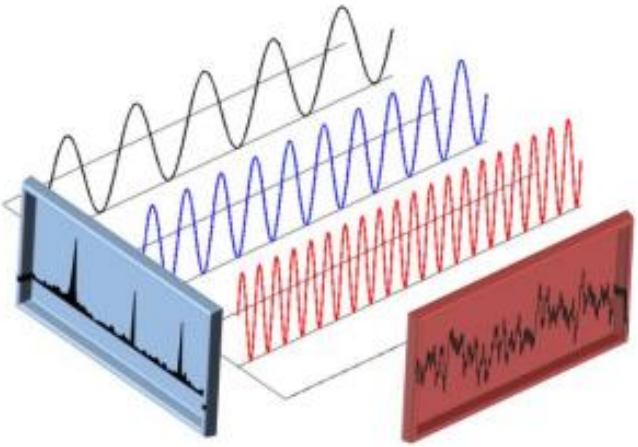
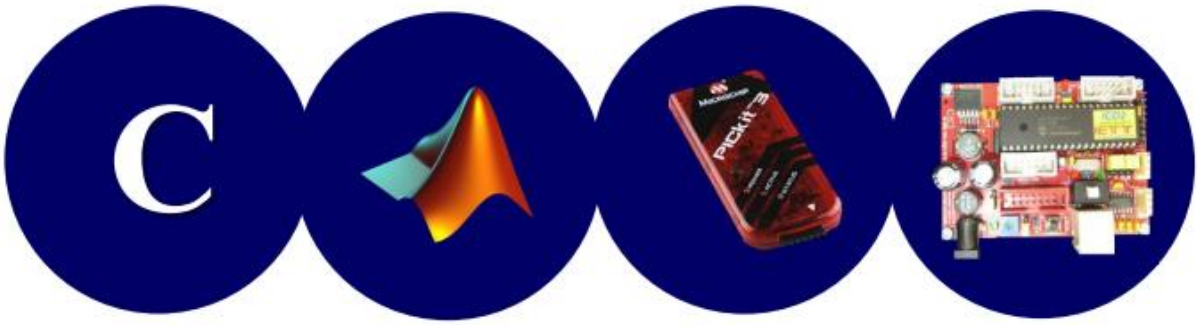


Procesamiento Digital de Señales con el dsPIC33EP256MU806



Julio 2016

ISCA ELECTRONICS

Grupo de Investigación en Sistemas de control Avanzado

Primera Clase

I) El Controlador Digital de Señales dsPIC33EP

La familia dsPIC33EP es un controlador digital de señales de 16 bits (ancho de palabra de la memoria de datos) y una velocidad de procesamiento de 70MIPS (millones de instrucciones por segundo). Cuenta con una máquina DSP interna que puede ejecutar cálculos matemáticos a gran velocidad como las instrucciones tipo MAC (a + b*c) en un solo ciclo de instrucción lo cual le permite ser usado en aplicación de motores DC y AC y en procesamiento digital de señales en filtros IIR, FIR y análisis con la FFT (Fast Fourier Transform).

Esta familia se diferencia de las anteriores, como la familia 30F, porque vienen con periféricos ampliados y versátiles como el ADC de 12 bits, el contador de encoder de 32 bits, o el modulo DMA, poseen mayor velocidad de procesamiento y se puede ubicar a los periféricos en diferentes pines del dsPIC, permitiendo diseñar un PCB más simple.

Mediante programación estructurada puede realizar las tareas de un PLC, ejecutando varias tareas simultáneamente en un mismo intervalo de tiempo.

Los dsPIC están divididos en tres partes: El CPU, el sistema de integración y los periféricos.

El compilador XC16: Es un programa que nos permite traducir nuestro código escrito en lenguaje C a lenguaje Asembler. Luego el Ensamblador traducirá el archivo asembler a archivo objeto *.o y finalmente el Linker une el archivo objeto con diferentes archivos compilados .a y .gld para producir el archivo *.cof y *.hex que se usarán para grabar el microcontrolador.

La versión que usaremos es XC16 v1.11 y el entorno de desarrollo será el MPLABX.

El compilador XC16 usa el lenguaje ANSI C-1989 que es una extensión del lenguaje C original junto con sintaxis propias del dsPIC como atributos de variables y funciones o introducción de funciones asembler en nuestro programa.

Entre los tipos de datos soportados por el compilador tenemos los siguientes:

Tipo de datos enteros:

Tipo	bits	Mínimo	Máximo
char, signed char	8	-128	127
unsigned char	8	0	255
short, signed short	16	-32768	32767
unsigned short	16	0	65535
int, signed int	16	-32768	32767
unsigned int	16	0	65535
long, signed long	32	-2^{31}	$2^{31}-1$
unsigned long	32	0	$2^{32}-1$
long long, signed long long	64	-2^{63}	$2^{63}-1$
unsigned long long	64	0	$2^{64}-1$

Tipos de datos flotantes:

Tipo	bits	N min	N max
float	32	2^{-126}	2^{128}
double	32	2^{-126}	2^{128}
Long double	64	2^{-1022}	2^{1024}

La norma IEEE-754 define el formato para la representación en coma flotante

Cuando se guarda cualquier tipo de dato en la memoria RAM primero se almacena los bytes menos significativos, por ejemplo para guardar el número entero largo x=0x12345678 en la posición 0x100 de la RAM, es como sigue:

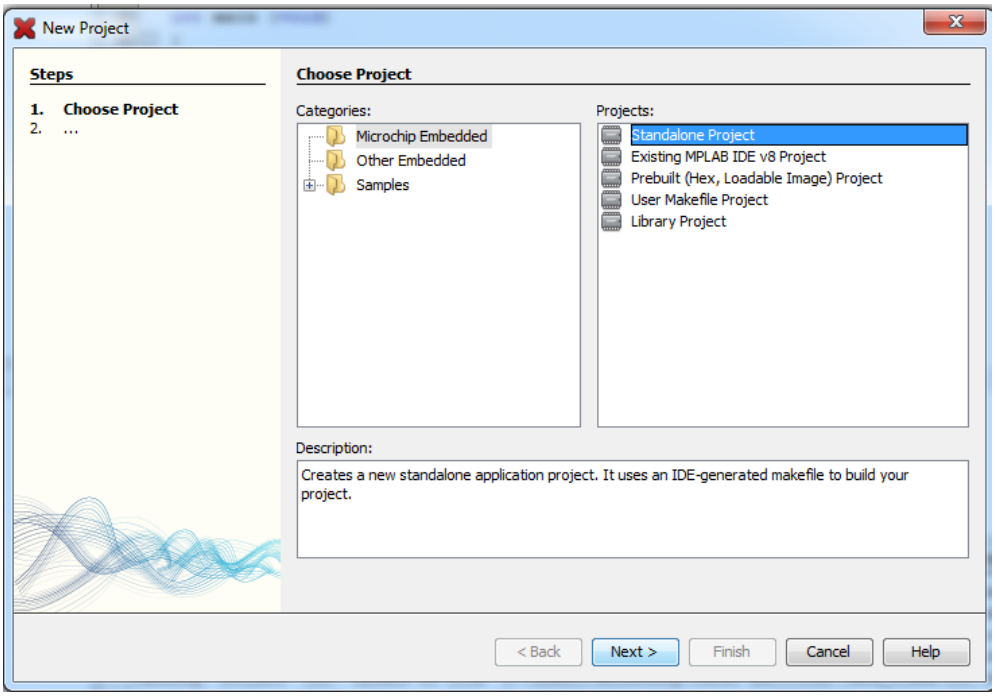
Dirección 0x100 valor=0x78
Dirección 0x101 valor=0x56
Dirección 0x102 valor=0x34
Dirección 0x103 valor=0x12

II) Manejo de Puertos

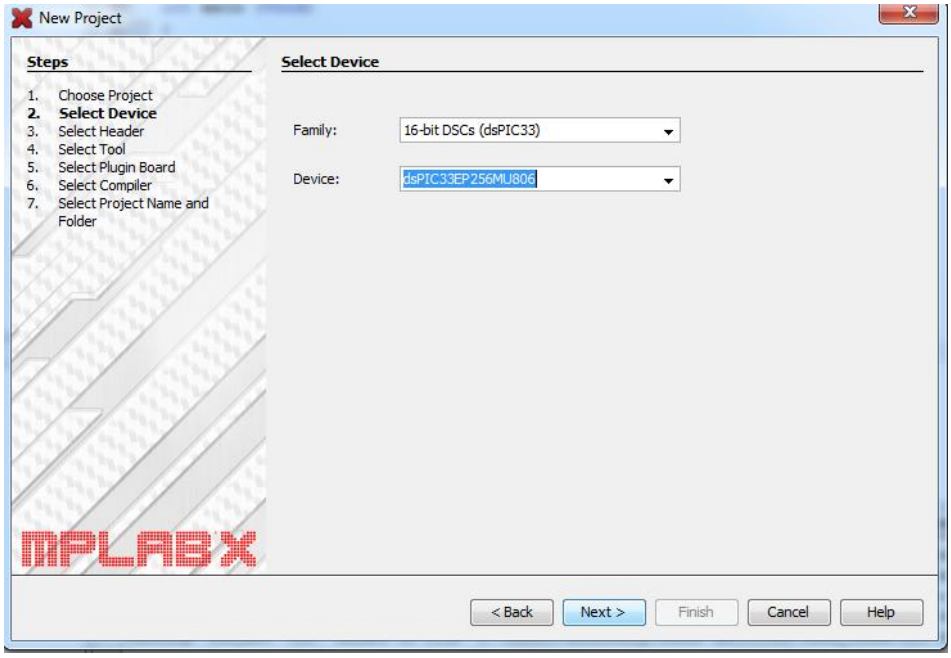
Para crear un proyecto usando el MPLABX IDE y el MPLAB XC16 se deben seguir los siguientes pasos:

1) Abrir el MPLABX, ir a File_New Project.

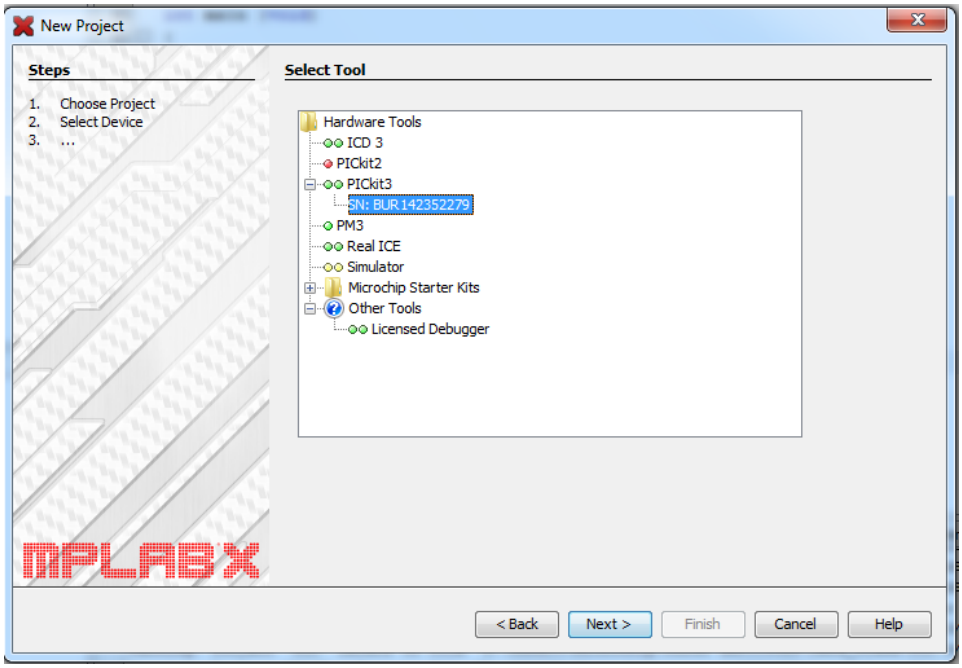
2) El primer paso es escoger el tipo de proyecto, debemos escoger el formato estándar, para ello seleccionamos la categoría Microchip Embedded y Projects_Standalone Project.



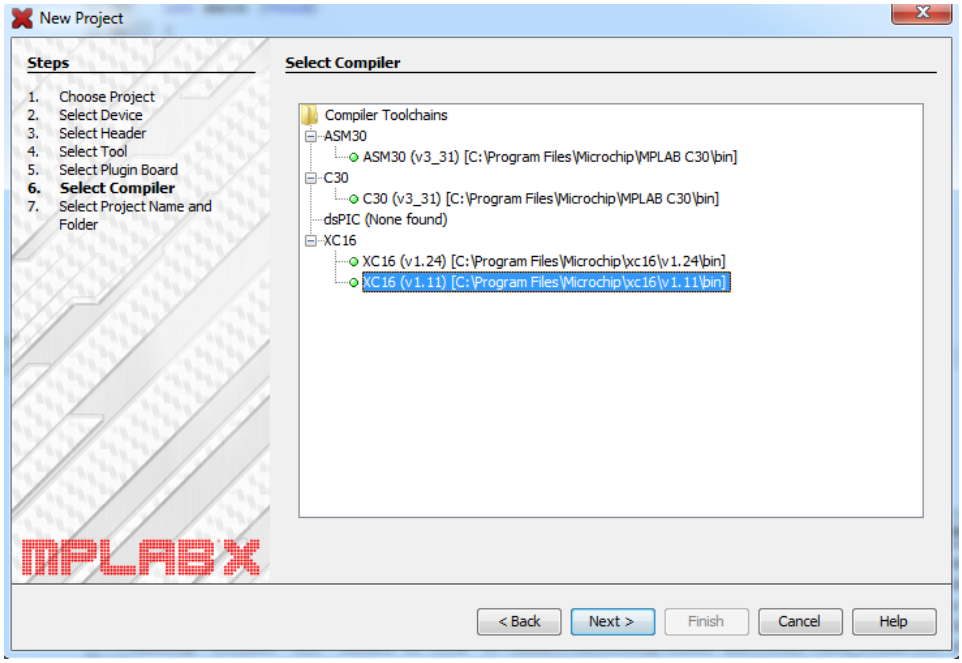
3) Seleccionamos el dispositivo a utilizar, en Family escogemos 16 bit DSC_dsPIC33 y en Device escogemos dsPIC33EP256MU806



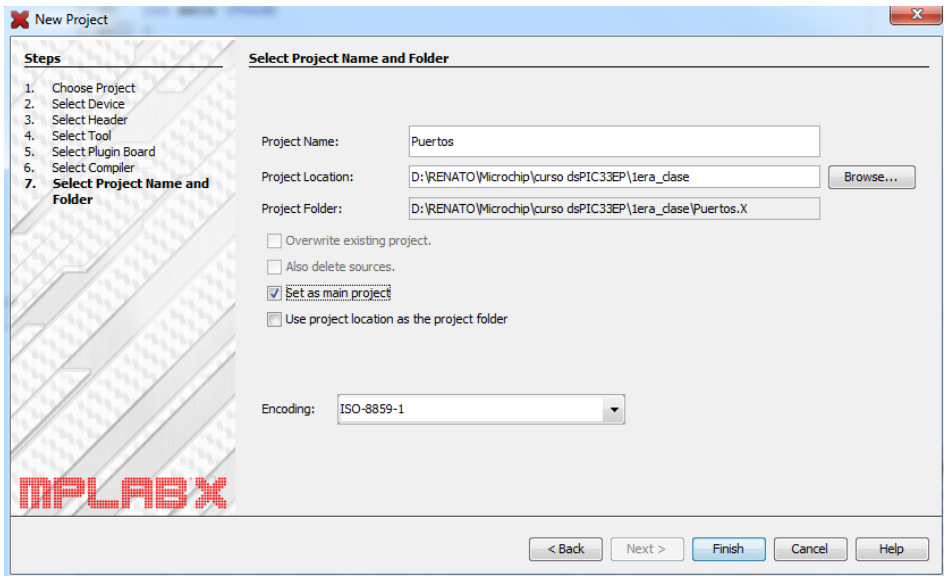
4) Seleccionamos la herramienta de programación y el debugger, este último es un programa depurador que sirve para testear nuestro programa y eliminar errores. Seleccionamos la herramienta de hardware PicKit3.



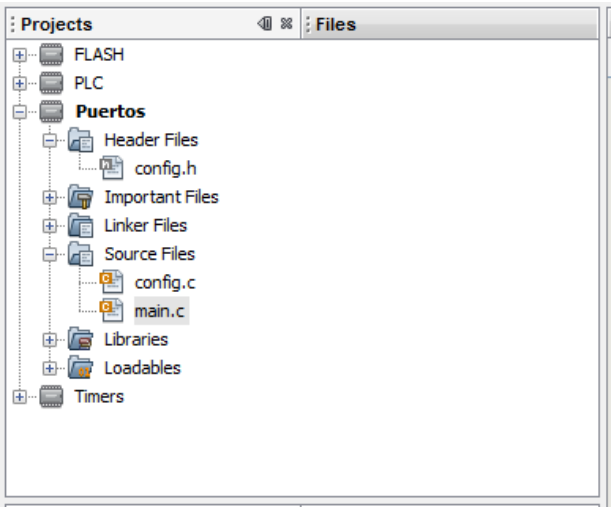
5) Seleccionamos el compilador XC16 v1.11, este traduce nuestro programa de lenguaje C a lenguaje Assembler.



6) Seleccionamos el nombre del proyecto y su ubicación en una carpeta. Para este caso el nombre del proyecto será Puertos. Seteamos el proyecto como proyecto principal “Set as main project”. Presionamos Finish.



7) Por último debemos seleccionar nuestros archivos de trabajo, para este caso los archivos son main.c, config.c y config.h deben estar ya ubicados en la carpeta del proyecto. Para hacerlo al lado izquierdo esta nuestro árbol del proyecto, hacer clic derecho en la pestaña Source Files y seleccionamos Add existing Item, se abre el buscador, debemos ir a la carpeta del proyecto y seleccionar estos tres archivos. Luego el archivo config.h debe arrastrarse a la pestaña Header Files.



La lista de pines del dsPIC33EP256MU806 es la siguiente:

A	B	C	D	E	F	G
-	RB0	-	RD0	RE0	RF0	-
-	RB1	-	RD1	RE1	RF1	-
-	RB2	-	RD2	RE2	-	RG2
-	RB3	-	RD3	RE3	RF3	RG3
-	RB4	-	RD4	RE4	RF4	-
-	RB5	-	RD5	RE5	RF5	-
-	RB6	-	RD6	RE6	-	RG6
-	RB7	-	RD7	RE7	-	RG7
-	RB8	-	RD8	-	-	RG8
-	RB9	-	RD9	-	-	RG9
-	RB10	-	RD10	-	-	-
-	RB11	-	RD11	-	-	-
-	RB12	RC12	-	-	-	-
-	RB13	RC13	-	-	-	-
-	RB14	RC14	-	-	-	-
-	RB15	RC15	-	-	-	-

Todos estos pines pueden usarse como periféricos de entrada y salida. Además muchos periféricos de entrada y salida pueden ubicarse en diferentes pines haciendo más flexible y fácil el diseño de nuestro PCB.

Registros asociados para el manejo de puertos.

TRISX: configura al puerto como entrada (1) o salida (0) digital.

PORTX: registro usado para leer el puerto.

LATX: registro usado para escribir el puerto.

ODCX: (Open Drain Control) Si un pin es configurado como salida digital tiene la posibilidad de actuar como drenador abierto colocando una resistencia externa tirada a por ejemplo 5v, pudiendo mandar señales de 5v a otros dispositivos. Todos los pines del dsPIC pueden actuar como Open Drain pero solo los que toleran 5v podrán usarse con este voltaje.

CNENX: (Change Notification) Cuando un pin del dsPIC está configurado como entrada puede generar una interrupción cuando ocurra un cambio de nivel en su entrada. Colocando a uno el bit asociado a nuestro pin habilitamos la interrupción.

CNPUX: Cuando un pin está configurado como entrada puede habilitarse resistencias pull_up, es decir internamente una resistencia se conecta entre el pin y 3.3v. Útil para usarse con teclados y botones.

CNPDX: De la misma forma que los pull_up, el pull_down son resistencias internas conectadas entre el pin y 0v.

Selección de Periféricos (PPS)

El PPS (Peripheral Pin Select) nos permite ubicar varios periféricos en diferentes pines de entrada y salida del dsPIC. Los pines con denominación RPI pueden ser usados por periféricos de entrada y los pines con denominación RP pueden ser usados por periférico de salida. No todos los periféricos pueden reubicarse, por ejemplo el ADC y el PWM tienen posiciones fijas en el dsPIC, así como otros periféricos que están ya escritos en el diagrama de pines.

En la tabla 11_1 del datasheet vemos los registros asociados a cada periférico mapeable. Por ejemplo, si queremos conectar el canal A del módulo de encoder 1 al pin RB10, tendremos que hacer lo siguiente:

```
RPINR14bits.QEA1R = 42;     // Assign QEA1 To Pin RPI42 (RB10).
```

En la tabla 11_3 y en la lista de registros de la página 261 del datasheet vemos como asociar los periféricos de salida a diferentes pines del dsPIC. Por ejemplo si queremos conectar la línea DO del módulo SPI1 al pin RE2, haremos lo siguiente:

```
RPOR5bits.RP82R = 5;        //Assign SDO1 To Pin RP82
```

El pin RE2 es RP82 que es configurable dentro del registro RPOR5 (lista de registros pág. 261) y en la tabla 11_3 vemos que la línea DO tiene el valor 5.

Manejo del Oscilador en los dsPIC33EP.

Puede cambiarse el oscilador en plena programación.

Posee un multiplicador de frecuencia PLL (Phase Locked Loop) y divisor de frecuencia.

Ante falla del clock puede generar una interrupción (FSCM Fail Safe Clock Monitor) donde se ordena detener la aplicación o cambiar de clock.

Los tipos de osciladores son dos:

Osciladores externos	Oscilador primario XT: [3.5_10]Mhz Media frecuencia Oscilador primario HS: [10_25]Mhz Alta frecuencia Oscilador Primario EC: [0.8_60]Mhz clock externo Oscilador secundario SOSC: 32.768khz
Osciladores internos	Oscilador Fast RC: FRC 7.37Mhz Oscilador FRC con PLL: FRCPLL Oscilador FRC con división: FRCDIVN Oscilador Low Power RC: LPRC 32khz es fuente de clock para Power Up Timer (PWRT), Watchdog Timer (WDT), Fail Safe Clock Monitor (FSCM)

Para configurar los MIPS (Millones de instrucciones por segundo) que vamos a trabajar debemos configurar el oscilador para que cumpla una serie de relaciones. Con un ejemplo explicamos estas relaciones, vamos a configurar el dsPIC para que trabaje a 70MIPS con un cristal de 8Mhz como entrada.

Tenemos **FIN=8Mhz, deseamos FCY=70Mhz (MIPS).**

Condiciones: 120Mhz < FSYS < 340Mhz

0.8Mhz < FPLLI < 8Mhz

Sea: N1=2, N2=2, M=70.

$$FPLLI = \frac{FIN}{N1} = \frac{8M}{2} = 4Mhz$$

$$FSYS = FPLLI * M = 4M * 70 = 280Mhz$$

$$FOSC = \frac{FSYS}{N2} = \frac{280M}{2} = 140Mhz$$

$$FP = FCY = \frac{FOSC}{2} = \frac{140M}{2} = 70Mhz = 70MIPS$$ si _DOZEN=1/1

Donde:

N1=PLLPRE+2=0+2=2

N2=2*(PLLPOST+1)=2*(0+1)=2

M=PLLDIV+2=68+2=70

Tal que:

_PLLDIV = (PLL divider)

_PLLPOST = (PLL postscaler)

_PLLPRE = (PLL prescaler)

III) Manejo de Timers

El dsPIC33EP256MU806 tiene nueve timers de 16bits (T1, T2, T3, T4, T5, T6, T7, T8, T9) y cuatro timers de 32bits (T2T3, T4T5, T6T7, T8T9).

Todos los timers pueden funcionar como contador interno, como contador externo síncrono y contador de ancho de pulso (Gated timer), además los timers se clasifican en tres grupos:

Timer tipo A (Timer1):

- Puede operar como contador externo asíncrono.
- Puede operar como contador de señal externa de clock 32khz

Timer tipo B: (Timer2, Timer4, Timer6, Timer8):

- Se puede juntar con un timer tipo C para formar un timer de 32bits.

Timer tipo C (Timer3, Timer5, Timer7, Timer9):

- Se puede juntar con un timer tipo B para formar un timer de 32bits
- Puede disparar al conversor AD

El periodo que genera un timer de 16bits se calcula como sigue:

$$Periodo = \frac{(PRX + 1) * PrescalerX}{FCY} \ldots\ldots (1)$$

El periodo que genera un timer de 32bits se calcula de la siguiente manera:

$$Periodo = \frac{(PR35 * 2^{16} + PR24 + 1) * Prescaler24}{FCY} \ldots (2)$$

Como los registros son de 16 bits el valor máximo que puede tomar PRX es $2^{16} - 1 = 65535$

Registros asociados:

- **TMRX:** registro contador de 16 bits
- **PRX:** registro de periodo de 16 bits
- **TXCON:** registro de control

Bits asociados para interrupción

- **TXIE:** bit habilita interrupción
- **TXIF:** bit flag de interrupción
- **TXIP:** bits prioridad de interrupción

Proceso de una Interrupciones:

Las interrupciones pueden ser generadas internamente como el fin de conteo de un timer o externas como el cambio de nivel en una patita del dsPIC, en cualquier caso estas interrumpen el flujo normal del programa, veamos con el siguiente ejemplo como es la secuencia de una interrupción:

Por ejemplo el flujo normal del programa ejecuta la operación flotante Z=X*Y, el programa guarda el dato X en los registros de trabajo W0 y W1 y guarda el dato Y en los registros W2 y W3, en ese momento ocurre una interrupción por cambio de nivel y el contador de programa salta al vector de interrupción por cambio de nivel (vector 23), en esta posición se encuentra la dirección de memoria de programa en donde comienza la rutina de interrupción. La rutina de interrupción puede hacer uso de los registros W0 a W3 y alterar luego el resultado de la multiplicación flotante, para que no ocurra esto, antes de ejecutar las rutinas de interrupción el programa salva los registros W0, W1, W2, y W3 en la memoria de datos. La dirección de inicio en la RAM donde están guardados estos registros se encuentra en el registro de pila W15.

Cuando termina la rutina de interrupción el programa vuelve a cargar los registros W0 a W3, luego el PC (program counter) regresa al flujo normal del programa. De esta manera nuestras operaciones no son destruidas cuando saltemos y regresemos de una interrupción.

IV) Manejo de la memoria FLASH

El dsPIC33EP256MU806 tiene 280Kbytes de memoria FLASH y 28Kbytes de memoria RAM. La lectura, borrado y escritura se ejecutan en páginas de 1024 bytes. El procedimiento para su uso es el siguiente:

1) Un puntero apunta a una página de la memoria FLASH.

_init_prog_address(puntero1, flashdata);

2) Copiar todo el contenido de la página FLASH a la memoria RAM.

(void) _memcpy_p2d16(ramdata, puntero1, _FLASH_PAGE);

3) Modificar los datos que deseamos en la memoria RAM.

ramdata[200]++;

4) Borrar la página FLASH.

_erase_flash(puntero1);

5) Escribir el contenido de la memoria RAM a la página FLASH. Debe empezar a escribirse desde la dirección cero a la dirección 1023, no puede empezarse en cualquier dirección.

```
for (i=0; i<_FLASH_PAGE/2; i++)
 {  pu1=puntero1+(4*i);
    _write_flash_word32(pu1, ramdata[2*i], ramdata[2*i+1]);
 }
```

Mientras se lee, borra o graba en la memoria FLASH el dsPIC no puede ejecutar otras instrucciones, ya que el CPU se encuentra atendiendo estas instrucciones y no puede ir a otras zonas de la FLASH.

V) Programación Multitasking o Multitarea, tipo PLC en tiempo real.

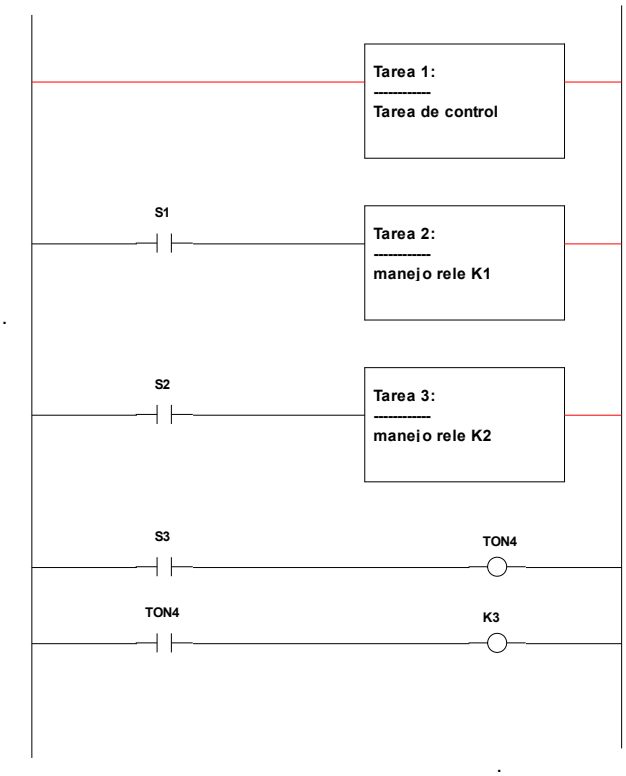
Este es el tipo de programación que usan los PLC, multitarea o determinístico, significa ejecutar varias tareas en un mismo intervalo de tiempo digamos 1mseg y nunca fallar o sacar resultados inesperados diferentes a la lógica diseñada. Los dos tipos de programación gráfica son diagramas Ladder y diagramas de flujo.

Veamos con un ejemplo la programación paralela: Programar en el dsPIC los siguientes diagramas gráficos.

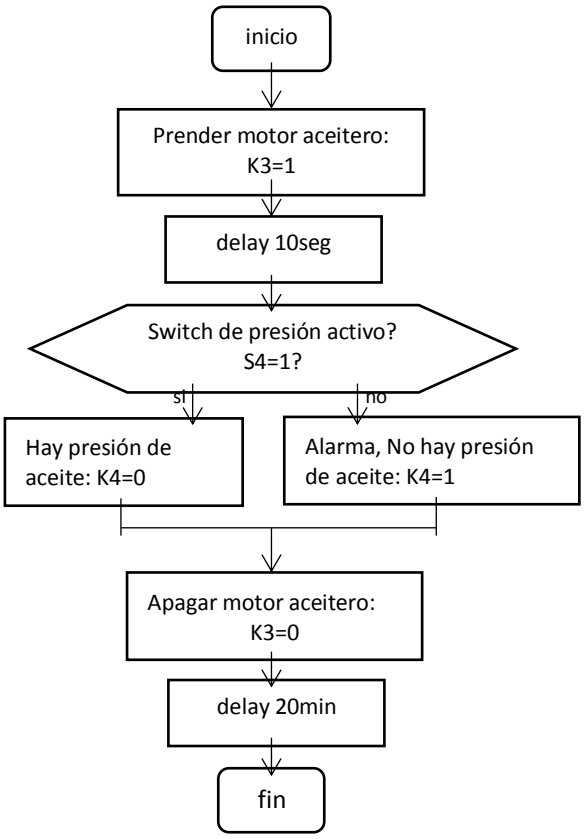
Diagrama Ladder

Tarea 1: Tarea de control
Tarea 2: prender relé K1 6 veces con periodo de 0,8seg, luego apagarlo por 2seg y repetir el proceso.
Tarea 3: prender relé K2 15 veces con periodo de 0,4seg, luego apagarlo por 4seg y repetir el proceso.
Tarea 4: retrasar la activación del relé K3 por 6seg, usando un timer ON delay
El diagrama Ladder se ejecuta cada 1mseg.
Realizar los flujos y la programación.

Desarrollar los diagramas de flujo de las tareas 2, 3 y 4.



Ejemplo de un diagrama de flujo: temporización de un motor aceitero



En un programa más complejo nuestra Tarea 1 puede ser una rutina de control de motores o de análisis matemático fuerte, la Tarea 2 el manejo de un teclado, la tarea 3 el manejo de una pantalla gráfica TFT y la tarea 4 de comunicación con otros dispositivos o con una PC. Todas estas tareas se ejecutan en paralelo respetando un tiempo límite para cada tarea por ejemplo las tareas 1 y 4 pueden ejecutarse cada 1mseg mientras que las tareas 2 y 3 cada 100mseg, todo ello usando el mismo procesador.

VI) Acceso Directo a Memoria (DMA)

Es un periférico que realiza la transferencia de datos entre el un periférico y la memoria RAM, sin intervención de la CPU, aumentando la eficiencia de las aplicaciones al disminuir los tiempos de latencia de las interrupciones muy importantes en aplicaciones de tiempo real.

Sus características son:

Puede trabajar con dos tipos de memoria RAM, la SRAM (Static RAM) y la DPSRAM (Dual Ported SRAM), como se observa en la fig. 1, las dos ubicadas en el mismo espacio de datos. El CPU y el DMA pueden leer y escribir en la DPSRAM sin interferencia al mismo tiempo ya que cuenta con dos puertas de acceso, mientras si quieren acceder a la SRAM, un árbitro permitirá el acceso a solo uno de ellos.

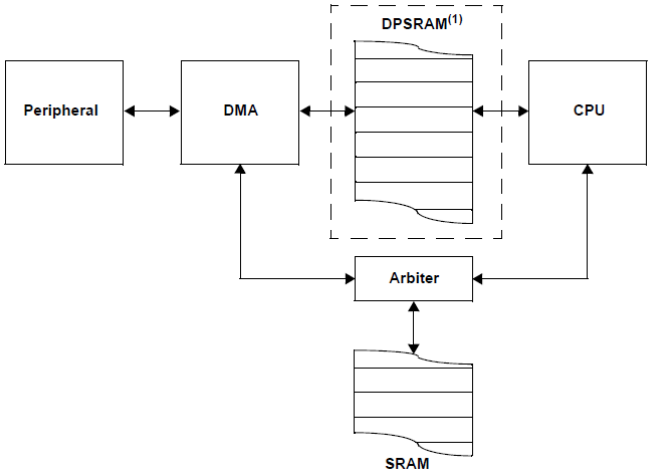


Figura 1. Diagrama general del proceso DMA

Soporta 15 canales, cada uno puede escribir datos del periférico a la RAM o viceversa, (periférico a memoria, memoria a periférico).

El tamaño de los datos puede ser byte (8 bits) o Word (16 bits).

Puede generar interrupción después de transferir la mitad o todo el bloque de datos.

El inicio de la transferencia del bloque de datos puede ser manual (software) o automático (DMA).

Modos de transferencia son OneShot o enviar un solo bloque y detenerse, la transferencia continua, y el modo PingPong donde se alternan dos bloques de datos cuyas direcciones están en DMAxSTA y DMAxSTB.

Diferentes modos de direccionamiento a la RAM.

Periféricos soportados:

ECAN: Enhanced Controller Area Network

DCI: Data Converter Interface

ADC: Analog to Digital Converter

SPI: Serial Peripheral Interface

UART: Universal Asynchronous Receiver Transmitter

IC: Input Capture

OC: output Compare

PMP: Parallel Master Port

Registros Asociados:

DMAxCON: (Control) Escoge el canal, dirección de la transferencia, tamaño del dato, modo de direccionamiento y modo de operación (OneShot o Continuo).

DMAxREQ: (IRQ) Asocia el DMA a un periférico

DMAxSTA: (Start Address A) Dirección de inicio del bloque de datos en la RAM.

DMAxSTB: (Start Address B) Dirección alterna de inicio de datos en la RAM, trabaja en el modo PingPong.

DMAxPAD: (Peripheral Address) Dirección en la RAM donde está ubicado el periférico o su registro de trabajo.

DMAxCNT: (Transfer Count) El número de datos a enviar del periférico a la RAM o viceversa es DMAxCNT+1.

DSADR: (Most Recent Address) captura la dirección del bloque de datos en la RAM (SRAM o DPSRAM).

DMAPWC: (Peripheral Write Collision Status) Detecta si hay colisiones de escritura mediante flags, esritura simultanea de la DMA y el CPU en el periférico.

DMARQC: (Request Collision Status) Cuando coinciden una petición de interrupción DMA y la inicialización manual de transferencia (FORCE=1) activa estos flags de colisión, el dsPIC da prioridad a la interrupción descartando el inicio manual.

DMALCA: (Last Channel Active) último canal activo usado por el DMA.

DMAPPS: (PingPong Status) Almacena la dirección del bloque de datos que se está manipulando, sea DMAxSTA o DMAxSTB.

Segunda Clase

I) Transmisión UART (Universal Asynchronous Receiver Transmitter)

Características:

- Cuatro módulos UART, comunicación asíncrona, no depende de una señal de clock.
- Transmisión full dúplex, tiene el canal de transmisión y recepción separados que le permite transmitir y recibir en cualquier momento y en full dúplex (al mismo tiempo), es decir, el funcionamiento del módulo transmisor no depende del módulo receptor y viceversa.
- 8bits, 9bits, transmisión de datos: En transmisión de 9 bits el noveno bit se usa para indicar si el byte enviado es dirección o dato.
- Paridad par, impar, no paridad
- Control de flujo por hardware usando los pines UxRTS y UxCTS.
- El buffer de transmisión y buffer de recepción tienen cuatro registros de almacenamiento cada uno.
- Periférico con DMA (Acceso directo a memoria) que nos permite leer y escribir los datos en un buffer de registro, este buffer trabaja con el hardware del UART para manejar las secuencias de lectura y escritura liberando al usuario de esta tarea.

La paridad par e impar se usa en 7 bits de transmisión, el octavo bit está para hacer la suma de los bits un número par (si es paridad par) o impar (si es paridad impar). Por ejemplo para transmitir el dato 0x3A en modo 7bits tenemos que el octavo bit en paridad par es cero y en paridad impar es uno:

0	1	1	1	0	1	0	**0**
7	6	5	4	3	2	1	0

Paridad par: 0x3A => transmite el dato 0x74

0	1	1	1	0	1	0	**1**
7	6	5	4	3	2	1	0

Paridad impar: 0x3A => transmite el dato 0x75

- Uno o dos bits de stop
- Tasas de transmisión desde 66 bps a 17.5 Mbps a F_{CY} = 70 MHz

Registros asociados:

- **UxMODE:** registro de configuración, habilita el UART, polaridad, paridad, numero de bits del dato, bit de stop, control de flujo por hardware.
- **UxSTA:** registro de configuración y estados o flags de transmisión y recepción.
- **UxRXREG:** registro buffer de recepción
- **UxTXREG:** registro buffer de transmisión
- **UXBRG:** registro de configuración de velocidad en baudios

Tasa de transmisión en baudios (bits/seg)

$$BaudRate = \frac{Fcy}{16 * (UxBRG + 1)}, si\ BRGH = 0$$

$$BaudRate = \frac{Fcy}{4 * (UxBRG + 1)}, si\ BRGH = 1$$

El módulo UART está formado por tres sistemas que son el Generador de Baudios, el módulo transmisor y el módulo receptor:

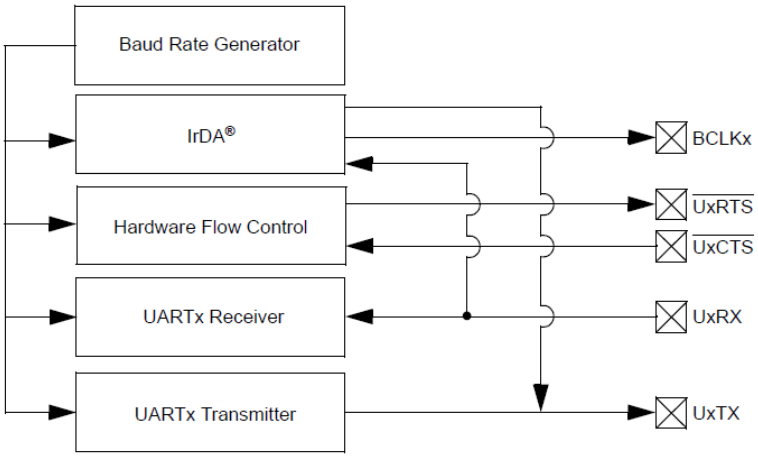
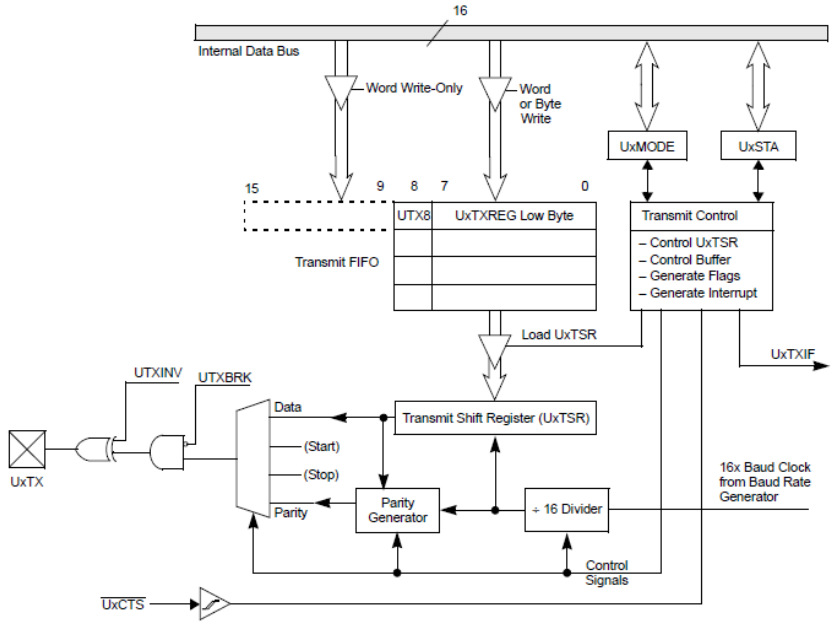


Figura : Bloques que conforman el módulo UART.



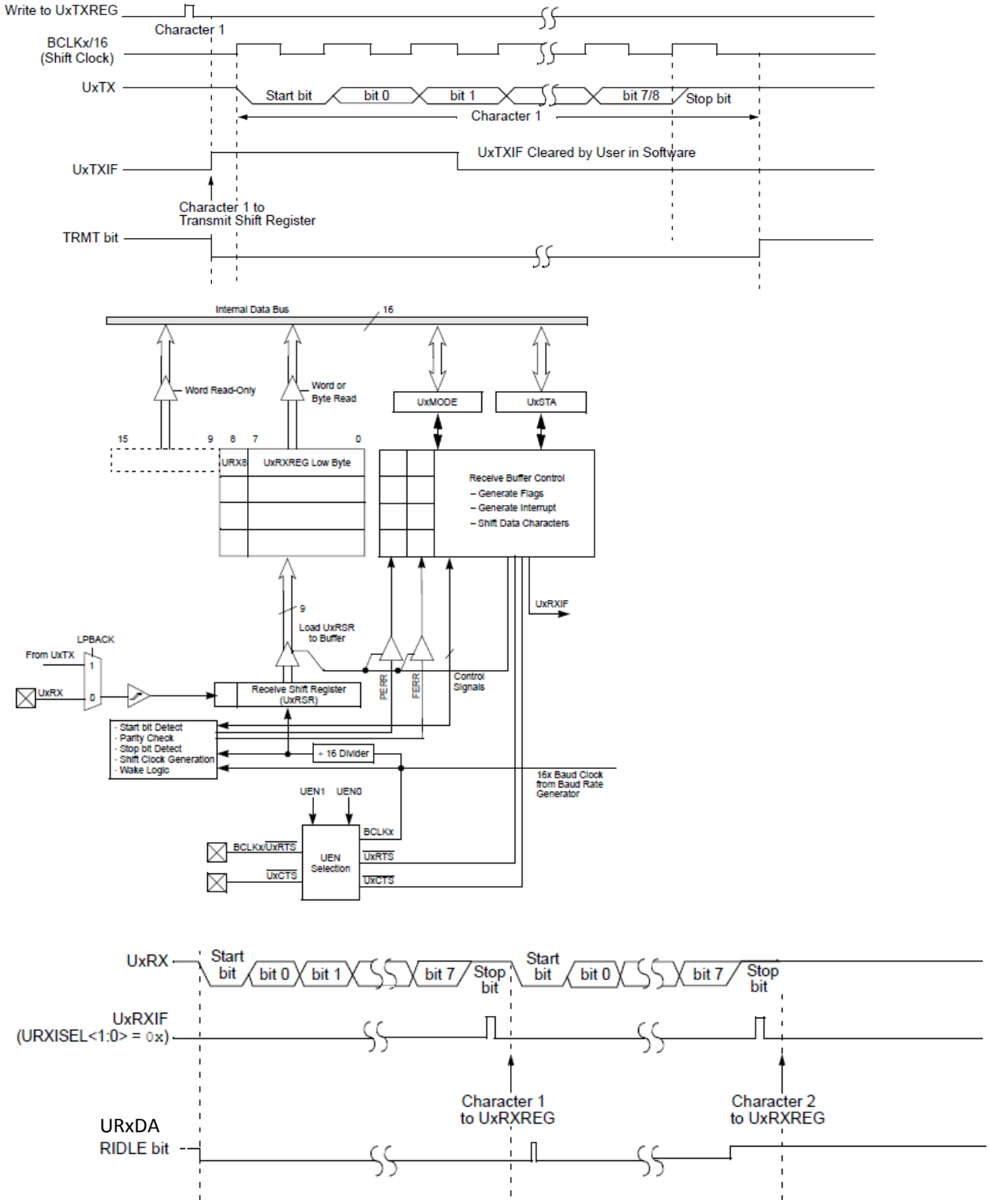


Figura 12: Módulo transmisión y recepción UART

Bits de detección de errores:

OERR bit: Flag que indica sobreflujo de caracteres recibidos en el buffer de recepción. Esto ocurre cuando el buffer de recepción tiene sus cuatro bytes llenos y un quinto carácter ha sido recibido en el registro de desplazamiento. En este caso debe leerse los cinco bytes en espera y borrar por software el flag OERR.

FERR bit: Flag que indica que el bit de STOP recibido está en nivel bajo.

PERR bits: Flag que indica error de paridad. En el caso de paridad par (EVEN) la suma de unos de los ocho bits debe ser un número par, si es impar el flag PERR será seteado.

Los bits FERR y PERR deben ser leídos antes de leer el dato.

II) Transmisión SPI (Serial Peripheral Interface)

Características:

- Cuatro módulos SPI, con comunicación síncrona que depende de la señal de clock.
- Útil para la comunicación con otros periféricos y microcontroladores.
- Periférico con DMA (Acceso directo a memoria) que nos permite leer y escribir los datos en un buffer de registro, este buffer trabaja con el hardware del SPI para manejar las secuencias de lectura y escritura liberando al usuario de esta tarea.

Registros asociados:

- SPIxBUF: Tiene dos registros internos separados SPIxRXB (recepción) y SPIxTXB (transmisión), el usuario no tiene acceso directo a estos, solo al registro SPIxBUF. En modo buffer reforzado el SPIxBUF tiene un buffer de 8 registros, ocho para el SPIxRXB y ocho para el SPIxTXB.
- SPIxCON1: registro de configuración, clock prescaler, modo maestro o esclavo, comunicación en modo byte (8bits) o Word (16bits) polaridad del clock, y formas de transmitir la data.
- SPIxCON2: registro de configuración para el modo Framed SPI
- SPIxSTAT: registro de configuración y estado, sobreflujo, flags de los buffer transmisión y recepción, habilita el módulo.

La transmisión y recepción ocurren simultáneamente

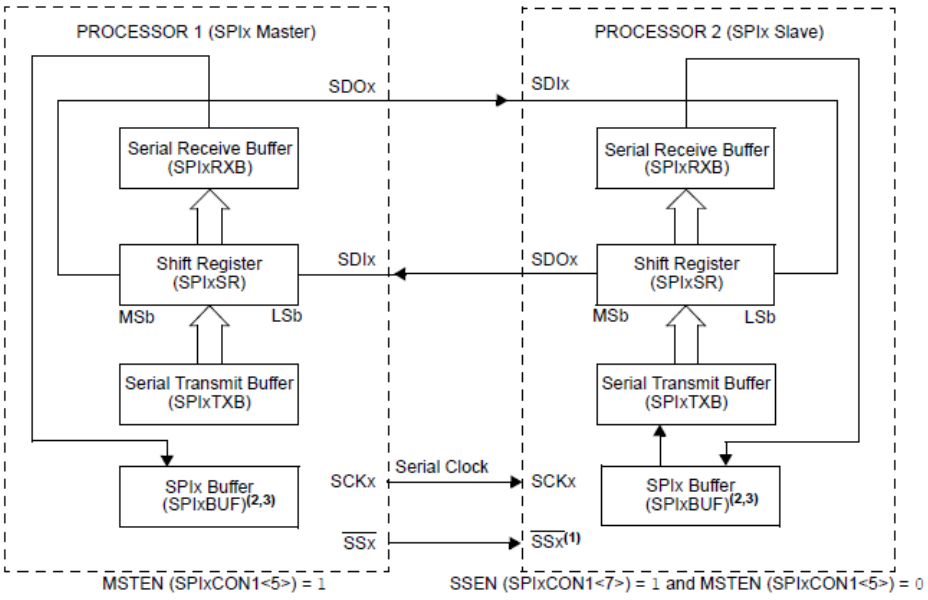


Figura 14: Esquema SPI maestro_esclavo

Pines usados:

- SPIx: serial data input
- SDOx: serial data output
- SCKx: entrada o salida de clock
- $\overline{SSx}$: chip select, activo en bajo

Formas de transmisión:

SMP (SPI data input):

- 1:Input data muestreado al final de la data Output
- 0: Input data muestreado en el medio de la data Output

CKE (SPI Clock Edge):

- 1: Output data cambia en la transición del clock de activo a inactivo
- 0: Output data cambia en la transición del clock de inactivo a activo

CKP (clock Polarity):

- 1: estado inactivo del clock (1), estado activo del clock (0)
- 0: estado inactivo del clock (0), estado inactivo del clock (1)

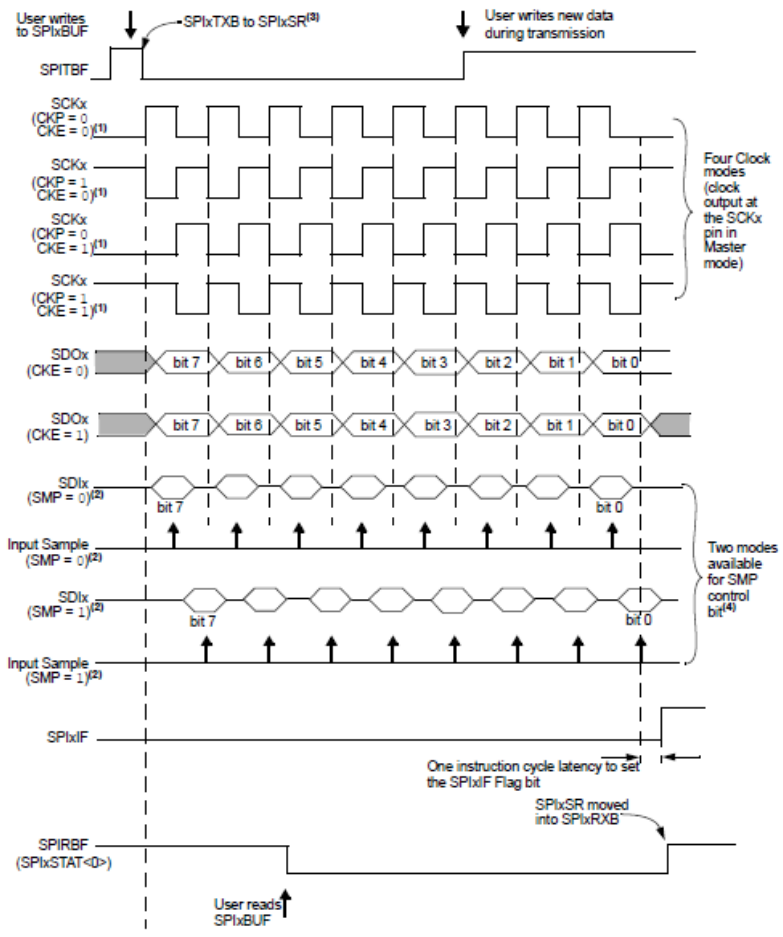


Figura 15: Formas de transmisión

Modos de operación:

- Transmisión y recepción de 8 bits y 16 bits.
- Modo Maestro y Esclavo.
- Modo Framed SPI

Frecuencia de transmisión:

$$F_{SCK} = \frac{Fcy}{Pri_{scaler} * Sec\_scaler} \dots\dots\dots (4)$$

Tercera Clase

Conversión Análoga Digital (ADC)

Características:

Conversión por aproximaciones sucesivas

ADC de 10 bits con cuatro canales de muestreo simultaneo, Fconv=1.1Msps

ADC de 12 bits con un canal de muestreo, Fconv=500ksps

24 entradas analógicas, AN0_AN15, AN24_AN31

Voltaje de referencia interno y externo

16 buffers de conversión y soporte DMA

Diferentes fuentes de disparo para la conversión

Escaneo automático de entradas

Cuatro formatos de conversión.

Registros usados:

AD1CON1	Prender módulo, ADC10_12bits, formato, fuente de disparo, canales usados, flag de conversión, flag de muestreo.
AD1CON2	Voltaje de referencia, escaneo de entradas, ocurrencia de interrupción.
AD1CON3	Fuente de clock, velocidad de conversión.
AD1CON4	Guardar datos en DMA, cantidad de RAM
AD1CHS123	Selecciona entradas positivas y negativas para los canales CH123.
AD1CHS0	Selecciona entradas positivas y negativas para el canal CH0.
AD1CSSH	Selecciona entradas para escaneo automático, solo CH0.
AD1CSSL	Selecciona entradas para escaneo automático, solo CH0.
ANSELy	configura pines de puertos B y E como analógico o digital.

Tiempos de conversión:

$T_{SAMP} = SAMC < 4:0 > * T_{AD}$ … (3.1) $T_{SAMP}: Tiempo\ de\ muestreo$

$T_{CONV} = 12 * T_{AD}$ … (3.2) $T_{CONV}: Tiempo\ de\ conversión\ para\ ADC\ de\ 10\ bits$

$T_{CONV} = 14 * T_{AD}$ … (3.3) $T_{CONV}: Tiempo\ de\ conversión\ para\ ADC\ de\ 12\ bits$

$T_{AD} = \frac{ADCS < 7:0 > +1}{F_{CY}}$ … (3.4) $T_{AD}: Periodo\ del\ clock\ para\ el\ ADC$

$T_{TCONV} = T_{SAMP} + T_{CONV}$ … (3.5) $T_{TCONV} = Tiempo\ total\ de\ conversión$

$T_{SAMP\_min} = 3 * T_{AD}$ … (3.6) $T_{SAMP\_min}: Tiempo\ de\ muestreo\ mínimo$

$F_{CONV\_max} = 500ksps$ … (3.7) $F_{CONV\_max}: Frecuencia\ de\ conversión\ máxima$

Diferentes formas de ordenar la conversión:

1) Muestreo manual y conversión manual
2) Muestreo manual y conversión automática
3) Muestreo manual y conversión iniciada por otro periférico
4) Muestreo automático y conversión manual
5) Muestreo automático y conversión automática
6) Muestreo automático y conversión iniciada por otro periférico

Se configura con los bits ASAM, SSRCG, SSRC

Dos voltajes de referencia:

Fijo: $A_{VDD} = 3.3v \;\; , \;\; A_{VSS} = 0v$

Variable: $V_{REF+} \;\; , \;\; V_{REF-}$

Se configura con los bits VCFG en AD1CON2

Dos formas de tomar las muestras:

1) Muestreo secuencial: muestreo un solo canal de conversión a la vez.
2) Muestreo simultaneo: Puede muestrear dos o cuatro canales simultáneamente, válido solo en ADC de 10 bits.

Usar los bits AD12B, SIMSAM, CHPS.

Escogiendo las entradas analógicas a convertir:

Podemos unir internamente por software las entradas analógicas y los canales de muestreo (opanes).

Usar los registros AD1CHS123 para canales CH123 y el registro AD1CHS0 para el canal CH0.

Dos fuentes de clock internas:

$T_{CY}: ciclo\ de\ instrucción, donde\ F_{CY} = 70\ MIPS$

$RC: Red\ RC\ interna$

$Si\ ADRC = 0 => \; T_{AD} = T_{CY} * (ADCS < 7:0 > +1) \qquad \ldots (3.8)$

$Si\ ADRC = 1 => \; T_{AD} = T_{ADRC} = 250nseg \qquad \ldots (3.9)$

Donde $T_{AD}: periodo\ del\ clock\ para\ el\ ADC, \; T_{AD\_min} = 117nseg \qquad \ldots (3.10)$

Formato de los datos convertidos:

1) Entero sin signo:

Rango: [0 , 4095] -> [0x0000 , 0x0FFF]

2) Entero con signo:

Rango: [-2048 , 2047] -> [0xF800 , … , 0xFFFF , 0 , 0x0001 , … , 0x07FF]

3) Fraccional sin signo:

Rango: [0 , casi 1] -> [0 , 0xFFF0]

Donde: $casi\ 1 = \frac{0xFFF0}{0x10000} = \frac{65520}{65536} = 0.9997$

4) Fraccional con signo:

Rango: [-1 , casi 1] -> [0x8000 , … , 0xFFFF , 0 , 0x0001 , … , 0x7FF0]

Donde: $casi\ 1 = \frac{0x7FF0}{0x8000} = \frac{32752}{32768} = 0.9995$

Escaneo automático de entradas:

Se seleccionan que entradas analógicas serán convertidas. Con cada secuencia de muestreo y conversión el hardware irá cambiando de entrada analógica con el canal CH0.

Con los registros AD1CSSH y AD1CSSL se seleccionan las entradas analógicas y el orden de conversión es de menor a mayor [AN0_AN31].

Configurar pines como digitales o analógicos:

Los registros ANSELB y ANSELE configuran al puerto como digital o analógico, siendo uno como analógico y cero como digital.

Tras un Reset ANSELB y ANSELE están en uno, por tanto los pines arrancan como analógicos.

Setear el TRISX como entrada antes de setear el ANSELB y ANSELE. Tras un reset todos los TRISX son seteados a uno, por tanto son estos pines arrancan como entradas.

Modulación por Ancho de Pulso (PWM)

Usado en:

- Control de motores AC de inducción (ACIM) y de imanes permanentes (PMSM).
- Control de motores DC, con escobillas (PMDC) y sin escobillas (BLDC).
- Fuentes de voltaje AC_DC, DC_DC, en UPS, PFC (Corrección del factor de potencia).

Características:

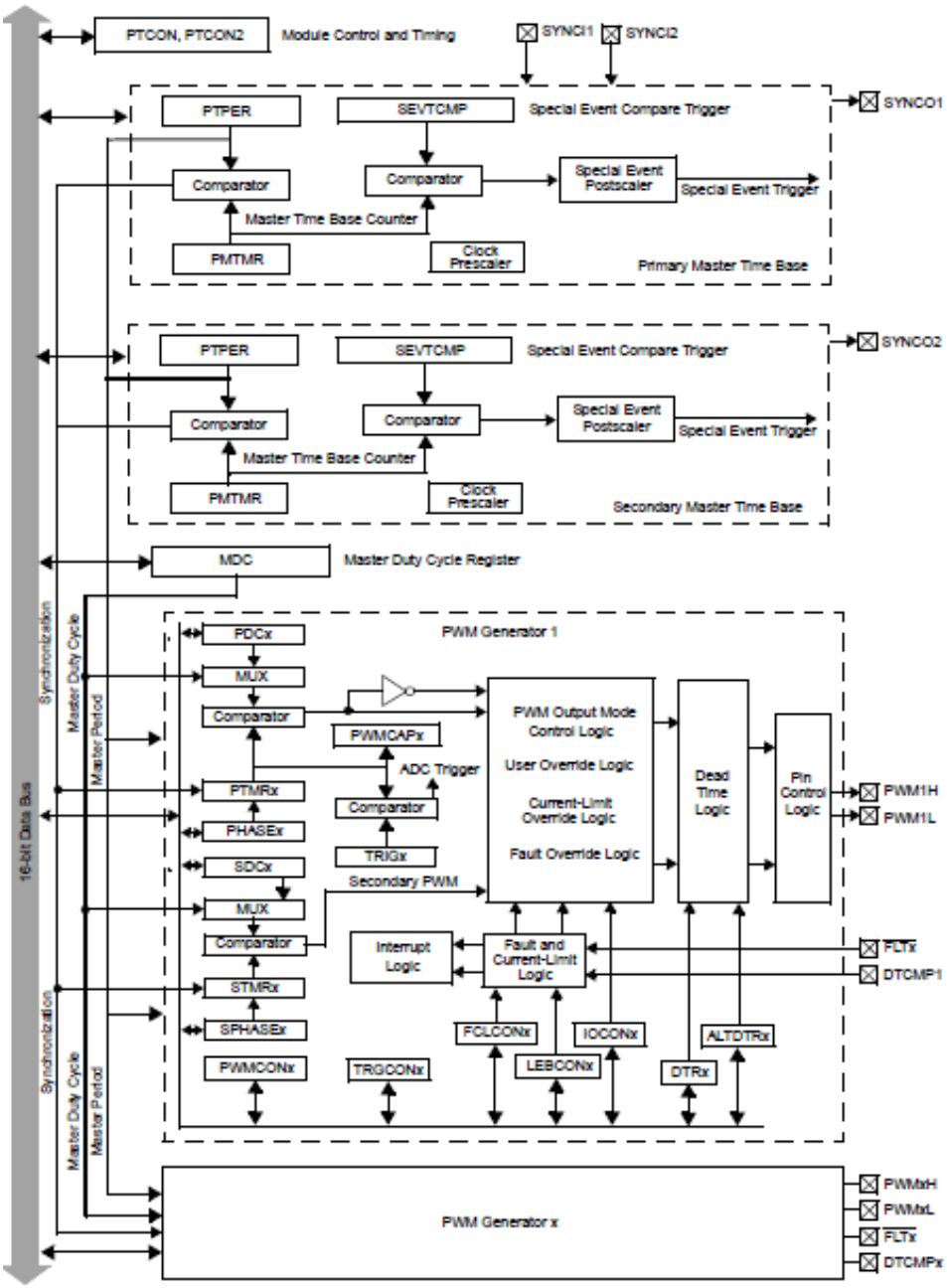
- Ocho salidas PWM, con cuatro generadores PWM, dos salidas PWM por generador (PWMXH, PWMXL).
- Cada salida PWM puede configurarse su periodo, ancho de pulso y fase.
- Actualización inmediata del periodo, ancho de pulso y fase en el mismo periodo.
- Puede disparar al ADC con cualquier generador PWM.

El dsPIC33EP256MU806 contiene cuatro generadores PWM cada uno con dos salidas PWMXH y PWMXL.
El periodo para los generadores PWM puede venir desde el módulo Master (PTPER) o generado por el mismo generador (PHASEX y SFHASEX).

Los generadores PWM tienen entradas para fallos y límites de corriente, si están habilitadas el generador PWM cancela la operación del PWM enviando las salidas a un estado preseteado.

También cada generador PWM puede disparar el ADC en cualquier punto de su periodo (TRIGX), así como el módulo Master puede también disparar al ADC (SEVTCMP).

El ancho de pulso de cada generador PWM es generado con su registro PDCX, también hay un ancho de pulso general para todos los generadores mediante el registro MDC (Master Duty Cycle).



Registros usados:

1) PTCON: Prender PWM, actualización de periodo, sincronización interna/externa.

2) PTCON2: Prescaler para el clock del módulo master primario.
3) PTPER: Periodo del módulo master primario.
4) SEVTCMP: Registro de comparación para evento especial del módulo master primario.

5) STCON: Actualización de periodo, sincronización interna/externa.
6) STCON2: Prescaler para el clock del módulo master secundario.
7) STPER: Periodo del módulo master secundario.
8) SSEVTCMP: Registro de comparación para evento especial del módulo master secundario.

9) CHOP: PWM CHOP clock generador.
10) MDC: Ancho de pulso para los generadores PWM.
11) PWMCONx: Habilita modos de fallo y sus flags. Fuentes de disparo y ancho de pulso, habilita tiempos muertos y su polaridad, habilita modo centro alineado, actualización inmediata de periodo, ancho de pulso y fase, x=1, 2, 3, 4.
12) PDCx: Ancho de pulso para los generadores PWM.
13) SDCx: Secundario ancho de pulso para los generadores PWM.
14) PHASEx: Fase para los generadores PWM.
15) SPHASEx: Secundario fase para los generadores PWM.

16) DTRx: Tiempo muerto para los generadores PWM.
17) ALTDTRx: Tiempo muerto alterno para lo generadores PWM.
18) TRGCONx: Registros de disparo para cada generador PWm.
19) IOCONx: Configura pines como PWM o puertos I/O y su polaridad. Quien controla pines PWM y sus estados tras un evento de fallo, inversión pines PWM.
20) TRIGx: Periodo para generar un evento ADC para cada generador PWM.
21) FCLCONx: Registro de control para fallo por límite de corriente (pines FAULT, o comparadores 123).
22) LEBCONx: Configura tiempos para blanqueo de señal de errores.
23) LEBDLYx: Tiempos para blanqueo de señal de errores.
24) AUXCONx: Registro de control auxiliar PWM., habilita o bloquea las señales de FAULT y Current Limit, habilita o bloquea las señales CHOP.
25) PWMCAPx: Captura base de tiempos PWM.

<u>Modo lado alineado:</u>

Bit CAM=0.

- Usado para mover motores DC.
- Cuenta siempre ascendente y contador retorna a cero cuando iguala el periodo.
- El PWM está seteado cuando el PWM es menor o igual al ancho de pulso.

<u>Modo centro alineado:</u>

- Bits CAM=1, ITB=1
- Simétrico respecto al centro del periodo.
- Usado para mover motores AC.
- Contador cuenta ascendente/descendente, el PWM es seteado en la parte central del periodo, cuyos flancos son limitados por el ancho de pulso.

CUARTA CLASE

I) Procesamiento digital de Señales

1) Filtros digitales

Sea la función de transferencia discreta de primer orden:

$$H_{(Z)} = \frac{Y_{(Z)}}{X_{(Z)}} = \frac{b_0 + b_1 * Z^{-1}}{1 + a_1 * Z^{-1}}$$

Descomponiendo en ecuación en diferencias:

$$y_{(k)} = b_0 * x_{(k)} + b_1 * x_{(k-1)} + a_1 * y_{(k-1)}$$

Las operaciones matemáticas serán con números enteros de 16 bits ya que el dsPIC realiza operaciones con enteros en un ciclo de instrucción utilizando la máquina dsp interna.

Las señales $x_k \; y \; x_{k-1}$ son señales analógicas en el rango de [0 1023($2^{10} - 1$)] para conversiones de 10 bits y en el rango de [0 4095($2^{12} - 1$)] para conversiones de 12 bits. Por tanto nuestra salida y_k también tendrá este mismo rango.

Si nuestra salida real usa otra escala, por ejemplo de [0 65535($2^{16} - 1$)] debemos multiplicar la salida por 2^4 donde n es 4, se hace un corrimiento de n bits a la izquierda para multiplicar y un corrimiento de n bits a la derecha para dividir, siempre la escala debe ser una potencia de dos.

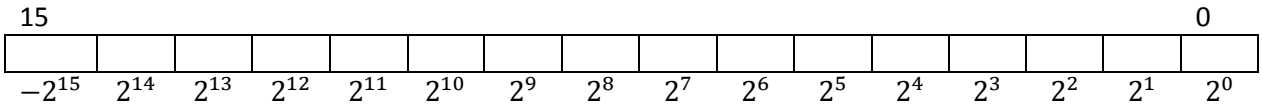
Las operaciones se realizan en formato entero tal que ningún componente de la ecuación supere el rango [-2^{15} , $2^{15} - 1$], nosotros debemos hacer que nuestros coeficientes del filtro sean lo más grandes posibles escalándolos a una potencia de dos.

Sean los coeficientes del filtro: $b_0 = 0.2451, b_1 = 0.0742, a_1 = 0.5094$, los multiplicamos por 2^{15}, luego realizamos las multiplicaciones y sumas en formato entero, al final la salida la dividimos entre 2^{15} para obtener el resultado correcto.

2) El formato fraccional 1.15

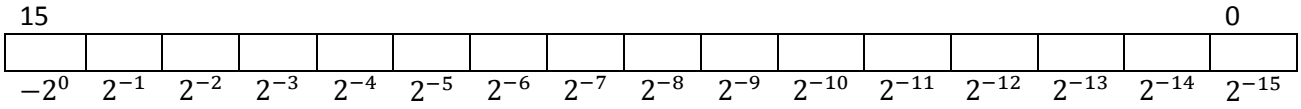
Tabla comparativa entre el formato entero de 16 bits y el formato de punto fijo 1.15.

Formato Entero de 16bits:



$$Valor = b15 * -2^{15} + b14 * 2^{14} + b13 * 2^{13} + \cdots + b0 * 2^0$$

Formato Fraccional Q1.15:



$$Valor = b15 * -2^0 + b14 * 2^{-1} + b13 * 2^{-2} + \cdots + b0 * 2^{-15}$$

Resumiendo:

Valor hexadecimal	[0x8000 , 0 , 0x7FFF]
Formato entero de 16bits	[-32768 , 0 , 32767]
Formato Fraccional Q1.15	[-1 , 0 , $1 - 2^{-15}$]

3) Transformada Rápida de Fourier (FFT)

Definiendo la DFT y la IFT:

$$X_{(k)} = \sum_{n=0}^{N-1} x_{(n)} * e^{-j2\pi kn/N} = \sum_{n=0}^{N-1} x_{(n)} * W_N^{kn} \quad \dots \quad k = [0, N-1]$$

$$x_{(n)} = \frac{1}{N} * \sum_{k=0}^{N-1} X_{(k)} * e^{\frac{j2\pi kn}{N}} = \frac{1}{N} * \sum_{k=0}^{N-1} X_{(k)} * W_N^{-kn} \quad \dots \quad n = [0, N-1]$$

Donde:
$W_N = e^{-j2\pi/N}$ factores de giro o TwiddleFactors
$X_{(k)}$ resultado en el dominio de la frecuencia
$x_{(n)}$ muestras en el dominico del tiempo

Algoritmos para la FFT

Algoritmo Radix-2 o base-2, $N = 2^k$	Diezmado en el tiempo Diezmado en frecuencia
Algoritmo Radix-4 o base-4, $N = 4^k$	Diezmado en el tiempo Diezmado en frecuencia
Algoritmo FFT Split Radix o Base partida	

Los algoritmos FFT usan las propiedades de simetría y periodicidad que no usan la DFT.

Propiedad de simetría: $W_N^{k+N/2} = -W_N^k$
Propiedad de periodicidad: $W_N^{k+N} = W_N^k$

El algoritmo FFT base 2 diezmado en frecuencia es el método usado por los dsPIC para calcular la FFT, reduce el número de sumas, restas y multiplicaciones complejas con respecto al algoritmo original de la DFT

Factores de giro o TwiddleFactors:

$$W_N^{kn} = e^{-j2\pi kn/N} = \cos\left(\left(\frac{2\pi}{N}\right) * kn\right) - j * sin\left(\left(\frac{2\pi}{N}\right) * kn\right)$$

Se necesitarían N*N TwiddleFactors o N^2 multiplicaciones complejas. El algoritmo FFT base 2 usa solo $\frac{N}{2} * log_2 N$ multiplicaciones complejas, usando solo N/2 TwiddleFactors $W_N^k, k = [0, \frac{N}{2} - 1]$

Representación gráfica algoritmo FFT base 2 diezmado en frecuencia

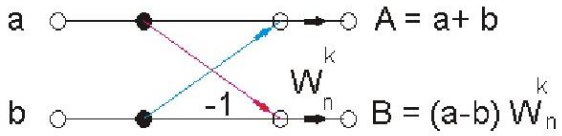


Figura 27: Mariposa básica, diezmado en frecuencia

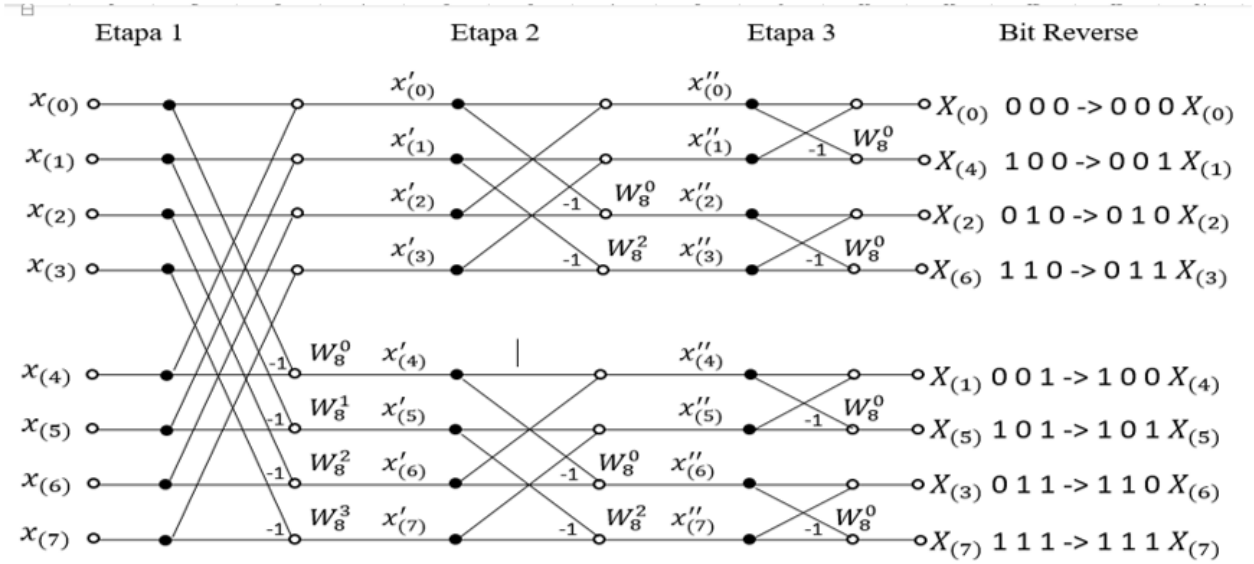


Figura 28: Algoritmo FFT diezmado en frecuencia, N=8 puntos

Veamos cómo trabaja el dsPIC para el cálculo de la FFT.

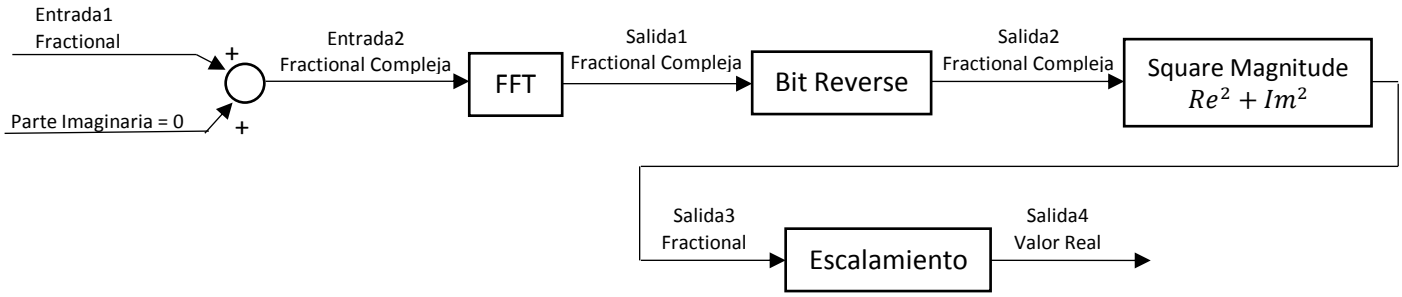


Figura 29: Procedimiento para calcular la FFT en el dsPIC

1) Tomar las muestras en formato fractional y almacenarlas en el array input. Input puede ser generado internamente con funciones matemáticas, recibida por un periférico de comunicación como el spi o adquirida del módulo ADC.

El vector de entrada SigCmpx es del tipo fractional complejo con parte real y parte imaginaria. El vector input nos proporciona la parte real y procedemos a llenar con ceros la parte imaginaria.

```
// declarando vector de entrada fractional compleja en memoria de datos Y
 extern fractcomplex sigCmpx[FFT_BLOCK_LENGTH]
__attribute__ ((eds, space(ymemory), aligned (FFT_BLOCK_LENGTH * 2 *2)));

// vector de entrada fractional real, donde pondremos la data de entrada.
extern fractional input[FFT_BLOCK_LENGTH];

// Twiddle Factor o Factores de giro como fractional complejo en memoria de programa
extern const fractcomplex twiddleFactors[FFT_BLOCK_LENGTH/2]
__attribute__ ((space(prog), aligned (FFT_BLOCK_LENGTH*2)));
```

$$ENTRADA1 = INPUT_{[512]} \quad (fractional)$$
$$ENTRADA2 = SigCmpx_{[512]} \quad (fractional)$$

```
// Divide la entrada por 2 para asegurar que esté en el rango de -0.5 a 0.5
  for ( i = 0; i < FFT_BLOCK_LENGTH; i++ )    /* The FFT function requires input data */
   {  /* to be in the fractional fixed-point range [-0.5, +0.5]*/
     sigCmpx[i].real = input[i] >> 1 ;                    /* So, we shift all data samples by 1 bit to the right. */
     sigCmpx[i].imag = 0;
                                                          /* Should you desire to optimize this process, perform */
   }
```

2) El cálculo de la FFT da el espectro en formato fractional complejo pero con las direcciones invertidas. BitReverse se encarga de poner los espectros en su correcta posición invirtiendo las posiciones de los bits de sus direcciones.

   Por ejemplo el vector complejo cuya dirección es 1010 0 0011 se ubicará en la dirección 1100 0 0101 mediante BitReverse, y el vector complejo que estaba en esta dirección pasa a la ubicación de la primera dirección.

$$SALIDA1 = SigCmpx_{[512]} \quad (fractional\ complejo)$$
$$SALIDA2 = SigCmpx_{[512]} \quad (fractional\ complejo)$$

Se realiza la FFT, con número de etapas ne, donde $2^{ne} = N\ numero\ de\ muestras$, dirección del array de entrada y dirección de los twiddlefactors como entradas. El resultado es de tipo fraccional complejo y es almacenado en el mismo array de entrada.

```
/* Perform FFT operation */
        FFTComplexIP (LOG2_BLOCK_LENGTH, &sigCmpx[0], (fractcomplex *) __builtin_psvoffset(&twiddleFactors[0]), (int)
       __builtin_psvpage(&twiddleFactors[0]));

        /* Store output samples in bit-reversed order of their addresses */
        BitReverseComplex (LOG2_BLOCK_LENGTH, &sigCmpx[0]);
```

3) El módulo al cuadrado de cada vector complejo es almacenado en SALIDA3.

$$SALIDA3_{[i]} = Re_i^2 + Im_i^2$$

$$SALIDA4_{[i]} = 2 * escala * \sqrt{SALIDA3_{[i]} * \frac{1}{32768}}$$

Donde escala es 2, ya que el dsPIC divide por 2 el array de entrada. SALIDA3 contiene 512 valores fractionals, los cuales la mitad de ellos es el espectro de la señal (256 valores), el resto es el espejo de estos. Por tanto tendremos una resolución en frecuencia de:

$$\Delta f = \frac{f_s}{N} = \frac{f_N}{N/2}$$

4) Además el dsPIC nos calcula la frecuencia en hz la componente de más lago espectro.

```
/* Find the frequency Bin ( = index into the SigCmpx[] array) that has the largest energy*/
/* i.e., the largest spectral component */
VectorMax(FFT_BLOCK_LENGTH/2, output, &peakFrequencyBin);

/* Compute the frequency (in Hz) of the largest spectral component */
peakFrequency = peakFrequencyBin*(SAMPLING_RATE/FFT_BLOCK_LENGTH);
```

II) Control de motores AC

Modulación Espacio Vector (SVM)

Tiene menor distorsión armónica THD en la corriente que el método PWM sinusoidal

Hace un mejor uso del voltaje V_{DCBUS} del inversor.

$$V_{\text{maxSVM}} = \frac{2}{\sqrt{3}} * V_{maxSENOPWM} = \frac{1}{\sqrt{3}} * V_{DCBUS}$$

Empleado en control en lazo cerrado (FOC) para motores AC (ACIM_PMSM).

1) Transformación para las corrientes:

Transformación de Clarke:

$$i_a + i_b + i_c = 0$$

$$i_\propto = i_a$$

$$i_\beta = (i_a + 2 * i_b) * \frac{1}{\sqrt{3}}$$

Transformación de Park:

$$i_d = i_\propto * \cos\theta_m + i_\beta * \sin\theta_m$$

$$i_q = -i_\propto * \sin\theta_m + i_\beta * \cos\theta_m$$

2) Transformación para los voltajes

Transformación de Park inversa:

$$v_\propto = v_d * \cos\theta - v_q * \sin\theta$$

$$v_\beta = v_d * \sin\theta + v_q * \cos\theta$$

$$v_d: voltaje\ directo\ (debilita\ al\ campo\ del\ rotor)$$
$$v_q: voltaje\ de\ cuadratura\ (produce\ torque)$$

Donde: $\theta = \theta_e: teta\ eléctrico\ para\ lazo\ abierto$

$$f_e = \frac{V_{rpm} * Npolos}{120}$$
$$\theta_e = \theta_e + d\theta_e \quad \dots \quad (rad)$$
$$d\theta_e = 2 * \pi * f_e * T_{PWM} \quad \dots \quad (rad)$$

$$\theta = \theta_m: teta\ mecánico\ para\ lazo\ cerrado$$
$$\theta_{mec} = \theta_{rotorQEI} * NparesPolos \quad \dots \quad (rad)$$

En lazo abierto:

Transformación de Clarke inversa:

$$v_{r1} = v_\beta$$
$$v_{r2} = \left(-v_\beta + \sqrt{3} * v_\alpha\right) * \frac{1}{2}$$
$$v_{r3} = \left(-v_\beta - \sqrt{3} * v_\alpha\right) * \frac{1}{2}$$

3) Calculando el sector

$$Si\ v_{r1} > 0 => A = 1, sino\ A = 0$$
$$Si\ v_{r2} > 0 => B = 1, sino\ B = 0$$
$$Si\ v_{r3} > 0 => C = 1, sino\ C = 0$$

Sector = A+2*B+4*C

4) Calcular la duración de los dos vectores del sector.

Se calcula primero las variables XYZ

$$X = \frac{\sqrt{3} * T_{PWM}}{V_{DC}} * v_\beta$$
$$Y = \frac{\left(3 * v_\alpha + \sqrt{3} * v_\beta\right) * T_{PWM}}{2 * V_{DC}}$$
$$Z = \frac{\left(-3 * v_\alpha + \sqrt{3} * v_\beta\right) * T_{PWM}}{2 * V_{DC}}$$

Se calculan los tiempos t1, t2 en base al sector:

sector	t1	t2
1	Z	Y
2	Y	-X
3	-Z	X
4	-X	Z
5	X	-Y
6	-Y	-Z

Calculando los tiempos Ta, tb, tc:

Sector 1	$t_{aON} = (T_{PWM} - t_1 + t_2)/2$ $t_{bON} = (T_{PWM} + t_1 + t_2)/2$ $t_{cON} = (T_{PWM} - t_1 - t_2)/2$
Sector 2	$t_{aON} = (T_{PWM} + t_1 + t_2)/2$ $t_{bON} = (T_{PWM} - t_1 - t_2)/2$ $t_{cON} = (T_{PWM} - t_1 + t_2)/2$
Sector 3	$t_{aON} = (T_{PWM} + t_1 + t_2)/2$ $t_{bON} = (T_{PWM} - t_1 + t_2)/2$ $t_{cON} = (T_{PWM} - t_1 - t_2)/2$
Sector 4	$t_{aON} = (T_{PWM} - t_1 - t_2)/2$ $t_{bON} = (T_{PWM} - t_1 + t_2)/2$ $t_{cON} = (T_{PWM} + t_1 + t_2)/2$
Sector 5	$t_{aON} = (T_{PWM} - t_1 - t_2)/2$ $t_{bON} = (T_{PWM} + t_1 + t_2)/2$ $t_{cON} = (T_{PWM} - t_1 + t_2)/2$
Sector 6	$t_{aON} = (T_{PWM} - t_1 + t_2)/2$ $t_{bON} = (T_{PWM} - t_1 - t_2)/2$ $t_{cON} = (T_{PWM} + t_1 + t_2)/2$

5) Calculando los anchos de pulso para los tres generadores PWM.

$$PDC1 = \frac{t_{aON} * Periodo1}{T_{PWM}}$$
$$PDC2 = \frac{t_{bON} * Periodo2}{T_{PWM}}$$
$$PDC3 = \frac{t_{cON} * Periodo3}{T_{PWM}}$$

Periodo1=Periodo2=Perido3=PTPER=3500 (o 10khz)

$T_{PWM} = 0.0001seg = 100useg$

PMSM: Brushless AC Permanent Magnetic Synchronous Motor

Tipos de PMSM	Características
SPM	Surface Permanent Magnet $L_d = L_q, \quad E = \frac{Ld}{Lq} = 1$
IPM	Interior Permanent Magnet E varía $L_d \neq L_q$ Puede operar en alta velocidad por debilitamiento de campo

A tener en cuenta:

- Especial cuidado en el arranque. La alineación de campos puede no ocurrir si el inicio es brusco con voltaje y frecuencia nominal.

- Debe conocerse el ángulo de inicio del motor, sino llevarlo a un ángulo conocido al inicio, por ejemplo el cero grados.
- Cuidado con el torque de la carga, puede impedir la alineación en el inicio.
- Puede arrancar en lazo abierto y enganchar a lazo cerrado cuando la estimación de posición del rotor sea precisa.
- Estimación de la posición del rotor en reposo: FOC + encoder.

Ventajas del PMSM sobre el ACIM (motor AC de inducción)

- Mayor eficiencia
- Mayor factor de potencia
- Menor tamaño para motores menores a 10kw.
- Mejor transferencia de calor
- Control FOC menos complejo.
- Mejor seguimiento de trayectorias de posición

Ventajas del PMSM sobre PMDC (motor DC de imanes permanentes)

- Mayor eficiencia
- Menor tamaño y costo para el mismo torque
- No mantenimiento de escobillas.

Velocidades mayores a la nominal debe darse por periodos cortos de tiempo, ya que un campo opuesto al campo natural del rotor puede desmagnetizarlo disminuyendo el torque y genera más calor.

Matemática en el dsPIC:

Clark: $i_\alpha = i_a \quad \dots \quad valores\ reales\ en\ punto\ flotante\ (en\ minúscula)$

$$\boxed{I_\alpha = I_a \quad \dots \quad valor\ escalado\ en\ entero\ (en\ mayúscula)}$$

$$i_\beta = \frac{(i_a + 2 * i_b)}{\sqrt{3}} \quad \dots \quad valores\ reales\ en\ punto\ flotante\ (en\ minúscula)$$

$$I_\beta = \frac{(I_a + 2 * I_b)}{\sqrt{3}}$$

$$I_\beta = \frac{\left(\frac{2^{14}}{\sqrt{3}} * I_a + \frac{2}{\sqrt{3}} * 2^{14} * I_\beta\right)}{2^{14}}$$

$$\boxed{I_\beta = \frac{(K_1 * I_a + K_2 * I_b)}{2^{14}} \quad \dots \quad valor\ escalado\ en\ entero\ (en\ mayúscula)}$$

$$donde: K_1 = \frac{2^{14}}{\sqrt{3}} = 9459, \quad K_2 = 2 * \frac{2^{14}}{\sqrt{3}} = 18918$$

Park: $i_d = i_\alpha * \cos\theta_m + i_\beta * \sin\theta_m$

$$I_d = I_\alpha * \cos\theta_m + I_\beta * \sin\theta_m$$

$$I_d = \frac{I_\alpha * (2^{15} * \cos\theta_m) + I_\beta * (2^{15} * \sin\theta_m)}{2^{15}}$$

$$\boxed{I_d = \frac{I_\alpha * COS\theta_m + I_\beta * SIN\theta_m}{2^{15}}}$$

$$i_q = -i_\alpha * \sin\theta_m + i_\beta * \cos\theta_m$$

$$I_q = -I_\alpha * \sin\theta_m + I_\beta * \cos\theta_m$$

$$I_q = \frac{-I_\alpha * (2^{15} * \sin\theta_m) + I_\beta * (2^{15} * \cos\theta_m)}{2^{15}}$$

$$\boxed{I_q = \frac{-I_\alpha * SIN\theta_m + I_\beta * COS\theta_m}{2^{15}}}$$

Calculando el θ_{rotor} para un encoder de 2000 ranuras:

$$\theta_{rotor} = (int)\left(POSCNT * \frac{2\pi rad}{2000 * 4} * \frac{32768}{\pi}\right)$$

$$\theta_m = \theta_{rotor} * Npares\_polos$$

Donde θ_m y θ_{rotor} varían de -32768 (-pi) a 32767 (casi pi)

$SIN\theta_m$ y $COS\theta_m$ varían de -32768 (-1) a 32767 (casi 1)

Calculando valores reales:

$$\boldsymbol{i_{ab\alpha\beta dq} = I_{ab\alpha\beta dq} * \frac{i_{max}}{\Delta I_{abmax}}} \quad \ldots \quad \boldsymbol{donde}: \boldsymbol{i_{max} = 100A, y\ \Delta I_{abmax} = 3685 - 2048},$$

$sensor\ usado\ ACS756\_100B$

Park inverso:

$$v_\alpha = v_d * cos\theta - v_q * sin\theta$$

$$V_\alpha = V_d * cos\theta - V_q * sin\theta$$

$$V_\alpha = \frac{V_d * (2^{15} * cos\theta) - V_q * (2^{15} * sin\theta)}{2^{15}}$$

$$\boxed{V_\alpha = \frac{V_d * COS\theta - V_q * SIN\theta}{2^{15}}}$$

$$v_\beta = v_d * sin\theta + v_q * cos\theta$$

$$V_\beta = V_d * sin\theta + V_q * cos\theta$$

$$V_\beta = \frac{V_d * (2^{15} * sin\theta) + V_q * (2^{15} * cos\theta)}{2^{15}}$$

$$\boxed{V_\beta = \frac{V_d * SIN\theta + V_q * COS\theta}{2^{15}}}$$

Donde: $\theta = \theta_e\ en\ lazo\ abierto, y\ \theta = \theta_m\ en\ lazo\ cerrado$

Calculando θ_e en lazo abierto: $f_e = \frac{V_{rpm} * Npolos}{120}$

$$d\theta_e = 2\pi * f_e * T_{PWM}$$
$$\Delta\theta_e = d\theta_e * \frac{32768}{\pi}$$
$$\theta_e = \theta_e + \Delta\theta_e \quad \ldots \quad \theta_e \; es \; entero$$

Calcular θ_m como en Park.

Los valores reales en punto flotante serían:

$$\boldsymbol{v_{dq\alpha\beta} = V_{dq\alpha\beta} * \frac{V_{DC_{BUS}}}{32768}} \quad \ldots \quad \boldsymbol{donde \; V_{DCBUS} = 300v}$$

Clark inverso:

Calculando los voltajes de las fases:

$$v_{r1} = v_\beta$$

$$\boxed{V_{r1} = V_\beta}$$

$$v_{r2} = \frac{-v_\beta + \sqrt{3} * v_\alpha}{2}$$

$$V_{r2} = \frac{\frac{-1}{2} * 2^{15} * V_\beta + \frac{\sqrt{3}}{2} * 2^{15} * V_\alpha}{2^{15}}$$

$$\boxed{V_{r2} = \frac{-K_3 * V_\beta + K_4 * V_\alpha}{2^{15}}}$$

$$v_{r3} = \frac{-v_\beta - \sqrt{3} * v_\alpha}{2}$$

$$V_{r3} = \frac{\frac{-1}{2} * 2^{15} * V_\beta - \frac{\sqrt{3}}{2} * 2^{15} * V_\alpha}{2^{15}}$$

$$\boxed{V_{r3} = \frac{-K_3 * V_\beta - k_4 * V_\alpha}{2^{15}}}$$

Donde: $K_3 = \frac{1}{2} * 2^{15} = 16384, y \; K_4 = \frac{\sqrt{3}}{2} * 2^{15} = 28377$

Los valores reales son: $\boldsymbol{v_{r1r2r3} = V_{r1r2r3} * \frac{V_{DCBUS}}{2^{15}}} \quad \ldots \quad \boldsymbol{donde \; V_{DCBUS} = 300v}$

Calculando los tiempos:

$$x = \frac{\sqrt{3} * T_{PWM}}{V_{DCBUS}} * v_\beta$$

$$\left(x * \frac{PTPER}{T_{PWM}}\right) = \frac{\sqrt{3} * T_{PWM}}{V_{DCBUS}} * \left(\frac{V_{DCBUS}}{2^{15}} * V_\beta\right) * \left(\frac{PTPER}{T_{PWM}}\right)$$

$$X = \left(\sqrt{3} * PTPER\right) * \frac{V_\beta}{2^{15}}$$

$$X = K_5 * \frac{V_\beta}{2^{15}}$$

$$donde: K_5 = \sqrt{3} * PTPER = \sqrt{3} * 3500 = 6062$$

$$y = \frac{\left(3 * v_\alpha + \sqrt{3} * v_\beta\right) * T_{PWM}}{2 * V_{DCBUS}}$$

$$\left(y * \frac{PTPER}{T_{PWM}}\right) = \left(\frac{3}{2} * V_\alpha + \frac{\sqrt{3}}{2} * V_\beta\right) * \frac{T_{PWM}}{V_{DCBUS}} * \frac{V_{DCBUS}}{2^{15}} * \frac{PTPER}{T_{PWM}}$$

$$Y = \frac{\left(\frac{3}{2} * PTPER * V_\alpha + \frac{\sqrt{3}}{2} * PTPER * V_\beta\right)}{2^{15}}$$

$$Y = \frac{K_6 * V_\alpha + K_7 * V_\beta}{2^{15}}$$

$$donde: K_6 = \frac{3}{2} * PTPER = \frac{3}{2} * 3500 = 5250$$

$$K_7 = \frac{\sqrt{3}}{2} * PTPER = \frac{\sqrt{3}}{2} * 3500 = 3031$$

$$z = \frac{\left(-3 * v_\alpha + \sqrt{3} * v_\beta\right) * T_{PWM}}{2 * V_{DCBUS}}$$

$$\left(z * \frac{PTPER}{T_{PWM}}\right) = \left(\frac{-3}{2} * V_\alpha + \frac{\sqrt{3}}{2} * V_\beta\right) * \frac{T_{PWM}}{V_{DCBUS}} * \frac{V_{DCBUS}}{2^{15}} * \frac{PTPER}{T_{PWM}}$$

$$Z = \frac{\left(\frac{-3}{2} * PTPER * V_\alpha + \frac{\sqrt{3}}{2} * PTPER * V_\beta\right)}{2^{15}}$$

$$Z = \frac{-K_6 * V_\alpha + K_7 * V_\beta}{2^{15}}$$

$$Los\ valores\ reales\ son: xyz_{real} = XYZ * \frac{T_{PWM}}{PTPER}$$

Tabla de sectores:

sector	t1	t2	T1	T2
1	z	y	Z	Y
2	Y	-x	Y	-X
3	-z	x	-Z	X
4	-x	z	-X	Z
5	x	-y	X	-Y
6	-y	-z	-Y	-Z

Valores de ancho de pulso PWM1, PWM2, PWM3:

Sector 1	$t_{aON} = (T_{PWM} - t_1 + t_2)/2$ $t_{bON} = (T_{PWM} + t_1 + t_2)/2$ $t_{cON} = (T_{PWM} - t_1 - t_2)/2$	$T_{aON} = (PTPER - T_1 + T_2)/2$ $T_{bON} = (PTPER + T_1 + T_2)/2$ $T_{cON} = (PTPER - T_1 - T_2)/2$
Sector 2	$t_{aON} = (T_{PWM} + t_1 + t_2)/2$ $t_{bON} = (T_{PWM} - t_1 - t_2)/2$ $t_{cON} = (T_{PWM} - t_1 + t_2)/2$	$T_{aON} = (PTPER + T_1 + T_2)/2$ $T_{bON} = (PTPER - T_1 - T_2)/2$ $T_{cON} = (PTPER - T_1 + T_2)/2$
Sector 3	$t_{aON} = (T_{PWM} + t_1 + t_2)/2$ $t_{bON} = (T_{PWM} - t_1 + t_2)/2$ $t_{cON} = (T_{PWM} - t_1 - t_2)/2$	$T_{aON} = (PTPER + T_1 + T_2)/2$ $T_{bON} = (PTPER - T_1 + T_2)/2$ $T_{cON} = (PTPER - T_1 - T_2)/2$
Sector 4	$t_{aON} = (T_{PWM} - t_1 - t_2)/2$ $t_{bON} = (T_{PWM} - t_1 + t_2)/2$ $t_{cON} = (T_{PWM} + t_1 + t_2)/2$	$T_{aON} = (PTPER - T_1 - T_2)/2$ $T_{bON} = (PTPER - T_1 + T_2)/2$ $T_{cON} = (PTPER + T_1 + T_2)/2$
Sector 5	$t_{aON} = (T_{PWM} - t_1 - t_2)/2$ $t_{bON} = (T_{PWM} + t_1 + t_2)/2$ $t_{cON} = (T_{PWM} - t_1 + t_2)/2$	$T_{aON} = (PTPER - T_1 - T_2)/2$ $T_{bON} = (PTPER + T_1 + T_2)/2$ $T_{cON} = (PTPER - T_1 + T_2)/2$
Sector 6	$t_{aON} = (T_{PWM} - t_1 + t_2)/2$ $t_{bON} = (T_{PWM} - t_1 - t_2)/2$ $t_{cON} = (T_{PWM} + t_1 + t_2)/2$	$T_{aON} = (PTPER - T_1 + T_2)/2$ $T_{bON} = (PTPER - T_1 - T_2)/2$ $T_{cON} = (PTPER + T_1 + T_2)/2$

$$PDC1 = T_{aON} = t_{aON} * \frac{PTPER}{T_{PWM}}$$
$$PDC2 = T_{bON} = t_{bON} * \frac{PTPER}{T_{PWM}}$$
$$PDC3 = T_{cON} = t_{cON} * \frac{PTPER}{T_{PWM}}$$