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"

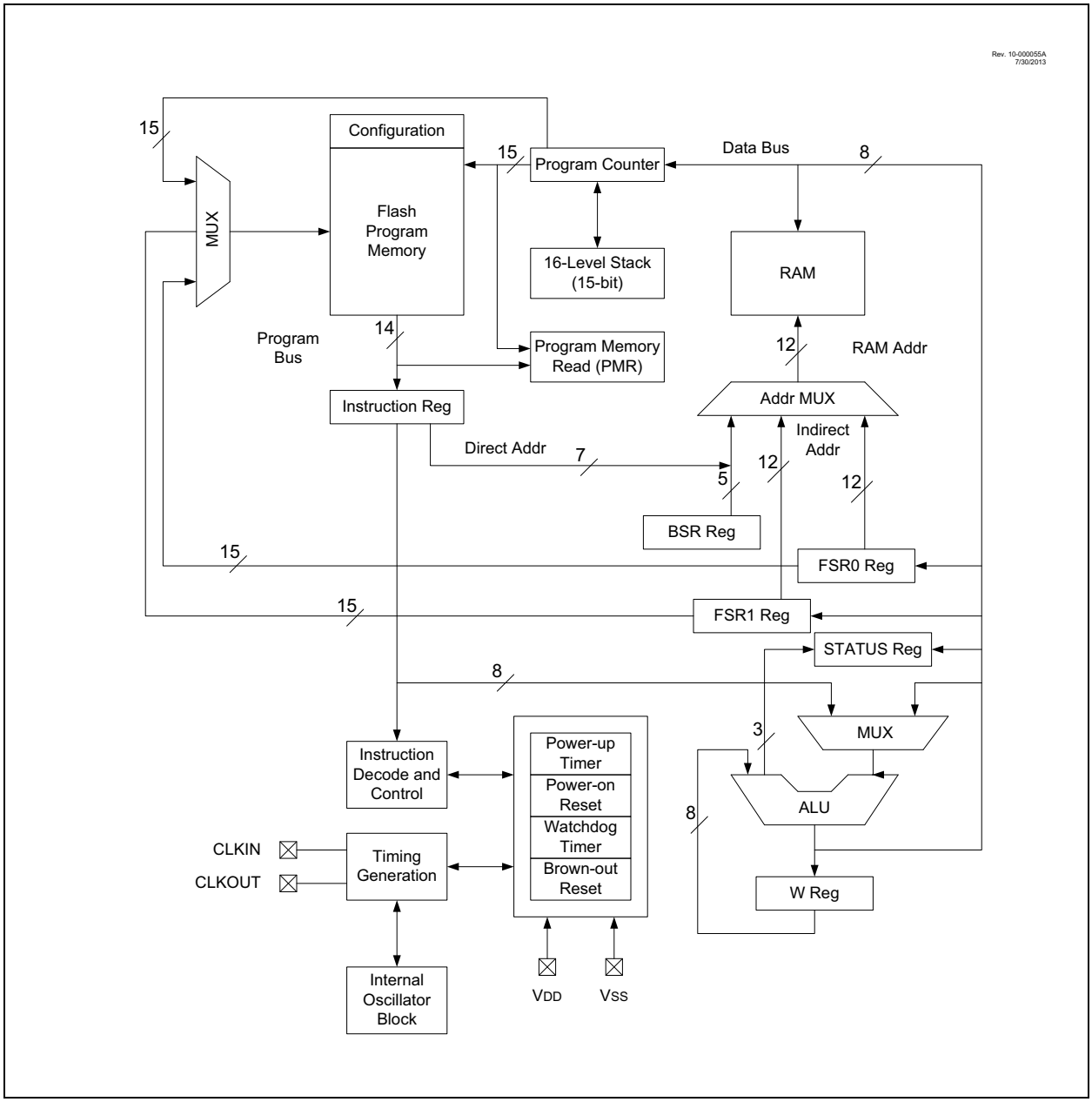
Details	
Product Status	Active
Core Processor	PIC
Core Size	8-Bit
Speed	32MHz
Connectivity	I²C, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	18
Program Memory Size	14KB (8K x 14)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	1K x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 3.6V
Data Converters	A/D 12x10b; D/A 1x8b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	20-SSOP (0.209", 5.30mm Width)
Supplier Device Package	20-SSOP
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic16lf1619-i-ss

2.0 ENHANCED MID-RANGE CPU

This family of devices contain an enhanced mid-range 8-bit CPU core. The CPU has 49 instructions. Interrupt capability includes automatic context saving. The hardware stack is 16 levels deep and has Overflow and Underflow Reset capability. Direct, Indirect, and Relative Addressing modes are available. Two File Select Registers (FSRs) provide the ability to read program and data memory.

- Automatic Interrupt Context Saving
- 16-level Stack with Overflow and Underflow
- File Select Registers
- Instruction Set

FIGURE 2-1: CORE BLOCK DIAGRAM



3.2.1.2 Indirect Read with FSR

The program memory can be accessed as data by setting bit 7 of the FSRxH register and reading the matching INDFx register. The `MOVIW` instruction will place the lower eight bits of the addressed word in the W register. Writes to the program memory cannot be performed via the INDF registers. Instructions that access the program memory via the FSR require one extra instruction cycle to complete. Example 3-2 demonstrates accessing the program memory via an FSR.

The `HIGH` operator will set bit<7> if a label points to a location in program memory.

EXAMPLE 3-2: ACCESSING PROGRAM MEMORY VIA FSR

```
constants
  DW DATA0             ;First constant
  DW DATA1             ;Second constant
  DW DATA2
  DW DATA3
my_function
  ;… LOTS OF CODE…
  MOVLW  DATA_INDEX
  ADDLW  LOW constants
  MOVWF  FSR1L
  MOVLW  HIGH constants;MSb sets
                        automatically
  MOVWF  FSR1H
  BTFSC  STATUS, C     ;carry from ADDLW?
  INCF   FSR1h, f      ;yes
  MOVIW  0[FSR1]
;THE PROGRAM MEMORY IS IN W
```

TABLE 3-14: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Addr	Name	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
Bank 12											
60Ch	PID1Z2L	Z2<7:0>								0000 0000	0000 0000
60Dh	PID1Z2H	Z2<15:8>								0000 0000	0000 0000
60Eh	PID1Z2U	—	—	—	—	—	—	—	Z216	---- ---0	---- ---0
60Fh	PID1ACCLL	ACC<7:0>								0000 0000	0000 0000
610h	PID1ACCLH	ACC<15:8>								0000 0000	0000 0000
611h	PID1ACCHL	ACC<23:16>								0000 0000	0000 0000
612h	PID1ACCHH	ACC<31:24>								0000 0000	0000 0000
613h	PID1ACCU	—	—	—	—	—	ACC<34:32>			---- -000	---- -000
614h	PID1CON	EN	BUSY	—	—	—	MODE<2:0>			00-- 0000	00-- 0000
615h	—	Unimplemented								—	—
616h	—	Unimplemented								—	—
617h	PWM3DCL	DC<1:0>		—	—	—	—	—	—	xx-- ----	xx-- ----
618h	PWM3DCH	DC<9:2>								xxxx xxxx	xxxx xxxx
619h	PWM3CON	EN	—	OUT	POL	—	—	—	—	0-x0 ----	0-x0 ----
61Ah	PWM4DCL	DC<1:0>		—	—	—	—	—	—	xx-- ----	xx-- ----
61Bh	PWM4DCH	DC<9:2>								xxxx xxxx	xxxx xxxx
61Ch	PWM4CON	EN	—	OUT	POL	—	—	—	—	0-x0 ----	0-x0 ----
61Dh to 61Fh	—	Unimplemented								—	—

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: PIC16F1615/9 only.

2: Unimplemented, read as '1'.

3: PIC16(L)F1615 only.

4: PIC16(L)F1619 only.

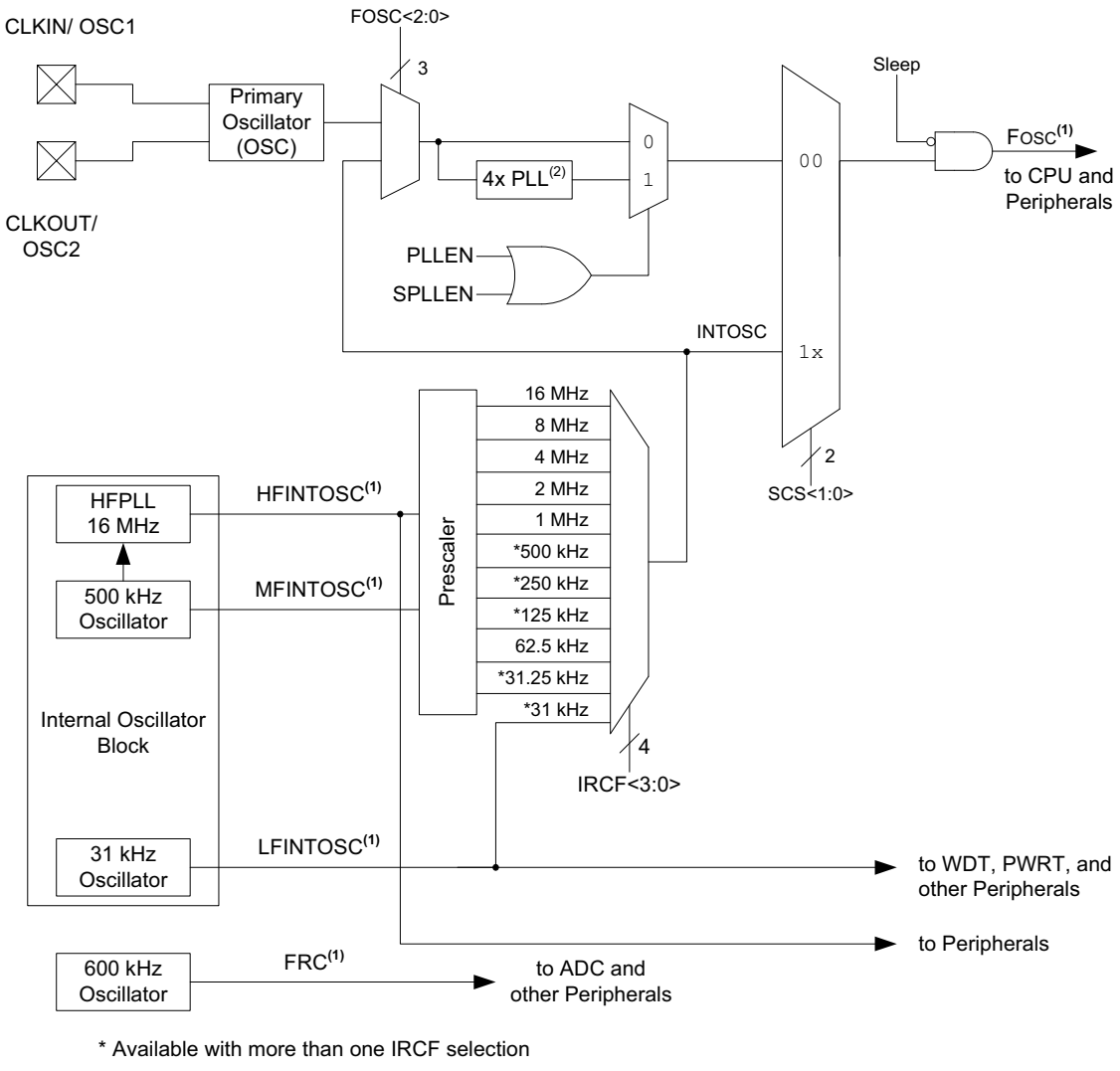
TABLE 3-14: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Addr	Name	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
Banks 28 (Continued)											
E24h	RXPPS(3)	—	—	—	RXPPS<4:0>					---1 0101	---1 0101
E24h	RXPPS(4)	—	—	—	RXPPS<4:0>					---0 1101	---0 1101
E25h	CKPPS(3)	—	—	—	CKPPS<4:0>					---1 0100	---1 0100
E25h	CKPPS(4)	—	—	—	CKPPS<4:0>					---0 1111	---0 1111
E26h	SMT1SIGPPS	—	—	—	SMT1SIGPPS<4:0>					---0 0100	---0 0100
E27h	SMT1WINPPS	—	—	—	SMT1WINPPS<4:0>					---0 0101	---0 0101
E28h	CLCIN0PPS	—	—	—	CLCIN0PPS<4:0>					---1 0011	---1 0011
E29h	CLCIN1PPS	—	—	—	CLCIN1PPS<4:0>					---1 0100	---1 0100
E2Ah	CLCIN2PPS	—	—	—	CLCIN2PPS<4:0>					---1 0001	---1 0001
E2Bh	CLCIN3PPS	—	—	—	CLCIN3PPS<4:0>					---0 0101	---0 0101
E2Ch	SMT2SIGPPS	—	—	—	SMT2SIGPPS<4:0>					---1 0001	---1 0001
E2Dh	SMT2WINPPS	—	—	—	SMT2WINPPS<4:0>					---0 0011	---0 0011
E2Eh	ATCC3PPS	—	—	—	ATCC3PPS<4:0>					---1 0101	---1 0101
E2Fh to E6Fh	—	Unimplemented								—	—

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: PIC16F1615/9 only.
2: Unimplemented, read as '1'.
3: PIC16(L)F1615 only.
4: PIC16(L)F1619 only.

FIGURE 5-1: SIMPLIFIED PIC® MCU CLOCK SOURCE BLOCK DIAGRAM



Note 1: See **Section 5.2 “Clock Source Types”**.

2: When FOSC<1:0> = 00, 4x PLL can only be used if IRCF<3:0> = 1110.

11.4 CRC Data Sources

Data can be input to the CRC module in two ways:

- User data using the CRCDAT registers
- Flash using the Program Memory Scanner

To set the number of bits of data, up to 16 bits, the DLEN bits of CRCCON1 must be set accordingly. Only data bits in CRCDATA registers up to DLEN will be used, other data bits in CRCDATA registers will be ignored.

Data is moved into the CRCSHIFT as an intermediate to calculate the check value located in the CRCACC registers.

The SHIFTM bit is used to determine the bit order of the data being shifted into the accumulator. If SHIFTM is not set, the data will be shifted in MSb first. The value of DLEN will determine the MSb. If SHIFTM bit is set, the data will be shifted into the accumulator in reversed order, LSb first.

The CRC module can be seeded with an initial value by setting the CRCACC<15:0> registers to the appropriate value before beginning the CRC.

11.4.1 CRC FROM USER DATA

To use the CRC module on data input from the user, the user must write the data to the CRCDAT registers. The data from the CRCDAT registers will be latched into the shift registers on any write to the CRCDATL register.

11.4.2 CRC FROM FLASH

To use the CRC module on data located in Flash memory, the user can initialize the Program Memory Scanner as defined in **Section 11.8, Program Memory Scan Configuration**.

11.5 CRC Check Value

The CRC check value will be located in the CRCACC registers after the CRC calculation has finished. The check value will depend on two mode settings of the CRCCON: ACCM and SHIFTM.

If the ACCM bit is set, the CRC module will augment the data with a number of zeros equal to the length of the polynomial to find the final check value. If the ACCM bit is not set, the CRC will stop at the end of the data. A number of zeros equal to the length of the polynomial can then be entered to find the same check value as augmented mode, alternatively the expected check value can be entered at this point to make the final result equal 0.

A final XOR value may be needed with the check value to find the desired CRC result

11.6 CRC Interrupt

The CRC will generate an interrupt when the BUSY bit transitions from `1` to `0`. The CRCIF interrupt flag bit of the PIR4 register is set every time the BUSY bit transitions, regardless of whether or not the CRC interrupt is enabled. The CRCIF bit can only be cleared in software. The CRC interrupt enable is the CRCIE bit of the PIE4 register.

REGISTER 12-5: WPUA: WEAK PULL-UP PORTA REGISTER

U-0	U-0	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
—	—	WPUA5	WPUA4	WPUA3	WPUA2	WPUA1	WPUA0
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 7-6 **Unimplemented:** Read as '0'

bit 5-0 **WPUA<5:0>**: Weak Pull-up Register bits(3)
1 = Pull-up enabled
0 = Pull-up disabled

Note 1: Global $\overline{\text{WPUEN}}$ bit of the OPTION_REG register must be cleared for individual pull-ups to be enabled.
2: The weak pull-up device is automatically disabled if the pin is configured as an output.
3: For the WPUA3 bit, when MCLRE = 1, weak pull-up is internally enabled, but not reported here.

REGISTER 12-6: ODCONA: PORTA OPEN-DRAIN CONTROL REGISTER

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
—	—	ODA5	ODA4	—	ODA2	ODA1	ODA0
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	

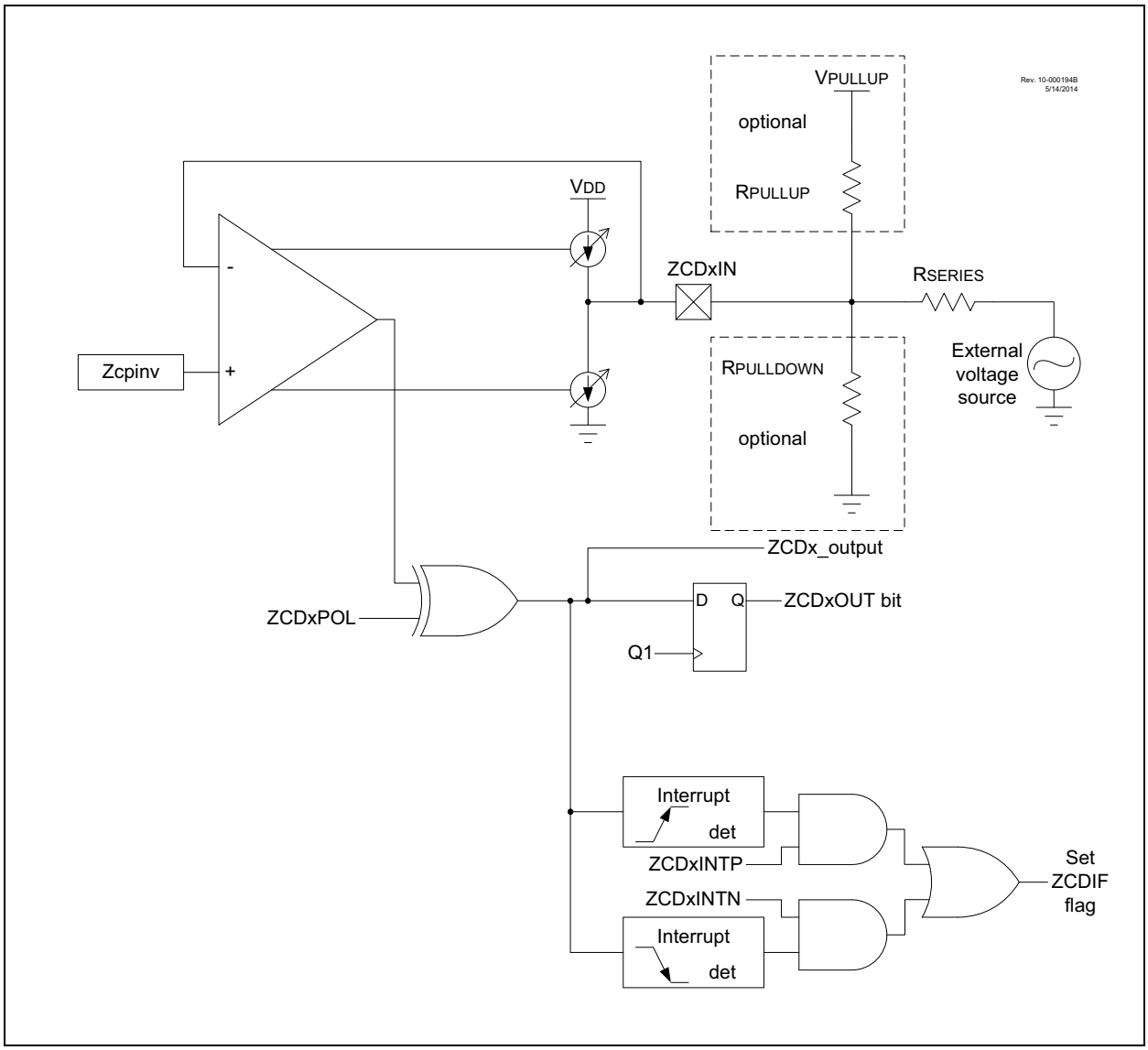
bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **ODA<5:4>:** PORTA Open-Drain Enable bits
For RA<5:4> pins, respectively
1 = Port pin operates as open-drain drive (sink current only)
0 = Port pin operates as standard push-pull drive (source and sink current)

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

bit 2-0 **ODA<2:0>:** PORTA Open-Drain Enable bits
For RA<2:0> pins, respectively
1 = Port pin operates as open-drain drive (sink current only)
0 = Port pin operates as standard push-pull drive (source and sink current)

FIGURE 20-2: SIMPLIFIED ZCD BLOCK DIAGRAM



When one device is transmitting a logical one, or letting the line float, and a second device is transmitting a logical zero, or holding the line low, the first device can detect that the line is not a logical one. This detection, when used on the SCL line, is called clock stretching. Clock stretching gives slave devices a mechanism to control the flow of data. When this detection is used on the SDA line, it is called arbitration. Arbitration ensures that there is only one master device communicating at any single time.

24.3.1 CLOCK STRETCHING

When a slave device has not completed processing data, it can delay the transfer of more data through the process of clock stretching. An addressed slave device may hold the SCL clock line low after receiving or sending a bit, indicating that it is not yet ready to continue. The master that is communicating with the slave will attempt to raise the SCL line in order to transfer the next bit, but will detect that the clock line has not yet been released. Because the SCL connection is open-drain, the slave has the ability to hold that line low until it is ready to continue communicating.

Clock stretching allows receivers that cannot keep up with a transmitter to control the flow of incoming data.

24.3.2 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.

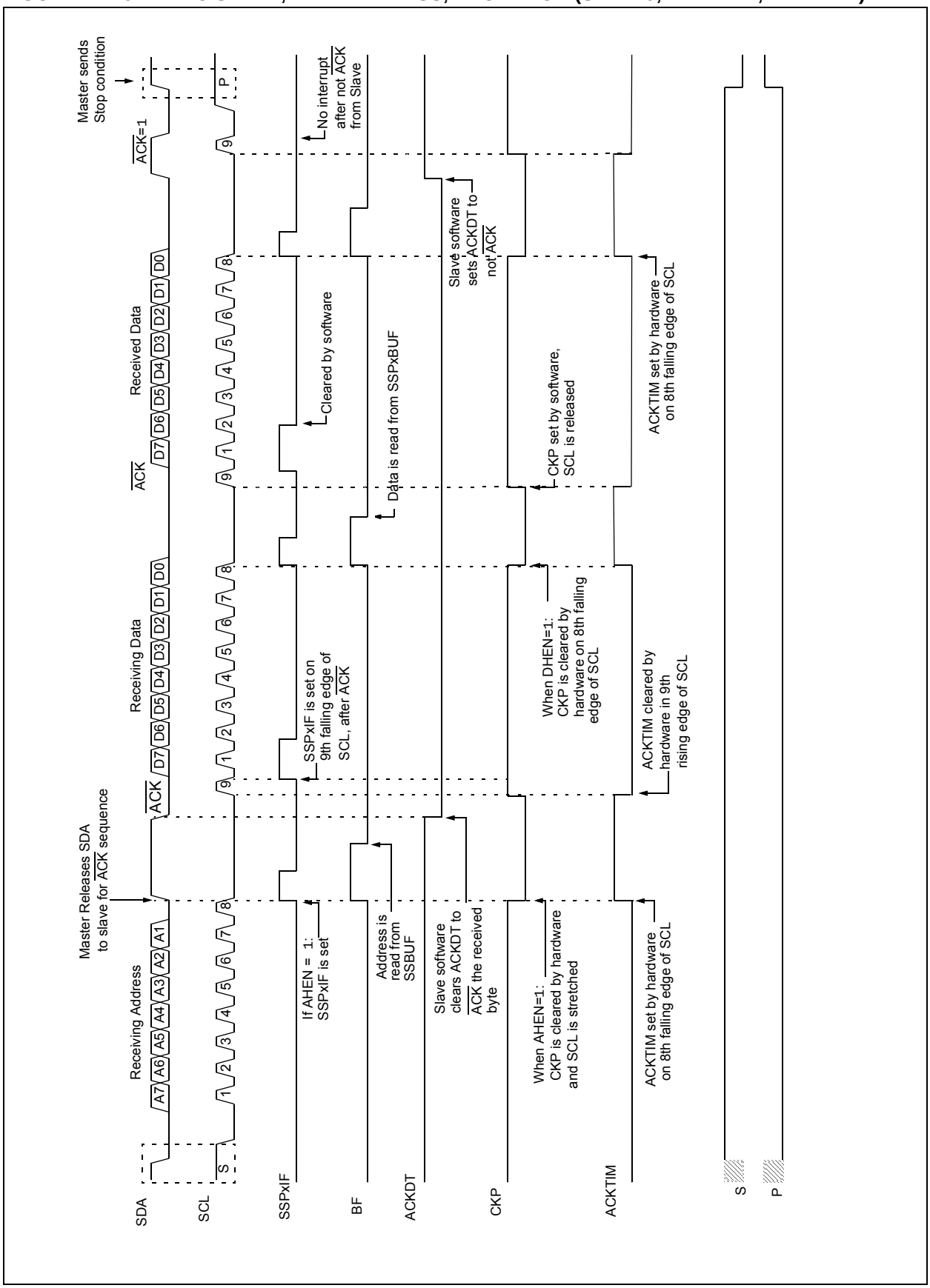
The first transmitter to notice this difference is the one that loses arbitration and must stop driving the SDA line. If this transmitter is also a master device, it also must stop driving the SCL line. It then can monitor the lines for a Stop condition before trying to reissue its transmission. In the 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.

FIGURE 24-16: I²C SLAVE, 7-BIT ADDRESS, RECEPTION (SEN = 0, AHEN = 1, DHEN = 1)



25.1 EUSART Asynchronous Mode

The EUSART transmits and receives data using the standard non-return-to-zero (NRZ) format. NRZ is implemented with two levels: a VOH Mark state which represents a '1' data bit, and a VOL Space state which represents a '0' data bit. NRZ refers to the fact that consecutively transmitted data bits of the same value stay at the output level of that bit without returning to a neutral level between each bit transmission. An NRZ transmission port idles in the Mark state. Each character transmission consists of one Start bit followed by eight or nine data bits and is always terminated by one or more Stop bits. The Start bit is always a space and the Stop bits are always marks. The most common data format is eight bits. Each transmitted bit persists for a period of 1/(Baud Rate). An on-chip dedicated 8-bit/16-bit Baud Rate Generator is used to derive standard baud rate frequencies from the system oscillator. See Table 25-5 for examples of baud rate configurations.

The EUSART transmits and receives the LSb first. The EUSART's transmitter and receiver are functionally independent, but share the same data format and baud rate. Parity is not supported by the hardware, but can be implemented in software and stored as the ninth data bit.

25.1.1 EUSART ASYNCHRONOUS TRANSMITTER

The EUSART transmitter block diagram is shown in Figure 25-1. The heart of the transmitter is the serial Transmit Shift Register (TSR), which is not directly accessible by software. The TSR obtains its data from the transmit buffer, which is the TXxREG register.

25.1.1.1 Enabling the Transmitter

The EUSART transmitter is enabled for asynchronous operations by configuring the following three control bits:

- TXEN = 1
- SYNC = 0
- SPEN = 1

All other EUSART control bits are assumed to be in their default state.

Setting the TXEN bit of the TXxSTA register enables the transmitter circuitry of the EUSART. Clearing the SYNC bit of the TXxSTA register configures the EUSART for asynchronous operation. Setting the SPEN bit of the RCxSTA register enables the EUSART and automatically configures the TX/CK I/O pin as an output. If the TX/CK pin is shared with an analog peripheral, the analog I/O function must be disabled by clearing the corresponding ANSEL bit.

Note:	The TXIF Transmitter Interrupt flag is set when the TXEN enable bit is set.

25.1.1.2 Transmitting Data

A transmission is initiated by writing a character to the TXxREG register. If this is the first character, or the previous character has been completely flushed from the TSR, the data in the TXxREG is immediately transferred to the TSR register. If the TSR still contains all or part of a previous character, the new character data is held in the TXxREG until the Stop bit of the previous character has been transmitted. The pending character in the TXxREG is then transferred to the TSR in one TCY immediately following the Stop bit transmission. The transmission of the Start bit, data bits and Stop bit sequence commences immediately following the transfer of the data to the TSR from the TXxREG.

25.1.1.3 Transmit Data Polarity

The polarity of the transmit data can be controlled with the SCKP bit of the BAUDxCON register. The default state of this bit is '0' which selects high true transmit idle and data bits. Setting the SCKP bit to '1' will invert the transmit data resulting in low true idle and data bits. The SCKP bit controls transmit data polarity in Asynchronous mode only. In Synchronous mode, the SCKP bit has a different function. See **Section 25.5.1.2 "Clock Polarity"**.

25.1.1.4 Transmit Interrupt Flag

The TXIF interrupt flag bit of the PIR1 register is set whenever the EUSART transmitter is enabled and no character is being held for transmission in the TXxREG. In other words, the TXIF bit is only clear when the TSR is busy with a character and a new character has been queued for transmission in the TXxREG. The TXIF flag bit is not cleared immediately upon writing TXxREG. TXIF becomes valid in the second instruction cycle following the write execution. Polling TXIF immediately following the TXxREG write will return invalid results. The TXIF bit is read-only, it cannot be set or cleared by software.

The TXIF interrupt can be enabled by setting the TXIE interrupt enable bit of the PIE1 register. However, the TXIF flag bit will be set whenever the TXxREG is empty, regardless of the state of TXIE enable bit.

To use interrupts when transmitting data, set the TXIE bit only when there is more data to send. Clear the TXIE interrupt enable bit upon writing the last character of the transmission to the TXxREG.

25.4.4 BREAK CHARACTER SEQUENCE

The EUSART module has the capability of sending the special Break character sequences that are required by the LIN bus standard. A Break character consists of a Start bit, followed by 12 '0' bits and a Stop bit.

To send a Break character, set the SENDB and TXEN bits of the TXxSTA register. The Break character transmission is then initiated by a write to the TXxREG. The value of data written to TXxREG will be ignored and all '0's will be transmitted.

The SENDB bit is automatically reset by hardware after the corresponding Stop bit is sent. This allows the user to preload the transmit FIFO with the next transmit byte following the Break character (typically, the Sync character in the LIN specification).

The TRMT bit of the TXxSTA register indicates when the transmit operation is active or idle, just as it does during normal transmission. See Figure 25-9 for the timing of the Break character sequence.

25.4.4.1 Break and Sync Transmit Sequence

The following sequence will start a message frame header made up of a Break, followed by an auto-baud Sync byte. This sequence is typical of a LIN bus master.

1. Configure the EUSART for the desired mode.
2. Set the TXEN and SENDB bits to enable the Break sequence.
3. Load the TXxREG with a dummy character to initiate transmission (the value is ignored).
4. Write '55h' to TXxREG to load the Sync character into the transmit FIFO buffer.
5. After the Break has been sent, the SENDB bit is reset by hardware and the Sync character is then transmitted.

When the TXxREG becomes empty, as indicated by the TXIF, the next data byte can be written to TXxREG.

25.4.5 RECEIVING A BREAK CHARACTER

The Enhanced EUSART module can receive a Break character in two ways.

The first method to detect a Break character uses the FERR bit of the RCxSTA register and the received data as indicated by RCxREG. The Baud Rate Generator is assumed to have been initialized to the expected baud rate.

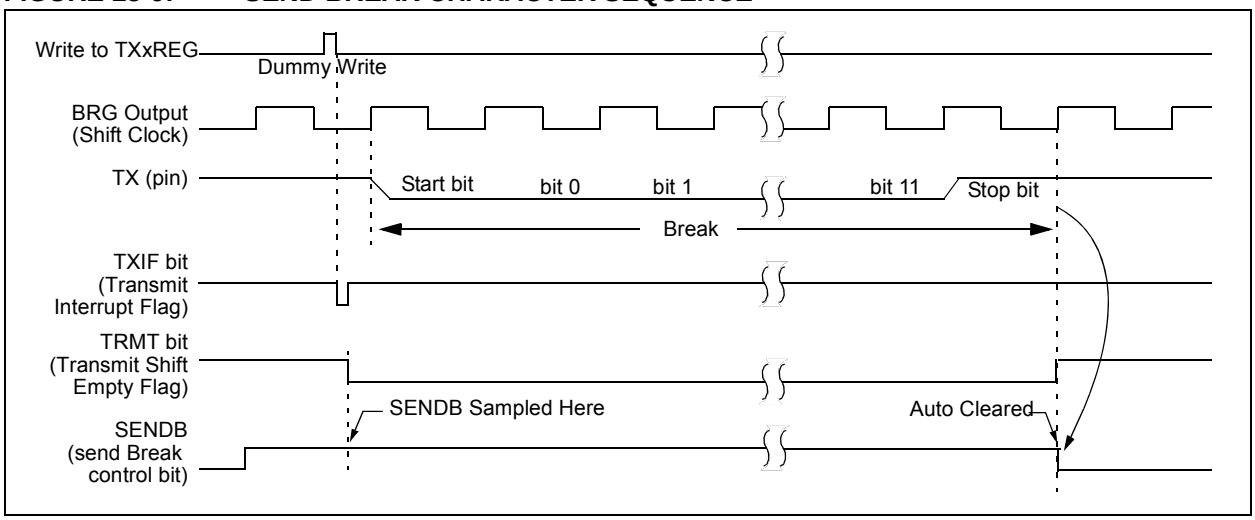
A Break character has been received when;

- RCIF bit is set
- FERR bit is set
- RCxREG = 00h

The second method uses the Auto-Wake-up feature described in **Section 25.4.3 "Auto-Wake-up on Break"**. By enabling this feature, the EUSART will sample the next two transitions on RX/DT, cause an RCIF interrupt, and receive the next data byte followed by another interrupt.

Note that following a Break character, the user will typically want to enable the Auto-Baud Detect feature. For both methods, the user can set the ABDEN bit of the BAUDxCON register before placing the EUSART in Sleep mode.

FIGURE 25-9: SEND BREAK CHARACTER SEQUENCE



25.5.2 SYNCHRONOUS SLAVE MODE

The following bits are used to configure the EUSART for synchronous slave operation:

- SYNC = 1
- CSRC = 0
- SREN = 0 (for transmit); SREN = 1 (for receive)
- CREN = 0 (for transmit); CREN = 1 (for receive)
- SPEN = 1

Setting the SYNC bit of the TXxSTA register configures the device for synchronous operation. Clearing the CSRC bit of the TXxSTA register configures the device as a slave. Clearing the SREN and CREN bits of the RCxSTA register ensures that the device is in the Transmit mode, otherwise the device will be configured to receive. Setting the SPEN bit of the RCxSTA register enables the EUSART.

25.5.2.1 EUSART Synchronous Slave Transmit

The operation of the Synchronous Master and Slave modes are identical (see **Section 25.5.1.3 "Synchronous Master Transmission")**, except in the case of the Sleep mode.

If two words are written to the TXxREG and then the SLEEP instruction is executed, the following will occur:

1. The first character will immediately transfer to the TSR register and transmit.
2. The second word will remain in the TXxREG register.
3. The TXIF bit will not be set.
4. After the first character has been shifted out of TSR, the TXxREG register will transfer the second character to the TSR and the TXIF bit will now be set.
5. If the PEIE and TXIE bits are set, the interrupt will wake the device from Sleep and execute the next instruction. If the GIE bit is also set, the program will call the Interrupt Service Routine.

25.5.2.2 Synchronous Slave Transmission Set-up:

1. Set the SYNC and SPEN bits and clear the CSRC bit.
2. Clear the ANSEL bit for the CK pin (if applicable).
3. Clear the CREN and SREN bits.
4. If interrupts are desired, set the TXIE bit of the PIE1 register and the GIE and PEIE bits of the INTCON register.
5. If 9-bit transmission is desired, set the TX9 bit.
6. Enable transmission by setting the TXEN bit.
7. If 9-bit transmission is selected, insert the Most Significant bit into the TX9D bit.
8. Start transmission by writing the Least Significant eight bits to the TXxREG register.

27.2 Register Definitions: PWM Control

REGISTER 27-1: PWMxCON: PWM CONTROL REGISTER

R/W-0/0	U-0	R-0/0	R/W-0/0	U-0	U-0	U-0	U-0
PWMxEN	—	PWMxOUT	PWMxPOL	—	—	—	—
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 **PWMxEN:** PWM Module Enable bit
1 = PWM module is enabled
0 = PWM module is disabled

bit 6 **Unimplemented:** Read as ‘0’

bit 5 **PWMxOUT:** PWM Module Output Value bit

bit 4 **PWMxPOL:** PWMx Output Polarity Select bit
1 = PWM output is active-low
0 = PWM output is active-high

bit 3-0 **Unimplemented:** Read as ‘0’

REGISTER 27-2: PWMxDCH: PWM DUTY CYCLE HIGH BITS

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
PWMxDCH<7:0>							
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 7-0 **PWMxDCH<7:0>:** PWM Duty Cycle Most Significant bits
These bits are the MSbs of the PWM duty cycle. The two LSbs are found in the PWMxDCL register.

REGISTER 27-3: PWMxDCL: PWM DUTY CYCLE LOW BITS

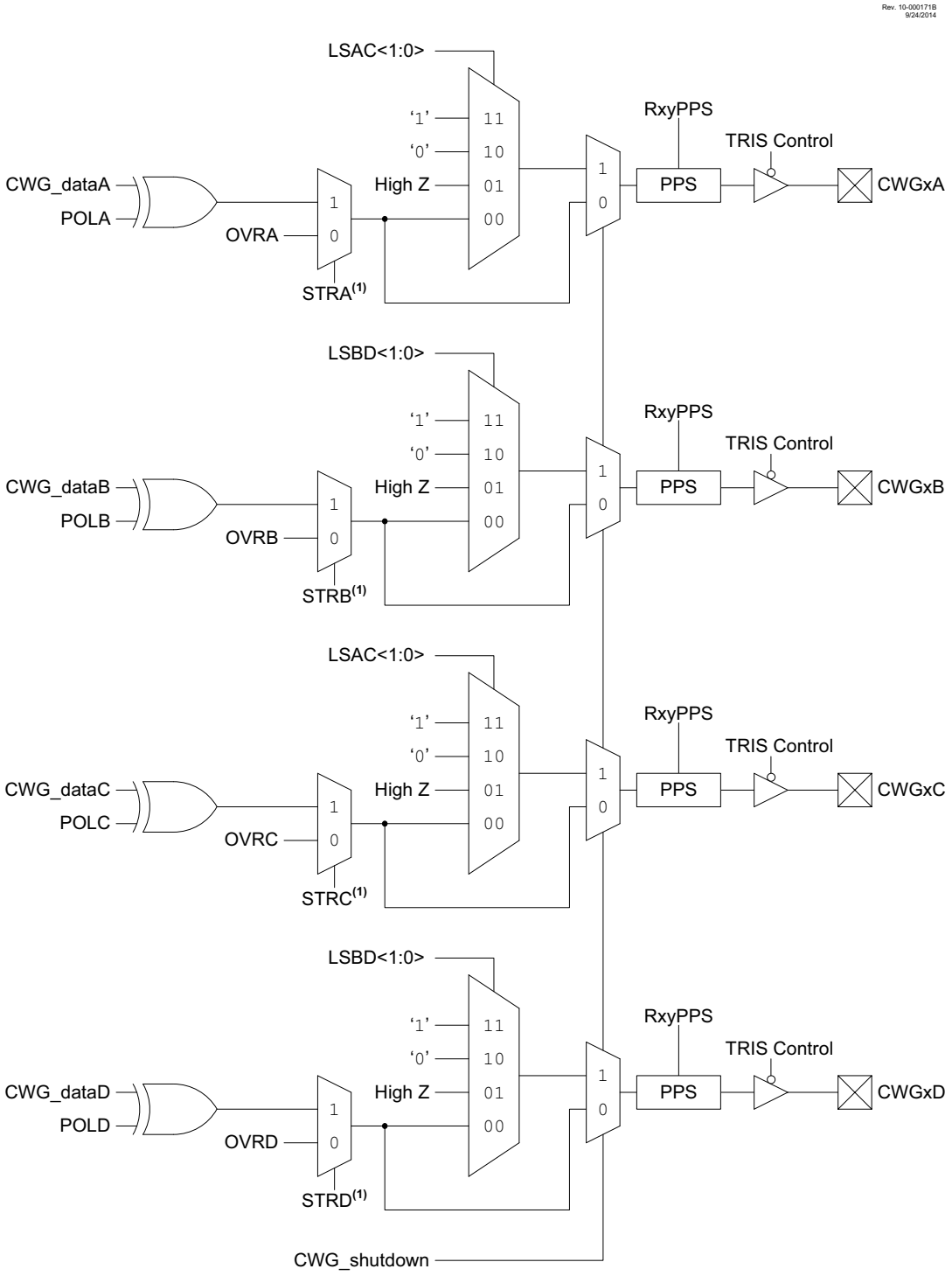
R/W-x/u	R/W-x/u	U-0	U-0	U-0	U-0	U-0	U-0
PWMxDCL<7:6>		—	—	—	—	—	—
bit 7							bit 0

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 7-6 **PWMxDCL<7:6>:** PWM Duty Cycle Least Significant bits
These bits are the LSbs of the PWM duty cycle. The MSbs are found in the PWMxDCH register.

bit 5-0 **Unimplemented:** Read as ‘0’

FIGURE 28-5: CWG OUTPUT BLOCK DIAGRAM



Note 1: STRx is held to 1 in all modes other than Output Steering Mode.

REGISTER 31-22: ATxCSELy: ANGULAR TIMER CAPTURE INPUT SELECT y REGISTER

U-0	U-0	U-0	U-0	U-0	R/W-0/0	R/W-0/0	R/W-0/0
—	—	—	—	—	CPyS<2:0>		
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 7-3 **Unimplemented:** Read as ‘0’

bit 2-0 **CPyS<2:0>:** Capture Input Source Select bits
111 = CWG_interrupt
110 = LC4_out
101 = LC3_out
100 = LC2_out
111 = LC1_out
010 = cmp2_sync
001 = cmp1_sync
000 = ATxCCy pin

REGISTER 32-12: PIDxOUTU: PID OUTPUT UPPER REGISTER

U-0	U-0	U-0	U-0	U-0	R/W-0/0	R/W-0/0	R/W-0/0
—	—	—	—	—	OUT<34:32>		
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 7-3 **Unimplemented:** Read as ‘0’

bit 2-0 **OUT<34:32>:** Bits <34:32> of OUT. OUT is the output value of the PID after completing the designated calculation on the specified inputs.

REGISTER 32-13: PIDxOUTHH: PID OUTPUT HIGH HIGH REGISTER

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
OUT<31:24>							
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 7-0 **OUT<31:24>:** Bits <31:24> of OUT. OUT is the output value of the PID after completing the designated calculation on the specified inputs.

Note: Unless otherwise noted, VIN = 5V, FOSC = 500 kHz, CIN = 0.1 µF, TA = 25°C.

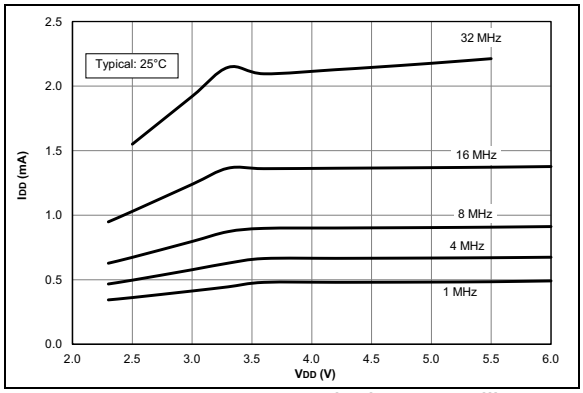


FIGURE 36-7: *IDD Typical, EC Oscillator MP Mode, PIC16F1615/9 Only.*

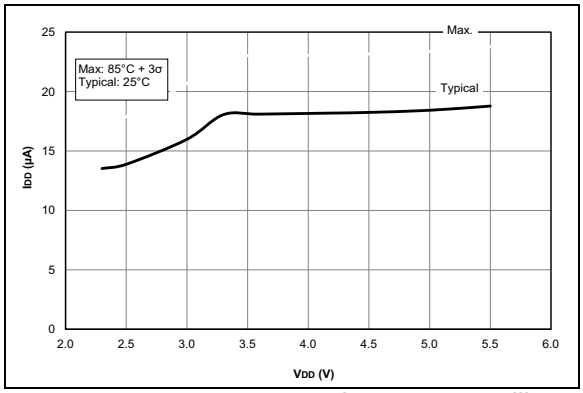


FIGURE 36-10: *IDD Maximum, EC Oscillator HP Mode, PIC16LF1615/9 Only.*

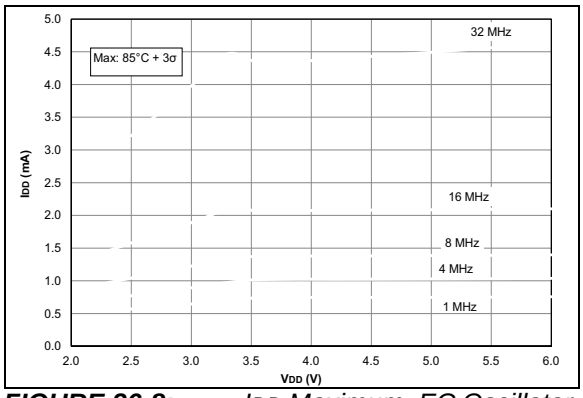


FIGURE 36-8: *IDD Maximum, EC Oscillator MP Mode, PIC16F1615/9 Only.*

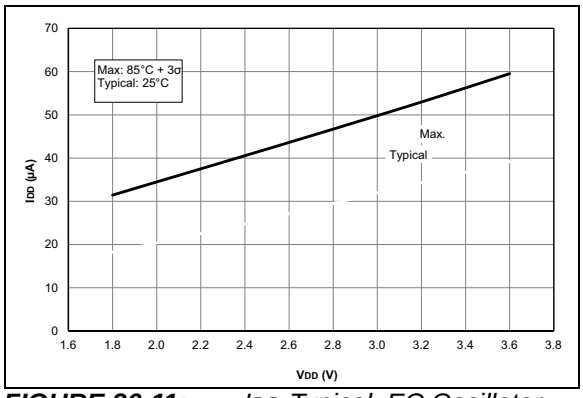


FIGURE 36-11: *IDD Typical, EC Oscillator HP Mode, PIC16F1615/9 Only.*

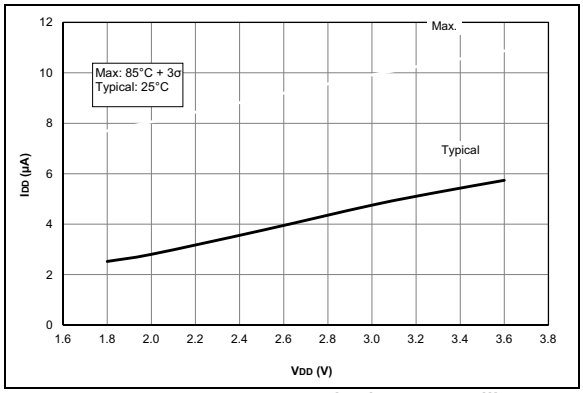


FIGURE 36-9: *IDD Typical, EC Oscillator HP Mode, PIC16LF1615/9 Only.*

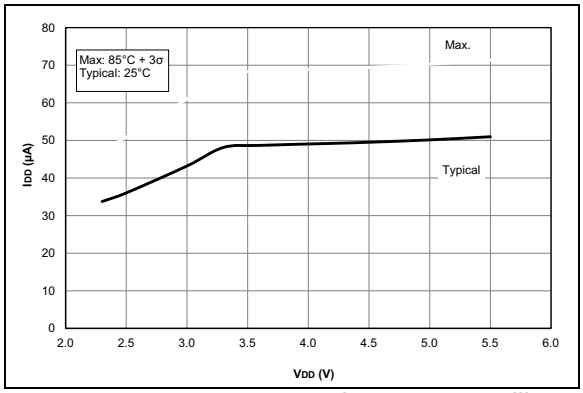


FIGURE 36-12: *IDD Maximum, EC Oscillator HP Mode, PIC16F1615/9 Only.*

Note: Unless otherwise noted, V_{IN} = 5V, F_{OSC} = 500 kHz, C_{IN} = 0.1 µF, T_A = 25°C.

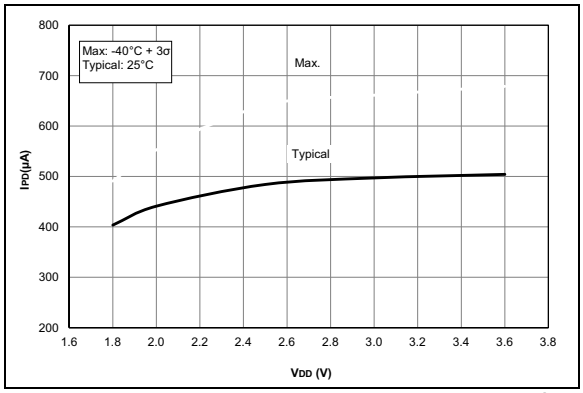


FIGURE 36-41: *IPD, Comparator, NP Mode (CxSP = 1), PIC16LF1615/9 Only.*

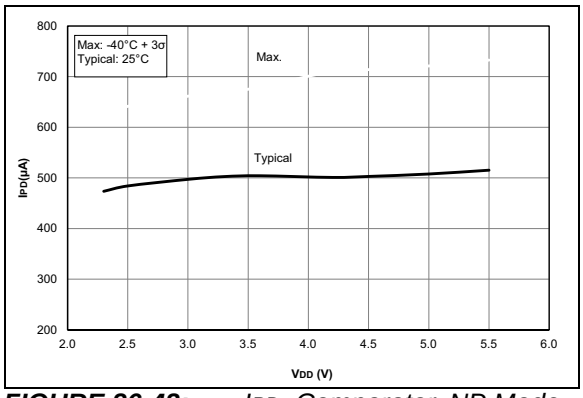


FIGURE 36-42: *IPD, Comparator, NP Mode (CxSP = 1), PIC16F1615/9 Only.*

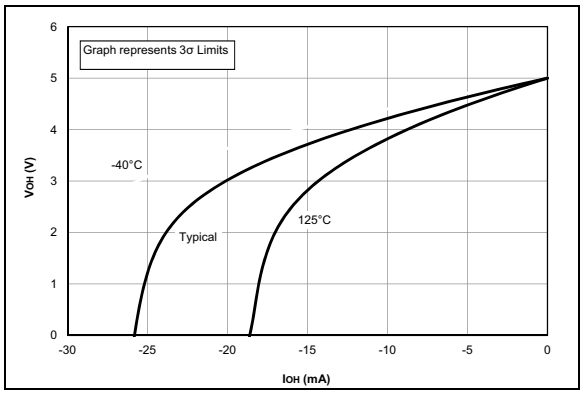


FIGURE 36-43: *VOH vs. IOH Over Temperature, VDD = 5.0V, PIC16F1615/9 Only.*

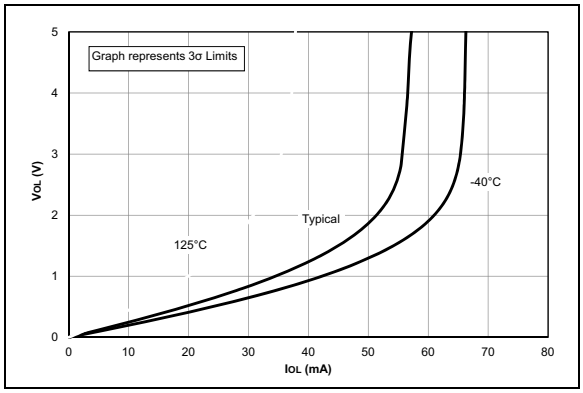


FIGURE 36-44: *VOL vs. IOL Over Temperature, VDD = 5.0V, PIC16F1615/9 Only.*

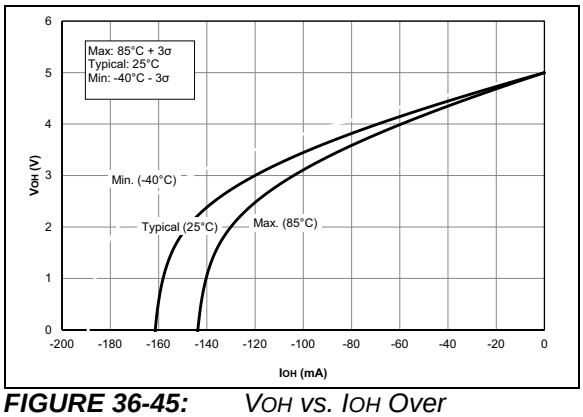


FIGURE 36-45: *VOH vs. IOH Over Temperature for High Drive Pins, VDD = 5.0V, PIC16F1615/9 Only.*

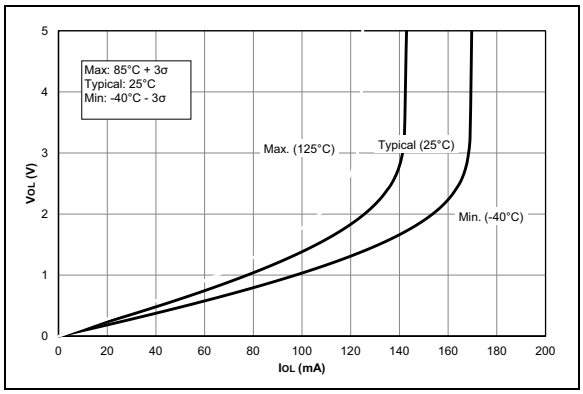


FIGURE 36-46: *VOL vs. IOL Over Temperature for High Drive Pins, VDD = 5.0V, PIC16F1615/9 Only.*

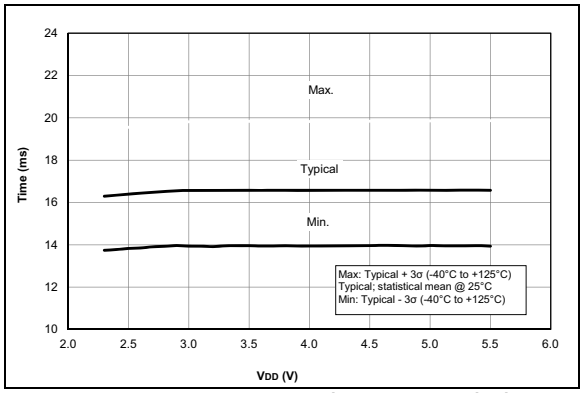


FIGURE 36-53: *WDT Time-Out Period, PIC16F1615/9 Only.*

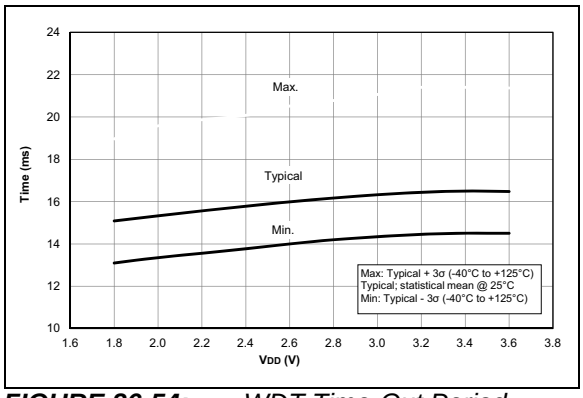


FIGURE 36-54: *WDT Time-Out Period, PIC16LF1615/9 Only.*