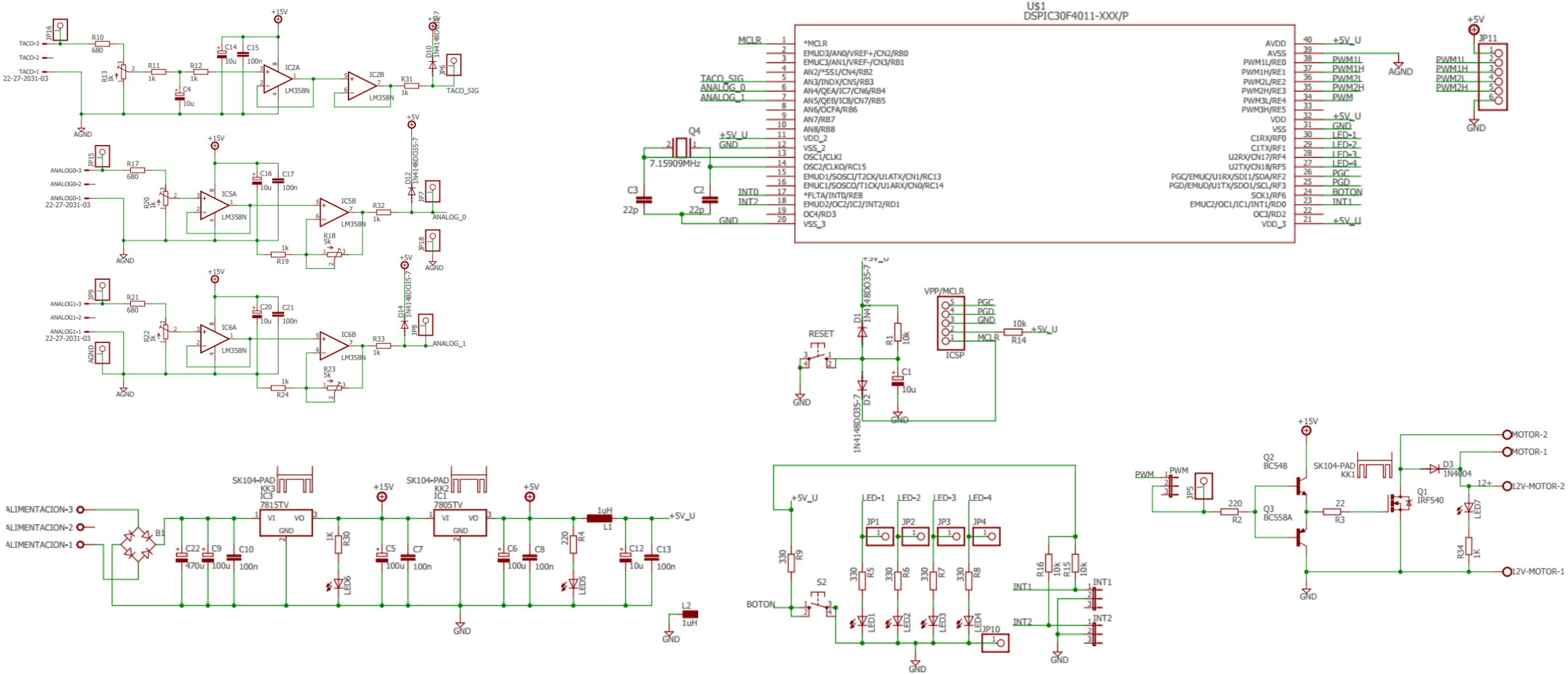
CONTROLADOR DIGITAL DE SEÑALES DSPIC30F4011

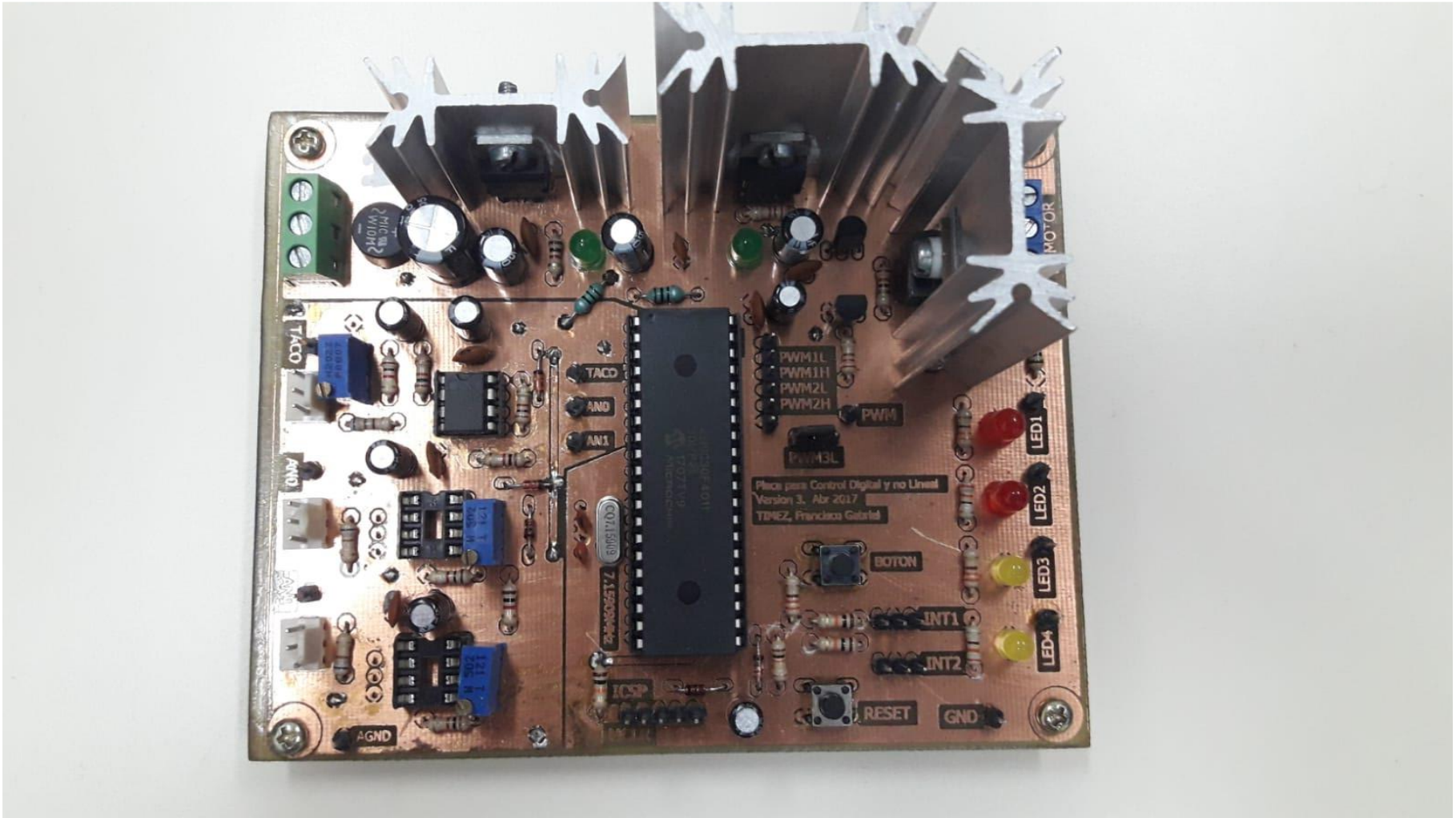
Sistemas de Control 2

2024

ESQUEMA ELÉCTRICO DEL DSPIC



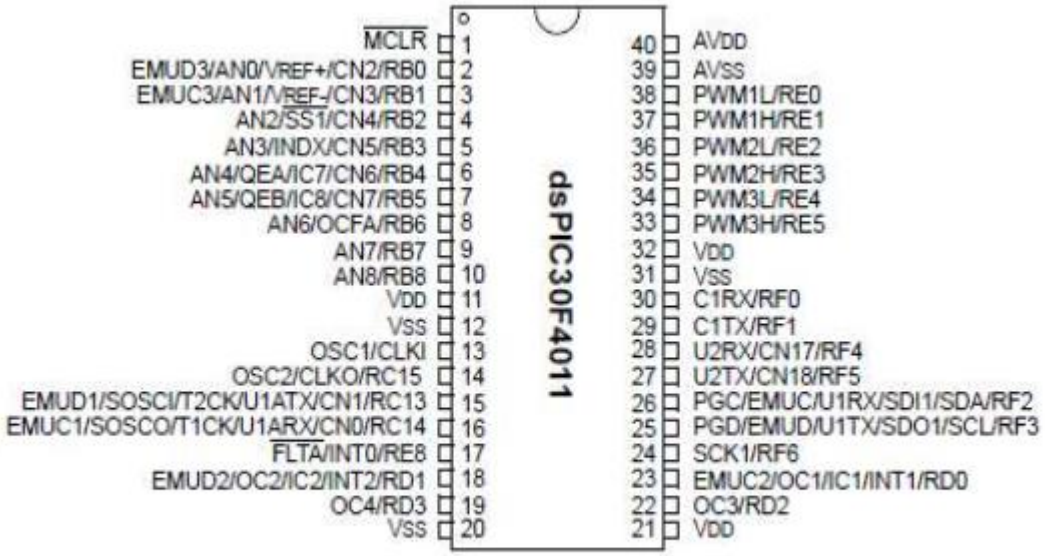
VISTA SUPERIOR DE LA PLACA DEL DSPIC



DSPIC30F4011

Características destacables:

- ❖ Cristal externo de 7,15909MHz
- ❖ Clock de 1,78MHz.
- ❖ Datos de 16 bits.
- ❖ 8 KB de RAM
- ❖ 144 KB de memoria flash
- ❖ Aritmética de punto fijo en hardware.
- ❖ ADC de 10 bits con 16 canales hasta 0,5 Msps
- ❖ Hasta 6 canales PWM, con salidas complementarias
- ❖ 40 pines
- ❖ Voltaje de operación 0 a 5V
- ❖ Alimentación de 5V
- ❖ Módulos de comunicación serial.
- ❖



ESQUEMA DE PINES

MAIN loop

```
// ------------------------------------------------------------------------------
// FUNCIÓN PRINCIPAL
// ------------------------------------------------------------------------------
int main (void)
{
  Configura_PUERTOS();  // Configura todo el puerto D como salida, utiliza RD2 como "testigo".
  Configura_ADC();      // Configura ADC para efectuar conversión sólo por AN0/RB0. Cada
                        // vez que finaliza la conversión, interrumpe.
  Configura_PWM();      // Configura PWMmotor para proporcionar una salida con Fpwm = 1kHz
                        // por la salida PWM1L/RE0. También se activa el disparo de un
                        // evento especial, que en este caso corresponde al inicio de la
                        // conversión en cada periodo PWM (postcaler 1:1) cada vez que
                        // PTMR = 0x00FF.
  LED1 = OFF;           // Apaga L1.
  LED2 = OFF;           // Apaga L2.
  LED3 = OFF;           // Apaga L3.
  LED4 = OFF;           // Apaga L4.
  Referencia = REF0;    // Referencia en cero.
  while(1);             // Bucle infinito para esperar interrupciones.
}
// ------------------------------------------------------------------------------
```

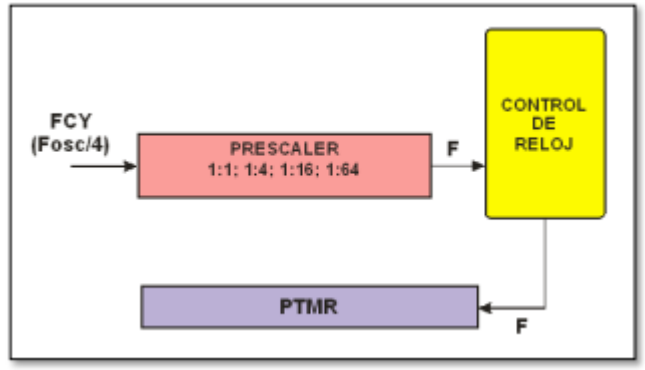
INTERRUPCIÓN DEL ADC

```
// ------------------------------------------------------------------------------------
// Rutina de Servicio a la Interrupción por final de conversión AD
// ------------------------------------------------------------------------------------
void __attribute__((interrupt, auto_psv)) _ADCInterrupt(void)
{
counter_1++;
if(counter_1 == N_Tpwm)
{ LED1 = ON;                // Cada vez que finaliza la conversión, RD0 en alto (apaga L1).
  convertido = ADCBUF0;     // Almacena resultado de la conversión en "convertido".
                            // La conversión AD corresponde a la medición de la velocidad.
  Velocidad = MulFix12(convertido,KbQ12);   // Normaliza la conversión y transforma a Q12
                                            // la velocidad.
// Variación de referencia en rampa ascendente
  counter_ref1++;
  counter_ref2++;
  if(Referencia < REF50)                                  // Incrementa referencia, sino
    { Referencia = Referencia + FloatToFix12(delta_Ref);} // todavía no alcanzó el 50%.
  else
    { Referencia = REF50;                                 // Referencia al 95%.
      LED2 = ON; }                                        // Enciende L2 para indicar que
                                                          // se alcanzó el 95% de la ref.
   if(counter_ref2 >= 500)      // Si el contador alcanzó t = 375*N_Tpwm*0.001s = 6s, la ref.
    { Referencia = REF99;}      // baja al 95%.
// Cálculo del error actual
  error_k = Referencia - Velocidad;
// Cálculo de las acciones de control

// Acción PI, aprox. Forward: upi(k) = upi(k-1) + Kpi*e(k) + a*Kpi*e(k-1)
  upi_k = AddFix(AddFix(MulFix12(error_k,Kpi),MulFix12(error_k1,aKpi)), upi_k1);
```

```
// Valores anteriores para el proximo periodo de muestreo:
  error_k1 = error_k;
  upi_k1 = upi_k;
  Velocidad_k1 = Velocidad;


// Aplica acción de control
uk = upi_k;      // PI
// Cálculo del ciclo util a partir de la acción de control
// ------------------------------------------------------------------------------------
// Almacena el valor en PDC3, registro ciclo útil del módulo PWM
// duty cycle = (PDCx/2)/(PTPER+1)
// MulFix12 multiplica a uk en formato Q12 por (PTPER+1)
// si uk=1pu=4096 entonces MulFix12 = (PTPER+1)
  PDC3 = MulFix12(uk,(PeriodoPWM + 1)) << 1; //corrimiento << 1 equivale a multiplicar por 2
// Finaliza el procesamiento del controlador:
  LED1 = OFF;    // Pone pin RD0 en bajo para indicar finalización del cálculo (enciende L1).
// Almacena en buffer
  if (ContadorBuffer < TamBuffer)
     {
       Var_buffer1[ContadorBuffer] = Velocidad;
       ContadorBuffer ++;}
  else
     { LED3 = ON;}          // Enciende el led LED3 para indicar que se llenaron los buffers.
  counter_1 = 0;            // Borra el contador de muestreo, para utilizarlo nuevamente.
}
  IFS0bits.ADIF = 0x00;     // Resetea el flag de interrupcion del AD.
}
// FIN DE LA RUTINA DE INTERRUPCIÓN ------------------------------------------------
```

FRECUENCIA DEL PWM

```
// Configuración módulo PWM_MOTOR
// PWM Period Calculation for Free Running Count Mode (up counting mode) PTMOD = 00
#define PeriodoPWM 28635      // Periodo del PWM calculado con: PTPER = [Fcy/(Fpwm*Prescaler)]-1
                              // Datos: Fcy = 7.15909MHz*16/4 = 28.63636 MHz; Fpwm = 1000Hz; Prescaler = 1
// Duty Cycle Calculation for Free Running: Duty Cycle = (PDCx/2)/(PTPER+1)
// Para Duty Cycle 100% el valor de PDCx es 2x(PTPER+1)=57272; Para Duty Cycle de 50% PDCx=PTPER+1=28636
void Configura_PWM(void)
{PTPERbits.PTPER = PeriodoPWM;    // Primero carga el periodo PTPER = PeriodoPWM
// Registro de control PWMCON1:
PWMCON1bits.PEN1L = 0x00;         // Pin PWM 1, parte baja: 1 = habilitado, 0 = deshabilitado
PWMCON1bits.PEN2L = 0x00;         // Pin PWM 2, parte baja: 1 = habilitado, 0 = deshabilitado
PWMCON1bits.PEN3L = 0x01;         // Pin PWM 3, parte baja: 1 = habilitado, 0 = deshabilitado
PWMCON1bits.PMOD1 = 0x01;         // 1 = modo independiente, 0 = modo complementado
PWMCON1bits.PMOD2 = 0x01;         // 1 = modo independiente, 0 = modo complementado
PWMCON1bits.PMOD3 = 0x01;         // 1 = modo independiente, 0 = modo complementado
// Registro de control PWMCON2:
PWMCON2bits.UDIS    = 0x00;       // Actualizacion de registros: 0 = habilitado, 1 = deshabilitado
PWMCON2bits.OSYNC   = 0x00;       // Sincronizacion del Output-Override
PWMCON2bits.IUE     = 0x00;       // Actualizacion inmediata del PDC: 1 = habilitada, 0 = deshabilitado
PWMCON2bits.SEVOPS  = 0x00;       // Postscaler del Trigger del generador de eventos especiales (1:1)
// Registro de control DTCON1 (Tiempo Muerto):
DTCON1bits.DTA    = 0x00;         // Valor del tiempo muerto
DTCON1bits.DTAPS  = 0x00;         // Prescaler para el tiempo muerto
// Registro de configuracion PTCON (Base de Tiempo):
PTCONbits.PTMOD   = 0x00;         // Bits de configuracion de la base de tiempo en free running mode.
PTCONbits.PTCKPS  = 0x00;         // Prescaler de entrada para la base de tiempo
PTCONbits.PTOPS   = 0x05;         // Postscaler de salida
PTCONbits.PTSIDL  = 0x00;         // Si la base de tiempo funciona en "Idle Mode"
PTCONbits.PTEN    = 0x01;         // Habilitacion de la base de tiempo
// Registro de configuracion SEVTCMP (Eventos Especiales)
SEVTCMPbits.SEVTCMP = 0x00FF;     // Valor de la base de tiempo PTMR al cual se produce el evento especial (0x000F)
SEVTCMPbits.SEVTDIR = 0x00;       // Direccion de la base de tiempo PTMR al cual se produce el evento especial (contando hacia arriba)
// Registro de configuracion interrupciones IEC2 (Base de Tiempo):
IEC2bits.PWMIE = 0x00;            // No habilita las interrupciones del Motor Control PWM
}
```



$$PTPER = \frac{FCY}{FPWM \cdot (PTMR\ Prescaler)} - 1$$

PWM Flanco Alineado (BT: *Free runing*)

$$PTPER = \frac{FCY}{FPWM \cdot (PTMR\ Prescaler) \cdot 2} - 1$$

PWM Pulso Centrado (BT: *Up/Down*)

Duty cycle for Free Running Mode: $\frac{\frac{PDCx}{2} - DT}{PTPER + 1}$

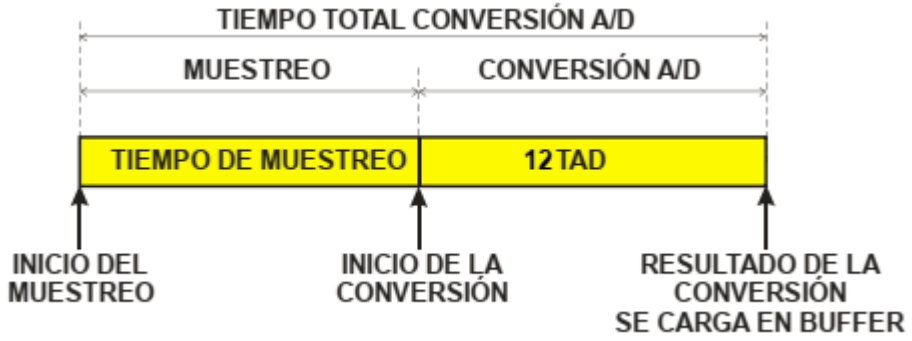
Duty cycle for Up/Down Mode: $\frac{PDCx - DT}{(PTPER + 1) \cdot 2}$

Note 1: DT (Dead Time) is the DTA<5:0> or DTB<5:0> register value.

2: For Independent PWM mode, ignore the value of DT.

CONVERSOR ADC

```
void Configura_ADC(void)
{// Configuración de las entradas analógicas (ADPCFG):
ADPCFGbits.PCFG0  = 0x01;    // AN0/RB0 como I/O digital.
ADPCFGbits.PCFG1  = 0x01;    // AN1/RB1 como I/O digital.
ADPCFGbits.PCFG2  = 0x01;    // AN2/RB2 como I/O digital.
ADPCFGbits.PCFG3  = 0x00;    // AN3/RB3 como entrada analógica.
ADPCFGbits.PCFG4  = 0x01;    // AN4/RB4 como I/O digital.
ADPCFGbits.PCFG5  = 0x01;    // AN5/RB5 como I/O digital.
ADPCFGbits.PCFG6  = 0x01;    // AN6/RB6 como I/O digital.
ADPCFGbits.PCFG7  = 0x01;    // AN7/RB7 como I/O digital.
ADPCFGbits.PCFG8  = 0x01;    // AN8/RB8 como I/O digital.
// Registro de configuración del ADC (ADCON1):
ADCON1bits.SSRC   = 0x03;    // Fuente de disparo de la conversión AD (evento especial):
                             // cada vez se completa un intervalo PWM (formado por , finaliza
                             // muestro e inicia la conversión AD.(Ojo afectado por postscaler de PWCON2 y comparación en SEVTCMP)
ADCON1bits.FORM   = 0x00;    // Formato de salida de la conversión: 00 = Integer (DOUT = 0000 00dd dddd dddd)
ADCON1bits.ADSIDL = 0x00;    // Continúa funcionamieneto del convresor AD en modo Idle.
ADCON1bits.ASAM   = 0x01;    // La captura comienza automáticamente después de la última conversión
// Registro de selección de de canal de entrada (ADCHS):
ADCHSbits.CH0SA   = 0x03;    // Para la entrada CH0+, selecciona la entrada AN0/RB0.
ADCHSbits.CH0NA   = 0x00;    // Para la entrada CH0-, selecciona a Vref-.
// Registro de configuración del ADC (ADCON3):
// Tiempo total del ADC = Tsamp + Tconv = 2Tad + 12Tad = SAMC*Tad + T(fijo) = 3.66us
ADCON3bits.ADCS   = 0x0E;    // Selección del Tad: ADCS<5:0> = (2*Tad/Tcy)-1, con: Tadmin = 256.41ns; Fcy = 7.159MHz*16/4 = 28,636 MHz
                             // Tad=TCY(ADCS+1)/2 para ADCS = E = 14 -> Tad=7,5Tcy
ADCON3bits.ADRC   = 0x00;    // Clock de conversor AD, derivado del clock del sistema.
ADCON3bits.SAMC   = 0x02;    // Selección del tiempo de muestreo automático (Tsamp): 2Tad.
// Registro de configuración del ADC (ADCON2):
ADCON2 = 0x00;       // Configura:
                     // - Vref+ = AVdd y Vref- = AVss.
                     // - Convierte sólo el CH0.
                     // - Si está habilitada la interrupción, interrumpe cada vez que se cumpleta una secuencia muestreo/conversión.
                     // - Y otras cosas mas.
ADCON1bits.ADON = 1;     // Se enciende el conversor AD.
IEC0bits.ADIE = 0x01;    // Habilita interrupción por finalización de conversión AD.
}
```



TIEMPO TOTAL DEL PROCESO DE CONVERSIÓN

$T_{TADC} = T_{SAMP} + T_{CONV}$

$T_{TADC} = T_{SAMP} + 12.T_{AD}$

T_{SAMP} = Tiempo de adquisición.
T_{AD} = Periodo de la señal de reloj de ADC.

$$T_{AD} = T_{CY} * (0.5 * (ADCS<5:0> + 1))$$

$$ADCS<5:0> = 2\,\frac{T_{AD}}{T_{CY}} - 1$$

ADCS<5:0>, Bits del registro ADCON3