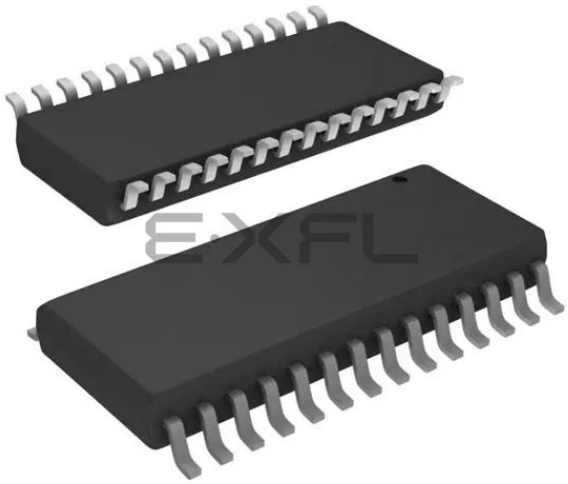


Welcome to E-XFL.COM

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, DMA, HLVD, POR, PWM, WDT
Number of I/O	25
Program Memory Size	64KB (32K x 16)
Program Memory Type	FLASH
EEPROM Size	1K x 8
RAM Size	4K x 8
Voltage - Supply (Vcc/Vdd)	2.3V ~ 5.5V
Data Converters	A/D 24x12b; D/A 1x5b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	28-SOIC (0.295", 7.50mm Width)
Supplier Device Package	28-SOIC
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18f26k42-i-so

1.3 Register and Bit naming conventions

1.3.1 REGISTER NAMES

When there are multiple instances of the same peripheral in a device, the peripheral control registers will be depicted as the concatenation of a peripheral identifier, peripheral instance, and control identifier. The control registers section will show just one instance of all the register names with an 'x' in the place of the peripheral instance number. This naming convention may also be applied to peripherals when there is only one instance of that peripheral in the device to maintain compatibility with other devices in the family that contain more than one.

1.3.2 BIT NAMES

There are two variants for bit names:

- Short name: Bit function abbreviation
- Long name: Peripheral abbreviation + short name

1.3.2.1 Short Bit Names

Short bit names are an abbreviation for the bit function. For example, some peripherals are enabled with the EN bit. The bit names shown in the registers are the short name variant.

Short bit names are useful when accessing bits in C programs. The general format for accessing bits by the short name is *RegisterName*bits.*ShortName*. For example, the enable bit, EN, in the T0CON0 register can be set in C programs with the instruction `T0CON0bits.EN = 1`.

Short names are generally not useful in assembly programs because the same name may be used by different peripherals in different bit positions. When this occurs, during the include file generation, all instances of that short bit name are appended with an underscore plus the name of the register in which the bit resides to avoid naming contentions.

1.3.2.2 Long Bit Names

Long bit names are constructed by adding a peripheral abbreviation prefix to the short name. The prefix is unique to the peripheral thereby making every long bit name unique. The long bit name for the Timer0 enable bit is the Timer0 prefix, T0, appended with the enable bit short name, EN, resulting in the unique bit name T0EN.

Long bit names are useful in both C and assembly programs. For example, in C the T0CON0 enable bit can be set with the `T0EN = 1` instruction. In assembly, this bit can be set with the `BSF  T0CON0,T0EN` instruction.

1.3.2.3 Bit Fields

Bit fields are two or more adjacent bits in the same register. For example, the four Least Significant bits of the T0CON0 register contain the output prescaler select bits. The short name for this field is OUTPS and the long name is T0OUTPS. Bit field access is only possible in C programs. The following example demonstrates a C program instruction for setting the Timer0 output prescaler to the 1:6 Postscaler:

```
T0CON0bits.OUTPS = 0x5;
```

Individual bits in a bit field can also be accessed with long and short bit names. Each bit is the field name appended with the number of the bit position within the field. For example, the Most Significant mode bit has the short bit name OUTPS3. The following two examples demonstrate assembly program sequences for setting the Timer0 output prescaler to 1:6 Postscaler:

Example 1:

```
MOVLW  ~(1<<OUTPS3 | 1<<OUTPS1)
ANDWF  T0CON0,F
MOVLW  1<<OUTPS2 | 1<<OUTPS0
IORWF  T0CON0,F
```

Example 2:

```
BCF    T0CON0,OUTPS3
BSF    T0CON0,OUTPS2
BCF    T0CON0,OUTPS1
BSF    T0CON0,OUTPS0
```

1.3.3 REGISTER AND BIT NAMING EXCEPTIONS

1.3.3.1 Status, Interrupt, and Mirror Bits

Status, interrupt enables, interrupt flags, and mirror bits are contained in registers that span more than one peripheral. In these cases, the bit name shown is unique so there is no prefix or short name variant.

REGISTER 9-2: INTCON1: INTERRUPT CONTROL REGISTER 1

R-0/0	R-0/0	U-0	U-0	U-0	U-0	U-0	U-0
STAT<1:0>		—	—	—	—	—	—
bit 7							bit 0

Legend:		
HC = Bit is cleared by hardware		
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 7-6 **STAT<1:0>:** Interrupt State Status bits

11 = High priority ISR executing, high priority interrupt was received while a low priority ISR was executing

10 = High priority ISR executing, high priority interrupt was received in main routine

01 = Low priority ISR executing, low priority interrupt was received in main routine

00 = Main routine executing

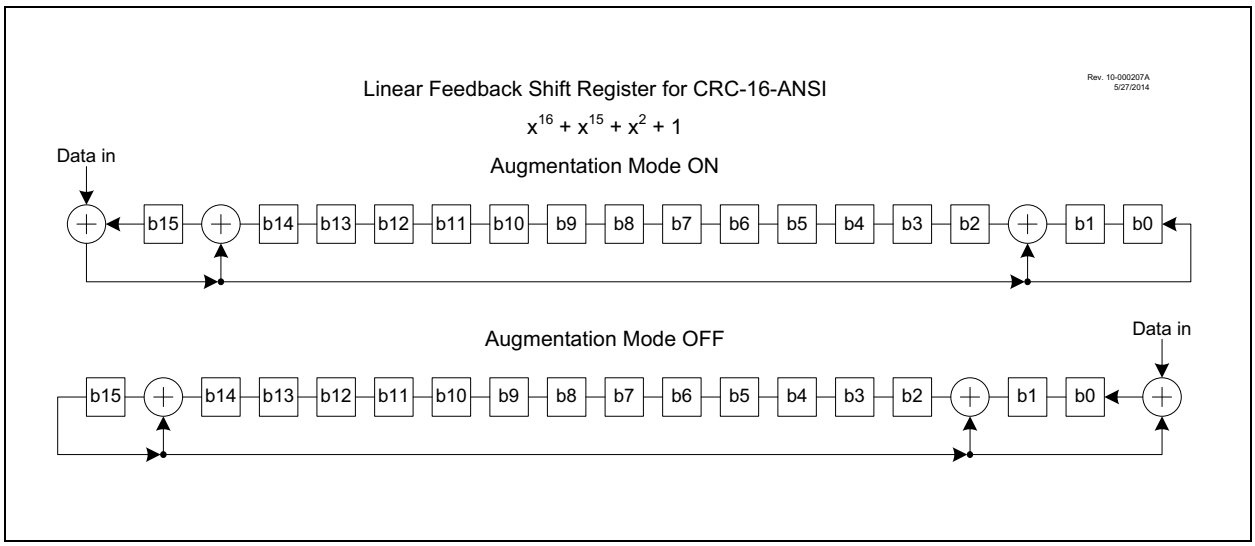
bit 5-0 **Unimplemented**: Read as ‘0’

14.3 CRC Polynomial Implementation

Any polynomial can be used. The polynomial and accumulator sizes are determined by the PLEN<3:0> bits. For an n-bit accumulator, PLEN = n-1 and the corresponding polynomial is n+1 bits. Therefore the accumulator can be any size up to 16 bits with a corresponding polynomial up to 17 bits. The MSb and LSb of the polynomial are always '1' which is forced by hardware. All polynomial bits between the MSb and LSb are specified by the CRCXOR registers. For example, when using CRC-16-ANSI, the polynomial is defined as $X^{16}+X^{15}+X^2+1$.

The X^{16} and X^0 = 1 terms are the MSb and LSb controlled by hardware. The X^{15} and X^2 terms are specified by setting the corresponding CRCXOR<15:0> bits with the value of '0x8004'. The actual value is '0x8005' because the hardware sets the LSb to 1. However, the LSb of the CRCXORL register is unimplemented and always reads as '0'. Refer to Example 14-1.

EXAMPLE 14-2: CRC LFSR EXAMPLE



14.4 CRC Data Sources

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

- User data using the CRCDAT registers (CRCDATH and CRCDATL)
- Program memory 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 CRCDAT registers up to DLEN will be used, other data bits in CRCDAT 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 (Big Endian). 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 (Little Endian).

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.

14.4.1 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.

14.4.2 CRC FROM FLASH

To use the CRC module on data located in Program memory, the user can initialize the Program Memory Scanner as defined in **Section 14.8, Scanner Module Overview**.

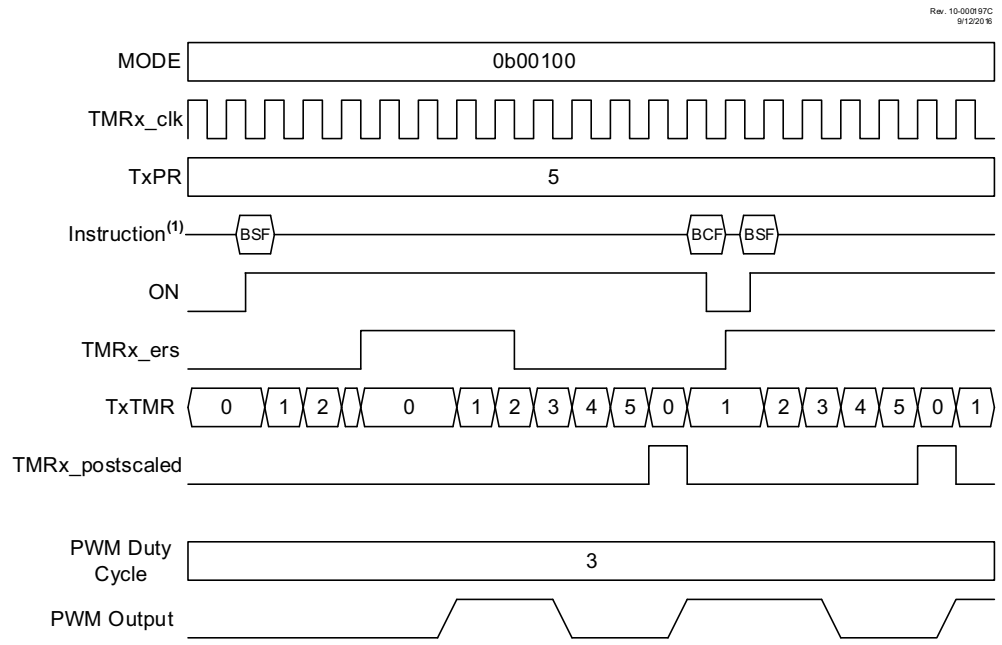
22.5.3 EDGE-TRIGGERED HARDWARE LIMIT MODE

In Hardware Limit mode the timer can be reset by the TMRx_ers external signal before the timer reaches the period count. Three types of Resets are possible:

- Reset on rising or falling edge (MODE<4:0> = 00011)
- Reset on rising edge (MODE<4:0> = 0010)
- Reset on falling edge (MODE<4:0> = 00101)

When the timer is used in conjunction with the CCP in PWM mode then an early Reset shortens the period and restarts the PWM pulse after a two clock delay. Refer to Figure 22-6.

FIGURE 22-6: EDGE TRIGGERED HARDWARE LIMIT MODE TIMING DIAGRAM (MODE=00100)



Note 1: BSF and BCF represent Bit-Set File and Bit-Clear File instructions executed by the CPU to set or clear the ON bit of TxCON. CPU execution is asynchronous to the timer clock input.

REGISTER 22-6: TxHLT: TIMERx HARDWARE LIMIT CONTROL 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
PSYNC	CKPOL	CKSYNC	MODE<4:0>				
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 **PSYNC:** Timerx Prescaler Synchronization Enable bit(1, 2)
1 = TxTMR Prescaler Output is synchronized to Fosc/4
0 = TxTMR Prescaler Output is not synchronized to Fosc/4

bit 6 **CKPOL:** Timerx Clock Polarity Selection bit(3)
1 = Falling edge of input clock clocks timer/prescaler
0 = Rising edge of input clock clocks timer/prescaler

bit 5 **CKSYNC:** Timerx Clock Synchronization Enable bit(4, 5)
1 = ON register bit is synchronized to T2TMR_clk input
0 = ON register bit is not synchronized to T2TMR_clk input

bit 4-0 **MODE<4:0>:** Timerx Control Mode Selection bits(6, 7)
See Table 22-1 for all operating modes.

Note 1: Setting this bit ensures that reading TxTMR will return a valid data value.
2: When this bit is '1', Timer2 cannot operate in Sleep mode.
3: CKPOL should not be changed while ON = 1.
4: Setting this bit ensures glitch-free operation when the ON is enabled or disabled.
5: When this bit is set then the timer operation will be delayed by two TxTMR input clocks after the ON bit is set.
6: Unless otherwise indicated, all modes start upon ON = 1 and stop upon ON = 0 (stops occur without affecting the value of TxTMR).
7: When TxTMR = TxPR, the next clock clears TxTMR, regardless of the operating mode.

FIGURE 25-7: PERIOD AND DUTY-CYCLE SINGLE ACQUISITION TIMING DIAGRAM

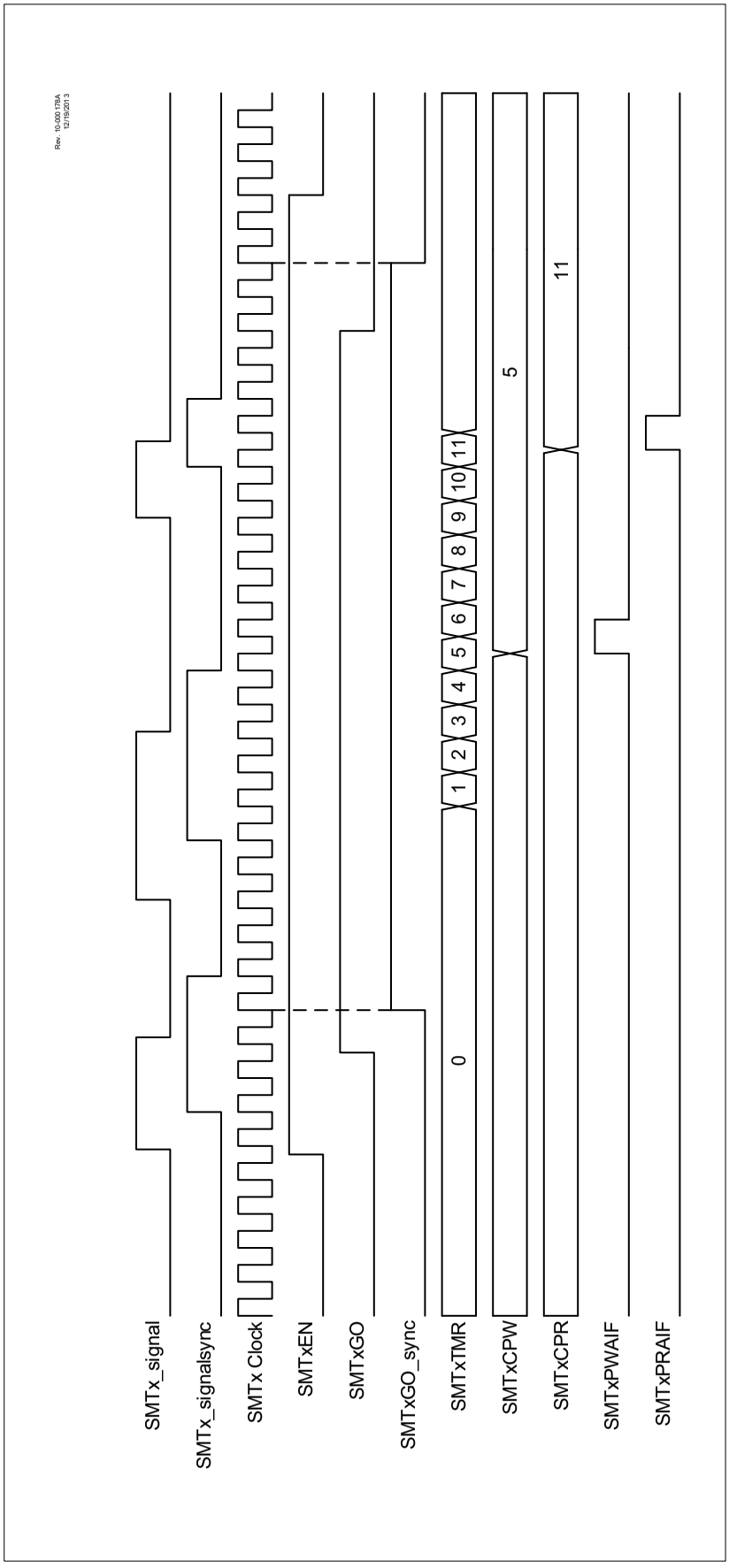


FIGURE 25-8: HIGH AND LOW MEASURE MODE REPEAT ACQUISITION TIMING DIAGRAM

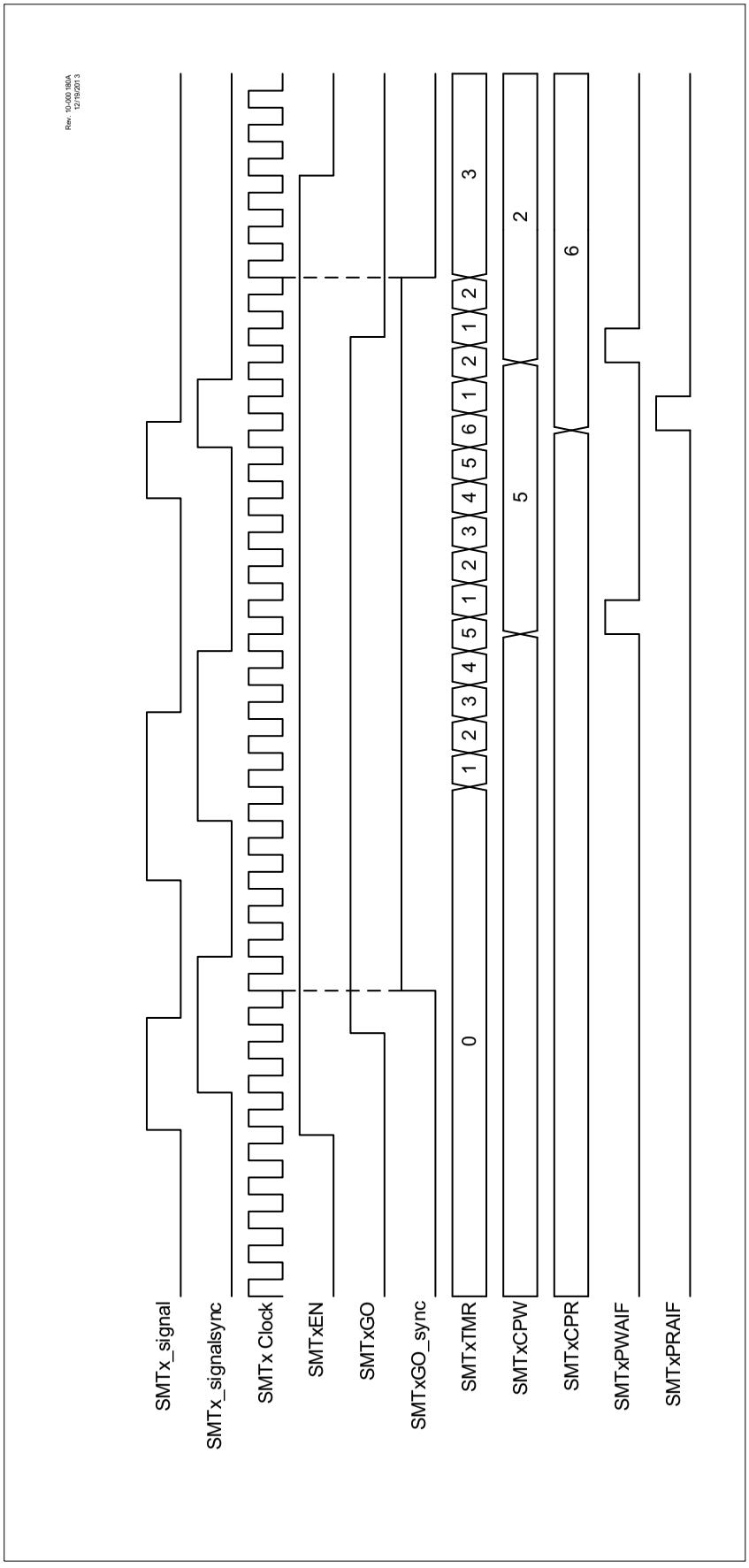


FIGURE 26-6: SIMPLIFIED CWG BLOCK DIAGRAM (FORWARD AND REVERSE FULL-BRIDGE MODES)

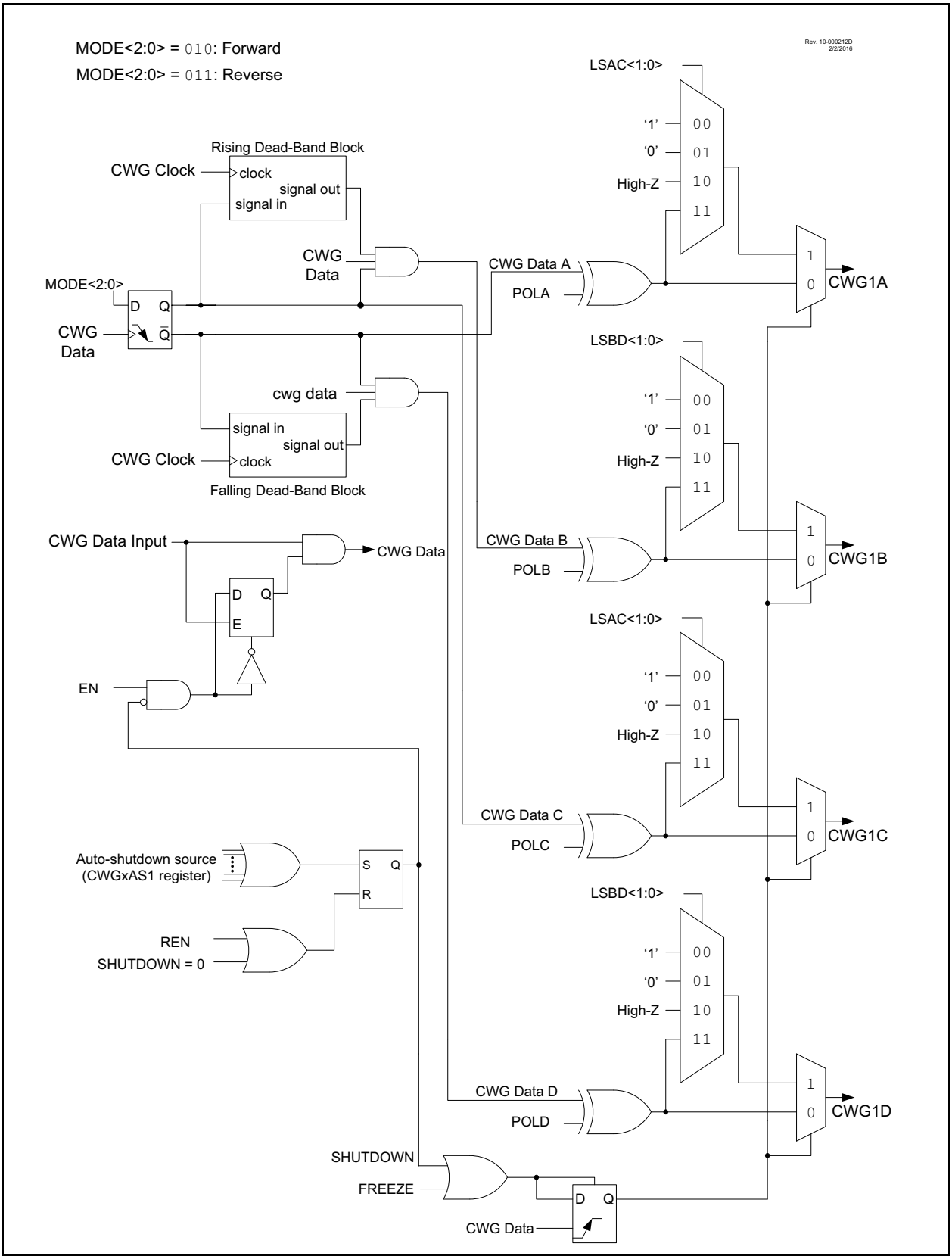
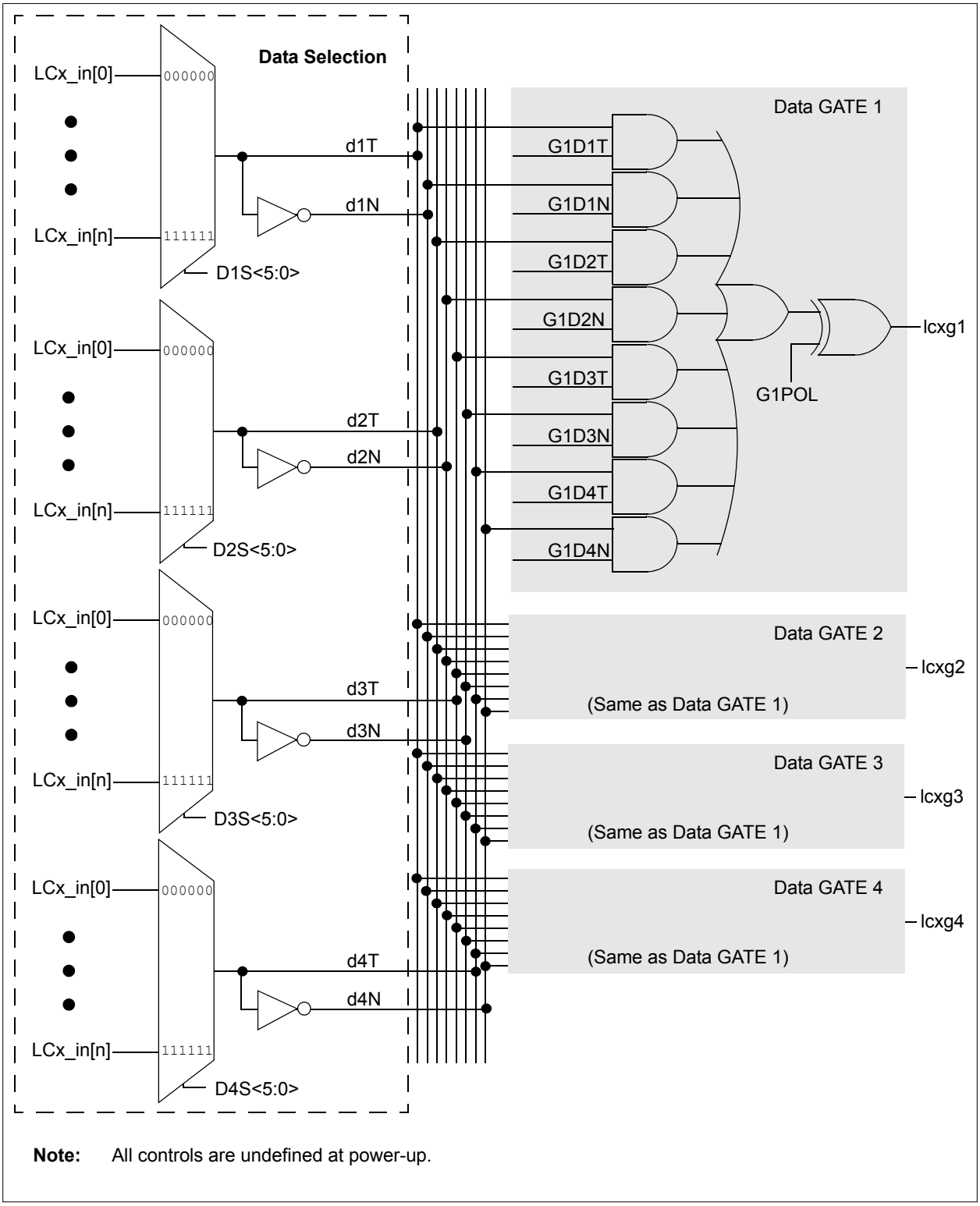


FIGURE 27-2: INPUT DATA SELECTION AND GATING



Note: All controls are undefined at power-up.

32.5.6 MASTER MODE SPI CLOCK CONFIGURATION

32.5.6.1 SPI Clock Selection

The clock source for SPI master modes is selected by the SPIxCLK register. Selections include the following:

- FOSC
- HFINTOSC
- CLKREF
- Timer0_overflow
- Timer2_Postscaled
- Timer4_Postscaled
- Timer6_Postscaled
- SMT_match

The SPIxBAUD register allows for dividing this clock. The frequency of the SCK output is defined by Equation 32-1:

EQUATION 32-1: FREQUENCY OF SCK OUTPUT SIGNAL

$$F_{BAUD} = \frac{F_{CSEL}}{(2 \cdot (BAUD + 1))}$$

where FBAUD is the baud rate frequency output on the SCK pin, FCSEL is the frequency of the input clock selected by the SPIxCLK register, and BAUD is the value contained in the SPIxBAUD register.

32.5.6.2 CKE, CKP and SMP

The CKP, CKE, and SMP bits control the relationship between the SCK clock output, SDO output data changes, and SDI input data sampling. The bit functions are as follows:

- CKP - SCK output polarity
- CKE - SDO output change relative to the SCK clock
- SMP - SDI input sampling relative to the clock edges

The CKE bit, when set, inverts the low Idle state of the SCK output to a high Idle state.

Figure 32-7 through Figure 32-10 illustrate the eight possible combinations of the CKP, CKE, and SMP bit selections.

When the CKE bit is set, the SDO data is valid before there is a clock edge on SCK. When the CKE bit is cleared, the SDO data is undefined prior to the first SCK edge.

Note:	All timing diagrams assume the LSBF bit of SPIxCON0 is cleared.

REGISTER 32-4: SPIxTCNTH: SPI TRANSFER COUNTER MSB REGISTER

U-0	U-0	U-0	U-0	U-0	R/W-0/0	R/W-0/0	R/W-0/0
—	—	—	—	—	TCNT10	TCNT9	TCNT8
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'

bit 7-3 **Unimplemented**: Read as '0'

bit 2-0 TCNT<10:8>:

BMODE = 0

Bits 13-11 of the Transfer Counter, counting the total number of bits to transfer

BMODE = 1

Bits 10-8 of the Transfer Counter, counting the total number of bytes to transfer

Note: This register should not be written to while a transfer is in progress (BUSY bit of SPIxCON2 is set).

REGISTER 32-5: SPIxTWIDTH: SPI TRANSFER WIDTH REGISTER

U-0	U-0	U-0	U-0	U-0	R/W-0/0	R/W-0/0	R/W-0/0
—	—	—	—	—	TWIDTH2	TWIDTH1	TWIDTH0
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'

bit 7-3 **Unimplemented**: Read as '0'

bit 2-0 TWIDTH<2:0>:

BMODE = 0

Bits 2-0 of the Transfer Counter, counting the total number of bits to transfer

BMODE = 1

Size (in bits) of each transfer counted by the transfer counter

111 = 7 bits

110 = 6 bits

101 = 5 bits

100 = 4 bits

011 = 3 bits

010 = 2 bits

001 = 1 bit

000 = 8 bits

Note: This register should not be written to while a transfer is in progress (BUSY bit of SPIxCON2 is set).

33.5.11 MASTER TRANSMISSION IN 10-BIT ADDRESSING MODE

This section describes the sequence of events for the I^2C module configured as an I^2C master in 10-bit Addressing mode and is transmitting data. Figure 33-21 is used as a visual reference for this description

1. If ABD = 0; i.e., Address buffers are enabled

Master software loads number of bytes to be transmitted in one sequence in I2CxCNT, high address byte of slave address in I2CxADB1 with R/W = 0, low address byte in I2CxADB0 and the first byte of data in I2CxTXB. Master software has to set the Start (S) bit to initiate communication.

If ABD = 1; i.e., Address buffers are disabled

Master software loads the number of bytes to be transmitted in one sequence in I2CxCNT and the high address byte of the slave address with R/W = 0 into the I2CxTXB register. Writing to the I2CxTXB will assert the start condition on the bus and sets the S bit. Software writes to the S bit are ignored in this case.

2. Master hardware waits for BFRE bit to be set; then shifts out the start and high address and waits for acknowledge.
3. If NACK, master hardware sends Stop.
4. If ABD = 0; i.e., Address buffer are enabled

If ACK, master hardware sends the low address byte from I2CxADB0.

If ABD = 1; i.e., Address buffer are disabled

If ACK, master hardware sets TXIF and MDR bits and the software has to write the low address byte into I2CxTXB. Writing to I2CxTXB sends the low address on the bus.

5. If TXBE = 1 and I2CxCNT! = 0, I2CxTXIF and MDR bits are set. Clock is stretched on 8th falling SCL edge until master software writes next data byte to I2CxTXB.
6. Master hardware sends ninth SCL pulse for ACK from slave and loads the shift register from I2CxTXB. I2CxCNT is decremented.
7. If slave sends a NACK, master hardware sends Stop and ends transmission.
8. If slave sends an ACK, master hardware outputs data in the shift register on SDA. I2CxCNT value is checked on the 8th falling SCL edge. If I2CxCNT = 0; master hardware sends 9th SCL pulse for ACK and CNTIF is set.
9. If I2CxCNT! = 0; go to step 5.

36.2 ADC Operation

36.2.1 STARTING A CONVERSION

To enable the ADC module, the ON bit of the ADCON0 register must be set to a '1'. A conversion may be started by any of the following:

- Software setting the GO bit of ADCON0 to '1'
- An external trigger (selected by Register 36-3)
- A continuous-mode retrigger (see section **Section 36.6.8 "Continuous Sampling mode"**)

.

Note:	The GO bit should not be set in the same instruction that turns on the ADC. Refer to **Section 36.2.6 "ADC Conversion Procedure (Basic Mode)"**.

36.2.2 COMPLETION OF A CONVERSION

When any individual conversion is complete, the value already in ADRES is written into PREV (if ADPSIS = 1) and the new conversion results appear in ADRES. When the conversion completes, the ADC module will:

- Clear the GO bit (unless the CONT bit of ADCON0 is set)
- Set the ADIF Interrupt Flag bit
- Set the MATH bit
- Update ACC

When ADDSEN = 0 then after every conversion, or when ADDSEN = 1 then after every other conversion, the following events occur:

- ERR is calculated
- ADTIF is set if ERR calculation meets threshold comparison

Importantly, filter and threshold computations occur after the conversion itself is complete. As such, interrupt handlers responding to ADIF should check ADTIF before reading filter and threshold results.

36.2.3 ADC OPERATION DURING SLEEP

The ADC module can operate during Sleep. This requires the ADC clock source to be set to the FRC option. When the FRC oscillator source is selected, the ADC waits one additional instruction before starting the conversion. This allows the SLEEP instruction to be executed, which can reduce system noise during the conversion. If the ADC interrupt is enabled, the device will wake-up from Sleep when the conversion completes. If the ADC interrupt is disabled, the ADC module is turned off after the conversion completes, although the ON bit remains set.

36.2.4 EXTERNAL TRIGGER DURING SLEEP

If the external trigger is received during sleep while ADC clock source is set to the FRC, ADC module will perform the conversion and set the ADIF bit upon completion.

If an external trigger is received when the ADC clock source is something other than FRC, the trigger will be recorded, but the conversion will not begin until the device exits Sleep.

REGISTER 36-35: ADACT: ADC AUTO CONVERSION TRIGGER CONTROL REGISTER

U-0	U-0	U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
—	—	—	ACT<4:0>				
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 7-5 **Unimplemented**: Read as '0'

bit 4-0 **ADACT<4:0>**: Auto-Conversion Trigger Select Bits

11111 = Reserved, do not use
•
•
•
11110 = Reserved, do not use
11101 = Software write to ADPCH
11100 = Reserved, do not use
11011 = Software read of ADRESH
11010 = Software read of ADERRH
11001 = CLC4_out
11000 = CLC3_out
10111 = CLC2_out
10110 = CLC1_out
10101 = Logical OR of all Interrupt-on-change Interrupt Flags
10100 = CMP2_out
10011 = CMP1_out
10010 = NCO1_out
10001 = PWM8_out
10000 = PWM7_out
01111 = PWM6_out
01110 = PWM5_out
01101 = CCP4_trigger
01100 = CCP3_trigger
01011 = CCP2_trigger
01010 = CCP1_trigger
01001 = SMT1_trigger
01000 = TMR6_postscaled
00111 = TMR5_overflow
00110 = TMR4_postscaled
00101 = TMR3_overflow
00100 = TMR2_postscaled
00011 = TMR1_overflow
00010 = TMR0_overflow
00001 = Pin selected by ADACTPPS
00000 = External Trigger Disabled

BNC **Branch if Not Carry**

Syntax: 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 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 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: 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 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 BNN Jump`

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

DECF **Decrement f**

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

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (f) – 1 → dest

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

Encoding:

0000	01da	ffff	ffff

Description: Decrement register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default).
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 41.2.3 "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: `DECF CNT, 1, 0`

Before Instruction
CNT = 01h
Z = 0
After Instruction
CNT = 00h
Z = 1

DECFSZ **Decrement f, skip if 0**

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

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

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

Status Affected: None

Encoding:

0010	11da	ffff	ffff

Description: The contents of register 'f' are decremented. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed back in register 'f' (default).
If the result is '0', the next instruction, which is already fetched, is discarded and a NOP is executed instead, making it a 2-cycle instruction.
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 41.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

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

If skip:

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

If skip and followed by 2-word instruction:

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

Example:
```
HERE    DECFSZ  CNT, 1, 1
        GOTO    LOOP
CONTINUE
```

Before Instruction
PC = Address (HERE)
After Instruction
CNT = CNT - 1
If CNT = 0;
PC = Address (CONTINUE)
If CNT ≠ 0;
PC = Address (HERE + 2)

ADDWF **ADD W to Indexed (Indexed Literal Offset mode)**

Syntax: ADDWF [k] {,d}

Operands: $0 \le k \le 95$
$d \in [0,1]$

Operation: (W) + ((FSR2) + k) → dest

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

Encoding:

0010	01d0	kkkk	kkkk

Description: The contents of W are added to the contents of the register indicated by FSR2, offset by the value 'k'.
If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default).

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: ADDWF [OFST], 0

Before Instruction
W = 17h
OFST = 2Ch
FSR2 = 0A00h
Contents of 0A2Ch = 20h
After Instruction
W = 37h
Contents of 0A2Ch = 20h

BSF **Bit Set Indexed (Indexed Literal Offset mode)**

Syntax: BSF [k], b

Operands: $0 \le f \le 95$
$0 \le b \le 7$

Operation: 1 → ((FSR2) + k)<b>

Status Affected: None

Encoding:

1000	bbb0	kkkk	kkkk

Description: Bit 'b' of the register indicated by FSR2, offset by the value 'k', is set.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: BSF [FLAG_OFST], 7

Before Instruction
FLAG_OFST = 0Ah
FSR2 = 0A00h
Contents of 0A0Ah = 55h
After Instruction
Contents of 0A0Ah = D5h

SETF **Set Indexed (Indexed Literal Offset mode)**

Syntax: SETF [k]

Operands: $0 \le k \le 95$

Operation: FFh → ((FSR2) + k)

Status Affected: None

Encoding:

0110	1000	kkkk	kkkk

Description: The contents of the register indicated by FSR2, offset by 'k', are set to FFh.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read 'k'	Process Data	Write register

Example: SETF [OFST]

Before Instruction
OFST = 2Ch
FSR2 = 0A00h
Contents of 0A2Ch = 00h
After Instruction
Contents of 0A2Ch = FFh

43.6 MPLAB X SIM Software Simulator

The MPLAB X SIM Software Simulator allows code development in a PC-hosted environment by simulating the PIC MCUs and dsPIC DSCs on an instruction level. On any given instruction, the data areas can be examined or modified and stimuli can be applied from a comprehensive stimulus controller. Registers can be logged to files for further run-time analysis. The trace buffer and logic analyzer display extend the power of the simulator to record and track program execution, actions on I/O, most peripherals and internal registers.

The MPLAB X SIM Software Simulator fully supports symbolic debugging using the MPLAB XC Compilers, and the MPASM and MPLAB Assemblers. The software simulator offers the flexibility to develop and debug code outside of the hardware laboratory environment, making it an excellent, economical software development tool.

43.7 MPLAB REAL ICE In-Circuit Emulator System

The MPLAB REAL ICE In-Circuit Emulator System is Microchip's next generation high-speed emulator for Microchip Flash DSC and MCU devices. It debugs and programs all 8, 16 and 32-bit MCU, and DSC devices with the easy-to-use, powerful graphical user interface of the MPLAB X IDE.

The emulator is connected to the design engineer's PC using a high-speed USB 2.0 interface and is connected to the target with either a connector compatible with in-circuit debugger systems (RJ-11) or with the new high-speed, noise tolerant, Low-Voltage Differential Signal (LVDS) interconnection (CAT5).

The emulator is field upgradable through future firmware downloads in MPLAB X IDE. MPLAB REAL ICE offers significant advantages over competitive emulators including full-speed emulation, run-time variable watches, trace analysis, complex breakpoints, logic probes, a ruggedized probe interface and long (up to three meters) interconnection cables.

43.8 MPLAB ICD 3 In-Circuit Debugger System

The MPLAB ICD 3 In-Circuit Debugger System is Microchip's most cost-effective, high-speed hardware debugger/programmer for Microchip Flash DSC and MCU devices. It debugs and programs PIC Flash microcontrollers and dsPIC DSCs with the powerful, yet easy-to-use graphical user interface of the MPLAB IDE.

The MPLAB ICD 3 In-Circuit Debugger probe is connected to the design engineer's PC using a high-speed USB 2.0 interface and is connected to the target with a connector compatible with the MPLAB ICD 2 or MPLAB REAL ICE systems (RJ-11). MPLAB ICD 3 supports all MPLAB ICD 2 headers.

43.9 PICkit 3 In-Circuit Debugger/ Programmer

The MPLAB PICkit 3 allows debugging and programming of PIC and dsPIC Flash microcontrollers at a most affordable price point using the powerful graphical user interface of the MPLAB IDE. The MPLAB PICkit 3 is connected to the design engineer's PC using a full-speed USB interface and can be connected to the target via a Microchip debug (RJ-11) connector (compatible with MPLAB ICD 3 and MPLAB REAL ICE). The connector uses two device I/O pins and the Reset line to implement in-circuit debugging and In-Circuit Serial Programming™ (ICSP™).

43.10 MPLAB PM3 Device Programmer

The MPLAB PM3 Device Programmer is a universal, CE compliant device programmer with programmable voltage verification at V_{DDMIN} and V_{DDMAX} for maximum reliability. It features a large LCD display (128 x 64) for menus and error messages, and a modular, detachable socket assembly to support various package types. The ICSP cable assembly is included as a standard item. In Stand-Alone mode, the MPLAB PM3 Device Programmer can read, verify and program PIC devices without a PC connection. It can also set code protection in this mode. The MPLAB PM3 connects to the host PC via an RS-232 or USB cable. The MPLAB PM3 has high-speed communications and optimized algorithms for quick programming of large memory devices, and incorporates an MMC card for file storage and data applications.

44.0 ELECTRICAL SPECIFICATIONS

44.1 Absolute Maximum Ratings(†)

Ambient temperature under bias ..... -40°C to +125°C
Storage temperature ..... -65°C to +150°C
Voltage on pins with respect to V_{SS}
- on V_{DD} pin
  - PIC18F26/45/46/55/56K42 ..... -0.3V to +6.5V
  - PIC18(L)F26/27/45/46/47/55/56/57K42 ..... -0.3V to +4.0V
- on $\overline{MCLR}$ pin ..... -0.3V to +9.0V
- on all other pins ..... -0.3V to (V_{DD} + 0.3V)

Maximum current
- on V_{SS} pin(1)
  - -40°C ≤ T_A ≤ +85°C ..... 350 mA
  - 85°C < T_A ≤ +125°C ..... 120 mA
- on V_{DD} pin for 28-Pin devices(1)
  - -40°C ≤ T_A ≤ +85°C ..... 250 mA
  - 85°C < T_A ≤ +125°C ..... 85 mA
- on V_{DD} pin for 40-Pin devices(1)
  - -40°C ≤ T_A ≤ +85°C ..... 350 mA
  - 85°C < T_A ≤ +125°C ..... 120 mA
- on any standard I/O pin ..... ±50 mA

Clamp current, I_K (V_{PIN} < 0 or V_{PIN} > V_{DD}) ..... ±20 mA
Total power dissipation(2) ..... 800 mW

Note 1: Maximum current rating requires even load distribution across I/O pins. Maximum current rating may be limited by the device package power dissipation characterizations, see Table 44-8 to calculate device specifications.

2: Power dissipation is calculated as follows:

$P_{DIS} = V_{DD} \times \{I_{DD} - \Sigma I_{OH}\} + \Sigma \{(V_{DD} - V_{OH}) \times I_{OH}\} + \Sigma (V_{OI} \times I_{OL})$

† NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure above maximum rating conditions for extended periods may affect device reliability.

TABLE 44-9: EXTERNAL CLOCK/OSCILLATOR TIMING REQUIREMENTS (CONTINUED)

Standard Operating Conditions (unless otherwise stated)							
Param No.	**Sym.**	**Characteristic**	**Min.**	**Typ†**	**Max.**	**Units**	**Conditions**
OS21	F_{CY}	Instruction Frequency	—	FOSC/4	—	MHz	
OS22	T_{CY}	Instruction Period	62.5	$1/F_{CY}$	—	ns	

* These parameters are characterized but not tested.

† Data in "Typ" column is at 3.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: Instruction cycle period (TCY) equals four times the input oscillator time base period. All specified values are based on characterization data for that particular oscillator type under standard operating conditions with the device executing code. Exceeding these specified limits may result in an unstable oscillator operation and/or higher than expected current consumption. All devices are tested to operate at "min" values with an external clock applied to OSC1 pin. When an external clock input is used, the "max" cycle time limit is "DC" (no clock) for all devices.

2: The system clock frequency (FOSC) is selected by the "main clock switch controls" as described in **Section 10.0 "Power-Saving Operation Modes"**.

3: The system clock frequency (FOSC) must meet the voltage requirements defined in the **Section 44.2 "Standard Operating Conditions"**.

4: LP, XT and HS oscillator modes require an appropriate crystal or resonator to be connected to the device. For clocking the device with the external square wave, one of the EC mode selections must be used.