

Welcome to [E-XFL.COM](https://www.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](#)"

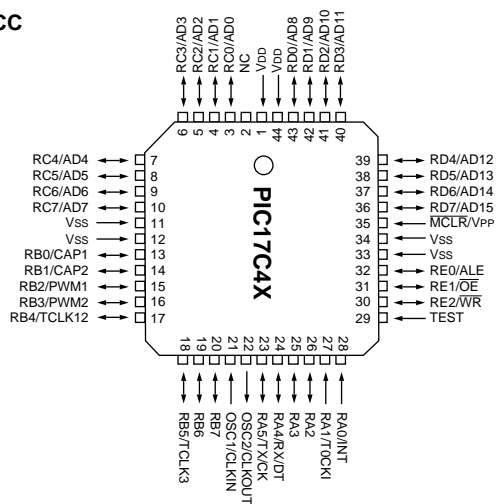
Details

Product Status	Obsolete
Core Processor	PIC
Core Size	8-Bit
Speed	16MHz
Connectivity	UART/USART
Peripherals	POR, PWM, WDT
Number of I/O	33
Program Memory Size	16KB (8K x 16)
Program Memory Type	OTP
EEPROM Size	-
RAM Size	454 x 8
Voltage - Supply (Vcc/Vdd)	4.5V ~ 6V
Data Converters	-
Oscillator Type	External
Operating Temperature	-40°C ~ 125°C (TA)
Mounting Type	Surface Mount
Package / Case	44-LCC (J-Lead)
Supplier Device Package	44-PLCC (16.59x16.59)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic17c44-16e-l

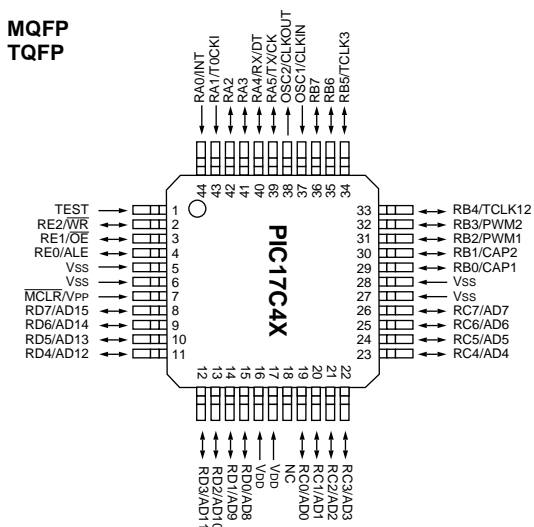
PIC17C4X

Pin Diagrams Cont'd

PLCC



MQFP
TQFP



All devices are available in all package types, listed in Section 21.0, with the following exceptions:

- ROM devices are not available in Windowed Cerdip Packages
- TQFP is not available for the PIC17C42.

PIC17C4X

NOTES:

7.1 Table Writes to Internal Memory

A table write operation to internal memory causes a long write operation. The long write is necessary for programming the internal EPROM. Instruction execution is halted while in a long write cycle. The long write will be terminated by any enabled interrupt. To ensure that the EPROM location has been well programmed, a minimum programming time is required (see specification #D114). Having only one interrupt enabled to terminate the long write ensures that no unintentional interrupts will prematurely terminate the long write.

The sequence of events for programming an internal program memory location should be:

1. Disable all interrupt sources, except the source to terminate EPROM program write.
2. Raise $\overline{\text{MCLR}}$ /VPP pin to the programming voltage.
3. Clear the WDT.
4. Do the table write. The interrupt will terminate the long write.
5. Verify the memory location (table read).

Note: Programming requirements must be met. See timing specification in electrical specifications for the desired device. Violating these specifications (including temperature) may result in EPROM locations that are not fully programmed and may lose their state over time.

7.1.1 TERMINATING LONG WRITES

An interrupt source or reset are the only events that terminate a long write operation. Terminating the long write from an interrupt source requires that the interrupt enable and flag bits are set. The GLINTD bit only enables the vectoring to the interrupt address.

If the T0CKI, RA0/INT, or TMR0 interrupt source is used to terminate the long write; the interrupt flag, of the highest priority enabled interrupt, will terminate the long write and automatically be cleared.

Note 1: If an interrupt is pending, the TABLWT is aborted (an NOP is executed). The highest priority pending interrupt, from the T0CKI, RA0/INT, or TMR0 sources that is enabled, has its flag cleared.

Note 2: If the interrupt is not being used for the program write timing, the interrupt should be disabled. This will ensure that the interrupt is not lost, nor will it terminate the long write prematurely.

If a peripheral interrupt source is used to terminate the long write, the interrupt enable and flag bits must be set. The interrupt flag will not be automatically cleared upon the vectoring to the interrupt vector address.

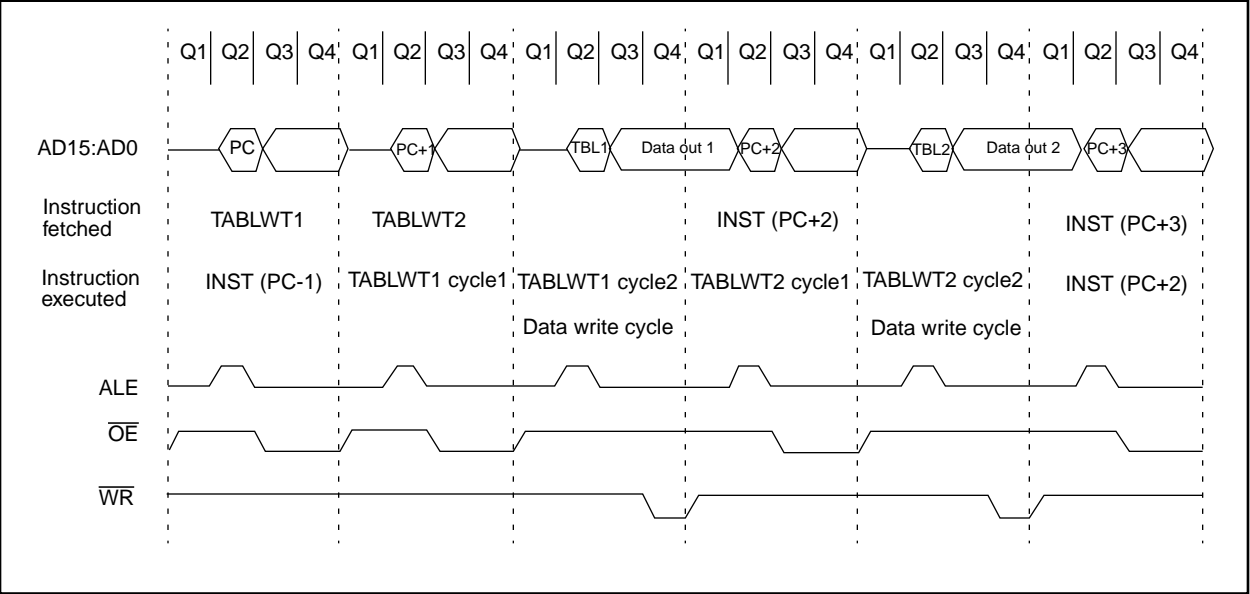
If the GLINTD bit is cleared prior to the long write, when the long write is terminated, the program will branch to the interrupt vector.

If the GLINTD bit is set prior to the long write, when the long write is terminated, the program will not vector to the interrupt address.

TABLE 7-1: INTERRUPT - TABLE WRITE INTERACTION

Interrupt Source	GLINTD	Enable Bit	Flag Bit	Action
RA0/INT, TMR0, T0CKI	0	1	1	Terminate long table write (to internal program memory), branch to interrupt vector (branch clears flag bit).
	0	1	0	None
	1	0	x	None
	1	1	1	Terminate table write, do not branch to interrupt vector (flag is automatically cleared).
Peripheral	0	1	1	Terminate table write, branch to interrupt vector.
	0	1	0	None
	1	0	x	None
	1	1	1	Terminate table write, do not branch to interrupt vector (flag is set).

FIGURE 7-6: CONSECUTIVE TABLWT WRITE TIMING (EXTERNAL MEMORY)



Example 8-3 shows the sequence to do a 16 x 16 unsigned multiply. Equation 8-1 shows the algorithm that is used. The 32-bit result is stored in 4 registers RES3:RES0.

EQUATION 8-1: 16 x 16 UNSIGNED MULTIPLICATION ALGORITHM

$$\begin{aligned}
 \text{RES3:RES0} &= \text{ARG1H:ARG1L} * \text{ARG2H:ARG2L} \\
 &= (\text{ARG1H} * \text{ARG2H} * 2^{16}) + \\
 &\quad (\text{ARG1H} * \text{ARG2L} * 2^8) + \\
 &\quad (\text{ARG1L} * \text{ARG2H} * 2^8) + \\
 &\quad (\text{ARG1L} * \text{ARG2L})
 \end{aligned}$$

EXAMPLE 8-3: 16 x 16 MULTIPLY ROUTINE

```

MOVFP    ARG1L, WREG
MULWF    ARG2L          ; ARG1L * ARG2L ->
                        ;   PRODH:PRODL

MOVFP    PRODH, RES1 ;
MOVFP    PRODL, RES0 ;

;

MOVFP    ARG1H, WREG
MULWF    ARG2H          ; ARG1H * ARG2H ->
                        ;   PRODH:PRODL

MOVFP    PRODH, RES3 ;
MOVFP    PRODL, RES2 ;

;

MOVFP    ARG1L, WREG
MULWF    ARG2H          ; ARG1L * ARG2H ->
                        ;   PRODH:PRODL

MOVFP    PRODL, WREG ;
ADDWF    RES1, F        ; Add cross
MOVFP    PRODH, WREG ;   products
ADDWFC   RES2, F        ;
CLRf     WREG, F        ;
ADDWFC   RES3, F        ;

;

MOVFP    ARG1H, WREG ;
MULWF    ARG2L          ; ARG1H * ARG2L ->
                        ;   PRODH:PRODL

MOVFP    PRODL, WREG ;
ADDWF    RES1, F        ; Add cross
MOVFP    PRODH, WREG ;   products
ADDWFC   RES2, F        ;
CLRf     WREG, F        ;
ADDWFC   RES3, F        ;

```

9.2 PORTB and DDRB Registers

PORTB is an 8-bit wide bi-directional port. The corresponding data direction register is DDRB. A '1' in DDRB configures the corresponding port pin as an input. A '0' in the DDRB register configures the corresponding port pin as an output. Reading PORTB reads the status of the pins, whereas writing to it will write to the port latch.

Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is done by clearing the $\overline{\text{RBP}}\overline{\text{U}}$ (PORTA<7>) bit. The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are enabled on any reset.

PORTB also has an interrupt on change feature. Only pins configured as inputs can cause this interrupt to occur (i.e. any RB7:RB0 pin configured as an output is excluded from the interrupt on change comparison). The input pins (of RB7:RB0) are compared with the value in the PORTB data latch. The "mismatch" outputs of RB7:RB0 are OR'ed together to generate the PORTB Interrupt Flag RBIF (PIR<7>).

This interrupt can wake the device from SLEEP. The user, in the interrupt service routine, can clear the interrupt by:

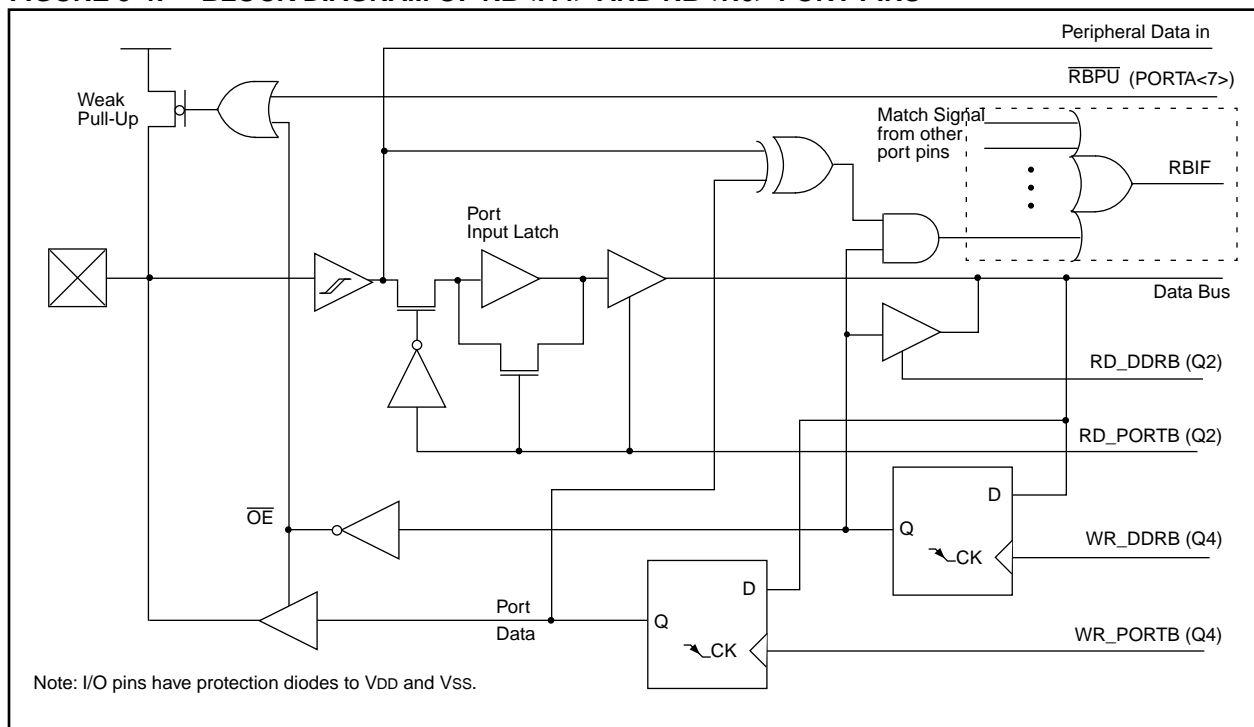
- Read-Write PORTB (such as; `MOVVPF PORTB, PORTB`). This will end mismatch condition.
- Then, clear the RBIF bit.

A mismatch condition will continue to set the RBIF bit. Reading then writing PORTB will end the mismatch condition, and allow the RBIF bit to be cleared.

This interrupt on mismatch feature, together with software configurable pull-ups on this port, allows easy interface to a key pad and make it possible for wake-up on key-depression. For an example, refer to AN552 in the *Embedded Control Handbook*.

The interrupt on change feature is recommended for wake-up on operations where PORTB is only used for the interrupt on change feature and key depression operation.

FIGURE 9-4: BLOCK DIAGRAM OF RB<7:4> AND RB<1:0> PORT PINS



11.3 Read/Write Consideration for TMR0

Although TMR0 is a 16-bit timer/counter, only 8-bits at a time can be read or written during a single instruction cycle. Care must be taken during any read or write.

11.3.1 READING 16-BIT VALUE

The problem in reading the entire 16-bit value is that after reading the low (or high) byte, its value may change from FFh to 00h.

Example 11-1 shows a 16-bit read. To ensure a proper read, interrupts must be disabled during this routine.

EXAMPLE 11-1: 16-BIT READ

```

MOVFP  TMR0L, TMPLO    ;read low tmr0
MOVFP  TMR0H, TMPHI    ;read high tmr0
MOVFP  TMPLO, WREG      ;tmplo -> wreg
CPFSLT TMR0L           ;tmr0l < wreg?
RETURN                ;no then return
MOVFP  TMR0L, TMPLO    ;read low tmr0
MOVFP  TMR0H, TMPHI    ;read high tmr0
RETURN                ;return
    
```

11.3.2 WRITING A 16-BIT VALUE TO TMR0

Since writing to either TMR0L or TMR0H will effectively inhibit increment of that half of the TMR0 in the next cycle (following write), but not inhibit increment of the other half, the user must write to TMR0L first and TMR0H next in two consecutive instructions, as shown in Example 11-2. The interrupt must be disabled. Any write to either TMR0L or TMR0H clears the prescaler.

EXAMPLE 11-2: 16-BIT WRITE

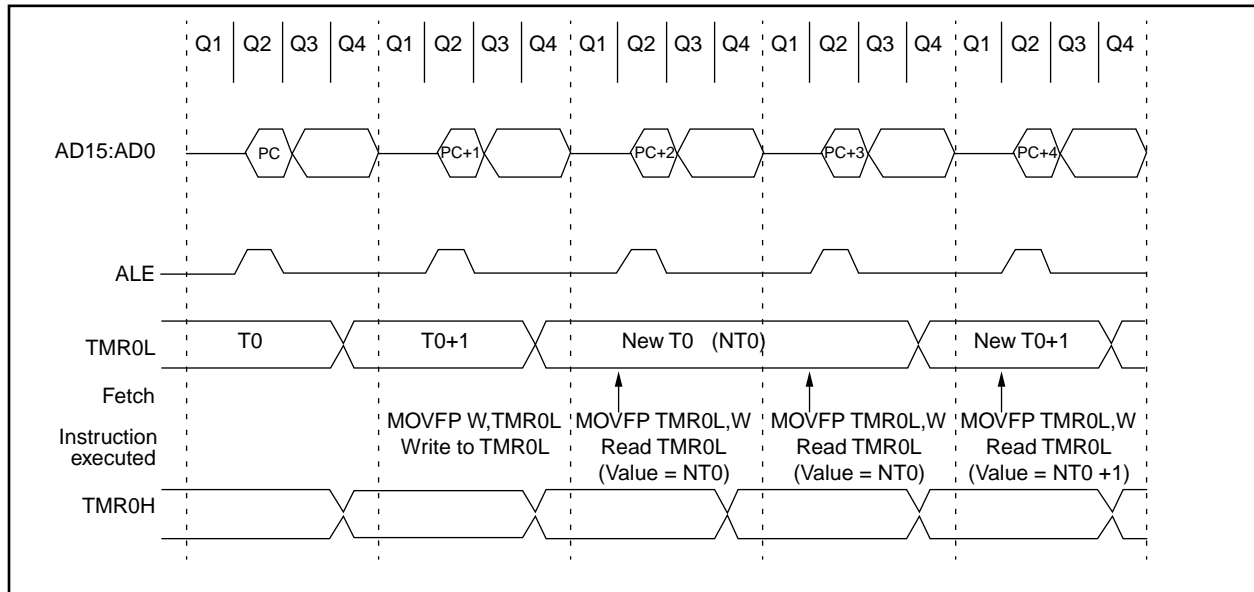
```

BSF    CPUSTA, GLINTD ; Disable interrupt
MOVFP  RAM_L, TMR0L   ;
MOVFP  RAM_H, TMR0H   ;
BCF    CPUSTA, GLINTD ; Done, enable interrupt
    
```

11.4 Prescaler Assignments

Timer0 has an 8-bit prescaler. The prescaler assignment is fully under software control; i.e., it can be changed “on the fly” during program execution. When changing the prescaler assignment, clearing the prescaler is recommended before changing assignment. The value of the prescaler is “unknown,” and assigning a value that is less than the present value makes it difficult to take this unknown time into account.

FIGURE 11-4: TMR0 TIMING: WRITE HIGH OR LOW BYTE



12.1 Timer1 and Timer2

12.1.1 TIMER1, TIMER2 IN 8-BIT MODE

Both Timer1 and Timer2 will operate in 8-bit mode when the T16 bit is clear. These two timers can be independently configured to increment from the internal instruction cycle clock or from an external clock source on the RB4/TCLK12 pin. The timer clock source is configured by the TMRxCS bit (x = 1 for Timer1 or = 2 for Timer2). When TMRxCS is clear, the clock source is internal and increments once every instruction cycle ($F_{osc}/4$). When TMRxCS is set, the clock source is the RB4/TCLK12 pin, and the timer will increment on every falling edge of the RB4/TCLK12 pin.

The timer increments from 00h until it equals the Period register (PRx). It then resets to 00h at the next increment cycle. The timer interrupt flag is set when the timer is reset. TMR1 and TMR2 have individual interrupt flag bits. The TMR1 interrupt flag bit is latched into TMR1IF, and the TMR2 interrupt flag bit is latched into TMR2IF.

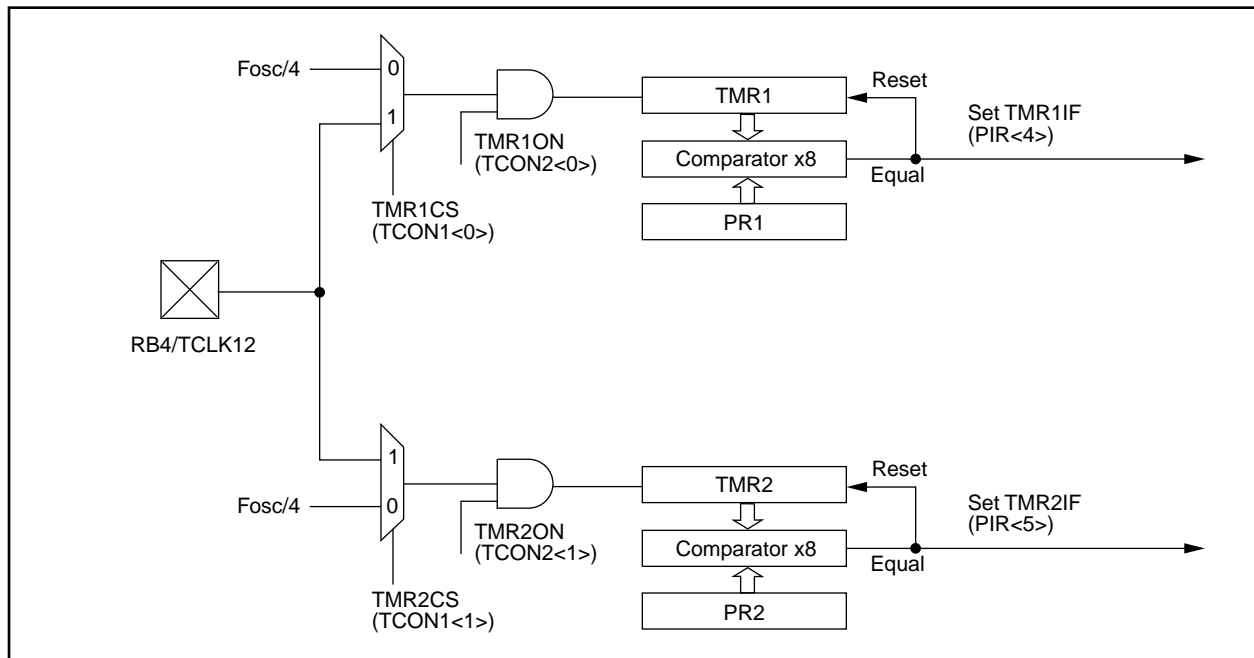
Each timer also has a corresponding interrupt enable bit (TMRxIE). The timer interrupt can be enabled by setting this bit and disabled by clearing this bit. For peripheral interrupts to be enabled, the Peripheral Interrupt Enable bit must be enabled (PEIE is set) and global interrupts must be enabled (GLINTD is cleared).

The timers can be turned on and off under software control. When the Timerx On control bit (TMRxON) is set, the timer increments from the clock source. When TMRxON is cleared, the timer is turned off and cannot cause the timer interrupt flag to be set.

12.1.1.1 EXTERNAL CLOCK INPUT FOR TIMER1 OR TIMER2

When TMRxCS is set, the clock source is the RB4/TCLK12 pin, and the timer will increment on every falling edge on the RB4/TCLK12 pin. The TCLK12 input is synchronized with internal phase clocks. This causes a delay from the time a falling edge appears on TCLK12 to the time TMR1 or TMR2 is actually incremented. For the external clock input timing requirements, see the Electrical Specification section.

FIGURE 12-3: TIMER1 AND TIMER2 IN TWO 8-BIT TIMER/COUNTER MODE



PIC17C4X

Steps to follow when setting up an Asynchronous Reception:

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 RCIE bit.
4. If 9-bit reception is desired, then set the RX9 bit.
5. Enable the reception by setting the CREN bit.
6. The RCIF bit will be set when reception completes and an interrupt will be generated if the RCIE bit was set.
7. Read RCSTA to get the ninth bit (if enabled) and FERR bit to determine if any error occurred during reception.
8. Read RCREG for the 8-bit received data.
9. If an overrun error occurred, clear the error by clearing the OERR bit.

Note: To terminate a reception, either clear the SREN and CREN bits, or the SPEN bit. This will reset the receive logic, so that it will be in the proper state when receive is re-enabled.

FIGURE 13-8: ASYNCHRONOUS RECEPTION

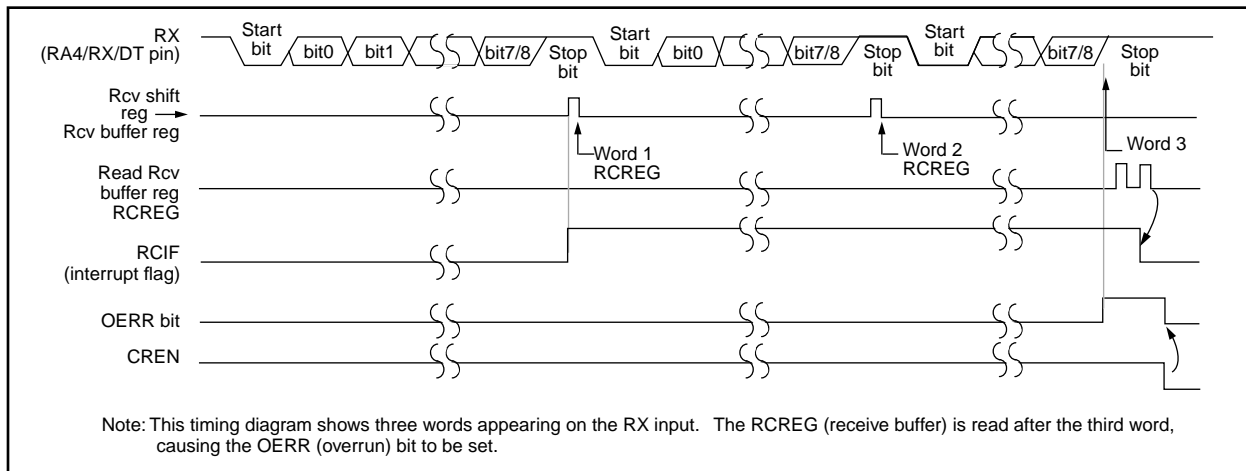


TABLE 13-6: REGISTERS ASSOCIATED WITH ASYNCHRONOUS RECEPTION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets (Note1)
16h, Bank 1	PIR	RBIF	TMR3IF	TMR2IF	TMR1IF	CA2IF	CA1IF	TXIF	RCIF	0000 0010	0000 0010
13h, Bank 0	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00u
14h, Bank 0	RCREG	RX7	RX6	RX5	RX4	RX3	RX2	RX1	RX0	xxxx xxxx	uuuu uuuu
17h, Bank 1	PIE	RBIE	TMR3IE	TMR2IE	TMR1IE	CA2IE	CA1IE	TXIE	RCIE	0000 0000	0000 0000
15h, Bank 0	TXSTA	CSRC	TX9	TXEN	SYNC	—	—	TRMT	TX9D	0000 --1x	0000 --1u
17h, Bank 0	SPBRG	Baud rate generator register								xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented read as a '0', shaded cells are not used for asynchronous reception.

Note 1: Other (non power-up) resets include: external reset through MCLR and Watchdog Timer Reset.

BSF	Bit Set f				
Syntax:	[<i>label</i>] BSF f,b				
Operands:	$0 \leq f \leq 255$ $0 \leq b \leq 7$				
Operation:	$1 \rightarrow (f < b)$				
Status Affected:	None				
Encoding:	<table><tr><td>1000</td><td>0bbb</td><td>ffff</td><td>ffff</td></tr></table>	1000	0bbb	ffff	ffff
1000	0bbb	ffff	ffff		
Description:	Bit 'b' in register 'f' is set.				
Words:	1				
Cycles:	1				
Q Cycle Activity:					

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Execute	Write register 'f'

Example: BSF FLAG_REG, 7

Before Instruction
FLAG_REG= 0x0A

After Instruction
FLAG_REG= 0x8A

BTFSC		Bit Test, skip if Clear							
Syntax:	[<i>label</i>] BTFSC f,b								
Operands:	$0 \leq f \leq 255$ $0 \leq b \leq 7$								
Operation:	skip if (f) = 0								
Status Affected:	None								
Encoding:	<table border="1"><tr><td>1001</td><td>1bbb</td><td>ffff</td><td>ffff</td></tr></table>					1001	1bbb	ffff	ffff
1001	1bbb	ffff	ffff						
Description:	<p>If bit 'b' in register 'f' is 0 then the next instruction is skipped.</p> <p>If bit 'b' is 0 then the next instruction fetched during the current instruction execution is discarded, and a NOP is executed instead, making this a two-cycle instruction.</p>								

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Execute	NOP

If skip:

Q1	Q2	Q3	Q4
Forced NOP	NOP	Execute	NOP

Example: HERE BTFSC FLAG, 1
FALSE :
TRUE :

Before Instruction
PC = address (HERE)

After Instruction
If FLAG<1> = 0;
PC = address (TRUE)
If FLAG<1> = 1;
PC = address (FALSE)

MULLW		Multiply Literal with WREG			
Syntax:	[<i>label</i>] MULLW k				
Operands:	0 ≤ k ≤ 255				
Operation:	(k x WREG) → PRODH:PRODL				
Status Affected:	None				
Encoding:	1011	1100	kkkk	kkkk	
Description:	<p>An unsigned multiplication is carried out between the contents of WREG and the 8-bit literal 'k'. The 16-bit result is placed in PRODH:PRODL register pair. PRODH contains the high byte.</p> <p>WREG is unchanged.</p> <p>None of the status flags are affected.</p> <p>Note that neither overflow nor carry is possible in this operation. A zero result is possible but not detected.</p>				
Words:	1				
Cycles:	1				

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Execute	Write registers PRODH: PRODL

Example: MULLW 0xC4

Before Instruction

WREG = 0xE2
 PRODH = ?
 PRODL = ?

After Instruction

WREG = 0xC4
 PRODH = 0xAD
 PRODL = 0x08

Note: This instruction is not available in the PIC17C42 device.

MULWF		Multiply WREG with f						
Syntax:	[<i>label</i>] MULWF f							
Operands:	$0 \leq f \leq 255$							
Operation:	$(\text{WREG} \times f) \rightarrow \text{PRODH:PRODL}$							
Status Affected:	None							
Encoding:	<table border="1"><tr><td>0011</td><td>0100</td><td>ffff</td><td>ffff</td></tr></table>				0011	0100	ffff	ffff
0011	0100	ffff	ffff					
Description:	<p>An unsigned multiplication is carried out between the contents of WREG and the register file location 'f'. The 16-bit result is stored in the PRODH:PRODL register pair. PRODH contains the high byte.</p> <p>Both WREG and 'f' are unchanged.</p> <p>None of the status flags are affected.</p> <p>Note that neither overflow nor carry is possible in this operation. A zero result is possible but not detected.</p>							
Words:	1							
Cycles:	1							

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Execute	Write registers PRODH: PRODL

Example: MULWF REG

Before Instruction

WREG = 0xC4
 REG = 0xB5
 PRODH = ?
 PRODL = ?

After Instruction

WREG = 0xC4
 REG = 0xB5
 PRODH = 0x8A
 PRODL = 0x94

Note: This instruction is not available in the PIC17C42 device.

PIC17C4X

RRNCF Rotate Right f (no carry)

Syntax: [label] RRNCF f,d
 Operands: $0 \leq f \leq 255$
 $d \in [0,1]$

Operation: $f \langle n \rangle \rightarrow d \langle n-1 \rangle$;
 $f \langle 0 \rangle \rightarrow d \langle 7 \rangle$

Status Affected: None

Encoding:

0010	000d	ffff	ffff
------	------	------	------

Description: The contents of register 'f' are rotated one bit to the right. 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'	Execute	Write to destination

Example 1: RRNCF REG, 1

Before Instruction

WREG = ?
 REG = 1101 0111

After Instruction

WREG = 0
 REG = 1110 1011

Example 2: RRNCF REG, 0

Before Instruction

WREG = ?
 REG = 1101 0111

After Instruction

WREG = 1110 1011
 REG = 1101 0111

SETF Set f

Syntax: [label] SETF f,s
 Operands: $0 \leq f \leq 255$
 $s \in [0,1]$

Operation: $FFh \rightarrow f$;
 $FFh \rightarrow d$

Status Affected: None

Encoding:

0010	101s	ffff	ffff
------	------	------	------

Description: If 's' is 0, both the data memory location 'f' and WREG are set to FFh. If 's' is 1 only the data memory location 'f' is set to FFh.

Words: 1
 Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Execute	Write register 'f' and other specified register

Example1: SETF REG, 0

Before Instruction

REG = 0xDA
 WREG = 0x05

After Instruction

REG = 0xFF
 WREG = 0xFF

Example2: SETF REG, 1

Before Instruction

REG = 0xDA
 WREG = 0x05

After Instruction

REG = 0xFF
 WREG = 0x05

SLEEP	Enter SLEEP mode				
Syntax:	[<i>label</i>] SLEEP				
Operands:	None				
Operation:	00h → WDT; 0 → WDT postscaler; 1 → \overline{TO} ; 0 → \overline{PD}				
Status Affected:	\overline{TO} , \overline{PD}				
Encoding:	<table><tr><td>0000</td><td>0000</td><td>0000</td><td>0011</td></tr></table>	0000	0000	0000	0011
0000	0000	0000	0011		
Description:	<p>The power down status bit (\overline{PD}) is cleared. The time-out status bit (\overline{TO}) is set. Watchdog Timer and its prescaler are cleared.</p> <p>The processor is put into SLEEP mode with the oscillator stopped.</p>				
Words:	1				
Cycles:	1				

Q Cycle Activity:	Q1	Q2	Q3	Q4
	Decode	Read register PCLATH	Execute	NOP

Example: SLEEP

Before Instruction

\overline{TO} = ?

\overline{PD} = ?

After Instruction

\overline{TO} = 1 †

\overline{PD} = 0

† If WDT causes wake-up, this bit is cleared

SUBLW	Subtract WREG from Literal				
Syntax:	[<i>label</i>] SUBLW k				
Operands:	$0 \leq k \leq 255$				
Operation:	$k - (WREG) \rightarrow (WREG)$				
Status Affected:	OV, C, DC, Z				
Encoding:	<table><tr><td>1011</td><td>0010</td><td>kkkk</td><td>kkkk</td></tr></table>	1011	0010	kkkk	kkkk
1011	0010	kkkk	kkkk		
Description:	WREG is subtracted from the eight bit literal 'k'. The result is placed in WREG.				
Words:	1				
Cycles:	1				

Q Cycle Activity:

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

Example 1: SUBLW 0x02

Before Instruction

WREG = 1

C = ?

After Instruction

WREG = 1

C = 1 ; result is positive

Z = 0

Example 2:

Before Instruction

WREG = 2

C = ?

After Instruction

WREG = 0

C = 1 ; result is zero

Z = 1

Example 3:

Before Instruction

WREG = 3

C = ?

After Instruction

WREG = FF ; (2's complement)

C = 0 ; result is negative

Z = 1

XORLW		Exclusive OR Literal with WREG							
Syntax:	[<i>label</i>] XORLW k								
Operands:	0 ≤ k ≤ 255								
Operation:	(WREG) .XOR. k → (WREG)								
Status Affected:	Z								
Encoding:	<table><tr><td>1011</td><td>0100</td><td>kkkk</td><td>kkkk</td></tr></table>					1011	0100	kkkk	kkkk
1011	0100	kkkk	kkkk						
Description:	The contents of WREG are XOR'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'	Execute	Write to WREG					

Example: XORLW 0xAF

Before Instruction
WREG = 0xB5

After Instruction
WREG = 0x1A

XORWF		Exclusive OR WREG with f						
Syntax:	[<i>label</i>] XORWF f,d							
Operands:	0 ≤ f ≤ 255 d ∈ [0,1]							
Operation:	(WREG) .XOR. (f) → (dest)							
Status Affected:	Z							
Encoding:	<table><tr><td>0000</td><td>110d</td><td>ffff</td><td>ffff</td></tr></table>				0000	110d	ffff	ffff
0000	110d	ffff	ffff					
Description:	Exclusive OR the contents of WREG with register 'f'. If 'd' is 0 the result is stored in WREG. If 'd' is 1 the result is stored back in the register 'f'.							
Words:	1							
Cycles:	1							
Q Cycle Activity:								
	Q1	Q2	Q3	Q4				
	Decode	Read register 'f'	Execute	Write to destination				

Example: XORWF REG, 1

Before Instruction
REG = 0xAF
WREG = 0xB5

After Instruction
REG = 0x1A
WREG = 0xB5

16.0 DEVELOPMENT SUPPORT

16.1 Development Tools

The PIC16/17 microcontrollers are supported with a full range of hardware and software development tools:

- PICMASTER/PICMASTER CE Real-Time In-Circuit Emulator
- ICEPIC Low-Cost PIC16C5X and PIC16CXXX In-Circuit Emulator
- PRO MATE® II Universal Programmer
- PICSTART® Plus Entry-Level Prototype Programmer
- PICDEM-1 Low-Cost Demonstration Board
- PICDEM-2 Low-Cost Demonstration Board
- PICDEM-3 Low-Cost Demonstration Board
- MPASM Assembler
- MPLAB-SIM Software Simulator
- MPLAB-C (C Compiler)
- Fuzzy logic development system (fuzzyTECH®-MP)

16.2 PICMASTER: High Performance Universal In-Circuit Emulator with MPLAB IDE

The PICMASTER Universal In-Circuit Emulator is intended to provide the product development engineer with a complete microcontroller design tool set for all microcontrollers in the PIC12C5XX, PIC14000, PIC16C5X, PIC16CXXX and PIC17CXX families. PICMASTER is supplied with the MPLAB™ Integrated Development Environment (IDE), which allows editing, "make" and download, and source debugging from a single environment.

Interchangeable target probes allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the PICMASTER allows expansion to support all new Microchip microcontrollers.

The PICMASTER Emulator System has been designed as a real-time emulation system with advanced features that are generally found on more expensive development tools. The PC compatible 386 (and higher) machine platform and Microsoft Windows® 3.x environment were chosen to best make these features available to you, the end user.

A CE compliant version of PICMASTER is available for European Union (EU) countries.

16.3 ICEPIC: Low-cost PIC16CXXX In-Circuit Emulator

ICEPIC is a low-cost in-circuit emulator solution for the Microchip PIC16C5X and PIC16CXXX families of 8-bit OTP microcontrollers.

ICEPIC is designed to operate on PC-compatible machines ranging from 286-AT® through Pentium™ based machines under Windows 3.x environment. ICEPIC features real time, non-intrusive emulation.

16.4 PRO MATE II: Universal Programmer

The PRO MATE II Universal Programmer is a full-featured programmer capable of operating in stand-alone mode as well as PC-hosted mode.

The PRO MATE II has programmable VDD and VPP supplies which allows it to verify programmed memory at VDD min and VDD max for maximum reliability. It has an LCD display for displaying error messages, keys to enter commands and a modular detachable socket assembly to support various package types. In stand-alone mode the PRO MATE II can read, verify or program PIC16C5X, PIC16CXXX, PIC17CXX and PIC14000 devices. It can also set configuration and code-protect bits in this mode.

16.5 PICSTART Plus Entry Level Development System

The PICSTART programmer is an easy-to-use, low-cost prototype programmer. It connects to the PC via one of the COM (RS-232) ports. MPLAB Integrated Development Environment software makes using the programmer simple and efficient. PICSTART Plus is not recommended for production programming.

PICSTART Plus supports all PIC12C5XX, PIC14000, PIC16C5X, PIC16CXXX and PIC17CXX devices with up to 40 pins. Larger pin count devices such as the PIC16C923 and PIC16C924 may be supported with an adapter socket.

PIC17C4X

Applicable Devices 42 R42 42A 43 R43 44

FIGURE 17-5: TIMER0 CLOCK TIMINGS

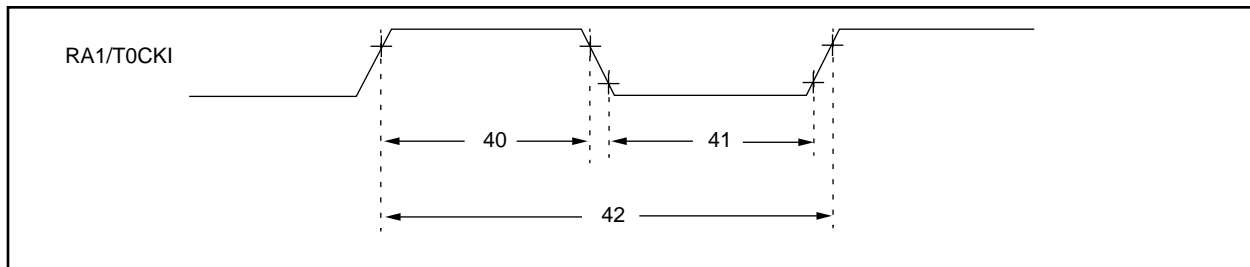


TABLE 17-5: TIMER0 CLOCK REQUIREMENTS

Parameter No.	Sym	Characteristic		Min	Typ†	Max	Units	Conditions
40	Tt0H	T0CKI High Pulse Width	No Prescaler	0.5TCY + 20 §	—	—	ns	
			With Prescaler	10*	—	—	ns	
41	Tt0L	T0CKI Low Pulse Width	No Prescaler	0.5TCY + 20 §	—	—	ns	
			With Prescaler	10*	—	—	ns	
42	Tt0P	T0CKI Period		$\frac{TCY + 40}{N}$ §	—	—	ns	N = prescale value (1, 2, 4, ..., 256)

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

§ This specification ensured by design.

FIGURE 17-6: TIMER1, TIMER2, AND TIMER3 CLOCK TIMINGS

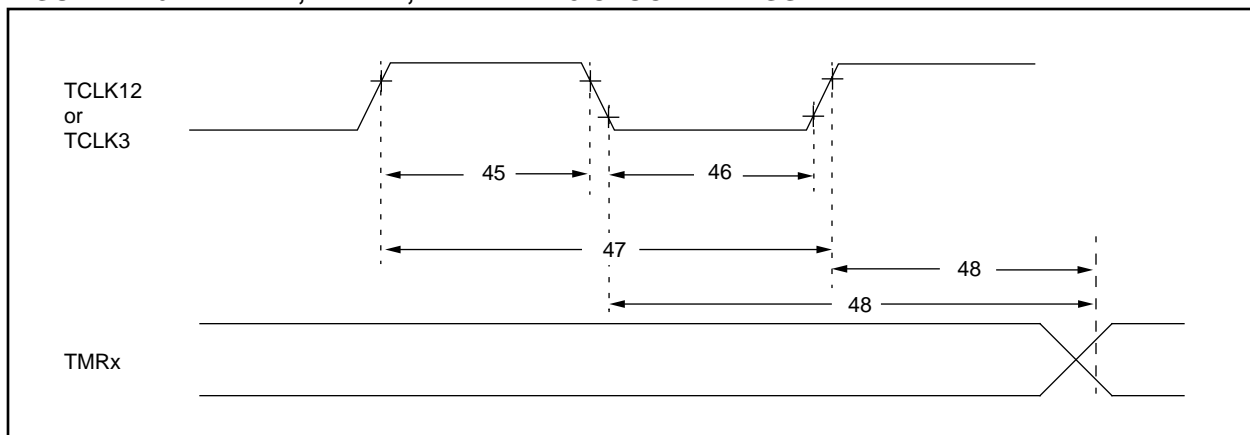


TABLE 17-6: TIMER1, TIMER2, AND TIMER3 CLOCK REQUIREMENTS

Parameter No.	Sym	Characteristic	Min	Typ †	Max	Units	Conditions
45	Tt123H	TCLK12 and TCLK3 high time	0.5 Tcy + 20 §	—	—	ns	
46	Tt123L	TCLK12 and TCLK3 low time	0.5 Tcy + 20 §	—	—	ns	
47	Tt123P	TCLK12 and TCLK3 input period	$\frac{Tcy + 40}{N}$ §	—	—	ns	N = prescale value (1, 2, 4, 8)
48	TckE2tmrl	Delay from selected External Clock Edge to Timer increment	2Tosc §	—	6 Tosc §	—	

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

§ This specification ensured by design.

PIC17C4X

Applicable Devices 42 R42 42A 43 R43 44

FIGURE 18-13: WDT TIMER TIME-OUT PERIOD vs. VDD

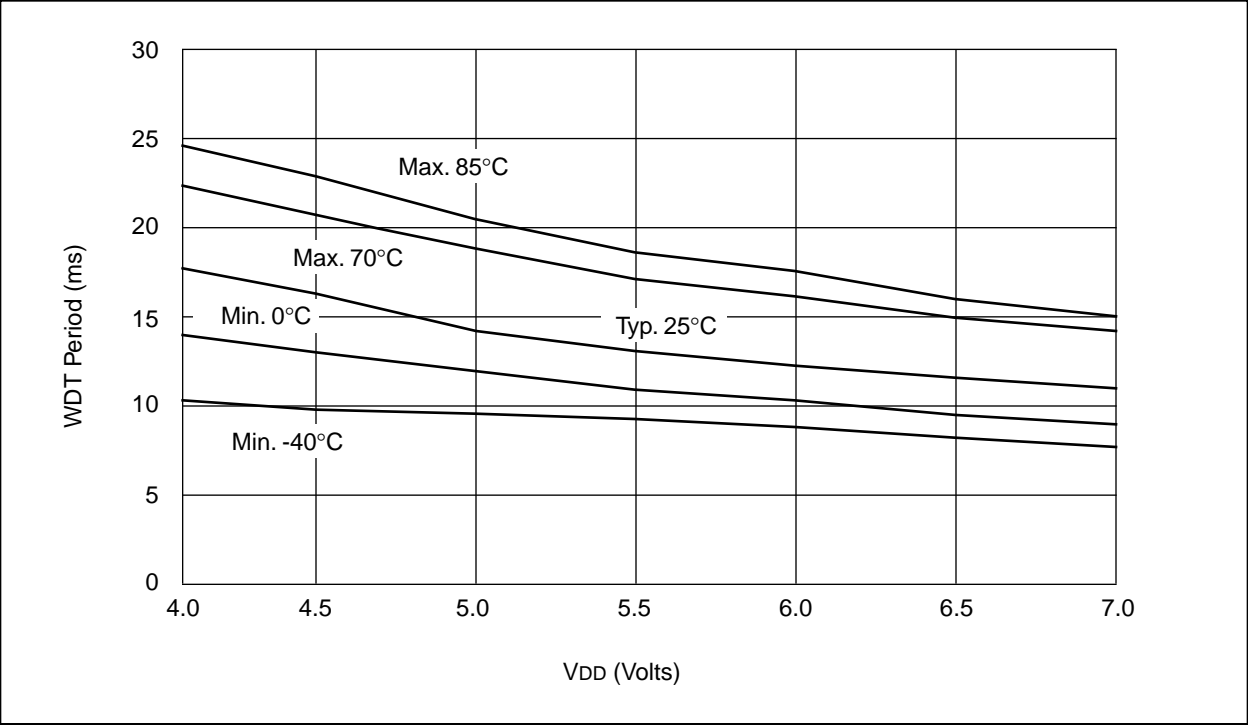


FIGURE 18-14: IOH vs. VOH, VDD = 3V

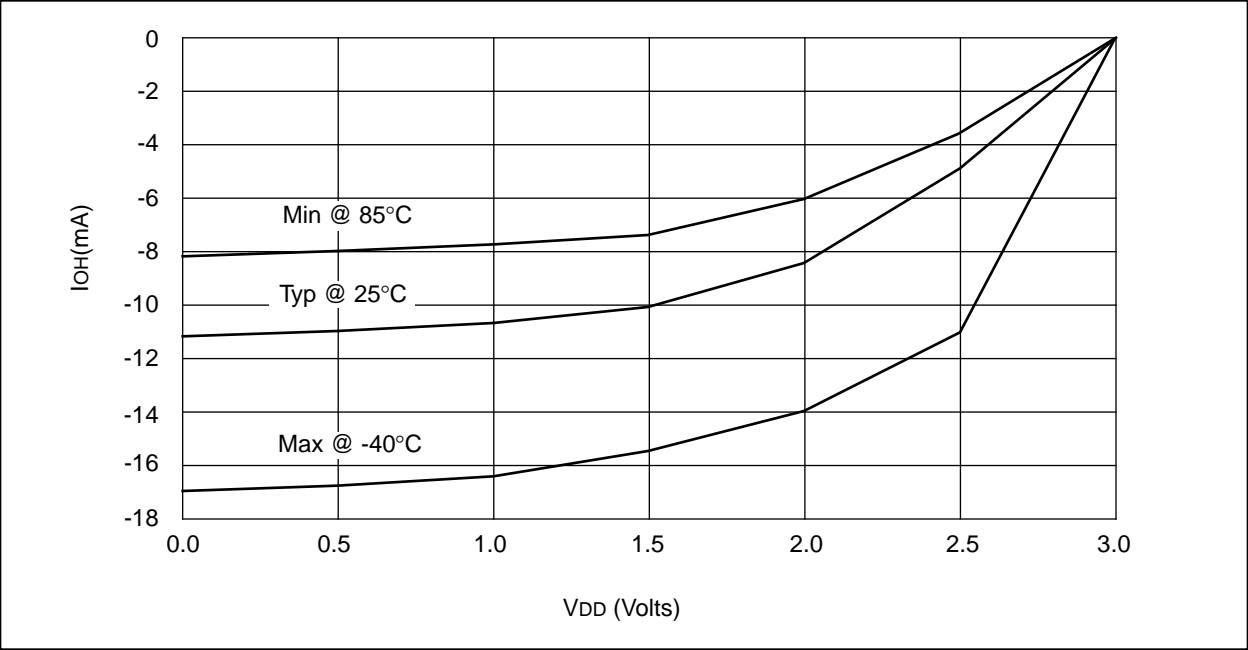


FIGURE 18-15: I_{OH} vs. V_{OH} , $V_{DD} = 5V$

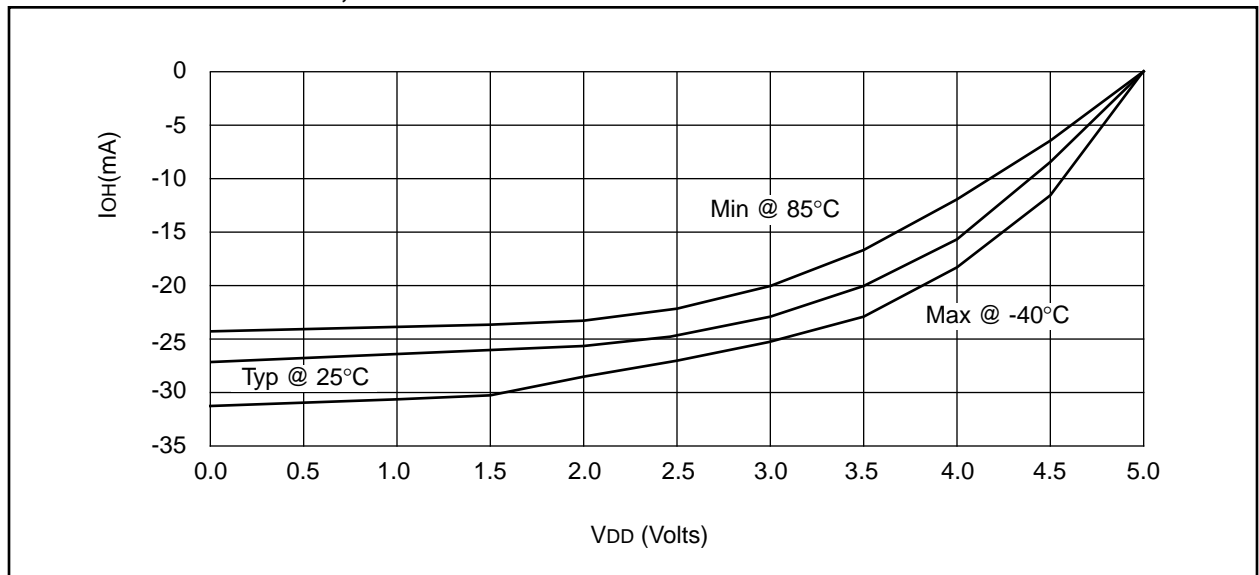
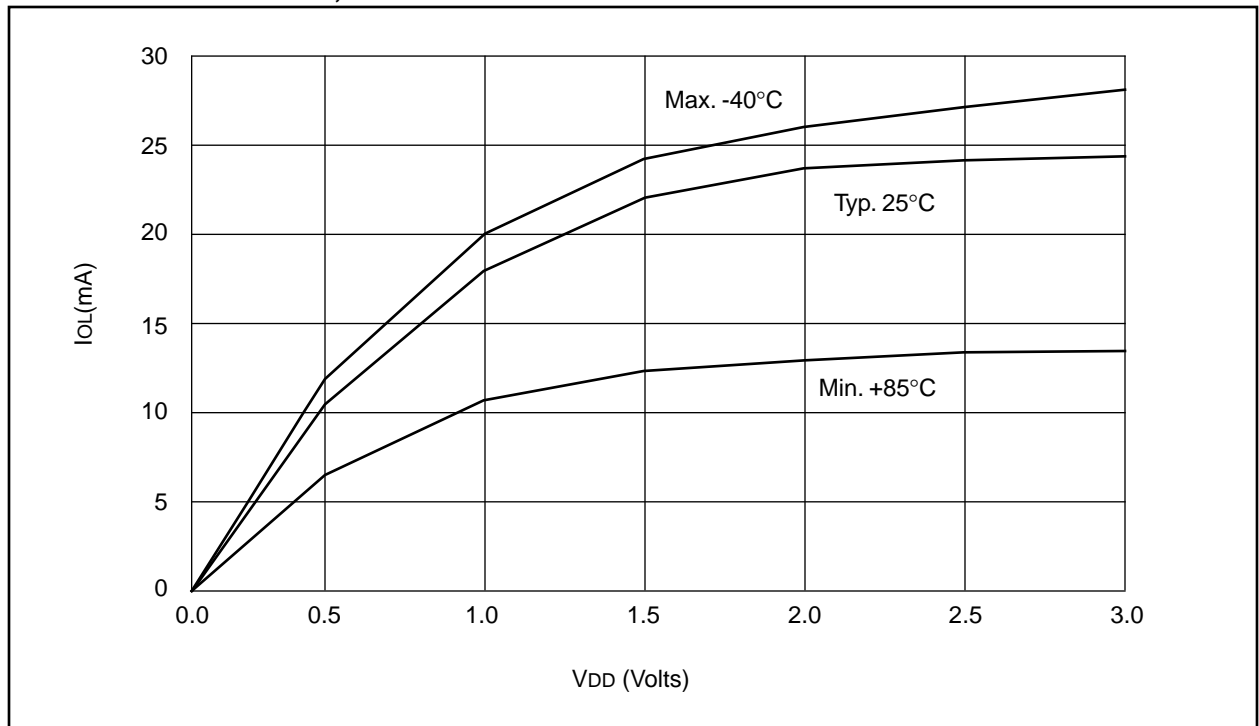


FIGURE 18-16: I_{OL} vs. V_{OL} , $V_{DD} = 3V$



19.3 DC CHARACTERISTICS: **PIC17CR42/42A/43/R43/44-16 (Commercial, Industrial)**
PIC17CR42/42A/43/R43/44-25 (Commercial, Industrial)
PIC17CR42/42A/43/R43/44-33 (Commercial, Industrial)
PIC17LCR42/42A/43/R43/44-08 (Commercial, Industrial)

Standard Operating Conditions (unless otherwise stated)							
Operating temperature							
DC CHARACTERISTICS $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ for industrial and $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ for commercial Operating voltage VDD range as described in Section 19.1							
Parameter No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
D030	VIL	Input Low Voltage					
		I/O ports					
		with TTL buffer	VSS	–	0.8	V	$4.5\text{V} \leq V_{DD} \leq 5.5\text{V}$
		with Schmitt Trigger buffer	VSS	–	0.2VDD	V	$2.5\text{V} \leq V_{DD} \leq 4.5\text{V}$
D031			VSS	–	0.2VDD	V	
D032		MCLR, OSC1 (in EC and RC mode)	VSS	–	0.2VDD	V	Note1
D033		OSC1 (in XT, and LF mode)	–	0.5VDD	–	V	
D040	VIH	Input High Voltage					
		I/O ports					
		with TTL buffer	2.0	–	VDD	V	$4.5\text{V} \leq V_{DD} \leq 5.5\text{V}$
		with Schmitt Trigger buffer	$1 + 0.2V_{DD}$	–	VDD	V	$2.5\text{V} \leq V_{DD} \leq 4.5\text{V}$
		MCLR	0.8VDD	–	VDD	V	Note1
D041			0.8VDD	–	VDD	V	
D042		MCLR	0.8VDD	–	VDD	V	Note1
D043		OSC1 (XT, and LF mode)	–	0.5VDD	–	V	
D050	VHYS	Hysteresis of Schmitt Trigger inputs	$0.15V_{DD}^*$	–	–	V	
D060	IIL	Input Leakage Current (Notes 2, 3)					
		I/O ports (except RA2, RA3)	–	–	± 1	μA	$V_{SS} \leq V_{PIN} \leq V_{DD}$, I/O Pin at hi-impedance PORTB weak pull-ups disabled
		MCLR	–	–	± 2	μA	$V_{PIN} = V_{SS}$ or $V_{PIN} = V_{DD}$
		RA2, RA3	–	–	± 2	μA	$V_{SS} \leq V_{RA2}, V_{RA3} \leq 12\text{V}$
		OSC1, TEST (EC, RC modes)	–	–	± 1	μA	$V_{SS} \leq V_{PIN} \leq V_{DD}$
		OSC1, TEST (XT, LF modes)	–	–	V _{PIN}	μA	$R_F \geq 1\text{ M}\Omega$, see Figure 14.2
		MCLR	–	–	10	μA	$V_{MCLR} = V_{PP} = 12\text{V}$ (when not programming)
D064			–	–	10	μA	
D070	IPURB	PORTB weak pull-up current	60	200	400	μA	$V_{PIN} = V_{SS}$, $\overline{\text{RBP}} = 0$ $4.5\text{V} \leq V_{DD} \leq 6.0\text{V}$

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

‡ These parameters are for design guidance only and are not tested, nor characterized.

Note 1: In RC oscillator configuration, the OSC1/CLKIN pin is a Schmitt Trigger input. It is not recommended that the PIC17CXX devices be driven with external clock in RC mode.

2: The leakage current on the MCLR pin is strongly dependent on the applied voltage level. The specified levels represent normal operating conditions. Higher leakage current may be measured at different input voltages.

3: Negative current is defined as coming out of the pin.

4: These specifications are for the programming of the on-chip program memory EPROM through the use of the table write instructions. The complete programming specifications can be found in: PIC17CXX Programming Specifications (Literature number DS30139).

5: The MCLR/VPP pin may be kept in this range at times other than programming, but is not recommended.

6: For TTL buffers, the better of the two specifications may be used.

PIC17C4X

NOTES: