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	PIC
Core Size	8-Bit
Speed	16MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	66
Program Memory Size	32KB (16K x 16)
Program Memory Type	OTP
EEPROM Size	-
RAM Size	902 x 8
Voltage - Supply (Vcc/Vdd)	4.5V ~ 5.5V
Data Converters	A/D 16x10b
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	84-LCC (J-Lead)
Supplier Device Package	84-PLCC (29.31x29.31)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic17c766t-16i-l

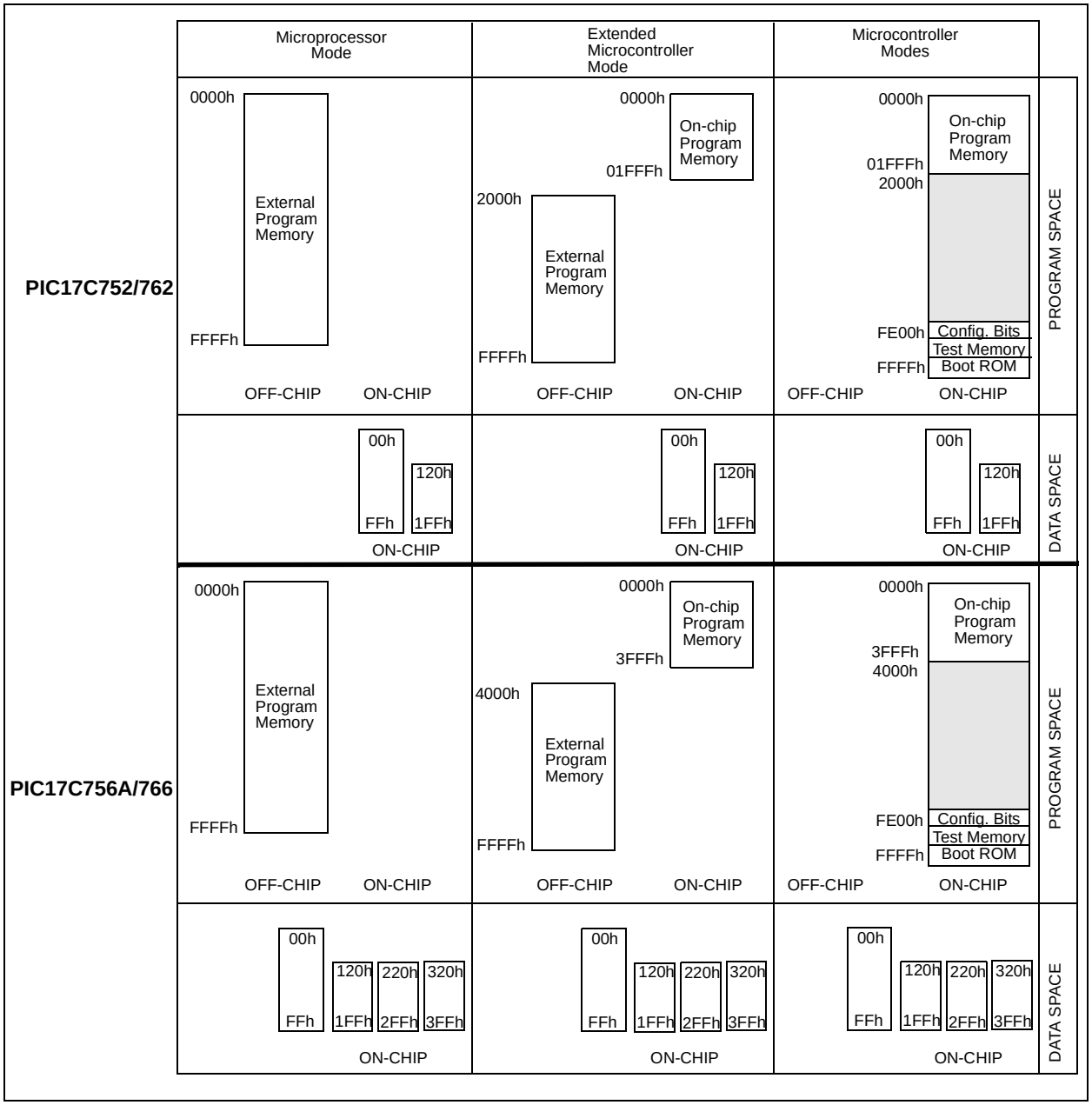
TABLE 7-1: MODE MEMORY ACCESS

Operating Mode	Internal Program Memory	Configuration Bits, Test Memory, Boot ROM
Microprocessor	No Access	No Access
Microcontroller	Access	Access
Extended Microcontroller	Access	No Access
Protected Microcontroller	Access	Access

The PIC17C7XX can operate in modes where the program memory is off-chip. They are the Microprocessor and Extended Microcontroller modes. The Microprocessor mode is the default for an unprogrammed device.

Regardless of the processor mode, data memory is always on-chip.

FIGURE 7-2: MEMORY MAP IN DIFFERENT MODES



7.4.2 INDIRECT ADDRESSING OPERATION

The indirect addressing capability has been enhanced over that of the PIC16CXX family. There are two control bits associated with each FSR register. These two bits configure the FSR register to:

- Auto-decrement the value (address) in the FSR after an indirect access
- Auto-increment the value (address) in the FSR after an indirect access
- No change to the value (address) in the FSR after an indirect access

These control bits are located in the ALUSTA register. The FSR1 register is controlled by the FS3:FS2 bits and FSR0 is controlled by the FS1:FS0 bits.

When using the auto-increment or auto-decrement features, the effect on the FSR is not reflected in the ALUSTA register. For example, if the indirect address causes the FSR to equal '0', the Z bit will not be set.

If the FSR register contains a value of 0h, an indirect read will read 0h (Zero bit is set) while an indirect write will be equivalent to a NOP (status bits are not affected).

Indirect addressing allows single cycle data transfers within the entire data space. This is possible with the use of the MOVPF and MOVFP instructions, where either 'p' or 'f' is specified as INDF0 (or INDF1).

If the source or destination of the indirect address is in banked memory, the location accessed will be determined by the value in the BSR.

A simple program to clear RAM from 20h - FFh is shown in Example 7-1.

EXAMPLE 7-1: INDIRECT ADDRESSING

```
    MOVLW  0x20          ;
    MOVWF  FSR0          ; FSR0 = 20h
    BCF    ALUSTA, FS1   ; Increment FSR
    BSF    ALUSTA, FS0   ; after access
    BCF    ALUSTA, C     ; C = 0
    MOVLW  END_RAM + 1   ;
LP  CLRF   INDF0, F      ; Addr(FSR) = 0
    CPFSEQ FSR0          ; FSR0 = END_RAM+1?
    GOTO   LP            ; NO, clear next
    :                    ; YES, All RAM is
    :                    ; cleared
```

7.5 Table Pointer (TBLPTRL and TBLPTRH)

File registers TBLPTRL and TBLPTRH form a 16-bit pointer to address the 64K program memory space. The table pointer is used by instructions TABLWT and TABLRD.

The TABLRD and the TABLWT instructions allow transfer of data between program and data space. The table pointer serves as the 16-bit address of the data word within the program memory. For a more complete description of these registers and the operation of Table Reads and Table Writes, see Section 8.0.

7.6 Table Latch (TBLATH, TBLATL)

The table latch (TBLAT) is a 16-bit register, with TBLATH and TBLATL referring to the high and low bytes of the register. It is not mapped into data or program memory. The table latch is used as a temporary holding latch during data transfer between program and data memory (see TABLRD, TABLWT, TLRD and TLWT instruction descriptions). For a more complete description of these registers and the operation of Table Reads and Table Writes, see Section 8.0.

9.0 HARDWARE MULTIPLIER

All PIC17C7XX devices have an 8 x 8 hardware multiplier included in the ALU of the device. By making the multiply a hardware operation, it completes in a single instruction cycle. This is an unsigned multiply that gives a 16-bit result. The result is stored into the 16-bit Product register (PRODH:PRODL). The multiplier does not affect any flags in the ALUSTA register.

Making the 8 x 8 multiplier execute in a single cycle gives the following advantages:

- Higher computational throughput
- Reduces code size requirements for multiply algorithms

The performance increase allows the device to be used in applications previously reserved for Digital Signal Processors.

Table 9-1 shows a performance comparison between PIC17CXXX devices using the single cycle hardware multiply and performing the same function without the hardware multiply.

Example 9-1 shows the sequence to do an 8 x 8 unsigned multiply. Only one instruction is required when one argument of the multiply is already loaded in the WREG register.

Example 9-2 shows the sequence to do an 8 x 8 signed multiply. To account for the sign bits of the arguments, each argument's most significant bit (MSb) is tested and the appropriate subtractions are done.

EXAMPLE 9-1: 8 x 8 UNSIGNED MULTIPLY ROUTINE

```
MOVFP    ARG1, WREG  ;
MULWF    ARG2        ; ARG1 * ARG2 ->
                     ;   PRODH:PRODL
```

EXAMPLE 9-2: 8 x 8 SIGNED MULTIPLY ROUTINE

```
MOVFP    ARG1, WREG
MULWF    ARG2        ; ARG1 * ARG2 ->
                     ;   PRODH:PRODL
BTFSC    ARG2, SB    ; Test Sign Bit
SUBWF    PRODH, F    ; PRODH = PRODH
                     ;         - ARG1
MOVFP    ARG2, WREG
BTFSC    ARG1, SB    ; Test Sign Bit
SUBWF    PRODH, F    ; PRODH = PRODH
                     ;         - ARG2
```

TABLE 9-1: PERFORMANCE COMPARISON

Routine	Multiply Method	Program Memory (Words)	Cycles (Max)	Time		
				@ 33 MHz	@ 16 MHz	@ 8 MHz
8 x 8 unsigned	Without hardware multiply	13	69	8.364 μs	17.25 μs	34.50 μs
	Hardware multiply	1	1	0.121 μs	0.25 μs	0.50 μs
8 x 8 signed	Without hardware multiply	—	—	—	—	—
	Hardware multiply	6	6	0.727 μs	1.50 μs	3.0 μs
16 x 16 unsigned	Without hardware multiply	21	242	29.333 μs	60.50 μs	121.0 μs
	Hardware multiply	24	24	2.91 μs	6.0 μs	12.0 μs
16 x 16 signed	Without hardware multiply	52	254	30.788 μs	63.50 μs	127.0 μs
	Hardware multiply	36	36	4.36 μs	9.0 μs	18.0 μs

13.1.2 TIMER1 AND TIMER2 IN 16-BIT MODE

To select 16-bit mode, set the T16 bit. In this mode, TMR2 and TMR1 are concatenated to form a 16-bit timer (TMR2:TMR1). The 16-bit timer increments until it matches the 16-bit period register (PR2:PR1). On the following timer clock, the timer value is reset to 0h, and the TMR1IF bit is set.

When selecting the clock source for the 16-bit timer, the TMR1CS bit controls the entire 16-bit timer and TMR2CS is a “don’t care”, however, ensure that TMR2ON is set (allows TMR2 to increment). When TMR1CS is clear, the timer increments once every instruction cycle (FOSC/4). When TMR1CS is set, the timer increments on every falling edge of the RB4/TCLK12 pin. For the 16-bit timer to increment, both TMR1ON and TMR2ON bits must be set (Table 13-2).

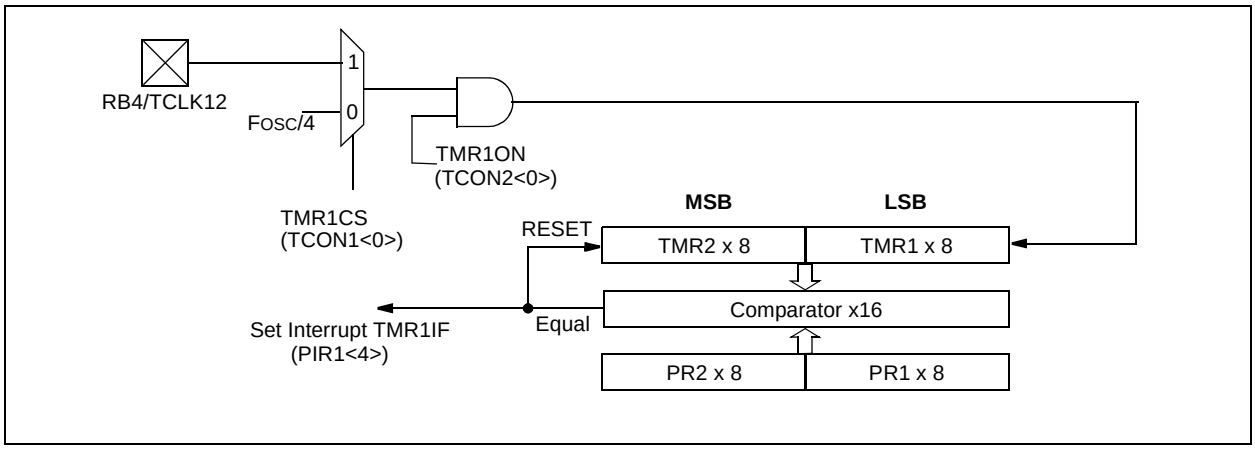
13.1.2.1 External Clock Input for TMR2:TMR1

When TMR1CS is set, the 16-bit TMR2:TMR1 increments on the falling edge of clock input TCLK12. The input on the RB4/TCLK12 pin is sampled and synchronized by the internal phase clocks twice every instruction cycle. This causes a delay from the time a falling edge appears on RB4/TCLK12 to the time TMR2:TMR1 is actually incremented. For the external clock input timing requirements, see the Electrical Specification section.

TABLE 13-2: TURNING ON 16-BIT TIMER

T16	TMR2ON	TMR1ON	Result
1	1	1	16-bit timer (TMR2:TMR1) ON
1	0	1	Only TMR1 increments
1	x	0	16-bit timer OFF
0	1	1	Timers in 8-bit mode

FIGURE 13-2: TMR2 AND TMR1 IN 16-BIT TIMER/COUNTER MODE



13.1.3.1 PWM Periods

The period of the PWM1 output is determined by Timer1 and its period register (PR1). The period of the PWM2 and PWM3 outputs can be individually software configured to use either Timer1 or Timer2 as the time-base. For PWM2, when TM2PW2 bit (PW2DCL<5>) is clear, the time base is determined by TMR1 and PR1 and when TM2PW2 is set, the time base is determined by Timer2 and PR2. For PWM3, when TM2PW3 bit (PW3DCL<5>) is clear, the time base is determined by TMR1 and PR1, and when TM2PW3 is set, the time base is determined by Timer2 and PR2.

Running two different PWM outputs on two different timers allows different PWM periods. Running all PWMs from Timer1 allows the best use of resources by freeing Timer2 to operate as an 8-bit timer. Timer1 and Timer2 cannot be used as a 16-bit timer if any PWM is being used.

The PWM periods can be calculated as follows:

$$\text{period of PWM1} = [(PR1) + 1] \times 4T_{OSC}$$

$$\text{period of PWM2} = [(PR1) + 1] \times 4T_{OSC} \text{ or } [(PR2) + 1] \times 4T_{OSC}$$

$$\text{period of PWM3} = [(PR1) + 1] \times 4T_{OSC} \text{ or } [(PR2) + 1] \times 4T_{OSC}$$

The duty cycle of PWMx is determined by the 10-bit value DCx<9:0>. The upper 8-bits are from register PWxDCH and the lower 2-bits are from PWxDCL<7:6> (PWxDCH:PWxDCL<7:6>). Table 13-4 shows the maximum PWM frequency (F_{PWM}), given the value in the period register.

The number of bits of resolution that the PWM can achieve depends on the operation frequency of the device as well as the PWM frequency (F_{PWM}).

Maximum PWM resolution (bits) for a given PWM frequency:

$$= \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log(2)} \text{ bits}$$

where: F_{PWM} = 1 / period of PWM

The PWMx duty cycle is as follows:

$$\text{PWMx Duty Cycle} = (DCx) \times T_{OSC}$$

where DCx represents the 10-bit value from PWxDCH:PWxDCL.

If DCx = 0, then the duty cycle is zero. If PRx = PWxDCH, then the PWM output will be low for one to four Q-clocks (depending on the state of the PWxDCL<7:6> bits). For a duty cycle to be 100%, the PWxDCH value must be greater then the PRx value.

The duty cycle registers for both PWM outputs are double buffered. When the user writes to these registers, they are stored in master latches. When TMR1 (or TMR2) overflows and a new PWM period begins, the master latch values are transferred to the slave latches and the PWMx pin is forced high.

Note:	For PW1DCH, PW1DCL, PW2DCH, PW2DCL, PW3DCH and PW3DCL registers, a write operation writes to the "master latches", while a read operation reads the "slave latches". As a result, the user may not read back what was just written to the duty cycle registers (until transferred to slave latch).

The user should also avoid any "read-modify-write" operations on the duty cycle registers, such as: `ADDWF PW1DCH`. This may cause duty cycle outputs that are unpredictable.

TABLE 13-4: PWM FREQUENCY vs. RESOLUTION AT 33 MHz

PWM Frequency	Frequency (kHz)				
	32.2	**64.5**	**90.66**	**128.9**	**515.6**
PRx Value	0xFF	0x7F	0x5A	0x3F	0x0F
High Resolution	10-bit	9-bit	8.5-bit	8-bit	6-bit
Standard Resolution	8-bit	7-bit	6.5-bit	6-bit	4-bit

13.1.3.2 PWM INTERRUPTS

The PWM modules make use of the TMR1 and/or TMR2 interrupts. A timer interrupt is generated when TMR1 or TMR2 equals its period register and on the following increment is cleared to zero. This interrupt also marks the beginning of a PWM cycle. The user can write new duty cycle values before the timer rollover. The TMR1 interrupt is latched into the TMR1IF bit and the TMR2 interrupt is latched into the TMR2IF bit. These flags must be cleared in software.

14.2 USART Asynchronous Mode

In this mode, the USART uses standard nonreturn-to-zero (NRZ) format (one START bit, eight or nine data bits, and one STOP bit). The most common data format is 8-bits. An on-chip dedicated 8-bit baud rate generator can be used to derive standard baud rate frequencies from the oscillator. The USART's transmitter and receiver are functionally independent but use the same data format and baud rate. The baud rate generator produces a clock x64 of the bit shift rate. Parity is not supported by the hardware, but can be implemented in software (and stored as the ninth data bit). Asynchronous mode is stopped during SLEEP.

The Asynchronous mode is selected by clearing the SYNC bit (TXSTA<4>).

The USART Asynchronous module consists of the following components:

- Baud Rate Generator
- Sampling Circuit
- Asynchronous Transmitter
- Asynchronous Receiver

14.2.1 USART ASYNCHRONOUS TRANSMITTER

The USART transmitter block diagram is shown in Figure 14-1. The heart of the transmitter is the transmit shift register (TSR). The shift register obtains its data from the read/write transmit buffer (TXREG). TXREG is loaded with data in software. The TSR is not loaded until the STOP bit has been transmitted from the previous load. As soon as the STOP bit is transmitted, the TSR is loaded with new data from the TXREG (if available). Once TXREG transfers the data to the TSR (occurs in one TCY at the end of the current BRG cycle), the TXREG is empty and an interrupt bit, TXIF, is set. This interrupt can be enabled/disabled by setting/clearing the TXIE bit. TXIF will be set, regardless of TXIE and cannot be reset in software. It will reset only when new data is loaded into TXREG. While TXIF indicates the status of the TXREG, the TRMT (TXSTA<1>) bit shows the status of the TSR.

TRMT is a read only bit which is set when the TSR is empty. No interrupt logic is tied to this bit, so the user has to poll this bit in order to determine if the TSR is empty.

Note:	The TSR is not mapped in data memory, so it is not available to the user.

Transmission is enabled by setting the TXEN (TXSTA<5>) bit. The actual transmission will not occur until TXREG has been loaded with data and the baud rate generator (BRG) has produced a shift clock (Figure 14-3). The transmission can also be started by first loading TXREG and then setting TXEN. Normally, when transmission is first started, the TSR is empty, so a transfer to TXREG will result in an immediate transfer to TSR, resulting in an empty TXREG. A back-to-back transfer is thus possible (Figure 14-4). Clearing TXEN during a transmission will cause the transmission to be aborted. This will reset the transmitter and the TX/CK pin will revert to hi-impedance.

In order to select 9-bit transmission, the TX9 (TXSTA<6>) bit should be set and the ninth bit value should be written to TX9D (TXSTA<0>). The ninth bit value must be written before writing the 8-bit data to the TXREG. This is because a data write to TXREG can result in an immediate transfer of the data to the TSR (if the TSR is empty).

Steps to follow when setting up an Asynchronous Transmission:

1. Initialize the SPBRG register for the appropriate baud rate.
2. Enable the asynchronous serial port by clearing the SYNC bit and setting the SPEN bit.
3. If interrupts are desired, then set the TXIE bit.
4. If 9-bit transmission is desired, then set the TX9 bit.
5. If 9-bit transmission is selected, the ninth bit should be loaded in TX9D.
6. Load data to the TXREG register.
7. Enable the transmission by setting TXEN (starts transmission).

FIGURE 14-3: ASYNCHRONOUS MASTER TRANSMISSION

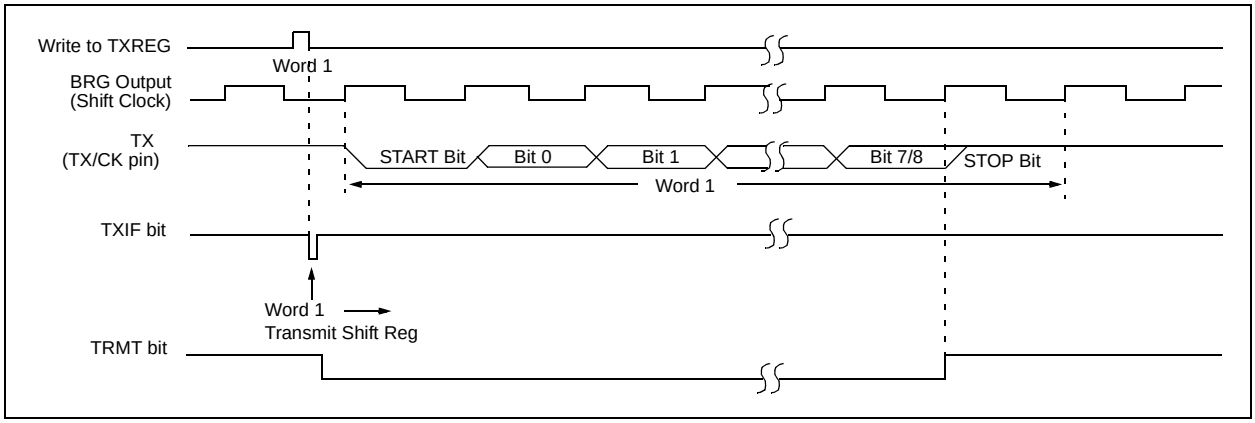


FIGURE 15-8: SPI MODE WAVEFORM (SLAVE MODE WITH CKE = 0)

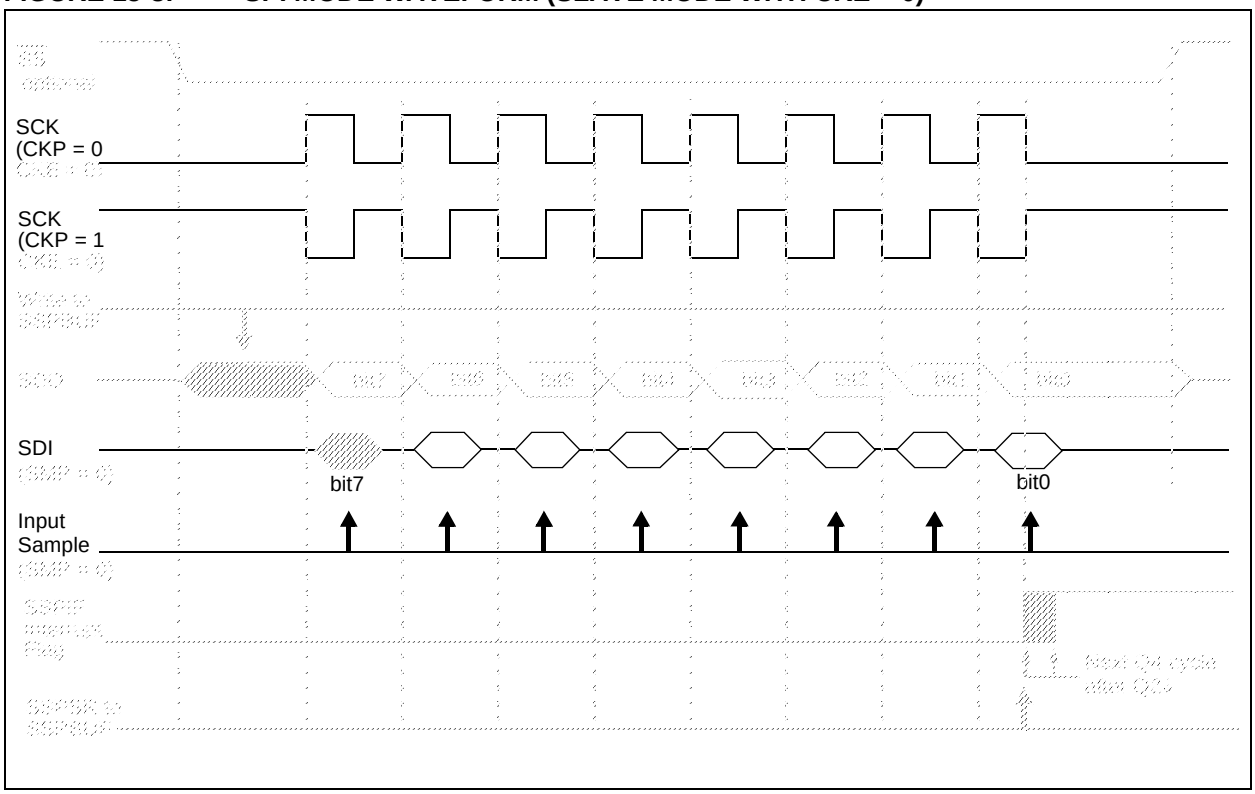
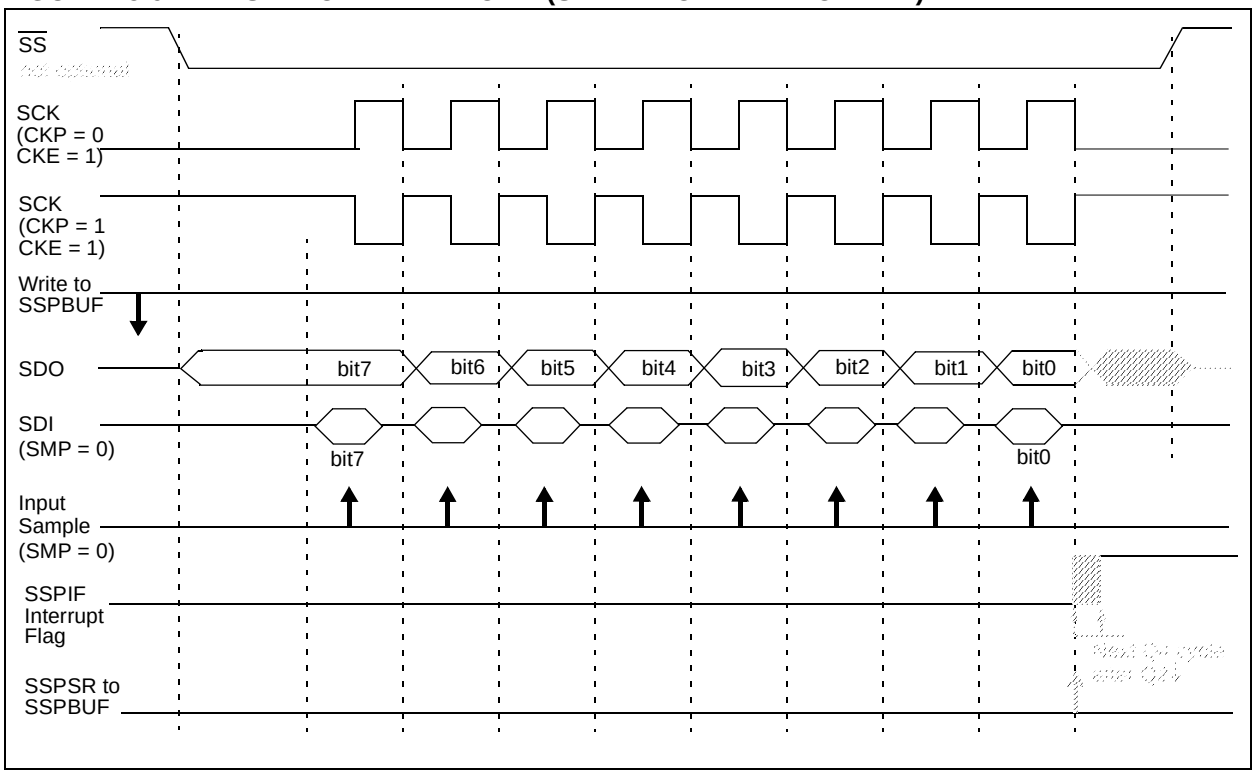


FIGURE 15-9: SPI MODE WAVEFORM (SLAVE MODE WITH CKE = 1)



The SSPSTAT register gives the status of the data transfer. This information includes detection of a START or STOP bit, specifies if the received byte was data or address if the next byte is the completion of 10-bit address and if this will be a read or write data transfer.

The SSPBUF is the register to which transfer data is written to or read from. The SSPSR register shifts the data in or out of the device. In receive operations, the SSPBUF and SSPSR create a doubled buffered receiver. This allows reception of the next byte to begin before reading the last byte of received data. When the complete byte is received, it is transferred to the SSPBUF register and flag bit SSPIF is set. If another complete byte is received before the SSPBUF register is read, a receiver overflow has occurred and bit SSPOV (SSPCON1<6>) is set and the byte in the SSPSR is lost.

The SSPADD register holds the slave address. In 10-bit mode, the user needs to write the high byte of the address (1111 0 A9 A8 0). Following the high byte address match, the low byte of the address needs to be loaded (A7:A0).

15.2.1 SLAVE MODE

In Slave mode, the SCL and SDA pins must be configured as inputs. The MSSP module will override the input state with the output data when required (slave-transmitter).

When an address is matched or the data transfer after an address match is received, the hardware automatically will generate the acknowledge ($\overline{ACK}$) pulse and then load the SSPBUF register with the received value currently in the SSPSR register.

There are certain conditions that will cause the MSSP module not to give this $\overline{ACK}$ pulse. These are if either (or both):

a) The buffer full bit BF (SSPSTAT<0>) was set before the transfer was received.

b) The overflow bit SSPOV (SSPCON1<6>) was set before the transfer was received.

If the BF bit is set, the SSPSR register value is not loaded into the SSPBUF, but bit SSPIF and SSPOV are set. Table 15-2 shows what happens when a data transfer byte is received, given the status of bits BF and SSPOV. The shaded cells show the condition where user software did not properly clear the overflow condition. Flag bit BF is cleared by reading the SSPBUF register, while bit SSPOV is cleared through software.

The SCL clock input must have a minimum high and low time for proper operation. The high and low times of the I^2C specification, as well as the requirement of the MSSP module, are shown in timing parameter #100 and parameter #101 of the Electrical Specifications.

15.2.18.1 Bus Collision During a START Condition

During a START condition, a bus collision occurs if:

a) SDA or SCL are sampled low at the beginning of the START condition (Figure 15-35).

b) SCL is sampled low before SDA is asserted low (Figure 15-36).

During a START condition, both the SDA and the SCL pins are monitored.

If:

the SDA pin is already low
or the SCL pin is already low,

then:

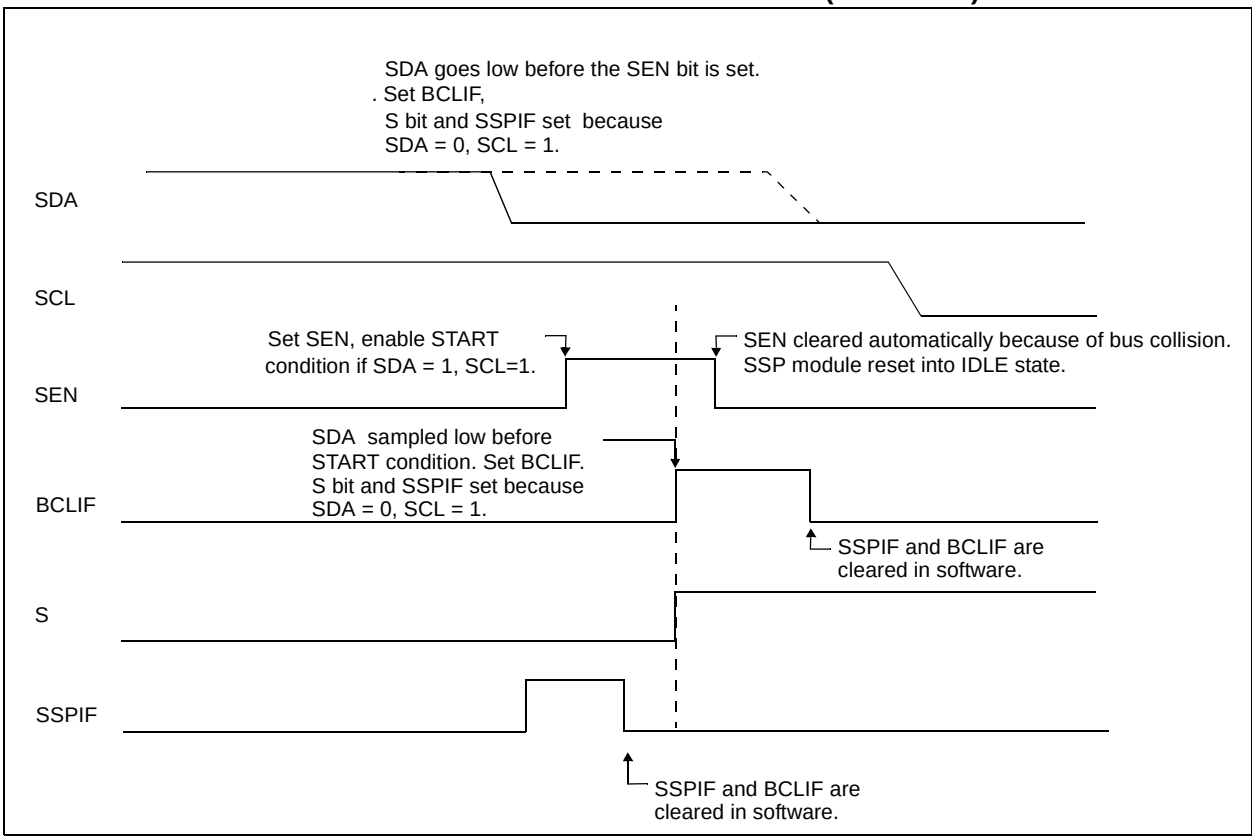
the START condition is aborted,
and the BCLIF flag is set,
and the SSP module is reset to its IDLE state (Figure 15-35).

The START condition begins with the SDA and SCL pins de-asserted. When the SDA pin is sampled high, the baud rate generator is loaded from SSPADD<6:0> and counts down to '0'. If the SCL pin is sampled low while SDA is high, a bus collision occurs, because it is assumed that another master is attempting to drive a data '1' during the START condition.

If the SDA pin is sampled low during this count, the BRG is reset and the SDA line is asserted early (Figure 15-37). If, however, a '1' is sampled on the SDA pin, the SDA pin is asserted low at the end of the BRG count. The baud rate generator is then reloaded and counts down to 0 and during this time, if the SCL pin is sampled as '0', a bus collision does not occur. At the end of the BRG count, the SCL pin is asserted low.

Note: The reason that bus collision is not a factor during a START condition is that no two bus masters can assert a START condition at the exact same time. Therefore, one master will always assert SDA before the other. This condition does not cause a bus collision because the two masters must be allowed to arbitrate the first address following the START condition and if the address is the same, arbitration must be allowed to continue into the data portion, Repeated Start, or Stop conditions.

FIGURE 15-35: BUS COLLISION DURING START CONDITION (SDA ONLY)



15.4 Example Program

Example 15-2 shows MPLAB® C17 'C' code for using the I^2C module in Master mode to communicate with a 24LC01B serial EEPROM. This example uses the PIC® MCU 'C' libraries included with MPLAB C17.

EXAMPLE 15-2: INTERFACING TO A 24LC01B SERIAL EEPROM (USING MPLAB C17)

```
// Include necessary header files
#include <p17c756.h>     // Processor header file
#include <delays.h>      // Delay routines header file
#include <stdlib.h>      // Standard Library header file
#include <i2c16.h>       // I2C routines header file

#define CONTROL 0xa0     // Control byte definition for 24LC01B

// Function declarations
void main(void);
void WritePORTD(static unsigned char data);
void ByteWrite(static unsigned char address,static unsigned char data);
unsigned char ByteRead(static unsigned char address);
void ACKPoll(void);

// Main program
void main(void)
{
static unsigned char address;    // I2C address of 24LC01B
static unsigned char datao;      // Data written to 24LC01B
static unsigned char datai;      // Data read from 24LC01B

    address = 0;                 // Preset address to 0
    OpenI2C(MASTER,SLEW_ON);     // Configure I2C Module Master mode, Slew rate control on
    SSPADD = 39;                 // Configure clock for 100KHz

    while(address<128)           // Loop 128 times, 24LC01B is 128x8
    {
        datao = PORTB;
        do
        {
            ByteWrite(address,datao);   // Write data to EEPROM
            ACKPoll();                  // Poll the 24LC01B for state
            datai = ByteRead(address);  // Read data from EEPROM into SSPBUF
        } while(datai != datao);        // Loop as long as data not correctly
                                        //    written to 24LC01B

        address++;                      // Increment address
    }
    while(1)                            // Done writing 128 bytes to 24LC01B, Loop forever
    {
        Nop();
    }
}
```

ADDWFC **ADD WREG and Carry bit to f**

Syntax: [*label*] ADDWFC f,d

Operands: $0 \le f \le 255$
$d \in [0,1]$

Operation: (WREG) + (f) + C → (dest)

Status Affected: OV, C, DC, Z

Encoding:

0001	000d	ffff	ffff

Description: Add WREG, the Carry Flag and data memory location 'f'. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed in data memory location 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `ADDWFC REG 0`

Before Instruction
Carry bit = 1
REG = 0x02
WREG = 0x4D

After Instruction
Carry bit = 0
REG = 0x02
WREG = 0x50

ANDLW **And Literal with WREG**

Syntax: [*label*] ANDLW k

Operands: $0 \le k \le 255$

Operation: (WREG) .AND. (k) → (WREG)

Status Affected: Z

Encoding:

1011	0101	kkkk	kkkk

Description: The contents of WREG are AND'ed with the 8-bit literal 'k'. The result is placed in WREG.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to WREG

Example: `ANDLW 0x5F`

Before Instruction
WREG = 0xA3

After Instruction
WREG = 0x03

DECF **Decrement f**

Syntax: [*label*] DECF f,d

Operands: $0 \le f \le 255$
$d \in [0,1]$

Operation: (f) – 1 → (dest)

Status Affected: OV, C, DC, Z

Encoding:

0000	011d	ffff	ffff

Description: Decrement register 'f'. If 'd' is 0, the result is stored in WREG. If 'd' is 1, the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `DECF CNT, 1`

Before Instruction
CNT = 0x01
Z = 0

After Instruction
CNT = 0x00
Z = 1

DECFSZ **Decrement f, skip if 0**

Syntax: [*label*] DECFSZ f,d

Operands: $0 \le f \le 255$
$d \in [0,1]$

Operation: (f) – 1 → (dest);
skip if result = 0

Status Affected: None

Encoding:

0001	011d	ffff	ffff

Description: The contents of register 'f' are decremented. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f'.

If the result is 0, the next instruction, which is already fetched is discarded and a `NOP` is executed instead, making it a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example:
```
HERE    DECFSZ  CNT, 1
        GOTO    HERE
NZERO
ZERO
```

Before Instruction
PC = Address `(HERE)`

After Instruction
CNT = CNT - 1
If CNT = 0;
PC = Address `(HERE)`
If CNT ≠ 0;
PC = Address `(NZERO)`

RLNCF **Rotate Left f (no carry)**

Syntax: [*label*] RLNCF f,d

Operands: $0 \le f \le 255$
$d \in [0,1]$

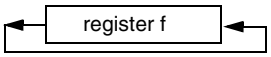
Operation: f<n> → d<n+1>;
f<7> → d<0>

Status Affected: None

Encoding:

0010	001d	ffff	ffff

Description: The contents of register 'f' are rotated one bit to the left. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is stored back in register 'f'.



Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: RLNCF REG, 1

Before Instruction
C = 0
REG = 1110 1011

After Instruction
C =
REG = 1101 0111

RRCF **Rotate Right f through Carry**

Syntax: [*label*] RRCF f,d

Operands: $0 \le f \le 255$
$d \in [0,1]$

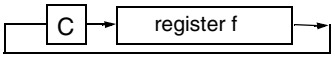
Operation: f<n> → d<n-1>;
f<0> → C;
C → d<7>

Status Affected: C

Encoding:

0001	100d	ffff	ffff

Description: The contents of register 'f' are rotated one bit to the right through the Carry Flag. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f'.



Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: RRCF REG1,0

Before Instruction
REG1 = 1110 0110
C = 0

After Instruction
REG1 = 1110 0110
WREG = 0111 0011
C = 0

20.3 Timing Parameter Symbology

The timing parameter symbols have been created following one of the following formats:

1. TppS2ppS
2. TppS
3. TCC:ST (I²C specifications only)
4. Ts (I²C specifications only)

T			
F	Frequency	T	Time

Lowercase symbols (pp) and their meanings:

pp			
ad	Address/Data	ost	Oscillator Start-Up Timer
al	ALE	pwrt	Power-Up Timer
cc	Capture1 and Capture2	rb	PORTB
ck	CLKOUT or clock	rd	$\overline{RD}$
dt	Data in	rw	$\overline{RD}$ or $\overline{WR}$
in	INT pin	t0	T0CKI
io	I/O port	t123	TCLK12 and TCLK3
mc	$\overline{MCLR}$	wdt	Watchdog Timer
oe	$\overline{OE}$	wr	$\overline{WR}$
os	OSC1		

Uppercase symbols and their meanings:

S			
D	Driven	L	Low
E	Edge	P	Period
F	Fall	R	Rise
H	High	V	Valid
I	Invalid (Hi-impedance)	Z	Hi-impedance

FIGURE 20-21: USART ASYNCHRONOUS MODE START BIT DETECT

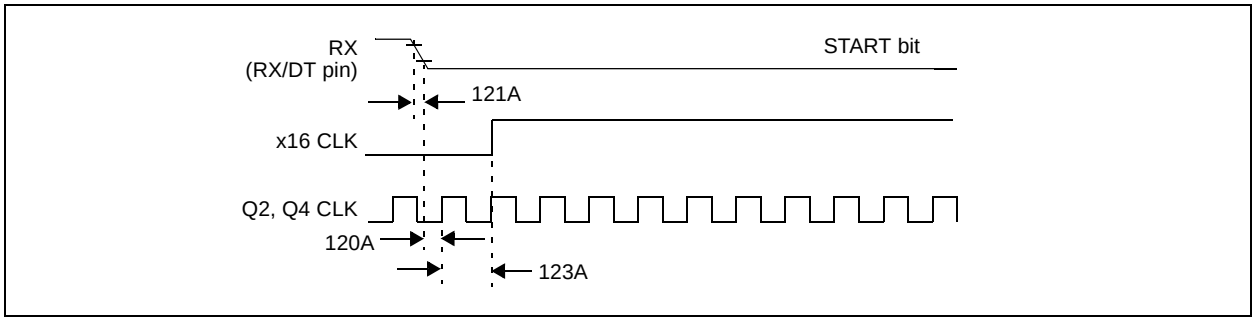


TABLE 20-16: USART ASYNCHRONOUS MODE START BIT DETECT REQUIREMENTS

Param No.	Sym	Characteristic		Min	Typ	Max	Units	Conditions
120A	TdtL2ckH	Time to ensure that the RX pin is sampled low		—	—	TCY	ns	
121A	TdtRF	Data rise time and fall time	Receive	—	—	**(Note 1)**	ns	
			Transmit	—	—	40	ns	
123A	TckH2bckL	Time from RX pin sampled low to first rising edge of x16 clock		—	—	TCY	ns	

Note 1: Schmitt trigger will determine logic level.

FIGURE 20-22: USART ASYNCHRONOUS RECEIVE SAMPLING WAVEFORM

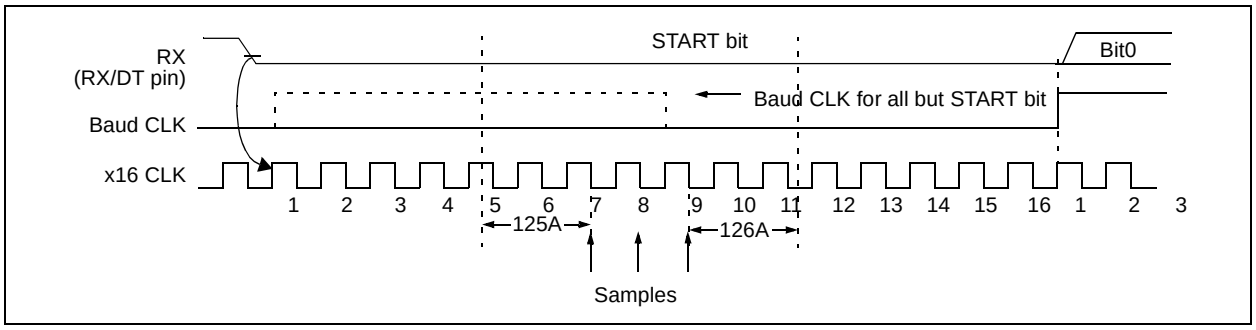
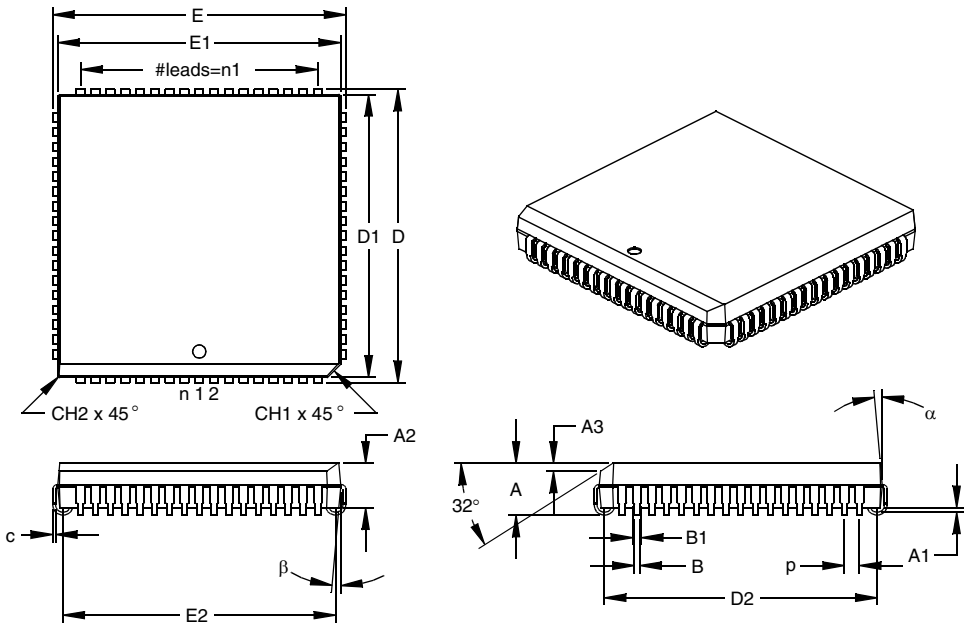


TABLE 20-17: USART ASYNCHRONOUS RECEIVE SAMPLING REQUIREMENTS

Param No.	Sym	Characteristic	Min	Typ	Max	Units	Conditions
125A	TdtL2ckH	Setup time of RX pin to first data sampled	TCY	—	—	ns	
126A	TdtL2ckH	Hold time of RX pin from last data sampled	TCY	—	—	ns	

68-Lead Plastic Leaded Chip Carrier (L) – Square (PLCC)

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging

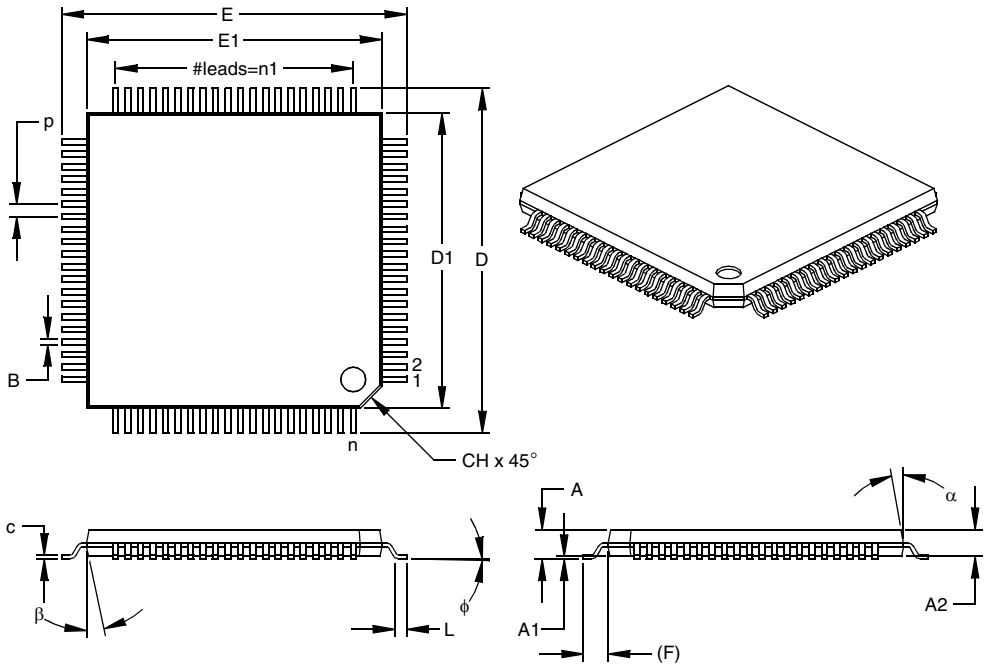


Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		68			68	
Pitch	p		.050			1.27	
Pins per Side	n1		17			17	
Overall Height	A	.165	.173	.180	4.19	4.39	4.57
Molded Package Thickness	A2	.145	.153	.160	3.68	3.87	4.06
Standoff §	A1	.020	.028	.035	0.51	0.71	0.89
Side 1 Chamfer Height	A3	.024	.029	.034	0.61	0.74	0.86
Corner Chamfer 1	CH1	.040	.045	.050	1.02	1.14	1.27
Corner Chamfer (others)	CH2	.000	.005	.010	0.00	0.13	0.25
Overall Width	E	.985	.990	.995	25.02	25.15	25.27
Overall Length	D	.985	.990	.995	25.02	25.15	25.27
Molded Package Width	E1	.950	.954	.958	24.13	24.23	24.33
Molded Package Length	D1	.950	.954	.958	24.13	24.23	24.33
Footprint Width	E2	.890	.920	.930	22.61	23.37	23.62
Footprint Length	D2	.890	.920	.930	22.61	23.37	23.62
Lead Thickness	c	.008	.011	.013	0.20	0.27	0.33
Upper Lead Width	B1	.026	.029	.032	0.66	0.74	0.81
Lower Lead Width	B	.013	.020	.021	0.33	0.51	0.53
Mold Draft Angle Top	α	0	5	10	0	5	10
Mold Draft Angle Bottom	β	0	5	10	0	5	10

* Controlling Parameter
§ Significant Characteristic
Notes:
Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010” (0.254mm) per side.
JEDEC Equivalent: MO-047
Drawing No. C04-049

80-Lead Plastic Thin Quad Flatpack (PT) 12x12x1 mm Body, 1.0/0.10 mm Lead Form (TQFP)

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Units		INCHES			MILLIMETERS*		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		80			80	
Pitch	p		.020			0.50	
Pins per Side	n1		20			20	
Overall Height	A	.039	.043	.047	1.00	1.10	1.20
Molded Package Thickness	A2	.037	.039	.041	0.95	1.00	1.05
Standoff §	A1	.002	.004	.006	0.05	0.10	0.15
Foot Length	L	.018	.024	.030	0.45	0.60	0.75
Footprint (Reference)	(F)		.039			1.00	
Foot Angle	ϕ	0	3.5	7	0	3.5	7
Overall Width	E	.541	.551	.561	13.75	14.00	14.25
Overall Length	D	.541	.551	.561	13.75	14.00	14.25
Molded Package Width	E1	.463	.472	.482	11.75	12.00	12.25
Molded Package Length	D1	.463	.472	.482	11.75	12.00	12.25
Lead Thickness	c	.004	.006	.008	0.09	0.15	0.20
Lead Width	B	.007	.009	.011	0.17	0.22	0.27
Pin 1 Corner Chamfer	CH	.025	.035	.045	0.64	0.89	1.14
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

* Controlling Parameter
§ Significant Characteristic

Notes:
Dimensions D1 and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010” (0.254mm) per side.
JEDEC Equivalent: MS-026
Drawing No. C04-092

ON-LINE SUPPORT

Microchip provides on-line support on the Microchip World Wide Web (WWW) site.

The web site is used by Microchip as a means to make files and information easily available to customers. To view the site, the user must have access to the Internet and a web browser, such as Netscape or Microsoft Explorer. Files are also available for FTP download from our FTP site.

Connecting to the Microchip Internet Web Site

The Microchip web site is available by using your favorite Internet browser to attach to:

www.microchip.com

The file transfer site is available by using an FTP service to connect to:

ftp://ftp.microchip.com

The web site and file transfer site provide a variety of services. Users may download files for the latest Development Tools, Data Sheets, Application Notes, User's Guides, Articles and Sample Programs. A variety of Microchip specific business information is also available, including listings of Microchip sales offices, distributors and factory representatives. Other data available for consideration is:

- Latest Microchip Press Releases
- Technical Support Section with Frequently Asked Questions
- Design Tips
- Device Errata
- Job Postings
- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip Products
- Conferences for products, Development Systems, technical information and more
- Listing of seminars and events