



Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



Contact us

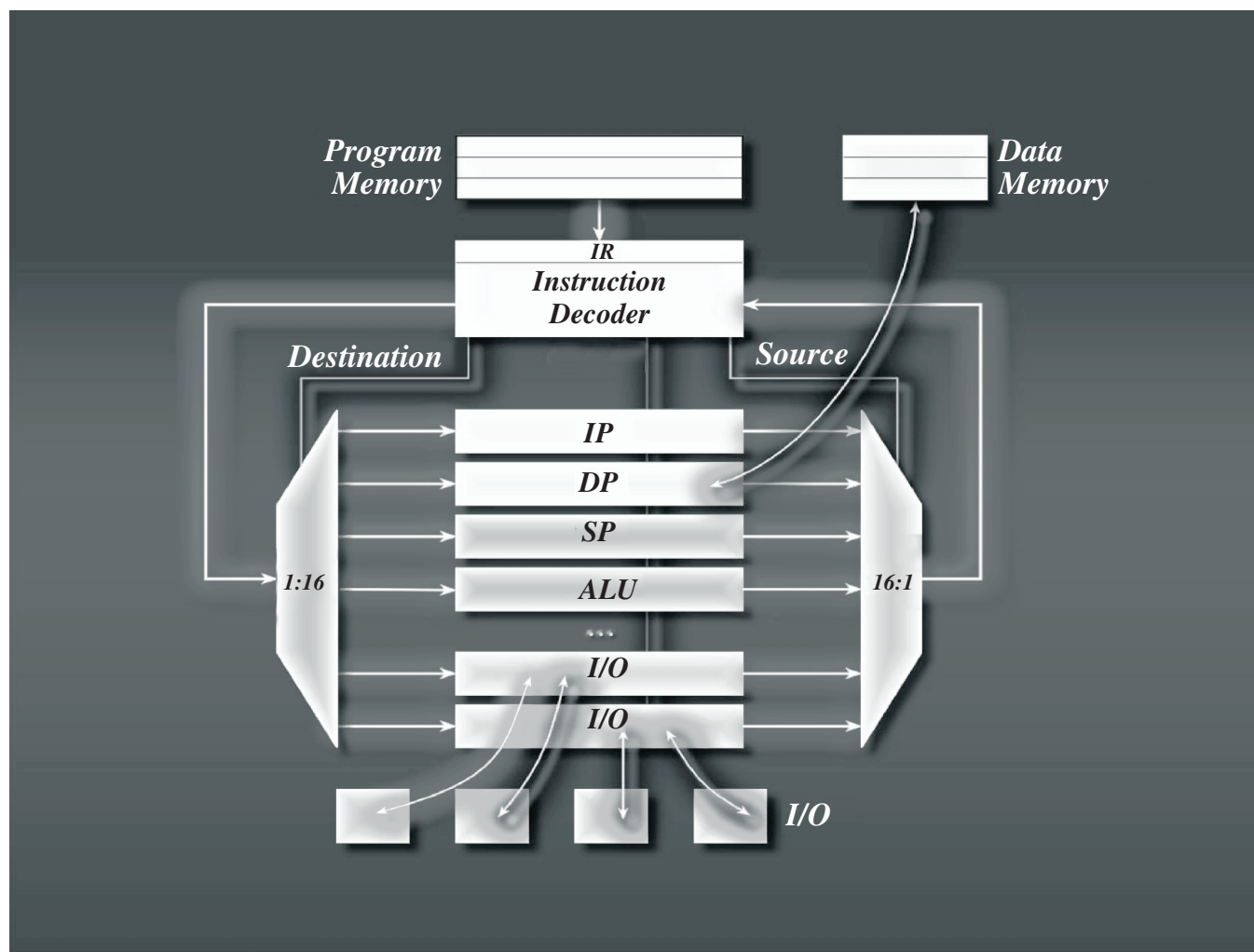
Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com

Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China



MAXQ FAMILY USER'S GUIDE



Note: The MAXQ Family User's Guide should be used in conjunction with the data sheet(s) for MAXQ microcontrollers.

TABLE OF CONTENTS

SECTION 1: Overview	1-1
SECTION 2: Architecture	2-1
SECTION 3: Programming	3-1
SECTION 4: System Register Descriptions	4-1
SECTION 5: Peripheral Register Modules	5-1
SECTION 6: General-Purpose I/O Module	6-1
SECTION 7: Timer/Counter Timer	7-1
SECTION 8: Timer/Counter 1 Module	8-1
SECTION 9: Timer/Counter 2 Module	9-1
SECTION 10: Serial I/O Module	10-1
SECTION 11: Serial Peripheral Interface (SPI Module)	11-1
SECTION 12: Hardware Multiplier Mode	12-1
SECTION 13: 1-Wire Bus Master	13-1
SECTION 14: Real-Time Clock Module	14-1
SECTION 15: Test Access Port (TAP)	15-1
SECTION 16: In-Circuit Debug Mode	16-1
SECTION 17: In-System Programming (JTAG)	17-1
SECTION 18: MAXQ Family Instruction Set Summary	18-1
REVISION HISTORY	Rev-1

SECTION 1: OVERVIEW

This section contains the following information:

1.1 Instruction Set	1-2
1.2 Harvard Memory Architecture	1-2
1.3 Register Set	1-2
1.4 MAXQ10 and MAXQ20 Microcontrollers	1-3

SECTION 1: OVERVIEW

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

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

1.1 Instruction Set

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

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

1.2 Harvard Memory Architecture

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

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

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

1.3 Register Set

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

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

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

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

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

*MAXQ is a registered trademark of Maxim Integrated Products, Inc.
SPI is a trademark of Motorola, Inc.*

MAXQ Family User's Guide

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

1.4 MAXQ10 and MAXQ20 Microcontrollers

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

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

SECTION 2: ARCHITECTURE

This section contains the following information:

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	2-6
2.3.4 Stack Memory	2-7
2.4 Pseudo-Von Neumann Memory Mapping	2-7
2.5 Pseudo-Von Neumann Memory Access	2-9
2.6 Data Alignment	2-10
2.6.1 Memory Management Unit	2-10
2.7 Clock Generation	2-13
2.7.1 Internal Ring Oscillator	2-14
2.7.2 External Clock (Crystal/Resonator)	2-14
2.7.3 External Clock (Direct Input)	2-14
2.7.4 External RC	2-15
2.7.5 Internal System Clock Generation	2-15
2.8 Interrupts	2-16
2.8.1 Servicing Interrupts	2-16
2.8.2 Interrupt System Operation	2-16
2.8.3 Synchronous vs. Asynchronous Interrupt Sources	2-17
2.8.4 Interrupt Prioritization by Software	2-18
2.8.5 Interrupt Exception Window	2-18
2.9 Operating Modes	2-18
2.9.1 Reset Mode	2-18
2.9.1.1 Power-On/Brownout Reset	2-18
2.9.1.2 External Reset	2-19
2.9.1.3 Watchdog Timer Reset	2-19
2.9.1.4 Internal System Reset	2-19

2.9.2 Power Management Mode	2-19
2.9.2.1 Switchback	2-20
2.9.3 Stop Mode	2-20

LIST OF FIGURES

Figure 2-1. MAXQ Transport-Triggered Architecture	2-3
Figure 2-2. Instruction Word Format	2-4
Figure 2-3. Pseudo Von Neumann Memory Map (MAXQ10 Default)	2-8
Figure 2-4. Pseudo Von Neumann Memory Map (MAXQ20 Default)	2-9
Figure 2-5. CDA Functions (Word Access Mode)	2-11
Figure 2-6. CDA Functions (Byte Access Mode)	2-12
Figure 2-7. MAXQ Clock Sources	2-13
Figure 2-8. On-Chip Crystal Oscillator	2-15
Figure 2-9. RC Relaxation Oscillator	2-15
Figure 2-10. MAXQ Interrupt Source Hierarchy Example	2-17

LIST OF TABLES

Table 2-1. Register-to-Register Transfer Operations	2-5
Table 2-2. System Clock Rate Control Settings	2-15

MAXQ Family User's Guide

SECTION 2: ARCHITECTURE

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

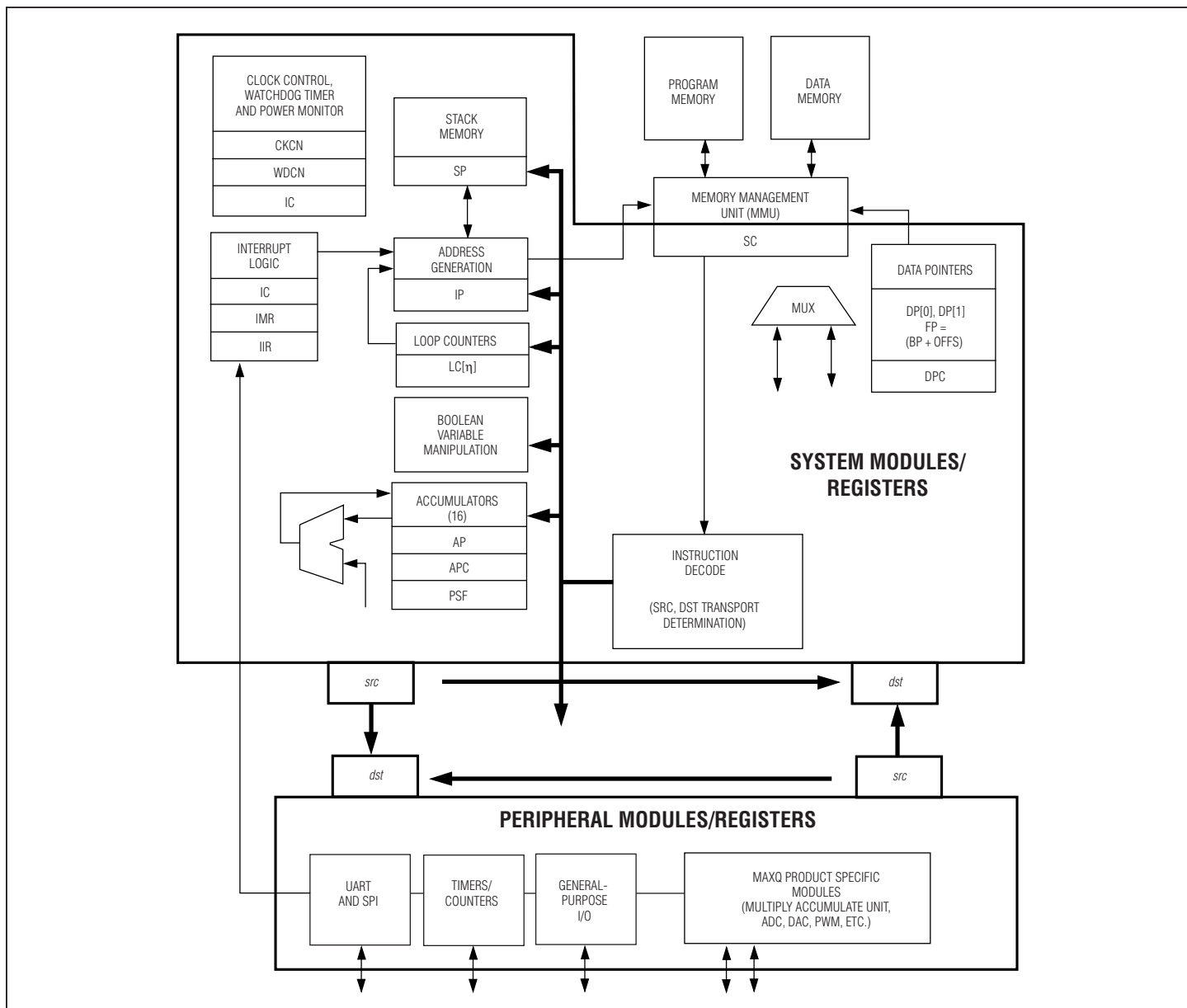


Figure 2-1. MAXQ Transport-Triggered Architecture

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

2.1 Instruction Decoding

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

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

Figure 2-2. Instruction Word Format

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

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

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

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

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

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

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

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

MAXQ Family User's Guide

2.2 Register Space

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

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

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

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

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

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

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

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

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

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

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

2.3 Memory Organization

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

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

2.3.1 Program Memory

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

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

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

2.3.2 Utility ROM

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

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

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

2.3.3 Data Memory

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

The core incorporates two 16-bit Data Pointers (DP[0] and DP[1]) to support data memory accessing. All Data Pointers support indirect addressing mode and indirect addressing with auto-increment or auto-decrement. Data Pointers DP[0] and DP[1] can be used as

MAXQ Family User's Guide

post increment/decrement source pointers by a MOVE instruction or pre increment/decrement destination pointers by a MOVE instruction. Using Data Pointer indirectly with "++" will automatically increase the content of the active Data Pointer by 1 immediately following the execution of read data transfer (@DP[n]++) or immediately preceding the execution of a write operation (@++DP[n]). Using Data Pointer indirectly with "--" will decrease the content of the active Data Pointer by 1 immediately following the execution of read data transfer (@DP[n]--) or immediately preceding the execution of a write operation (@--DP[n]).

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

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

2.3.4 Stack Memory

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

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

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

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

2.4 Pseudo-Von Neumann Memory Mapping

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

The program memory is logically divided into four program pages:

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

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

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

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

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

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

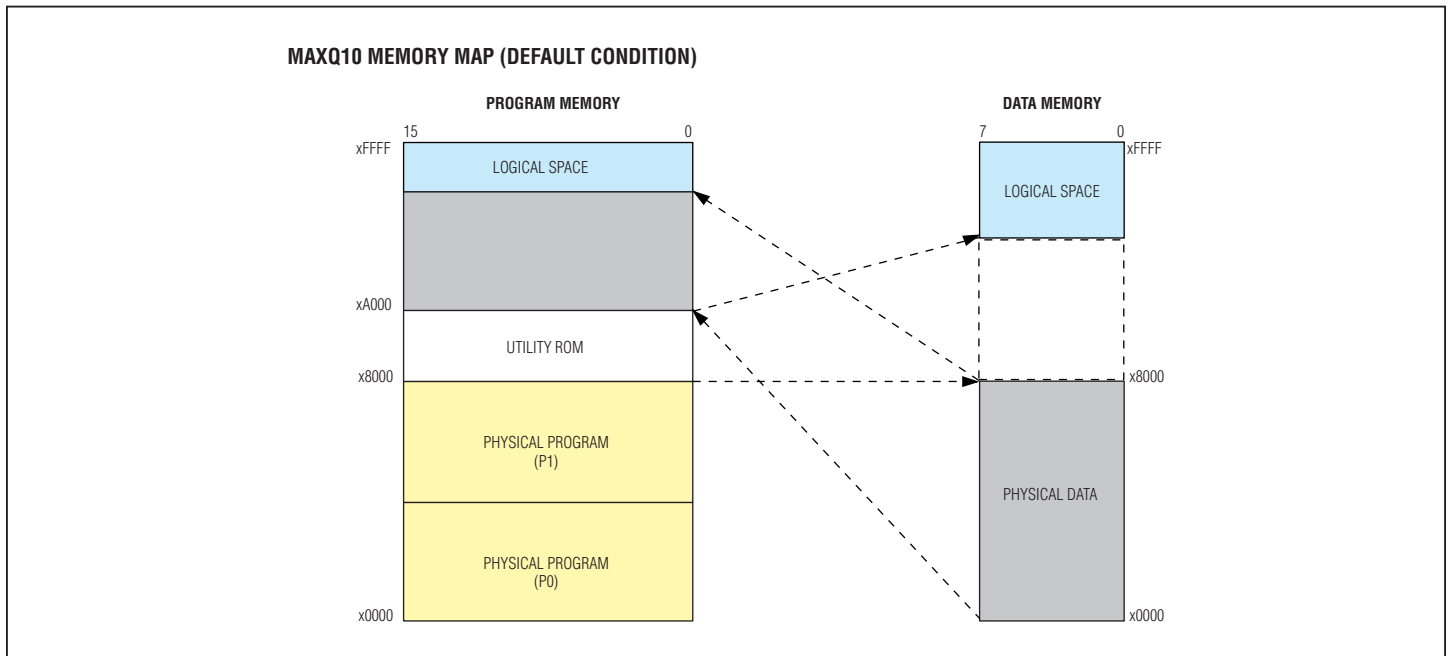


Figure 2-3. Pseudo Von Neumann Memory Map (MAXQ10 Default)

MAXQ Family User's Guide

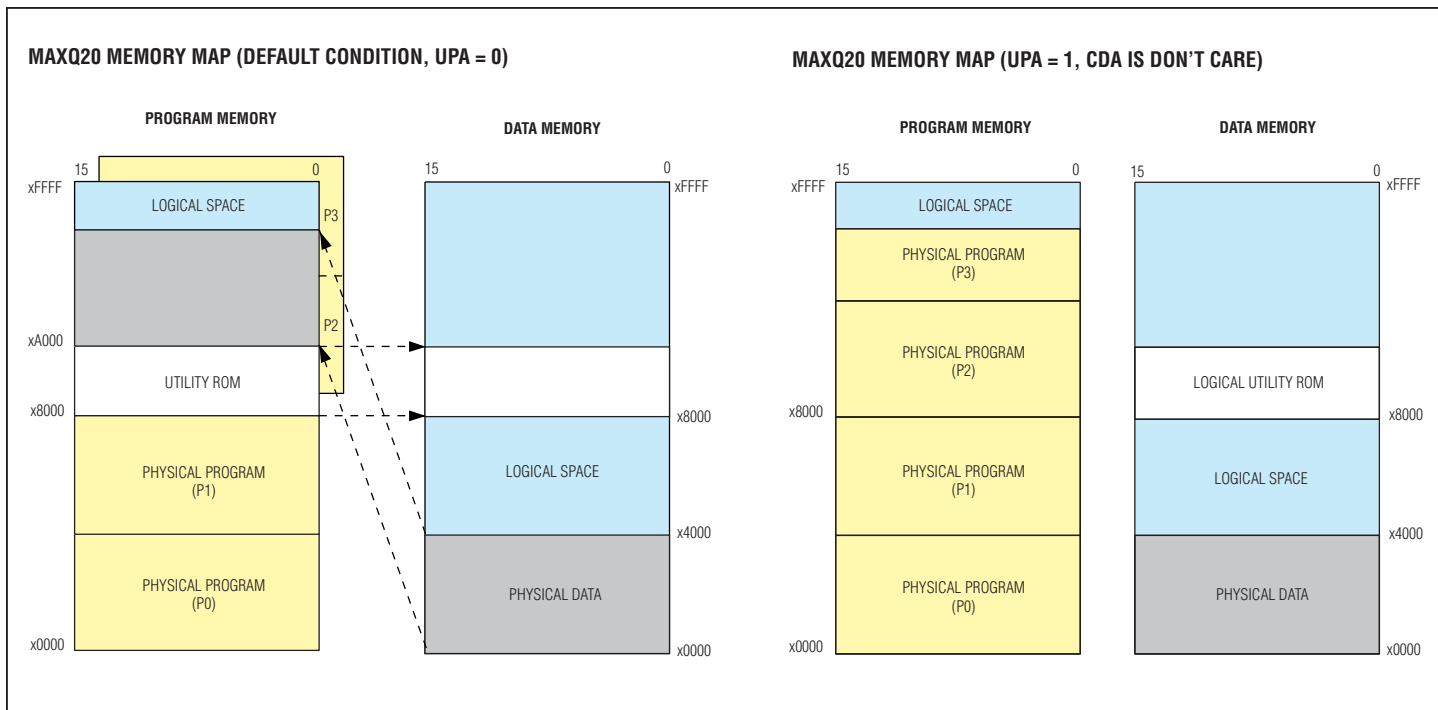


Figure 2-4. Pseudo Von Neumann Memory Map (MAXQ20 Default)

2.5 Pseudo-Von Neumann Memory Access

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

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

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

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

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

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

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

2.6 Data Alignment

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

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

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

2.6.1 Memory Management Unit

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

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

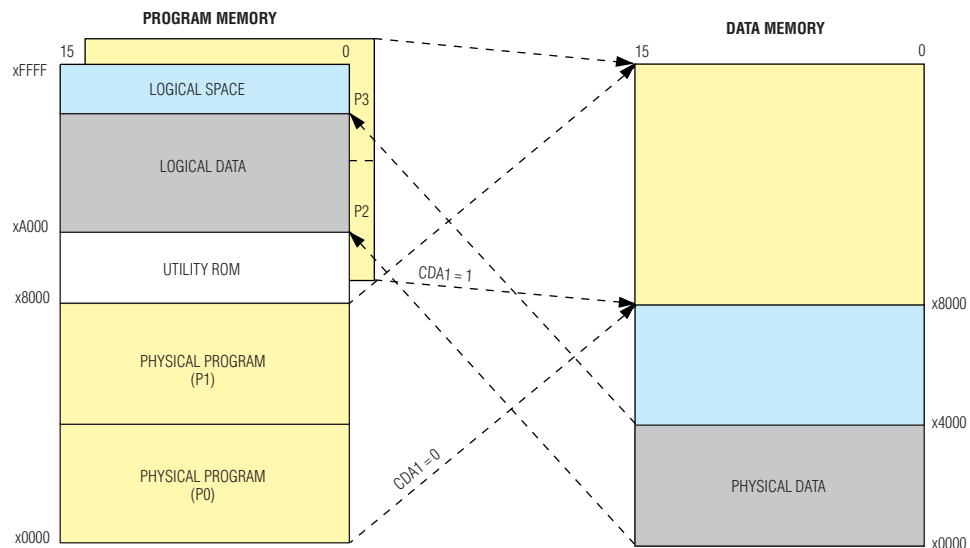
- User program memory up to 32k x 16 (up to 64k x 16 with inclusion of UPA bit).
- Utility ROM up to 8k x 16.
- Data memory SRAM up to 16k x 16.
- In-system and in-application programming of embedded EEPROM, Flash, or SRAM memories.
- Access to any of the three memory areas (SRAM, code memory, utility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

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

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

MAXQ Family User's Guide

MAXQ20 MEMORY MAP (UPA = 0, EXECUTING FROM UTILITY ROM)



MAXQ20 MEMORY MAP (UPA = 0, EXECUTING FROM LOGICAL DATA MEMORY)

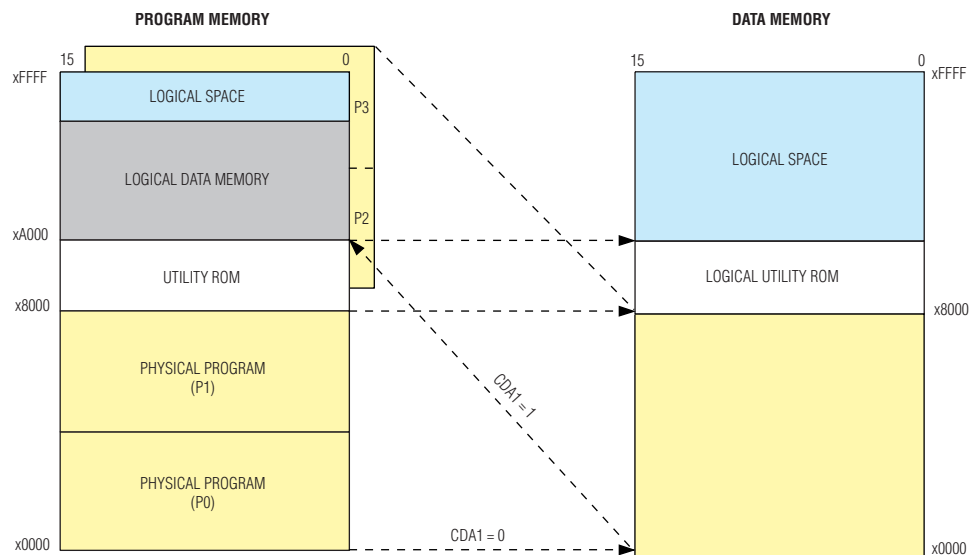
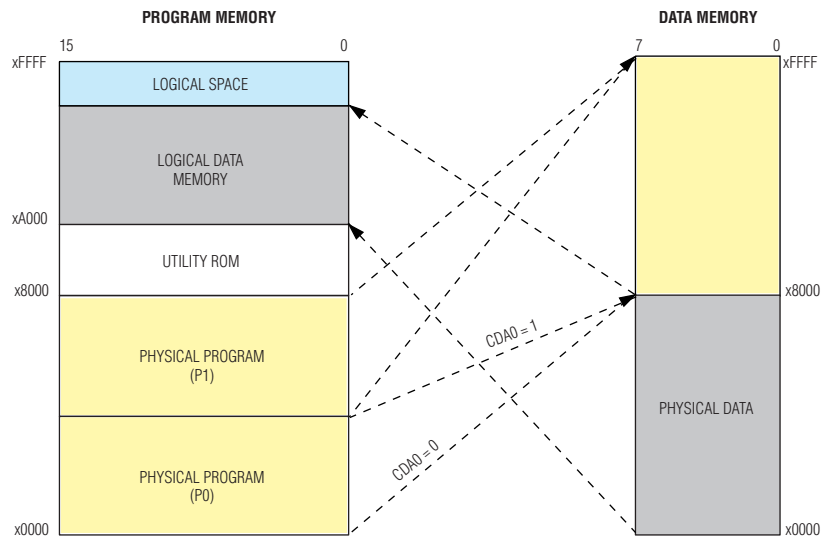


Figure 2-5. CDA Functions (Word Access Mode)

EXECUTING FROM UTILITY ROM (UPA = 0, ONLY P1, P2 PRESENT)



EXECUTING FROM LOGICAL DATA MEMORY (UPA = 0, ONLY P1, P2 PRESENT)

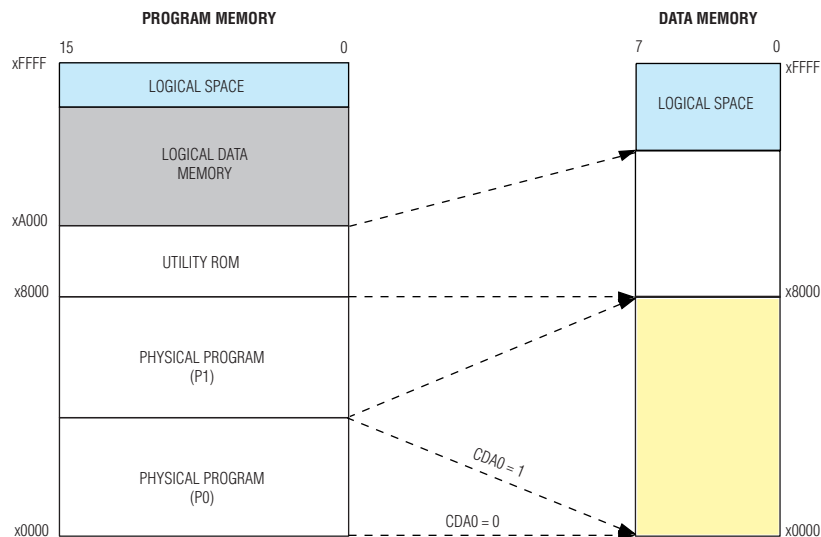


Figure 2-6. CDA Functions (Byte Access Mode)

- ## 2.7 Clock Generation

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



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

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

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

2.7.1 Internal Ring Oscillator

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

2.7.2 External Clock (Crystal/Resonator)

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

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

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

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

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

2.7.3 External Clock (Direct Input)

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

To operate the core from an external clock, connect the clock source to the XTAL1 pin and leave the XTAL2 pin floating. The clock source should be driven through a CMOS driver. If the clock driver is a TTL gate, its output must be connected to VCC through a pullup

MAXQ Family User's Guide

resistor to ensure a satisfactory logic level for active clock pulses. To minimize system noise on the clock circuitry, the external clock source must meet the maximum rise and fall times and the minimum high and low times specified for the clock source. The external noise can affect clock generation circuit if these parameters do not meet the specification.

2.7.4 External RC

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

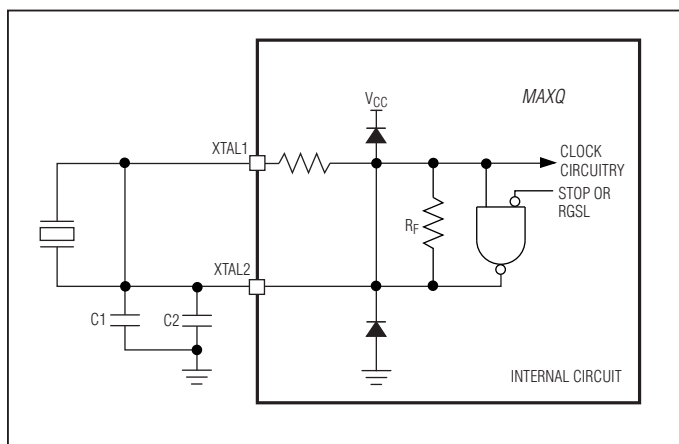


Figure 2-8. On-Chip Crystal Oscillator

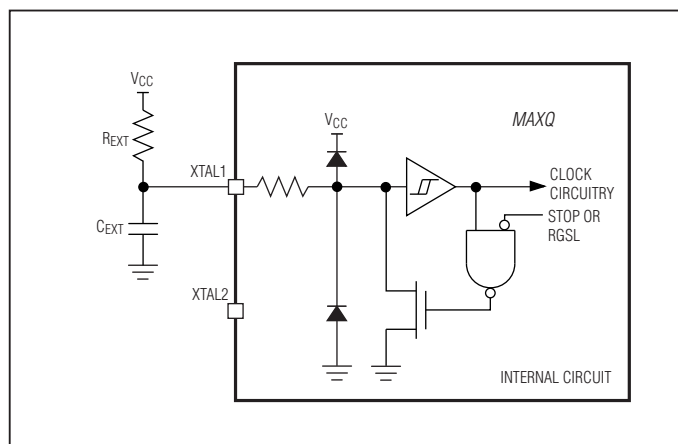


Figure 2-9. RC Relaxation Oscillator

2.7.5 Internal System Clock Generation

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

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

Table 2-2. System Clock Rate Control Settings

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

2.8 Interrupts

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

2.8.1 Servicing Interrupts

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

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

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

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

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

2.8.2 Interrupt System Operation

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

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

- 1) The next instruction fetch from program memory is cancelled.
- 2) The return address is pushed on to the stack.
- 3) The INS bit is set to 1 to prevent recursive interrupt calls.
- 4) The instruction pointer is set to the location of the interrupt service routine (contained in the Interrupt Vector register).
- 5) The CPU begins executing the interrupt service routine.

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

- 1) The return address is popped off the stack.
- 2) The INS bit is cleared to 0 to re-enable interrupt handling.
- 3) The instruction pointer is set to the return address that was popped off the stack.
- 4) The CPU continues execution of the main program.

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

MAXQ Family User's Guide

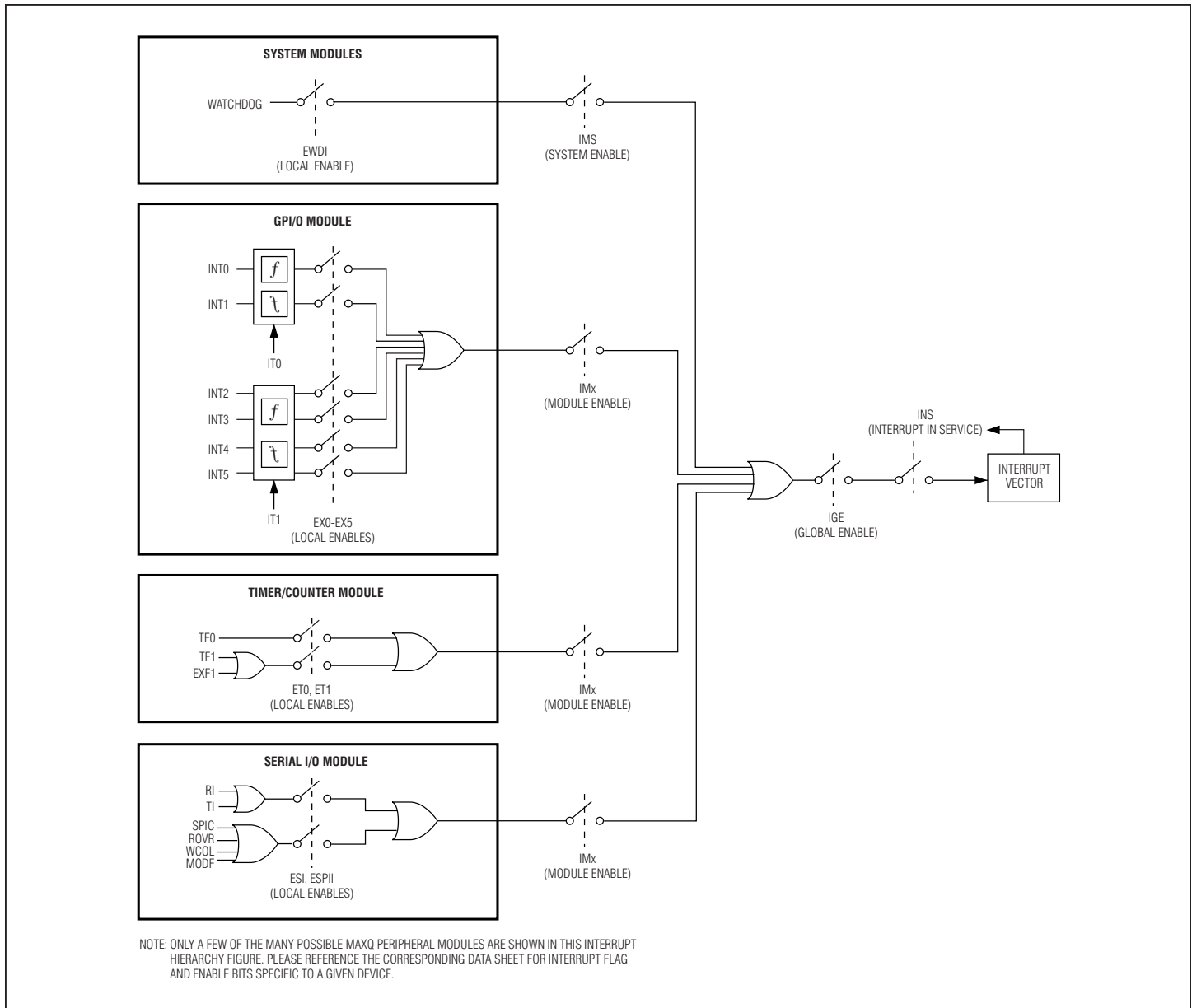


Figure 2-10. MAXQ Interrupt Source Hierarchy Example

2.8.3 Synchronous vs. Asynchronous Interrupt Sources

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

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

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

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

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

2.8.4 Interrupt Prioritization by Software

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

2.8.5 Interrupt Exception Window

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

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

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

2.9 Operating Modes

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

2.9.1 Reset Mode

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

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

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

2.9.1.1 Power-On/Brownout Reset

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

MAXQ Family User's Guide

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

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

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

2.9.1.2 External Reset

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

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

2.9.1.3 Watchdog Timer Reset

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

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

2.9.1.4 Internal System Reset

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

2.9.2 Power Management Mode

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

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

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

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

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

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

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

2.9.2.1 Switchback

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

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

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

2.9.3 Stop Mode

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

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

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

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

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