TECNOLÓGICO DE MONTERREY®

Subject 8
ADVANCED USE OF TIMERS

Capture

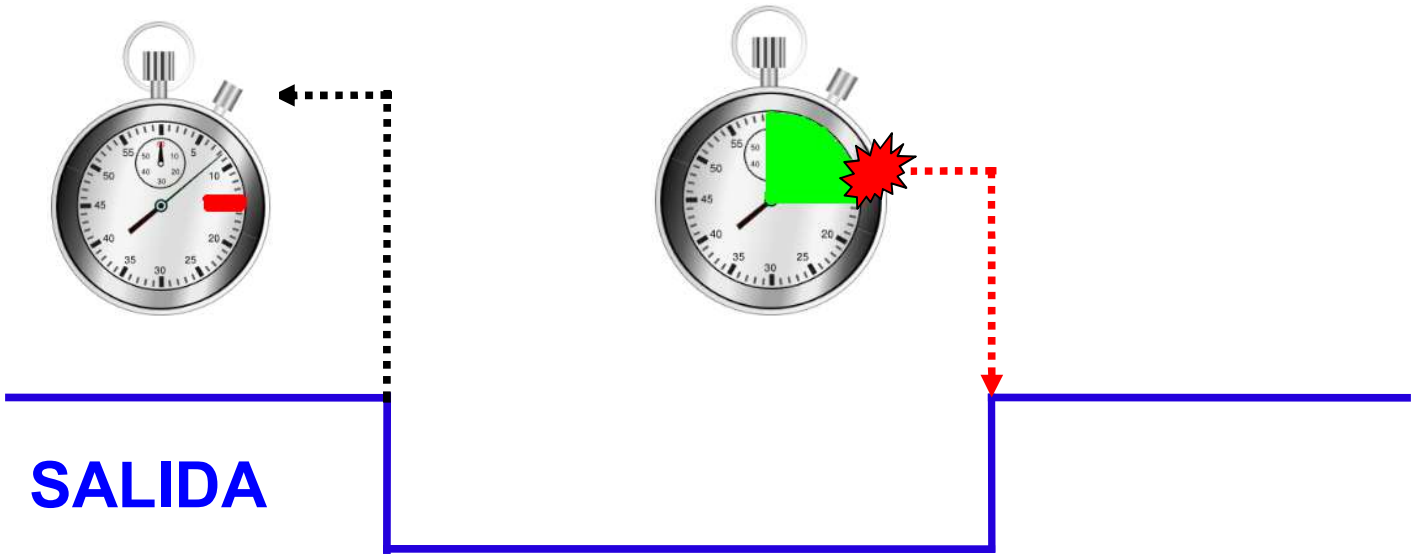
- Used to measure the duration of an event

Capture

MakeAGIF.com

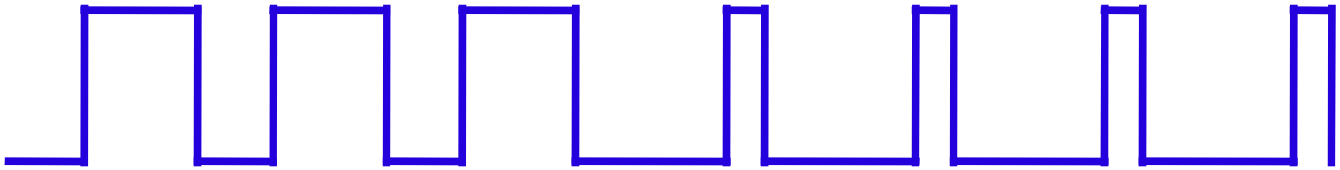
Compare

- Trigers hardware events after a pre-configured time lapse.



PWM

- Generates pulse width modulation signals (configurable frequency and duty cycle)



SALIDA

Modules CAPTURE/COMPARE/PWM

- The PIC18F45K50 provides
  - An enhanced module (ECCP1)
  - A conventional standard module (CCP2)

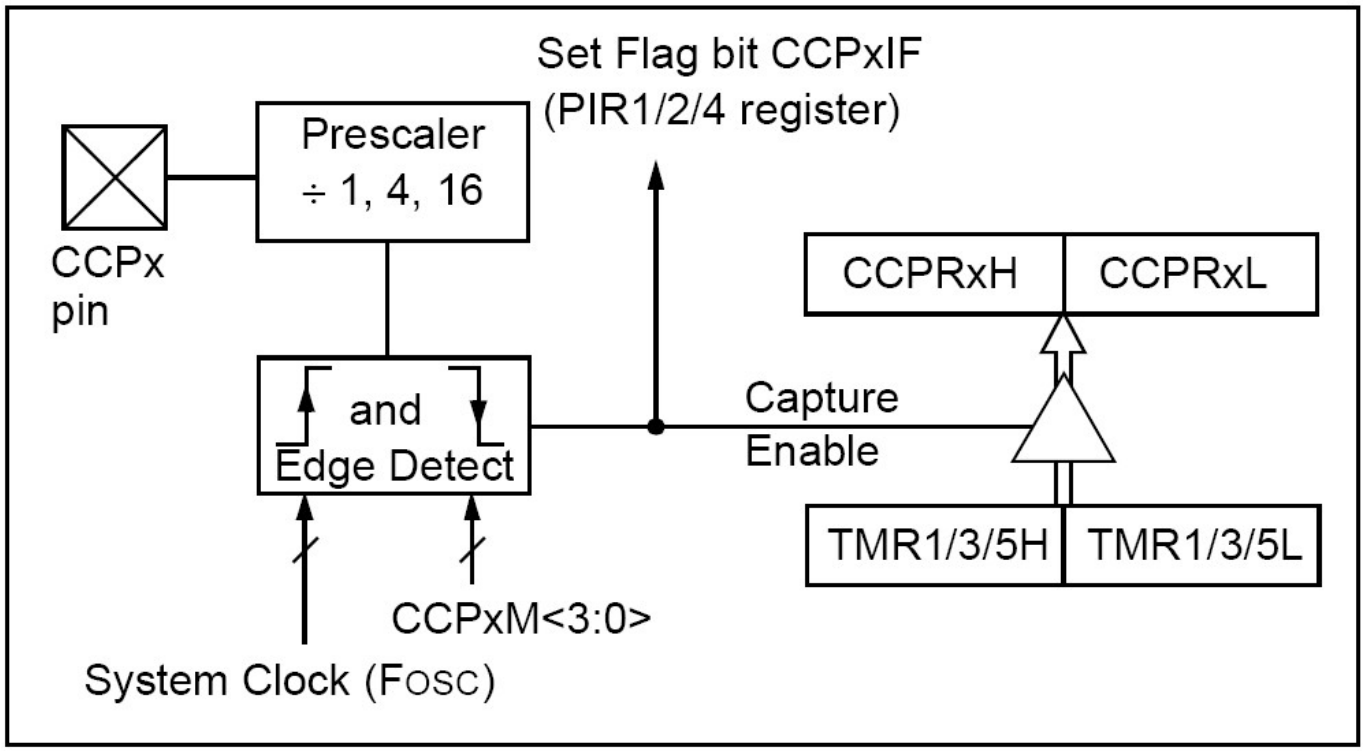
Capture

- Identical in the two modules CPP y ECCP
- It requires a 16 bits timer, Timer1, Timer3
- When a change of state occurs in the CPPx pin, the count register is copied to CCPRxH:CCPRxL
- The “event” is captured on the positive or negative transition of the pin, it can also be after several transitions. (4^{th} or 10^{th})

Capture

- Each capture , the interrupt flag CCPxIF los located on registers PIR1, PIR2 is set to one
- The flag must be cleared by firmware
- If the capture is unattended the CCPRxH:CCPRxL will be rewritten

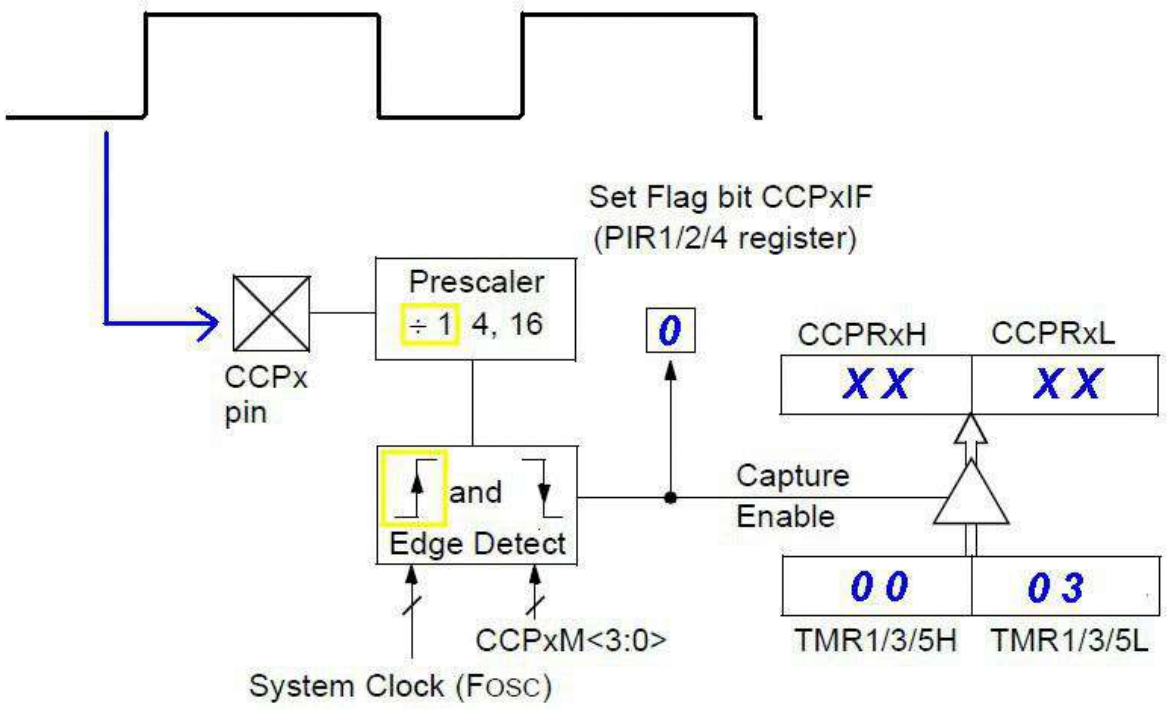
Capture



CCP1: Is assigned to pin RC2

CCP2: Assigned to pint RC1 (can be changed to RB3)

Capture



MakeAGIF.com

Configuration of the pin CPPx

- The CCPx pin that triggers the capture must be configured as an in input using the TRISx register
- There is a transition pre-scale, avoid changing its configuration value on when the module in operation since it can false trigger events.

CPPxCON Register

REGISTER 15-1: CCPxCON: STANDARD CCPx CONTROL REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB<1:0>		CCPxM<3:0>			
bit 7							bit 0

bit 7-6 **Unused**

bit 5-4 **DCxB<1:0>:** PWM Duty Cycle Least Significant bits

Capture mode:
Unused

Compare mode:
Unused

PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCPxM<3:0>:** ECCPx Mode Select bits

0000 = Capture/Compare/PWM off (resets the module)
0001 = Reserved
0010 = Compare mode: toggle output on match
0011 = Reserved

0100 = Capture mode: every falling edge
0101 = Capture mode: every rising edge
0110 = Capture mode: every 4th rising edge
0111 = Capture mode: every 16th rising edge

1000 = Compare mode: set output on compare match (CCPx pin is set, CCPxIF is set)
1001 = Compare mode: clear output on compare match (CCPx pin is cleared, CCPxIF is set)
1010 = Compare mode: generate software interrupt on compare match (CCPx pin is unaffected, CCPxIF is set)
1011 = Compare mode: Special Event Trigger (CCPx pin is unaffected, CCPxIF is set)
TimerX (selected by CxTSEL bits) is reset
ADON is set, starting A/D conversion if A/D module is enabled(1)
11xx =: PWM mode

Note 1: This feature is available on CCP5 only.

CPPx Register

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on page
CCP1CON	P1M<1:0>		DC1B<1:0>		CCP1M<3:0>				206
CCP2CON	—	—	DC2B<1:0>		CCP2M<3:0>				206

Timer select register

REGISTER 15-3: CCPTMRS: PWM TIMER SELECTION CONTROL REGISTER 0

U-0	U-0	U-0	U-0	R/W-0	U-0	U-0	R/W-0
—	—	—	—	C2TSEL	—	—	C1TSEL
bit 7							bit 0

bit 7-4 **Unimplemented:** Read as '0'

bit 3 **C2TSEL:** CCP2 Timer Selection bit
0 = CCP2 – Capture/Compare modes use TMR1, PWM modes use TMR2
1 = CCP2 – Capture/Compare modes use TMR3, PWM modes use TMR2

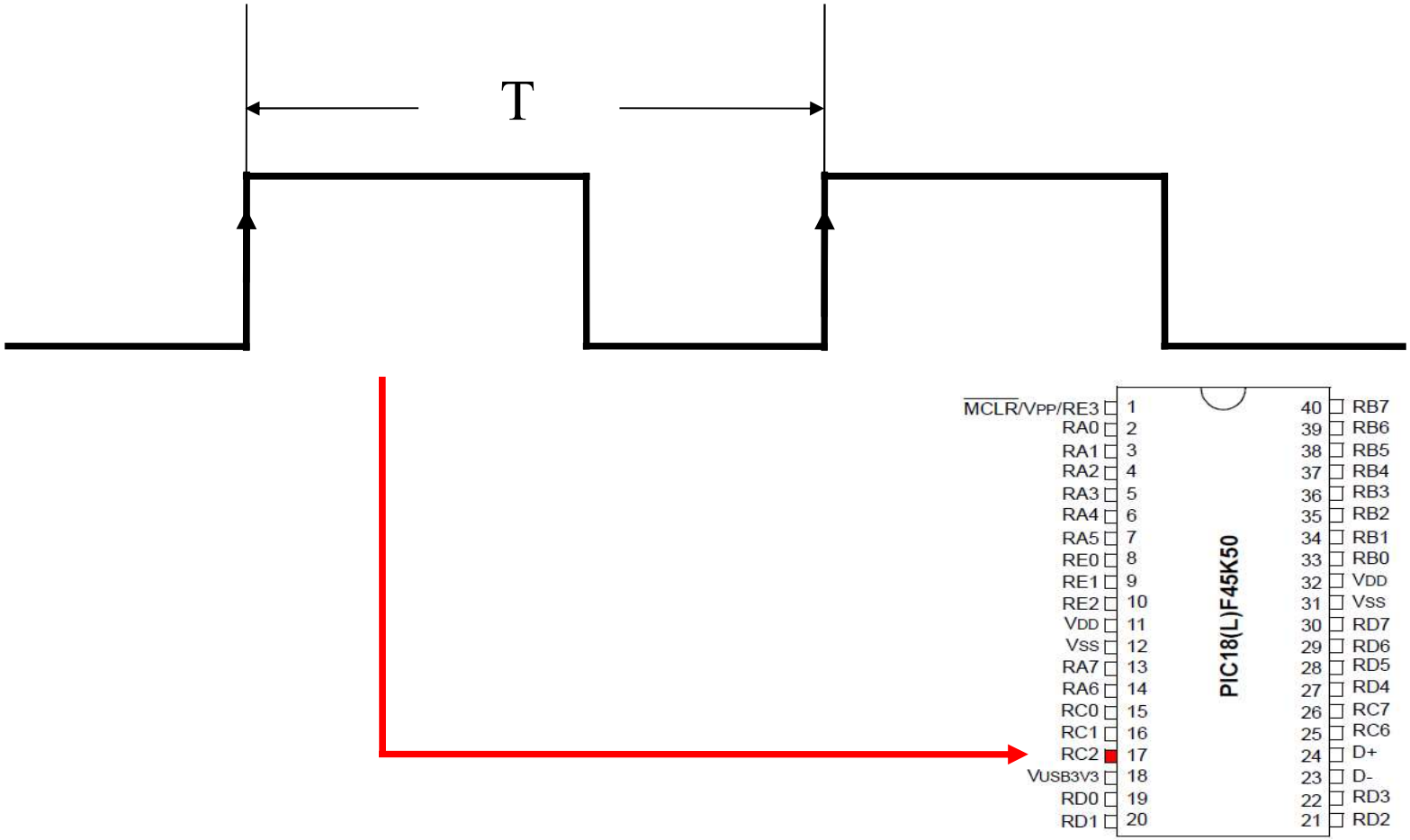
bit 2-1 **Unimplemented:** Read as '0'

bit 0 **C1TSEL:** ECCP1 Timer Selection bit
0 = ECCP1 – Capture/Compare modes use TMR1, PWM modes use TMR2
1 = ECCP1 – Capture/Compare modes use TMR3, PWM modes use TMR2

Application for the capture mode

- Measuring the duration of an event
- Measuring the period of a signal
- General purpose interrupt generation on signal transitions
- Event counting

Example of period measurement using Timer1 and CCP1



Example, measuring the period of a signal using Timer1 y CCP1

- Suppose an oscillator of16Mhz
- The program must measure the period of a signal of 35Hz to 80Hz
- The periods to measure will then be1/35 to 1/80 or 0.028 to 0.0125 seconds
- The worst case will be the longest period, that is the 0.028 seconds since the longer the time to measure means more timer ticks to count

Example, measuring the period of a signal using Timer1 y CCP1

- We need to find a time-base (input clock) for Timer 1 that allows me to count the timer ticks that represent 28 msec. The smaller the timer tick value the better since it will give greater resolution of the measurement and this means more accuracy.

Example, measuring the period of a signal using Timer1 y CCP1

- We will use the CCP1 input (RC2)
- Timer 1 resolution is 16 bits.
- If we use the internal clock source of Fosc/4 with a pre-scaler of 2 the value of each timer tick will be 8/Fos = 0.5 usec
- We will capture each positive transition of CCP1
- We wont use interrupts (polled method):

Ejemplo medición de período usando Timer1 y CCP1

- Since each timer tick will be 0.5u seconds. The maximum time value the TMR1 count register can count will be 0.5usec * 65536 = 0.032 msec before it overflows
- This means we can measure a frequency starting at 30.61Hz and up
- The highest the frequency the bigger the error since it we will approach to the timer tick value .

```
#include<xc.h>
void init_ports(void);          //Inits pors and board oscillator

main(){
unsigned int period1 = 0;       //Stores the first capture
unsigned int period2 = 0;       //Stores the second capture
unsigned int total_period = 0;
unsigned long frequency = 0;    //Used in debug to know the frequency
init_ports();               //Init the ports

//Configure module CCP1
CCP1CON = 0X05;     //Capture each positive transition
//Tell the module to use TIMER1 as the time base
CCPTMRS = 0x00;    //Assign TIMER1 to CCP1

//Configure Timer 1 (T1CON), 16 bits
T1CONbits.TMR1CS = 0b00;        //Clock source FOSC/4
T1CONbits.T1CKPS = 0b01;        //Prescale 1:2
T1CONbits.T1SOSCEN = 0b0;       //Secundary oscillator disabled
T1CONbits.T1SYNC = 0b1;         //External clock sync
//T1CONbits.T1RD16 = 0b0;         //Deshabilita escritura 16 bits (no importa)
//The later defines = T1CON     = 0b0001010x we still need to start timer
//Gate configuration to start the timer
T1GCONbits.TMR1GE = 0;           //Gate control not required we use firmware
//Start timer 1
T1CONbits.TMR1ON = 1;
```

```
//Configure Timer 1 (T1CON), 16 bits
T1CONbits.TMR1CS = 0b00;        //Clock source FOSC/4
T1CONbits.T1CKPS = 0b01;        //Prescale 1:2
T1CONbits.T1SOSCEN = 0b0;       //Secundary oscillator disabled
T1CONbits.T1SYNC = 0b1;         //External clock sync
//T1CONbits.T1RD16 = 0b0;         //Deshabilita escritura 16 bits (no importa)
//The later defines = T1CON     = 0b0001010x we still need to start timer
//Gate configuration to start the timer
T1GCONbits.TMR1GE = 0;           //Gate control not required we use firmware
//Start timer 1
T1CONbits.TMR1ON = 1;

while(PIR1bits.CCP1IF == 0);        //Wait for the timer to be captured first
 period1 = CCPR1;                   //transition and load value to period1
 PIR1bits.CCP1IF = 0;               //Clear the flag that indicates the event

while(PIR1bits.CCP1IF == 0);         //Wait for the second transition
period2 = CCPR1;                     //Read the captured value
T1CONbits.TMR1ON = 0;                //Turn off timer 1

//To know the period value in time, you must multiply the ticks but the value the
//time value of each Tick.
//Its more clear to see the frequency instead of the period.
//To make this more clear, lets calculate the frequency
//The peridod in secods = (total_period)*(2)(4/Fosc)
//Frequency in hertz = Fosc/(total_period*8), Since we know Fosc
//Frequency in hertz = 16,000,000/(8 *total_period)
//Frequency in herts = 2,000,000 /total_period
//If we want to see de 10th's of HZ we add anoter 0 ie 20,000,000
frequency = 20000000/total_period;
while(1); //We place a break in this instruction and a watch on variable frequency

} //from main() TEMA_08_PIC_6.C

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that inits the internal oscillator at 16Mhz
//+ and the ports
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void init_ports(void){
    //RC2 is input for CCP1
    TRISCbits.TRISC2 = 1;   //Input
    ANSELCbits.ANSC2 = 0;   //Digital
    //Set the internal oscillator frequency to 16Mhz
    OSCCON = 0b01111100;
}
```

REGISTER 15-1: CCPxCON: STANDARD CCPx CONTROL REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB<1:0>		CCPxM<3:0>			
bit 7							bit 0

bit 7-6 **Unused**

bit 5-4 **DCxB<1:0>:** PWM Duty Cycle Least Significant bits

Capture mode:
Unused

Compare mode:
Unused

PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCPxM<3:0>:** ECCPx Mode Select bits

0000 = Capture/Compare/PWM off (resets the module)
0001 = Reserved
0010 = Compare mode: toggle output on match
0011 = Reserved

0100 = Capture mode: every falling edge
0101 = Capture mode: every rising edge
0110 = Capture mode: every 4th rising edge
0111 = Capture mode: every 16th rising edge

1000 = Compare mode: set output on compare match (CCPx pin is set, CCPxIF is set)
1001 = Compare mode: clear output on compare match (CCPx pin is cleared, CCPxIF is set)
1010 = Compare mode: generate software interrupt on compare match (CCPx pin is unaffected, CCPxIF is set)
1011 = Compare mode: Special Event Trigger (CCPx pin is unaffected, CCPxIF is set)
TimerX (selected by CxTSEL bits) is reset
ADON is set, starting A/D conversion if A/D module is enabled(1)
11xx =: PWM mode

Note 1: This feature is available on CCP5 only.

REGISTER 15-3: CCPTMRS: PWM TIMER SELECTION CONTROL REGISTER 0

U-0	U-0	U-0	U-0	R/W-0	U-0	U-0	R/W-0
—	—	—	—	C2TSEL	—	—	C1TSEL
bit 7							bit 0

bit 7-4 **Unimplemented:** Read as '0'

bit 3 **C2TSEL:** CCP2 Timer Selection bit
0 = CCP2 – Capture/Compare modes use TMR1, PWM modes use TMR2
1 = CCP2 – Capture/Compare modes use TMR3, PWM modes use TMR2

bit 2-1 **Unimplemented:** Read as '0'

bit 0 **C1TSEL:** ECCP1 Timer Selection bit
0 = ECCP1 – Capture/Compare modes use TMR1, PWM modes use TMR2
1 = ECCP1 – Capture/Compare modes use TMR3, PWM modes use TMR2

13.13 Register Definitions: Timer1/3 Control

REGISTER 13-1: TxCON: TIMER1/3 CONTROL REGISTER

R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/0	R/W-0/u
TMRxCS<1:0>		TxCKPS<1:0>		SOSCEN	$\overline{\text{TxSYNC}}$	RD16	TMRxON
bit 7							bit 0

bit 7-6 **TMRxCS<1:0>:** Timer1/3 Clock Source Select bits
11 = Reserved. Do not use.
10 = Timer1/3 clock source is pin or oscillator:
If SOSCEN = 0:
External clock from TxCKI pin (on the rising edge)
If SOSCEN = 1:
Crystal oscillator on SOSCI/SOSCO pins
01 = Timer1/3 clock source is system clock (FOSC)
00 = Timer1/3 clock source is instruction clock (FOSC/4)

bit 5-4 **TxCKPS<1:0>:** Timer1/3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **SOSCEN:** Secondary Oscillator Enable Control bit
1 = Dedicated secondary oscillator circuit enabled
0 = Dedicated secondary oscillator circuit disabled

bit 2 **$\overline{\text{TxSYNC}}$:** Timer1/3 External Clock Input Synchronization Control bit
TMRxCS<1:0> = 1X
1 = Do not synchronize external clock input
0 = Synchronize external clock input with system clock (FOSC)

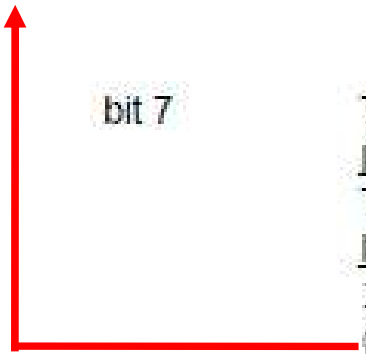
TMRxCS<1:0> = 0X
This bit is ignored. Timer1/3 uses the internal clock when TMRxCS<1:0> = 0X.

bit 1 **RD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer1/3 in one 16-bit operation
0 = Enables register read/write of Timer1/3 in two 8-bit operation

bit 0 **TMRxON:** Timer1/3 On bit
1 = Enables Timer1/3
0 = Stops Timer1/3
Clears Timer1/3 Gate flip-flop

REGISTER 13-2: TxGCON: TIMER1/3 GATE CONTROL REGISTER

R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W/HC-0/u	R-x/x	R/W-0/u	R/W-0/u
TMRxGE	TxGPOL	TxGTM	TxGSPM	TxGGO/$\overline{\text{DONE}}$	TxGVAL	TxGSS<1:0>	
bit 7							bit 0



bit 7 **TMRxGE:** Timer1/3/5 Gate Enable bit

If TMRxON = 0:

This bit is ignored

If TMRxON = 1:

1 = Timer1/3/5 counting is controlled by the Timer1/3/5 gate function

0 = Timer1/3/5 counts regardless of Timer1/3/5 gate function

REGISTER 13-2: TxGCON: TIMER1/3 GATE CONTROL REGISTER

R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W/HC-0/u	R-x/x	R/W-0/u	R/W-0/u
TMRxGE	TxGPOL	TxGTM	TxGSPM	TxGGO/$\overline{\text{DONE}}$	TxGVAL	TxGSS<1:0>	
bit 7							bit 0

bit 7 **TMRxGE:** Timer1/3/5 Gate Enable bit

If TMRxON = 0:
This bit is ignored
If TMRxON = 1:
1 = Timer1/3/5 counting is controlled by the Timer1/3/5 gate function
0 = Timer1/3/5 counts regardless of Timer1/3/5 gate function

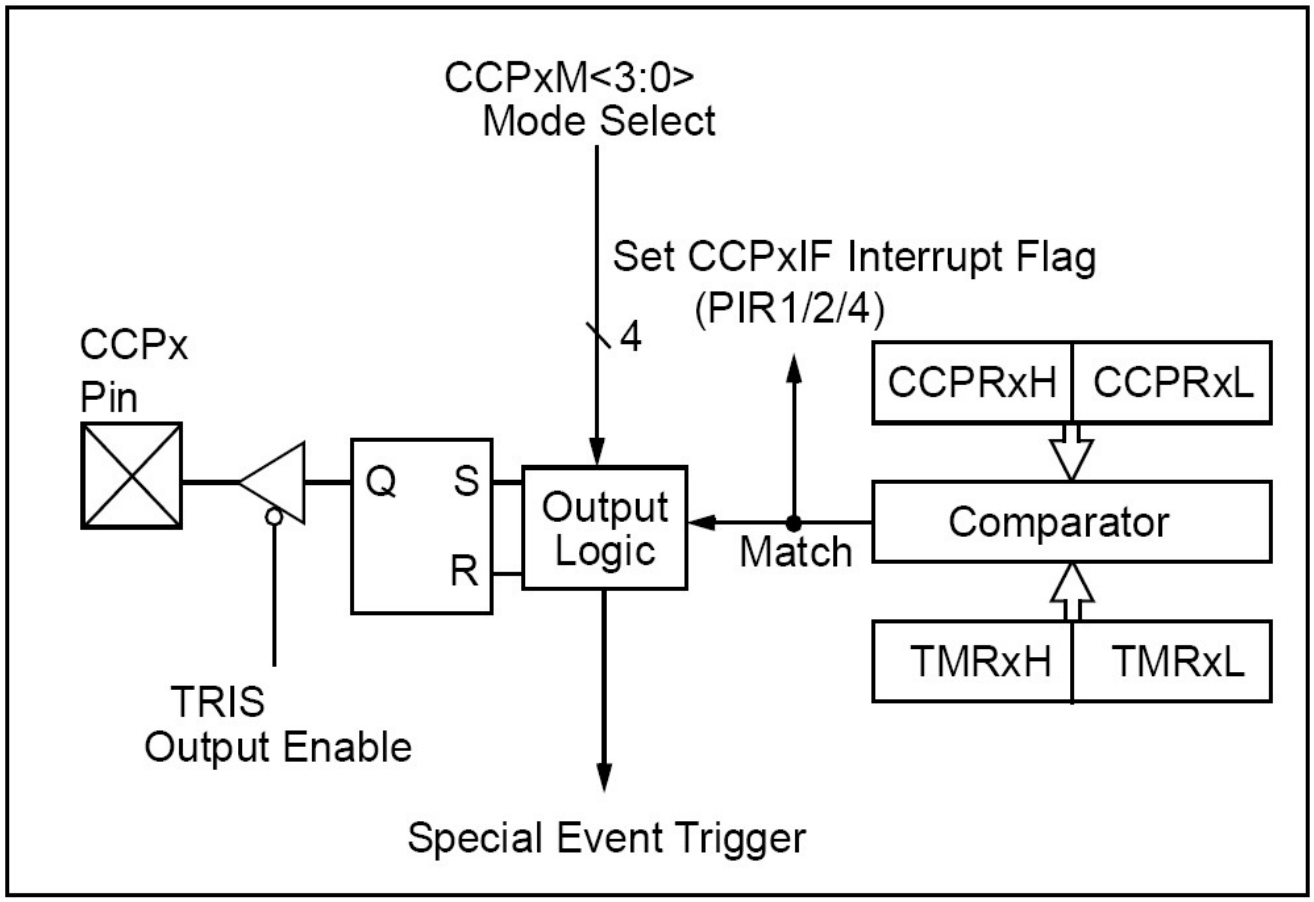
¿How to measure longer periods?

- Increasing the pre-scaler value of the timer
- Use interrupts
- Each overflow of the timer will trigger an interrupt, in the ISR you can have a firmware counter.
- Each overflow means that 65536 must be added to the total count of timer ticks
- Example in book section 8.6 (page 357).

Compare (Comparación)

- En este modo el registro CCPRx es comparado constantemente con los registros de TMR1\3
- Cuando ocurre una igualación las siguientes acciones pueden ocurrir en el pin CCPx
  - Estado alto
  - Estado bajo
  - Generar un evento de disparo
  - Generar una interrupción

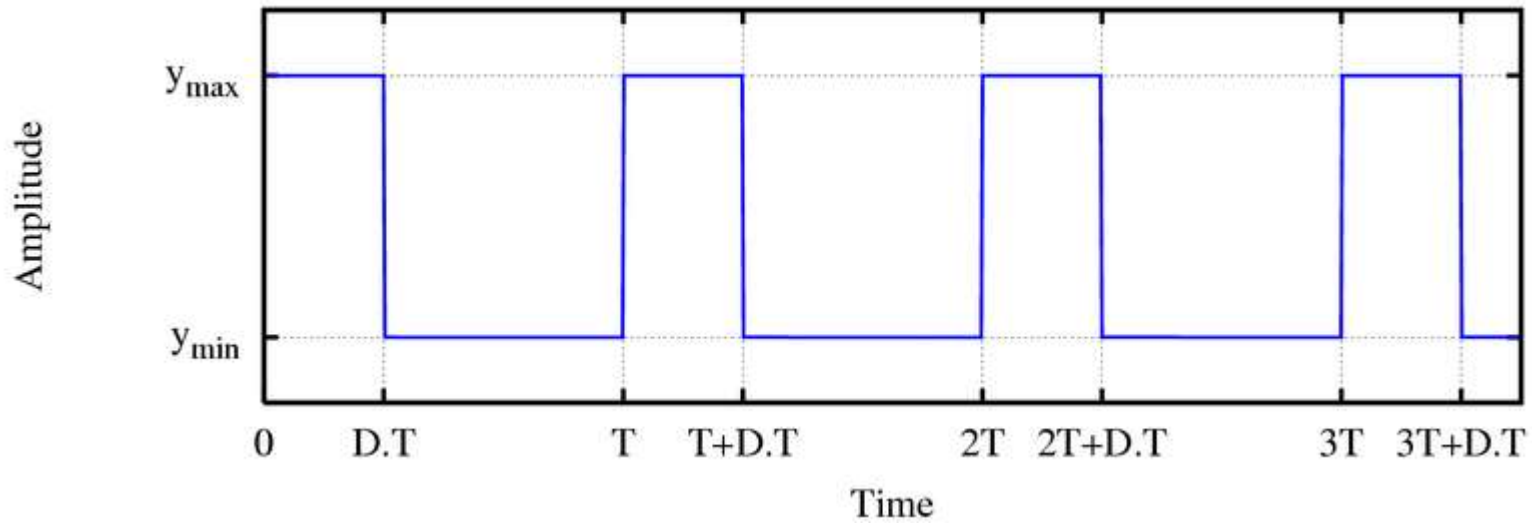
Compare (Comparación)



PWM

- PWM
  - Pulse Width Modulation
- PWM applications
  - Digital an analog conversion
  - Motor speed control
  - Audio generator
  - DC/DC converters (Regulators)
  - Inverters (DC to AC converters)
  - Data transmisssion

The average voltage of a square wave

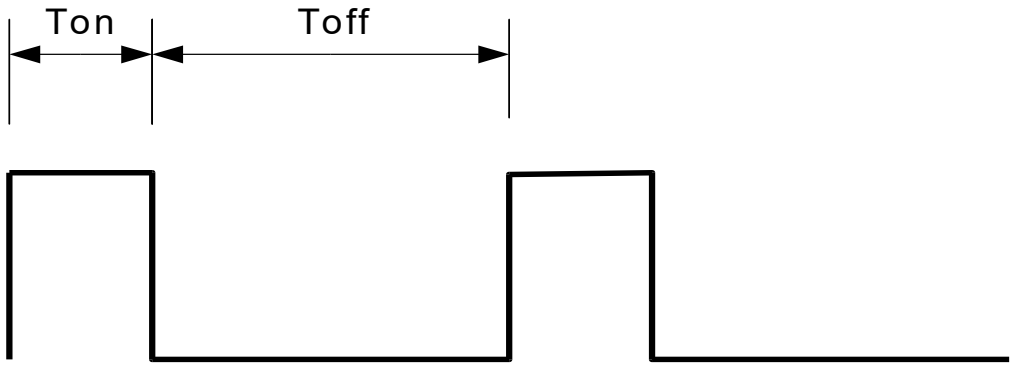


$$\bar{y} = \frac{1}{T}\int_0^T f(t)\,dt.$$

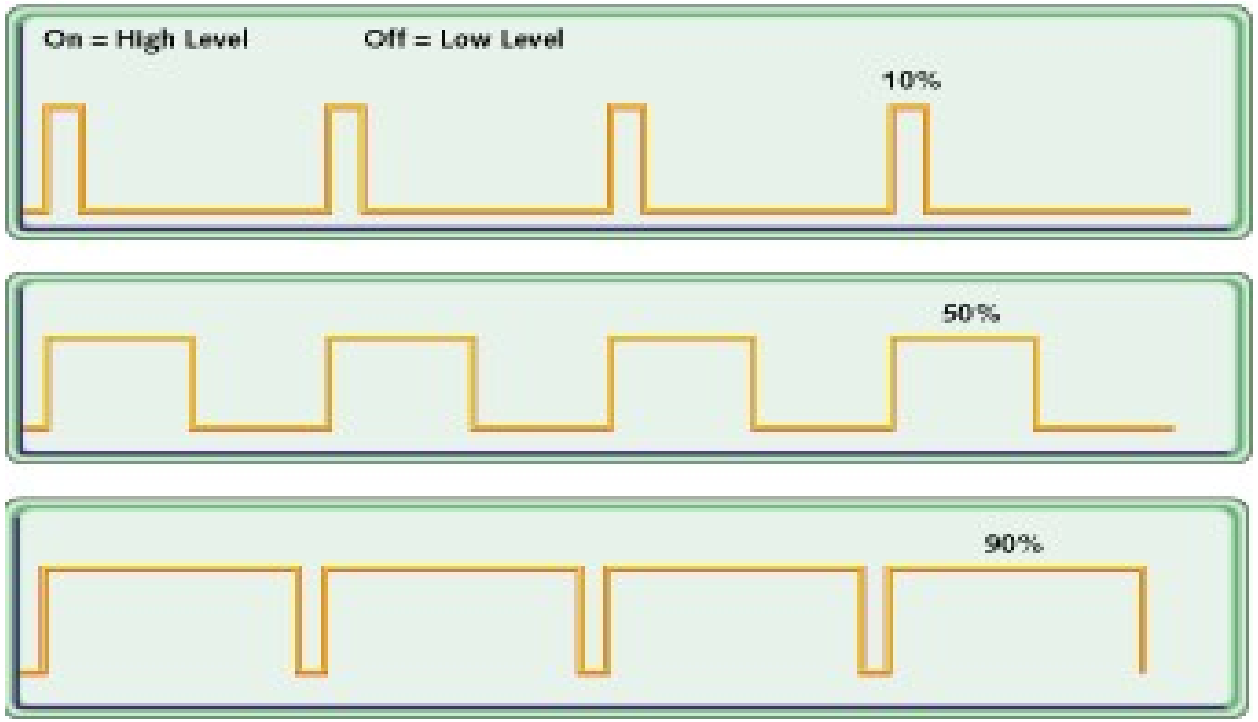
$$\begin{aligned}\bar{y} &= \frac{1}{T}\left(\int_0^{DT} y_{max}\,dt + \int_{DT}^{T} y_{min}\,dt\right)\\ &= \frac{D\cdot T\cdot y_{max} + T\,(1-D)\,y_{min}}{T}\\ &= D\cdot y_{max} + (1-D)\,y_{min}\end{aligned}$$

The average voltage of a square wave

- For a signal with a Ymax = Vcc and Ymin = 0.0V
- The average voltage is = D*Vcc
  - Where D is the duty cycle
  - And the duty cycle D = Ton/(Ton+Toff)



Ymax = 5V, Ymin = 0V

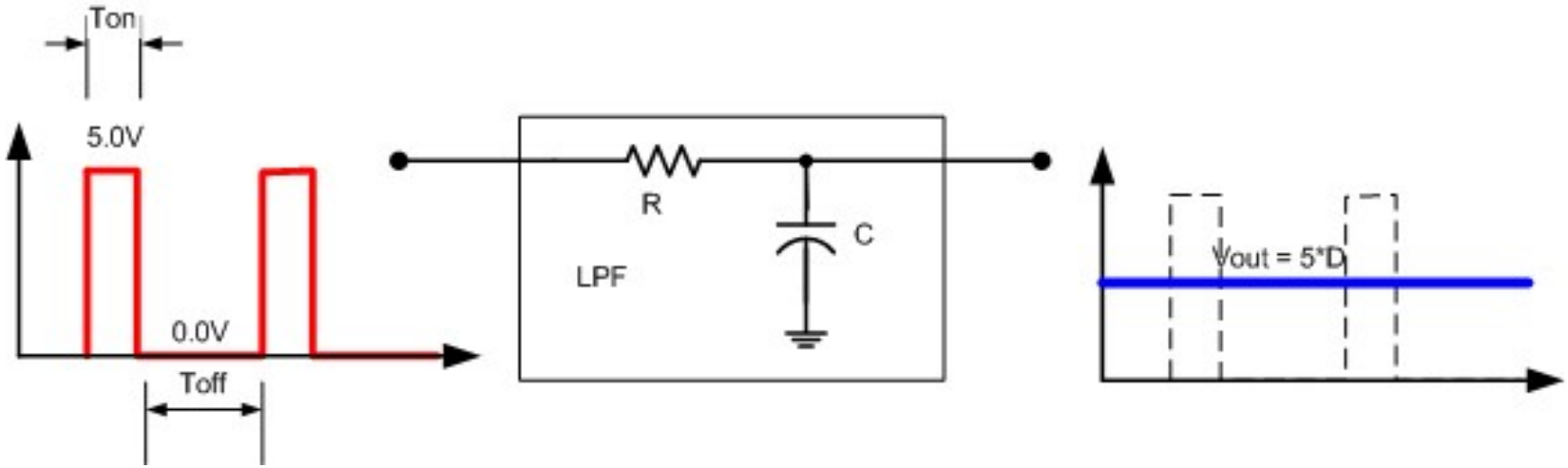


Average = 0.5V

Aveage= 2.5V

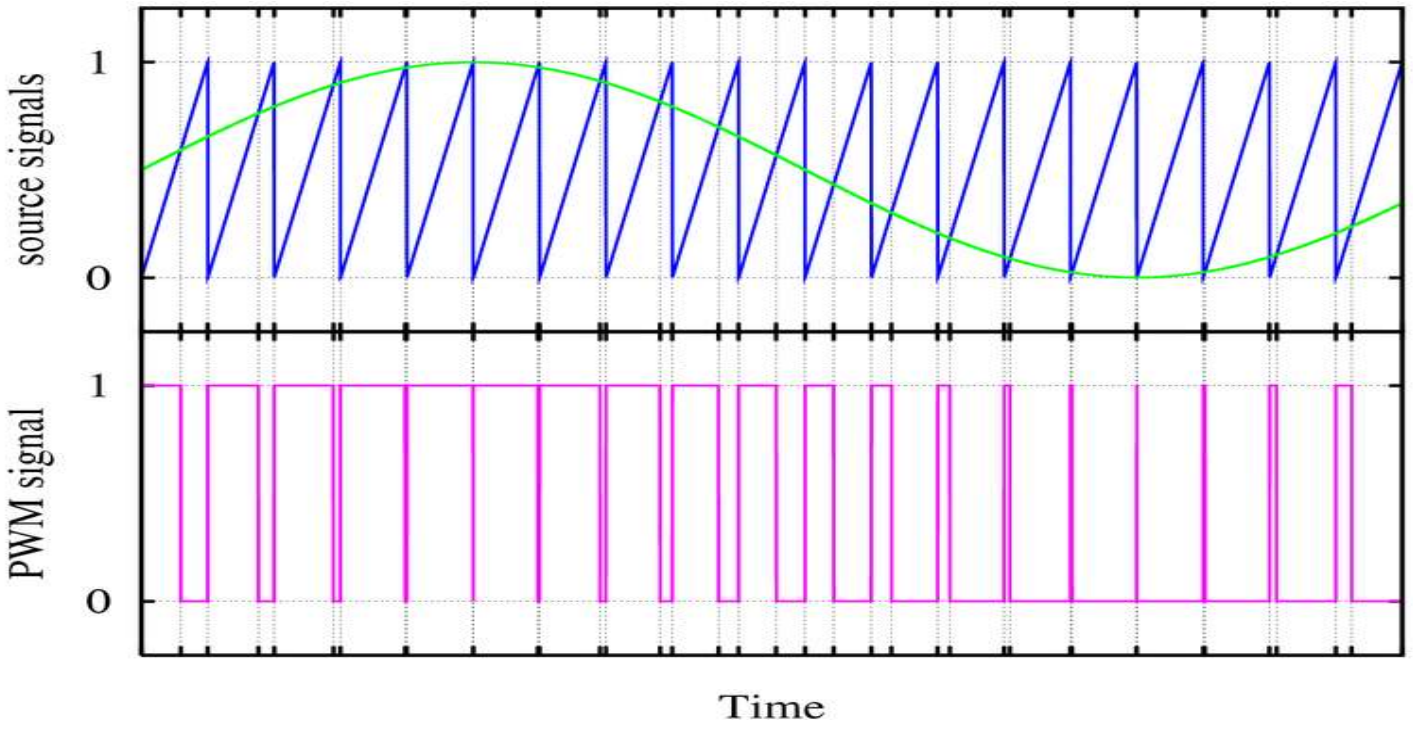
Average= 4.5V

Electrically obtaining the average value of the square wave



See simulation

How to obtain the PWM from a signal



See simulation

Standard operation of PWM

- Identican in the two modules CPP and ECPP
- The maximum resolution is 10 bits
- The control registers are:

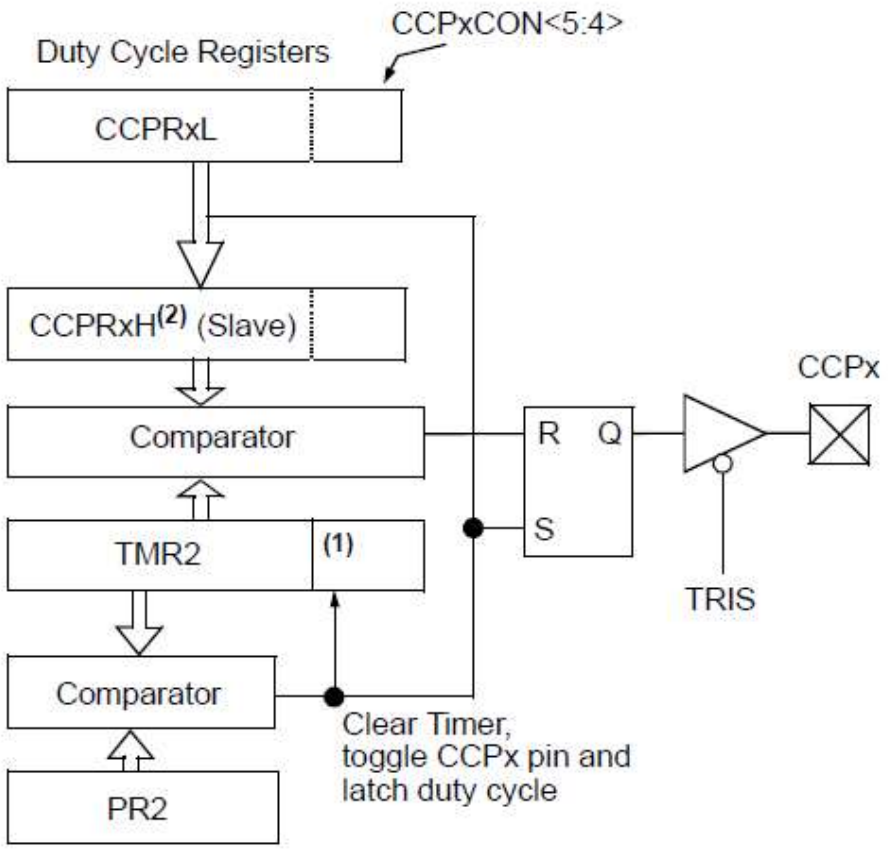
  PR2 registers

  T2CON registers

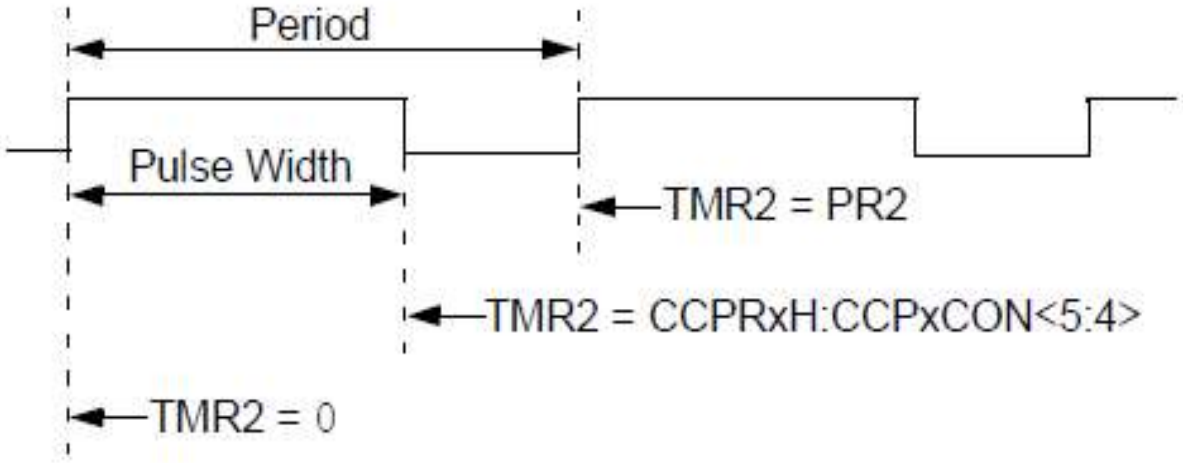
  CCPRxL registers

  CCPxCON registers

Simplified PWM module



The PWM signal



The PWM configuration

15.3.2 SETUP FOR PWM OPERATION

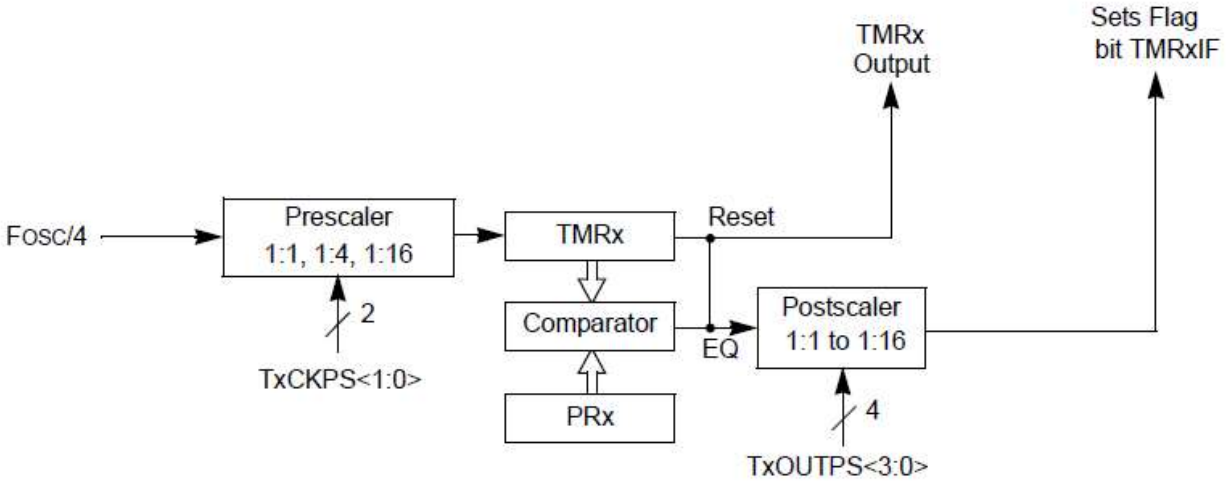
The following steps should be taken when configuring the CCP module for standard PWM operation:

1. Disable the CCPx pin output driver by setting the associated TRIS bit.
2. Load the PR2 register for Timer2 with the PWM period value.
3. Configure the CCP module for the PWM mode by loading the CCPxCON register with the appropriate values.
4. Load the CCPRxL register and the DCxB<1:0> bits of the CCPxCON register, with the PWM duty cycle value.
5. Configure and start the 8-bit Timer2:
   - Clear the TMR2IF interrupt flag bit of the PIR1 register. See Note 1 below.
   - Configure the T2CKPS bits of the T2CON register with the Timer prescale value.
   - Enable the Timer by setting the TMR2ON bit of the T2CON register.
6. Enable PWM output pin:
   - Wait until the Timer overflows and the TMR2IF bit of the PIR1 register is set. See Note 1 below.
   - Enable the CCPx pin output driver by clearing the associated TRIS bit.

Note 1: In order to send a complete duty cycle and period on the first PWM output, the above steps must be included in the setup sequence. If it is not critical to start with a complete PWM signal on the first output, then step 5 may be ignored.

Clock source for the PWM

- We use TIMER 2

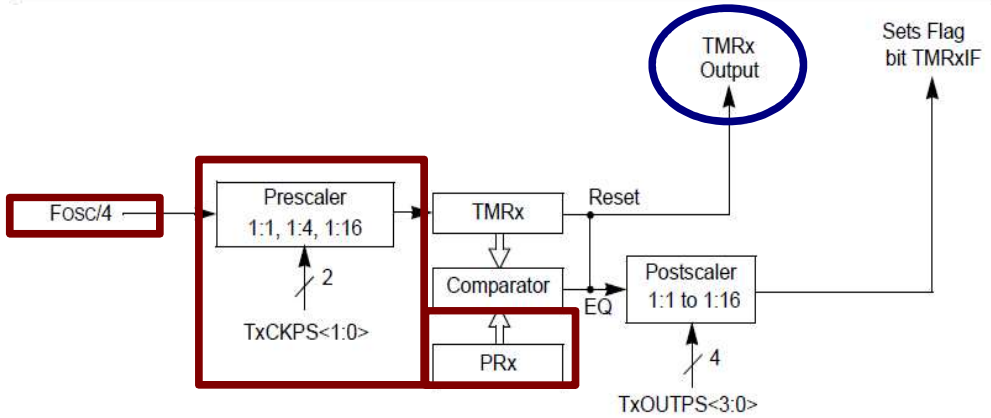


The period of PWM

- The period of the PWM is defined by PRx register Timer2 (8 bits)
- The de PWM period can be calculated by

$$PWM\ Period = [(PR2) + 1] \bullet 4 \bullet Tosc \bullet (TMR2\ Prescale\ Value)$$

Note 1: Tosc = 1/Fosc



EQUATION 15-1: PWM PERIOD

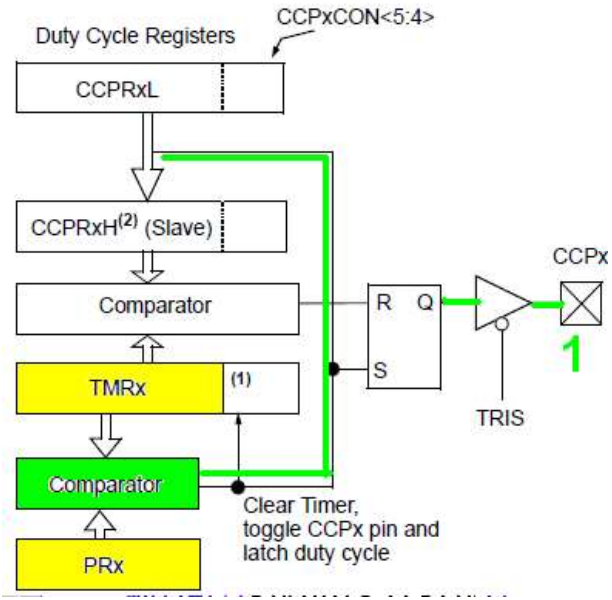
$$PWM\ Period = [(PR2) + 1] \bullet 4 \bullet Tosc \bullet (TMR2\ Prescale\ Value)$$

Note 1: TOSC = 1/FOSC

When TMR2 is equal to PR2, the following three events occur on the next increment cycle:

- TMR2 is cleared
- The CCPx pin is set. (Exception: If the PWM duty cycle = 0%, the pin will not be set.)
- The PWM duty cycle is latched from CCPRxL into CCPRxH.

Note: The Timer postscaler (see **Section 14.0 "Timer2 Module"**) is not used in the determination of the PWM frequency.



The duty cycle

- It is defined by writing the 8 more significant bits in CCPRxL and the least significan bits in (DCxB<1:0>) the register CCPxCON
- This register can be written on run-time.

REGISTER 15-1: CCPxCON: STANDARD CCPx CONTROL REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB<1:0>		CCPxM<3:0>			
bit 7							bit 0

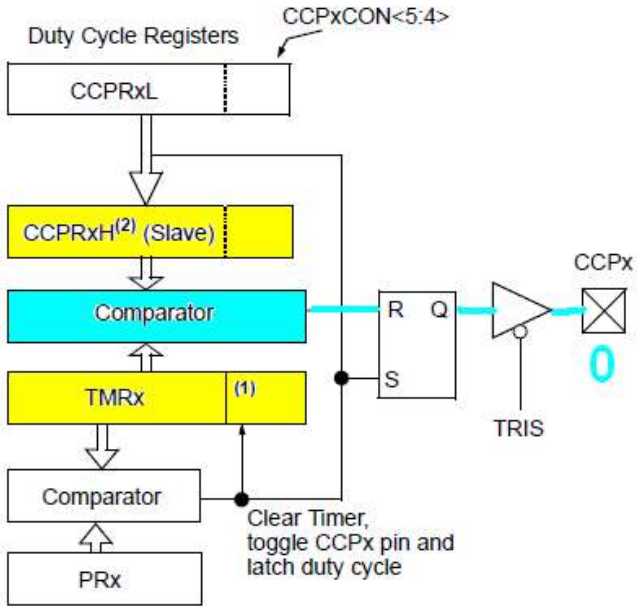
Duty cycle

EQUATION 15-2: PULSE WIDTH

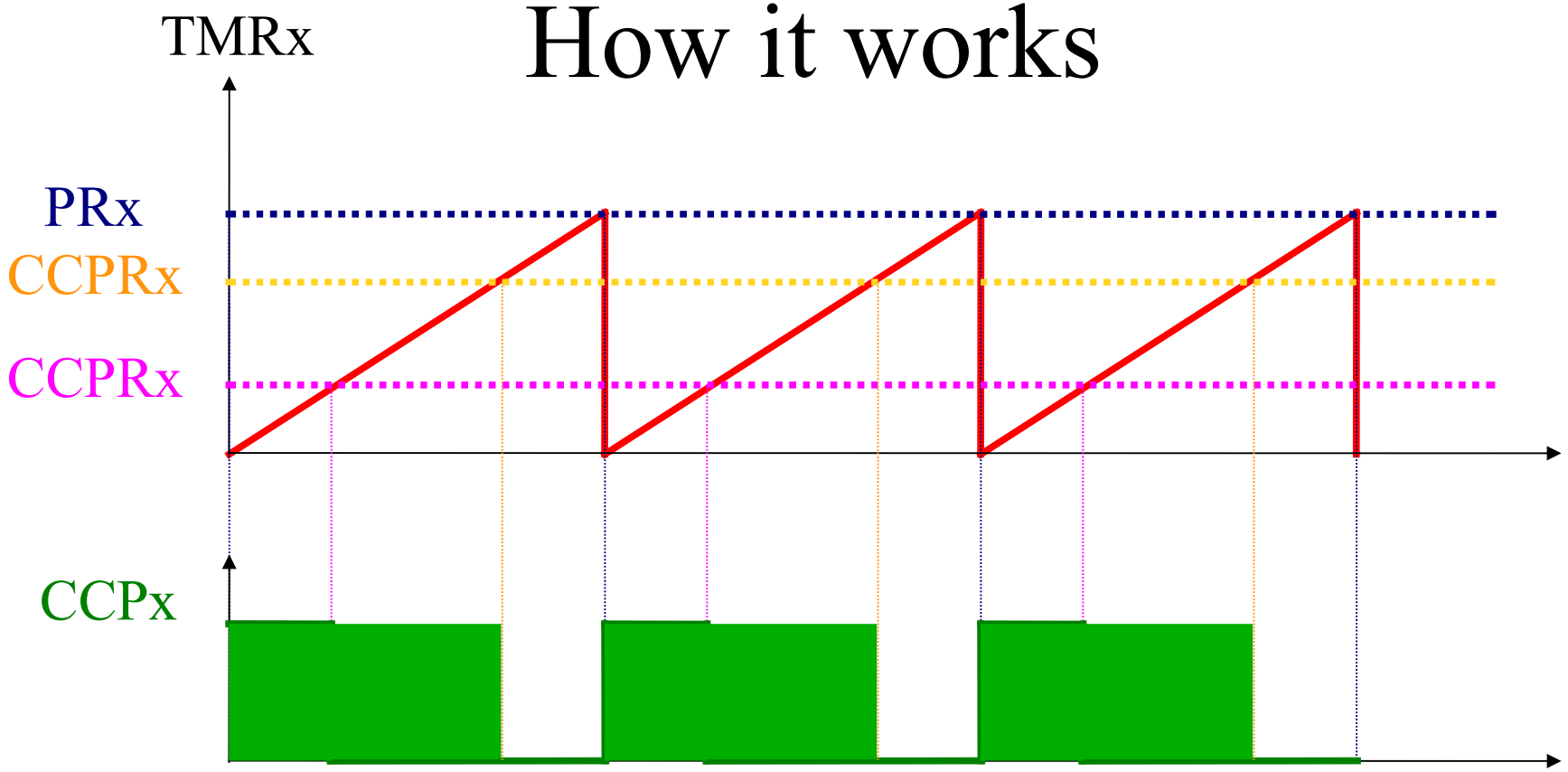
$$Pulse\ Width = (CCPRxL{:}CCPxCON{<}5{:}4{>}) \bullet Tosc \bullet (TMR2\ Prescale\ Value)$$

EQUATION 15-3: DUTY CYCLE RATIO

$$Duty\ Cycle\ Ratio = \frac{(CCPRxL{:}CCPxCON{<}5{:}4{>})}{4(PR2 + 1)}$$



How it works



Resolution

- The resolution is how many possible duty cycle values can there be in a period.
- A resolution of 10 bits means that a period can have 1024 possible values, 8 bits will be 256 possibilities
- The maximum resolution of 10 bits can only be give when the Prx register is equal to 255

$$Resolution = \frac{log[4(PRx+1)]}{log(2)} \ bits$$

Note:	If the pulse width value is greater than the period the assigned PWM pin(s) will remain unchanged.

Resolution

TABLE 15-6: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (Fosc = 32 MHz)

PWM Frequency	1.95 kHz	7.81 kHz	31.25 kHz	125 kHz	250 kHz	333.3 kHz
Timer Prescale (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	6.6

TABLE 15-7: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (Fosc = 20 MHz)

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12 kHz	156.3 kHz	208.3 kHz
Timer Prescale (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	6.6

TABLE 15-8: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (Fosc = 8 MHz)

PWM Frequency	1.22 kHz	4.90 kHz	19.61 kHz	76.92 kHz	153.85 kHz	200.0 kHz
Timer Prescale (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0x65	0x65	0x65	0x19	0x0C	0x09
Maximum Resolution (bits)	8	8	8	6	5	5

Example of PWM

- Configure the CCP1 module as PWM to generate a signal with a frequency of 10Khz and a duty cycle of 40%. The PIC18 has an internal oscillator of 16Mhz.

Solution alternative:

- The TIMER2 is used as time base for the CPP1, lets try to get the best resolution possible
- To calculate the period we have:

$$PWM\ Period = [(PRx) + 1] \bullet 4 \bullet Tosc \bullet (TMRx\ Prescale\ Value)$$

Tosc = 1/Fosc

- 1/10Khz = [PR2 + 1] * 4*(1/16e6) * PRE

  PR2 = [(16e6)/(10e3 * 4 * PRE] – 1

  PRE = 16e6/[10e3*4*(PR2+1)]

For the maximum resolution PR2 = 255

Solution alternative:

Lets see what would be the pre-scaler value if we aim for maximum resolution PR2 = 255

PRE = 16e6/[10e3*4*(255+1)]

PRE = 1.56,

Since this 1.56 value is not possible lets use the closest one since it ca be 1,4, or 16 The closest one would be PRE = 4

Solution alternative:

We now calculate the value of de PR2 using a presale value of 4

PR2 = [(16e6)/(10e3 * 4 * 4] – 1

PR2 = 99

$$Resolution = \frac{log[4(PRx + 1)]}{log(2)} \ bits$$

Resolution = log[4(99+1)]/log(2) =8.64 bits

Solution Alternative

- We now obtain the value of the CCP register to produce a 40% duty cycle on the 10Khz signal :

$$Duty\ Cycle\ Ratio = \frac{(CCPRxL:CCPxCON<5:4>)}{4(PRx + 1)}$$

0.40 = CCPR1L:CCP1CON<5:4> / 4 (99 + 1)

CCPR1L:CCP1CON<5:4> = 160

Solution Alternative

CCPR1L:CCP1CON<5:4> = 160

0010100000b

0x28	0b00
CCPR1L	CCP1CON

Solution alternative

Configuracion del TIMER2

REGISTER 13-1: TxCON: TIMER2/TIMER4/TIMER6 CONTROL REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TxOUTPS<3:0>				TMRxON	TxCKPS<1:0>	
bit 7							bit 0

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

bit 6-3 **TxOUTPS<3:0>:** TimerX Output Postscaler Select bits
0000 = 1:1 Postscaler
0001 = 1:2 Postscaler
0010 = 1:3 Postscaler
0011 = 1:4 Postscaler
0100 = 1:5 Postscaler
0101 = 1:6 Postscaler
0110 = 1:7 Postscaler
0111 = 1:8 Postscaler
1000 = 1:9 Postscaler
1001 = 1:10 Postscaler
1010 = 1:11 Postscaler
1011 = 1:12 Postscaler
1100 = 1:13 Postscaler
1101 = 1:14 Postscaler
1110 = 1:15 Postscaler
1111 = 1:16 Postscaler

bit 2 **TMRxON:** TimerX On bit
1 = TimerX is on
0 = TimerX is off

bit 1-0 **TxCKPS<1:0>:** Timer2-type Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Post scaler is not used by the module

T2CON = 00000101b

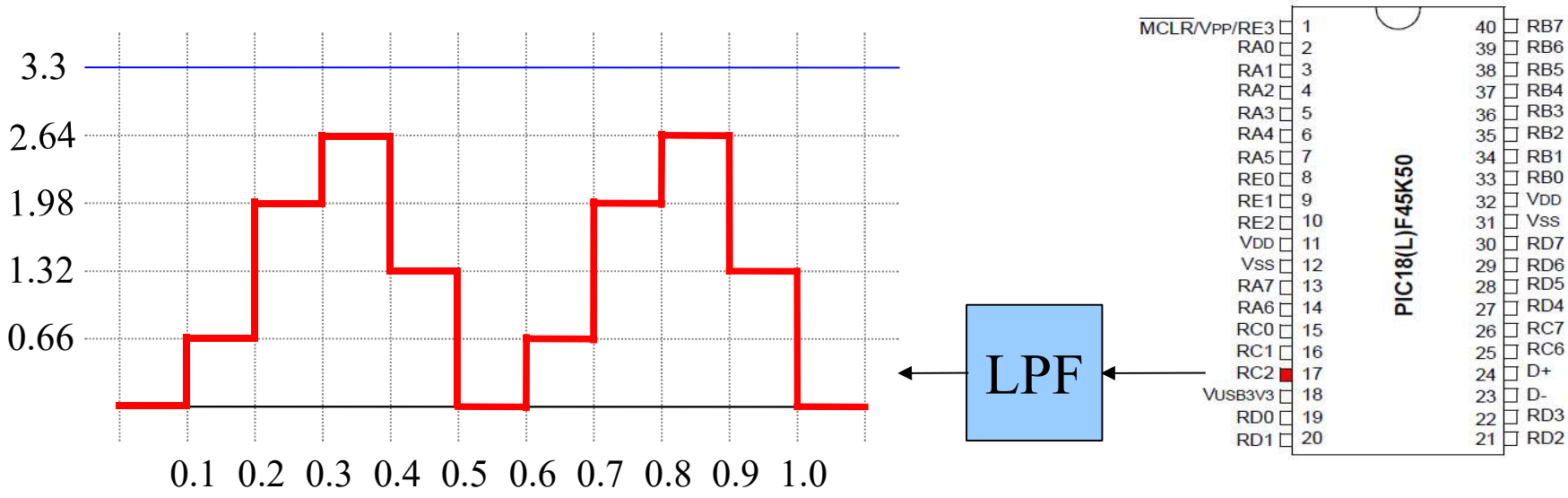
```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Main program
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
main(){
int_ports();            //Call init oscillator
//Ourput for RC2 since is the PWM I/O for  CCP1
CCP1CON = 0b00001100;   //Set PWM mode, remember bits 5 y 4 are LSB of
                        //of the duty cycle
//Init the CCP module using TIMER 2
CCPTMRS = 0x00;
//Configure the PR that defines the frequency or carrier (period)
PR2 = 99;
//Configure register CPPR1L they contay the rest of the duty cycle
//perdiod, remember that CCP1CON contais the last two (LSB)
CCPR1L = 0x28;

T2CON = 0b00000101;         //Prescale 1 Postcale 1 and turn on timer
while(1);                   //Stay here doing nothing

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that inits the internal oscillator at 16Mhz
//+ and the ports
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void init_ports(void){
    //RC2 is input for CCP1
    TRISCbits.TRISC2 = 1;   //Input
    ANSELCbits.ANSC2 = 0;   //Digital
    //Set the internal oscillator frequency to 16Mhz
    OSCCON = 0b01111100;
}
```

Example:

- Using the following circuit, generate the shown waveform using PWM with a carrier of 10Khz. The PIC oscillator is 16Mhz



Solution Alternative

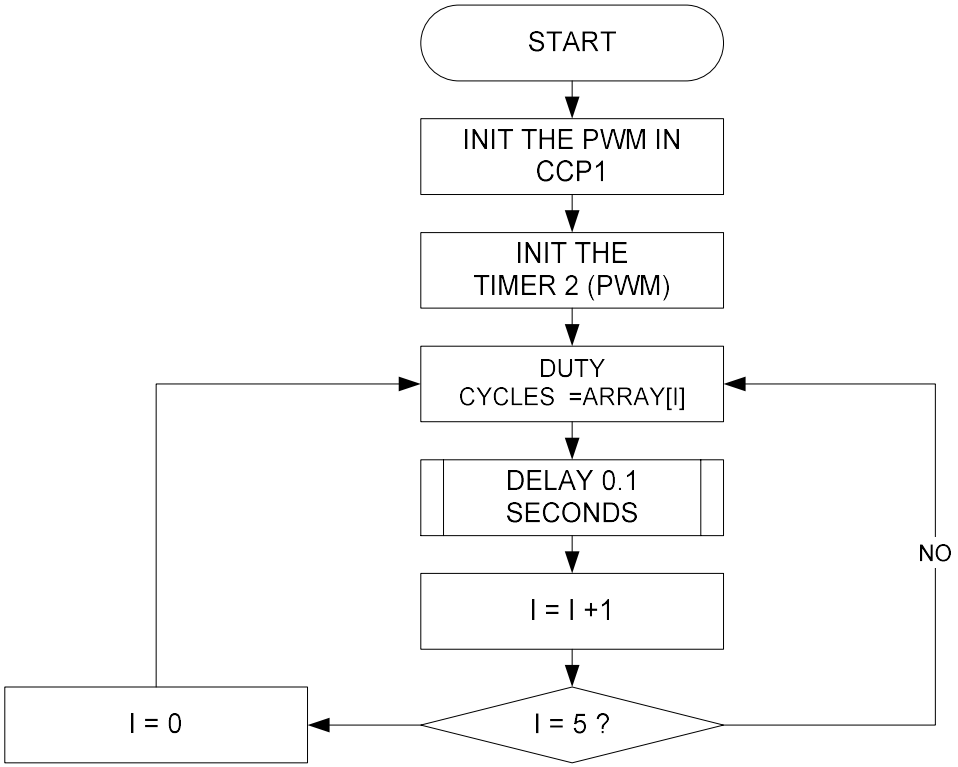
- Changes in the amplitude happens at 0.1 seconds, we will use TIMER0 as a time base for these changes.
- From the amplitude values and since VCC = 3.3V we will need to have the following duty cycles for each step

T	V	D
0	0	0.00/3.3 = 0%
0.1	0.66	0.66/3.3 = 20%
0.2	1.98	1.98/3.3 = 60%
0.3	2.64	2.64/3.3 = 80%
0.4	1.32	1.32/3.3 = 40%

- We will use the previous example to generate the duty cycles.

Solution Alternative

- We will store the required values for the duty cycles in an array of 5 elements (one for each step of the waveform).



PWM period calculations

- Use TIMER2 as the time base for module CPP1, timer pre-scale = 4
- To calculate the period we have:

$$PWM\ Period = [(PRx) + 1] \bullet 4 \bullet Tosc \bullet (TMRx\ Prescale\ Value)$$

Tosc = 1/Fosc

$$1/10Khz = [\ PR2 + 1] * 4*(1/16e6) * 4$$

$$PR2 = 99$$

Solution Alternative

- CCP value based on the waveform simple amplitude:

$$Duty\ Cycle\ Ratio = \frac{(CCPRxL:CCPxCON<5:4>)}{4(PRx+1)}$$

99

V	D(%)	CCPR1L:CCP1CON<5:4>	CCPR1L:CCP1CON<5:4>	CCPR1L	CCP1CON<5:4>
0	0	0	0x000	0X00	0b00
0.66	20	80	0x050	0x14	0b00
1.98	60	240	0x0F0	0X3C	0b00
2.64	80	320	0X140	0X50	0b00
1.32	40	160	0X0A0	0X28	0b00

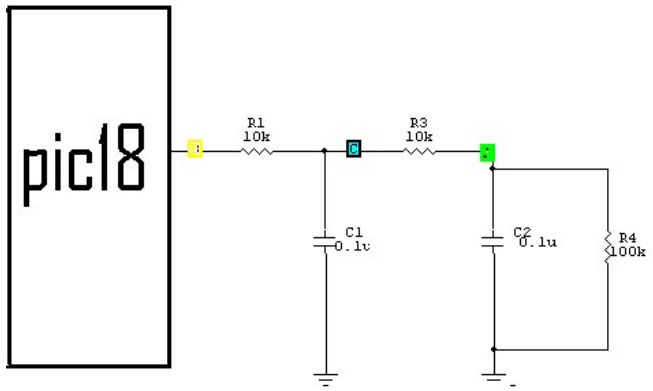
```
unsigned char D_CYCLE[5] = {0X00,0x14,0X3C,0X50,0X28};
```

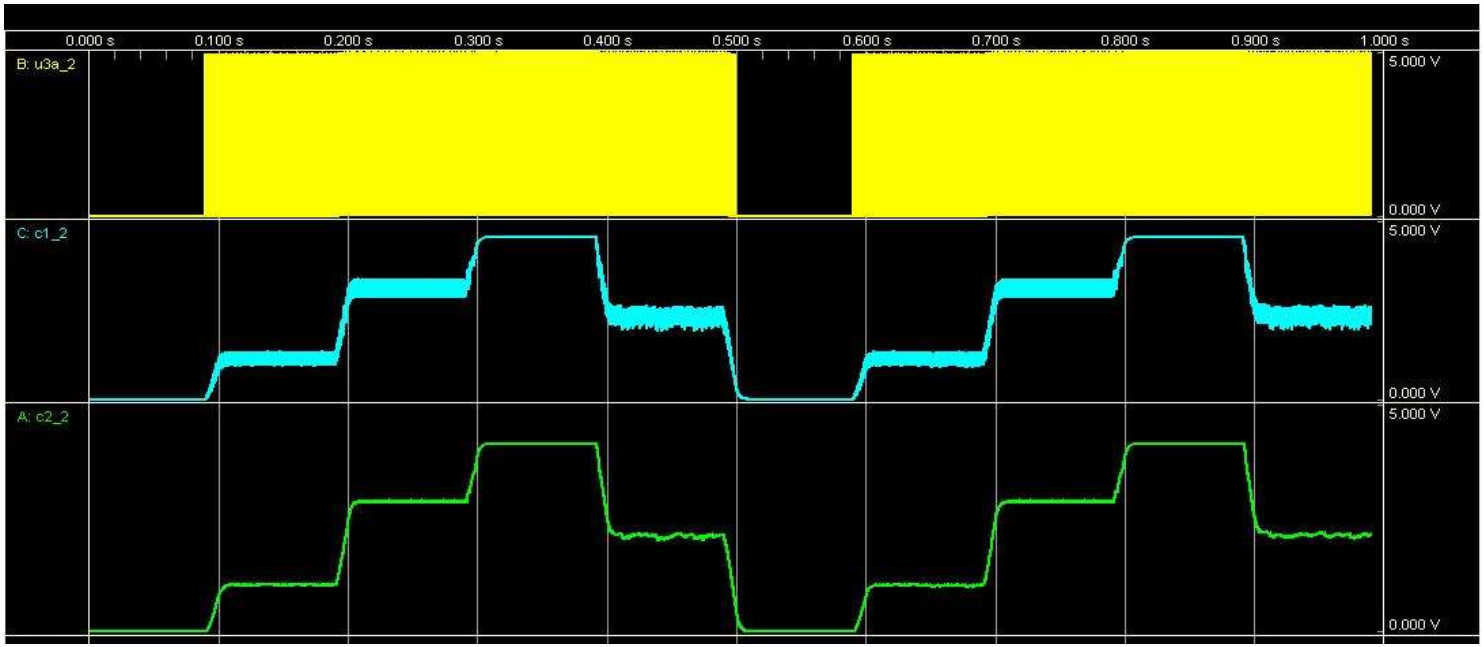
```
#include<xc.h>
void init_ports(void);                    //Inits ports and oscillator
void delay_100ms(void);                   //Function generates a delay
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Main program
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

main(){
unsigned char i;                 //Used as array index
unsigned char D_CYCLE[5] = {0x00,0x14,0x3C,0x50,0x28};  //Duty cycles array
init_ports();                    //Inits port and oscillaotr
CCP1CON = 0b00001100;           //PWM mode, remember bits 4 and 5 are LSB for duty
                                 //cycle
//Indicate the CCP1 to use TIMER 2
CCPTMRS = 0x00;
//Confiture the PR2 register to set the carrier frequenc
PR2 = 99;
T2CON = 0b00000101;       //Prescale 4 Postcale 1 and sart timer
while(1){
    for(i=0;i<5;i++){
        delay_100ms();           //Wait the 100ms
        CCPR1L = D_CYCLE[i];     //Chancy cycle using next element in array
    };
}; //From while
} //From main
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that generats a 100msec delay using timer 0
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void delay_100ms(void){
   //Count 40536 or 0x9E58 ticks
    TMR0H = 0x9E;           //High byte 0x9E58
    TMR0L = 0x58;           //Low bte de 0x9E58
    INTCONbits.TMR0IF = 0;  //Clear the timer overflow flag
    //Configure the timer
    //16 bits
    //Set a 16 pre-scaler
    T0CON = 0b10000011;
    while(INTCONbits.TMR0IF == 0); //Wait for overflow
    T0CON = 0X00;                  //Stop the timer
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that inits the ports used and oscilator
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void init_ports(void){

    //Init  RC2 (CPP1) as output
    TRISCbits.TRISC2 = 0;   //Ouput
    ANSELCbits.ANSC2 = 0;   //Digital
    OSCCON = 0b01111110;    //Set the board oscillator to 16Mhz
}
```


pic18
R1
10k
R3
10k
C1
0.1u
C2
0.1u
R4
100k


0.000 s
0.100 s
0.200 s
0.300 s
0.400 s
0.500 s
0.600 s
0.700 s
0.800 s
0.900 s
1.000 s
B: u3a_2
C: c1_2
A: c2_2
5.000 V
0.000 V