Microchip Technology - PIC17C766T-33/L Datasheet

Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	PIC
Core Size	8-Bit
Speed	33MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	66
Program Memory Size	32KB (16K x 16)
Program Memory Type	OTP
EEPROM Size	-
RAM Size	902 x 8
Voltage - Supply (Vcc/Vdd)	4.5V ~ 5.5V
Data Converters	A/D 16x10b
Oscillator Type	External
Operating Temperature	0°C ~ 70°C (TA)
Mounting Type	Surface Mount
Package / Case	84-LCC (J-Lead)
Supplier Device Package	84-PLCC (29.31x29.31)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic17c766t-33-l

TABLE 3-1: PINOUT DESCRIPTIONS (CONTINUED)

Name	PIC17C75X			PIC17C76X		I/O/P Type	Buffer Type	Description
	DIP No.	PLCC No.	TQFP No.	PLCC No.	QFP No.			
								PORTC is a bi-directional I/O Port.
RC0/AD0	2	3	58	3	72	I/O	TTL	This is also the least significant byte (LSB) of the 16-bit wide system bus in Microprocessor mode or Extended Microcontroller mode. In multiplexed system bus configuration, these pins are address output as well as data input or output.
RC1/AD1	63	67	55	83	69	I/O	TTL	
RC2/AD2	62	66	54	82	68	I/O	TTL	
RC3/AD3	61	65	53	81	67	I/O	TTL	
RC4/AD4	60	64	52	80	66	I/O	TTL	
RC5/AD5	58	63	51	79	65	I/O	TTL	
RC6/AD6	58	62	50	78	64	I/O	TTL	
RC7/AD7	57	61	49	77	63	I/O	TTL	
								PORTD is a bi-directional I/O Port.
RD0/AD8	10	11	2	15	4	I/O	TTL	This is also the most significant byte (MSB) of the 16-bit system bus in Microprocessor mode or Extended Microcontroller mode. In multiplexed system bus configuration, these pins are address output as well as data input or output.
RD1/AD9	9	10	1	14	3	I/O	TTL	
RD2/AD10	8	9	64	9	78	I/O	TTL	
RD3/AD11	7	8	63	8	77	I/O	TTL	
RD4/AD12	6	7	62	7	76	I/O	TTL	
RD5/AD13	5	6	61	6	75	I/O	TTL	
RD6/AD14	4	5	60	5	74	I/O	TTL	
RD7/AD15	3	4	59	4	73	I/O	TTL	
								PORTE is a bi-directional I/O Port.
RE0/ALE	11	12	3	16	5	I/O	TTL	In Microprocessor mode or Extended Microcontroller mode, RE0 is the Address Latch Enable (ALE) output. Address should be latched on the falling edge of ALE output.
RE1/$\overline{\text{OE}}$	12	13	4	17	6	I/O	TTL	In Microprocessor or Extended Microcontroller mode, RE1 is the Output Enable ($\overline{\text{OE}}$) control output (active low).
RE2/$\overline{\text{WR}}$	13	14	5	18	7	I/O	TTL	In Microprocessor or Extended Microcontroller mode, RE2 is the Write Enable ($\overline{\text{WR}}$) control output (active low).
RE3/CAP4	14	15	6	19	8	I/O	ST	RE3 can also be the Capture4 input pin.
								PORTF is a bi-directional I/O Port.
RF0/AN4	26	28	18	36	24	I/O	ST	RF0 can also be analog input 4.
RF1/AN5	25	27	17	35	23	I/O	ST	RF1 can also be analog input 5.
RF2/AN6	24	26	16	30	18	I/O	ST	RF2 can also be analog input 6.
RF3/AN7	23	25	15	29	17	I/O	ST	RF3 can also be analog input 7.
RF4/AN8	22	24	14	28	16	I/O	ST	RF4 can also be analog input 8.
RF5/AN9	21	23	13	27	15	I/O	ST	RF5 can also be analog input 9.
RF6/AN10	20	22	12	26	14	I/O	ST	RF6 can also be analog input 10.
RF7/AN11	19	21	11	25	13	I/O	ST	RF7 can also be analog input 11.

Legend: I = Input only; O = Output only; I/O = Input/Output;
P = Power; — = Not Used; TTL = TTL input; ST = Schmitt Trigger input

Note 1: The output is only available by the peripheral operation.
2: Open drain input/output pin. Pin forced to input upon any device RESET.

6.3 Peripheral Interrupt Request Register1 (PIR1) and Register2 (PIR2)

These registers contains the individual flag bits for the peripheral interrupts.

Note: These bits will be set by the specified condition, even if the corresponding interrupt enable bit is cleared (interrupt disabled), or the GLINTD bit is set (all interrupts disabled). Before enabling an interrupt, the user may wish to clear the interrupt flag to ensure that the program does not immediately branch to the peripheral Interrupt Service Routine.

REGISTER 6-4: PIR1 REGISTER (ADDRESS: 16h, BANK 1)

R/W-x	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R-0
RBIF	TMR3IF	TMR2IF	TMR1IF	CA2IF	CA1IF	TX1IF	RC1IF
bit 7							bit 0

bit 7 **RBIF**: PORTB Interrupt-on-Change Flag bit
1 = One of the PORTB inputs changed (software must end the mismatch condition)
0 = None of the PORTB inputs have changed

bit 6 **TMR3IF**: TMR3 Interrupt Flag bit

<u>If Capture1 is enabled (CA1/$\overline{\text{PR3}}$ = 1):</u>
1 = TMR3 overflowed
0 = TMR3 did not overflow

<u>If Capture1 is disabled (CA1/$\overline{\text{PR3}}$ = 0):</u>
1 = TMR3 value has rolled over to 0000h from equalling the period register (PR3H:PR3L) value
0 = TMR3 value has not rolled over to 0000h from equalling the period register (PR3H:PR3L) value

bit 5 **TMR2IF**: TMR2 Interrupt Flag bit
1 = TMR2 value has rolled over to 0000h from equalling the period register (PR2) value
0 = TMR2 value has not rolled over to 0000h from equalling the period register (PR2) value

bit 4 **TMR1IF**: TMR1 Interrupt Flag bit

<u>If TMR1 is in 8-bit mode (T16 = 0):</u>
1 = TMR1 value has rolled over to 0000h from equalling the period register (PR1) value
0 = TMR1 value has not rolled over to 0000h from equalling the period register (PR1) value

<u>If Timer1 is in 16-bit mode (T16 = 1):</u>
1 = TMR2:TMR1 value has rolled over to 0000h from equalling the period register (PR2:PR1) value
0 = TMR2:TMR1 value has not rolled over to 0000h from equalling the period register (PR2:PR1) value

bit 3 **CA2IF**: Capture2 Interrupt Flag bit
1 = Capture event occurred on RB1/CAP2 pin
0 = Capture event did not occur on RB1/CAP2 pin

bit 2 **CA1IF**: Capture1 Interrupt Flag bit
1 = Capture event occurred on RB0/CAP1 pin
0 = Capture event did not occur on RB0/CAP1 pin

bit 1 **TX1IF**: USART1 Transmit Interrupt Flag bit (state controlled by hardware)
1 = USART1 Transmit buffer is empty
0 = USART1 Transmit buffer is full

bit 0 **RC1IF**: USART1 Receive Interrupt Flag bit (state controlled by hardware)
1 = USART1 Receive buffer is full
0 = USART1 Receive buffer is empty

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

7.7 Program Counter Module

The Program Counter (PC) is a 16-bit register. PCL, the low byte of the PC, is mapped in the data memory. PCL is readable and writable just as is any other register. PCH is the high byte of the PC and is not directly addressable. Since PCH is not mapped in data or program memory, an 8-bit register PCLATH (PC high latch) is used as a holding latch for the high byte of the PC. PCLATH is mapped into data memory. The user can read or write PCH through PCLATH.

The 16-bit wide PC is incremented after each instruction fetch during Q1 unless:

- Modified by a `GOTO`, `CALL`, `LCALL`, `RETURN`, `RETLW`, or `RETFIE` instruction
- Modified by an interrupt response
- Due to destination write to PCL by an instruction

“Skips” are equivalent to a forced `NOP` cycle at the skipped address.

Figure 7-7 and Figure 7-8 show the operation of the program counter for various situations.

FIGURE 7-7: PROGRAM COUNTER OPERATION

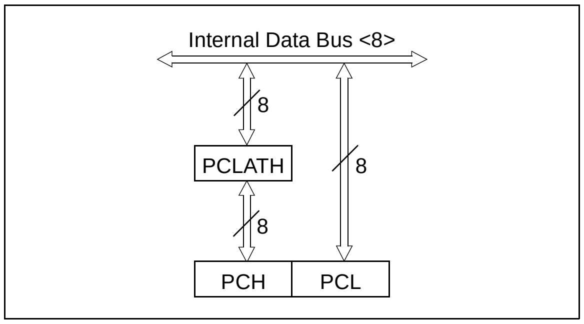
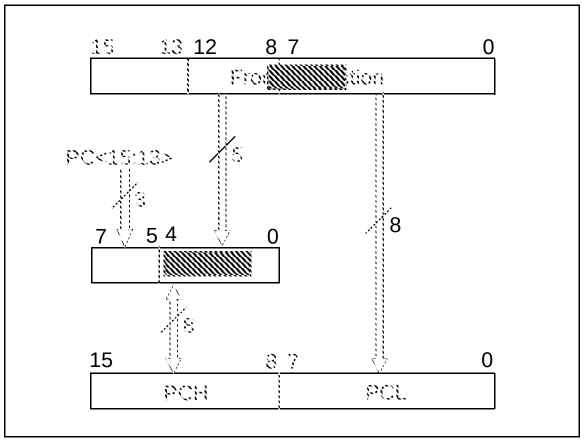


FIGURE 7-8: PROGRAM COUNTER USING THE `CALL` AND `GOTO` INSTRUCTIONS



Using Figure 7-7, the operations of the PC and PCLATH for different instructions are as follows:

a) <u>`LCALL` instructions</u>:
   An 8-bit destination address is provided in the instruction (opcode). PCLATH is unchanged.
   PCLATH → PCH
   Opcode<7:0> → PCL

b) <u>Read instructions on PCL</u>:
   Any instruction that reads PCL.
   PCL → data bus → ALU or destination
   PCH → PCLATH

c) <u>Write instructions on PCL</u>:
   Any instruction that writes to PCL.
   8-bit data → data bus → PCL
   PCLATH → PCH

d) <u>Read-Modify-Write instructions on PCL:</u>
   Any instruction that does a read-write-modify operation on PCL, such as `ADDWF PCL`.
   Read: PCL → data bus → ALU
   Write: 8-bit result → data bus → PCL
   PCLATH → PCH

e) <u>`RETURN` instruction:</u>
   Stack<MRU> → PC<15:0>

Using Figure 7-8, the operation of the PC and PCLATH for `GOTO` and `CALL` instructions is as follows:

<u>`CALL, GOTO` instructions</u>:
A 13-bit destination address is provided in the instruction (opcode).
Opcode<12:0> → PC<12:0>
PC<15:13> → PCLATH<7:5>
Opcode<12:8> → PCLATH<4:0>

The read-modify-write only affects the PCL with the result. PCH is loaded with the value in the PCLATH. For example, `ADDWF PCL` will result in a jump within the current page. If PC = 03F0h, WREG = 30h and PCLATH = 03h before instruction, PC = 0320h after the instruction. To accomplish a true 16-bit computed jump, the user needs to compute the 16-bit destination address, write the high byte to PCLATH and then write the low value to PCL.

The following PC related operations do not change PCLATH:

a) `LCALL`, `RETLW`, and `RETFIE` instructions.
b) Interrupt vector is forced onto the PC.
c) Read-modify-write instructions on PCL (e.g. `BSF PCL`).

TABLE 10-7: PORTD FUNCTIONS

Name	Bit	Buffer Type	Function
RD0/AD8	bit0	TTL	Input/output or system bus address/data pin.
RD1/AD9	bit1	TTL	Input/output or system bus address/data pin.
RD2/AD10	bit2	TTL	Input/output or system bus address/data pin.
RD3/AD11	bit3	TTL	Input/output or system bus address/data pin.
RD4/AD12	bit4	TTL	Input/output or system bus address/data pin.
RD5/AD13	bit5	TTL	Input/output or system bus address/data pin.
RD6/AD14	bit6	TTL	Input/output or system bus address/data pin.
RD7/AD15	bit7	TTL	Input/output or system bus address/data pin.

Legend: TTL = TTL input

TABLE 10-8: REGISTERS/BITS ASSOCIATED WITH PORTD

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	$\overline{\text{MCLR}}$, WDT
13h, Bank 1	PORTD	RD7/ AD15	RD6/ AD14	RD5/ AD13	RD4/ AD12	RD3/ AD11	RD2/ AD10	RD1/ AD9	RD0/ AD8	xxxx xxxx	uuuu uuuu
12h, Bank 1	DDRD	Data Direction Register for PORTD								1111 1111	1111 1111

Legend: x = unknown, u = unchanged

13.0 TIMER1, TIMER2, TIMER3, PWMS AND CAPTURES

The PIC17C7XX has a wealth of timers and time based functions to ease the implementation of control applications. These time base functions include three PWM outputs and four Capture inputs.

Timer1 and Timer2 are two 8-bit incrementing timers, each with an 8-bit period register (PR1 and PR2, respectively) and separate overflow interrupt flags. Timer1 and Timer2 can operate either as timers (increment on internal FOSC/4 clock), or as counters (increment on falling edge of external clock on pin RB4/TCLK12). They are also software configurable to operate as a single 16-bit timer/counter. These timers are also used as the time base for the PWM (Pulse Width Modulation) modules.

Timer3 is a 16-bit timer/counter which uses the TMR3H and TMR3L registers. Timer3 also has two additional registers (PR3H/CA1H:PR3L/CA1L) that are configurable as a 16-bit period register or a 16-bit capture register. TMR3 can be software configured to increment from the internal system clock (FOSC/4), or from an external signal on the RB5/TCLK3 pin. Timer3 is the time base for all of the 16-bit captures.

Six other registers comprise the Capture2, Capture3, and Capture4 registers (CA2H:CA2L, CA3H:CA3L, and CA4H:CA4L).

Figure 13-1, Figure 13-2 and Figure 13-3 are the control registers for the operation of Timer1, Timer2 and Timer3, as well as PWM1, PWM2, PWM3, Capture1, Capture2, Capture3 and Capture4.

Table 13-1 shows the Timer resource requirements for these time base functions. Each timer is an open resource so that multiple functions may operate with it.

TABLE 13-1: TIME-BASE FUNCTION/ RESOURCE REQUIREMENTS

Time Base Function	Timer Resource
PWM1	Timer1
PWM2	Timer1 or Timer2
PWM3	Timer1 or Timer2
Capture1	Timer3
Capture2	Timer3
Capture3	Timer3
Capture4	Timer3

REGISTER 13-1: TCON1 REGISTER (ADDRESS: 16h, BANK 3)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CA2ED1	CA2ED0	CA1ED1	CA1ED0	T16	TMR3CS	TMR2CS	TMR1CS
bit 7							bit 0

bit 7-6 **CA2ED1:CA2ED0**: Capture2 Mode Select bits
`00` = Capture on every falling edge
`01` = Capture on every rising edge
`10` = Capture on every 4th rising edge
`11` = Capture on every 16th rising edge

bit 5-4 **CA1ED1:CA1ED0**: Capture1 Mode Select bits
`00` = Capture on every falling edge
`01` = Capture on every rising edge
`10` = Capture on every 4th rising edge
`11` = Capture on every 16th rising edge

bit 3 **T16**: Timer2:Timer1 Mode Select bit
`1` = Timer2 and Timer1 form a 16-bit timer
`0` = Timer2 and Timer1 are two 8-bit timers

bit 2 **TMR3CS**: Timer3 Clock Source Select bit
`1` = TMR3 increments off the falling edge of the RB5/TCLK3 pin
`0` = TMR3 increments off the internal clock

bit 1 **TMR2CS**: Timer2 Clock Source Select bit
`1` = TMR2 increments off the falling edge of the RB4/TCLK12 pin
`0` = TMR2 increments off the internal clock

bit 0 **TMR1CS**: Timer1 Clock Source Select bit
`1` = TMR1 increments off the falling edge of the RB4/TCLK12 pin
`0` = TMR1 increments off the internal clock

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

REGISTER 13-3: TCON3 REGISTER (ADDRESS: 16h, BANK 7)

U-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	CA4OVF	CA3OVF	CA4ED1	CA4ED0	CA3ED1	CA3ED0	PWM3ON
bit 7							bit 0

bit 7 **Unimplemented:** Read as ‘0’

bit 6 **CA4OVF**: Capture4 Overflow Status bit
This bit indicates that the capture value had not been read from the capture register pair (CA4H:CA4L) before the next capture event occurred. The capture register retains the oldest unread capture value (last capture before overflow). Subsequent capture events will not update the capture register with the TMR3 value until the capture register has been read (both bytes).
`1` = Overflow occurred on Capture4 registers
`0` = No overflow occurred on Capture4 registers

bit 5 **CA3OVF**: Capture3 Overflow Status bit
This bit indicates that the capture value had not been read from the capture register pair (CA3H:CA3L) before the next capture event occurred. The capture register retains the oldest unread capture value (last capture before overflow). Subsequent capture events will not update the capture register with the TMR3 value until the capture register has been read (both bytes).
`1` = Overflow occurred on Capture3 registers
`0` = No overflow occurred on Capture3 registers

bit 4-3 **CA4ED1:CA4ED0**: Capture4 Mode Select bits
`00` = Capture on every falling edge
`01` = Capture on every rising edge
`10` = Capture on every 4th rising edge
`11` = Capture on every 16th rising edge

bit 2-1 **CA3ED1:CA3ED0**: Capture3 Mode Select bits
`00` = Capture on every falling edge
`01` = Capture on every rising edge
`10` = Capture on every 4th rising edge
`11` = Capture on every 16th rising edge

bit 0 **PWM3ON**: PWM3 On bit
`1` = PWM3 is enabled (the RG5/PWM3 pin ignores the state of the DDRG<5> bit)
`0` = PWM3 is disabled (the RG5/PWM3 pin uses the state of the DDRG<5> bit for data direction)

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

13.1.3 USING PULSE WIDTH MODULATION (PWM) OUTPUTS WITH TIMER1 AND TIMER2

Three high speed pulse width modulation (PWM) outputs are provided. The PWM1 output uses Timer1 as its time base, while PWM2 and PWM3 may independently be software configured to use either Timer1 or Timer2 as the time base. The PWM outputs are on the RB2/PWM1, RB3/PWM2 and RG5/PWM3 pins.

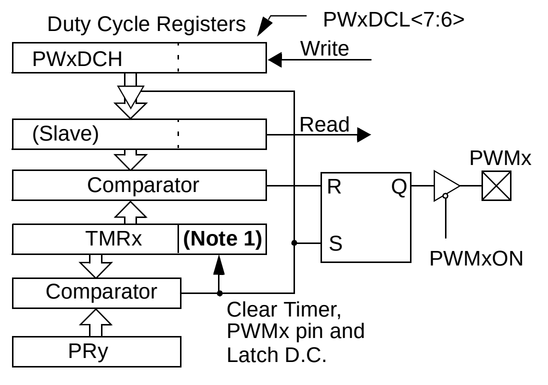
Each PWM output has a maximum resolution of 10-bits. At 10-bit resolution, the PWM output frequency is 32.2 kHz (@ 32 MHz clock) and at 8-bit resolution the PWM output frequency is 128.9 kHz. The duty cycle of the output can vary from 0% to 100%.

Figure 13-3 shows a simplified block diagram of a PWM module.

The duty cycle registers are double buffered for glitch free operation. Figure 13-4 shows how a glitch could occur if the duty cycle registers were not double buffered.

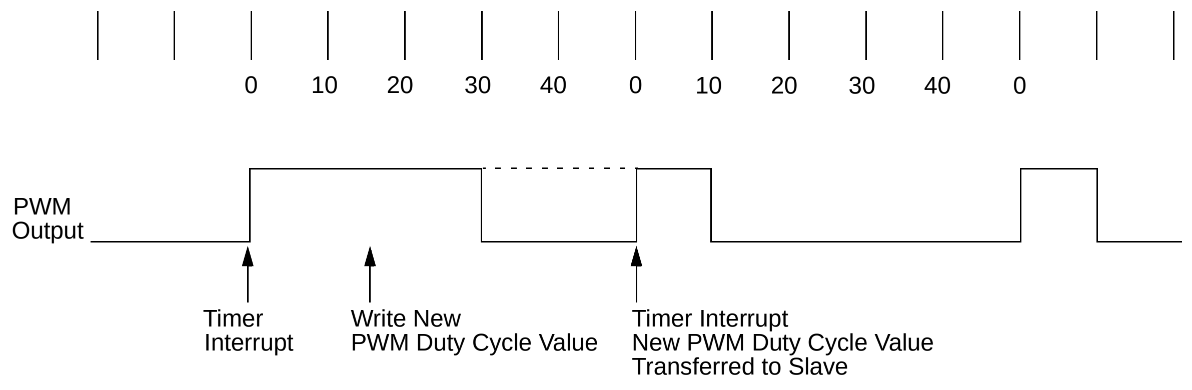
The user needs to set the PWM1ON bit (TCON2<4>) to enable the PWM1 output. When the PWM1ON bit is set, the RB2/PWM1 pin is configured as PWM1 output and forced as an output, irrespective of the data direction bit (DDRB<2>). When the PWM1ON bit is clear, the pin behaves as a port pin and its direction is controlled by its data direction bit (DDRB<2>). Similarly, the PWM2ON (TCON2<5>) bit controls the configuration of the RB3/PWM2 pin and the PWM3ON (TCON3<0>) bit controls the configuration of the RG5/PWM3 pin.

FIGURE 13-3: SIMPLIFIED PWM BLOCK DIAGRAM



Note 1: 8-bit timer is concatenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit time base.

FIGURE 13-4: PWM OUTPUT (NOT BUFFERED)



0
10
20
30
40
0
10
20
30
40
0
PWM Output
Timer Interrupt
Write New PWM Duty Cycle Value
Timer Interrupt New PWM Duty Cycle Value Transferred to Slave

Note: The dotted line shows PWM output if duty cycle registers were not double buffered. If the new duty cycle is written after the timer has passed that value, then the PWM does not reset at all during the current cycle, causing a “glitch”.

In this example, PWM period = 50. Old duty cycle is 30. New duty cycle value is 10.

13.1.3.1 PWM Periods

The period of the PWM1 output is determined by Timer1 and its period register (PR1). The period of the PWM2 and PWM3 outputs can be individually software configured to use either Timer1 or Timer2 as the time-base. For PWM2, when TM2PW2 bit (PW2DCL<5>) is clear, the time base is determined by TMR1 and PR1 and when TM2PW2 is set, the time base is determined by Timer2 and PR2. For PWM3, when TM2PW3 bit (PW3DCL<5>) is clear, the time base is determined by TMR1 and PR1, and when TM2PW3 is set, the time base is determined by Timer2 and PR2.

Running two different PWM outputs on two different timers allows different PWM periods. Running all PWMs from Timer1 allows the best use of resources by freeing Timer2 to operate as an 8-bit timer. Timer1 and Timer2 cannot be used as a 16-bit timer if any PWM is being used.

The PWM periods can be calculated as follows:

$$\text{period of PWM1} = [(\text{PR1}) + 1] \times 4T_{OSC}$$

$$\text{period of PWM2} = [(\text{PR1}) + 1] \times 4T_{OSC} \quad \text{or} \quad [(\text{PR2}) + 1] \times 4T_{OSC}$$

$$\text{period of PWM3} = [(\text{PR1}) + 1] \times 4T_{OSC} \quad \text{or} \quad [(\text{PR2}) + 1] \times 4T_{OSC}$$

The duty cycle of PWMx is determined by the 10-bit value DCx<9:0>. The upper 8-bits are from register PWxDCH and the lower 2-bits are from PWxDCL<7:6> (PWxDCH:PWxDCL<7:6>). Table 13-4 shows the maximum PWM frequency (F_{PWM}), given the value in the period register.

The number of bits of resolution that the PWM can achieve depends on the operation frequency of the device as well as the PWM frequency (F_{PWM}).

Maximum PWM resolution (bits) for a given PWM frequency:

$$= \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log(2)} \text{ bits}$$

where: F_{PWM} = 1 / period of PWM

The PWMx duty cycle is as follows:

$$\text{PWMx Duty Cycle} = (\text{DCx}) \times T_{OSC}$$

where DCx represents the 10-bit value from PWxDCH:PWxDCL.

If DCx = 0, then the duty cycle is zero. If PRx = PWxDCH, then the PWM output will be low for one to four Q-clocks (depending on the state of the PWxDCL<7:6> bits). For a duty cycle to be 100%, the PWxDCH value must be greater then the PRx value.

The duty cycle registers for both PWM outputs are double buffered. When the user writes to these registers, they are stored in master latches. When TMR1 (or TMR2) overflows and a new PWM period begins, the master latch values are transferred to the slave latches and the PWMx pin is forced high.

Note: For PW1DCH, PW1DCL, PW2DCH, PW2DCL, PW3DCH and PW3DCL registers, a write operation writes to the "master latches", while a read operation reads the "slave latches". As a result, the user may not read back what was just written to the duty cycle registers (until transferred to slave latch).

The user should also avoid any "read-modify-write" operations on the duty cycle registers, such as: `ADDWF PW1DCH`. This may cause duty cycle outputs that are unpredictable.

TABLE 13-4: PWM FREQUENCY vs. RESOLUTION AT 33 MHz

PWM Frequency	Frequency (kHz)				
	32.2	**64.5**	**90.66**	**128.9**	**515.6**
PRx Value	0xFF	0x7F	0x5A	0x3F	0x0F
High Resolution	10-bit	9-bit	8.5-bit	8-bit	6-bit
Standard Resolution	8-bit	7-bit	6.5-bit	6-bit	4-bit

13.1.3.2 PWM INTERRUPTS

The PWM modules make use of the TMR1 and/or TMR2 interrupts. A timer interrupt is generated when TMR1 or TMR2 equals its period register and on the following increment is cleared to zero. This interrupt also marks the beginning of a PWM cycle. The user can write new duty cycle values before the timer rollover. The TMR1 interrupt is latched into the TMR1IF bit and the TMR2 interrupt is latched into the TMR2IF bit. These flags must be cleared in software.

TABLE 14-4: BAUD RATES FOR SYNCHRONOUS MODE

BAUD RATE (K)	FOSC = 33 MHz KBAUD	%ERROR	SPBRG VALUE (DECIMAL)	FOSC = 25 MHz KBAUD	%ERROR	SPBRG VALUE (DECIMAL)	FOSC = 20 MHz KBAUD	%ERROR	SPBRG VALUE (DECIMAL)	FOSC = 16 MHz KBAUD	%ERROR	SPBRG VALUE (DECIMAL)
0.3	NA	—	—	NA	—	—	NA	—	—	NA	—	—
1.2	NA	—	—	NA	—	—	NA	—	—	NA	—	—
2.4	NA	—	—	NA	—	—	NA	—	—	NA	—	—
9.6	NA	—	—	NA	—	—	NA	—	—	NA	—	—
19.2	NA	—	—	NA	—	—	19.53	+1.73	255	19.23	+0.16	207
76.8	77.10	+0.39	106	77.16	+0.47	80	76.92	+0.16	64	76.92	+0.16	51
96	95.93	-0.07	85	96.15	+0.16	64	96.15	+0.16	51	95.24	-0.79	41
300	294.64	-1.79	27	297.62	-0.79	20	294.1	-1.96	16	307.69	+2.56	12
500	485.29	-2.94	16	480.77	-3.85	12	500	0	9	500	0	7
HIGH	8250	—	0	6250	—	0	5000	—	0	4000	—	0
LOW	32.22	—	255	24.41	—	255	19.53	—	255	15.625	—	255

BAUD RATE (K)	FOSC = 10 MHz KBAUD	%ERROR	SPBRG VALUE (DECIMAL)	FOSC = 7.159 MHz KBAUD	%ERROR	SPBRG VALUE (DECIMAL)	FOSC = 5.068 MHz KBAUD	%ERROR	SPBRG VALUE (DECIMAL)
0.3	NA	—	—	NA	—	—	NA	—	—
1.2	NA	—	—	NA	—	—	NA	—	—
2.4	NA	—	—	NA	—	—	NA	—	—
9.6	9.766	+1.73	255	9.622	+0.23	185	9.6	0	131
19.2	19.23	+0.16	129	19.24	+0.23	92	19.2	0	65
76.8	75.76	-1.36	32	77.82	+1.32	22	79.2	+3.13	15
96	96.15	+0.16	25	94.20	-1.88	18	97.48	+1.54	12
300	312.5	+4.17	7	298.3	-0.57	5	316.8	+5.60	3
500	500	0	4	NA	—	—	NA	—	—
HIGH	2500	—	0	1789.8	—	0	1267	—	0
LOW	9.766	—	255	6.991	—	255	4.950	—	255

BAUD RATE (K)	FOSC = 3.579 MHz KBAUD	%ERROR	SPBRG VALUE (DECIMAL)	FOSC = 1 MHz KBAUD	%ERROR	SPBRG VALUE (DECIMAL)	FOSC = 32.768 kHz KBAUD	%ERROR	SPBRG VALUE (DECIMAL)
0.3	NA	—	—	NA	—	—	0.303	+1.14	26
1.2	NA	—	—	1.202	+0.16	207	1.170	-2.48	6
2.4	NA	—	—	2.404	+0.16	103	NA	—	—
9.6	9.622	+0.23	92	9.615	+0.16	25	NA	—	—
19.2	19.04	-0.83	46	19.24	+0.16	12	NA	—	—
76.8	74.57	-2.90	11	83.34	+8.51	2	NA	—	—
96	99.43	_3.57	8	NA	—	—	NA	—	—
300	298.3	-0.57	2	NA	—	—	NA	—	—
500	NA	—	—	NA	—	—	NA	—	—
HIGH	894.9	—	0	250	—	0	8.192	—	0
LOW	3.496	—	255	0.976	—	255	0.032	—	255

14.2 USART Asynchronous Mode

In this mode, the USART uses standard nonreturn-to-zero (NRZ) format (one START bit, eight or nine data bits, and one STOP bit). The most common data format is 8-bits. An on-chip dedicated 8-bit baud rate generator can be used to derive standard baud rate frequencies from the oscillator. The USART's transmitter and receiver are functionally independent but use the same data format and baud rate. The baud rate generator produces a clock x64 of the bit shift rate. Parity is not supported by the hardware, but can be implemented in software (and stored as the ninth data bit). Asynchronous mode is stopped during SLEEP.

The Asynchronous mode is selected by clearing the SYNC bit (TXSTA<4>).

The USART Asynchronous module consists of the following components:

- Baud Rate Generator
- Sampling Circuit
- Asynchronous Transmitter
- Asynchronous Receiver

14.2.1 USART ASYNCHRONOUS TRANSMITTER

The USART transmitter block diagram is shown in Figure 14-1. The heart of the transmitter is the transmit shift register (TSR). The shift register obtains its data from the read/write transmit buffer (TXREG). TXREG is loaded with data in software. The TSR is not loaded until the STOP bit has been transmitted from the previous load. As soon as the STOP bit is transmitted, the TSR is loaded with new data from the TXREG (if available). Once TXREG transfers the data to the TSR (occurs in one TCY at the end of the current BRG cycle), the TXREG is empty and an interrupt bit, TXIF, is set. This interrupt can be enabled/disabled by setting/clearing the TXIE bit. TXIF will be set, regardless of TXIE and cannot be reset in software. It will reset only when new data is loaded into TXREG. While TXIF indicates the status of the TXREG, the TRMT (TXSTA<1>) bit shows the status of the TSR. TRMT is a read only bit which is set when the TSR is empty. No interrupt logic is tied to this bit, so the user has to poll this bit in order to determine if the TSR is empty.

Note:	The TSR is not mapped in data memory, so it is not available to the user.

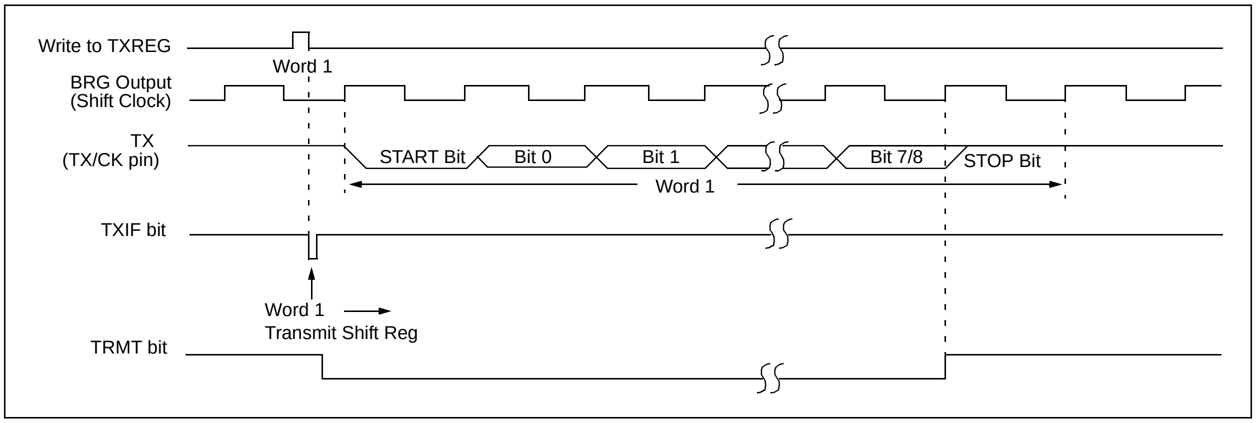
Transmission is enabled by setting the TXEN (TXSTA<5>) bit. The actual transmission will not occur until TXREG has been loaded with data and the baud rate generator (BRG) has produced a shift clock (Figure 14-3). The transmission can also be started by first loading TXREG and then setting TXEN. Normally, when transmission is first started, the TSR is empty, so a transfer to TXREG will result in an immediate transfer to TSR, resulting in an empty TXREG. A back-to-back transfer is thus possible (Figure 14-4). Clearing TXEN during a transmission will cause the transmission to be aborted. This will reset the transmitter and the TX/CK pin will revert to hi-impedance.

In order to select 9-bit transmission, the TX9 (TXSTA<6>) bit should be set and the ninth bit value should be written to TX9D (TXSTA<0>). The ninth bit value must be written before writing the 8-bit data to the TXREG. This is because a data write to TXREG can result in an immediate transfer of the data to the TSR (if the TSR is empty).

Steps to follow when setting up an Asynchronous Transmission:

1. Initialize the SPBRG register for the appropriate baud rate.
2. Enable the asynchronous serial port by clearing the SYNC bit and setting the SPEN bit.
3. If interrupts are desired, then set the TXIE bit.
4. If 9-bit transmission is desired, then set the TX9 bit.
5. If 9-bit transmission is selected, the ninth bit should be loaded in TX9D.
6. Load data to the TXREG register.
7. Enable the transmission by setting TXEN (starts transmission).

FIGURE 14-3: ASYNCHRONOUS MASTER TRANSMISSION



14.2.2 USART ASYNCHRONOUS RECEIVER

The receiver block diagram is shown in Figure 14-2. The data comes in the RX/DT pin and drives the data recovery block. The data recovery block is actually a high speed shifter operating at 16 times the baud rate, whereas the main receive serial shifter operates at the bit rate or at FOSC.

Once Asynchronous mode is selected, reception is enabled by setting bit CREN (RCSTA<4>).

The heart of the receiver is the receive (serial) shift register (RSR). After sampling the STOP bit, the received data in the RSR is transferred to the RCREG (if it is empty). If the transfer is complete, the interrupt bit, RCIF, is set. The actual interrupt can be enabled/disabled by setting/clearing the RCIE bit. RCIF is a read only bit which is cleared by the hardware. It is cleared when RCREG has been read and is empty. RCREG is a double buffered register (i.e., it is a two-deep FIFO). It is possible for two bytes of data to be received and transferred to the RCREG FIFO and a third byte begin shifting to the RSR. On detection of the STOP bit of the third byte, if the RCREG is still full, then the overrun error bit, OERR (RCSTA<1>) will be set. The word in the RSR will be lost. RCREG can be read twice to retrieve the two bytes in the FIFO. The OERR bit has to be cleared in software which is done by resetting the receive logic (CREN is set). If the OERR bit is set, transfers from the RSR to RCREG are inhibited, so it is essential to clear the OERR bit if it is set. The framing error bit FERR (RCSTA<2>) is set if a STOP bit is not detected.

Note: The FERR and the 9th receive bit are buffered the same way as the receive data. Reading the RCREG register will allow the RX9D and FERR bits to be loaded with values for the next received data. Therefore, it is essential for the user to read the RCSTA register before reading RCREG, in order not to lose the old FERR and RX9D information.

14.2.3 SAMPLING

The data on the RX/DT pin is sampled three times by a majority detect circuit to determine if a high or a low level is present at the RX/DT pin. The sampling is done on the seventh, eighth and ninth falling edges of a x16 clock (Figure 14-5).

The x16 clock is a free running clock and the three sample points occur at a frequency of every 16 falling edges.

FIGURE 14-5: RX PIN SAMPLING SCHEME

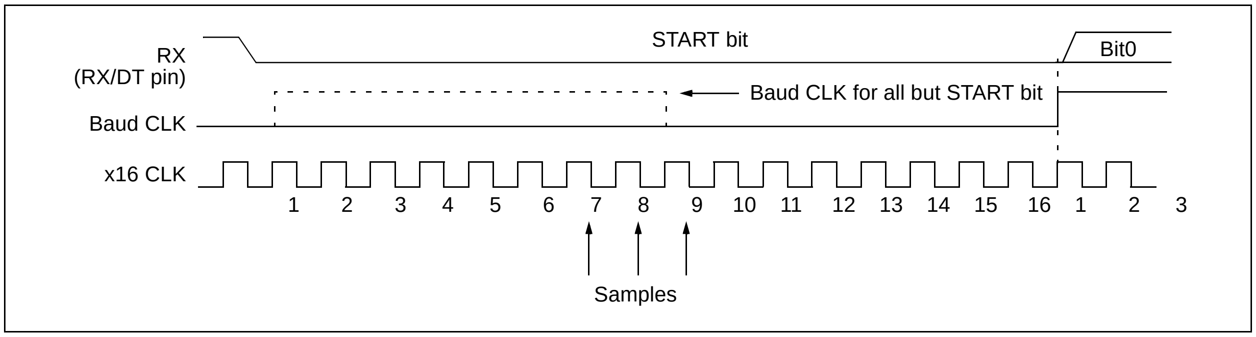
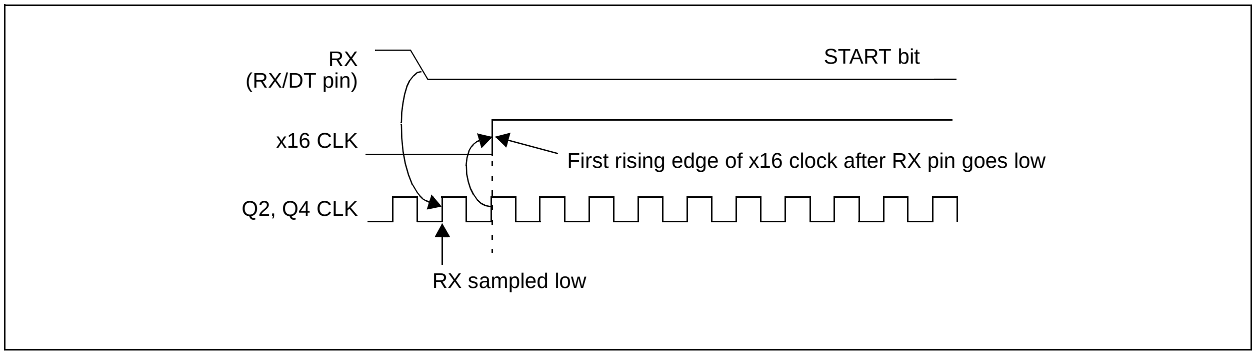


FIGURE 14-6: START BIT DETECT



15.2.11 I^{2}C MASTER MODE TRANSMISSION

Transmission of a data byte, a 7-bit address, or either half of a 10-bit address, is accomplished by simply writing a value to SSPBUF register. This action will set the buffer full flag (BF) and allow the baud rate generator to begin counting and start the next transmission. Each bit of address/data will be shifted out onto the SDA pin after the falling edge of SCL is asserted (see data hold time spec). SCL is held low for one baud rate generator roll over count (TBRG). Data should be valid before SCL is released high (see Data setup time spec). When the SCL pin is released high, it is held that way for TBRG, the data on the SDA pin must remain stable for that duration and some hold time after the next falling edge of SCL. After the eighth bit is shifted out (the falling edge of the eighth clock), the BF flag is cleared and the master releases SDA, allowing the slave device being addressed to respond with an $\overline{\text{ACK}}$ bit during the ninth bit time, if an address match occurs or if data was received properly. The status of $\overline{\text{ACK}}$ is read into the ACKDT on the falling edge of the ninth clock. If the master receives an acknowledge, the acknowledge status bit (AKSTAT) is cleared. If not, the bit is set. After the ninth clock, the SSPIF is set and the master clock (baud rate generator) is suspended until the next data byte is loaded into the SSPBUF, leaving SCL low and SDA unchanged (Figure 15-26).

After the write to the SSPBUF, each bit of address will be shifted out on the falling edge of SCL until all seven address bits and the R/$\overline{\text{W}}$ bit are completed. On the falling edge of the eighth clock, the master will de-assert the SDA pin, allowing the slave to respond with an acknowledge. On the falling edge of the ninth clock, the master will sample the SDA pin to see if the address was recognized by a slave. The status of the ACK bit is loaded into the ACKSTAT status bit (SSPCON2<6>). Following the falling edge of the ninth clock transmission of the address, the SSPIF is set, the BF flag is cleared and the baud rate generator is turned off until another write to the SSPBUF takes place, holding SCL low and allowing SDA to float.

15.2.11.1 BF Status Flag

In Transmit mode, the BF bit (SSPSTAT<0>) is set when the CPU writes to SSPBUF and is cleared when all 8 bits are shifted out.

15.2.11.2 WCOL Status Flag

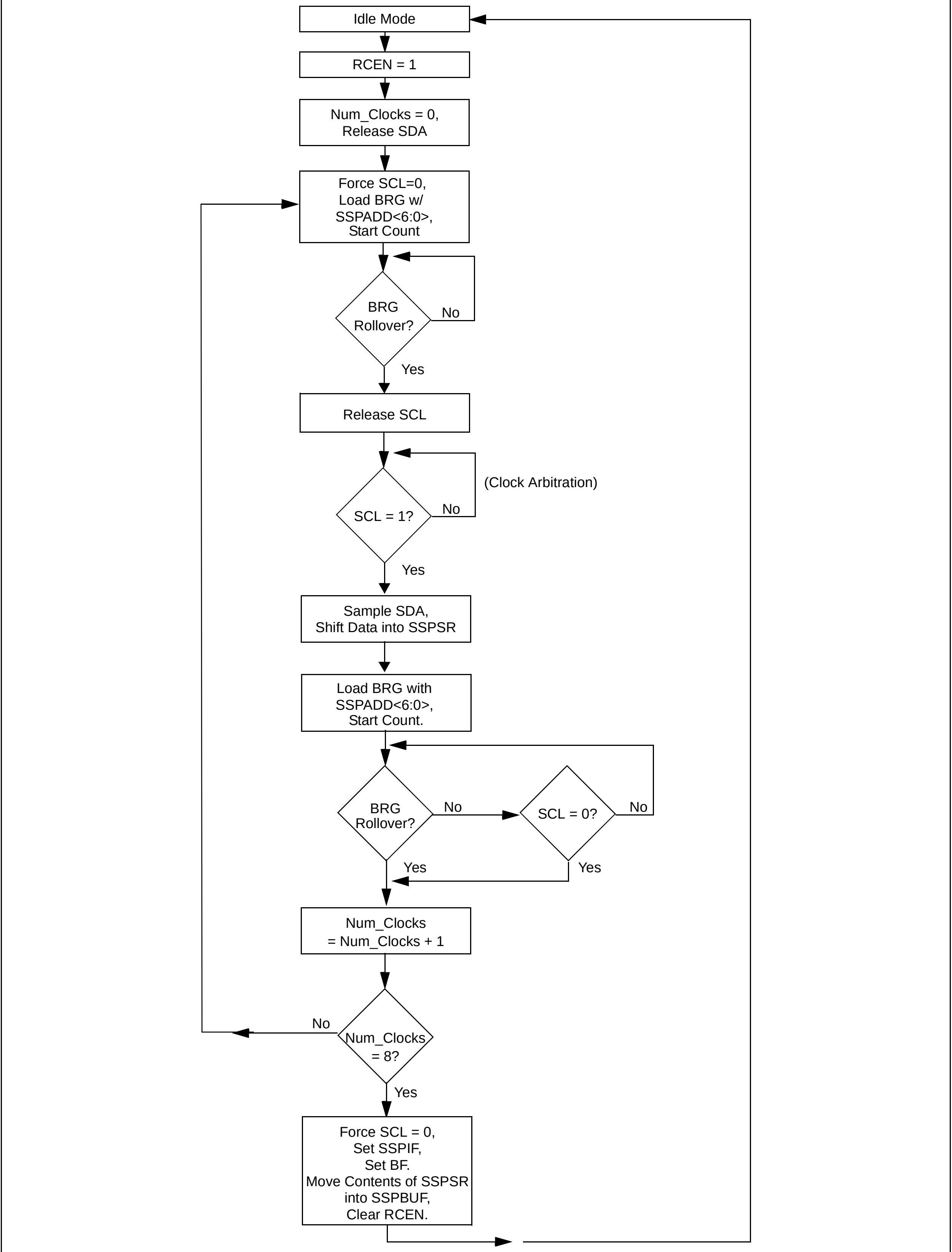
If the user writes the SSPBUF when a transmit is already in progress (i.e., SSPSR is still shifting out a data byte), then WCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

WCOL must be cleared in software.

15.2.11.3 AKSTAT Status Flag

In Transmit mode, the AKSTAT bit (SSPCON2<6>) is cleared when the slave has sent an acknowledge ($\overline{\text{ACK}}$ = 0) and is set when the slave does not acknowledge ($\overline{\text{ACK}}$ = 1). A slave sends an acknowledge when it has recognized its address (including a general call), or when the slave has properly received its data.

FIGURE 15-27: MASTER RECEIVER FLOW CHART



15.2.18.2 Bus Collision During a Repeated Start Condition

During a Repeated Start condition, a bus collision occurs if:

a) A low level is sampled on SDA when SCL goes from low level to high level.

b) SCL goes low before SDA is asserted low, indicating that another master is attempting to transmit a data '1'.

When the user de-asserts SDA and the pin is allowed to float high, the BRG is loaded with SSPADD<6:0> and counts down to '0'. The SCL pin is then de-asserted and when sampled high, the SDA pin is sampled. If SDA is low, a bus collision has occurred (i.e., another master is attempting to transmit a data '0'). If, however, SDA is sampled high, then the BRG is reloaded and begins counting. If SDA goes from high to low before the BRG times out, no bus collision occurs because no two masters can assert SDA at exactly the same time.

If, however, SCL goes from high to low before the BRG times out and SDA has not already been asserted, then a bus collision occurs. In this case, another master is attempting to transmit a data '1' during the Repeated Start condition.

If, at the end of the BRG time-out, both SCL and SDA are still high, the SDA pin is driven low, the BRG is reloaded and begins counting. At the end of the count, regardless of the status of the SCL pin, the SCL pin is driven low and the Repeated Start condition is complete (Figure 15-38).

FIGURE 15-38: BUS COLLISION DURING A REPEATED START CONDITION (CASE 1)

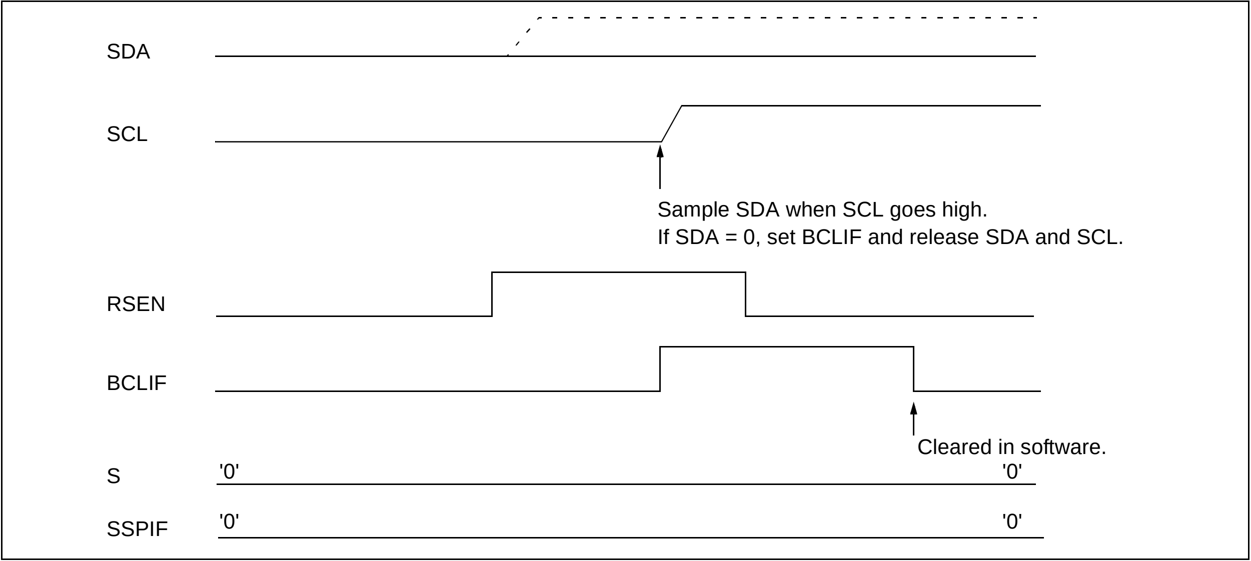
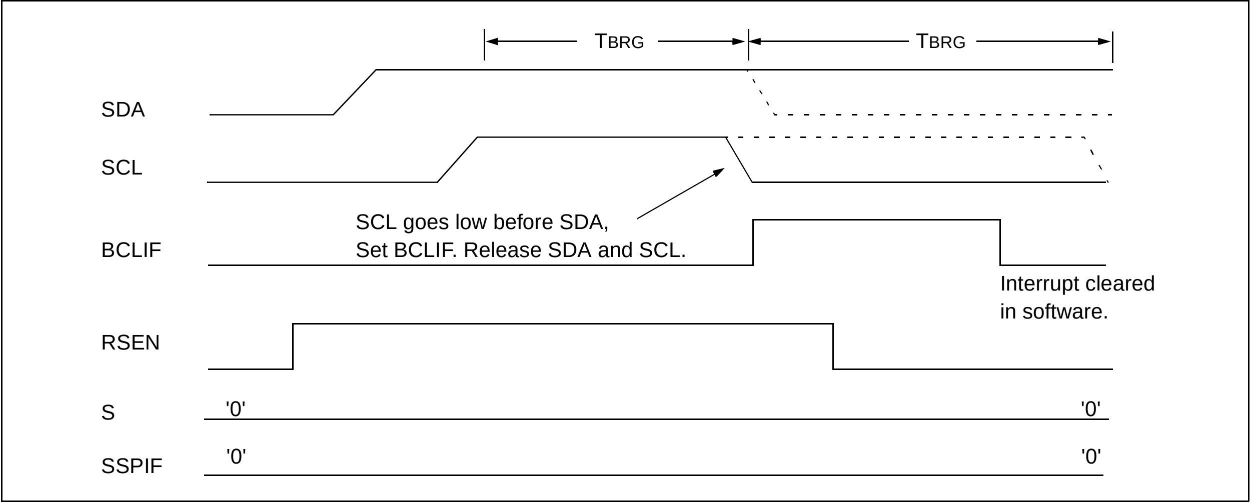

SDA
SCL
Sample SDA when SCL goes high.
If SDA = 0, set BCLIF and release SDA and SCL.
RSEN
BCLIF
Cleared in software.
S
'0'
'0'
SSPIF
'0'
'0'

FIGURE 15-39: BUS COLLISION DURING REPEATED START CONDITION (CASE 2)



EXAMPLE 15-2: INTERFACING TO A 24LC01B SERIAL EEPROM (USING MPLAB C17)

```
void ACKPoll(void)
{
        StartI2C();                     // Send start bit
        IdleI2C();                      // Wait for idle condition
        WriteI2C(CONTROL);              // Send control byte
        IdleI2C();                      // Wait for idle condition
        // Poll the ACK bit coming from the 24LC01B
        // Loop as long as the 24LC01B NACKs
        while (SSPCON2bits.ACKSTAT)
        {
                RestartI2C();           // Send a restart bit
                IdleI2C();              // Wait for idle condition
                WriteI2C(CONTROL);      // Send control byte
                IdleI2C();              // Wait for idle condition
        }
        IdleI2C();                      // Wait for idle condition
        StopI2C();                      // Send stop bit
        IdleI2C();                      // Wait for idle condition
        return;
}
```

BSF — Bit Set f

Syntax:	[*label*] BSF f,b
Operands:	0 ≤ f ≤ 255 0 ≤ b ≤ 7
Operation:	1 → (f<b>)
Status Affected:	None
Encoding:	`1000` `0bbb` `ffff` `ffff`
Description:	Bit 'b' in register 'f' is set.
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `BSF FLAG_REG, 7`

Before Instruction
FLAG_REG = 0x0A

After Instruction
FLAG_REG = 0x8A

BTFSC — Bit Test, skip if Clear

Syntax:	[*label*] BTFSC f,b
Operands:	0 ≤ f ≤ 255 0 ≤ b ≤ 7
Operation:	skip if (f<b>) = 0
Status Affected:	None
Encoding:	`1001` `1bbb` `ffff` `ffff`
Description:	If bit 'b' in register 'f' is 0, then the next instruction is skipped. If bit 'b' is 0, then the next instruction fetched during the current instruction execution is discarded and a `NOP` is executed instead, making this a two-cycle instruction.
Words:	1
Cycles:	1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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

Example:

```
HERE    BTFSC   FLAG,1
FALSE   :
TRUE    :
```

Before Instruction
PC = address (`HERE`)

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

CPFSLT — Compare f with WREG, skip if f < WREG

Syntax:	[*label*] CPFSLT f
Operands:	$0 \le f \le 255$
Operation:	(f) – (WREG), skip if (f) < (WREG) (unsigned comparison)
Status Affected:	None
Encoding:	`0011` `0000` `ffff` `ffff`
Description:	Compares the contents of data memory location 'f' to the contents of WREG by performing an unsigned subtraction. If the contents of 'f' are less than the contents of WREG, then the fetched instruction is discarded and a `NOP` is executed instead, making this a two-cycle instruction.
Words:	1
Cycles:	1 (2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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

Example:

```
HERE    CPFSLT REG
NLESS   :
LESS    :
```

Before Instruction

PC	=	Address (`HERE`)
W	=	?

After Instruction

If REG	<	WREG;
PC	=	Address (`LESS`)
If REG	≥	WREG;
PC	=	Address (`NLESS`)

DAW — Decimal Adjust WREG Register

Syntax:	[*label*] DAW f,s
Operands:	$0 \le f \le 255$ $s \in [0,1]$
Operation:	If [[WREG<7:4> > 9].OR.[C = 1]].AND. [WREG<3:0> > 9] then WREG<7:4> + 7→ f<7:4>, s<7:4>; If [WREG<7:4> > 9].OR.[C = 1] then WREG<7:4> + 6→ f<7:4>, s<7:4>; else WREG<7:4>→ f<7:4>, s<7:4>; If [WREG<3:0> > 9].OR.[DC = 1] then WREG<3:0> + 6→ f<3:0>, s<3:0>; else WREG<3:0>→ f<3:0>, s<3:0>
Status Affected:	C
Encoding:	`0010` `111s` `ffff` `ffff`
Description:	DAW adjusts the eight-bit value in WREG, resulting from the earlier addition of two variables (each in packed BCD format) and produces a correct packed BCD result. s = 0: Result is placed in Data memory location 'f' and WREG. s = 1: Result is placed in Data memory location 'f'.
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `DAW   REG1, 0`

Before Instruction

WREG	=	0xA5
REG1	=	??
C	=	0
DC	=	0

After Instruction

WREG	=	0x05
REG1	=	0x05
C	=	1
DC	=	0

TABLRD **Table Read**

Example1: `TABLRD 1, 1, REG ;`

Before Instruction

REG	=	0x53
TBLATH	=	0xAA
TBLATL	=	0x55
TBLPTR	=	0xA356
MEMORY(TBLPTR)	=	0x1234

After Instruction (table write completion)

REG	=	0xAA
TBLATH	=	0x12
TBLATL	=	0x34
TBLPTR	=	0xA357
MEMORY(TBLPTR)	=	0x5678

Example2: `TABLRD 0, 0, REG ;`

Before Instruction

REG	=	0x53
TBLATH	=	0xAA
TBLATL	=	0x55
TBLPTR	=	0xA356
MEMORY(TBLPTR)	=	0x1234

After Instruction (table write completion)

REG	=	0x55
TBLATH	=	0x12
TBLATL	=	0x34
TBLPTR	=	0xA356
MEMORY(TBLPTR)	=	0x1234

TABLWT **Table Write**

Syntax: [*label*] TABLWT t,i,f

Operands: $0 \leq f \leq 255$
$i \in [0,1]$
$t \in [0,1]$

Operation: If t = 0,
f → TBLATL;
If t = 1,
f → TBLATH;
TBLAT → Prog Mem (TBLPTR);
If i = 1,
TBLPTR + 1 → TBLPTR
If i = 0,
TBLPTR is unchanged

Status Affected: None

Encoding:

1010	11ti	ffff	ffff

Description:

1. Load value in 'f' into 16-bit table latch (TBLAT)
   If t = 1: load into high byte;
   If t = 0: load into low byte
2. The contents of TBLAT are written to the program memory location pointed to by TBLPTR.
   If TBLPTR points to external program memory location, then the instruction takes two-cycle.
   If TBLPTR points to an internal EPROM location, then the instruction is terminated when an interrupt is received.

Note: The $\overline{\text{MCLR}}$/VPP pin must be at the programming voltage for successful programming of internal memory.
If $\overline{\text{MCLR}}$/VPP = VDD
the programming sequence of internal memory will be interrupted. A short write will occur (2 TCY). The internal memory location will not be affected.

3. The TBLPTR can be automatically incremented
   If i = 1; TBLPTR is not incremented
   If i = 0; TBLPTR is incremented

Words: 1

Cycles: 2 (many if write is to on-chip EPROM program memory)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register TBLATH or TBLATL
No operation	No operation (Table Pointer on Address bus)	No operation	No operation (Table Latch on Address bus, $\overline{\text{WR}}$ goes low)

TLWT — Table Latch Write

Syntax:	[*label*] TLWT t,f
Operands:	0 ≤ f ≤ 255 t ∈ [0,1]
Operation:	If t = 0, f → TBLATL; If t = 1, f → TBLATH
Status Affected:	None
Encoding:	`1010` `01tx` `ffff` `ffff`
Description:	Data from file register 'f' is written into the 16-bit table latch (TBLAT). If t = 1; high byte is written If t = 0; low byte is written This instruction is used in conjunction with `TABLWT` to transfer data from data memory to program memory.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register TBLATH or TBLATL

Example: `TLWT t, RAM`

Before Instruction

t	=	0	
RAM	=	0xB7	
TBLAT	=	0x0000	(TBLATH = 0x00) (TBLATL = 0x00)

After Instruction

RAM	=	0xB7	
TBLAT	=	0x00B7	(TBLATH = 0x00) (TBLATL = 0xB7)

Before Instruction

t	=	1	
RAM	=	0xB7	
TBLAT	=	0x0000	(TBLATH = 0x00) (TBLATL = 0x00)

After Instruction

RAM	=	0xB7	
TBLAT	=	0xB700	(TBLATH = 0xB7) (TBLATL = 0x00)

TSTFSZ — Test f, skip if 0

Syntax:	[*label*] TSTFSZ f
Operands:	0 ≤ f ≤ 255
Operation:	skip if f = 0
Status Affected:	None
Encoding:	`0011` `0011` `ffff` `ffff`
Description:	If 'f' = 0, the next instruction, fetched during the current instruction execution, is discarded and a `NOP` is executed, making this a two-cycle instruction.
Words:	1
Cycles:	1 (2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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

Example:

```
HERE     TSTFSZ  CNT
NZERO      :
ZERO       :
```

Before Instruction
PC = Address (`HERE`)

After Instruction

If CNT	=	0x00,
PC	=	Address (`ZERO`)
If CNT	¼	0x00,
PC	=	Address (`NZERO`)

FIGURE 20-11: CAPTURE TIMINGS

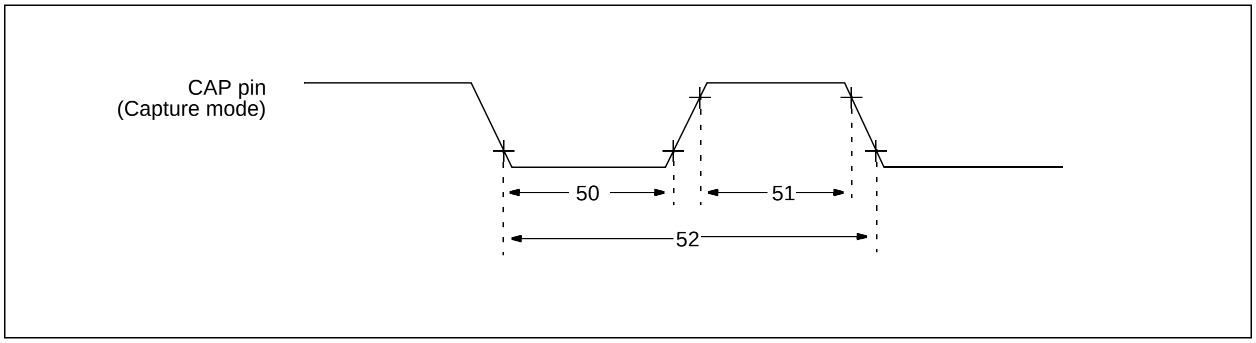


TABLE 20-6: CAPTURE REQUIREMENTS

Param No.	Sym	Characteristic	Min	Typ †	Max	Units	Conditions
50	TccL	Capture pin input low time	10	—	—	ns	
51	TccH	Capture pin input high time	10	—	—	ns	
52	TccP	Capture pin input period	$\frac{2T_{CY}}{N}$	—	—	ns	N = prescale value (4 or 16)

† Data in “Typ” column is at 5V, 25°C unless otherwise stated.

FIGURE 20-12: PWM TIMINGS

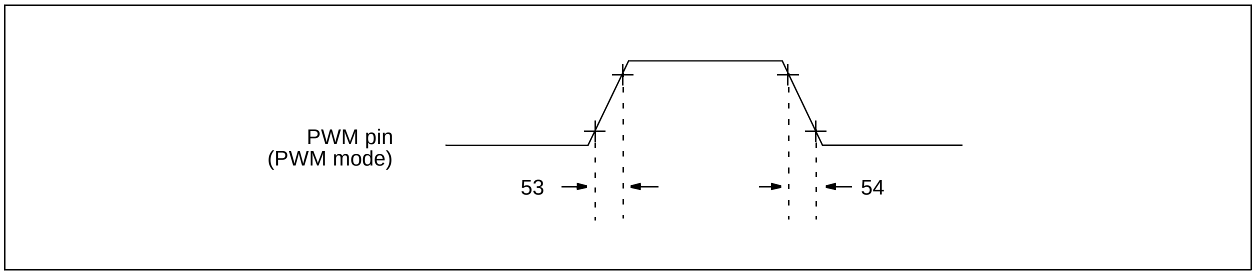


TABLE 20-7: PWM REQUIREMENTS

Param No.	Sym	Characteristic	Min	Typ †	Max	Units	Conditions
53	TccR	PWM pin output rise time	—	10	35	ns	
54	TccF	PWM pin output fall time	—	10	35	ns	

† Data in “Typ” column is at 5V, 25°C unless otherwise stated.