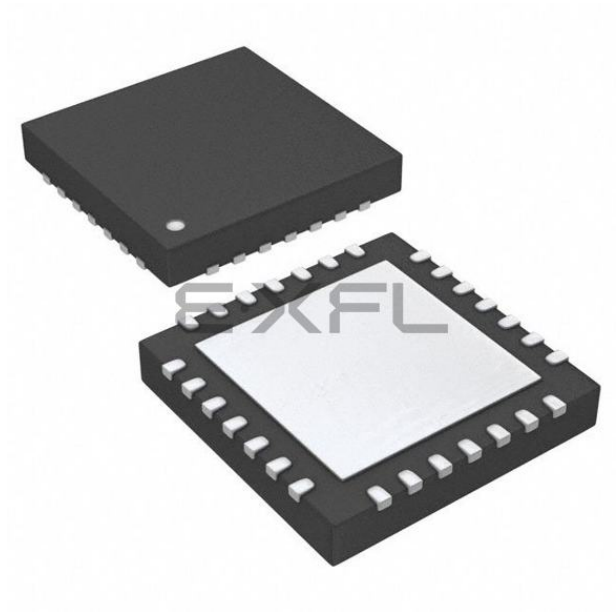


**Welcome to E-XFL.COM**

## 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 | Active |
| Core Processor | PIC |
| Core Size | 8-Bit |
| Speed | 64MHz |
| Connectivity | I²C, LINbus, SPI, UART/USART |
| Peripherals | Brown-out Detect/Reset, LVD, POR, PWM, WDT |
| Number of I/O | 25 |
| Program Memory Size | 32KB (16K x 16) |
| Program Memory Type | FLASH |
| EEPROM Size | 256 x 8 |
| RAM Size | 2K x 8 |
| Voltage - Supply (Vcc/Vdd) | 1.8V ~ 3.6V |
| Data Converters | A/D 35x10b; D/A 1x5b |
| Oscillator Type | Internal |
| Operating Temperature | -40°C ~ 85°C (TA) |
| Mounting Type | Surface Mount |
| Package / Case | 28-VQFN Exposed Pad |
| Supplier Device Package | 28-QFN (6x6) |
| Purchase URL | https://www.e-xfl.com/product-detail/microchip-technology/pic18lf25k40-i-ml |

**REGISTER 7-3: PMD2: PMD CONTROL REGISTER 2**

| U-0 | R/W-0/0 | R/W-0/0 | U-0 | U-0 | R/W-0/0 | R/W-0/0 | R/W-0/0 |
|---|---|---|---|---|---|---|---|
| — | DACMD | ADCMD | — | — | CMP2MD | CMP1MD | ZCDMD(1) |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | |
|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as '0' |
| u = Bit is unchanged | x = Bit is unknown | -n/n = Value at POR and BOR/Value at all other Resets |
| '1' = Bit is set | '0' = Bit is cleared | q = Value depends on condition |

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

bit 6 **DACMD:** Disable DAC bit
1 = DAC module disabled
0 = DAC module enabled

bit 5 **ADCMD:** Disable ADC bit
1 = ADC module disabled
0 = ADC module enabled

bit 4-3 **Unimplemented:** Read as '0'

bit 2 **CMP2MD:** Disable Comparator CMP2 bit
1 = CMP2 module disabled
0 = CMP2 module enabled

bit 1 **CMP1MD:** Disable Comparator CMP1 bit
1 = CMP1 module disabled
0 = CMP1 module enabled

bit 0 **ZCDMD:** Disable Zero-Cross Detect module bit(1)
1 = ZCD module disabled
0 = ZCD module enabled

**Note 1:** Subject to $\overline{\text{ZCD}}$ bit in CONFIG2H.

**REGISTER 9-5: WDTTMR: WDT TIMER REGISTER (READ-ONLY)**

| R-0/0 | R-0/0 | R-0/0 | R-0/0 | R-0/0 | R-0/0 | R-0/0 | R-0/0 |
|---|---|---|---|---|---|---|---|
| WDTTMR<4:0> | | | | | STATE | PSCNT<17:16> | |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | |
|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ |
| u = Bit is unchanged | x = Bit is unknown | -n/n = Value at POR and BOR/Value at all other Resets |
| ‘1’ = Bit is set | ‘0’ = Bit is cleared | |

bit 7-3 **WDTTMR<4:0>:** Watchdog Window Value bits

| WINDOW | WDT Window State | | Open Percent |
|---|---|---|---|
| | Closed | Open | |
| 111 | N/A | 00000-11111 | 100 |
| 110 | 00000-00011 | 00100-11111 | 87.5 |
| 101 | 00000-00111 | 01000-11111 | 75 |
| 100 | 00000-01011 | 01100-11111 | 62.5 |
| 011 | 00000-01111 | 10000-11111 | 50 |
| 010 | 00000-10011 | 10100-11111 | 37.5 |
| 001 | 00000-10111 | 11000-11111 | 25 |
| 000 | 00000-11011 | 11100-11111 | 12.5 |

bit 2 **STATE:** WDT Armed Status bit
1 = WDT is armed
0 = WDT is not armed

bit 1-0 **PSCNT<17:16>:** Prescale Select Upper Byte bits(1)

**Note 1:** The 18-bit WDT prescale value, PSCNT<17:0> includes the WDTPSL, WDTPSH and the lower bits of the WDTTMR registers. PSCNT<17:0> is intended for debug operations and should be read during normal operation.

**TABLE 10-5: REGISTER FILE SUMMARY FOR PIC18(L)F24/25K40 DEVICES (CONTINUED)**

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on POR, BOR |
|---|---|---|---|---|---|---|---|---|---|---|
| F74h | CRCDATL | DATA<7:0> | | | | | | | | xxxxxxxx |
| F73h | ADFLTRH | ADFLTRH<15:8> | | | | | | | | xxxxxxxx |
| F72h | ADFLTRL | ADFLTRL<7:0> | | | | | | | | xxxxxxxx |
| F71h | ADACCH | ADACCH<15:8> | | | | | | | | xxxxxxxx |
| F70h | ADACCL | ADACCL<7:0> | | | | | | | | xxxxxxxx |
| F6Fh | ADERRH | ADERRH<15:8> | | | | | | | | 00000000 |
| F6Eh | ADERRL | ADERRL<7:0> | | | | | | | | 00000000 |
| F6Dh | ADUTHH | ADUTHH<15:8> | | | | | | | | 00000000 |
| F6Ch | ADUTHL | ADUTHL<7:0> | | | | | | | | 00000000 |
| F6Bh | ADLTHH | ADLTHH<15:8> | | | | | | | | 00000000 |
| F6Ah | ADLTHL | ADLTHL<7:0> | | | | | | | | 00000000 |
| F69h | ADSTPTH | ADSTPTH<15:8> | | | | | | | | 00000000 |
| F68h | ADSTPTL | ADSTPTL<7:0> | | | | | | | | 00000000 |
| F67h | ADCNT | ADCNT<7:0> | | | | | | | | 00000000 |
| F66h | ADRPT | ADRPT<7:0> | | | | | | | | 00000000 |
| F65h | ADSTAT | ADAOV | ADUTHR | ADLTHR | ADMATH | — | ADSTAT<2:0> | | | 0000-000 |
| F64h | ADRESH | ADRESH<7:0> | | | | | | | | 00000000 |
| F63h | ADRESL | ADRESL<7:0> | | | | | | | | 00000000 |
| F62h | ADPREVH | ADPREVH<15:8> | | | | | | | | 00000000 |
| F61h | ADPREVL | ADPREVL<7:0> | | | | | | | | 00000000 |
| F60h | ADCON0 | ADON | ADCONT | — | ADSC | — | ADFM | — | ADGO | 00-000-0 |
| F5Fh | ADPCH | — | — | ADPCH<5:0> | | | | | | --000000 |
| F5Eh | ADPRE | ADPRE<7:0> | | | | | | | | 00000000 |
| F5Dh | ADCAP | — | — | — | ADCAP<4:0> | | | | | ---00000 |
| F5Ch | ADACQ | ADACQ<7:0> | | | | | | | | 00000000 |
| F5Bh | ADCON3 | — | ADCALC<2:0> | | | ADSOI | ADTMD<2:0> | | | -0000000 |
| F5Ah | ADCON2 | ADPSIS | ADCRS<2:0> | | | ADACLR | ADMD<2:0> | | | 00000000 |
| F59h | ADCON1 | ADPPOL | ADIPEN | ADGPOL | — | — | — | — | ADDSEN | 000----0 |
| F58h | ADREF | — | — | — | ADNREF | — | — | ADPREF<1:0> | | ---0--00 |
| F57h | ADCLK | — | — | ADCS<5:0> | | | | | | --000000 |
| F56h | ADACT | — | — | — | ADACT<4:0> | | | | | ---00000 |
| F55h | MDCARH | — | — | — | — | — | CHS<2:0> | | | -----000 |
| F54h | MDCARL | — | — | — | — | — | CLS<2:0> | | | -----000 |
| F53h | MDSRC | — | — | — | — | SRCS<3:0> | | | | ----0000 |
| F52h | MDCON1 | — | — | CHPOL | CHSYNC | — | — | CLPOL | CLSYNC | --00--00 |
| F51h | MDCON0 | EN | — | **OUT** | **OPOL** | — | — | — | MDBIT | 0-00---0 |
| F50h | SCANTRIG | — | — | — | — | TSEL<3:0> | | | | ----0000 |
| F4Fh | SCANCON0 | SCANEN | SCANGO | BUSY | INVALID | INTM | — | MODE<1:0> | | 00000-00 |
| F4Eh | SCANHADRU | — | — | HADR<21:16> | | | | | | --111111 |

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

**Note 1:** Not available on LF devices.

**TABLE 10-5: REGISTER FILE SUMMARY FOR PIC18(L)F24/25K40 DEVICES (CONTINUED)**

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on POR, BOR |
|---|---|---|---|---|---|---|---|---|---|---|
| F26h to F22h | — | Unimplemented | | | | | | | | — |
| F21h | ANSELC | ANSELC7 | ANSELC6 | ANSELC5 | ANSELC4 | ANSELC3 | ANSELC2 | ANSELC1 | ANSELC0 | 11111111 |
| F20h | WPUC | WPUC7 | WPUC6 | WPUC5 | WPUC4 | WPUC3 | WPUC2 | WPUC1 | WPUC0 | 00000000 |
| F1Fh | ODCONC | ODCC7 | ODCC6 | ODCC5 | ODCC4 | ODCC3 | ODCC2 | ODCC1 | ODCC0 | 00000000 |
| F1Eh | SLRCONC | SLRC7 | SLRC6 | SLRC5 | SLRC4 | SLRC3 | SLRC2 | SLRC1 | SLRC0 | 11111111 |
| F1Dh | INLVLC | INLVLC7 | INLVLC6 | INLVLC5 | INLVLC4 | INLVLC3 | INLVLC2 | INLVLC1 | INLVLC0 | 11111111 |
| F1Ch | IOCCP | IOCCP7 | IOCCP6 | IOCCP5 | IOCCP4 | IOCCP3 | IOCCP2 | IOCCP1 | IOCCP0 | 00000000 |
| F1Bh | IOCCN | IOCCN7 | IOCCN6 | IOCCN5 | IOCCN4 | IOCCN3 | IOCCN2 | IOCCN1 | IOCCN0 | 00000000 |
| F1Ah | IOCCF | IOCCF7 | IOCCF6 | IOCCF5 | IOCCF4 | IOCCF3 | IOCCF2 | IOCCF1 | IOCCF0 | 00000000 |
| F19h | ANSELB | ANSELB7 | ANSELB6 | ANSELB5 | ANSELB4 | ANSELB3 | ANSELB2 | ANSELB1 | ANSELB0 | 11111111 |
| F18h | WPUB | WPUB7 | WPUB6 | WPUB5 | WPUB4 | WPUB3 | WPUB2 | WPUB1 | WPUB0 | 00000000 |
| F17h | ODCONB | ODCB7 | ODCB6 | ODCB5 | ODCB4 | ODCB3 | ODCB2 | ODCB1 | ODCB0 | 00000000 |
| F16h | SLRCONB | SLRB7 | SLRB6 | SLRB5 | SLRB4 | SLRB3 | SLRB2 | SLRB1 | SLRB0 | 11111111 |
| F15h | INLVLB | INLVLB7 | INLVLB6 | INLVLB5 | INLVLB4 | INLVLB3 | INLVLB2 | INLVLB1 | INLVLB0 | 11111111 |
| F14h | IOCBP | IOCBP7 | IOCBP6 | IOCBP5 | IOCBP4 | IOCBP3 | IOCBP2 | IOCBP1 | IOCBP0 | 00000000 |
| F13h | IOCBN | IOCBN7 | IOCBN6 | IOCBN5 | IOCBN4 | IOCBN3 | IOCBN2 | IOCBN1 | IOCBN0 | 00000000 |
| F12h | IOCBF | IOCBF7 | IOCBF6 | IOCBF5 | IOCBF4 | IOCBF3 | IOCBF2 | IOCBF1 | IOCBF0 | 00000000 |
| F11h | ANSELA | ANSELA7 | ANSELA6 | ANSELA5 | ANSELA4 | ANSELA3 | ANSELA2 | ANSELA1 | ANSELA0 | 11111111 |
| F10h | WPUA | WPUA7 | WPUA6 | WPUA5 | WPUA4 | WPUA3 | WPUA2 | WPUA1 | WPUA0 | 00000000 |
| F0Fh | ODCONA | ODCA7 | ODCA6 | ODCA5 | ODCA4 | ODCA3 | ODCA2 | ODCA1 | ODCA0 | 00000000 |
| F0Eh | SLRCONA | SLRA7 | SLRA6 | SLRA5 | SLRA4 | SLRA3 | SLRA2 | SLRA1 | SLRA0 | 11111111 |
| F0Dh | INLVLA | INLVLA7 | INLVLA6 | INLVLA5 | INLVLA4 | INLVLA3 | INLVLA2 | INLVLA1 | INLVLA0 | 11111111 |
| F0Ch | IOCAP | IOCAP7 | IOCAP6 | IOCAP5 | IOCAP4 | IOCAP3 | IOCAP2 | IOCAP1 | IOCAP0 | 00000000 |
| F0Bh | IOCAN | IOCAN7 | IOCAN6 | IOCAN5 | IOCAN4 | IOCAN3 | IOCAN2 | IOCAN1 | IOCAN0 | 00000000 |
| F0Ah | IOCAF | IOCAF7 | IOCAF6 | IOCAF5 | IOCAF4 | IOCAF3 | IOCAF2 | IOCAF1 | IOCAF0 | 00000000 |
| F09h to EFFh | — | Unimplemented | | | | | | | | — |
| EFEh | RC7PPS | — | — | — | RC7PPS<4:0> | | | | | ---00000 |
| EFDh | RC6PPS | — | — | — | RC6PPS<4:0> | | | | | ---00000 |
| EFCh | RC5PPS | — | — | — | RC5PPS<4:0> | | | | | ---00000 |
| EFBh | RC4PPS | — | — | — | RC4PPS<4:0> | | | | | ---00000 |
| EFAh | RC3PPS | — | — | — | RC3PPS<4:0> | | | | | ---00000 |
| EF9h | RC2PPS | — | — | — | RC2PPS<4:0> | | | | | ---00000 |
| EF8h | RC1PPS | — | — | — | RC1PPS<4:0> | | | | | ---00000 |
| EF7h | RC0PPS | — | — | — | RC0PPS<4:0> | | | | | ---00000 |
| EF6h | RB7PPS | — | — | — | RB7PPS<4:0> | | | | | ---00000 |
| EF5h | RB6PPS | — | — | — | RB6PPS<4:0> | | | | | ---00000 |
| EF4h | RB5PPS | — | — | — | RB5PPS<4:0> | | | | | ---00000 |
| EF3h | RB4PPS | — | — | — | RB4PPS<4:0> | | | | | ---00000 |

**Legend:** x = unknown, u = unchanged, — = unimplemented, q = value depends on condition
**Note 1:** Not available on LF devices.

**REGISTER 14-10: PIE0: PERIPHERAL INTERRUPT ENABLE REGISTER 0**

| U-0 | U-0 | R/W-0/0 | R/W-0/0 | U-0 | R/W-0/0 | R/W-0/0 | R/W-0/0 |
|---|---|---|---|---|---|---|---|
| — | — | TMR0IE(1) | IOCIE(1) | — | INT2IE(1) | INT1IE(1) | INT0IE(1) |
| bit 7 | | | | | | | bit 0 |

| **Legend:** IE | | | |
|---|---|---|---|
| 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 |

bit 7-6 **Unimplemented:** Read as '0'

bit 5 **TMR0IE:** Timer0 Interrupt Enable bit(1)
- 1 = Enabled
- 0 = Disabled

bit 4 **IOCIE:** Interrupt-on-Change Enable bit(1)
- 1 = Enabled
- 0 = Disabled

bit 3 **Unimplemented:** Read as '0'

bit 2 **INT2IE:** External Interrupt 2 Enable bit(1)
- 1 = Enabled
- 0 = Disabled

bit 1 **INT1IE:** External Interrupt 1 Enable bit(1)
- 1 = Enabled
- 0 = Disabled

bit 0 **INT0IE:** External Interrupt 0 Enable bit(1)
- 1 = Enabled
- 0 = Disabled

**Note 1:** PIR0 interrupts are not disabled by the PEIE bit in the INTCON register. are not disabled by the PEIE bit in the INTCON register.

**TABLE 16-1: IOC REGISTERS**

| Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| IOCAP | IOCAP7 | IOCAP6 | IOCAP5 | IOCAP4 | IOCAP3 | IOCAP2 | IOCAP1 | IOCAP0 |
| IOCAN | IOCAN7 | IOCAN6 | IOCAN5 | IOCAN4 | IOCAN3 | IOCAN2 | IOCAN1 | IOCAN0 |
| IOCAF | IOCAF7 | IOCAF6 | IOCAF5 | IOCAF4 | IOCAF3 | IOCAF2 | IOCAF1 | IOCAF0 |
| IOCBP | IOCBP7 | IOCBP6 | IOCBP5 | IOCBP4 | IOCBP3 | IOCBP2 | IOCBP1 | IOCBP0 |
| IOCBN | IOCBN7 | IOCBN6 | IOCBN5 | IOCBN4 | IOCBN3 | IOCBN2 | IOCBN1 | IOCBN0 |
| IOCBF | IOCBF7 | IOCBF6 | IOCBF5 | IOCBF4 | IOCBF3 | IOCBF2 | IOCBF1 | IOCBF0 |
| IOCCP | IOCCP7 | IOCCP6 | IOCCP5 | IOCCP4 | IOCCP3 | IOCCP2 | IOCCP1 | IOCCP0 |
| IOCCN | IOCCN7 | IOCCN6 | IOCCN5 | IOCCN4 | IOCCN3 | IOCCN2 | IOCCN1 | IOCCN0 |
| IOCCF | IOCCF7 | IOCCF6 | IOCCF5 | IOCCF4 | IOCCF3 | IOCCF2 | IOCCF1 | IOCCF0 |
| IOCEP | — | — | — | — | IOCEP3(1) | — | — | — |
| IOCEN | — | — | — | — | IOCEN3(1) | — | — | — |
| IOCEF | — | — | — | — | IOCEF3(1) | — | — | — |

**Note 1:** If MCLRE = 1 or LVP = 1, RE3 port functionality is disabled and IOC on RE3 is not available.

**TABLE 16-2: SUMMARY OF REGISTERS ASSOCIATED WITH INTERRUPT-ON-CHANGE**

| Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Register on Page |
|---|---|---|---|---|---|---|---|---|---|
| INTCON | GIE/GIEH | PEIE/GIEL | IPEN | — | — | INT2EDG | INT1EDG | INT0EDG | 166 |
| IOCxF | IOCxF7 | IOCxF6 | IOCxF5 | IOCxF4 | IOCxF3 | IOCxF2 | IOCxF1 | IOCxF0 | 206 |
| IOCxN | IOCxN7 | IOCxN6 | IOCxN5 | IOCxN4 | IOCxN3 | IOCxN2 | IOCxN1 | IOCxN0 | 206 |
| IOCxP | IOCxP7 | IOCxP6 | IOCxP5 | IOCxP4 | IOCxP3 | IOCxP2 | IOCxP1 | IOCxP0 | 206 |

**Legend:** — = unimplemented location, read as '0'. Shaded cells are not used by interrupt-on-change.

## 18.3 Programmable Prescaler

A software programmable prescaler is available for exclusive use with Timer0. There are 16 prescaler options for Timer0 ranging in powers of two from 1:1 to 1:32768. The prescaler values are selected using the T0CKPS<3:0> bits of the T0CON1 register.

The prescaler is not directly readable or writable. Clearing the prescaler register can be done by writing to the TMR0L register or the T0CON0, T0CON1 registers or by any Reset.

## 18.4 Programmable Postscaler

A software programmable postscaler (output divider) is available for exclusive use with Timer0. There are 16 postscaler options for Timer0 ranging from 1:1 to 1:16. The postscaler values are selected using the T0OUTPS<3:0> bits of the T0CON0 register.

The postscaler is not directly readable or writable. Clearing the postscaler register can be done by writing to the TMR0L register or the T0CON0, T0CON1 registers or by any Reset.

## 18.5 Operation During Sleep

When operating synchronously, Timer0 will halt. When operating asynchronously, Timer0 will continue to increment and wake the device from Sleep (if Timer0 interrupts are enabled) provided that the input clock source is active.

## 18.6 Timer0 Interrupts

The Timer0 interrupt flag bit (TMR0IF) is set when either of the following conditions occur:

- 8-bit TMR0L matches the TMR0H value
- 16-bit TMR0 rolls over from ‘FFFFh’

When the postscaler bits (T0OUTPS<3:0>) are set to 1:1 operation (no division), the T0IF flag bit will be set with every TMR0 match or rollover. In general, the TMR0IF flag bit will be set every T0OUTPS +1 matches or rollovers.

If Timer0 interrupts are enabled (TMR0IE bit of the PIE0 register = ‘1’), the CPU will be interrupted and the device may wake from Sleep (see **Section 18.2 “Clock Source Selection”** for more details).

## 18.7 Timer0 Output

The Timer0 output can be routed to any I/O pin via the RxyPPS output selection register (see **Section 17.0 “Peripheral Pin Select (PPS) Module”** for additional information). The Timer0 output can also be used by other peripherals, such as the auto-conversion trigger of the Analog-to-Digital Converter. Finally, the Timer0 output can be monitored through software via the Timer0 output bit (T0OUT) of the T0CON0 register (Register 18-1).

TMR0_out will be a pulse of one postscaled clock period when a match occurs between TMR0L and PR0 (Period register for TMR0) in 8-bit mode, or when TMR0 rolls over in 16-bit mode. The Timer0 output is a 50% duty cycle that toggles on each TMR0_out rising clock edge.

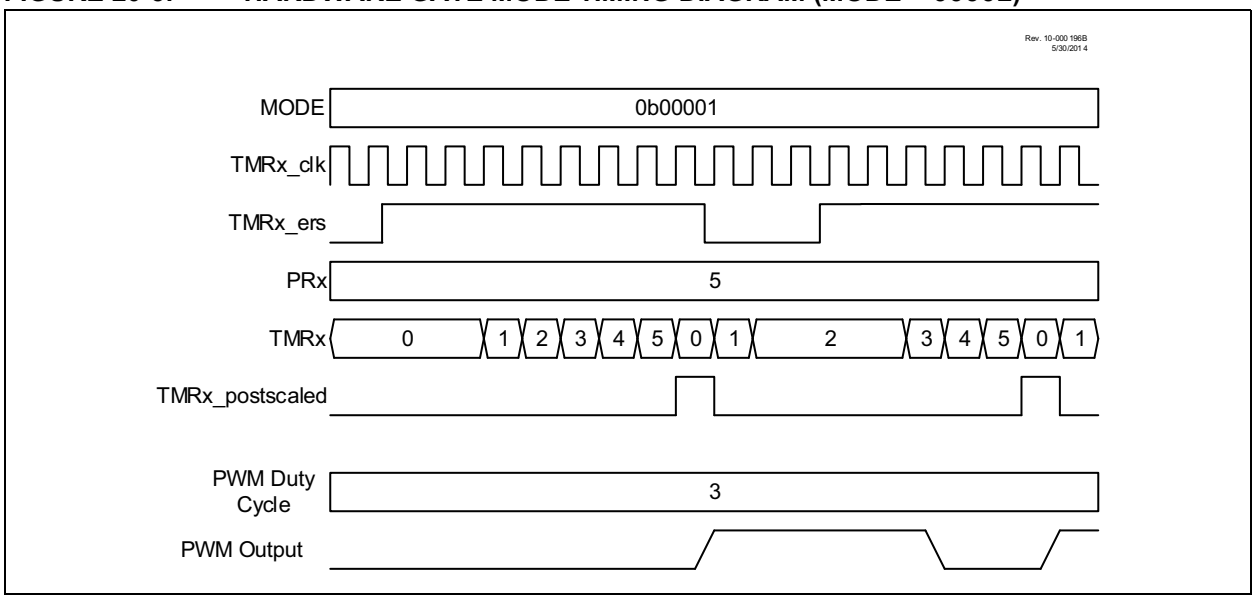
### 20.5.2 HARDWARE GATE MODE

The Hardware Gate modes operate the same as the Software Gate mode except the TMRx_ers external signal can also gate the timer. When used with the CCP the gating extends the PWM period. If the timer is stopped when the PWM output is high then the duty cycle is also extended.

When MODE<4:0> = 00001 then the timer is stopped when the external signal is high. When MODE<4:0> = 00010 then the timer is stopped when the external signal is low.

Figure 20-5 illustrates the Hardware Gating mode for MODE<4:0> = 00001 in which a high input level starts the counter.

**FIGURE 20-5: HARDWARE GATE MODE TIMING DIAGRAM (MODE = 00001)**



## 21.2 Register Definitions: CCP Control

Long bit name prefixes for the CCP peripherals are shown in Table 21-2. Refer to **Section 1.4.2.2 "Long Bit Names"** for more information.

**TABLE 21-2:**

| Peripheral | Bit Name Prefix |
|---|---|
| CCP1 | CCP1 |
| CCP2 | CCP2 |

**REGISTER 21-1: CCPxCON: CCPx CONTROL REGISTER**

| R/W-0/0 | U-0 | R-x | R/W-0/0 | R/W-0/0 | R/W-0/0 | R/W-0/0 | R/W-0/0 |
|---|---|---|---|---|---|---|---|
| EN | — | OUT | FMT | MODE<3:0> | | | |
| bit 7 | | | | | | | bit 0 |

| **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 |

bit 7 **EN:** CCP Module Enable bit
1 = CCP is enabled
0 = CCP is disabled

bit 6 **Unimplemented:** Read as '0'

bit 5 **OUT:** CCPx Output Data bit (read-only)

bit 4 **FMT:** CCPW (pulse-width) Alignment bit
MODE = Capture mode:
Unused
MODE = Compare mode:
Unused
MODE = PWM mode:
1 = Left-aligned format
0 = Right-aligned format

**Note 1:** The set and clear operations of the Compare mode are reset by setting MODE = 4'b0000 or EN = 0.

**2:** When MODE = 0001 or 1011, then the timer associated with the CCP module is cleared. TMR1 is the default selection for the CCP module, so it is used for indication purpose only.

## 21.3 Capture Mode

Capture mode makes use of the 16-bit Timer1 resource. When an event occurs on the capture source, the 16-bit CCPRxH:CCPRxL register pair captures and stores the 16-bit value of the TMRxH:TMRxL register pair, respectively. An event is defined as one of the following and is configured by the MODE<3:0> bits of the CCPxCON register:

- Every falling edge of CCPx input
- Every rising edge of CCPx input
- Every 4th rising edge of CCPx input
- Every 16th rising edge of CCPx input
- Every edge of CCPx input (rising or falling)

When a capture is made, the Interrupt Request Flag bit CCPxIF of the PIR6 register is set. The interrupt flag must be cleared in software. If another capture occurs before the value in the CCPRxH:CCPRxL register pair is read, the old captured value is overwritten by the new captured value.

| **Note:** | If an event occurs during a 2-byte read, the high and low-byte data will be from different events. It is recommended while reading the CCPRxH:CCPRxL register pair to either disable the module or read the register pair twice for data integrity. |
|---|---|

Figure 21-1 shows a simplified diagram of the capture operation.

### 21.3.1 CAPTURE SOURCES

In Capture mode, the CCPx pin should be configured as an input by setting the associated TRIS control bit.

| **Note:** | If the CCPx pin is configured as an output, a write to the port can cause a capture condition. |
|---|---|

The capture source is selected by configuring the CTS<1:0> bits of the CCPxCAP register. The following sources can be selected:

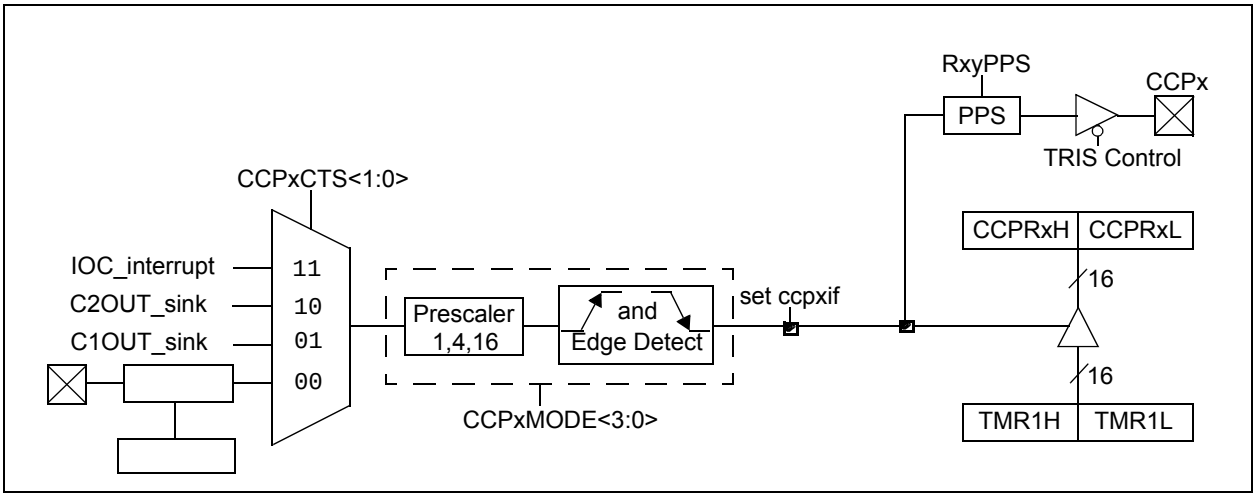
- Pin selected by CCPxPPS
- C1_output
- C2_output
- IOC_interrupt

### 21.3.2 TIMER1 MODE RESOURCE

Timer1 must be running in Timer mode or Synchronized Counter mode for the CCP module to use the capture feature. In Asynchronous Counter mode, the capture operation may not work.

- See **Section 19.0 "Timer1/3/5 Module with Gate Control"** for more information on configuring Timer1.

**FIGURE 21-1: CAPTURE MODE OPERATION BLOCK DIAGRAM**



## 25.2 DSM Operation

The DSM module can be enabled by setting the MDEN bit in the MDCON0 register. Clearing the MDEN bit in the MDCON0 register, disables the DSM module output and switches the carrier high and carrier low signals to the default option of MDCARHPPS and MDCARLPPS, respectively. The modulator signal source is also switched to the MDBIT in the MDCON0 register. This not only assures that the DSM module is inactive, but that it is also consuming the least amount of current.

The values used to select the carrier high, carrier low, and modulator sources held by the Modulation Source, Modulation High Carrier, and Modulation Low Carrier control registers are not affected when the MDEN bit is cleared and the DSM module is disabled. The values inside these registers remain unchanged while the DSM is inactive. The sources for the carrier high, carrier low and modulator signals will once again be selected when the MDEN bit is set and the DSM module is again enabled and active.

The modulated output signal can be disabled without shutting down the DSM module. The DSM module will remain active and continue to mix signals, but the output value will not be sent to the DSM pin. During the time that the output is disabled, the DSM pin will remain low. The modulated output can be disabled by clearing the MDEN bit in the MDCON register.

## 25.3 Modulator Signal Sources

The modulator signal can be supplied from the following sources:

- External signal on pin selected by MDSRCPPS
- MDBIT bit in the MDCON0 register
- CCP1/2 Output
- PWM3/4 Output
- Comparator C1/C2 Output
- EUSART RX Signal
- EUSART TX Signal
- MSSP SDO Signal (SPI Mode Only)

The modulator signal is selected by configuring the MDSRCS<3:0> bits in the MDSRC register.

## 25.4 Carrier Signal Sources

The carrier high signal and carrier low signal can be supplied from the following sources:

- External signal on pin selected by MDCARHPPS/ MDCARLPPS
- FOSC (system clock)
- HFINTOSC
- Reference Clock Module Signal
- CCP1/2 Output Signal
- PWM3/4 Output

The carrier high signal is selected by configuring the MDCHS<2:0> bits in the MDCARH register. The carrier low signal is selected by configuring the MDCLS<2:0> bits in the MDCARL register.

## 25.5 Carrier Synchronization

During the time when the DSM switches between carrier high and carrier low signal sources, the carrier data in the modulated output signal can become truncated. To prevent this, the carrier signal can be synchronized to the modulator signal. When synchronization is enabled, the carrier pulse that is being mixed at the time of the transition is allowed to transition low before the DSM switches over to the next carrier source.

Synchronization is enabled separately for the carrier high and carrier low signal sources. Synchronization for the carrier high signal is enabled by setting the MDCHSYNC bit in the MDCON1 register. Synchronization for the carrier low signal is enabled by setting the MDCLSYNC bit in the MDCON1 register.

Figure 25-2 through Figure 25-6 show timing diagrams of using various synchronization methods.

### 26.5.5 SPI OPERATION IN SLEEP MODE

In SPI Master mode, module clocks may be operating at a different speed than when in Full-Power mode; in the case of the Sleep mode, all clocks are halted.

Special care must be taken by the user when the MSSP clock is much faster than the system clock.

In Slave mode, when MSSP interrupts are enabled, after the master completes sending data, an MSSP interrupt will wake the controller from Sleep.

If an exit from Sleep mode is not desired, MSSP interrupts should be disabled.

In SPI Master mode, when the Sleep mode is selected, all module clocks are halted and the transmission/reception will remain in that state until the device wakes. After the device returns to Run mode, the module will resume transmitting and receiving data.

In SPI Slave mode, the SPI Transmit/Receive Shift register operates asynchronously to the device. This allows the device to be placed in Sleep mode and data to be shifted into the SPI Transmit/Receive Shift register. When all eight bits have been received, the MSSP interrupt flag bit will be set and if enabled, will wake the device.

**TABLE 26-1: SUMMARY OF REGISTERS ASSOCIATED WITH SPI OPERATION**

| Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Register on Page |
|---|---|---|---|---|---|---|---|---|---|
| INTCON | GIE/GIEH | PEIE/GIEL | IPEN | — | — | INT2EDG | INT1EDG | INT0EDG | 166 |
| PIE3 | — | — | RC1IE | TX1IE | — | — | BCL1IE | SSP1IE | 178 |
| PIR3 | — | — | RC1IF | TX1IF | — | — | BCL1IF | SSP1IF | 170 |
| IPR3 | — | — | RC1IP | TX1IP | — | — | BCL1IP | SSP1IP | 186 |
| RxyPPS | — | — | — | RxyPPS<4:0> | | | | | 213 |
| SSPxBUF | BUF<7:0> | | | | | | | | 330* |
| SSPxCLKPPS | — | — | — | SSPxCLKPPS<4:0> | | | | | 211 |
| SSPxCON1 | WCOL | SSPOV | SSPEN | CKP | SSPM<3:0> | | | | 332 |
| SSPxCON3 | ACKTIM | PCIE | SCIE | BOEN | SDAHT | SBCDE | AHEN | DHEN | 333 |
| SSPxDATPPS | — | — | — | SSPDATPPS<4:0> | | | | | 211 |
| SSPxSSPPS | — | — | — | SSPSSPPS<4:0> | | | | | 211 |
| SSPxSTAT | SMP | CKE | D/$\overline{A}$ | P | S | R/$\overline{W}$ | UA | BF | 347 |

**Legend:** — = Unimplemented location, read as '0'. Shaded cells are not used by the MSSP in SPI mode.

* Page provides register information.

### 26.6.2 ARBITRATION

Each master device must monitor the bus for Start and Stop bits. If the device detects that the bus is busy, it cannot begin a new message until the bus returns to an Idle state.

However, two master devices may try to initiate a transmission on or about the same time. When this occurs, the process of arbitration begins. Each transmitter checks the level of the SDA data line and compares it to the level that it expects to find. The first transmitter to observe that the two levels do not match, loses arbitration, and must stop transmitting on the SDA line.

For example, if one transmitter holds the SDA line to a logical one (lets it float) and a second transmitter holds it to a logical zero (pulls it low), the result is that the SDA line will be low. The first transmitter then observes that the level of the line is different than expected and concludes that another transmitter is communicating.

The first transmitter to notice this difference is the one that loses arbitration and must stop driving the SDA line. If this transmitter is also a master device, it also must stop driving the SCL line. It then can monitor the lines for a Stop condition before trying to reissue its transmission. In the meantime, the other device that has not noticed any difference between the expected and actual levels on the SDA line continues with its original transmission. It can do so without any complications, because so far, the transmission appears exactly as expected with no other transmitter disturbing the message.

Slave Transmit mode can also be arbitrated, when a master addresses multiple slaves, but this is less common.

If two master devices are sending a message to two different slave devices at the address stage, the master sending the lower slave address always wins arbitration. When two master devices send messages to the same slave address, and addresses can sometimes refer to multiple slaves, the arbitration process must continue into the data stage.

Arbitration usually occurs very rarely, but it is a necessary process for proper multi-master support.

## 26.7 Register Definitions: $I^2C$ Mode

The MSSPx module has seven registers for $I^2C$ operation.

These are:

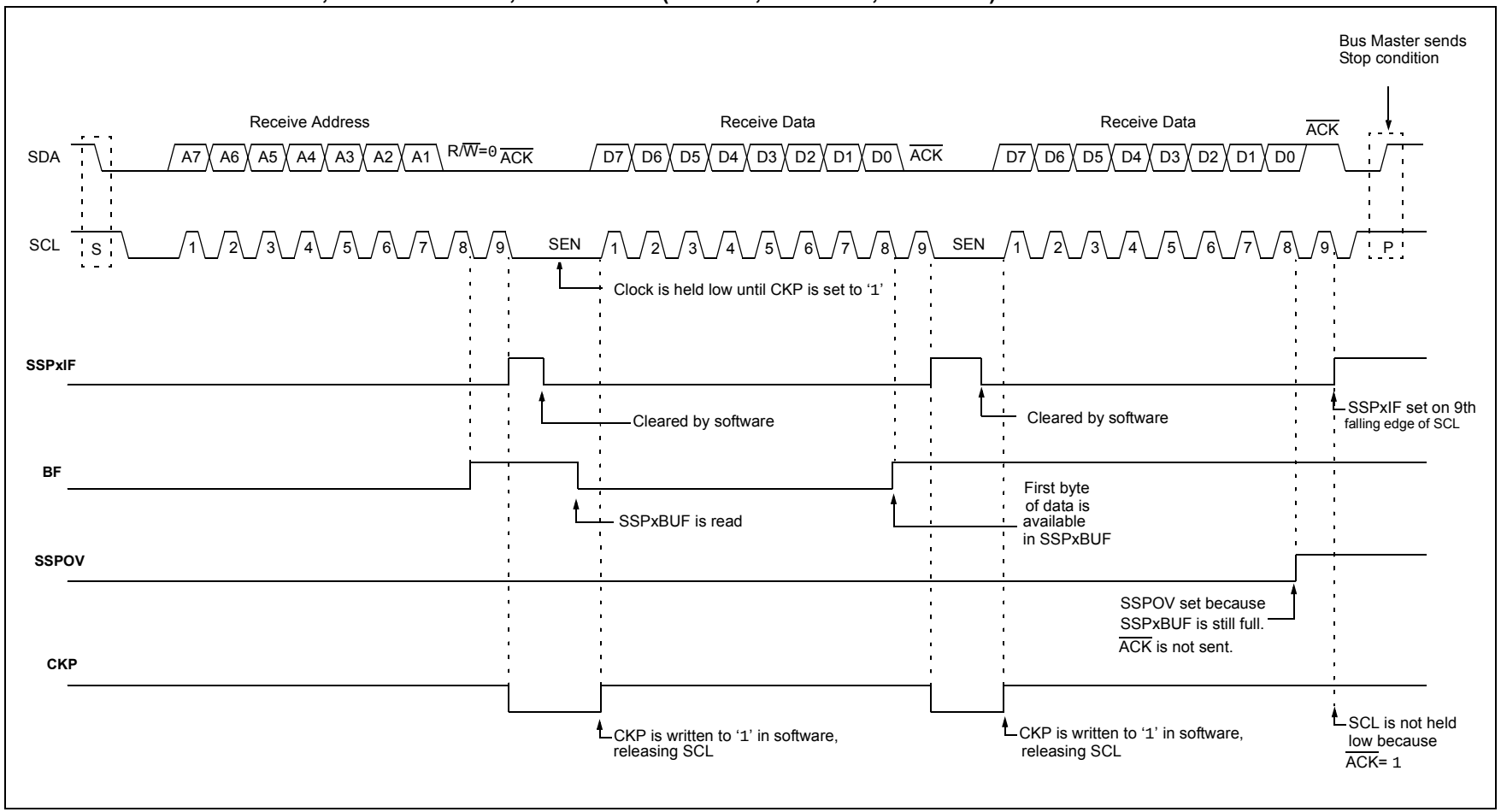
- MSSP Status Register (SSPxSTAT)
- MSSP Control Register 1 (SSPxCON1)
- MSSP Control Register 2 (SSPxCON2)
- MSSP Control Register 3 (SSPxCON3)
- Serial Receive/Transmit Buffer Register (SSPxBUF)
- MSSP Address Register (SSPxADD)
- $I^2C$ Slave Address Mask Register (SSPxMSK)
- MSSP Shift Register (SSPSR) – not directly accessible

SSPxCON1, SSPxCON2, SSPxCON3 and SSPxSTAT are the Control and Status registers in $I^2C$ mode operation. The SSPxCON1, SSPxCON2, and SSPxCON3 registers are readable and writable. The lower six bits of the SSPxSTAT are read-only. The upper two bits of the SSPxSTAT are read/write.

SSPSR is the Shift register used for shifting data in or out. SSPxBUF is the buffer register to which data bytes are written to or read from. SSPxADD contains the slave device address when the MSSP is configured in $I^2C$ Slave mode. When the MSSP is configured in Master mode, the lower seven bits of SSPxADD act as the Baud Rate Generator reload value.

SSPxMSK holds the slave address mask value when the module is configured for 7-Bit Address Masking mode. While it is a separate register, it shares the same SFR address as SSPxADD; it is only accessible when the SSPM<3:0> bits are specifically set to permit access. In receive operations, SSPSR and SSPxBUF together, create a double-buffered receiver. When SSPSR receives a complete byte, it is transferred to SSPxBUF and the SSPxIF interrupt is set. During transmission, the SSPxBUF is not double-buffered. A write to SSPxBUF will write to both SSPxBUF and SSPSR.

**FIGURE 26-15: I²C SLAVE, 7-BIT ADDRESS, RECEPTION (SEN = 1, AHEN = 0, DHEN = 0)**



## 27.4 EUSART Baud Rate Generator (BRG)

The Baud Rate Generator (BRG) is an 8-bit or 16-bit timer that is dedicated to the support of both the asynchronous and synchronous EUSART operation. By default, the BRG operates in 8-bit mode. Setting the BRG16 bit of the BAUDxCON register selects 16-bit mode.

The SPxBRGH, SPxBRGL register pair determines the period of the free running baud rate timer. In Asynchronous mode the multiplier of the baud rate period is determined by both the BRGH bit of the TXxSTA register and the BRG16 bit of the BAUDxCON register. In Synchronous mode, the BRGH bit is ignored.

Table 27-3 contains the formulas for determining the baud rate. Example 27-1 provides a sample calculation for determining the baud rate and baud rate error.

Typical baud rates and error values for various asynchronous modes have been computed for your convenience and are shown in Table 27-5. It may be advantageous to use the high baud rate (BRGH = 1), or the 16-bit BRG (BRG16 = 1) to reduce the baud rate error. The 16-bit BRG mode is used to achieve slow baud rates for fast oscillator frequencies.

Writing a new value to the SPxBRGH, SPxBRGL register pair causes the BRG timer to be reset (or cleared). This ensures that the BRG does not wait for a timer overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receive error or data loss may result. To avoid this problem, check the status of the RCIDL bit to make sure that the receive operation is idle before changing the system clock.

**EXAMPLE 27-1: CALCULATING BAUD RATE ERROR**

For a device with FOSC of 16 MHz, desired baud rate of 9600, Asynchronous mode, 8-bit BRG:

$$Desired\ Baud\ Rate = \frac{FOSC}{64([SPxBRGH:SPxBRGL] + 1)}$$

Solving for SPxBRGH:SPxBRGL:

$$SPBRGH:SPBRGL = \frac{\frac{FOSC}{Desired\ Baud\ Rate}}{64} - 1$$

$$= \frac{\frac{16000000}{9600}}{64} - 1$$

$$= [25.042] = 25$$

$$Calculated\ Baud\ Rate = \frac{16000000}{64(25 + 1)}$$

$$= 9615$$

$$Error = \frac{Calc.\ Baud\ Rate - Desired\ Baud\ Rate}{Desired\ Baud\ Rate}$$

$$= \frac{(9615 - 9600)}{9600} = 0.16\%$$

**BCF** **Bit Clear f**

Syntax: BCF f, b {,a}

Operands: $0 \le f \le 255$
$0 \le b \le 7$
$a \in [0,1]$

Operation: $0 \rightarrow f<b>$

Status Affected: None

Encoding:

| 1001 | bbba | ffff | ffff |
|---|---|---|---|

Description: Bit 'b' in register 'f' is cleared.
If 'a' is '0', the Access Bank is selected.
If 'a' is '1', the BSR is used to select the GPR bank.
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 35.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `BCF FLAG_REG, 7, 0`

Before Instruction
FLAG_REG = C7h
After Instruction
FLAG_REG = 47h

**BN** **Branch if Negative**

Syntax: BN n

Operands: $-128 \le n \le 127$

Operation: if NEGATIVE bit is '1'
$(PC) + 2 + 2n \rightarrow PC$

Status Affected: None

Encoding:

| 1110 | 0110 | nnnn | nnnn |
|---|---|---|---|

Description: If the NEGATIVE bit is '1', 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 2-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    BN   Jump
```
Before Instruction
PC = address (HERE)
After Instruction
If NEGATIVE = 1;
PC = address (Jump)
If NEGATIVE = 0;
PC = address (HERE + 2)

**BZ** **Branch if Zero**

Syntax: BZ n

Operands: -128 ≤ n ≤ 127

Operation: if ZERO bit is '1'
(PC) + 2 + 2n → PC

Status Affected: None

Encoding:

| 1110 | 0000 | nnnn | nnnn |
|---|---|---|---|

Description: If the ZERO bit is '1', 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 2-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 BZ Jump

Before Instruction
PC = address (HERE)
After Instruction
If ZERO = 1;
PC = address (Jump)
If ZERO = 0;
PC = address (HERE + 2)

**CALL** **Subroutine Call**

Syntax: CALL k {,s}

Operands: 0 ≤ k ≤ 1048575
s ∈ [0,1]

Operation: (PC) + 4 → TOS,
k → PC<20:1>,
if s = 1
(W) → WS,
(Status) → STATUSS,
(BSR) → BSRS

Status Affected: None

Encoding:

| | | | |
|---|---|---|---|
| 1st word (k<7:0>) | 1110 | 110s | $k_7$kkk | kkkk$_0$ |
| 2nd word(k<19:8>) | 1111 | $k_{19}$kkk | kkkk | kkkk$_8$ |

Description: Subroutine call of entire 2-Mbyte memory range. First, return address (PC + 4) is pushed onto the return stack. If 's' = 1, the W, Status and BSR registers are also pushed into their respective shadow registers, WS, STATUSS and BSRS. If 's' = 0, no update occurs (default). Then, the 20-bit value 'k' is loaded into PC<20:1>. CALL is a 2-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

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

Example: HERE CALL THERE, 1

Before Instruction
PC = address (HERE)
After Instruction
PC = address (THERE)
TOS = address (HERE + 4)
WS = W
BSRS = BSR
STATUSS = Status

**RRNCF** **Rotate Right f (No Carry)**

Syntax: RRNCF f {,d {,a}}

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

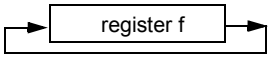
Operation: (f<n>) → dest<n – 1>,
(f<0>) → dest<7>

Status Affected: N, Z

Encoding:

| 0100 | 00da | ffff | ffff |
|---|---|---|---|

Description: The contents of register 'f' are rotated one bit to the right. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed back in register 'f' (default).
If 'a' is '0', the Access Bank will be selected (default), overriding the BSR value. If 'a' is '1', then the bank will be selected as per the BSR value.
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 35.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.




Words: 1

Cycles: 1

Q Cycle Activity:

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

Example 1:
```
RRNCF   REG, 1, 0
Before Instruction
    REG     =   1101 0111
After Instruction
    REG     =   1110 1011
```

Example 2:
```
RRNCF   REG, 0, 0
Before Instruction
    W       =   ?
    REG     =   1101 0111
After Instruction
    W       =   1110 1011
    REG     =   1101 0111
```

**SETF** **Set f**

Syntax: SETF f {,a}

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

Operation: FFh → f

Status Affected: None

Encoding:

| 0110 | 100a | ffff | ffff |
|---|---|---|---|

Description: The contents of the specified register are set to FFh.
If 'a' is '0', the Access Bank is selected.
If 'a' is '1', the BSR is used to select the GPR bank.
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 35.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example:
```
SETF    REG, 1
Before Instruction
    REG     =   5Ah
After Instruction
    REG     =   FFh
```

### 35.2.2 EXTENDED INSTRUCTION SET

**ADDFSR** **Add Literal to FSR**

Syntax: ADDFSR f, k

Operands: $0 \le k \le 63$
$f \in [0, 1, 2]$

Operation: FSR(f) + k → FSR(f)

Status Affected: None

Encoding:

| 1110 | 1000 | ffkk | kkkk |
|---|---|---|---|

Description: The 6-bit literal 'k' is added to the contents of the FSR specified by 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read literal 'k' | Process Data | Write to FSR |

Example: `ADDFSR 2, 23h`

Before Instruction
FSR2 = 03FFh
After Instruction
FSR2 = 0422h

**ADDULNK** **Add Literal to FSR2 and Return**

Syntax: ADDULNK k

Operands: $0 \le k \le 63$

Operation: FSR2 + k → FSR2,
(TOS) → PC

Status Affected: None

Encoding:

| 1110 | 1000 | 11kk | kkkk |
|---|---|---|---|

Description: The 6-bit literal 'k' is added to the contents of FSR2. A RETURN is then executed by loading the PC with the TOS.
The instruction takes two cycles to execute; a NOP is performed during the second cycle.
This may be thought of as a special case of the ADDFSR instruction, where f = 3 (binary '11'); it operates only on FSR2.

Words: 1

Cycles: 2

Q Cycle Activity:

| Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|
| Decode | Read literal 'k' | Process Data | Write to FSR |
| No Operation | No Operation | No Operation | No Operation |

Example: `ADDULNK 23h`

Before Instruction
FSR2 = 03FFh
PC = 0100h
After Instruction
FSR2 = 0422h
PC = (TOS)

**Note:** All PIC18 instructions may take an optional label argument preceding the instruction mnemonic for use in symbolic addressing. If a label is used, the instruction syntax then becomes: {label} instruction argument(s).

## 37.2 Standard Operating Conditions

The standard operating conditions for any device are defined as:

Operating Voltage: $V_{DDMIN} \leq V_{DD} \leq V_{DDMAX}$
Operating Temperature: $T_{A\_MIN} \leq T_A \leq T_{A\_MAX}$

**$V_{DD}$ — Operating Supply Voltage(1)**

PIC18LF24/25K40

- $V_{DDMIN}$ (Fosc ≤ 16 MHz) ..... +1.8V
- $V_{DDMIN}$ (Fosc ≤ 32 MHz) ..... +2.5V
- $V_{DDMIN}$ (Fosc ≤ 64 MHz) ..... +3.0V
- $V_{DDMAX}$ ..... +3.6V

PIC18F24/25K40

- $V_{DDMIN}$ (Fosc ≤ 16 MHz) ..... +2.3V
- $V_{DDMIN}$ (Fosc ≤ 32 MHz) ..... +2.5V
- $V_{DDMIN}$ (Fosc ≤ 64 MHz) ..... +3.0V
- $V_{DDMAX}$ ..... +5.5V

**$T_A$ — Operating Ambient Temperature Range**

Industrial Temperature

- $T_{A\_MIN}$ ..... -40°C
- $T_{A\_MAX}$ ..... +85°C

Extended Temperature

- $T_{A\_MIN}$ ..... -40°C
- $T_{A\_MAX}$ ..... +125°C

**Note 1:** See Parameter Supply Voltage, DS Characteristics: Supply Voltage.