ← Acc + src	f100 1010 ssss ssss	C, S, Z, OV	Y	1	1
	ADDC src	Acc ← Acc + (src + C)	f110 1010 ssss ssss	C, S, Z, OV	Y	1	1
	SUB src	Acc ← Acc – src	f101 1010 ssss ssss	C, S, Z, OV	Y	1	1
	SUBB src	Acc ← Acc – (src + C)	f111 1010 ssss ssss	C, S, Z, OV	Y	1	1

Table 14-1. MAXQ610 Instruction Set Summary (continued)

	MNEMONIC	DESCRIPTION	16-BIT INSTRUCTION WORD	STATUS BITS AFFECTED	AP INC/DEC	EXECUTION CYCLES	NOTES
BRANCHING	{L/S}JUMP src	IP ← IP + src or src	f000 1100 ssss ssss	—	—	2	4
	{L/S}JUMP C, src	If C=1, IP ← (IP + src) or src	f010 1100 ssss ssss	—	—	2	4
	{L/S}JUMP NC, src	If C=0, IP ← (IP + src) or src	f110 1100 ssss ssss	—	—	2	4
	{L/S}JUMP Z, src	If Z=1, IP ← (IP + src) or src	f001 1100 ssss ssss	—	—	2	4
	{L/S}JUMP NZ, src	If Z=0, IP ← (IP + src) or src	f101 1100 ssss ssss	—	—	2	4
	{L/S}JUMP E, src	If E=1, IP ← (IP + src) or src	0011 1100 ssss ssss	—	—	2	4
	{L/S}JUMP NE, src	If E=0, IP ← (IP + src) or src	0111 1100 ssss ssss	—	—	2	4
	{L/S}JUMP S, src	If S=1, IP ← (IP + src) or src	f100 1100 ssss ssss	—	—	2	4
	{L/S}DJNZ LC[n], src	If --LC[n] <> 0, IP← (IP + src) or src	f10n 1101 ssss ssss	—	—	2	4
	{L/S}CALL src	@++SP ← IP+1; IP ← (IP+src) or src	f011 1101 ssss ssss	—	—	2	4, 5
	RET	IP ← @SP--	1000 1100 0000 1101	—	—	2	—
	RET C	If C=1, IP ← @SP--	1010 1100 0000 1101	—	—	2	—
	RET NC	If C=0, IP ← @SP--	1110 1100 0000 1101	—	—	2	—
	RET Z	If Z=1, IP ← @SP--	1001 1100 0000 1101	—	—	2	—
	RET NZ	If Z=0, IP ← @SP--	1101 1100 0000 1101	—	—	2	—
	RET S	If S=1, IP ← @SP--	1100 1100 0000 1101	—	—	2	—
	RETI	IP ← @SP-- ; IPS←11b	1000 1100 1000 1101	—	—	2	—
	RETI C	If C=1, IP ← @SP-- ; IPS←11b	1010 1100 1000 1101	—	—	2	—
	RETI NC	If C=0, IP ← @SP-- ; IPS←11b	1110 1100 1000 1101	—	—	2	—
	RETI Z	If Z=1, IP ← @SP-- ; IPS←11b	1001 1100 1000 1101	—	—	2	—
	RETI NZ	If Z=0, IP ← @SP-- ; IPS←11b	1101 1100 1000 1101	—	—	2	—
	RETI S	If S=1, IP ← @SP-- ; IPS←11b	1100 1100 1000 1101	—	—	2	—
DATA TRANSFER	XCH	Swap Acc bytes	1000 1010 1000 1010	S	Y	1	—
	XCHN	Swap nibbles in each Acc byte	1000 1010 0111 1010	S	Y	1	—
	MOVE dst, src	dst ← src	fddd dddd ssss ssss	C, S, Z, E	(Note 6)	(Notes 2, 7)	5, 6
	PUSH src	@++SP ← src	f000 1101 ssss ssss	—	—	(Note 2)	5
	POP dst	dst ← @SP--	1ddd dddd 0000 1101	C, S, Z, E	—	(Note 2)	5
	POPI dst	dst ← @SP-- ; IPS←11b	1ddd dddd 1000 1101	C, S, Z, E	—	(Note 2)	5
	CMP src	E ← (Acc = src)	f111 1000 ssss ssss	E	—	1	—
	NOP	No operation	1101 1010 0011 1010	—	—	1	—

Note 1: The active accumulator (Acc) is not allowed as the src in operations where it is the implicit destination.

Note 2: The CPU stalls when code is executed from flash with the destination being an IP register or when the code pointer is used. This stall requires two execution cycles to complete the instruction.

Note 3: Only module 8 and modules 0 to 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 4: The '{L/S}' prefix is optional.

Note 5: 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 6: The enabled AP autoincrement or decrement operation occurs for operations when specifying the active accumulator (Acc) as the source or destination (i.e., MOVE Acc, src; MOVE dst, Acc; MOVE Acc, Acc). Special cases: If 'MOVE APC, Acc' sets the APC.CLR bit, AP is cleared, overriding any autoinc/dec/modulo operation specified for AP. If 'MOVE AP, Acc' causes an autoinc/dec/modulo operation on AP, this overrides the specified data transfer (i.e., Acc is not transferred to AP).

Note 7: Exception for MOVE instruction, MOVE dp, @cp requires three cycles.

Note 8: The terms Acc and A[AP] can be used interchangeably to denote the active accumulator.

Note 9: Any index represented by <b> or found inside [ ] brackets is considered variable, but required.

Note 10: The active accumulator (Acc) is not allowed as the dst if A[AP] is specified as the src.

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 PFX[n] register may be used 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

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

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 PFX[n] register may be used 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

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

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
```

{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**) will result in an absolute long call and use of the PFX[0] register. Using the optional 'S' prefix (i.e., **SCALL**) will attempt to generate a relative short call, but will be flagged by the assembler if the destination is out or 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 PFX[n] register may be used 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

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 (one'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

Example(s):

```
                  ; Acc = 0FFFFh, 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 Flag: 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* opcode.

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**) will result in an absolute long jump and use of the PFX[0] register. Using the optional 'S' prefix (i.e. **SJUMP**) will attempt to generate a relative short jump, but will be flagged by the assembler if the destination is out or 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* opcode. Special *src* restrictions apply to **JUMP E** and **JUMP NE**.

Status Flags: None

JUMP C

Operation:
C=1: IP ← IP + *src (relative)* –or— *src (absolute)*
C=0: IP ← IP + 1

Encoding:

15			0
f010	1100	ssss	ssss

Example(s):
```
JUMP C, label1          ; C=0, branch not taken
```

JUMP NC

Operation:
C=0: IP ← IP + *src (relative)* –or— *src (absolute)*
C=1: IP ← IP +1

Encoding:

15			0
f010	1100	ssss	ssss

Example(s):
```
JUMP NC, label1         ; C=0, branch taken
```

JUMP Z

Operation:
Z=1: IP ← IP + *src*
Z=0: IP ← IP + 1

Encoding:

15			0
f001	1100	ssss	ssss

Example(s):
```
JUMP Z, label1          ; Z=1, branch taken
```

JUMP NZ

Operation:
Z=0: IP ← IP + *src (*relative) *–or— src* (absolute)
Z=1: IP ← IP + 1

Encoding:

15			0
f101	1100	ssss	ssss

Example(s):
```
JUMP NZ, label1         ; Z=1, branch not taken
```

JUMP E

Operation: E=1: IP ← IP + *src* (relative) –or— *src* (absolute)
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=1, branch not taken
Special Notes: The *src* specifier must be immediate data.

JUMP S

Operation: S=1: IP ← IP + *src* (relative) –or— *src* (absolute)
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 14-2. Source Specifier Codes

src	src BIT ENCODING f ssssssss	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 1st 16 Registers in Module NNN, shere NNN = 0 to 5; Access to 2nd 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
SC	1 1000 1000	8	System Control Register
IPR0	1 1001 1000	8	Interrupt Priority Register Zero
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, Pop (Postincrement SP)
SP	1 0001 1101	16	Stack Pointer
IV	1 0010 1101	16	Interrupt Vector
LC[n]	1 011n 1101	16	n Selects One of Two Loop Counter Registers
@SPI--	1 1000 1101	16	16-Bit Word @SP, Pop (Postincrement SP), IPS = 11b
@BP[OFFS]	1 0000 1110	8/16	Data Memory @BP[OFFS]
@BP[OFFS++]	1 0001 1110	8/16	Data Memory @BP[OFFS]; Postincrement OFFS
@BP[OFFS--]	1 0010 1110	8/16	Data Memory @BP[OFFS]; Postdecrement 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], Postincrement DP[n]
@DP[n]--	1 0n10 1111	8/16	Data Memory @DP[n], Postdecrement DP[n]
DP[n]	1 0n11 1111	16	n Selects One of Two Data Pointers
@CP	1 1000 1111	8/16	Code Memory @CP
@CP++	1 1001 1111	8/16	Code Memory @CP, Postincrement DP[n]
@CP--	1 1010 1111	8/16	Code Memory @CP, Postdecrement DP[n]
CP	1 1011 1111	16	Code Pointer

Table 14-3. 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 1st Eight 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
A[n]	nnn 1001	8/16	nnn Selects One of 1st Eight Accumulators: A[0] to A[7]
Acc	000 1010	8/16	Active Accumulator = A[AP]
PFX[n]	nnn 1011	8	nnn Selects One of Eight Prefix Registers
@++SP	000 1101	16	16-Bit Word @SP, Push (predecrement SP)
SP	001 1101	16	Stack Pointer
IV	010 1101	16	Interrupt Vector
LC[n]	11n 1101	16	n Selects One of Two Loop Counter Registers
@BP[OFFS]	000 1110	8/16	Data Memory @BP[OFFS]
@BP[++OFFS]	001 1110	8/16	Data Memory @BP[OFFS]; Preincrement OFFS
@BP[--OFFS]	010 1110	8/16	Data Memory @BP[OFFS]; Predecrement 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], Preincrement DP[n]
@--DP[n]	n10 1111	8/16	Data Memory @DP[n], Predecrement DP[n]
DP[n]	n11 1111	16	n Selects One of Two Data Pointers
2-CYCLE DESTINATION ACCESS USING PFX[n] Register (see Special Notes)			
SC	000 1000	16	System Control Register
IPR0	001 1000	16	Interrupt Priority Register Zero
CKCN	110 1000	8	Clock Control Register
WDCN	111 1000	8	Watchdog Control Register
A[n]	nnn 1001	16	nnn Selects One of 2nd Eight Accumulators A[8] to A[15]
GRH	001 1110	8	High Byte of GR Register
CP	011 1111	16	Code Pointer

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 DP[0], #00h
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

Example(s):

```
                         ; Acc = 8000h, S=1, Z=0, C=0
MOVE Acc.15, C           ; Acc = 0000h, S=0, Z=1
```

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

Description: Replaces the Carry (C) status flag with the specified active accumulator bit.
Status Flag: C
Operation: C ← Acc.<b>

Encoding:

15			0
1110	1010	bbbb	1010

Example(s):

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

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

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

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 to 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

Example(s):

```
                          ; Acc = 0FEEDh, 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 PFX[n] register may be used 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

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

Example(s):

```
OR Acc.1        ; Acc.1=0 → C=0
OR Acc.2        ; Acc.2=1 → C=1
```

POP *dst* — Pop Word from the Stack

Description: Pops a single word from the stack (@SP) to the specified *dst* and decreases the stack depth (increments 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):

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

Stack Data:

xxxxh	
xxxxh	← SP (after POP @DP[0])
0876h	← SP (after POP GR)
1234h	← SP (initial)
xxxxh	

POPI *dst* — Pop Word from the Stack Enable Interrupts

Description: Pops a single word from the stack (@SP) to the specified *dst* and decreases the stack depth (increments 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--
IPS ← 11b

Encoding:

15			0
1ddd	dddd	1000	1101

Example(s): See **POP**

PUSH *src* — Push Word to the Stack

Description: Increases the stack depth (decments 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	
xxxxh	← SP (initial)
0F3Fh	← SP (after PUSH GR)
0040h	← SP (after PUSH #40h)
xxxxh	

RET — Return from Subroutine

Description: RET pops a single word from the stack (@SP) into the Instruction Pointer (IP) and decreases the stack depth (increments the stack pointer SP). The modified 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)	Opcode
0311h	…
0312h	RET
→ 0103h	…

Stack Data:

xxxxh	
xxxxh	← SP (after RET)
0103h	← SP (before RET)
xxxxh	
xxxxh	

RET C/RET NC
RET Z/RET NZ Conditional Return on Status Flag
RET S

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

Operation:
C=1: IP ← @SP--
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--
IPS ← 11b

Encoding:

15			0
1000	1100	1000	1101

Example(s): see **RET**

RETI C/RETI NC
RETI Z/RETI NZ Conditional Return from Interrupt on Status Flag
RETI S

Description: Performs conditional return from interrupt (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 from interrupt operation.

Status Flags: None

RETI C

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: C=1: IP ← @SP--
IPS ← 11b
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--
IPS ← 11b
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--
IPS ← 11b
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--
IPS ← 11b
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--
IPS ← 11b
S=0: IP ← IP + 1

Encoding:

15			0
1100	1100	1000	1101

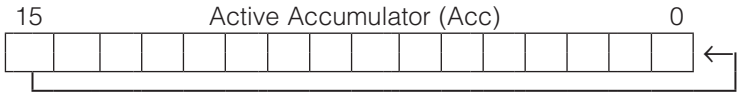
Example(s):
```
RETI S                  ; S=0, return from interrupt (RETI) does not occur
```

RL/RLC — Rotate Left Accumulator Carry Flag Exclusive/Inclusive

Description: Rotates the active accumulator left by a single bit position. The **RL** instruction circulates the msbit of the accumulator (bit 15) back to the lsbit (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

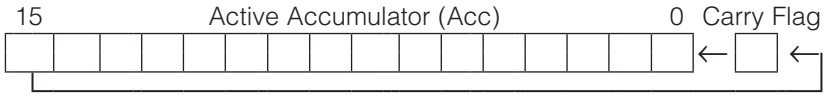
Encoding:

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):

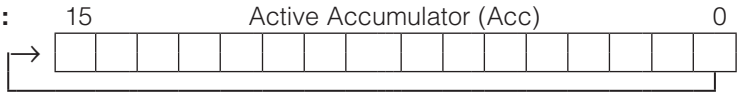
```
                        ; Acc = A345h, C=1, S=1, Z=0
RLC                     ; Acc = 468Bh, C=1, S=0, Z=0
RLC                     ; Acc = 8D17h, C=0, S=1, Z=0
```

RR/RRC — Rotate Right Accumulator Carry Flag Exclusive/Inclusive

Description: Rotates the active accumulator right by a single bit position. The **RR** instruction circulates the lsbit of the accumulator (bit 0) back to the msbit (bit 15) while the **RRC** instruction includes the Carry (C) flag in the circular right shift.

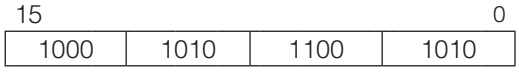
Status Flags: C (for RRC only), S, Z (for RRC only)

RR Operation:



Acc.[14:0] ← Acc.[15:1]; Acc.15 ← Acc.0

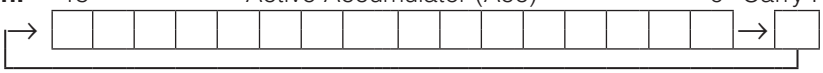
Encoding:



Example(s):

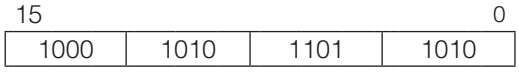
```
                        ; Acc = A345h, S=1, Z=0
RR                      ; Acc = D1A2h, S=1, Z=0
RR                      ; Acc = 68D1h, S=0, Z=0
```

RRC Operation:



Acc.[14:0] ← Acc.[15:1]; Acc.15 ← C; C ← Acc.0

Encoding:



Example(s):

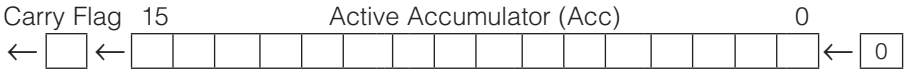
```
                        ; Acc = A345h, C=1, S=1, Z=0
RRC                     ; Acc = D1A2h, C=1, S=1, Z=0
RRC                     ; Acc = E8D1h, C=0, S=1, Z=0
```

SLA/SLA2/SLA4 — Shift Accumulator Left Arithmetically 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 lsbit and the msbit 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

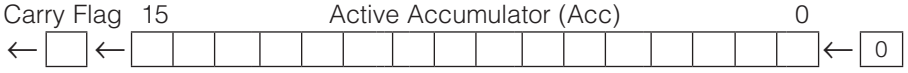
Encoding:

15			0
1000	1010	0010	1010

Example(s):

```
                        ; Acc = E345h, C=0, S=1, Z=0
SLA                     ; Acc = C68Ah, 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

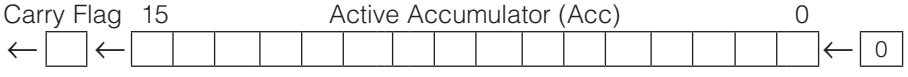
Encoding:

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):

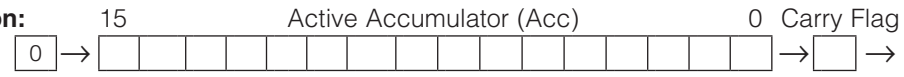
```
                        ; Acc = E345h, C=0, S=1, Z=0
SLA4                    ; Acc = 3450h, C=0, S=0, Z=0
```

SR — Shift Accumulator Right
SRA/SRA2/SRA4 — 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 msbit while the **SRA**, **SRA2**, and **SRA4** instructions effectively shift a copy of the current msbit into the accumulator, thereby preserving any sign orientation. For each shift iteration, the accumulator lsbit is shifted into the Carry (C) flag.

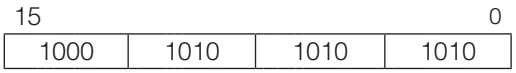
Status Flags: C, S (changes for SR only), Z

SR Operation:



Acc.15 ← 0; Acc.[14:0] ← Acc.[15:1]; C ← Acc.0

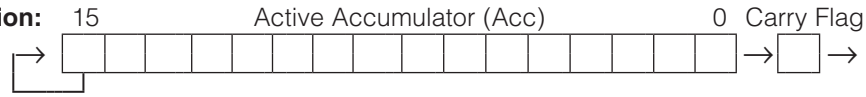
Encoding:



Example(s):

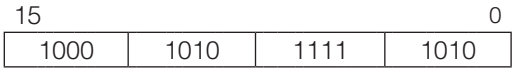
```
                        ; Acc = A345h, C=1, S=1, Z=0
SR                      ; Acc = 51A2h, C=1, S=0, Z=0
SR                      ; Acc = 28D1h, C=0, S=0, Z=0
```

SRA Operation:



Acc.[14:0] ← Acc.[15:1]
Acc.15 ← Acc.15
C ← Acc.0

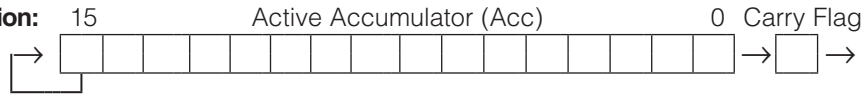
Encoding:



Example(s):

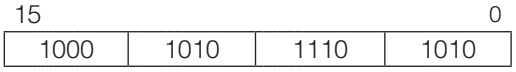
```
                        ; Acc = 0003h, C=0, Z=0
SRA                     ; Acc = 0001h, C=1, Z=0
SRA                     ; Acc = 0000h, C=1, Z=1
```

SRA2 Operation:



Acc.[13:0] ← Acc.[15:2]
Acc.[15:14] ← Acc.15
C ← Acc.1

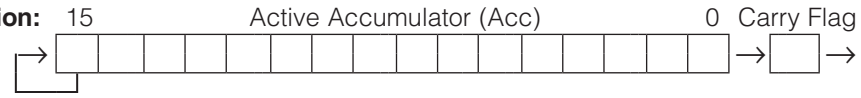
Encoding:



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 PFX[n] register may be used 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

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

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.

XCH — Exchange Accumulator Bytes

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
XCH                     ; 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

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 PFX[n] register may be used 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

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

Example(s):

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

SECTION 15: UTILITY ROM

This section contains the following information:

15.1 In-Application Programming Functions 15-3
15.1.1 UROM_flashWrite 15-3
15.1.2 UROM_flashErasePage 15-3
15.1.3 UROM_flashEraseAll 15-3
15.2 Data Transfer Functions 15-4
15.2.1 UROM_moveDP0 15-4
15.2.2 UROM_moveDP0inc 15-4
15.2.3 UROM_moveDP0dec 15-4
15.2.4 UROM_moveDP1 15-5
15.2.5 UROM_moveDP1inc 15-5
15.2.6 UROM_moveDP1dec 15-5
15.2.7 UROM_moveFP 15-6
15.2.8 UROM_moveFPinc 15-6
15.2.9 UROM_moveFPdec 15-6
15.2.10 UROM_moveBP 15-7
15.2.11 UROM_copyBuffer 15-7
15.3 Miscellaneous Functions 15-7
15.3.1 UROM_stopMode 15-7
15.4 ROM Example 1: Calling A Utility ROM Function Directly 15-8
15.5 ROM Example 2: Calling A Utility ROM Function Indirectly 15-9

LIST OF TABLES

Table 15-1. Functions for MAXQ610 Utility ROM Version 1.00 15-2

SECTION 15: UTILITY ROM

The MAXQ610 utility ROM includes routines that provide the following functions to application software.

- In-application programming routines for flash memory (program, erase, mass erase)
- Single word/byte copy and buffer copy routines for use with lookup tables
- Entry into stop mode
- Ability to check a value against a stored secret

To provide backwards compatibility among different versions of the utility ROM, a function address table is included that contains the entry points for all user-callable functions. With this table, user code can determine the entry point for a given function as follows:

1) Read the location of the function address table from address 0800Dh in the utility ROM.
2) The entry points for each function listed below are contained in the function address table, one word per function, in the order given by their function numbers.

For example, the entry point for the **UROM_flashEraseAll** function can be determined by the following procedure:

1) functionTable = dataMemory[0800Dh]
2) flashWriteEntry = dataMemory[functionTable + 0]

Table 15-1. Functions for MAXQ610 Utility ROM Version 1.00

INDEX	FUNCTION NAME	ENTRY POINT	SUMMARY
0	UROM_flashWrite	8544h	Programs a single word of flash memory.
1	UROM_flashErasePage	8566h	Erases (programs to FFFFh) a 512-word sector of flash memory.
2	UROM_flashEraseAll	857Bh	Erases (programs to FFFFh) all flash memory.
3	UROM_moveDP0	8589h	Reads a byte/word at DP[0].
4	UROM_moveDP0inc	858Ch	Reads a byte/word at DP[0], then increments DP[0].
5	UROM_moveDP0dec	858Fh	Reads a byte/word at DP[0], then decrements DP[0].
6	UROM_moveDP1	8592h	Reads a byte/word at DP[1].
7	UROM_moveDP1inc	8595h	Reads a byte/word at DP[1], then increments DP[0].
8	UROM_moveDP1dec	8598h	Reads a byte/word at DP[1], then decrements DP[0].
9	UROM_moveFP	859Bh	Reads a byte/word at BP[OFFS].
10	UROM_moveFPinc	859Eh	Reads a byte/word at BP[OFFS], then increments OFFS.
11	UROM_moveFPdec	85A1h	Reads a byte/word at BP[OFFS], then decrements OFFS.
12	UROM_copyBuffer	85A4h	Copies LC[0] values (up to 255) from DP[0] to BP[OFFS].
13	UROM_stopMode	85AAh	Enters stop mode.

It is also possible to call utility ROM functions directly, using the entry points given above. Standard include files are provided for this purpose with the MAXQ development toolset. This method calls functions more quickly, but the application might need to be recompiled in order to run properly with a different version of the utility ROM.

15.1 In-Application Programming Functions

15.1.1 UROM_flashWrite

Function: UROM_flashWrite
Summary: Programs a single word of flash memory.
Inputs: A[0]: Word address in program flash memory to write to.
A[1]: Word value to write to flash memory.
Outputs: Carry: Set on error and cleared on success.
Destroys: PSF, LC[1]

Notes:

- This function uses one stack level to save and restore values.
- If the watchdog reset function is active, it should be disabled before calling this function.
- If the flash location has already been programmed to a non-FFFF value, this function returns with an error (carry set). To reprogram a flash location, it must first be erased by calling **UROM_flashErasePage** or **UROM_flashEraseAll**.

15.1.2 UROM_flashErasePage

Function: UROM_flashErasePage
Summary: Erases (programs to 0FFFFh) a 256-word page of flash memory.
Inputs: A[0]: Word address located in the page to be erased. (The page number is the high byte of A[0].)
Outputs: Carry: Set on error and cleared on success.
Destroys: LC[1], A[0]

Notes:

- If the watchdog reset function is active, it should be disabled before calling this function.
- When calling this function from flash, care should be taken that the return address is not in the page that is being erased.

15.1.3 UROM_flashEraseAll

Function: UROM_flashEraseAll
Summary: Erases (programs to 0FFFFh) all locations in flash memory.
Inputs: None.
Outputs: Carry: Set on error and cleared on success.
Destroys: LC[1], A[0]

Notes:

- If the watchdog reset function is active, it should be disabled before calling this function.
- This function can only be called by code running from the RAM. Attempting to call this function while running from the flash results in an error.

15.2 Data Transfer Functions

15.2.1 UROM_moveDP0

Function: UROM_moveDP0
Summary: Reads the byte/word value pointed to by DP[0].
Inputs: DP[0]: Address to read from.
Outputs: GR: Data byte/word read.
Destroys: None.

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 2-4 and Figure 2-5. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash/ROM memory.
- This function automatically refreshes the data pointer before reading the byte/word value.

15.2.2 UROM_moveDP0inc

Function: UROM_moveDP0inc
Summary: Reads the byte/word value pointed to by DP[0], then increments DP[0].
Inputs: DP[0]: Address to read from.
Outputs: GR: Data byte/word read.
DP[0] is incremented.
Destroys: None.

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 2-4 and Figure 2-5. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash/ROM memory.
- This function automatically refreshes the data pointer before reading the byte/word value.

15.2.3 UROM_moveDP0dec

Function: UROM_moveDP0dec
Summary: Reads the byte/word value pointed to by DP[0], then decrements DP[0].
Inputs: DP[0]: Address to read from.
Outputs: GR: Data byte/word read.
DP[0] is decremented.
Destroys: None.

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 2-4 and Figure 2-5. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash/ROM memory.
- This function automatically refreshes the data pointer before reading the byte/word value.

15.2.4 UROM_moveDP1

Function: UROM_moveDP1
Summary: Reads the byte/word value pointed to by DP[1].
Inputs: DP[1]: Address to read from.
Outputs: GR: Data byte/word read.
Destroys: None.

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 2-4 and Figure 2-5. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash/ROM memory.
- This function automatically refreshes the data pointer before reading the byte/word value.

15.2.5 UROM_moveDP1inc

Function: UROM_moveDP1inc
Summary: Reads the byte/word value pointed to by DP[1], then increments DP[1].
Inputs: DP[1]: Address to read from.
Outputs: GR: Data byte/word read.
DP[1] is incremented.
Destroys: None.

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 2-4 and Figure 2-5. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash/ROM memory.
- This function automatically refreshes the data pointer before reading the byte/word value.

15.2.6 UROM_moveDP1dec

Function: UROM_moveDP1dec
Summary: Reads the byte/word value pointed to by DP[1], then decrements DP[1].
Inputs: DP[1]: Address to read from.
Outputs: G: Data byte/word read.
DP[1] is decremented.
Destroys: None.

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 2-4 and Figure 2-5. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash/ROM memory.
- This function automatically refreshes the data pointer before reading the byte/word value.

15.2.7 UROM_moveFP

Function: UROM_moveFP
Summary: Lookup table access using BP[OFFS].
Inputs: BP[OFFS]: Location to read from in data space.
Outputs: GR: Data byte/word read.
Destroys: None.

Notes:

- Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 2-4 and Figure 2-5. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash/ROM memory.
- This function automatically refreshes the data pointer before reading the byte/word value.

15.2.8 UROM_moveFPinc

Function: UROM_moveFPinc
Summary: Lookup table access using BP[OFFS], then increments OFFS.
Inputs: BP[OFFS]: Location to read from in data space.
Outputs: GR: Data byte/word read.
OFFS is incremented.
Destroys: None.

Notes:

- Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 2-4 and Figure 2-5. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash/ROM memory.
- This function automatically refreshes the data pointer before reading the byte/word value.

15.2.9 UROM_moveFPdec

Function: UROM_moveFPdec
Summary: Lookup table access using BP[OFFS], then decrements OFFS.
Inputs: BP[OFFS]: Location to read from in data space.
Outputs: GR: Data byte/word read.
OFFS is decremented.
Destroys: None.

Notes:

- Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 2-4 and Figure 2-5. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash/ROM memory.
- This function automatically refreshes the data pointer before reading the byte/word value.

15.2.10 UROM_moveBP

Function: UROM_moveBP
Summary: Reads the byte/word value pointed to by BP[OFFS].
Inputs: BP[OFFS]: Address to read from.
Outputs: GR: Data byte/word read.
Destroys: None.

Notes:

- Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 2-4 and Figure 2-5. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash/ROM memory.
- This function automatically refreshes the data pointer before reading the byte/word value.

15.2.11 UROM_copyBuffer

Function: UROM_copyBuffer
Summary: Copies LC[0] bytes/words (up to 255) from DP[0] to BP[OFFS].
Inputs: DP[0]: Address to copy from.
BP[OFFS]: Address to copy to.
LC[0]: Number of bytes or words to copy.
Outputs: OFFS is incremented by LC[0].
DP[0] is incremented by LC[0].
Destroys: LC[0].

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[0] and BP[OFFS] for byte or word mode. Both DP[0] and BP[OFFS] should be configured to the same mode (byte or word) for correct buffer copying.
- The addresses passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 2-4 and Figure 2-5. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash/ROM memory.
- This function automatically refreshes the data pointers before reading the byte/word values.

15.3 Miscellaneous Functions

15.3.1 UROM_stopMode

Function: UROM_stopMode
Summary: Enters stop mode.
Inputs: None.
Outputs: None.
Destroys: None.

15.4 ROM Example 1: Calling A Utility ROM Function Directly

This example shows the direct addressing method for calling utility functions, using the function **moveDP1inc** to read a static string from code space. Note the equate UROM_MOVEDP1INC.

```
UROM_MOVEDP1INC EQU 087DBh
Text:
   DB "Hello World!",0      ; Define a string in code space.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function:       PrintText
;; Description:    Prints the string stored at the "Text" label.
;; Returns:        N/A
;; Destroys:       ACC, DP[1], DP[0], and GR.
;; Notes:          This function assumes that DP[0] is set to word mode,
;;                 DP[1] is in byte mode, and the device has 16-bit accumulators.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
PrintText:
   move DP[1], #Text        ; Point to the string to display.
   move ACC, DP[1]          ; "Text" is a word address and we need a
   sla                      ; byte address, so shift left 1 bit.
   or   #08000h             ; Code space is mapped to 8000h when running
   move DP[1], ACC          ; from the ROM, so the address must be masked.
PrintText_Loop:
   call UROM_MOVEDP1INC     ; Fetch the byte from code space.
   move ACC, GR
   jump Z, PrintText_Done   ; Reached the null terminator.
   call PrintChar           ; Call a routine to output the char in ACC
   jump PrintText_Loop      ; Process the next byte.
PrintText_Done:
   ret
```

15.5 ROM Example 2: Calling A Utility ROM Function Indirectly

The second example shows the indirect addressing method (lookup table) for calling utility functions. We use the same function (**UROM_MoveDP1Inc**) to read our static string, but this time we must figure out the address we want dynamically. Note the inserted code where we before had a direct call to the function. Also note that the function index of **moveDP1inc** is 7.

```
Text:
   DB "Hello World!",0     ; Define a string in code space.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function:       PrintText
;; Description:    Prints the string stored at the "Text" label.
;; Returns:        N/A
;; Destroys:       ACC, DP[1], DP[0], and GR.
;; Notes:          This function assumes that DP[0] is set to word mode,
;;                 DP[1] is in byte mode, and the device has 16-bit
;;                 accumulators.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
PrintText:
   move DP[1], #Text       ; Point to the string to display.
   move ACC, DP[1]         ; "Text" is a word address and we need a
   sla                     ; byte address, so shift left 1 bit.
   or   #08000h            ; Code space is mapped to 8000h when running
   move DP[1], ACC         ; from the ROM, so the address must be masked.
PrintText_Loop:
;
; Fetch the byte from code space.
;
   move DP[0], #0800Dh     ; This is where the address of the table is stored.
   move ACC, @DP[0]        ; Get the location of the function table.
   add  #7                 ; Add the index to the moveDP1inc function.
   move DP[0], ACC         ; Point to where the address of moveDP1 is stored.
   move ACC, @DP[0]        ; Retrieve the address of the function.
   call ACC                ; Execute the function.

   move ACC, GR
   jump Z, PrintText_Done  ; Reached the null terminator.
   call PrintChar          ; Call a routine to output the char in ACC
   jump PrintText_Loop     ; Process the next byte.
PrintText_Done:
   ret
```

REVISION HISTORY

REVISION NUMBER	REVISION DATE	SECTION NUMBER	DESCRIPTION	PAGES CHANGED
0	10/09	—	Initial release	—
1	6/10	2	Added *Section 2.6.3: Memory Access Protection Impact on Data Pointers (and Code Pointer)*; added note about nop to *Section 2.13: Stop Mode*	2-15, 2-30
2	7/10	12	Added two bullet points to *Section 12.3.5: Debug Mode Special Considerations* to explain how to debug UAPP/ULDR and note that the stack plug-in should be disabled	12-11