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	40MHz
Connectivity	CANbus, I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LVD, POR, PWM, WDT
Number of I/O	68
Program Memory Size	32KB (16K x 16)
Program Memory Type	OTP
EEPROM Size	-
RAM Size	1.5K x 8
Voltage - Supply (Vcc/Vdd)	4.2V ~ 5.5V
Data Converters	A/D 16x10b
Oscillator Type	External
Operating Temperature	-40°C ~ 125°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/pic18c858-e-l

3.0 RESET

The PIC18CXX8 differentiates between various kinds of RESET:

a) Power-on Reset (POR)
b) MCLR Reset during normal operation
c) MCLR Reset during SLEEP
d) Watchdog Timer (WDT) Reset (during normal operation)
e) Programmable Brown-out Reset (PBOR)
f) RESET Instruction
g) Stack Full Reset
h) Stack Underflow Reset

Most registers are unaffected by a RESET. Their status is unknown on POR and unchanged by all other RESETs. The other registers are forced to a "RESET" state on Power-on Reset, MCLR, WDT Reset, Brown-out Reset, MCLR Reset during SLEEP and by the RESET instruction.

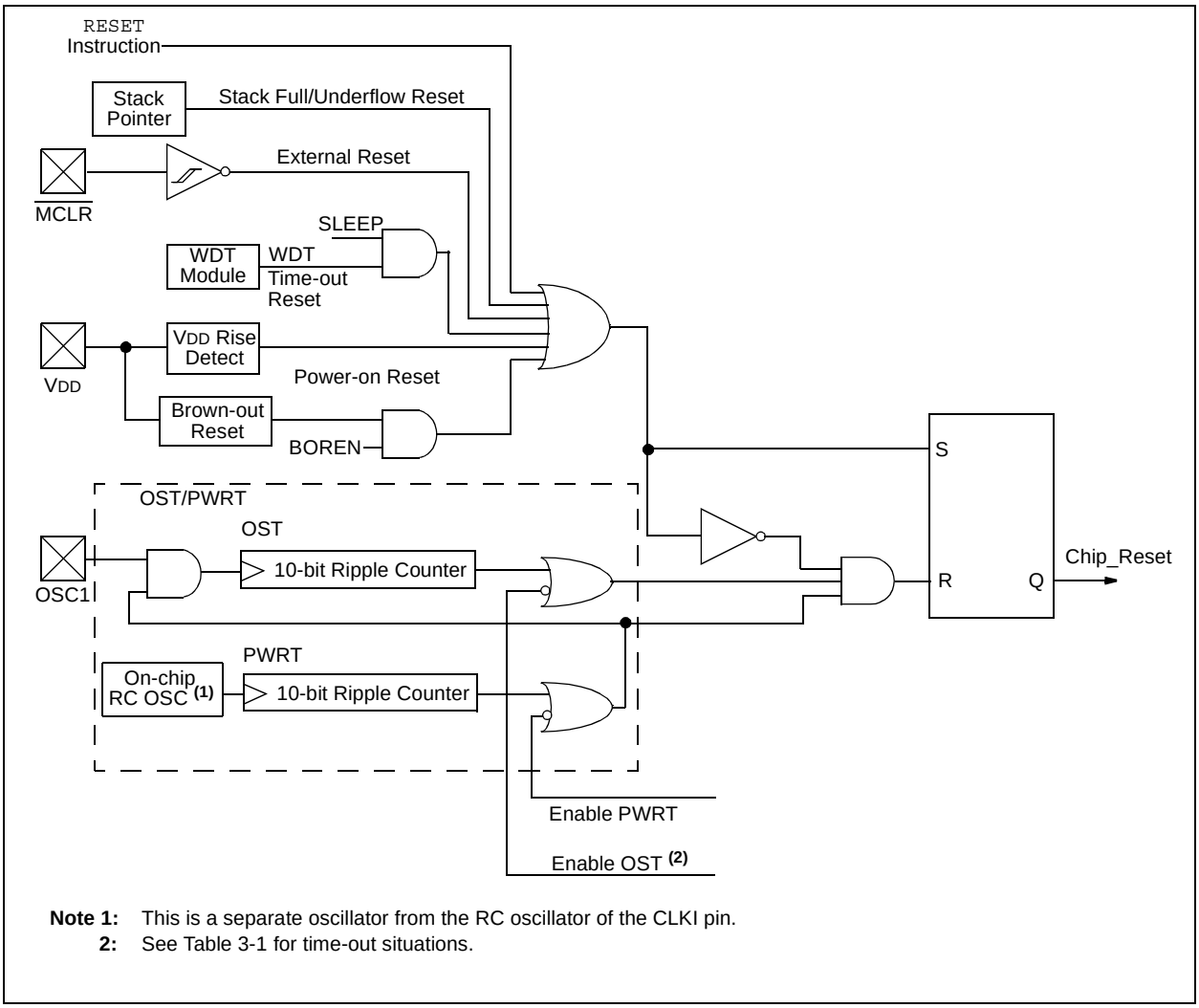
Most registers are not affected by a WDT wake-up, since this is viewed as the resumption of normal operation. Status bits from the RCON register, RI, TO, PD, POR and BOR are set or cleared differently in different RESET situations, as indicated in Table 3-2. These bits are used in software to determine the nature of the RESET. See Table 3-3 for a full description of the RESET states of all registers.

A simplified block diagram of the on-chip RESET circuit is shown in Figure 3-1.

The Enhanced MCU devices have a MCLR noise filter in the MCLR Reset path. The filter will detect and ignore small pulses.

A WDT Reset does not drive MCLR pin low.

FIGURE 3-1: SIMPLIFIED BLOCK DIAGRAM OF ON-CHIP RESET CIRCUIT



Note 1: This is a separate oscillator from the RC oscillator of the CLKI pin.
2: See Table 3-1 for time-out situations.

Filename	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other RESETS(3)
LATJ(4)	Read PORTJ Data Latch, Write PORTJ Data Latch								xxxx xxxx	uuuu uuuu
LATH(4)	Read PORTH Data Latch, Write PORTH Data Latch								xxxx xxxx	uuuu uuuu
LATG	—	—	—	Read PORTG Data Latch, Write PORTG Data Latch					---x xxxx	---u uuuu
LATF	Read PORTF Data Latch, Write PORTF Data Latch								xxxx xxxx	uuuu uuuu
LATE	Read PORTE Data Latch, Write PORTE Data Latch								xxxx xxxx	uuuu uuuu
LATD	Read PORTD Data Latch, Write PORTD Data Latch								xxxx xxxx	uuuu uuuu
LATC	Read PORTC Data Latch, Write PORTC Data Latch								xxxx xxxx	uuuu uuuu
LATB	Read PORTB Data Latch, Write PORTB Data Latch								xxxx xxxx	uuuu uuuu
LATA	—	Bit 6(1)	Read PORTA Data Latch, Write PORTA Data Latch						--xx xxxx	--uu uuuu
PORTJ(4)	Read PORTJ pins, Write PORTJ Data Latch								xxxx xxxx	uuuu uuuu
PORTH(4)	Read PORTH pins, Write PORTH Data Latch								xxxx xxxx	uuuu uuuu
PORTG	—	—	—	Read PORTG pins, Write PORTG Data Latch					---x xxxx	uuuu uuuu
PORTF	Read PORTF pins, Write PORTF Data Latch								0000 0000	0000 0000
PORTE	Read PORTE pins, Write PORTE Data Latch								xxxx xxxx	uuuu uuuu
PORTD	Read PORTD pins, Write PORTD Data Latch								xxxx xxxx	uuuu uuuu
PORTC	Read PORTC pins, Write PORTC Data Latch								xxxx xxxx	uuuu uuuu
PORTB	Read PORTB pins, Write PORTB Data Latch								xxxx xxxx	uuuu uuuu
PORTA	—	Bit 6(1)	Read PORTA pins, Write PORTA Data Latch						--0x 0000	--0u 0000
TRISK(4)	Data Direction Control Register for PORTK								1111 1111	1111 1111
LATK(4)	Read PORTK Data Latch, Write PORTK Data Latch								xxxx xxxx	uuuu uuuu
PORTK(4)	Read PORTK pins, Write PORTK Data Latch								xxxx xxxx	uuuu uuuu
TXERRCNT	TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0	0000 0000	0000 0000
RXERRCNT	REC7	REC6	REC5	REC4	REC3	REC2	REC1	REC0	0000 0000	0000 0000
COMSTAT	RXB0OVFL	RXB1OVFL	TXBO	TXBP	RXBP	TXWARN	RXWARN	EWARN	0000 0000	0000 0000
CIOCON	TX1SRC	TX1EN	ENDRHI	CANCAP	—	—	—	—	1000 ----	1000 ----
BRGCON3	—	WAKFIL	—	—	—	SEG2PH2	SEG2PH1	SEG2PH0	-0-- -000	-0-- -000
BRGCON2	SEG2PHTS	SAM	SEG1PH2	SEG1PH1	SEG1PH0	PRSEG2	PRSEG1	PRSEG0	0000 0000	0000 0000
BRGCON1	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	0000 0000	0000 0000
CANCON	REQOP2	REQOP1	REQOP0	ABAT	WIN2	WIN1	WIN0	—	xxxx xxx-	uuuu uuu-
CANSTAT	OPMODE2	OPMODE1	OPMODE0	—	ICODE2	ICODE1	ICOED0	—	xxx- xxx-	uuu- uuu-

Legend: x = unknown, u = unchanged, - = unimplemented, q = value depends on condition

Note 1: Bit 6 of PORTA, LATA and TRISA are enabled in ECIO and RCIO oscillator modes only. In all other oscillator modes, they are disabled and read '0'.

2: Bit 21 of the TBLPTRU allows access to the device configuration bits.

3: Other (non-power-up) RESETs include external RESET through $\overline{\text{MCLR}}$ and Watchdog Timer Reset.

4: These registers are reserved on PIC18C658.

4.12 Indirect Addressing, INDF and FSR Registers

Indirect addressing is a mode of addressing data memory, where the data memory address in the instruction is not fixed. A SFR register is used as a pointer to the data memory location that is to be read or written. Since this pointer is in RAM, the contents can be modified by the program. This can be useful for data tables in the data memory and for software stacks. Figure 4-6 shows the operation of indirect addressing. This shows the moving of the value to the data memory address specified by the value of the FSR register.

Indirect addressing is possible by using one of the INDF registers. Any instruction using the INDF register actually accesses the register indicated by the File Select Register, FSR. Reading the INDF register itself indirectly (FSR = '0') will read 00h. Writing to the INDF register indirectly results in a no-operation. The FSR register contains a 12-bit address, which is shown in Figure 4-6.

The INDFn ($0 \le n \le 2$) register is not a physical register. Addressing INDFn actually addresses the register whose address is contained in the FSRn register (FSRn is a pointer). This is indirect addressing.

Example 4-4 shows a simple use of indirect addressing to clear the RAM in Bank 1 (locations 100h-1FFh) in a minimum number of instructions.

EXAMPLE 4-4: HOW TO CLEAR RAM (BANK 1) USING INDIRECT ADDRESSING

```
     LFSR     FSR0, 0x100  ;
NEXT CLRF     POSTINC0     ; Clear INDF
                           ; register
                           ; & inc pointer
     BTFSS    FSR0H, 1     ; All done
                           ; w/ Bank1?
     GOTO     NEXT         ; NO, clear next
CONTINUE                   ;
     :                     ; YES, continue
```

There are three indirect addressing registers. To address the entire data memory space (4096 bytes), these registers are 12-bit wide. To store the 12-bits of addressing information, two 8-bit registers are required. These indirect addressing registers are:

1. FSR0: composed of FSR0H:FSR0L
2. FSR1: composed of FSR1H:FSR1L
3. FSR2: composed of FSR2H:FSR2L

In addition, there are registers INDF0, INDF1 and INDF2, which are not physically implemented. Reading or writing to these registers activates indirect addressing, with the value in the corresponding FSR register being the address of the data.

If an instruction writes a value to INDF0, the value will be written to the address indicated by FSR0H:FSR0L. A read from INDF1 reads the data from the address indicated by FSR1H:FSR1L. INDFn can be used in code anywhere an operand can be used.

If INDF0, INDF1 or INDF2 are read indirectly via an FSR, all '0's are read (zero bit is set). Similarly, if INDF0, INDF1 or INDF2 are written to indirectly, the operation will be equivalent to a `NOP` instruction and the STATUS bits are not affected.

4.12.1 INDIRECT ADDRESSING OPERATION

Each FSR register has an INDF register associated with it, plus four additional register addresses. Performing an operation on one of these five registers determines how the FSR will be modified during indirect addressing.

When data access is done to one of the five INDFn locations, the address selected will configure the FSRn register to:

- Do nothing to FSRn after an indirect access (no change) - INDFn
- Auto-decrement FSRn after an indirect access (post-decrement) - POSTDECn
- Auto-increment FSRn after an indirect access (post-increment) - POSTINCn
- Auto-increment FSRn before an indirect access (pre-increment) - PREINCn
- Use the value in the WREG register as an offset to FSRn. Do not modify the value of the WREG or the FSRn register after an indirect access (no change) - PLUSWn

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

Incrementing or decrementing an FSR affects all 12 bits. That is, when FSRnL overflows from an increment, FSRnH will be incremented automatically.

Adding these features allows the FSRn to be used as a software stack pointer in addition to its uses for table operations in data memory.

Each FSR has an address associated with it that performs an indexed indirect access. When a data access to this INDFn location (PLUSWn) occurs, the FSRn is configured to add the 2's complement value in the WREG register and the value in FSR to form the address before an indirect access. The FSR value is not changed.

If an FSR register contains a value that indicates one of the INDFn, an indirect read will read 00h (zero bit is set), while an indirect write will be equivalent to a `NOP` (STATUS bits are not affected).

If an indirect addressing operation is done where the target address is an FSRnH or FSRnL register, the write operation will dominate over the pre- or post-increment/decrement functions.

5.2 Program Memory Read/Writes

5.2.1 TABLE READ OVERVIEW (TBLRD)

The TBLRD instructions are used to read data from program memory to data memory.

TBLPTR points to a byte address in program space. Executing TBLRD places the byte pointed to into TABLAT. In addition, TBLPTR can be modified automatically for the next Table Read operation.

Table Reads from program memory are performed one byte at a time. The instruction will load TABLAT with the one byte from program memory pointed to by TBLPTR.

5.2.2 PROGRAM MEMORY WRITE BLOCK SIZE

The program memory of PIC18CXX8 devices is written in blocks. For PIC18CXX8 devices, the write block size is 2 bytes. Consequently, Table Write operations to program memory are performed in pairs, one byte at a time.

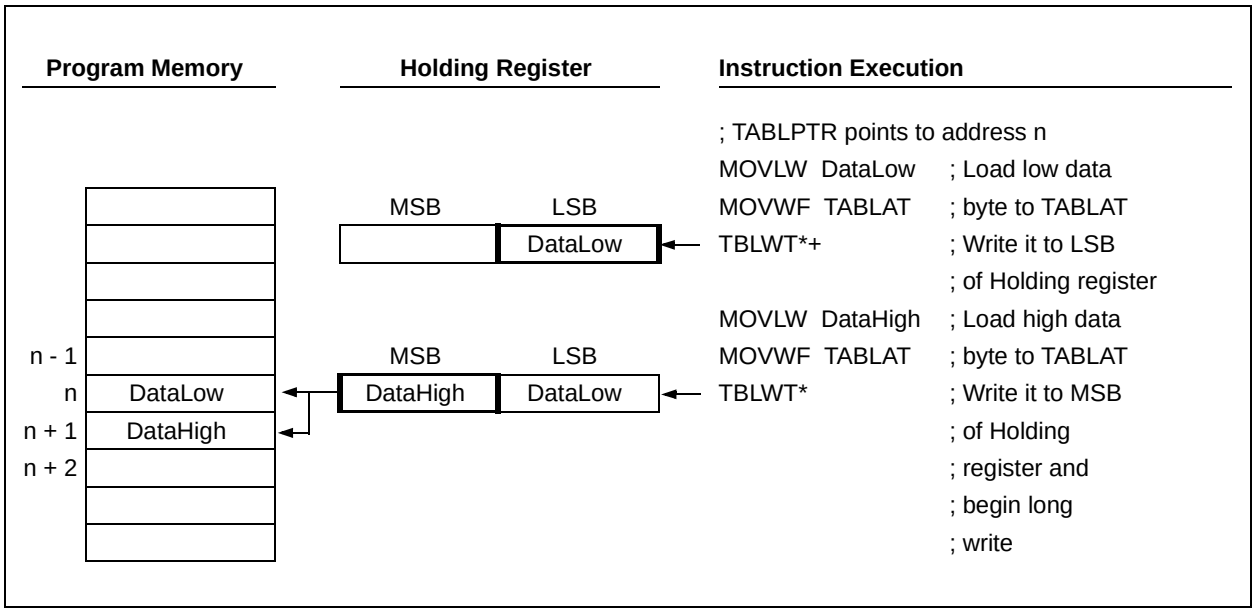
When a Table Write occurs to an even program memory address (TBLPTR<0> = 0), the contents of TABLAT are transferred to an internal holding register. This is performed as a short write and the program memory block is not actually programmed at this time. The holding register is not accessible by the user.

When a Table Write occurs to an odd program memory address (TBLPTR<0> = 1), a long write is started. During the long write, the contents of TABLAT are written to the high byte of the program memory block and the contents of the holding register are transferred to the low byte of the program memory block.

Figure 5-3 shows the holding register and the program memory write blocks.

If a single byte is to be programmed, the low (even) byte of the destination program word should be read using TBLRD*, modified or changed, if required, and written back to the same address using TBLWT*+. The high (odd) byte should be read using TBLRD*, modified or changed if required, and written back to the same address using TBLWT. The write to an odd address will cause a long write to begin. This process ensures that existing data in either byte will not be changed unless desired.

FIGURE 5-3: HOLDING REGISTER AND THE WRITE



EXAMPLE 5-1: TABLE READ CODE EXAMPLE

```
; Read a byte from location 0x0020
CLRF   TBLPTRU          ; Load upper 5 bits of
                        ; 0x0020
CLRF   TBLPTRH          ; Load higher 8 bits of
                        ; 0x0020
MOVLW  0x20             ; Load 0x20 into
MOVWF  TBLPTRL          ; TBLPTRL
MOVWF  TBLRD*           ; Data is in TABLAT
```

 © 2000 Microchip Technology Inc.

REGISTER 7-7: IPR REGISTERS (CONT'D)

IPR2

bit 7 **Unimplemented:** Read as '0'

bit 6 **CMIP:** Comparator Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5-4 **Unimplemented:** Read as '0'

bit 3 **BCLIP**: Bus Collision Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **LVDIP**: Low Voltage Detect Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR3IP**: TMR3 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **CCP2IP**: CCP2 Interrupt Priority bit
1 = High priority
0 = Low priority

IPR3

bit 7 **IVRP:** Invalid Message Received Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **WAKIP:** Bus Activity Wake-up Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **ERRIP:** CAN Bus Error Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **TXB2IP:** Transmit Buffer 2 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **TXB1IP:** Transmit Buffer 1 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **TXB0IP:** Transmit Buffer 0 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **RXB1IP:** Receive Buffer 1 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **RXB0IP:** Receive Buffer 0 Interrupt Priority bit
1 = High priority
0 = Low priority

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

7.1.6 INT INTERRUPTS

External interrupts on the RB0/INT0, RB1/INT1, RB2/INT2, and RB3/INT3 pins are edge triggered: either rising if the corresponding INTEDGx bit is set in the INTCON2 register, or falling, if the INTEDGx bit is clear. When a valid edge appears on the RBx/INTx pin, the corresponding flag bit INTxIF is set. This interrupt can be disabled by clearing the corresponding enable bit INTxIE. Flag bit INTxIF must be cleared in software in the Interrupt Service Routine before re-enabling the interrupt. All external interrupts (INT0, INT1, INT2, and INT3) can wake-up the processor from SLEEP, if bit INTxIE was set prior to going into SLEEP. If the global interrupt enable bit GIE is set, the processor will branch to the interrupt vector following wake-up.

Interrupt priority for INT1, INT2 and INT3 is determined by the value contained in the interrupt priority bits INT1IP (INTCON3 register), INT3IP (INTCON3 register), and INT2IP (INTCON2 register). There is no priority bit associated with INT0; it is always a high priority interrupt source.

7.1.7 TMR0 INTERRUPT

In 8-bit mode (which is the default), an overflow (FFh → 00h) in the TMR0 register will set flag bit TMR0IF. In 16-bit mode, an overflow (FFFFh → 0000h) in the TMR0H:TMR0L registers will set flag bit TMR0IF. The interrupt can be enabled/disabled by setting/clearing enable bit TMR0IE (INTCON register). Interrupt priority for Timer0 is determined by the value contained in the interrupt priority bit TMR0IP (INTCON2 register). See Section 10.0 for further details on the Timer0 module.

7.1.8 PORTB INTERRUPT-ON-CHANGE

An input change on PORTB<7:4> sets flag bit RBIF (INTCON register). The interrupt can be enabled/disabled by setting/clearing enable bit RBIE (INTCON register). Interrupt priority for PORTB interrupt-on-change is determined by the value contained in the interrupt priority bit RBIP (INTCON2 register).

7.2 Context Saving During Interrupts

During an interrupt, the return PC value is saved on the stack. Additionally, the WREG, STATUS and BSR registers are saved on the fast return stack. If a fast return from interrupt is not used (See Section 4.3), the user may need to save the WREG, STATUS and BSR registers in software. Depending on the user's application, other registers may also need to be saved. Example 7-1 saves and restores the WREG, STATUS and BSR registers during an Interrupt Service Routine.

EXAMPLE 7-1: SAVING STATUS, WREG AND BSR REGISTERS IN RAM

```
MOVWF    W_TEMP                    ; W_TEMP is in Low Access bank
MOVFF    STATUS, STATUS_TEMP       ; STATUS_TEMP located anywhere
MOVFF    BSR, BSR_TEMP             ; BSR located anywhere
;
; USER ISR CODE
;
MOVFF   BSR_TEMP, BSR              ; Restore BSR
MOVF    W_TEMP, W                  ; Restore WREG
MOVFF   STATUS_TEMP, STATUS        ; Restore STATUS
```

 © 2000 Microchip Technology Inc.

15.4.5 BAUD RATE GENERATOR

In I^2C Master mode, the reload value for the BRG is located in the lower 7 bits of the SSPADD register (Figure 15-11). When the BRG is loaded with this value, the BRG counts down to 0 and stops until another reload has taken place. The BRG count is decremented twice per instruction cycle (TCY) on the Q2 and Q4 clocks. In I^2C Master mode, the BRG is reloaded automatically. If Clock Arbitration is taking place, for instance, the BRG will be reloaded when the SCL pin is sampled high (Figure 15-12).

FIGURE 15-11: BAUD RATE GENERATOR BLOCK DIAGRAM

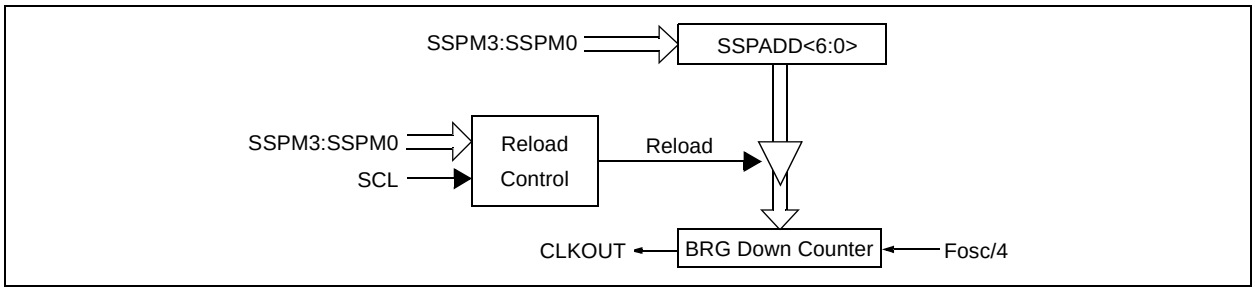
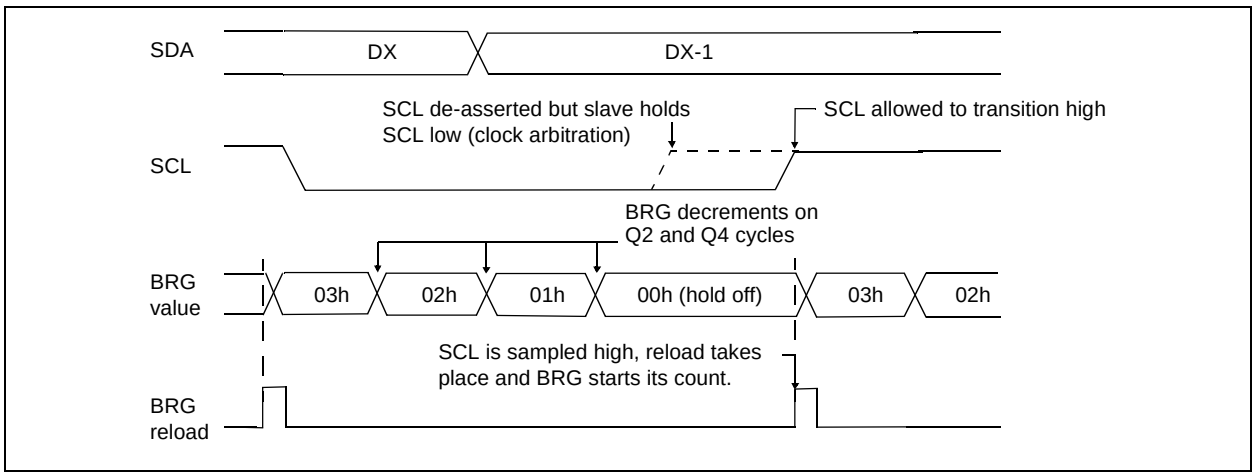


FIGURE 15-12: BAUD RATE GENERATOR TIMING WITH CLOCK ARBITRATION



REGISTER 17-9: TXBnDm – TRANSMIT BUFFER n DATA FIELD BYTE m REGISTER

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
TXBnDm7	TXBnDm6	TXBnDm5	TXBnDm4	TXBnDm3	TXBnDm2	TXBnDm1	TXBnDm0
bit 7							bit 0

bit 1-0 **TXBnDm7:TXBnDm0:** Transmit Buffer n Data Field Byte m bits (where 0≤n<3 and 0<m<8)

Each Transmit Buffer has an array of registers. For example, Transmit buffer 0 has 7 registers: TXB0D0 to TXB0D7.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 17-10: TXBnDLC – TRANSMIT BUFFER n DATA LENGTH CODE REGISTER

U-0	R/W-x	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x
—	TXRTR	—	—	DLC3	DLC2	DLC1	DLC0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6 **TXRTR:** Transmission Frame Remote Transmission Request bit
1 = Transmitted message will have TXRTR bit set
0 = Transmitted message will have TXRTR bit cleared.

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **DLC3:DLC0:** Data Length Code bits
1111 = Reserved
1110 = Reserved
1101 = Reserved
1100 = Reserved
1011 = Reserved
1010 = Reserved
1001 = Reserved
1000 = Data Length = 8 bytes
0111 = Data Length = 7 bytes
0110 = Data Length = 6 bytes
0101 = Data Length = 5 bytes
0100 = Data Length = 4 bytes
0011 = Data Length = 3 bytes
0010 = Data Length = 2 bytes
0001 = Data Length = 1 bytes
0000 = Data Length = 0 bytes

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

 © 2000 Microchip Technology Inc.

23.1 Instruction Set

ADDLW **ADD literal to W**

Syntax: [*label*] ADDLW k

Operands: $0 \le k \le 255$

Operation: (WREG) + k → WREG

Status Affected: N,OV, C, DC, Z

Encoding:

0000	1111	kkkk	kkkk

Description: The contents of WREG are added to the 8-bit literal 'k' and 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 W

Example: `ADDLW 0x15`

Before Instruction
- WREG = 0x10
- N = ?
- OV = ?
- C = ?
- DC = ?
- Z = ?

After Instruction
- WREG = 0x25
- N = 0
- OV = 0
- C = 0
- DC = 0
- Z = 0

ADDWF **ADD W to f**

Syntax: [*label*] ADDWF f [,d] [,a]

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

Operation: (WREG) + (f) → dest

Status Affected: N,OV, C, DC, Z

Encoding:

0010	01da	ffff	ffff

Description: Add WREG to register 'f'. If 'd' is 0, the result is stored in WREG. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ADDWF REG, W`

Before Instruction
- WREG = 0x17
- REG = 0xC2
- N = ?
- OV = ?
- C = ?
- DC = ?
- Z = ?

After Instruction
- WREG = 0xD9
- REG = 0xC2
- N = 1
- OV = 0
- C = 0
- DC = 0
- Z = 0

BNC **Branch if Not Carry**

Syntax: [*label*] BNC n

Operands: -128 ≤ n ≤ 127

Operation: if carry bit is '0'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0011	nnnn	nnnn

Description: If the Carry bit is '0', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BNC Jump`

Before Instruction
PC = address `(HERE)`

After Instruction
If Carry = 0;
PC = address `(Jump)`
If Carry = 1;
PC = address `(HERE+2)`

BNN **Branch if Not Negative**

Syntax: [*label*] BNN n

Operands: -128 ≤ n ≤ 127

Operation: if negative bit is '0'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

1110	0111	nnnn	nnnn

Description: If the Negative bit is '0', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BNN Jump`

Before Instruction
PC = address `(HERE)`

After Instruction
If Negative = 0;
PC = address `(Jump)`
If Negative = 1;
PC = address `(HERE+2)`

GOTO **Unconditional Branch**

Syntax: [*label*] GOTO k

Operands: $0 \le k \le 1048575$

Operation: $k \rightarrow PC<20:1>$

Status Affected: None

Encoding:

1st word (k<7:0>)	1110	1111	k_7kkk	$kkkk_0$
2nd word(k<19:8>)	1111	$k_{19}kkk$	kkkk	$kkkk_8$

Description: GOTO allows an unconditional branch anywhere within entire 2M byte memory range. The 20-bit value 'k' is loaded into PC<20:1>. GOTO is always a two-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	No operation	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

Example: GOTO THERE

After Instruction
PC = Address (THERE)

INCF **Increment f**

Syntax: [*label*] INCF f [,d [,a]]

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

Operation: $(f) + 1 \rightarrow dest$

Status Affected: C,DC,N,OV,Z

Encoding:

0010	10da	ffff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: INCF CNT

Before Instruction
CNT = 0xFF
Z = 0
C = ?
DC = ?

After Instruction
CNT = 0x00
Z = 1
C = 1
DC = 1

© 2000 Microchip Technology Inc.

LFSR **Load FSR**

Syntax: [*label*] LFSR f,k

Operands: $0 \le f \le 2$
$0 \le k \le 4095$

Operation: $k \rightarrow FSRf$

Status Affected: None

Encoding:

1110 1111	1110 0000	00ff $k_7$kkk	k_{11}kkk kkkk

Description: The 12-bit literal 'k' is loaded into the file select register pointed to by 'f'

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k' MSB	Process Data	Write literal 'k' MSB to FSRfH
Decode	Read literal 'k' LSB	Process Data	Write literal 'k' to FSRfL

Example: `LFSR FSR2, 0x3AB`

After Instruction
FSR2H = 0x03
FSR2L = 0xAB

MOVF **Move f**

Syntax: [*label*] MOVF f [,d [,a]]

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

Operation: $f \rightarrow dest$

Status Affected: N,Z

Encoding:

0101	00da	ffff	ffff

Description: The contents of register 'f' is moved to a destination dependent upon the status of 'd'. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default). Location 'f' can be anywhere in the 256 byte Bank. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `MOVF REG, W`

Before Instruction
REG = 0x22
WREG = 0xFF
N = ?
Z = ?

After Instruction
REG = 0x22
WREG = 0x22
N = 0
Z = 0

24.13 PICDEM 3 Low Cost PIC16CXXX Demonstration Board

The PICDEM 3 demonstration board is a simple demonstration board that supports the PIC16C923 and PIC16C924 in the PLCC package. It will also support future 44-pin PLCC microcontrollers with an LCD Module. All the necessary hardware and software is included to run the basic demonstration programs. The user can program the sample microcontrollers provided with the PICDEM 3 demonstration board on a PRO MATE II device programmer, or a PICSTART Plus development programmer with an adapter socket, and easily test firmware. The MPLAB ICE in-circuit emulator may also be used with the PICDEM 3 demonstration board to test firmware. A prototype area has been provided to the user for adding hardware and connecting it to the microcontroller socket(s). Some of the features include an RS-232 interface, push button switches, a potentiometer for simulated analog input, a thermistor and separate headers for connection to an external LCD module and a keypad. Also provided on the PICDEM 3 demonstration board is an LCD panel, with 4 commons and 12 segments, that is capable of displaying time, temperature and day of the week. The PICDEM 3 demonstration board provides an additional RS-232 interface and Windows 3.1 software for showing the demultiplexed LCD signals on a PC. A simple serial interface allows the user to construct a hardware demultiplexer for the LCD signals.

24.14 PICDEM 17 Demonstration Board

The PICDEM 17 demonstration board is an evaluation board that demonstrates the capabilities of several Microchip microcontrollers, including PIC17C752, PIC17C756, PIC17C762 and PIC17C766. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5-inch disk. A programmed sample is included and the user may erase it and program it with the other sample programs using the PRO MATE II device programmer, or the PICSTART Plus development programmer, and easily debug and test the sample code. In addition, the PICDEM 17 demonstration board supports down-loading of programs to and executing out of external FLASH memory on board. The PICDEM 17 demonstration board is also usable with the MPLAB ICE in-circuit emulator, or the PICMASTER emulator and all of the sample programs can be run and modified using either emulator. Additionally, a generous prototype area is available for user hardware.

24.15 KEELOQ Evaluation and Programming Tools

KEELOQ evaluation and programming tools support Microchip's HCS Secure Data Products. The HCS evaluation kit includes an LCD display to show changing codes, a decoder to decode transmissions and a programming interface to program test transmitters.

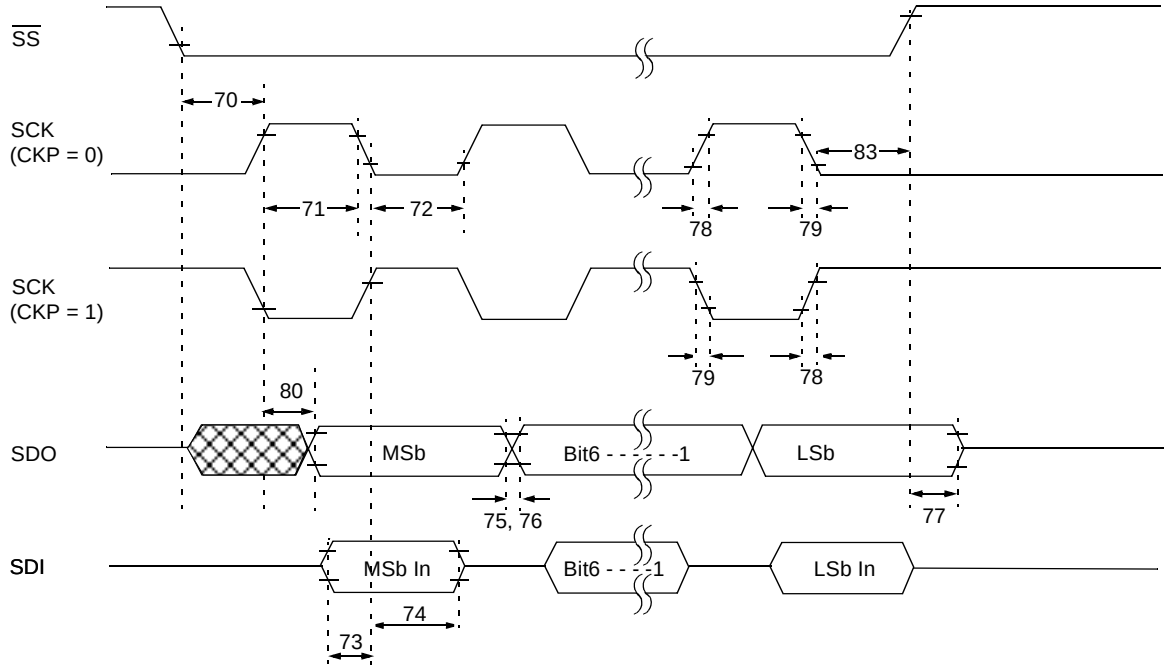
 © 2000 Microchip Technology Inc.

TABLE 25-5: PLL CLOCK TIMING SPECIFICATION (V_{DD} = 4.2V - 5.5V)

Param No.	Symbol	Characteristic	Min	Max	Units	Conditions
7	TPLL	PLL Start-up Time (Lock Time)	—	2	ms	
	ΔCLK	CLKOUT Stability (Jitter) using PLL	-2	+2	%	

PRELIMINARY

FIGURE 25-14: EXAMPLE SPI SLAVE MODE TIMING (CKE = 0)



Note: Refer to Figure 25-4 for load conditions.

TABLE 25-13: EXAMPLE SPI MODE REQUIREMENTS (SLAVE MODE TIMING (CKE = 0))

Parm. No.	Symbol	Characteristic		Min	Max	Units	Conditions
70	TssL2scH, TssL2scL	$\overline{SS}\downarrow$ to SCK$\downarrow$ or SCK$\uparrow$ input		TCY	—	ns	
71	TscH	SCK input high time (Slave mode)	Continuous	1.25TCY + 30	—	ns	
71A			Single Byte	40	—	ns	**(Note 1)**
72	TscL	SCK input low time (Slave mode)	Continuous	1.25TCY + 30	—	ns	
72A			Single Byte	40	—	ns	**(Note 1)**
73	TdiV2scH, TdiV2scL	Setup time of SDI data input to SCK edge		100	—	ns	
73A	TB2B	Last clock edge of Byte1 to the 1st clock edge of Byte2		1.5TCY + 40	—	ns	**(Note 2)**
74	TscH2diL, TscL2diL	Hold time of SDI data input to SCK edge		100	—	ns	
75	TdoR	SDO data output rise time	PIC18**C**XX8	—	25	ns	
			PIC18**LC**XX8		45	ns	
76	TdoF	SDO data output fall time		—	25	ns	
77	TssH2doZ	$\overline{SS}\uparrow$ to SDO output hi-impedance		10	50	ns	
78	TscR	SCK output rise time (Master mode)	PIC18**C**XX8	—	25	ns	
			PIC18**LC**XX8		45	ns	
79	TscF	SCK output fall time (Master mode)		—	25	ns	
80	TscH2doV, TscL2doV	SDO data output valid after SCK edge	PIC18**C**XX8	—	50	ns	
			PIC18**LC**XX8		100	ns	
83	TscH2ssH, TscL2ssH	$\overline{SS}\uparrow$ after SCK edge		1.5TCY + 40	—	ns	

Note 1: Requires the use of parameter # 73A.
2: Only if parameter #s 71A and 72A are used.

PRELIMINARY

APPENDIX E: DEVELOPMENT TOOL VERSION REQUIREMENTS

This lists the minimum requirements (software/firmware) of the specified development tool to support the devices listed in this data sheet.

MPLAB-IDE: version 5.11

MPLAB-SIM: version 7.10

MPLAB-ICE 2000:

PIC18CXX8 Processor Module:
Part Number - PCM 18XB0

PIC18CXX8 Device Adapter:

Socket	Part Number
64-pin TQFP	DVD18P2640
68-pin PLCC	DVD18XL680
80-pin TQFP	DVD18PQ800
84-pin PLCC	DVD18XL840

MPLAB-ICD: Not Available

PROMATE II: version 5.20

PICSTART Plus: version 2.20

MPASM: version 2.50

MPLAB-C18: version 1.00

CAN-TOOL: Not available at time of printing.

Note: Please read all associated README.TXT files that are supplied with the development tools. These "read me" files will discuss product support and any known limitations.

 © 2000 Microchip Technology Inc.

NOTES:

 © 2000 Microchip Technology Inc.

NOTES:

 © 2000 Microchip Technology Inc.

NOTES: