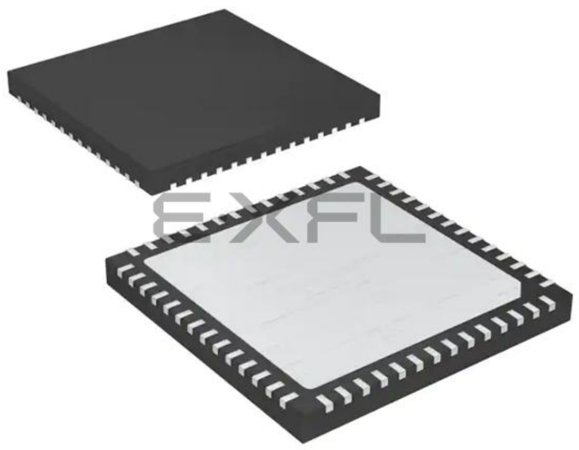

Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

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

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	-
Core Size	16-Bit
Speed	20MHz
Connectivity	1-Wire®, SPI, UART/USART
Peripherals	LCD, POR, PWM, WDT
Number of I/O	50
Program Memory Size	64KB (32K x 16)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	2K x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 2.75V
Data Converters	-
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	56-WFQFN Exposed Pad
Supplier Device Package	56-TQFN (8x8)
Purchase URL	https://www.e-xfl.com/product-detail/analog-devices/maxq2000-qbx

SECTION 2: ARCHITECTURE

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

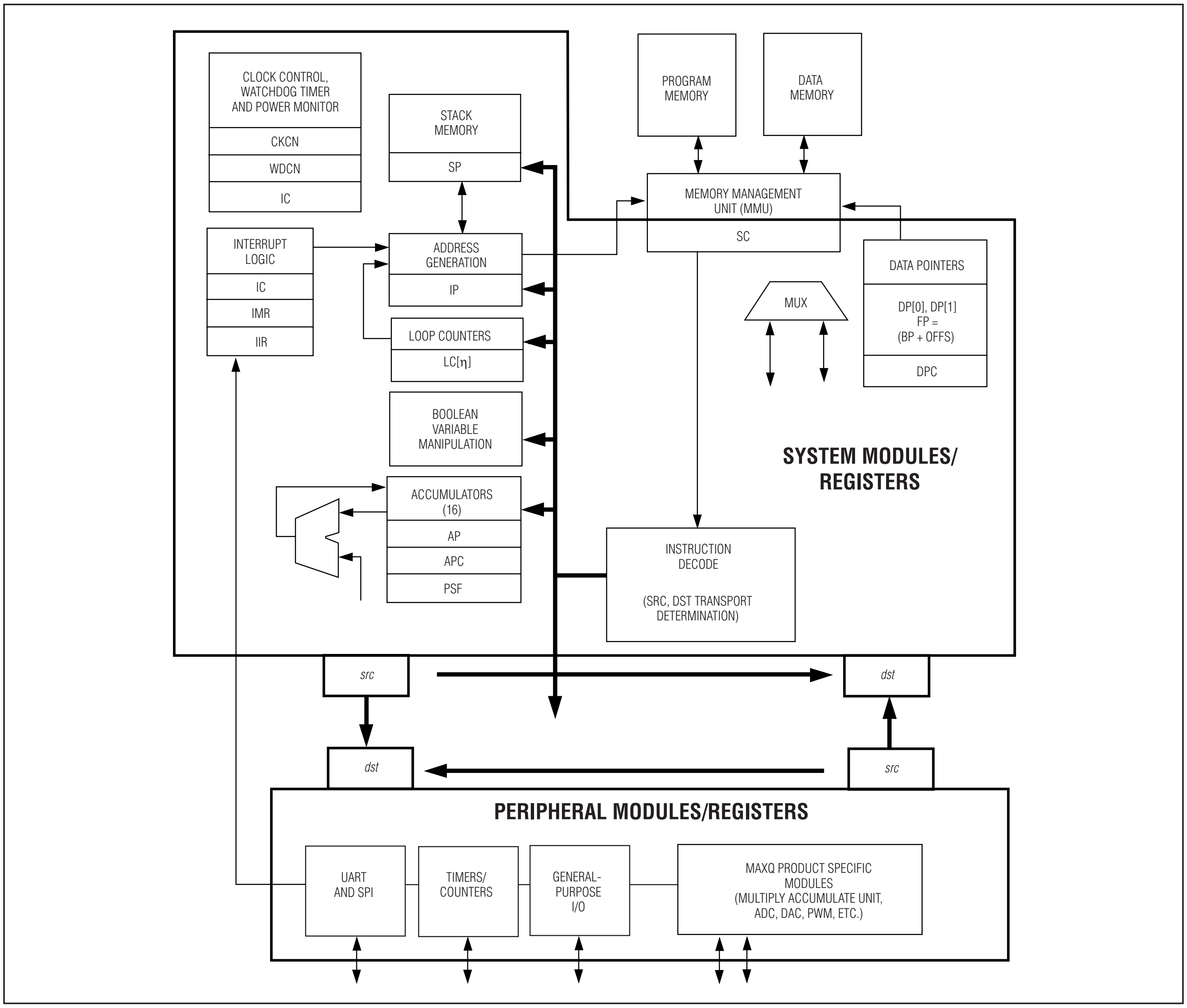
CLOCK CONTROL, WATCHDOG TIMER AND POWER MONITOR
CKCN
WDCN
IC
STACK MEMORY
SP
PROGRAM MEMORY
DATA MEMORY
MEMORY MANAGEMENT UNIT (MMU)
SC
INTERRUPT LOGIC
IC
IMR
IIR
ADDRESS GENERATION
IP
LOOP COUNTERS
LC[η]
DATA POINTERS
DP[0], DP[1]
FP = (BP + OFFS)
DPC
MUX
BOOLEAN VARIABLE MANIPULATION
SYSTEM MODULES/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

Figure 2-1. MAXQ Transport-Triggered Architecture

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

Table 3-3. Watchdog Timeout Period Selection

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

Table 3-4. System Register Map

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

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

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

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

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

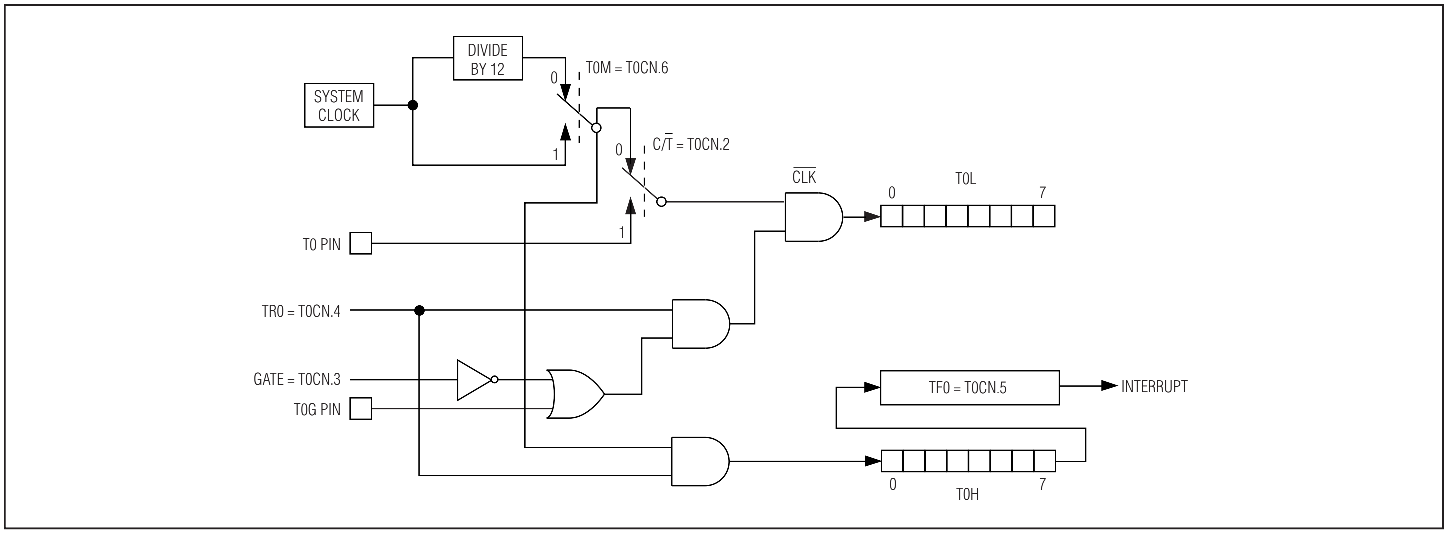

Figure 7-3. Timer/Counter 0 Dual 8-Bit Mode

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

8.2 Timer/Counter 1 Peripheral Registers

8.2.1 Timer/Counter 1 Control Register (T1CN)

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	TF1	EXF1	T1OE	DCEN	EXEN1	TR1	$C/\overline{T1}$	$CP/\overline{RL1}$
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

r = read, w = write

Bit 7: Timer 1 Overflow Flag (TF1). This bit is set when Timer 1 overflows from FFFFh or the count is equal to the capture register in down count mode. It must be cleared by software.

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

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

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

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

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

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

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

8.2.6 Timer/Counter 1 Mode Register (T1MD)

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

r = read, w = write

Bits 7 to 2: Reserved

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

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

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

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

8.3 Time-Base Selection for Timers 0 and 1

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

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

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

9.2.1 16-Bit Timer: Auto-Reload/Compare

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

9.2.1.1 Output Enable (PWM Out)

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

9.2.1.2 Polarity Control

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

9.2.1.3 Gated

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

9.2.1.4 Single Shot (and Gating)

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

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

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

9.4 Timer/Counter 2 Peripheral Registers

9.4.1 Timer/Counter 2 Configuration Register (T2CFG)

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

r = read, w = write

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

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

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

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

0 = 16-bit mode (default)

1 = dual 8-bit mode

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

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

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

0 = timer mode

1 = counter mode

9.4.7 Timer 2 Reload High Register (T2RH)

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

r = read, w = write

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

9.4.8 Timer 2 Capture/Compare Register (T2C)

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

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

r = read, w = write

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

9.4.9 Timer 2 Capture/Compare High Register (T2CH)

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

r = read, w = write

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

10.1.2 UART Mode 1

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

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

RI must = 0, and either

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

or

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

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

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

10.1.3 UART Mode 2

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

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

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

RI must = 0, and either

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

or

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

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

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

SECTION 11: SERIAL PERIPHERAL INTERFACE (SPI) MODULE

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

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

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

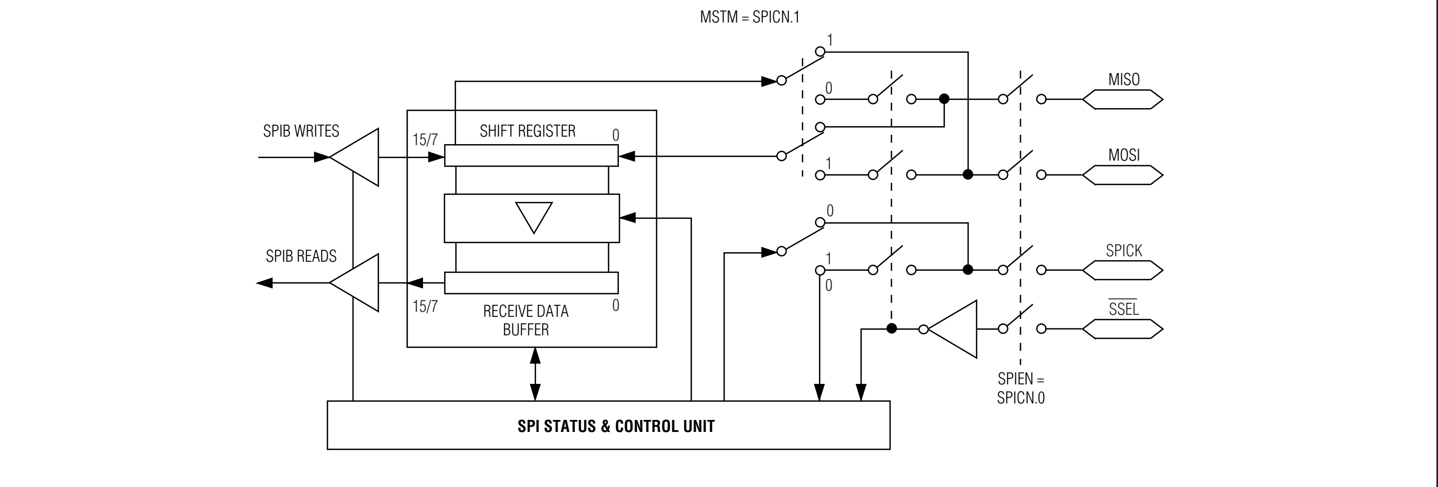

Figure 11-1. SPI Block Diagram

12.6 Hardware Multiplier Examples

The following are code examples of multiplier operations.

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

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

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

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

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

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

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

SECTION 13: 1-Wire BUS MASTER

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

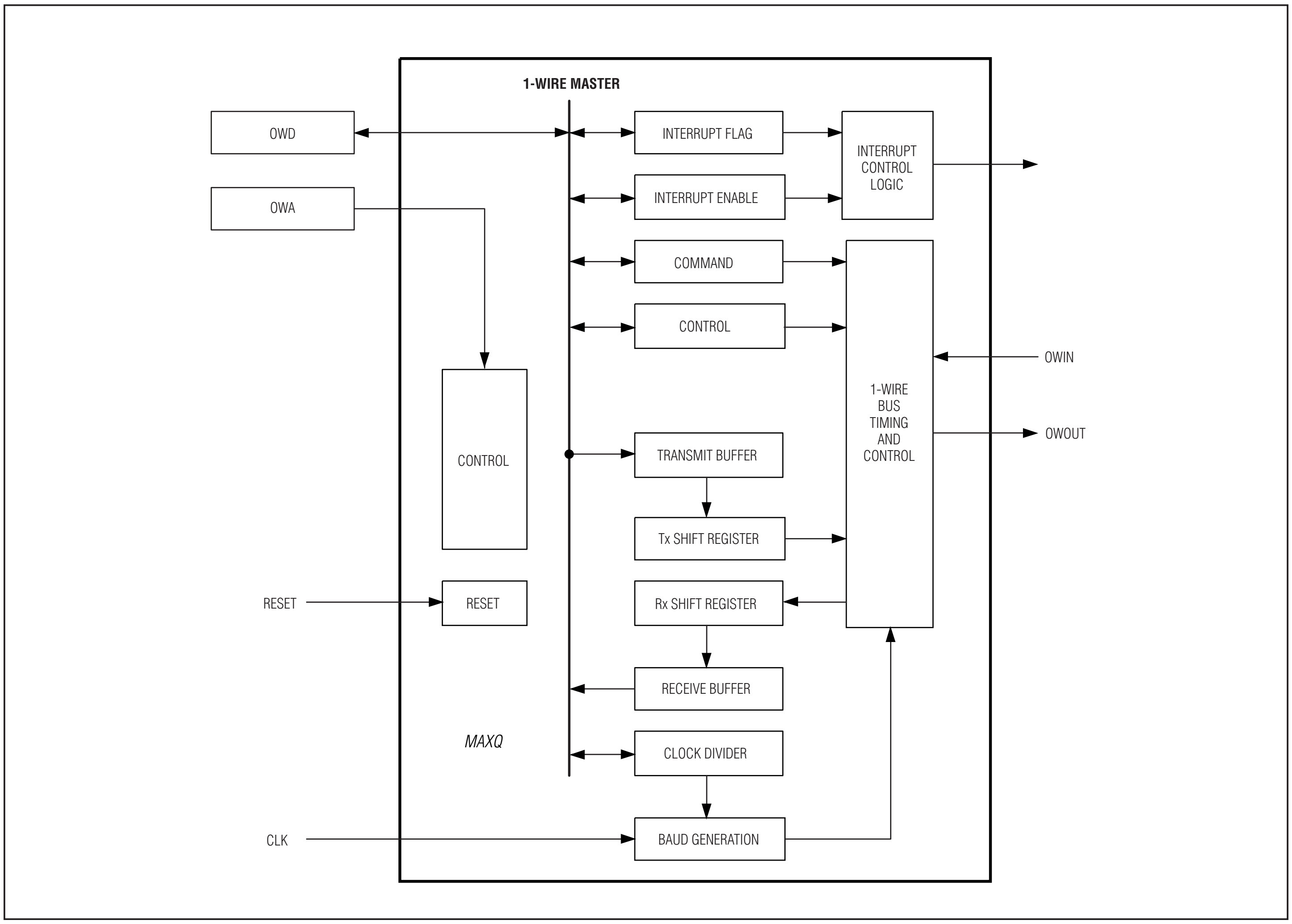

Figure 13-1. 1-Wire Bus Master Functional Diagram

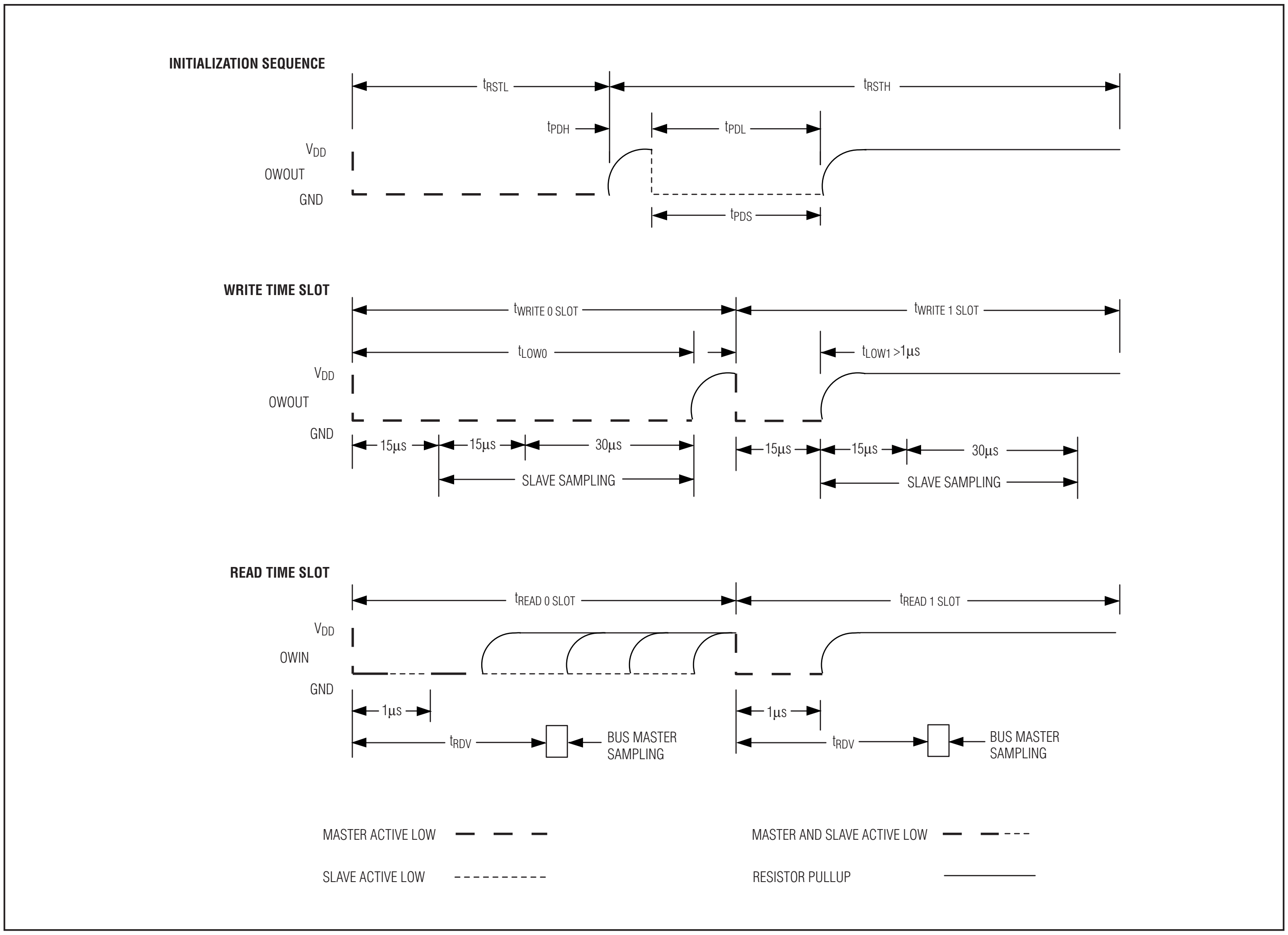

Figure 13-2. 1-Wire Bus Signaling in Standard Mode

SECTION 14: REAL-TIME CLOCK MODULE

The real-time clock (RTC) is a binary timer that keeps the time of day and provides time-of-day and sub-second alarm functionality in the form of system interrupts. The RTC consists of cascaded 32-bit and 8-bit ripple counters that respectively represent absolute seconds (~136 years) and sub-seconds (in 1/256 second resolution). The 8-bit sub-second counter increments with each 256Hz clock tick derived from the 32.768kHz oscillator. A separate auto-reload sub-second alarm counter can be used to generate interval alarms with granularity of 1/256 seconds. The 32-bit seconds counter increments with each rollover of the 8-bit sub-second counter. The 32-bit counter can be used with the programmable time-of-day comparison alarm to provide a single event timer. The RTC must be stopped for the counter registers to initially be written, but once enabled, the RTC counts continuously as long as it is enabled and does not stop for reads of the counter registers. The RTC also supports a digital trim facility for those applications requiring high accuracy.

User-application code accesses the RTC via eight peripheral registers. The 16-bit RTC Control register (RCNT) provides the control and status for RTC functions, including RTC read/write access controls, clock selection/control, RTC enable, square-wave output enable, alarm enables, and their interrupt flags. The seconds count information is set or initialized only by writing to the 16-bit RTC Second High and the 16-bit RTC Second Low register (RTSH and RTSL registers). Similarly, the 8-bit RTC Sub-Second register (RTSS) can be used to establish and access the sub-second counter. The Time-of-Day alarm is composed of the RASH:RASL register pair and the interval alarm is programmable via the RSSA register. The digital trim value is held in the RTRM register.

Figure 14-1 shows a functional diagram of a full-featured RTC. Certain MAXQ microcontroller devices may be equipped with a subset of the RTC functionality described. Refer to the individual device data sheet or supplemental user guide for more information.

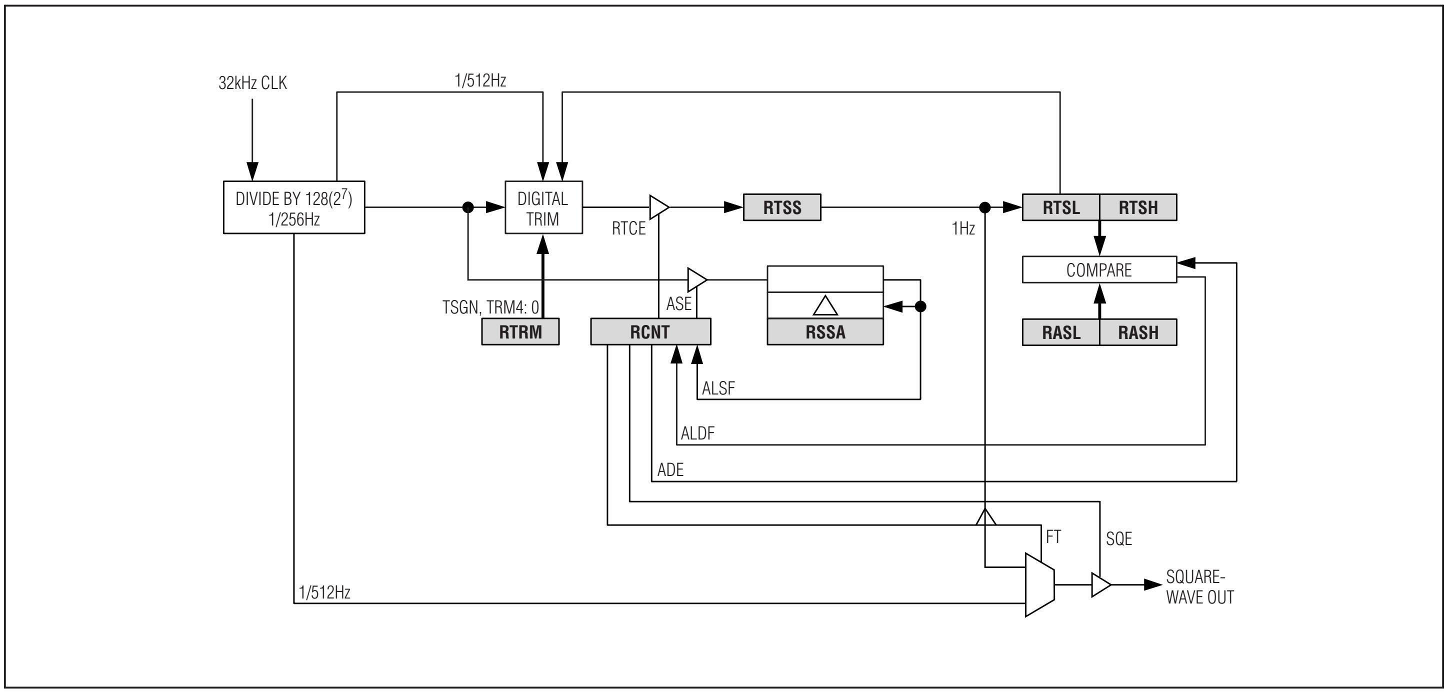

Figure 14-1. RTC Functional Block Diagram

Table 16-1 shows the background mode commands supported by the MAXQ microcontroller. Encodings not listed in this table are not supported in background mode and are treated as no operations.

Table 16-1. Background Mode Commands

OP CODE	COMMAND	OPERATION
0000-0000	No Operation	**No Operation.** Default state for Debug Shift register.
0000-0001	Read ICDC	**Read Control Data from the ICDC.** The contents of the ICDC register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires one follow-on transfer cycle.
0000-0010	Read ICDF	**Read Flags from the ICDF.** The contents of the ICDF register (one byte) are loaded into the Debug Shift Register via the ICDB register for host read. This command requires one follow-on transfer cycle.
0000-0011	Read ICDA	**Read Data from the ICDA.** The contents of the ICDA register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0100	Read ICDD	**Read Data from the ICDD.** The contents of the ICDD register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0101	Read BP0	**Read Data from the BP0.** The contents of the BP0 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0110	Read BP1	**Read Data from the BP1.** The contents of the BP1 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0111	Read BP2	**Read Data from the BP2.** The contents of the BP2 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1000	Read BP3	**Read Data from the BP3.** The contents of the BP3 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1001	Read BP4	**Read Data from the BP4.** The contents of the BP4 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1010	Read BP5	**Read Data from the BP5.** The contents of the BP5 register are loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0001-0001	Write ICDC	**Write Control Data to the ICDC.** The contents of ICDB are loaded into the ICDC register by the debug engine at the end of the data transfer cycle.
0001-0011	Write ICDA	**Write Data to the ICDA.** The contents of ICDB are loaded into the ICDA register by the debug engine at the end of the data transfer cycles. Data is transferred with the least significant byte first.
0001-0100	Write ICDD	**Write Data to the ICDD.** The contents of ICDB are loaded into the ICDD register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0101	Write BP0	**Write Data to the BP0.** The contents of ICDB are loaded into the BP0 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0110	Write BP1	**Write Data to the BP1.** The contents of ICDB are loaded into the BP1 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0111	Write BP2	**Write Data to the BP2.** The contents of ICDB are loaded into the BP2 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1000	Write BP3	**Write Data to the BP3.** The contents of ICDB are loaded into the BP3 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1001	Write BP4	**Write Data to the BP4.** The contents of ICDB are loaded into the BP4 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1010	Write BP5	**Write Data to the BP5.** The contents of ICDB are loaded into the BP5 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1111	Debug	**Debug Command.** This command forces the debug engine into debug mode and halts the CPU operation at the completion of the current instruction after the debug engine recognizes the debug command.

16.2.2 Read Register Map Command Host-ROM Interaction

A read register map command reads out data contents for all implemented system and peripheral registers. The host does not specify a target register but instead should expect register data output in successive order, starting with the lowest order register in register module 0. Data is loaded by the ROM to the 8-bit ICDB register and is output one byte per transfer cycle. Thus, for a 16-bit register, two transfer cycles are necessary. The host initiates each transfer cycle to shift out the data bytes and will find valid data output tagged with a debug-valid (status = 11b). At the end of each transfer cycle, the debug engine clears the TXC flag to signal the ROM service routine that another byte may be loaded to ICDB. The ROM service routine sets the TXC flag each time after loading data to the ICDB register. This process is repeated until all registers have been read and output to the host. The host system recognizes the completion of the register read when the status debug-idle is presented. This indicates that the debug engine is ready for another operation.

16.2.3 Single-Step Operation (Trace)

The debug engine supports single step operation in debug mode by executing a trace command from the host. The debug engine allows the CPU to return to its normal program execution for one cycle and then forces a debug mode re-entry:

1) Set status to 10b (debug-busy).
2) Pop the return address from the stack.
3) Set the IGE bit to logic 1 if debug mode was activated when IGE = 1.
4) Supply the CPU with an instruction addressed by the return address.
5) Stall the CPU at the end of the instruction execution.
6) Block the next instruction fetch from program memory.
7) Push the return address onto the stack.
8) Set the contents of IP to x8010h.
9) Clear the IGE bit to 0 to disable the interrupt handler.
10) Halt CPU operation.
11) Set the status to debug-idle.

Note that the trace operation uses a return address from the stack as a legitimate address for program fetching. The host must maintain consistency of program flow during the debug process. The Instruction Pointer is automatically incremented after each trace operation, thus a new return address will be pushed onto the stack before returning the control to the debug engine. Also, note that the interrupt handler is an essential part of the CPU and a pending interrupt could be granted during single step operation since the IGE bit state present on debug mode entry is restored for the single step.

16.2.4 Return

To terminate the debug mode and return the debug engine to background mode, the host must issue a Return command to the debug engine. This command causes the following actions:

1) Pop the return address from the stack.
2) Set the IGE bit to logic 1 if debug mode was activated when IGE = 1.
3) Supply the CPU with an instruction addressed by the return address.
4) Allow the CPU to execute the normal user program.
5) Set the status to 00b (non-debug).

To prevent a possible endless-breakpoint matching loop, no break occurs for a breakpoint match on the first instruction after returning from debug mode to background mode. Returning to background mode also enables all internal timer functions.

16.2.5 Debug Mode Special Considerations

The following are special considerations when using Debug Mode.

- The debug engine cannot be operated reliably when the CPU is configured in the Power Management Mode (divide-by-256 system clock mode). To allow for proper execution of debug mode commands when invoked during PMM, the Switchback enable (SWB) bit should be configured to a logic 1. With SWB = 1, entering active debug mode (whether by breakpoint match or issuance of the debug command) forces a switchback to the divide-by-1 system clock mode and allow the debug engine to function correctly. This allows user code to configure breakpoints that occur inside PMM, thus providing reliable use of debug commands. However, it does not allow a good means for re-entering PMM.

- Special caution should be exercised when using the Write Register command on register bits that globally affect system operation (e.g., IGE, STOP). If the write register command is used to invoke stop mode (setting STOP = 1), the $\overline{\text{RST}}$ pin may be asserted to reset the debug engine and return to the background mode of operation.
- Single stepping ('Trace') through any IGE bit change operation results in the debug engine overriding the bit change since it retains the IGE bit setting captured when active debug mode was entered.
- Single stepping ('Trace') into an operation that sets STOP = 1 when IGE = 1 effectively allows enabled interrupts normally capable of causing exit from stop mode to do so.
- Single stepping ('Trace') through any memory read instruction that reads from the utility ROM (such as 'move Acc,' @DP[0] with DP[0] set to 8000h) will cause the memory read to return an incorrect value.
- Single stepping ('Trace') cannot be used when executing code from the utility ROM.
- Data memory allocation is important during system development if in-circuit debug is planned. The top 32-byte memory location may be used by the debug service routine during debug mode. The data contents in these locations may be altered and cannot be recovered.
- One available stack location is needed for debug mode. If the stack is full when entering debug mode, the oldest data in the stack will be overwritten.
- The crystal warmup counter is the only counter not disabled when active debug mode is entered. If the crystal warmup counter completes while in active debug mode, a glitchless switch will be made to selected clock source (which was being counted). It is important that the user recognize that this action will occur since the TAP clock should be run no faster than 1/8 the system clock frequency.
- Any signal sampling that relies upon the internal system clock (e.g., counter inputs) can be unreliable since the system clock is turned off inside active debug mode between debug mode commands.
- Power Management Mode cannot be invoked in the first instruction executed when returning from active debug mode. The PMME bit will not be set if such an attempt is made.

16.3 In-Circuit Debug Peripheral Registers

16.3.1 In-Circuit Debug Temp 0 Register (ICDT0)

Bit #	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Name	ICDT0.15	ICDT0.14	ICDT0.13	ICDT0.12	ICDT0.11	ICDT0.10	ICDT0.9	ICDT0.8
Reset	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	ICDT0.7	ICDT0.6	ICDT0.5	ICDT0.4	ICDT0.3	ICDT0.2	ICDT0.1	ICDT0.0
Reset	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s

s = special

Bits 15 to 0: In-Circuit Debug Temp 0 (ICDT0.[15:0]). This register is read/write accessible by the CPU only in background mode or debug mode. This register is intended for use by the utility ROM routines as temporary storage to save registers that might otherwise have to be placed in the stack.

SECTION 17: IN-SYSTEM PROGRAMMING (JTAG)

Internal nonvolatile memory of MAXQ microcontrollers can be initialized via Bootstrap Loader mode. To enable the Bootstrap loader and establish a desired communication channel, the System Programming instruction (100b) must be loaded into the TAP instruction register using the IR-Scan sequence. Once the instruction is latched in the instruction parallel buffer (IR2:0) and is recognized by the TAP controller in the Update-IR state, a 3-bit data shift register is activated as the communication channel for DR-Scan sequences. The TAP retains the System Programming instruction until a new instruction is shifted in or the TAP controller returns to the Test-Logic-Reset state. This 3-bit shift register formed between the TDI and TDO pins is directly interfaced to the 3-bit Serial Programming Buffer (SPB). The System Programming Buffer (SPB) contains three bits with the following functions:

- SPB.0: System Programming Enable (SPE). Setting this bit to logic 1 denotes that system programming is desired upon exiting reset. When it is cleared to logic 0, no system programming is needed. The logic state of SPE is examined by the reset vector in the Utility ROM to determine the program flow after a reset. When SPE = 1, the Bootstrap Loader selected by the PSS1:0 bits are activated to perform a Bootstrap Loader function. When SPE = 0, the Utility ROM transfers execution control to the normal user program.
- SPB.2:1: Programming Source Select (PSS1:PSS0). These bits allow the host to select programming interface sources. The PSS bits have no functions when the SPE bit is cleared.

PSS1	PSS0	PROGRAMMING SOURCE
0	0	JTAG
0	1	UART
1	0	SPI
1	1	Reserved

The DR-Scan sequence is used to configure the SPB bits. The data content of the SPB register is reflected in the ICDF register and allows read/write access by the CPU. These bits are cleared by power-on reset or Test-Logic-Reset of the TAP controller.

17.1 JTAG Bootloader Operation

Devices that support a JTAG bootloader have the benefit of using the same status bit handshaking hardware as is used for in-circuit debugging. When the SPE bit of the System Programming Buffer (SPB) is set to 1 and JTAG is selected as the programming source (PSS1:0 = 00b), the background and active debug mode state machines are disabled. Once the host loads the Debug instruction into the TAP instruction register (IR2:0), the 10-bit shift register interface to ICDB and the status bits becomes available for host-to-ROM bootloader communication. The status bits should be interpreted as follows for JTAG bootloader operation:

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

When the using the JTAG bootloader option (SPE = 1, PSS1:0 = 00b), the sole purpose of the debug hardware is to simultaneously transfer the data byte shifted in from the host into the ICDB register and transfer the contents of an internal holding register (loaded by ROM code writes of ICDB) into the shift register for output to the host. This transfer takes place on the falling edge of TCK at the Update-DR state. The debug hardware additionally clears the TXC bit at this point in the state diagram. The ROM loader code controls the status bit output to the host by asserting TXC = 1 when it has valid data to be shifted out. The ROM code may flexibly implement whatever communication protocol and command set it wishes within the data byte portion of the shifted 10-bit word.

LIST OF TABLES