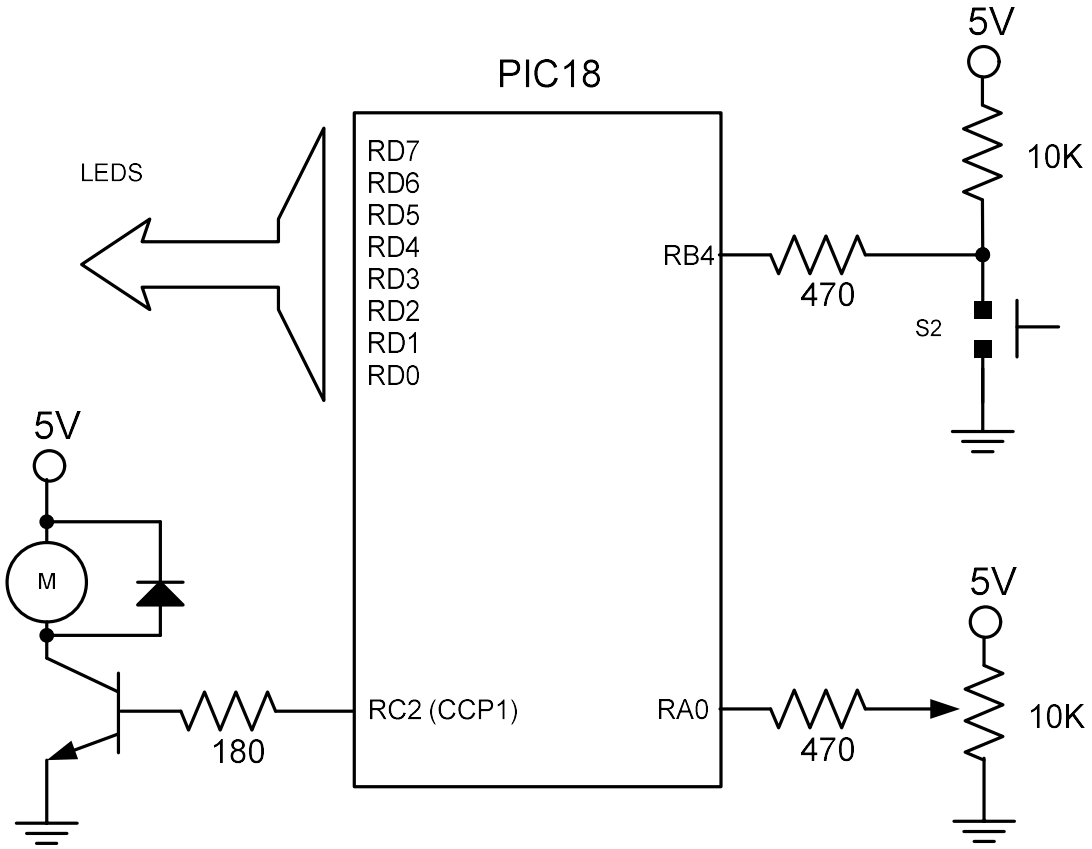
TECNOLÓGICO DE MONTERREY®

Subject 14
TIMER-ADC-PWM EXAMPLE

Problem description

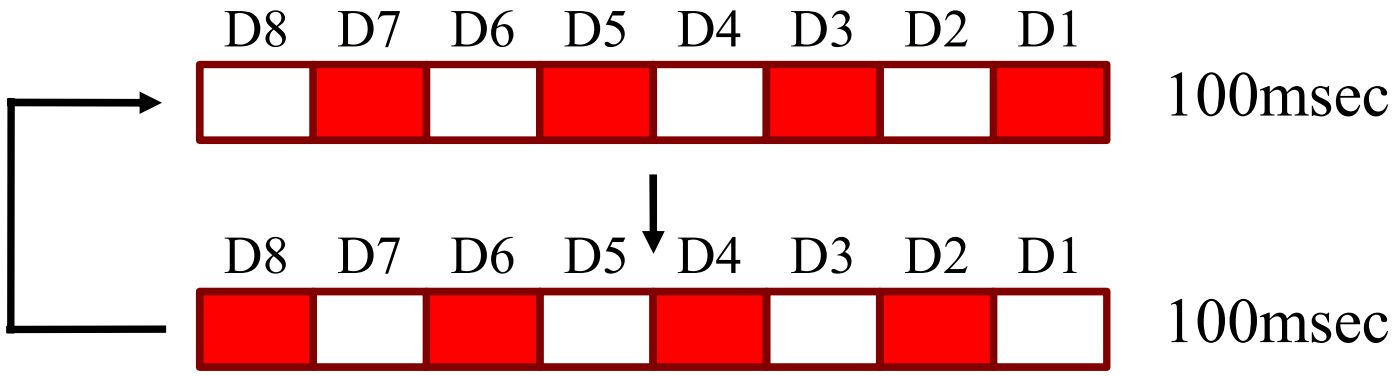
- We want to make an open loop control for a DC motor.
- To define the speed, we will use a potentiometer connected to the A/D converter
- The speed of the motor is controlled using PWM
- The user interface is a button and 8 LED's

Circuit Schematic



System operation

- When the system is turned on, the stand-by mode will be executed, this basically stops the motor and displays the following LED sequence:



System Operation

- When pressing button S2, the systems ends the stand-by mode and starts the control mode.
- The process that captures the button must provide de-bounce filtering and the action is considered to be valid when the button is de-pressed.

System Operation

- For the control mode, the value of the potentiometer is sampled at a 50msec and 8 bits resolution is enough for the application.
- The digitized voltage value modifies the LED to show the magnitude value with the following relation:

D8	D7	D6	D5	D4	D3	D2	D1
>224	>196	>168	>140	>112	>84	>56	>28

System Operation

- The analog value of the potentiometer will be used to modify the duty cycle of the PWM signal that controls the motor speed.
- We will use CCP1 module for the PWM generation, this mode uses Timer 2.
- Since he switching transistor is kind of slow, a low carrier frequency for the PWM is recommended. We will use 4Khz.

System Operation

- If the button SW1 is pressed during the control mode, the program muts return to the stand-by mode
- The program must operate using a time base of 1msec, we will also use Timer 2 to generate a precise value.
- The microcontroller will use the internal oscillator at 16Mhz.

Inputs and outputs

- Port D, outputs assigned to the LED's
- Pin RC2 is assigned to CCP1 that is the PWM output
- The switch is assigned to RB4 so it will defined as the input
- The potentiometer is assigned to RA0 that is the analog input AN0.

Input and outpus

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Init the ports
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void init_io(void){
    //LED outputs
    TRISD = 0b00000000;          // Output
    ANSELD = 0b00000000;         // Digital
    PORTD = 0b00000000;          // Turn off all the LED's
    //Output for the PWM to control the motor
    TRISCbits.TRISC2 = 0;        // Output
    ANSELCbits.ANSC2 = 0;        // Digital
    PORTCbits.RC2 = 0;           // Muto off
    //Button S1 in RB4,
    TRISBbits.TRISB4 = 1;        //Input
    ANSELBbits.ANSB4 = 0;        //Digital
    //Analog input AN0 (RA0)
    TRISAbits.RA0 = 1;           //Input
    ANSELAbits.ANSA0 = 1;        //Analog
```

Time base

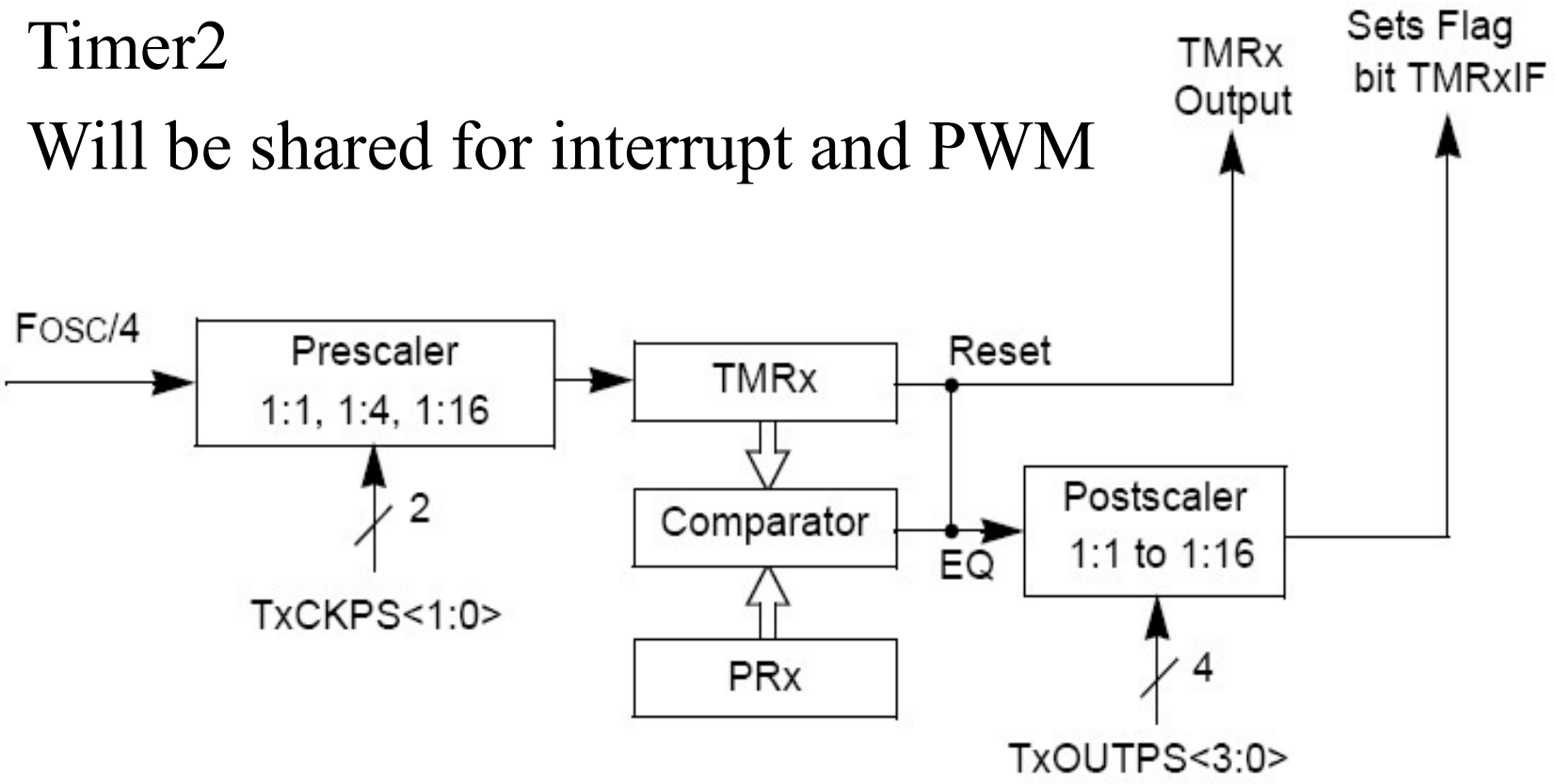
- We need stable time base controlled by the timer
- We will generate a periodict interrupt each 1 msec.
- The oscillator frequency is FOSC = 16Mhz
- We will use timer 2 since it is specialized in generating periodic interrupts with out reloading

Time base

- In this microcontroller we only have one “even” timer, and it is required for the PWM
- We can share the same time base for the PWM and the interrupt generation since they are in the same order of magnitude.
- 4Khz for PWM and 1Khz of sampling
- We can also use other timers (0,1,3)

Time base

- Timer2
- Will be shared for interrupt and PWM

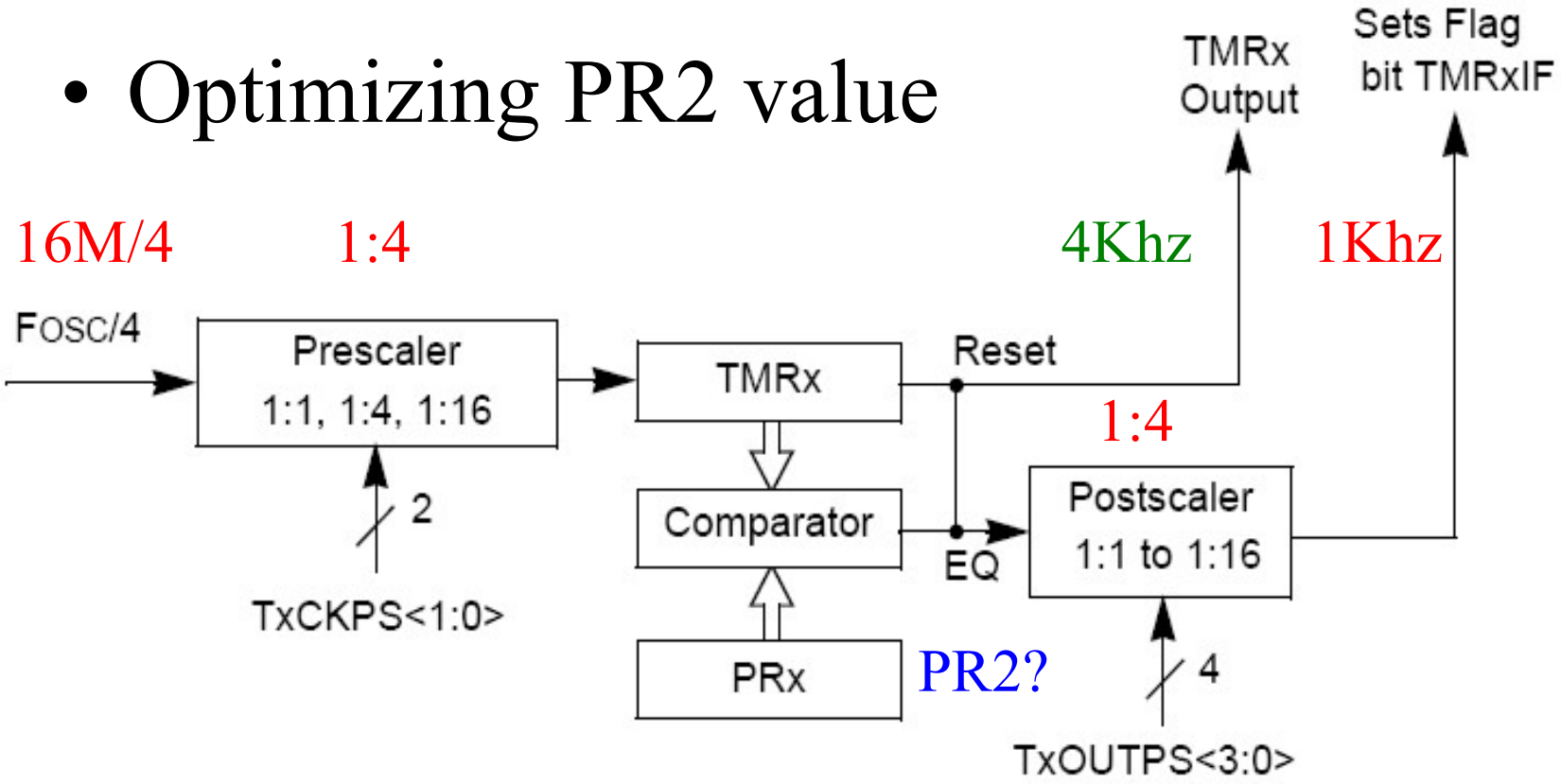


$$Resolution = \frac{\log[4(PR2+1)]}{\log(2)} \ bits$$

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

Time base

- Optimizing PR2 value



$16x10^6/4 = 4 \times (PR4+1) \times 4000$

PR4 =249

Time base

- For a PR2 with a value of 249, the interrupt period will be :

$$F_{TMR2IF} = (16x10^6/4)/(4*(249+1)*4)$$
$$F_{TMR2IF} = 1000Hz$$
$$T_{TMR2IF} = 1msec$$

Time base

- Timer 2 will be assigned to an interrupt of high priority
- The ISR we will place the code of the state machines to perform the following actions:
  - The A/D conversion
  - Reading the button (using a polled scheme)
  - Execute the visual UI

Time base

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Init the time base using timer 2 for interrupt and PWM generation
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void init_time_base(void){
    RCONbits.IPEN = 1;       //Enable priority
    PIE1bits.TMR2IE = 1;     //Enable the interrupt for TIMER2
    IPR1bits.TMR2IP = 1;     //Set the priority to high (because I say so)
    PIR1bits.TMR2IF = 0;     //We clear the interrupt flag
    INTCONbits.GIEH = 1;     //General enable the high priority interrupts
    INTCONbits.GIEL = 0;     //No hay ninguna asignada por el momento
    PR2 = 249;               //Generates 4Khz for the PWM carrier
    T2CON = 0b00011101;      //Post 1:4,Tmr ON, Pres 1:4
}
```

Time base IRQ processing

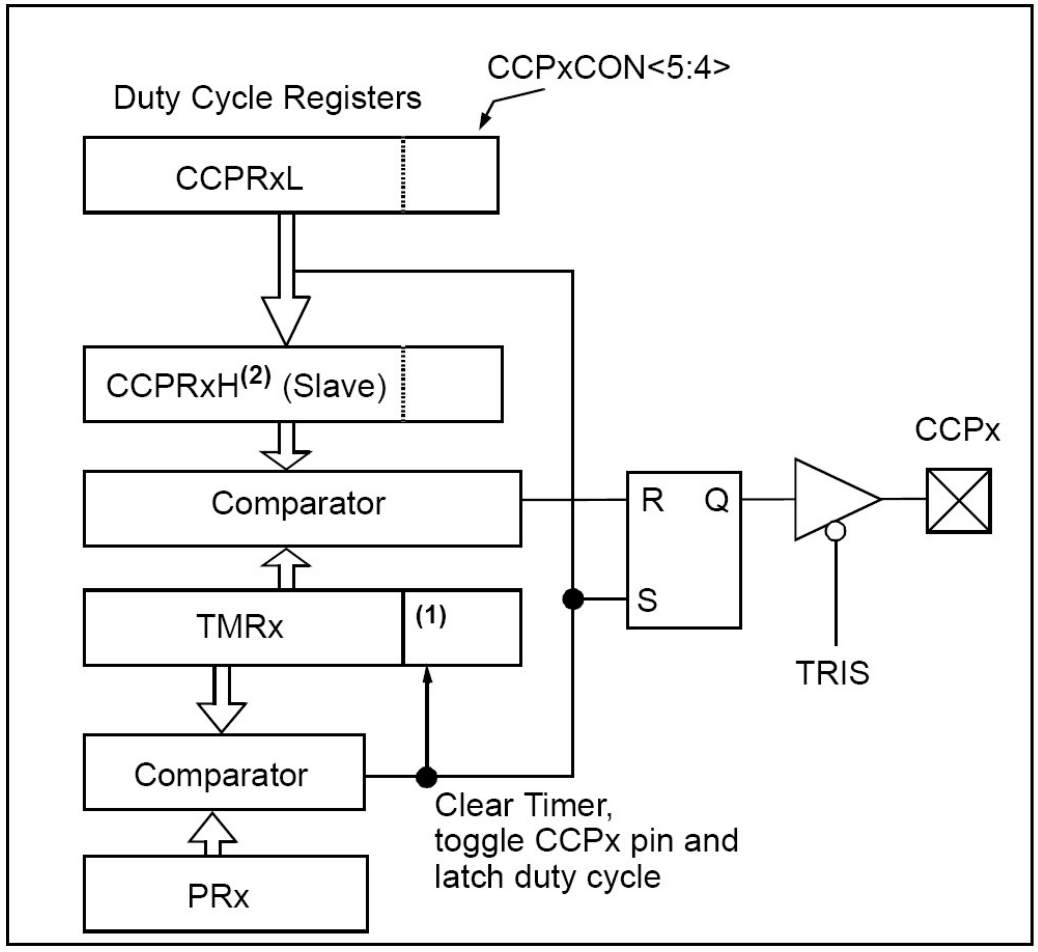
```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ High interrupt to TIMER2
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void __interrupt (high_priority)  high_priority_ISR(void){
static unsigned char i;
        convert_adc();          //Convert the ADC
        sw1_scan();             //Scan the button
        stand_by();             //Rutina standby
        PIR1bits.TMR2IF = 0;    //Turn off

 }
```

PWM

- We will use CCP1 for PWM using Timer2, the carrier frequency must be 4Kz (Fosc = 16Mhz) or as close as possible.

- We require an 8 bit resolution so the two least significant bits will be discarded.

PWM


Duty Cycle Registers
CCPxCON<5:4>
CCPRxL
CCPRxH(2) (Slave)
Comparator
TMRx
(1)
Comparator
PRx
R
Q
S
CCPx
TRIS
Clear Timer,
toggle CCPx pin and
latch duty cycle

PWM

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Inicializacion del PWM usando CCP1 con TIMER2
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void init_pwm(void){

 //Configuracion de PWM, la portadora estara a 4khz (ver init_time_base())

        CCP1CON = 0b00001100;    //Modo PWM, ojo bits 5 y 4 son los menos
        CCPTMRS = 0x00;          //Utilizar el Timer2
        CCPR1L = 0;              //Valor inicial
}
```

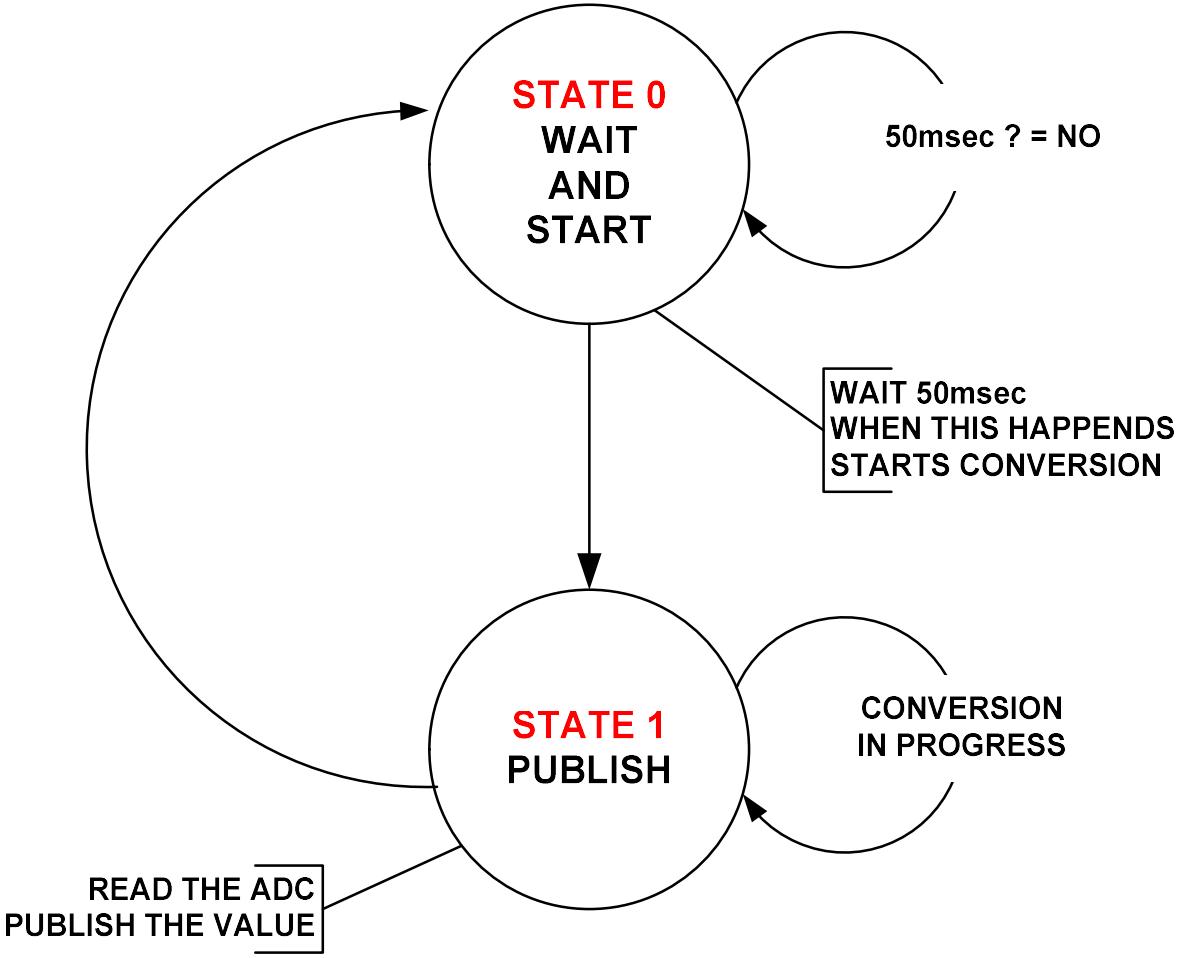
ADC

- For the analog conversion, the potentiometer outputs a voltage between 0V and VCC
- We will use an 8 bit resolution so the two least significant bits are also discarded.
- Since the potentiometer is resistive and uses a relative high value, use the longest conversion and acquisition time possible.

ADC

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Init the ADC
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void init_adc(void){

    //Concigure the  A/D
    ADCON0 = 0b00000001;     //Select channel AN0 trn on the ADC
    ADCON1 = 0b00000000;     //Vref+ (VCC), Vref-(GND)
    ADCON2 = 0b00111110;     //Right justificatin, 20TAD,TAD= FOSC/64
}
```

ADC


STATE 0
WAIT
AND
START
50msec ? = NO
WAIT 50msec
WHEN THIS HAPPENDS
STARTS CONVERSION
STATE 1
PUBLISH
CONVERSION
IN PROGRESS
READ THE ADC
PUBLISH THE VALUE

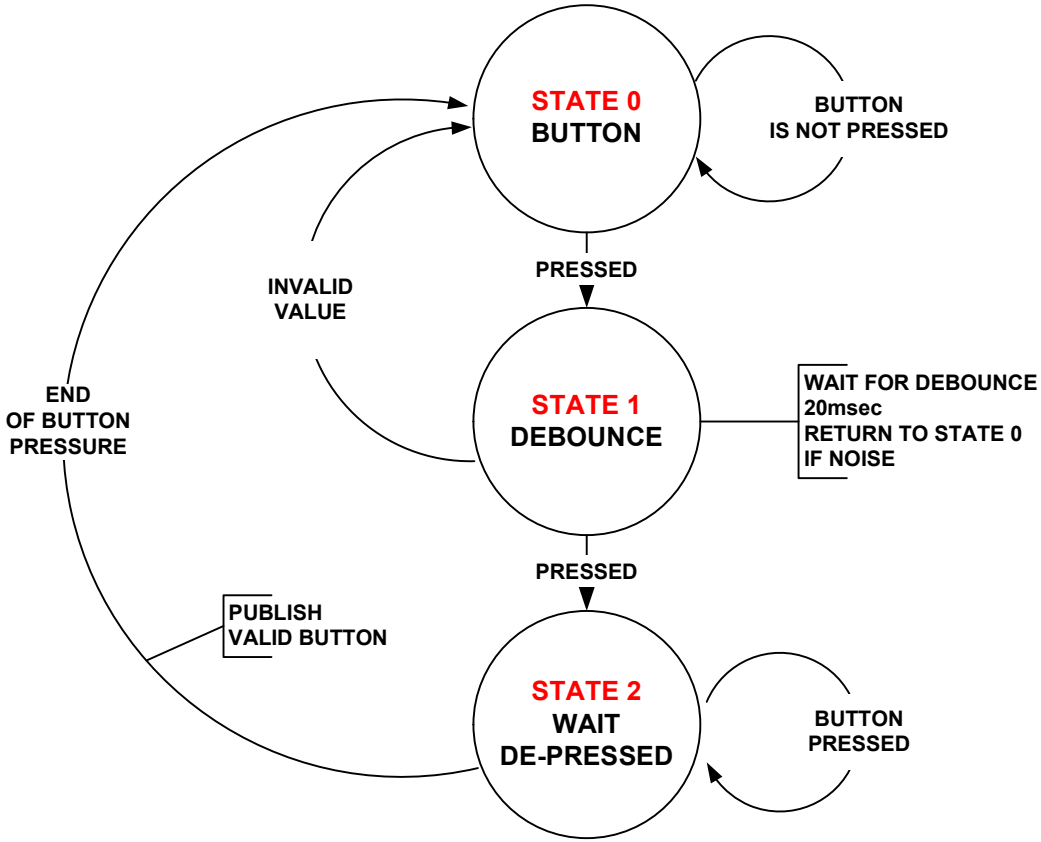
ADC state machine

- This function is called from the ISR each1msec

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that reads the ADC each 50msec
//+ must be called with a periodicity of 1msec
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void convert_adc(void){
static unsigned char state = 0;
static unsigned char i = 0;
    i++;
    switch(state){
        case 0:
                    if(i!=50) break;          //50 mesc ?
                    i = 0;              //si
                    ADCON0bits.DONE = 1;     //Start the conversion
                    state = 1;               //Next state
                    break;
        case 1:
                    if(ADCON0bits.DONE) break;  //Finish conversion ?
                    value_adc = ADRESH;         //We will use 8 bits
                    state = 0;                  //Next state
                    break;
    }
}
```

Button sampling state machine

- Called from the ISR each 1msec



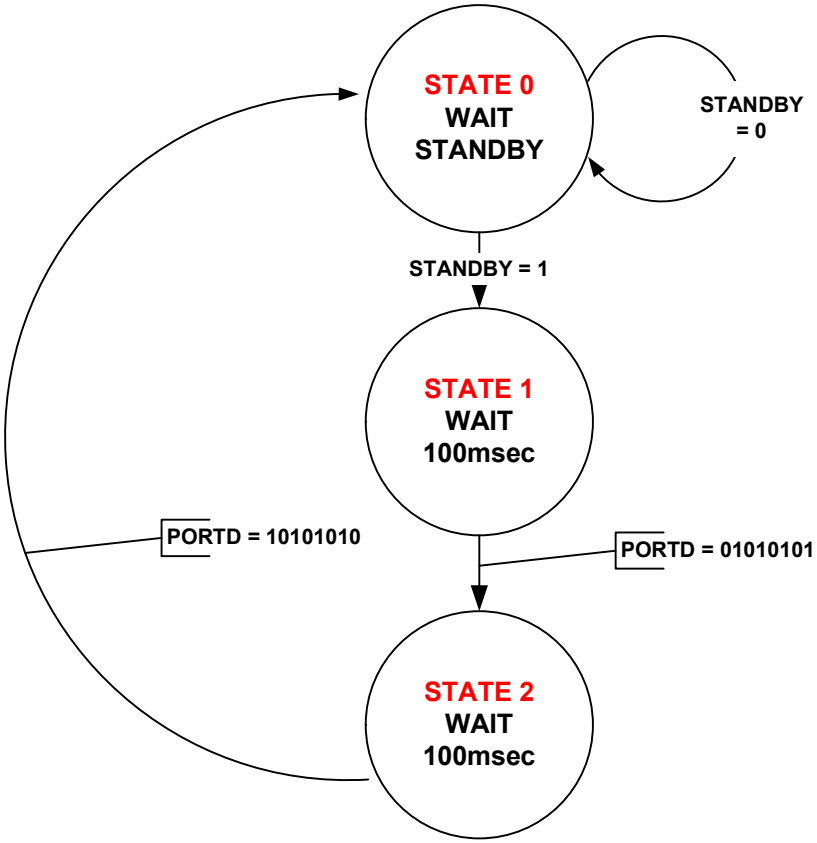
Button sampling

- Called from the ISR each 1msec

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Funtion that captures SW1
//+ It provides de-bouce
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void sw1_scan(void){
static unsigned char state = 0;
static unsigned char debounce = 0;

    switch(state){
        case 0:
            if(PORTBbits.RB4 == 1) break;//Button pressed ?
            debounce = 0;
            state = 1;
            break;
        case 1:
            debounce ++;
            if(debounce != 20) break;   //End of delay 20 x 1 msec = 20msec ?
            if(PORTBbits.RB4 == 1){     //If not it was a gith
            state = 0;
            break;
            }
            state = 2;
             break;
        case 2:
             if(PORTBbits.RB4 == 0) break;//Wait to de-press the button
             button = 1;
             state = 0;
             break;
    }
}
```

Stand-by

- This function is called at 1msec rate and uses a global variable to control its operation



Stand by function

- Called at 1msec rate

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that displays the
//+ has as input the variable stand_by_on
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void stand_by(void){
static unsigned char state = 0;
static unsigned int delay = 0;
    delay++;
    switch(state){
        case 0:
                    if(standby_on == 0) break;
                    delay = 0;
                    state = 1;
                    break;
        case 1:
                    if(delay !=100) break;  //Wait 100ms
                    PORTD = 0b01010101;
                    state = 2;
                    delay = 0;
                    break;
        case 2:
                    if(delay !=100) break;  //Wait 100ms
                    PORTD = 0b10101010;
                    state = 0;
                    break;
    }
}
```

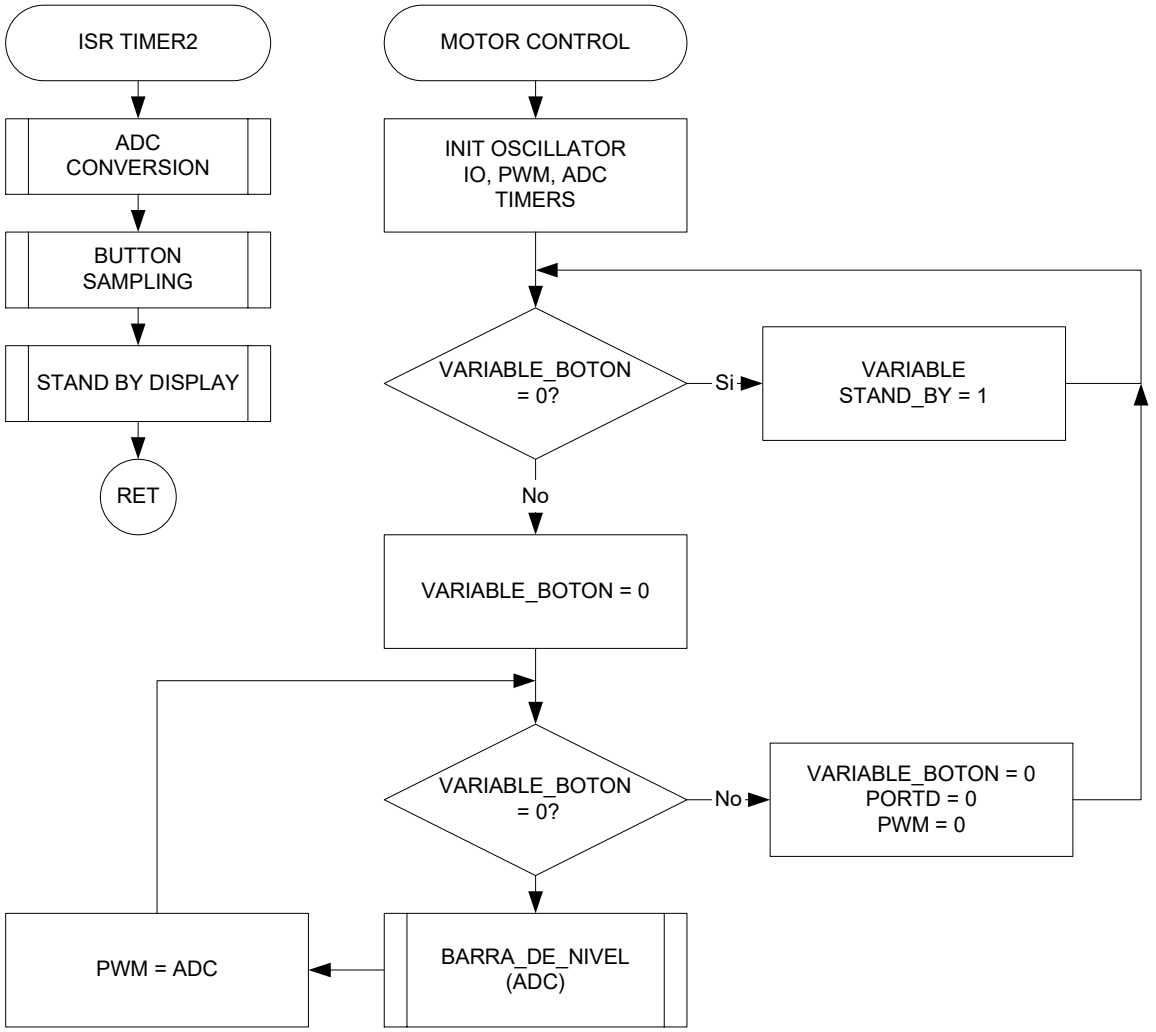
Level barr

- This function is called from the main program

```
void level_barr(unsigned char level){
    if( level > 28) PORTDbits.RD0 = 1; else PORTDbits.RD0 = 0;
    if( level > 56) PORTDbits.RD1 = 1; else PORTDbits.RD1 = 0;
    if( level > 84) PORTDbits.RD2 = 1; else PORTDbits.RD2 = 0;
    if( level > 112) PORTDbits.RD3 = 1; else PORTDbits.RD3 = 0;
    if( level > 140) PORTDbits.RD4 = 1; else PORTDbits.RD4 = 0;
    if( level > 168) PORTDbits.RD5 = 1; else PORTDbits.RD5 = 0;
    if( level > 196) PORTDbits.RD6 = 1; else PORTDbits.RD6 = 0;
    if( level > 224) PORTDbits.RD7 = 1; else PORTDbits.RD7 = 0;
}
```

D8	D7	D6	D5	D4	D3	D2	D1
>224	>196	>168	>140	>112	>84	>56	>28

Main program



Main Program

```
#include<xc.h>

#define _XTAL_FREQ 16000000          //Required for coded delay

void init_io(void);                  //Init the ports
void init_time_base(void);           //Init time base
void init_pwm(void);                 //Init  PWM
void init_adc(void);                 //Init  ADC
void convert_adc(void);              //Rutina de acceso al convertidor
void sw1_scan(void);                 //Rutina de captura interruptor 1
void level_barr(unsigned char);      //Despliega barra de nivel
void stand_by(void);                 //Rutina de despligue standby
void high_priority_ISR(void);        //Interrupcion prioridad alta
unsigned char value_adc;             //Almacena valor de conversion
unsigned char button;                //Almacena estado del boton
unsigned char standby_on;            //Enciende o apaga proceso de stand-by
```

Main program

```
main(){

    OSCCON = 0b01111110;     //Set the board oscillator to 16Mhz
    button = 0;              //Valor del boton = 0;
    __delay_ms(20);          //Delay
    init_io();               //Init ports
    init_pwm();              //Init the PWM
    init_time_base();        //Init the time base
    init_adc();              //Init de ADC

    //Control Loop
    while(1){

        while(button == 0) standby_on = 1; //If buttonstand-by
        button = 0;                 //Clear the variable

        while(!button){
            level_barr(value_adc);  //Set the barr level
            CCPR1L = value_adc;     //Set the PWM value
        }
        button = 0;  //Clear the butto state
        CCPR1L = 0; //We make 0 the PWM (Motor is off)
        PORTD = 0;  //Turn off all the LED'S
    }//From while(1)
} //from main() TEMA_14_PIC_1.C
```


IS THIS
THE END
?