PIC24FV32KA304 FAMILY

SERIAL

PIC24F Device	Pins	Memory			Timers 16-Bit	Capture Input	Compare/PWM Output	UART w/ IrDA®	SPI	I²C™	12-Bit A/D (ch)	Comparators	CTMU (ch)	RTCC
		Flash Program (bytes)	SRAM (bytes)	EE Data (bytes)										
PIC24FV16KA301 /PIC24F16KA301	20	16K	2K	512	5	3	3	2	2	2	12	3	12	Y
PIC24FV32KA301 /PIC24F32KA301	20	32K	2K	512	5	3	3	2	2	2	12	3	12	Y
PIC24FV16KA302 /PIC24F16KA302	28	16K	2K	512	5	3	3	2	2	2	13	3	13	Y
PIC24FV32KA302 /PIC24F32KA302	28	32K	2K	512	5	3	3	2	2	2	13	3	13	Y
PIC24FV16KA304 /PIC24F16KA304	44	16K	2K	512	5	3	3	2	2	2	16	3	16	Y
PIC24FV32KA304 /PIC24F32KA304	44	32K	2K	512	5	3	3	2	2	2	16	3	16	Y

SERIAL

SERIAL

PIC24FV32KA304 FAMILY

20/28/44/48-Pin, General Purpose, 16-Bit Flash Microcontrollers with XLP Technology

Power Management Modes:

- Run – CPU, Flash, SRAM and Peripherals On
- Doze – CPU Clock Runs Slower than Peripherals
- Idle – CPU Off, Flash, SRAM and Peripherals On
- Sleep – CPU, Flash and Peripherals Off and SRAM on
- Deep Sleep – CPU, Flash, SRAM and Most Peripherals Off; Multiple Autonomous Wake-up Sources
- Low-Power Consumption:
  - Run mode currents down to 8 µA, typical
  - Idle mode currents down to 2.2 µA, typical
  - Deep Sleep mode currents down to 20 nA, typical
  - Real-Time Clock/Calendar currents down to 700 nA, 32 kHz, 1.8V
  - Watchdog Timer 500 nA, 1.8V typical

High-Performance CPU:

- Modified Harvard Architecture
- Up to 16 MIPS Operation @ 32 MHz
- 8 MHz Internal Oscillator with 4x PLL Option and Multiple Divide Options
- 17-Bit by 17-Bit Single-Cycle Hardware Multiplier
- 32-Bit by 16-Bit Hardware Divider 16-Bit x 16-Bit Working Register Array
- C Compiler Optimized Instruction Set Architecture

Peripheral Features:

- Hardware Real-Time Clock and Calendar (RTCC):
  - Provides clock, calendar and alarm functions
  - Can run in Deep Sleep mode
  - Can use 50/60 Hz power line input as clock source
- Programmable 32-bit Cyclic Redundancy Check (CRC)
- Multiple Serial Communication modules:
  - Two 3-/4-wire SPI modules
  - Two I^2C™ modules with multi-master/slave support
  - Two UART modules supporting RS-485, RS-232, LIN/J2602, IrDA®
- Five 16-Bit Timers/Counters with Programmable Prescaler:
  - Can be paired as 32-bit timers/counters
- Three 16-Bit Capture Inputs with Dedicated Timers
- Three 16-Bit Compare/PWM Output with Dedicated Timers
- Configurable Open-Drain Outputs on Digital I/O Pins
- Up to Three External Interrupt Sources

Analog Features:

- 12-Bit, up to 16-Channel Analog-to-Digital Converter:
  - 100 ksps conversion rate
  - Conversion available during Sleep and Idle
  - Auto-sampling timer-based option for Sleep and Idle modes
  - Wake on auto-compare option
- Dual Rail-to-Rail Analog Comparators with Programmable Input/Output Configuration
- On-Chip Voltage Reference
- Internal Temperature Sensor
- Charge Time Measurement Unit (CTMU):
  - Used for capacitance sensing, 16 channels
  - Time measurement, down to 200 ps resolution
  - Delay/pulse generation, down to 1 ns resolution

Special Microcontroller Features:

- Wide Operating Voltage Range:
  - 1.8V to 3.6V (PIC24F devices)
  - 2.0V to 5.5V (PIC24FV devices)
- Low Power Wake-up Sources and Supervisors:
  - Ultra-Low Power Wake-up (ULPWU) for Sleep/Deep Sleep
  - Low-Power Watchdog Timer (DSWDT) for Deep Sleep
  - Extreme Low-Power Brown-out Reset (DSBOR) for Deep Sleep, LPBOR for all other modes
- System Frequency Range Declaration bits:
  - Declaring the frequency range optimizes the current consumption.
- Standard Watchdog Timer (WDT) with On-Chip, Low-Power RC Oscillator for Reliable Operation
- Programmable High/Low-Voltage Detect (HLVD)
- Standard Brown-out Reset (BOR) with 3 Programmable Trip Points that can be Disabled in Sleep
- High-Current Sink/Source (18 mA/18 mA) on All I/O Pins
- Flash Program Memory:
  - Erase/write cycles: 10,000 minimum
  - 40 years' data retention minimum
- Data EEPROM:
  - Erase/write cycles: 100,000 minimum
  - 40 years' data retention minimum
- Fail-Safe Clock Monitor
- Programmable Reference Clock Output
- Self-Programmable under Software Control
- In-Circuit Serial Programming™ (ICSP™) and In-Circuit Debug (ICD) via 2 Pins

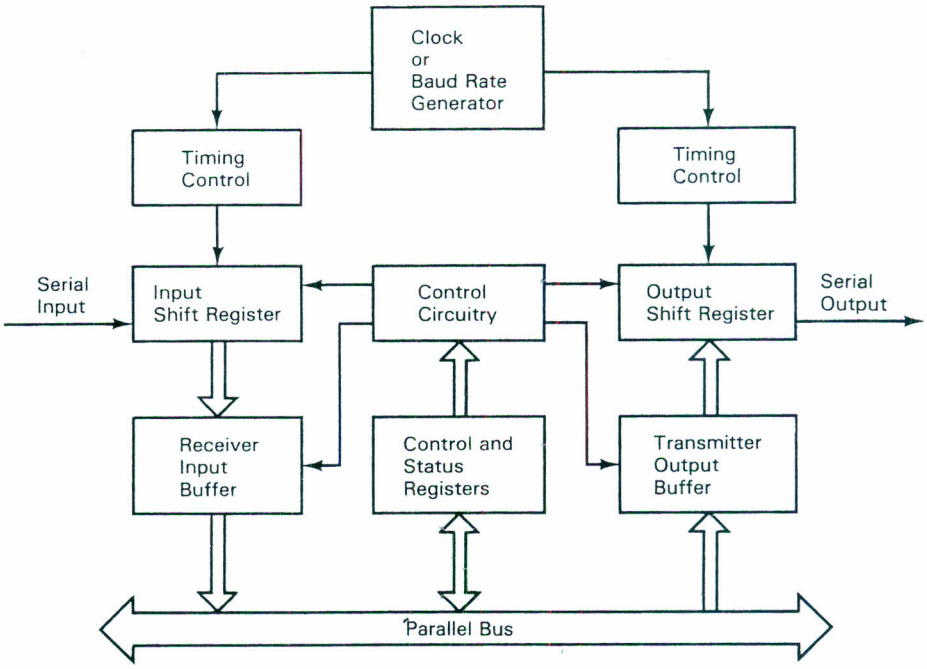


Figure 12.7 Serial data communication circuitry.

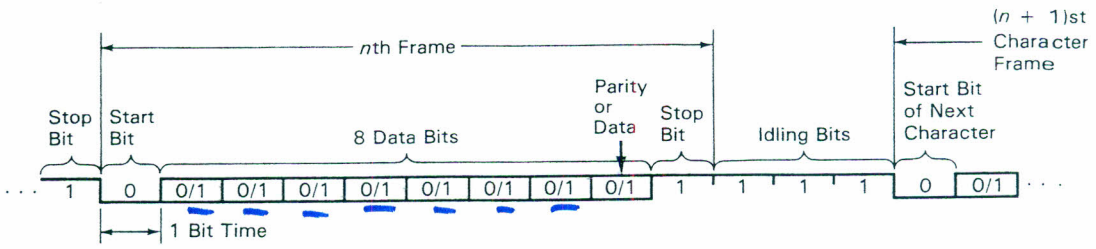
Asynchronous and synchronous communication. In *asynchronous communication*, information is transmitted as individual data items bracketed by a start bit and either one or two stop bits. A typical example is shown in Figure 12.8(a) for the transmission of 8 data bits bracketed by one start bit and one stop bit. This format could be used to transmit an ASCII character plus a parity bit.[3] The complete sequence of bits is called a *frame*.

Some interfaces, such as the SCI of the Queued Serial Module, allow a ninth bit to be added to the data frame. The format for each frame with the added bit is shown in Figure 12.8(b). The extra bit is used for communication between master and slave devices as will be described in Section 12.3.

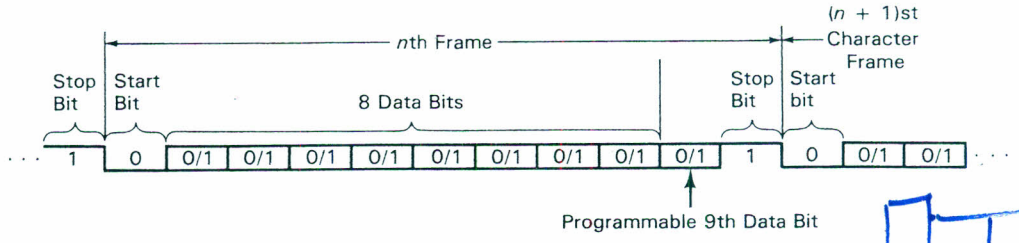
The time between frames is called the *idle time* and it may vary from frame to frame. Figure 12.8(a) shows three idle bits after the n^{th} frame. No idle bits between frames are shown in Figure 12.8(b). A frame is detected by the receiving device when the signal voltage drops from HIGH (logic {1}) to LOW (logic {0}) as the start bit is recognized. This synchronizes the frame between receiver and transmitter. However, the transmission method is called *asynchronous* because the receiver and transmitter have separate clocks which are not synchronized in time. As described in Example 12.2 and Example 12.3, the clocks determine the bit transmission rate (baud rate).

After the start bit is recognized, the receiver must determine the location of each bit boundary and sample the value. The receiver does this by measuring the voltage of the input signal a fixed number of times during the period of each bit. The bit is stored as a {0} or a {1} according to the results of the measurement. Subsection 12.3.1

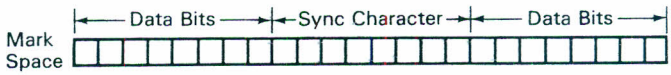
[3] Section 8.3 presented a program example to add a parity bit to an ASCII character.



(a) Asynchronous, 8-bit format



(b) Asynchronous, 9-bit format



(c) Synchronous

Figure 12.8 Asynchronous and synchronous frame formats.

explains the relationship between the receiver clock frequency and the baud rate for the SCI.

The asynchronous method is often used for data transmission between a computer and a low-speed peripheral device such as an operator's terminal or printer. Since start and stop bits are needed in every frame, the method is not generally as efficient for the transfer of information as synchronous transfers.

In *synchronous communication* information is transmitted in blocks. A possible format is shown in Figure 12.8(c). The receiver and transmitter clocks are synchronized in time because a clock signal is transmitted along with the data. The clock signal can be sent on a separate conductor or it may be encoded with the data in a self-clocking scheme. In either case, the start and stop bits of asynchronous transfer are eliminated. There is no idle time between blocks, but when no data are being sent a "sync" (synchronizing) character is transmitted. The receiver's clock is thus always synchronized with the transmitter's clock. Very high transmission rates can be achieved reliably as compared to asynchronous transfer.

Signaling Characteristics. Once the type of format for serial data transmissions is selected, the signaling characteristics must be defined.[4] Two important

[4] Encoding techniques and other characteristics of the signal are discussed in several of the references cited in the Further Reading section of this chapter. One common encoding technique for asynchronous communication is termed non-return-to-zero (NRZ).

parameters are the bit rate measured in bits per second and the digital encoding technique.

Since the bit is the smallest unit of information for serial communications, the signaling speed is defined in terms of bits per second transmitted. This is frequently called the *baud rate* when only two-level signaling is used.[5] The baud rate is the reciprocal of the length (in seconds) of the shortest element in the signaling code. If more than two voltage levels are used in transmission, the bit rate is higher than the baud rate.

Example 12.2

The baud rate for two-level signaling is the reciprocal of the time duration of a bit. The frequency of the transmitter and receiver clocks determine the baud rate for an asynchronous communications channel. The clocks must not differ in frequency by more than several percent or transmission errors may result. At 9600 baud for example, each bit has a duration of 1/9600 seconds or about 0.104 milliseconds. A frame of 10 bits thus requires about a millisecond to be received or transmitted. This rate would allow the transmission of 1000 ASCII characters per second.

In applications using the SCI for interrupt controlled transfers, the CPU would be interrupted after each character is received. The interrupt handling routine would read the character from the SCI receiver's input register and store it in memory. For transmission, the CPU is interrupted each time the transmitter is ready for a new character. The interrupt handling routine must transfer each character from one of the CPU registers or memory to the transmitter's output register for transmission.

Since the parallel registers of the SCI hold only one character, the CPU is involved in each transfer. However, at the transfer rates typically used by the SCI, only a small percentage of the CPU's time is spent servicing the interrupts. The QSPI can provide even more efficient operation since it has a buffer area in the form of a queue that stores up to 16 data values for transmission and reception. It is straightforward to calculate the percentage of time used by the CPU for transfers when the transfer rate and the duration of the interrupt handling routine is known. Several of the exercises at the end of Section 12.2 require this calculation.

The baud rate for the SCI and QSPI is derived from the system clock. The rate is programmable for the SCI between 64 baud and 524,000 baud with a 16.78MHz system clock. A signaling rate of between 33kHz and 4.19MHz is possible for the QSPI. The baud rate of the SCI and the signaling rate of the QSPI correspond to the number of bits per second that are transmitted or received.

QSM electrical characteristics. The input and output pins of the SCI and QSPI operate with a logic HIGH of +5 volts and a logic LOW of 0 volts when the power supply voltage is +5 volts. These voltage levels are appropriate for data transmission between microcontrollers. Signal lines from one SCI serial port, for example, can be connected directly to another SCI port without any intervening circuitry. The connection is described in Section 12.3.

When the MC68332 SCI is used as a UART (Universal Asynchronous Receiver Transmitter) port, the voltage levels from the SCI must be translated to those of the

[5] Baud is a contraction of the surname of J.M.F. Baudot according to the *Encyclopedia of Computer Science*, Van Nostrand Reinhold Company, 1976. Baudot invented a French telegraph code adopted in 1877.

Table 12-4 Summary of RS-232-C, RS-423, RS-422, and RS-485 Standards

Specification	RS-232-C	RS-423	RS-422	RS-485
Receiver input voltage	±3 to ±15 V	±200 mV to ±12 V	±200 mV to ±7 V	±200 mV to –7 to +12 V
Driver output signal	±5 to ±15 V	±3.6 to ±6 V	±2 to ±5 V	±1.5 to ±5 V
Maximum data rate	20 Kbit/s	100 Kbit/s	10 Mbit/s	10 Mbit/s
Maximum cable length	50 feet	4000 feet	4000 feet	4000 feet
Driver source impedance	3–7 kΩ	450 Ω min	100 Ω	54 Ω
Receiver input resistance	3 kΩ	4 kΩ	4 kΩ min	12 kΩ
Mode	Single-ended	Single-ended	Differential	Differential
Number of drivers and receivers allowed on one line	1 driver 1 receiver	1 driver 10 receivers	1 driver 10 receivers	32 drivers 32 receivers

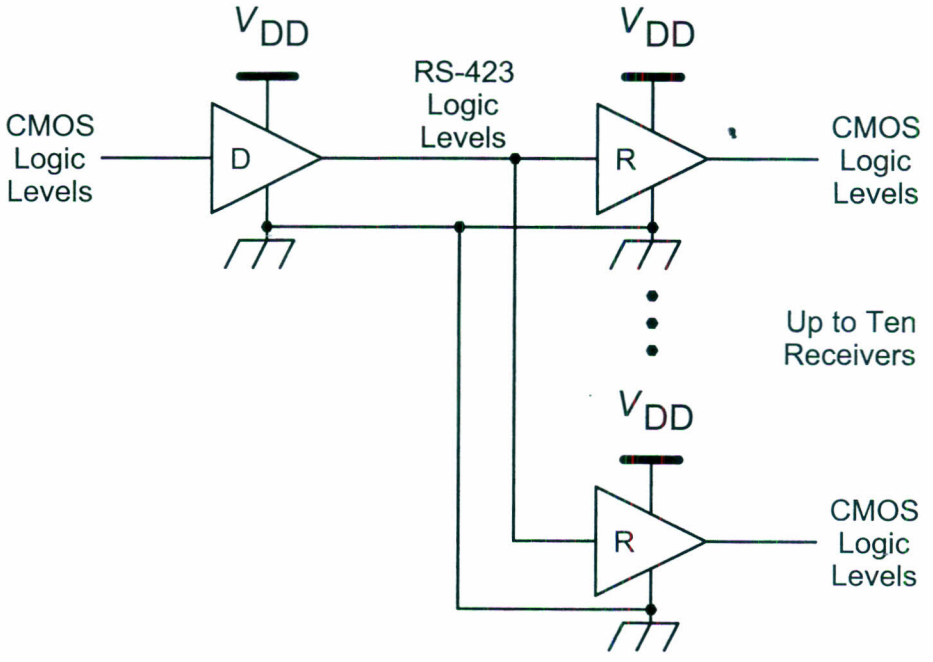


Figure 12-11 RS-423 interface.

RS-422 Standard

A problem experienced with the single-ended drivers and receivers of RS-232-C and RS-423 is that for long line lengths, noise and ground shifts can cause errors in the received data. Noise and ground shifts appear as common-mode signals; that is, they affect each line equally. The RS-422 line drivers and receivers operate with differential amplifiers as shown in Figure 12-12. These drivers eliminate much of the common-mode noise experienced with long transmission lines. Their source and load impedances match twisted-pair transmission lines[5]; the line lengths and data rates that can be achieved will be shown in Table 12-4, along with the RS-422 electrical specifications.

[5] Approximately 100 Ω.

Table 12-5 ASCII 7-bit Codes for Alphanumeric Characters

	MS Digit							
LS Digit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Example 12-1 Finding the ASCII Code for a Character

Use Table 12-5 to find the hexadecimal ASCII codes for the characters A, a, and].

Solution

A = 0x41, a = 0x61,] = 0x5D

Example 12-2 Finding the ASCII Code for a Character

Use Table 12-5 to find the hexadecimal ASCII codes for the control characters CR, BEL, and LF.

Solution

CR = 0x0D, BEL = 0x07, LF = 0x0A

18.0 UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER (UART)

Note: This data sheet summarizes the features of this group of PIC24F devices. It is not intended to be a comprehensive reference source. For more information on the Universal Asynchronous Receiver Transmitter, refer to the *"PIC24F Family Reference Manual"*, **Section 21. "UART"** (DS39708).

The Universal Asynchronous Receiver Transmitter (UART) module is one of the serial I/O modules available in this PIC24F device family. The UART is a full-duplex asynchronous system that can communicate with peripheral devices, such as personal computers, LIN/J2602, RS-232 and RS-485 interfaces. This module also supports a hardware flow control option with the $\overline{\text{UxCTS}}$ and $\overline{\text{UxRTS}}$ pins, and also includes an IrDA® encoder and decoder.

The primary features of the UART module are:

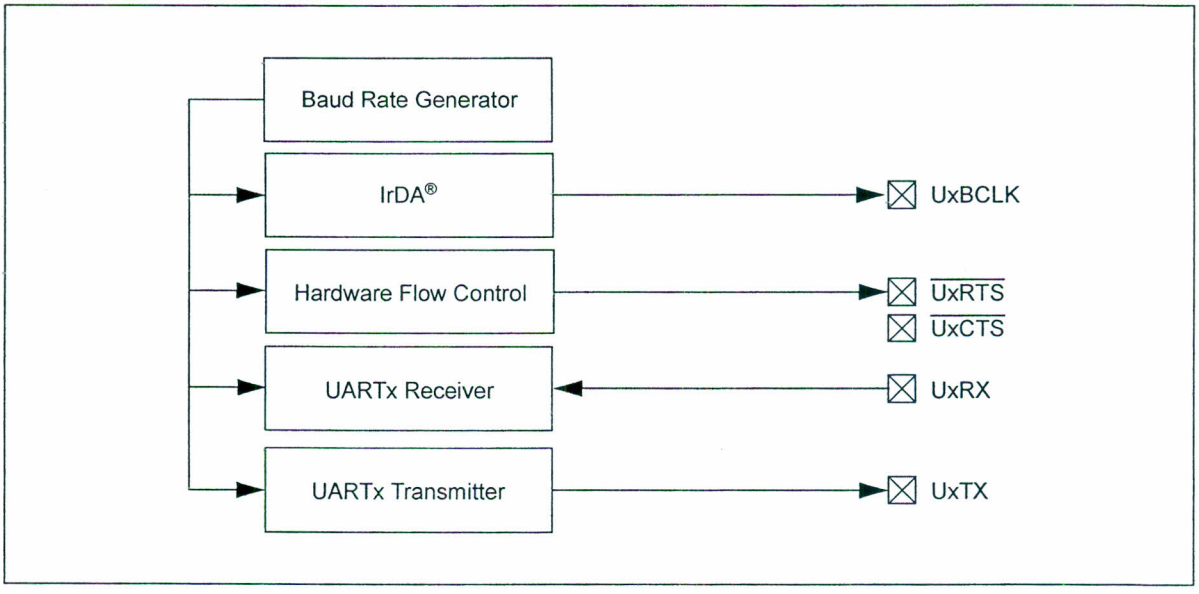
- Full-Duplex, 8-Bit or 9-Bit Data Transmission through the UxTX and UxRX Pins
- Even, Odd or No Parity Options (for 8-bit data)
- One or Two Stop bits
- Hardware Flow Control Option with $\overline{\text{UxCTS}}$ and $\overline{\text{UxRTS}}$ pins
- Fully Integrated Baud Rate Generator (IBRG) with 16-bit Prescaler
- Baud Rates Ranging from 1 Mbps to 15 bps at 16 MIPS
- 4-Deep, First-In-First-Out (FIFO) Transmit Data Buffer
- 4-Deep FIFO Receive Data Buffer
- Parity, Framing and Buffer Overrun Error Detection
- Support for 9-bit mode with Address Detect (9^{th} bit = 1)
- Transmit and Receive Interrupts
- Loopback mode for Diagnostic Support
- Support for Sync and Break Characters
- Supports Automatic Baud Rate Detection
- IrDA® Encoder and Decoder Logic
- 16x Baud Clock Output for IrDA Support

A simplified block diagram of the UART is shown in Figure 18-1. The UART module consists of these important hardware elements:

- Baud Rate Generator
- Asynchronous Transmitter
- Asynchronous Receiver

THE UART IS SOMETIMES KNOWN AS THE SCI!

FIGURE 18-1: UART SIMPLIFIED BLOCK DIAGRAM



18.1 UART Baud Rate Generator (BRG)

The UART module includes a dedicated 16-bit Baud Rate Generator (BRG). The UxBRG register controls the period of a free-running, 16-bit timer. Equation 18-1 provides the formula for computation of the baud rate with BRGH = 0.

EQUATION 18-1: UART BAUD RATE WITH BRGH = 0(1)

$$\text{Baud Rate} = \frac{FCY}{16 \cdot (UxBRG + 1)}$$

$$UxBRG = \frac{FCY}{16 \cdot \text{Baud Rate}} - 1$$

Note 1: Based on FCY = FOSC/2; Doze mode and PLL are disabled.

Example 18-1 provides the calculation of the baud rate error for the following conditions:

- FCY = 4 MHz
- Desired Baud Rate = 9600

The maximum baud rate (BRGH = 0) possible is FCY/16 (for UxBRG = 0) and the minimum baud rate possible is FCY/(16 * 65536).

Equation 18-2 shows the formula for computation of the baud rate with BRGH = 1.

EQUATION 18-2: UART BAUD RATE WITH BRGH = 1(1)

$$\text{Baud Rate} = \frac{FCY}{4 \cdot (UxBRG + 1)}$$

$$UxBRG = \frac{FCY}{4 \cdot \text{Baud Rate}} - 1$$

Note 1: Based on FCY = FOSC/2; Doze mode and PLL are disabled.

The maximum baud rate (BRGH = 1) possible is FCY/4 (for UxBRG = 0) and the minimum baud rate possible is FCY/(4 * 65536).

Writing a new value to the UxBRG register causes the BRG timer to be reset (cleared). This ensures the BRG does not wait for a timer overflow before generating the new baud rate.

EXAMPLE 18-1: BAUD RATE ERROR CALCULATION (BRGH = 0)(1)

Desired Baud Rate = FCY/(16 (UxBRG + 1))

Solving for UxBRG value:

UxBRG = ((FCY/Desired Baud Rate)/16) – 1
UxBRG = ((4000000/9600)/16) – 1
UxBRG = 25

Calculated Baud Rate = 4000000/(16 (25 + 1))
= 9615

Error = (Calculated Baud Rate – Desired Baud Rate) / Desired Baud Rate
= (9615 – 9600)/9600
= 0.16%

Note 1: Based on FCY = FOSC/2; Doze mode and PLL are disabled.

18.2 Transmitting in 8-Bit Data Mode

1. Set up the UART:
   a) Write appropriate values for data, parity and Stop bits.
   b) Write appropriate baud rate value to the UxBRG register.
   c) Set up transmit and receive interrupt enable and priority bits.
2. Enable the UART.
3. Set the UTXEN bit (causes a transmit interrupt two cycles after being set).
4. Write data byte to lower byte of UxTXREG word. The value will be immediately transferred to the Transmit Shift Register (TSR), and the serial bit stream will start shifting out with the next rising edge of the baud clock.
5. Alternately, the data byte may be transferred while UTXEN = 0, and then, the user may set UTXEN. This will cause the serial bit stream to begin immediately, because the baud clock will start from a cleared state.
6. A transmit interrupt will be generated as per interrupt control bit, UTXISELx.

18.3 Transmitting in 9-Bit Data Mode

1. Set up the UART (as described in **Section 18.2 "Transmitting in 8-Bit Data Mode"**).
2. Enable the UART.
3. Set the UTXEN bit (causes a transmit interrupt, two cycles after being set).
4. Write UxTXREG as a 16-bit value only.
5. A word write to UxTXREG triggers the transfer of the 9-bit data to the TSR. The serial bit stream will start shifting out with the first rising edge of the baud clock.
6. A transmit interrupt will be generated as per the setting of control bit, UTXISELx.

18.4 Break and Sync Transmit Sequence

The following sequence will send a message frame header made up of a Break, followed by an auto-baud Sync byte.

1. Configure the UART for the desired mode.
2. Set UTXEN and UTXBRK – sets up the Break character.
3. Load the UxTXREG with a dummy character to initiate transmission (value is ignored).
4. Write '55h' to UxTXREG – loads the Sync character into the transmit FIFO.
5. After the Break has been sent, the UTXBRK bit is reset by hardware. The Sync character now transmits.

18.5 Receiving in 8-Bit or 9-Bit Data Mode

1. Set up the UART (as described in **Section 18.2 "Transmitting in 8-Bit Data Mode"**).
2. Enable the UART.
3. A receive interrupt will be generated when one or more data characters have been received as per interrupt control bit, URXISELx.
4. Read the OERR bit to determine if an overrun error has occurred. The OERR bit must be reset in software.
5. Read UxRXREG.

The act of reading the UxRXREG character will move the next character to the top of the receive FIFO, including a new set of PERR and FERR values.

18.6 Operation of $\overline{\text{UxCTS}}$ and $\overline{\text{UxRTS}}$ Control Pins

UARTx Clear to Send ($\overline{\text{UxCTS}}$) and Request to Send ($\overline{\text{UxRTS}}$) are the two hardware-controlled pins that are associated with the UART module. These two pins allow the UART to operate in Simplex and Flow Control modes. They are implemented to control the transmission and reception between the Data Terminal Equipment (DTE). The UEN<1:0> bits in the UxMODE register configure these pins.

18.7 Infrared Support

The UART module provides two types of infrared UART support: one is the IrDA clock output to support an external IrDA encoder and decoder device (legacy module support), and the other is the full implementation of the IrDA encoder and decoder.

As the IrDA modes require a 16x baud clock, they will only work when the BRGH bit (UxMODE<3>) is '0'.

18.7.1 EXTERNAL IrDA SUPPORT – IrDA CLOCK OUTPUT

To support external IrDA encoder and decoder devices, the UxBCLK pin (same as the $\overline{\text{UxRTS}}$ pin) can be configured to generate the 16x baud clock. When UEN<1:0> = 11, the UxBCLK pin will output the 16x baud clock if the UART module is enabled; it can be used to support the IrDA codec chip.

18.7.2 BUILT-IN IrDA ENCODER AND DECODER

The UART has full implementation of the IrDA encoder and decoder as part of the UART module. The built-in IrDA encoder and decoder functionality is enabled using the IREN bit (UxMODE<12>). When enabled (IREN = 1), the receive pin (UxRX) acts as the input from the infrared receiver. The transmit pin (UxTX) acts as the output to the infrared transmitter.

21.3.2 Baud Rate Tables

UART baud rates are provided in Table 21-1 and Table 21-2 for common device instruction cycle frequencies (FCY). The minimum and maximum baud rates for each frequency are also shown.

Table 21-1: UART Baud Rates (BRGH = 0)

BAUD RATE	FCY = 16 MHz			FCY = 12 MHz		
	Actual Baud Rate	% Error	BRG Value (Decimal)	Actual Baud Rate	% Error	BRG Value (Decimal)
110	110.0	0.00	9090	110.0	0.00	6817
300	300.0	0.01	3332	300.0	0.00	2499
1200	1200.5	0.04	832	1200.0	0.00	624
2400	2398.1	-0.08	416	2403.8	0.16	311
9600	9615.4	0.16	103	9615.3	0.16	77
19.2K	19230.8	0.16	51	19230.7	0.15	38
38.4K	38461.5	0.16	25	37500.0	-2.34	19
56K	55555.6	-0.79	17	57692.3	-3.02	12
115K	111111.1	-3.38	8			
250K	250000.0	0.00	3			
300K						
500K	500000.0	0.00	1			
Min.	15.0	0.00	65535	11.0	0.00	65535
Max.	1000000.0	0.00	0	480000.0	0.00	0

BAUD RATE	FCY = 8 MHz			FCY = 4 MHz			FCY = 1 MHz		
	Actual Baud Rate	% Error	BRG Value (Decimal)	Actual Baud Rate	% Error	BRG Value (Decimal)	Actual Baud Rate	% Error	BRG Value (Decimal)
110	917.4	0.00	4544	110.0	0.00	2272	110.0	0.00	567
300	299.9	0.00	1666	300.1	0.00	832	300.4	0.10	207
1200	1199.0	0.00	416	1201.9	0.16	207	1201.9	0.16	51
2400	2403.8	0.16	207	2403.8	0.15	103	2403.8	0.15	25
9600	9615.4	0.16	51	9615.4	0.20	25			
19.2K	19230.8	0.16	25	19230.8	0.20	12			
38.4K	38461.5	0.16	12						
56K	55555.6	-0.79	8						
115K									
250K									
300K									
500K									
Min.	8.0	0.00	65535	4.0	0.00	65535	0.95	0.00	65535
Max.	500000.0	0.00	0	250000.0	0.00	0	62500.0	0.00	0

SEE REESE P382

16.0 SERIAL PERIPHERAL INTERFACE (SPI)

Note: This data sheet summarizes the features of this group of PIC24F devices. It is not intended to be a comprehensive reference source. For more information on the Serial Peripheral Interface, refer to the *"PIC24F Family Reference Manual"*, **Section 23. "Serial Peripheral Interface (SPI)"** (DS39699).

The Serial Peripheral Interface (SPI) module is a synchronous serial interface useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial data EEPROMs, shift registers, display drivers, A/D Converters, etc. The SPI module is compatible with Motorola® SPI and SIOP interfaces.

The module supports operation in two buffer modes. In Standard mode, data is shifted through a single serial buffer. In Enhanced Buffer mode, data is shifted through an 8-level FIFO buffer.

Note: Do not perform read-modify-write operations (such as bit-oriented instructions) on the SPI1BUF register in either Standard or Enhanced Buffer mode.

The module also supports a basic framed SPI protocol while operating in either Master or Slave mode. A total of four framed SPI configurations are supported.

The SPI serial interface consists of four pins:

- SDI1: Serial Data Input
- SDO1: Serial Data Output
- SCK1: Shift Clock Input or Output
- $\overline{SS1}$: Active-Low Slave Select or Frame Synchronization I/O Pulse

The SPI module can be configured to operate using 2, 3 or 4 pins. In the 3-pin mode, $\overline{SS1}$ is not used. In the 2-pin mode, both SDO1 and $\overline{SS1}$ are not used.

Block diagrams of the module in Standard and Enhanced Buffer modes are shown in Figure 16-1 and Figure 16-2.

The devices of the PIC24FV32KA304 family offer two SPI modules on a device.

Note: In this section, the SPI modules are referred to as SPIx. Special Function Registers (SFRs) will follow a similar notation. For example, SPI1CON1 or SPI1CON2 refers to the control register for the SPI1 module.

To set up the SPI1 module for the Standard Master mode of operation:

1. If using interrupts:
   a) Clear the respective SPI1IF bit in the IFS0 register.
   b) Set the respective SPI1IE bit in the IEC0 register.
   c) Write the respective SPI1IPx bits in the IPC2 register to set the interrupt priority.
2. Write the desired settings to the SPI1CON1 and SPI1CON2 registers with the MSTEN bit (SPI1CON1<5>) = 1.
3. Clear the SPIROV bit (SPI1STAT<6>).
4. Enable SPI operation by setting the SPIEN bit (SPI1STAT<15>).
5. Write the data to be transmitted to the SPI1BUF register. Transmission (and reception) will start as soon as data is written to the SPI1BUF register.

To set up the SPI module for the Standard Slave mode of operation:

1. Clear the SPI1BUF register.
2. If using interrupts:
   a) Clear the respective SPI1IF bit in the IFS0 register.
   b) Set the respective SPI1IE bit in the IEC0 register.
   c) Write the respective SPI1IP bits in the IPC2 register to set the interrupt priority.
3. Write the desired settings to the SPI1CON1 and SPI1CON2 registers with the MSTEN bit (SPI1CON1<5>) = 0.
4. Clear the SMP bit.
5. If the CKE bit is set, then the SSEN bit (SPI1CON1<7>) must be set to enable the $\overline{SS1}$ pin.
6. Clear the SPIROV bit (SPI1STAT<6>).
7. Enable SPI operation by setting the SPIEN bit (SPI1STAT<15>).

ever, the received data represent the latest values input. For example, if a number of A/D converters are supplying values to the queue entries, the CPU program can read the latest values at any time. In effect, the peripheral devices with serial interfaces connected to the QSPI appear to transfer parallel data to the CPU program. The CPU program simply reads the appropriate queue entry to acquire a data value. No conflict will occur between serial data acquisition and the CPU reads because of the dual-access capability of the receiver RAM.

SPI

12.4.3 I/O Expansion with the QSPI and Chip Selects

In many applications, the most limited resource of a microcontroller is the number of I/O signal-lines available. One approach to I/O expansion is to use serial transfers rather than parallel I/O ports. The QSPI, with very few additional circuit chips, can control up to sixteen external devices using synchronous, serial transfers. A block diagram of a possible system is shown in Figure 12.26. The signal lines MOSI and MISO are used for data transfers. Selection of a specific peripheral unit and the timing of the transfers is controlled by the peripheral chip select and the serial clock SCK signal.

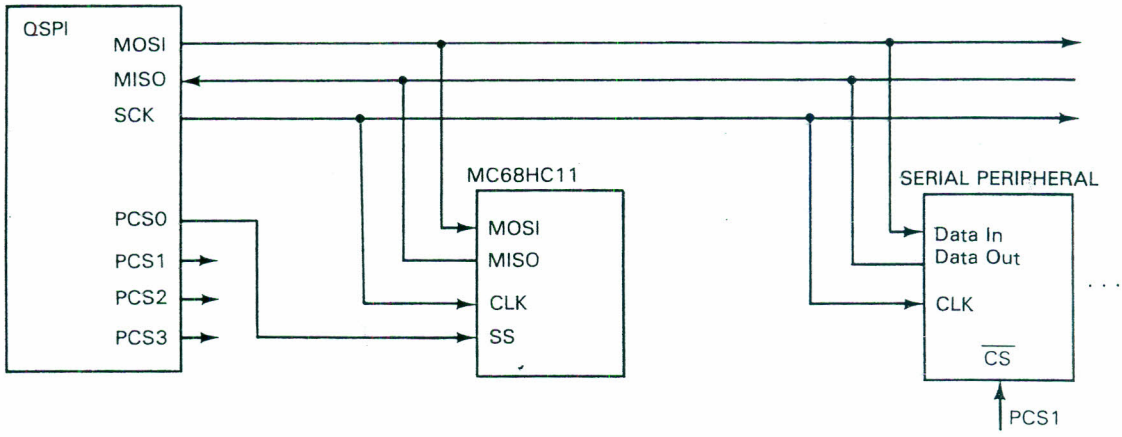
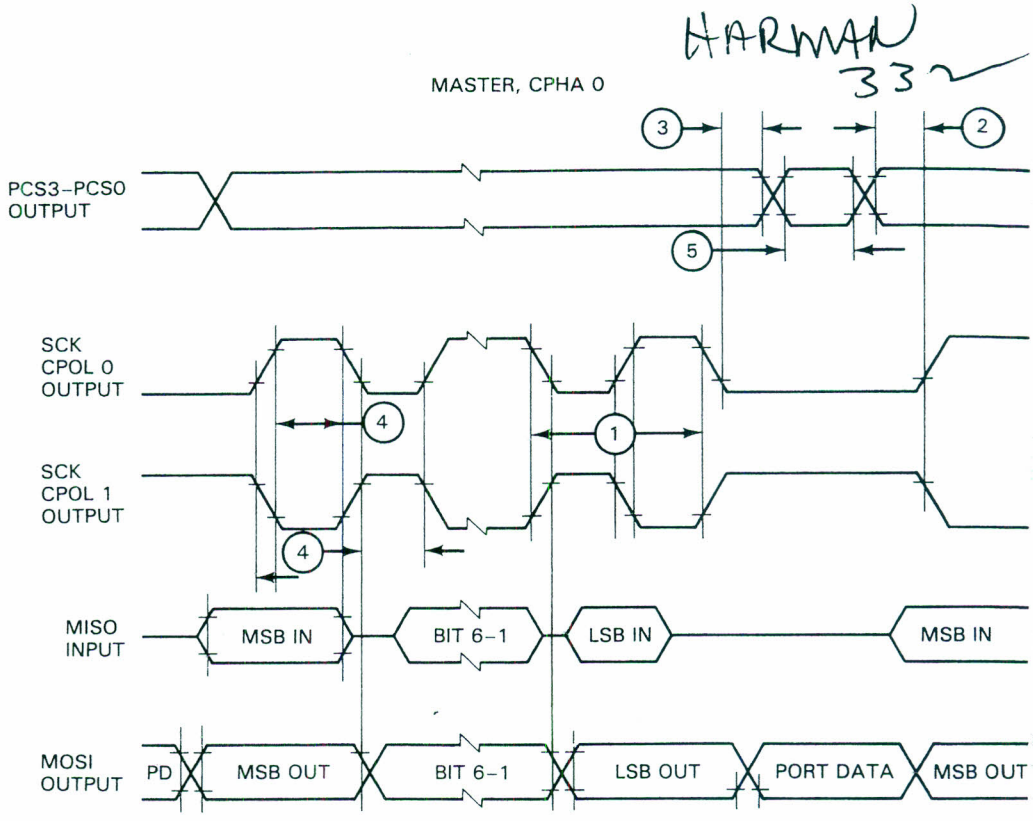


Figure 12.26 Block diagram of a serial interface system.

As drawn, the system of Figure 12.26 would allow up to four peripheral units to be attached to the QSPI. Sixteen peripherals could be attached by decoding the 4 peripheral chip select signals properly. In the figure, the MC68HC11 attaches directly to the QSPI since it has a compatible interface. Other peripheral units with serial interfaces including another MC68332 could be connected to the system. The electrical and timing requirements of the peripheral unit can be accommodated without additional circuitry in most cases since the QSPI signal characteristics can be selected by programming. In Figure 12.26, the Slave Select signal ($\overline{SS}$) of the MC68HC11 is connected to a peripheral select signal to enable the MC68HC11.



Num	Function	Min	Max	Unit
	Operating Frequency Master–SCK Slave	 DC DC	 1/4 1/4	 System Clock Frequency System Clock Frequency
1	Cycle Time Master–SCK	4	510	System Clocks
2	Enable Lead Time Master–DSCKL Slave	2	128	System Clocks
3	Enable Lag Time Master Slave	1/2	1/2 —	SCK
4	Clock (SCK) High or Low Time Master Slave	2 2	255 —	System Clocks System Clocks
5	Transfer Delay Master–DTL Slave (Does Not Require Deselect)	17 13	8192 —	System Clocks System Clocks

Figure 12.27 QSPI timing as a master (CPHA={0}).

The minimum delay PCSx to SCK is two system clock periods or 0.12μs using a 16.78MHz system clock. A value of $00 for DSCKL causes a delay of 128 system clock periods or 7.6 μs.

QSPI data transfers and chip selects. In the master mode, the QSPI begins execution of queue commands as soon as the QSPI is enabled. The chip select signal for a particular transfer is asserted and enables the receiving device. The SCK signal begins after the Enable Lead Time in Figure 12.27 elapses. Then, the transfer of the data bits begins on the MOSI line from the QSPI and on the MISO line to the QSPI. The external device reads the data bits on the selected edge of the SCK signal. When all the bits are transferred and received, the chip select signal is negated unless the Continue (CONT) bit is set in the command byte. The remaining commands in the queue are then executed in turn in a similar manner.

The continue (CONT) bit of the queue command byte determines the use of the peripheral chip select signals between transfers. If CONT = {1}, the peripheral select signals do not change between transfers. Thus, an external device could be continuously selected for several transfers. If CONT is set to {0}, the state of the peripheral chip select signals is determined by the value in QPDR. This register should be initialized with an appropriate value before the QSPI is enabled if the continuous mode for the peripheral chip select signals is not chosen.

The reader is referred to the *MC68332 User's Manuals* for a complete description of the electrical characteristics of the QSPI signal lines. The exact timing, drive capability, voltage range and other details are defined in the manuals. The *User's Manuals* for the M68HC11 family of 8-bit microcontrollers also contain detailed descriptions of the compatible Serial Peripheral Interface (SPI) for various applications.

A/D converter example. The Motorola MC145050 A/D converter is a serial output device that is used here to illustrate the selection of the parameters just discussed. The CPU program controlling the A/D converter must follow the timing constraints and protocol published in the A/D converter data sheet. Figure 12.28 defines the circuit connections between the QSPI and the A/D converter used in the examples to follow. The necessary power and grounding connections are not shown in the figure.

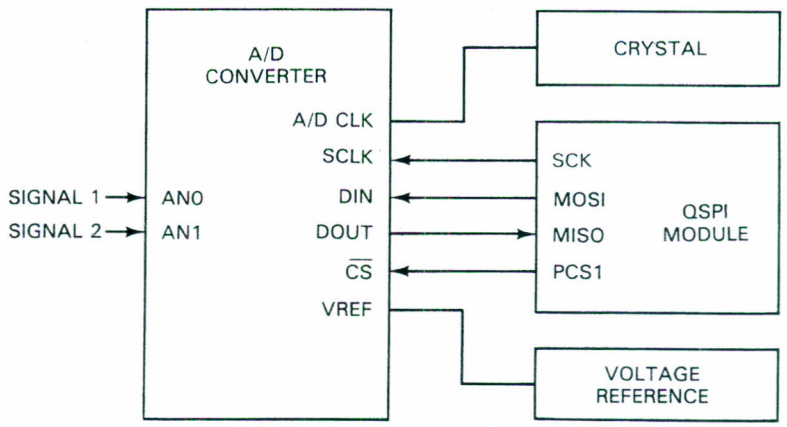


Figure 12.28 A/D converter to QSPI connection.

The MC145050 is a 10-bit A/D converter with eleven analog input channels

and three internal calibration channels. Figure 12.28 shows only the two channels used in examples in this section. The serial input (DIN) and output (DOUT) pins in Figure 12.28 allow simultaneous transfers between the QSPI and the A/D converter. When the A/D converter is selected by the QSPI, a setup time delay must occur before a particular A/D channel is addressed. A converted value is obtained by sending the address of the *next* channel to be converted while simultaneously reading the value converted on the previously addressed channel. However, the QSPI must not request a new conversion until the previous conversion is completed.

The A/D channels are addressed by the four *most significant* bits of the serial input to the A/D converter signal line DIN. The fourteen channels are thus addressed as

$$\$0XXX, \$1XXX, \ldots, \$DXXX$$

where the hexadecimal digits indicated as X are ignored. Each transfer can be from 10 bits to 16 bits. However, the A/D converter ignores the bits after the 10^{th}. The conversion for a channel begins after the 10^{th} bit is received.

Channels 0 through 10 can be connected to analog input signals. Three internal channels are available to calibrate the system. Reading channel 11 ($B) results in a half-scale 10-bit value of $200. Channel 12 yields a zero value. Channel 13 outputs the full-scale value of $3FF. The full-scale value corresponds to a voltage input equal to V_{REF} shown in Figure 12.28. The resulting reading is right justified in the QSPI receiver queue entry if 10 bit transfers are used. If 16 bit transfers are used, the result is left justified in the RAM.

Using 16-bit transfers, the received values for half-scale and full-scale would be $8000 and $FFC0, respectively. In this case, the result read into the QSPI receiver queue should be shifted right by 6-bit positions if the purpose is to measure the voltage range from $0000 to $03FF since each transfer is 16 bits but the A/D converter has only 10 bit precision. The corresponding true analog voltage is defined by the voltage reference in Figure 12.28.

Figure 12.28 also shows a separate A/D clock oscillator that is crystal controlled. The maximum A/D clock (A/DCLK) frequency is 2.1MHz. The maximum independent serial clock (SCK) frequency is also 2.1MHz.

Example 12.10

This example first shows how to calculate the minimum A/D converter setup time. Then, the necessary delay time between transfers is calculated. The example assumes 16 bit transfers between the A/D converter and the QSPI.

For this particular example, the following clock frequencies are used:

(a) 864kHz crystal oscillator for A/DCLK;
(b) 1.05MHz SCK selected in QSPI register SPCR0;
(c) 16.78MHz system clock ($f_{(\text{system})}$).

The periods for the clocks are thus

$$t_{\text{SCK}} = 1/(1.05 \times 10^6) = 0.95\mu\text{s}$$

and

$$t_{(\text{A/DCLK})} = 1/(864 \times 10^3) = 1.16\mu\text{s}.$$

The A/D *setup time* T_{SU} is defined in the A/D converter specifications as the minimum delay between the assertion of chip select ($\overline{CS}$) and the first rising edge of SCK. This is the *Enable Lead Time* in Figure 12.27 which is defined by DSCKL in SPCR1[14:8] when the DSCK bit is {1} in the command byte of a queue entry. The published setup time is

$$T_{SU} \geq 2 \times t_{(A/DCLK)} + 0.425\mu s.$$

The 864kHz A/DCLK with a 1.16μs period yields a minimum setup time of approximately 2.74μs. Equation 12.3 is used to determine the minimum DSCKL value as

$$DSCKL = T_{SU} \times f_{(system)}$$

where the setup time T_{SU} was substituted for the PCSx to SCK delay time. For this example, the result for the minimum value of DSCKL is

$$DSCKL = 2.74 \times 10^{-6}\text{sec} \times (16.78 \times 10^{6}\text{Hz}) = 46\ (\$2E).$$

which is the value to be written in SPCR1[14:8].

Next, the *Transfer Delay* in Figure 12.27 is calculated. The conversion time for this A/D converter is 44 A/DCLK cycles after the 10^{th} bit is received. Thus, the delay time between conversion commands to select the channel to be read must be at least

$$T_{(A/D\ delay)} = \frac{44}{864 \times 10^{3}\text{Hz}} = 50.9\mu s.$$

$T_{(A/D\ delay)}$ will determine the value of the DTL parameter in SPCR1[7:0]. However, the Transfer Delay time in Figure 12.27 is not necessarily the same as $T_{(A/D\ delay)}$ since the A/D converter begins conversion after the 10^{th} bit is received.

If the transfer length between the QSPI and the A/D converter is longer than 10 bits, the transfer time of the bits $11, 12, \ldots, N$ can be counted as part of the necessary Transfer Delay time. In this example, the QSPI is programmed to transfer 16 bits with an SCK frequency of 1.05MHz. Since the time required to transmit each bit is t_{SCK} = 0.95μs, the transmission of the first 10 bits takes 9.5μs. The last 6 bits which are ignored by the A/D converter require 6 $\times$ .95μs=5.7μs to transmit. Thus, the required Transfer Delay time is calculated as the difference between the A/D converter conversion time and the time taken to transmit the last 6 bits. This time is

$$t_{DTL} = T_{(A/D\ delay)} - 5.7\mu s = 50.9\mu s - 5.7\mu s = 45.2\mu s$$

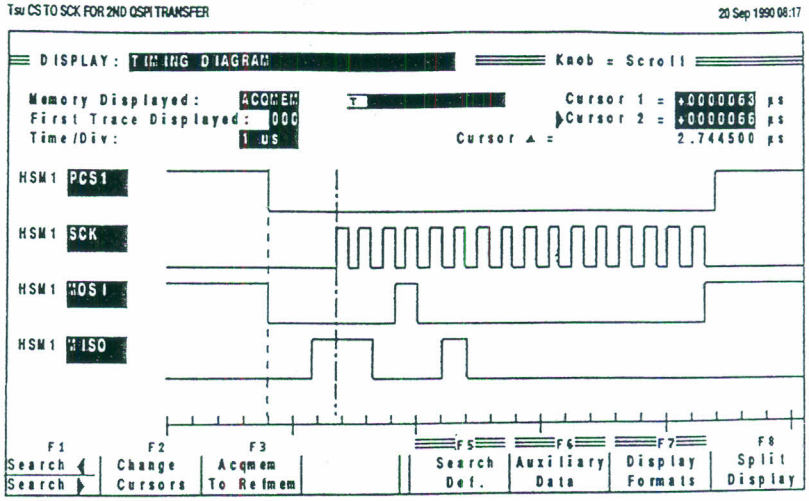
with the clock frequencies selected for this example. Using Equation 12.4, DTL is selected as

$$DTL = \frac{16.78 \times 10^{6}\text{Hz}}{32} \times (45.2 \times 10^{-6}\ \text{seconds}) = 23.7$$

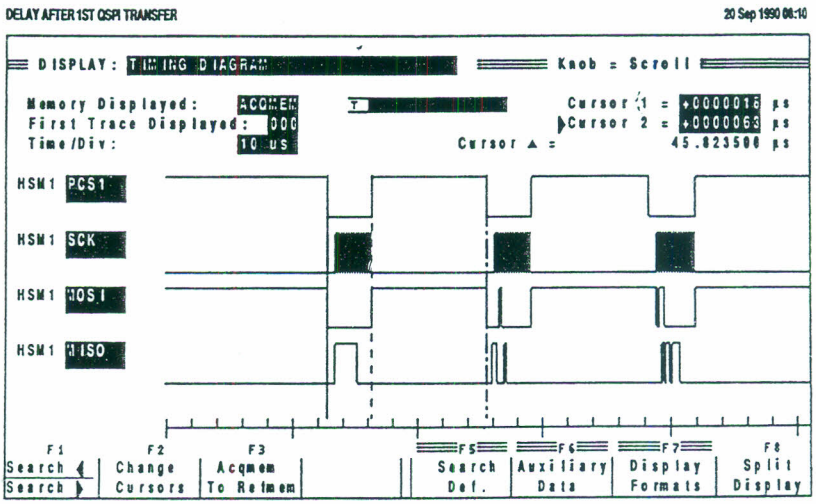
assuming a system clock frequency of 16.78MHz. As an integer, DTL is chosen to be 24 ($18) in SPCR1[7:0]. Using the value $18, the actual delay is then 45.8μs. Exercise 12.4.7 considers cases in which other values are necessary for the time delay between transfers.

The MC145050 data sheet indicates that the A/D channel address is clocked in on the first four rising edges of SCK. Thus, the parameters

$$CPOL = \{0\} \text{ and } CPHA = \{0\}$$



(a) PCS1 to SCK for transfer



(b) A/D conversion timing

Figure 12.29 Timing diagram for Example 12.10 and Example 12.11.

17.0 INTER-INTEGRATED CIRCUIT™ (I^2C™)

Note: This data sheet summarizes the features of this group of PIC24F devices. It is not intended to be a comprehensive reference source. For more information on the Inter-Integrated Circuit, refer to the *"PIC24F Family Reference Manual"*, **Section 24. "Inter-Integrated Circuit™ (I^2C™)"** (DS39702).

The Inter-Integrated Circuit (I^2C™) module is a serial interface useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial data EEPROMs, display drivers, A/D Converters, etc.

The I^2C module supports these features:

- Independent master and slave logic
- 7-bit and 10-bit device addresses
- General call address, as defined in the I^2C protocol
- Clock stretching to provide delays for the processor to respond to a slave data request
- Both 100 kHz and 400 kHz bus specifications
- Configurable address masking
- Multi-Master modes to prevent loss of messages in arbitration
- Bus Repeater mode, allowing the acceptance of all messages as a slave, regardless of the address
- Automatic SCL

A block diagram of the module is shown in Figure 17-1.

17.1 Pin Remapping Options

The I^2C module is tied to a fixed pin. To allow flexibility with peripheral multiplexing, the I2C1 module, in 28-pin devices, can be reassigned to the alternate pins. These alternate pins are designated as SCL1 and SDA1 during device configuration.

Pin assignment is controlled by the I2C1SEL Configuration bit. Programming this bit (= 0) multiplexes the module to the SCL1 and SDA1 pins.

17.2 Communicating as a Master in a Single Master Environment

The details of sending a message in Master mode depends on the communications protocol for the device being communicated with. Typically, the sequence of events is as follows:

1. Assert a Start condition on SDA1 and SCL1.
2. Send the I^2C device address byte to the slave with a write indication.
3. Wait for and verify an Acknowledge from the slave.
4. Send the first data byte (sometimes known as the command) to the slave.
5. Wait for and verify an Acknowledge from the slave.
6. Send the serial memory address low byte to the slave.
7. Repeat Steps 4 and 5 until all data bytes are sent.
8. Assert a Repeated Start condition on SDA1 and SCL1.
9. Send the device address byte to the slave with a read indication.
10. Wait for and verify an Acknowledge from the slave.
11. Enable master reception to receive serial memory data.
12. Generate an ACK or NACK condition at the end of a received byte of data.
13. Generate a Stop condition on SDA1 and SCL1.