UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITECNICA SUPERIOR

Dpto. de INGENIERA DE SISTEMAS Y AUTOMATICA

# DISEÑO DE UNA MINI-MOTO CONTROLADA POR COMPUTADOR: ARQUITECTURA SOFTWARE

PROYECTO FIN DE CARRERA
INGENIERIA TÉCNICA INDUSTRIAL: ELECTRÓNICA

**Autor:** Pablo Núñez Yáñez

**Tutor:** Luis Moreno Lorente
Septiembre 2010

# Agradecimientos

Agradecer a mi compañero en este proyecto Rubén Espinosa Polo, porque este proyecto es tan mío como suyo, por todas las horas que hemos compartido en estos últimos meses, porque es el único que sabe por todo lo que hemos pasado y el esfuerzo que hemos invertido y porque sin él, este proyecto no habría sido lo mismo.

También agradecer a mis hermanos, y en especial a mis padres, por todos los valores que me han inculcado, por la educación recibida y por todo lo que aún me siguen aportando. Gracias por haberme apoyado y creído en mí durante la carrera, espero que se sientan orgullosos.

A mi grupo de amigos por estar ahí cuando se les necesita, por todos los momentos vividos y por las horas de estudio que hemos compartido en la biblioteca durante las épocas de exámenes, haciendo que todo fuera más llevadero. Gracias por estar siempre a mi lado y espero que sigan ahí por mucho tiempo.

Por último agradecer a mis compañeros de clase por todos los momentos compartidos durante la carrera, las horas de estudio y prácticas juntos, porque me han ayudado cuando lo necesitaba. No solo me llevo compañeros, sino amigos.

Gracias a todos.

Pablo Núñez Yáñez
28 de Septiembre de 2010

# Índice general

# Índice de figuras

## CAPITULO 1: INTRODUCCION Y OBJETIVOS



## CAPITULO 2: ESTADO DEL ARTE



## CAPITULO 3: MICROCONTROLADOR PIC 32



## CAPITULO 4: MPLAB

# Índice de tablas

# Capítulo 1.

# Introducción y objetivos

## 1.1. Introducción

A lo largo de la historia de la humanidad, el ser humano ha tratado de desarrollar todo tipo de maquinas que de algún modo u otro buscaban automatizar procesos de la vida cotidiana del hombre, de manera que estos artefactos hiciesen más fácil la vida de este. Con el paso de los años estas maquinas han ido ganando en complejidad y recibiendo todo tipo de nombres, hasta que en el año 1921, de la mano del escritor checo Karel Capek, fue usado, por primera vez, el termino robot en una novela de ficción.

Hay muchos trabajos que las personas no les gusta hacer, sea ya por ser aburrido o bien peligroso, siempre se va a tratar de evitar para no hacerlo. La solución más práctica en la historia era obligar a alguien para que hiciera el trabajo, esto se le llama esclavitud y se usaba prácticamente en todo el mundo.

Ahora los robots son ideales para trabajos que requieren movimientos repetitivos y precisos. Una ventaja para las empresas es que los humanos necesitan descansos, salarios, comida, dormir, y una área segura para trabajar, los robots no. La fatiga y aburrimiento de los humanos afectan directamente a la producción de una compañía, los robots nunca se cansan.

Uno de los campos de estudio dentro de la Robótica es la navegación autónoma de los robots. Desde hace muchos años, numerosos investigadores trabajan día a día para conseguir que los robots sean cada vez mas autónomos y puedan desplazarse por entornos cada vez más complejos con una mayor facilidad e independencia.

## 1.2. Objetivos

El objetivo que persigue el Proyecto Final de Carrera descrito en este documento es el desarrollo de una moto de automodelismo controlada mediante la obtención de datos de un ordenador y su control de movimiento mediante un microcontrolador. Esta moto deberá de mantenerse estable durante el trayecto a velocidades medias-bajas.

Para ello contamos con un microcontrolador PIC32, con el cual se obtendrán los datos introducidos en el PC y a partir del procesamiento de estos datos, obtendremos los resultados en el movimiento de la moto.

Con los valores introducidos por el usuario, y usando la técnica PWM para el control de los motores, el usuario podrá describir con la moto el trayecto que considere necesario controlando la velocidad y el giro.

Todo el control viene estudiado por numerosas simulaciones y trabajos de campo para que conociendo los puntos máximos y mínimos a los que puede llegar el servo y el motor DC, se pueda mantener el equilibrio de la moto a velocidades medias-bajas.

Este equilibrio viene determinado por un giroscopio situado en la rueda trasera que proporcionará a la moto una estabilidad adicional basándose en el efecto giroscópico.

***Figura 1.1.-*** *Moto*

## 1.3. Fases del desarrollo

Conociendo el objetivo a realizar, la primera fase a desarrollar fue hacer un trabajo de estudio e investigación para llevar a cabo el proyecto. Este estudio se basó en la búsqueda de proyectos ya existentes, comparando las diferentes motos del mercado y sus características para conseguir las mejores soluciones.

Se decidió comprar una moto de radio control por piezas modelo ARX-540 [4]. Una vez conseguidas las piezas de la moto, la siguiente fase consistió en la construcción mecánica de la misma siguiendo su manual [1]. Se comenzó por la rueda delantera, continuando por la rueda trasera junto el giroscopio y todo su carenado. A continuación se montó el cuerpo de la moto, donde se colocaron los dos motores necesarios, motor brushless y servomotor. Todo el montaje se realizó detenidamente para que cada pieza quedara bien ajustada y en un futuro no diera ningún problema de estabilidad, ya que el más mínimo error de construcción podría hacer que la moto se desestabilice. Finalmente se realizo la conexión de todas las partes y su final puesta a punto.

Una vez construida, se decidió realizar una serie de pruebas con la moto totalmente radiocontrolada para conocer y estudiar su funcionamiento antes de controlarla mediante un microcontrolador. Las pruebas nos hicieron ver a que velocidades podíamos trabajar con ella y controlar el motor BRUSHLESS y los valores de servo para los que la moto funciona correctamente. Después, mediante su visión en un osciloscopio, poder saber los valores de PWM que necesitaríamos utilizar para su futuro control.

El siguiente paso a seguir fue la programación de nuestro microcontrolador basándonos en sus salidas de PWM y el estudio de la programación del PIC32. Este microcontrolador contiene 5 salidas PWM de las cuales se utilizan 2 en el proyecto, OC1 para controlar el servomotor y OC3 para controlar el motor brushless. La programación se realiza en C, mediante el programa MPLAB, que nos permite escribir y compilar el código para poder volcarlo en nuestro microcontrolador. Esto se realiza mediante la conexión USB entre el microcontrolador y el PC. Una vez depurado y descargado el código basándonos en los datos conseguidos en la prueba de radio control, observamos en un osciloscopio si las funciones de PWM se correspondían con los valores que necesitábamos para el control de la moto de automodelismo. Se extrajo

la parte de radio control y realizamos diferentes pruebas hasta conseguir que todos los valores coincidieran.

Para realizar el conexionado de los motores con el PIC se tuvo que resolver un problema. El microcontrolador se alimenta con una batería de 9 V y mediante éste conseguimos alimentar el servomotor con una de sus salidas de 5 V. El problema estaba en el motor brushless, ya que se alimenta a través del variador mediante una batería de 3300 mAh. Al ser un motor brushless de gran potencia, cuando acelera o frena introduce grandes picos de corriente que al entrar en la tarjeta del PIC producen un fallo en la misma.

Conseguimos solucionar este problema mediante el montaje de un circuito intermedio entre el variador y el microcontrolador. Este circuito consiste en un puente H. Este pequeño circuito electrónico permite alimentar el motor brushless con una tensión distinta a los 5 V que emite el PIC, y controlar el motor en ambos sentidos, avance y retroceso. Además consta de 4 diodos que nos permiten controlar los picos de corriente para que así, no lleguen a estropear nuestro microcontrolador que no los aguantaría.

Resuelto el problema, realizamos el cableado entre nuestros motores y el PIC, integrando toda la parte electrónica en la moto y terminando así el diseño y montaje electrónico.

Finalmente, se volvieron a realizar pruebas a pie de campo, probando varios programas realizados previamente para comprobar que la moto funciona de forma correcta y es capaz de realizar de manera autónoma las trayectorias deseadas.

Para la correcta puesta en marcha, es necesario activar primeramente el interruptor encargado de accionar el giróscopo debido a que necesita 40 segundos para trabajar a su máximo rendimiento. Pasado ese tiempo se alimentará el microcontrolador para que comience a ejecutar el programa previamente descargado.

## 1.4. Estructura de la memoria

El presente proyecto se encuentra dividido en 7 capítulos, los cuales se van a describir a continuación.

- Capitulo 1: Se describen los objetivos, las fases del proyecto y los medios empleados.
- Capitulo 2: Se habla de la historia relacionada con el mundo de la robótica, los robots móviles y las motos de RC.
- Capitulo 3: Se habla del microcontrolador empleado y los medios utilizados para su programación.
- Capitulo 4: Se describe el programa utilizado para realizar la programación además de una explicación de cómo crear, editar y ejecutar el programa creado.
- Capitulo 5: Se habla de las funciones, códigos y utilidades que poseen los osciladores y los timers en el proyecto. Además se calculan los datos para la futura programación.
- Capitulo 6: Se describe el programa creado y las funciones con las que lo hemos realizado.
- Capitulo 7: Se describen los programas descargados sobre el microcontrolador y los distintos comportamientos que tiene la moto con cada uno de ellos. Además se muestran las simulaciones realizadas con el osciloscopio para ver las PWM.
- Capitulo 8: conclusiones y trabajos futuros
- Capitulo 9: costes del proyecto

# Capítulo 2.

# Estado del arte

## 2.1. Introducción a la robótica

Según la Real Academia Española se define robot como: máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas sólo a las personas [7].

El término procede de la palabra checa robota, que significa 'trabajo obligatorio'; fue empleado por primera vez en la obra teatral R.U.R. (Robots Universales de Rossum), estrenada en Enero de 1921 en Praga por el novelista y dramaturgo checo Karel Capek. La obra fue un éxito inmediato y pronto se estrenó en multitud de teatros por toda Europa y Estados Unidos. En ella, el gerente de una fábrica construía unos seres al absoluto servicio del hombre, que realizaban todas las tareas mientras los humanos se dedicaban al ocio permanente. Cuando el gerente de la fábrica decide construir robots más perfectos que experimentaran felicidad y dolor, todo cambia. Los robots se sublevan contra los hombres y destruyen al género humano.

El concepto de máquinas automatizadas se remonta a la antigüedad, con mitos de seres mecánicos vivientes. Los autómatas, o máquinas semejantes a personas, ya aparecían en los relojes de las iglesias medievales, y los relojeros del siglo XVIII eran famosos por sus ingeniosas criaturas mecánicas.

El control por realimentación, el desarrollo de herramientas especializadas y la división del trabajo en tareas más pequeñas que pudieran realizar obreros o máquinas fueron ingredientes esenciales en la automatización de las fábricas en el siglo XVIII. A medida que mejoraba la tecnología se desarrollaron máquinas especializadas para tareas como poner tapones a las botellas o verter caucho líquido en moldes para neumáticos. Sin embargo, ninguna de estas máquinas tenía la versatilidad del brazo humano, y no podía alcanzar objetos alejados y colocarlos en la posición deseada.

En la década de 1890 el científico Nikola Tesla, inventor, entre muchos otros dispositivos, de los motores de inducción, ya construía vehículos controlados a distancia por radio. Tesla fue un visionario que escribió sobre mecanismos inteligentes tan capaces como los humanos.

Las máquinas más próximas a lo que hoy en día se entiende como robots fueron los "teleoperadores", utilizados en la industria nuclear para la manipulación de sustancias radiactivas. Básicamente se trataba de servomecanismos que, mediante sistemas mecánicos, repetían las operaciones que simultáneamente estaba realizando un operador.

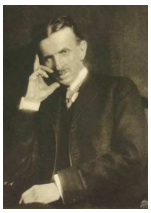


***Figura 2.1.-*** *Nikola Tesla*

Inmediatamente después de la Segunda Guerra Mundial comienzan los primeros trabajos que llevan a los robots industriales. A finales de los 40 se inician programas de investigación en los laboratorios de Oak Ridge y Argonne National Laboratories para desarrollar manipuladores mecánicos para elementos radiactivos. Estos manipuladores eran del tipo "maestro-esclavo", diseñados para que reprodujeran fielmente los movimientos de brazos y manos realizados por un operario.

El inventor estadounidense George C. Devol desarrolló en 1954 un dispositivo de transferencia programada articulada (según su propia definición); un brazo primitivo que se podía programar para realizar tareas específicas.

En 1958, Devol se unió a Joseph F. Engelberger y, en el garaje de este último, construyeron un robot al que llamaron Unimate. Era un dispositivo que utilizaba un computador junto con un manipulador que conformaba una "máquina" que podía ser "enseñada" para la realización de tareas variadas de forma automática. En 1962, el primer Unimate fue instalado a modo de prueba en una planta de la General Motors para funciones de manipulación de piezas y ensamblaje, con lo que pasó a convertirse en el primer robot industrial. Devol y Engelberger fundarían más tarde la primera compañía dedicada expresamente a fabricar robots, Unimation, Inc., abreviación de Universal Automation

Se puede considerar este punto como el inicio de la era de la Robótica tal como la conocemos, mediante la utilización de los robots programados, una nueva y potente herramienta de fabricación.

Durante la década de los 60, un nuevo concepto surge en relación con los anteriores avances. En vistas a una mayor flexibilidad, se hace necesaria la realimentación sensorial. En 1962, H. A. Ernst publica el desarrollo de una mano mecánica controlada por computador con sensores táctiles llamada MH-1. Este modelo evolucionó adaptándole una cámara de televisión dentro del proyecto MAC. También en 1962, Tomovic y Boni desarrollan una mano con un sensor de presión para la detección del objeto que proporcionaba una señal de realimentación al motor.

En 1963 se introduce el robot comercial VERSATRAN por la American Machine and Foundry Company (AMF). En el mismo año se desarrollan otros brazos manipuladores como el Roehampton y el Edinburgh.

En 1967 y 1968 Unimation recibe sus primeros pedidos para instalar varios robots de la serie Unimate 2000 en las cadenas de montaje de la General Motors. Al año siguiente los robots ensamblaban todos los coches Chevrolet Vega de esta compañía.

En 1968 se publica el desarrollo de un computador con "manos", "ojos" y "oídos" (manipuladores, cámaras de TV y micrófonos) por parte de McCarthy en el Stanford Artificial Intelligence Laboratory. En el mismo año, Pieper estudia el problema cinemático de un manipulador controlado por un computador. También este año, la compañía japonesa Kawasaki Heavy Industries negocia con Unimation la licencia de sus robots. Este momento marca el inicio de la investigación y difusión de los robots industriales en Japón.

En 1969 se demuestran las propiedades de la visión artificial para vehículos autoguiados en el Stanford Research Institute. Este mismo año se desarrollaron los brazos Boston y Stanford, este último dotado de una cámara y controlado por computador. Sobre el brazo Stanford se desarrolló un experimento en el que el manipulador apilaba bloques según determinados criterios.

Las primeras aplicaciones industriales en Europa, aplicaciones de robots industriales en cadenas de fabricación de automóviles, datan de los años 1970 y 1971. En este último año, Kahn y Roth analizan el comportamiento dinámico y el control de un brazo manipulador.

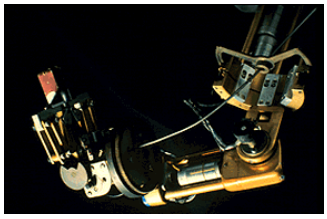


***Figura 2.2.-*** *Brazo Stanford*

Durante la década de los 70, la investigación en robótica se centra en gran parte en el uso de sensores externos para su utilización en tareas de manipulación. Es también en estos años cuando se consolida definitivamente la presencia de robots en las cadenas de montaje y plantas industriales en el ámbito mundial.

En 1972 se desarrolló en la universidad de Nottingham, Inglaterra, el SIRCH, un robot capaz de reconocer y orientar objetos en dos dimensiones. Este mismo año, la empresa japonesa Kawasaki instala su primera cadena de montaje automatizada en Nissan, Japón, usando robots suministrados por Unimation, Inc.

En 1973, Bolles y Paul utilizan realimentación visual en el brazo Stanford para el montaje de bombas de agua de automóvil. También este mismo año, la compañía sueca ASEA (futura ABB), lanza al mercado su familia de robots IRB 6 e IRB 60, para funciones de perforación de piezas.

En 1974, Nevins y sus colaboradores, en el Draper Laboratory, investigan técnicas de control basadas en la coordinación de fuerzas y posiciones, y Bejczy, en el Jet Propulsion Laboratory, desarrolla una técnica para el control de par basada en el robot Stanford. El mismo año, Inoue, en el Artificial Intelligence Laboratory del MIT, desarrolla trabajos de investigación en los que aplica la inteligencia artificial en la realimentación de fuerzas.

También este mismo año, la empresa Cincinatti Milacron introduce el T3 (The Tomorrow Tool), su primer robot industrial controlado por computador. Este manipulador podía levantar más de 100 libras y seguir objetos móviles en una línea de montaje.

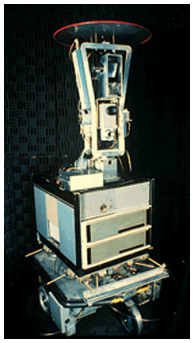


***Figura 2.3.-*** *Shakey, el primer vehículo autoguiado controlado por inteligencia artificial*

En 1975, Will y Grossman, en IBM, desarrollaron un manipulador controlado por computador con sensores de contacto y fuerza para montajes mecánicos. Este mismo año, el ingeniero mecánico estadounidense Victor Scheinman, cuando estudiaba la carrera en la Universidad de Stanford, California, desarrolló un manipulador polivalente realmente flexible conocido como Brazo Manipulador Universal Programable (PUMA, siglas en inglés). El PUMA era capaz de mover un objeto y colocarlo en cualquier orientación en un lugar deseado que estuviera a su alcance. El concepto básico multiarticulado del PUMA es la base de la mayoría de los robots actuales.

En 1976, estudios sobre el control dinámico llevados a cabo en los laboratorios Draper, Cambridge, permiten a los robots alinear piezas con movimientos laterales y rotacionales a la vez.

En 1979 Japón introduce el robot SCARA (Selective Compliance Assembly Robot Arm), y la compañía italiana DEA (Digital Electric Automation), desarrolla el robot PRAGMA para la General Motors.

En la década de los 80 se avanza en las técnicas de reconocimiento de voz, detección de objetos móviles y factores de seguridad. También se desarrollan los primeros robots en el campo de la rehabilitación, la seguridad, con fines militares y para la realización de tareas peligrosas. Así por ejemplo, en 1982, el robot Pedesco, se usa para limpiar un derrame de combustible en una central nuclear. También se pone un gran énfasis en los campos de visión artificial, sensorización táctil y lenguajes de programación. Gracias a los primeros pasos dados por compañías como IBM o Intelledex Corporation, que introdujo en 1984 el modelo ligero de ensamblaje 695, basado en el microprocesador Intel 8087 y con software Robot Basic, una modificación del Microsoft Basic, actualmente se tiende al uso de una interfaz (el ordenador) y diversos lenguajes de programación especialmente diseñados, que evitan el "cuello de

botella" que se producía con la programación "clásica". Esta puede ser ahora on-line u off-line, con interfaces gráficas (user-friendly interfaces) que facilitan la programación, y un soporte SW+HW que tiende a ser cada vez más versátil.

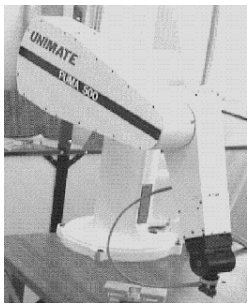



***Figura 2.4.-*** *Robot PUMA*

## 2.2. Clasificación de los robots

Actualmente es posible encontrar una variada gama de formas de clasificar a los robots [8]. A continuación ofrecemos un completo detalle.

### 2.2.1. Arquitectura

- **Androides:** Los androides son robots que se parecen y actúan como seres humanos. Los robots de hoy en día vienen en todas las formas y tamaños, pero a excepción de los que aparecen en las ferias y espectáculos, no se parecen a las personas y por tanto no son androides. Actualmente, los androides reales sólo existen en la imaginación y en las películas de ficción.


***Figura 2.5.-*** *Robot Androide*

- **Móviles:** Los robots móviles están provistos de patas, ruedas u orugas que los capacitan para desplazarse de acuerdo su programación. Elaboran la información que reciben a través de sus propios sistemas de sensores y se emplean en determinado tipo de instalaciones industriales, sobre todo para el transporte de mercancías en cadenas de producción y almacenes. También se utilizan robots de este tipo para la investigación en lugares de difícil acceso o muy distantes, como es el caso de la exploración espacial y las investigaciones o rescates submarinos.

***Figura 2.6.-*** *Robot móvil*

- **Zoomórficos:** Robots caracterizados principalmente por sus sistema de locomoción que imita a diversos seres vivos. Los androides también podrían considerarse robots zoomórficos.

***Figura 2.7.-*** *Robot zoomórfico*

- **Médicos:** Los robots médicos son, fundamentalmente, prótesis para disminuidos físicos que se adaptan al cuerpo y están dotados de potentes sistemas de mando. Con ellos se logra igualar con precisión los movimientos y funciones de los órganos o extremidades que suplen.

***Figura 2.8.-*** *Robot médico*

Cabe decir que pese a que la clasificación anterior es la más conocida, existe otra no menos importante donde se tiene en cuenta la potencia del software en el controlador, lo que es determinante de la utilidad y flexibilidad del robot dentro de las limitantes del diseño mecánico y la capacidad de los sensores. Los robots han sido clasificados de acuerdo a su generación, a su nivel de inteligencia, a su nivel de control, y a su nivel de lenguaje de programación.

### 2.2.2. Generación

La generación de un robot se determina por el orden histórico de desarrollos en la robótica. Cinco generaciones son normalmente asignadas a los robots industriales. La tercera generación es utilizada en la industria, la cuarta se desarrolla en los laboratorios de investigación, y la quinta generación es un gran sueño.

- **Robots de 1º Generación**
  El sistema de control usado en la primera generación de robots está basado en la "paradas fijas" mecánicamente. Como ejemplo de esta primera etapa están los mecanismos de relojería que mueven las cajas musicales o los juguetes de cuerda.
- **Robots de 2º Generación**
  El movimiento se controla a través de una secuencia numérica almacenada en disco o cinta magnética. Por regla general, este tipo de robots se utiliza en la industria automotriz y son de gran tamaño.
- **Robots de 3º Generación**
  Utilizan las computadoras para su control y tienen cierta percepción de su entorno a través del uso de sensores. Con esta generación se inicia la era de los robots inteligentes y aparecen los lenguajes de programación para escribir los programas de control.

- **Robots de 4º Generación**
  Se trata de robots altamente inteligentes con más y mejores extensiones sensoriales, para entender sus acciones y captar el mundo que los rodea. Incorporan conceptos "modélicos" de conducta.
- **Robots de 5º Generación**
  Actualmente en desarrollo. Esta nueva generación de robots basará su acción principalmente en modelos conductuales establecidos.

También podría decirse que dentro del aspecto generacional nos encontramos con:

- **Robots Play-back**, los cuales regeneran una secuencia de instrucciones grabadas, como un robot utilizado en recubrimiento por spray o soldadura por arco. Estos robots comúnmente tienen un control de lazo abierto.

***Figura 2.9.-*** *Robot playback*

- **Robots controlados por sensores**, éstos tienen un control en lazo cerrado de movimientos manipulados, y toman decisiones basados en datos obtenidos por sensores.
- **Robots controlados por visión**, donde los robots pueden manipular un objeto al utilizar información desde un sistema de visión.
- **Robots controlados adaptablemente**, donde los robots pueden automáticamente reprogramar sus acciones sobre la base de los datos obtenidos por los sensores.
- **Robots con inteligencia artificial**, donde los robots utilizan las técnicas de inteligencia artificial para hacer sus propias decisiones y resolver problemas.

### 2.2.3. Nivel de inteligencia

La Asociación de Robots Japonesa (JIRA) ha clasificado a los robots dentro de seis clases sobre la base de su nivel de inteligencia:

- **Dispositivos de manejo manual**, controlados por una persona.
- **Robots de secuencia arreglada.**
- **Robots de secuencia variable**, donde un operador puede modificar la secuencia fácilmente.
- **Robots regeneradores,** donde el operador humano conduce el robot a través de la tarea.
- **Robots de control numérico**, donde el operador alimenta la programación del movimiento, hasta que se enseñe manualmente la tarea.
- **Robots inteligentes**, los cuales pueden entender e interactuar con cambios en el medio ambiente.

### 2.2.4. Nivel de control

Los programas en el controlador del robot pueden ser agrupados de acuerdo al nivel de control que realizan en tres grupos:

- **Nivel de inteligencia artificial**.

  El programa aceptará un comando como "levantar el producto" y descomponerlo dentro de una secuencia de comandos de bajo nivel basados en un modelo estratégico de las tareas.

- **Nivel de modo de control**.

  Los movimientos del sistema son modelados, para lo que se incluye la interacción dinámica entre los diferentes mecanismos, trayectorias planeadas, y los puntos de asignación seleccionados.

- **Niveles de servosistemas**

  Los actuadores controlan los parámetros de los mecanismos con el uso de una retroalimentación interna de los datos obtenidos por los sensores, y la ruta es modificada sobre la base de los datos que se obtienen de sensores externos.

Todas las detecciones de fallas y mecanismos de corrección son implementados en este nivel.

### 2.2.5. Lenguaje de programación

La clave para una aplicación efectiva de los robots para una amplia variedad de tareas, es el desarrollo de lenguajes de alto nivel. Existen muchos sistemas de programación de robots, aunque la mayoría del software más avanzado se encuentra en los laboratorios de investigación. Los sistemas de programación de robots se dividen en tres clases:

- **Sistemas guiados**, donde el usuario conduce el robot a través de los movimientos a ser realizados.
- **Sistemas de programación de nivel-robot,** donde el usuario escribe un programa de computadora al especificar el movimiento y el sensado.
- **Sistemas de programación de nivel-tarea,** donde el usuario especifica la operación por sus acciones sobre los objetos que el robot manipula.

## 2.3. Robots móviles

Con la aparición de nuevas tecnologías de planificación y razonamiento automático, de 1966 a 1972 se desarrolló en el SRI el primer robot móvil llamado Shakey [Nilsson, 1984], que era una plataforma móvil independiente controlada por visión mediante una cámara y dotada con un detector táctil. A partir de ese momento, la investigación y diseño de robots móviles (que contaron con características muy diferentes entre ellos) creció de manera exponencial [9].

A principios de la década del setenta, el robot Newt [Hollis, 1977] fue desarrollado por Hollis. El robot Hilare [Giralt, 1979] desarrollado en el LAAS en Francia.

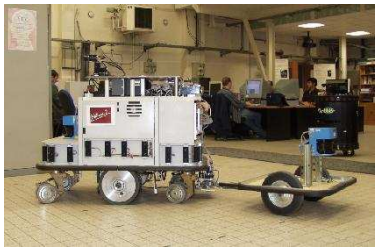


***Figura 2.10.-*** *Robot H ilare*

En el Jet Propulsion Laboratory (JPL) se desarrolló el Lunar rover [Thompson, 1977], diseñado particularmente para la exploración planetaria. A finales de esa década, Moravec desarrolló el robot Stanford cart [Movarec, 1979], capaz de seguir una trayectoria delimitada por una línea establecida en una superficie, en el SAIL. En 1983, el robot Raibert [Raibert, 1986], fue desarrollado en el MIT, un robot de una sola pata diseñado para estudiar la estabilidad de éstos sistemas. A principios de la década del noventa, Vos *et al.* desarrollaron un robot "uniciclo" [Vos et al., 1990] (una sola rueda, similar a la de una bicicleta) en el MIT.

Años más tarde, en 1994, el Instituto de robótica CMU desarrolló el robot Dante II [Bares et al., 1999], un sistema de seis patas.

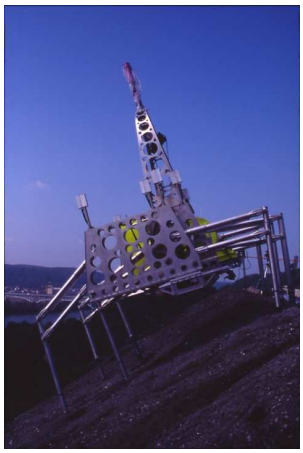


***Figura 2.11.-*** *Robot Dante II*

En 1996 también en el CMU, se desarrolló el robot Gyrover [Brown et al., 1997], un mecanismo ausente de ruedas y patas basado en el funcionamiento del giroscopio. Ese mismo año se desarrolló en el MIT el Spring Flamingo [Pratt et al., 1998], un robot que emulaba el movimiento de un flamingo. Por su parte, la NASA en 1997 envío a Marte un robot móvil teleoperado llamado Sojourner rover [Muirhead, 1997], dedicado a enviar fotografías del entorno de dicho planeta. Ese mismo año, la empresa japonesa

HONDA, dio a conocer el robot P3 [Tanie, 2003], el primer humanoide capaz de imitar movimientos del cuerpo humano. Al siguiente año, se desarrolla en la universidad Waseda en Japón, el WABIAN R-III [Hashimoto, 1998], un robot humanoide. En 1999 en el CMU, Zeglin propuso un nuevo diseño de robot con una pata llamado Bow Leg Hopper [Zeglin, 1999], un diseño que permite almacenar la energía potencial de la pata.

En 2006, Hollis *et al.* desarrollaron el robot Ballbot [Lauwers, 2006], un sistema holónomo cuyo movimiento es proporcionado por una sola esfera ubicada en la parte inferior de la estructura. Sin embargo, el estudio de este tipo de robots con una esfera, fue iniciado por Koshiyama y Yamafuji [Koshiyama et al., 1991] en 1991.

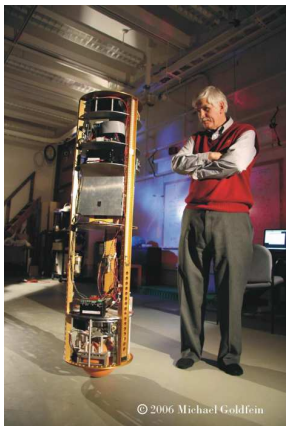



***Figura 2.12.-*** *Robot Ballbot*

Actualmente los robots teleoperados Spirit rover y Opportunity rover, [Aronson et al, 2001], se encuentran explorando la superficie del planeta Marte en busca de mantos acuíferos. Los robots aquí mencionados, son únicamente una porción de los tantos que se han diseñado, sin embargo, es posible notar que las aplicaciones de estos son vastas y que las mismas son ilimitadas debido al desarrollo cada vez más vertiginoso de la tecnología.

Los robots móviles brindan la posibilidad de navegar en distintos terrenos y tienen aplicaciones como: exploración minera, exploración planetaria, misiones de búsqueda y rescate de personas, limpieza de desechos peligrosos, automatización de procesos, vigilancia, reconocimiento de terreno, y también son utilizados como plataformas móviles que incorporan un brazo manipulador.

Los robots móviles se pueden clasificar por el tipo de locomoción utilizado, en general, los tres medios de movimiento son: por ruedas [Muir et al., 1992], por patas

[Todd, 1985] y orugas [Grasnoik et al., 2005]. Cabe señalar que aunque la locomoción por patas y orugas han sido ampliamente estudiadas, el mayor desarrollo se presenta en los Robots Móviles con Ruedas (RMR). Dentro de los atributos más relevantes de los RMR, destacan su eficiencia en cuanto a energía en superficies lisas y firmes, a la vez que no causan desgaste en la superficie donde se mueven y requieren un número menor de partes y menos complejas, en comparación con los robots de patas y de orugas, lo que permite que su construcción sea más sencilla.

Son precisamente estos argumentos los que motivan el análisis de este tipo de robots, y surge la necesidad, en primera instancia, de tener una definición que satisfaga el contexto de los RMR. De esta manera, se puede definir un robot móvil de ruedas como un sistema electromecánico controlado, que utiliza como locomoción ruedas de algún tipo, y que es capaz de trasladarse de forma autónoma a una meta preestablecida en un determinado espacio de trabajo.

Se entiende como autonomía de un robot móvil, al dominio que tiene éste para determinar su curso de acción, mediante un proceso propio de razonamiento en base a sensores, en lugar de seguir una secuencia fija de instrucciones. En lo referente a las partes de las que se compone un RMR, se tiene un arreglo cinemático y un sistema de actuadores. Ambos sistemas están íntimamente ligados y son dignos de estudiarse en conjunto, no obstante, se ha logrado un mayor avance, en el estado del arte, al estudiarlos por separado.

***Figura 2.13.-*** *Grupo de robots móviles*

### 2.3.1. Configuraciones cinemáticas de los RMR

Existen diferentes configuraciones cinemáticas para los RMR [Azcon, 2003], estas dependen principalmente de la aplicación hacia dónde va enfocado, no obstante, de manera general se tienen las siguientes configuraciones: Ackerman, triciclo clásico, tracción diferencial, skid steer, síncrona y tracción omnidireccional (figura 1).

Dependiendo de la configuración cinemática que lo conforme, los RMR utilizan cuatro tipos de ruedas para su locomoción [Goris, 2005], estas son: convencionales, tipo castor, ruedas de bolas y omnidireccionales, se pueden observar en la figura 2.

En el marco de las configuraciones cinemáticas posibles y las ruedas que estas utilizan, los RMR documentados en la literatura utilizan comúnmente la configuración de tracción diferencial, (figura 1c), donde se utilizan ruedas convencionales (figura 2a), como ruedas motrices y una o dos ruedas tipo castor, de bola, u omnidireccionales, (figuras 2b, 2c, 2d), respectivamente, para proveer de estabilidad al móvil.


***Figura 2.14.-*** *Configuración de los RMR*


***Figura 2.15.-*** *Ruedas utilizadas en los RMR*

Una variante de la configuración de tracción diferencial muy conveniente, que utiliza el robot Shakey, es la que cuenta con dos ruedas motrices, que reduce costos de

desarrollo y complejidad de control porque es suficiente un par de actuadores para lograr movimiento, también incorpora dos ruedas de bola, que se encuentran diametralmente opuestas y que además de brindar estabilidad al móvil, simplifican el movimiento de giro total a sólo invertir el sentido de giro de las ruedas motrices, por lo que no es necesaria propiamente la rotación y por lo tanto el control de ese movimiento es bastante más sencillo.

Con el objeto de hacer más tratable el problema del modelado en las configuraciones cinemáticas, se suelen establecer algunas suposiciones de diseño y de operación [Muir et al., 1992]. Por una parte, dentro de las suposiciones de diseño generalmente se toman tres. La primera va dirigida a considerar que las partes dinámicas del RMR son insignificantes i.e., que no contiene partes flexibles, de esta manera pueden aplicarse mecanismos de cuerpo rígido para el modelado cinemático. La segunda limita que la rueda tenga a lo más un eslabón de dirección, con la finalidad de reducir la complejidad del modelado. La tercera es asumir que todos los ejes de dirección son perpendiculares a la superficie, de esta manera se reducen todos los movimientos a un solo plano.

Por otra parte, respecto a las suposiciones de operación, al igual que en las de diseño, se toman tres. Una de ellas descarta toda irregularidad de la superficie donde se mueve el RMR. Otra, considera que la fricción de traslación en el punto de contacto de la rueda con la superficie donde se mueve, es lo suficientemente grande para que no exista un desplazamiento de traslación del móvil. Como complemento a lo anterior, una tercer suposición de operación establece que la fricción rotacional en el punto de contacto de la rueda con la superficie donde se mueve, es lo suficientemente pequeña para que exista un desplazamiento rotatorio.

Aunque las suposiciones mencionadas son realistas, el deslizamiento que ocurre en el punto de contacto de las ruedas con la superficie se ha convertido en un tópico importante debido a las repercusiones que tiene sobre el móvil.

### 2.3.2. Actuadores en los RMR

Relativo a los actuadores utilizados para dotar de movimiento a los RMR, es común que se utilicen motores. Existe una gama bastante amplia dependiendo de su empleo [Sandin, 2005], los más utilizados en la robótica móvil son los de corriente directa (CD), por el argumento de que su modelo es lineal, lo que facilita enormemente su control, y específicamente los de imán permanente debido a que el voltaje de control es aplicado al circuito de armadura y el circuito de campo es excitado de manera independiente.

Hablando de motores de CD de imán permanente, se tienen dos tipos: con escobillas y sin escobillas. Ambos tipos brindan ventajas semejantes, sin embargo, los motores sin escobillas tienen algunas ventajas significativas sobre los motores con escobillas, como por ejemplo:

- Al no contar con escobillas, no se requiere el reemplazo de éstas ni mantenimiento por residuos originados de las mismas.

- No presentan chispas que las escobillas generan, de esta forma se pueden considerar más seguros en ambientes con vapores o líquidos inflamables.

- La interferencia causada por la conmutación mecánica de las escobillas se minimiza considerablemente mediante una conmutación electrónica.

- Los motores sin escobillas alcanzan velocidades de hasta 50,000 rpm comparadas con las 5,000 rpm aprox. máximas de los motores con escobillas.

A pesar de que estas ventajas parecieran tender la balanza a favor de los motores sin escobillas, existen desventajas cruciales que pueden cambiar la tendencia:

- En los motores sin escobillas no se puede invertir el sentido de giro cambiando la polaridad de sus terminales, esto agrega complejidad y costo a su manejo.

- Los motores sin escobillas son más caros.

- Se requiere un sistema adicional para la conmutación electrónica.

- El controlador de movimiento para un motor sin escobillas es más costoso y complejo que el de su equivalente con escobillas. Al igual que en el arreglo cinemático, cuando se modela un motor de CD se asumen algunas consideraciones, de esta forma se establece que la única fricción presente es la viscosa, aunque en la práctica se involucran otros tipos de fricción no lineales, sin embargo, la suposición es válida al elegir un motor cuyo efecto de las fricciones no lineales sea muy pequeño.

### 2.3.3. Control de los RMR

En tiempos actuales, el tema del control de los RMR ha venido acaparando la atención de gran cantidad de investigadores. Desde el punto de vista de la teoría de control, estos se encuentran en el área que se conoce como control de sistemas no holónomos, estos sistemas se caracterizan por tener un número menor de grados de libertad controlables respecto al número de grados de libertad totales, en el caso de un RMR de tracción diferencial, el número total de grados de libertad son 3 (posición $x,y$ y su orientación $\varphi$) sin embargo únicamente se puede controlar el desplazamiento hacia adelante y hacia atrás así como su orientación, quedando como incontrolable el desplazamiento transversal.

Matemáticamente se dice que el sistema esta sujeto a restricciones no integrables en las velocidades, es decir, su plano de velocidades está restringido. El control del movimiento de los RMR, a grosso modo, se puede clasificar en cuatro tareas fundamentales; localización, planificación de trayectoria, seguimiento de la misma y evasión de obstáculos. Existen diversos trabajos donde se han estudiado estos tópicos de manera detallada, no obstante, sólo se mencionarán los más relevantes relativos.

## 2.4. Historia de las motos RC

Ya hacia 1980 aparecieron las primeras motos R/C (Graupner, Kyosho, etc), por entonces eléctricas y con variador constituido por un simple reóstato, sin fines competitivos [10].

Es en 1985 cuando una fábrica italiana (DWA) saca al mercado una moto escala 1/4 con motor de explosión de 3.5 cc, la DWA Comando, orientada a la competición.

Tiene un relativo éxito, sobre todo en Italia, y se disputaron campeonatos europeos entre 1986 y 1989, así como muchas competiciones en Italia y también en España.


***Figura 2.16.-*** *DWA Commando*

La moto DWA Comando con el tiempo manifestó importantes carencias o defectos, que fueron paliados o solucionados por sus usuarios:

- Sistema de dirección con dos servos que hacía la horquilla girar, además de en un eje casi vertical como el de una moto real, en un eje horizontal.
- Falta de un verdadero depósito que permitiese un rápido y cómodo repostaje.
- Amortiguador trasero deficiente.
- Corona de transmisión excesivamente grande, que rozaba en el suelo al inclinar la moto.
- Volante de inercia oculto que no permitía el arranque. Éste se hacía con un adaptador roscado en la punta del cigüeñal (que había que serrar). Se utilizaba un arrancador externo similar al de los coches, movido por batería.
- Carencia de freno y suspensión delanteros. Éstas fueron opciones de fábrica posteriores, si bien añadiendo un servo para el freno delantero. La moto llegó a requerir cuatro servos: dos para la horquilla, otro para carburador y freno trasero, y otro para freno delantero.

La principal evolución fue en el sistema de dirección, que se explicará con detalle más adelante. El resto evolucionó así:

- Depósito: se añadían verdaderos tapones cortando el depósito de serie, o éste se construía entero en latón.
- Amortiguadores: se adaptaron amortiguadores de coches de todo terreno, tanto el trasero como el delantero.
- Corona de transmisión: fue posible localizar coronas de coches de pista con el mismo anclaje y de menos dientes, así como el piñón correspondiente en su campana de embrague.

- Volante: el volante y el adaptador de arranque se sustituyeron por un volante de inercia usado en coches con forma de gorro chino, que posibilitaba el uso del mismo arrancador, sin necesidad de cortar el cigüeñal, y utilizar motores con cigüeñal tipo SG.
- Freno delantero: con el tiempo se vio su absoluta necesidad, pues la moto, sólo con el freno trasero, era simplemente imparable, y muy inestable si se bloqueaba la rueda trasera (se anulaba el giróscopo trasero). La opción de fábrica que requería un servo adicional se modificó, siendo un único servo el que accionaba carburador y frenos trasero y delantero, éste último por cable. El disco delantero único de la opción de fábrica se pudo sustituir con dos discos finos y tres pastillas.

La principal evolución fue en el sistema de dirección, que fue radicalmente transformado hasta llegar al utilizado en las motos R/C actuales. El sistema de dirección inicial hacía girar, mediante dos servos en el canal de dirección, la horquilla, además de en un eje casi vertical como el de una moto real, en un eje horizontal: la horquilla hacía un "barrido". De hecho, el neumático delantero era macizo, buscando un alto momento de inercia, con forma de pico, y su eje de giro estaba atrasado respecto a los brazos de la horquilla. Al hacer la horquilla su movimiento de barrido, en la práctica, debido al elevado momento cinético de la rueda delantera, ésta mantenía su verticalidad y era el resto de masas de la moto el que se desplazaba, moviendo una masa que tumbaba la moto y, con el movimiento normal de la horquilla en su eje casi vertical, provocaba que la moto tumbase y girase.

El sistema era efectivo pero requería un ajuste continuo: había que "apretar" los servos contra la pieza que movía la horquilla para evitar holgura en el movimiento de barrido, lo que provocaba un fuerte consumo de corriente y requería servos de alto par. Además, la moto resultaba perezosa: costaba tumbarla y levantarla, y el sistema no tenía ajuste posible. Una mejora fue añadir en la parte alta de la horquilla una chapa con corte en V, y un alambre de acero doblado a 90º solidario al chasis que finalizaba en el hueco de la V: al girar y realizar la horquilla el movimiento de barrido, la V alcanzaba el alambre y hacía girar la horquilla al contrario del giro pretendido, lo que contribuía a tumbar la moto. Con el grosor del alambre y la posición de la V se tenía un cierto ajuste.

En 1987 el danés J. Jorgensen da con una idea más sencilla, y que, a pesar de ser un sistema muy diferente, era fácil adaptar al sistema de serie:

- Bloquea el movimiento de barrido de la horquilla (eje horizontal), sin más que invertir una pieza de la moto.
- Sustituye los dos servos de dirección por un único servo con dos varillas a la horquilla, que la hacen girar mediante dos muelles, ajustables con collarines.
- Para girar la moto hacia un lado el servo de dirección hace girar la horquilla al lado contrario y la moto cae del lado al que queremos girar. Los muelles permiten que la horquilla quede en su ángulo correcto una vez que la moto ha tumbado.

El sistema es definitivo: se terminan los problemas de consumo y holguras de dirección, y ésta es ajustable (muelles blandos para aprender, si la moto se nota nerviosa, o para circuitos de curvas amplias, y se endurecen los muelles mediante los collarines si se tiene experiencia o en circuitos con muchas curvas). El sistema de muelles con los que el servo de dirección mueve la horquilla constituye un salvaservos natural. El servo de dirección no requiere ser de par alto y puede incluso utilizarse un miniservo.

En 1988 se mejora el sistema de Jorgensen con una pieza nueva que constituye un salvahorquillas y que evita debilidades estructurales en la modificación de Jorgensen (en las motos actuales se conoce como "cabeza del caballo"). El servo de dirección se baja, así como las varillas a la horquilla.

El sistema de Jorgensen y el salvahorquillas han prevalecido hasta hoy día, normalmente con varilla única al servo de dirección, que aloja los dos muelles, y con amortiguador con aceite para amortiguar las vibraciones de la horquilla, que puede ser:

- Varilla con muelles con el pistón del amortiguador solidario con la varilla.
- Varilla y amortiguador independientes.

Salvo la aparición de neumáticos PMT para la moto DWA Comando, ligeramente mejores que los de serie, ésta no evolucionó más y dejó de fabricarse. Su mérito histórico fue ser la base de desarrollo y mejoras durante tres años de la moto R/C hasta las motos actuales.

En la década de los 90, y del 2000 en adelante aparecen otros fabricantes de motos (Nuova Faor, Thunder Tiger, AR Modelling, KP, etc), que redefinen la moto actual:

- Se abandona la escala 1/4 para motos de pista, y casi todas las motos de pista fabricadas son de escala 1/5.
- En motos de explosión se abandona el motor de 3.5 cc en favor del de 2.1 cc (0.12 c.u.). Para simplificar el arranque, al cigüeñal se le incorpora un rodamiento "one-way" y el arranque puede ser:
    - Por tirador.
    - Por "rotostart": un motor eléctrico externo con reductora, movido por un paquete de seis elementos de NiCd o NiMH similar al utilizado en tracción de coches eléctricos.
- Frenos movidos por cable.
- Se populariza la moto eléctrica, también en escala 1/5, más simple al no llevar embrague, escape, carburación, depósito, sistema de arranque, etc. Se populariza el uso de motores sin escobillas (no utilizando marcha atrás) y baterías LiPo. El freno en las motos eléctricas puede ser sólo eléctrico a la rueda trasera (la moto sólo llevaría un servo para la dirección), o además mecánico, requiriendo otro servo, y actuando sólo sobre la rueda delantera, o sobre ambas ruedas delantera y trasera.
- Aparecen realizaciones artesanales con motores de explosión de cuatro tiempos.
- Aparecen motos para circuitos de tierra en escala 1/4, tanto eléctricas (2007) como de explosión (2008).
- Aparecen motos eléctricas para pista en escala 1/8.


***Figura 2.17.-*** *Thunder Tiger escala 1:5*

En junio de 2006 se celebra en Brookland (Inglaterra) el primer campeonato del mundo de motos R/C de pista escala 1/5, que es seguido en septiembre de 2007 por el celebrado en sl Miniautódromo Jody Schekter (Fiorano, Italia) (ver resultados) en sus tres categorías ("Stockbike", "Superbike" y "Nitrobike"), aunque casi todos los participantes son europeos. En España se celebra en 2008 el primer campeonato nacional de motos R/C de pista en el circuito de Cerdanyola y el mundial de 2008 se celebra en el circuito Autet.

# Capítulo 3.

# Microcontrolador PIC32

# 3.1. Introducción

La MCU PIC32 es una familia de microcontroladores de 32 bits diseñados para un rendimiento de 32 bits, siendo la mejor en su clase y acompañado de una amplia gama de software. Desde su introducción en 2007, la familia de PIC32 se ha establecido como líder en rendimiento derivados del más alto DMIPs/MHz nominal MIPS M4K de núcleo, la arquitectura de bus interno altamente eficiente y el almacenamiento en caché de avanzada de instrucciones. El PIC32 ofrece una gama de general y periféricos, incluyendo Ethernet, CAN y USB host. Va desde la memoria flash integrada de 32 K a 512 K y rangos de RAM a bordo de 8 k a 128 k.

La familia de PIC32 es compatible con la del microchip MPLAB, entorno de desarrollo y proveedores de software y herramientas de la industria conocida. Los clientes de microchip se benefician de la amplia oferta de bibliotecas de software libre, comúnmente disponibles en forma de código fuente. La mayoría de software es compatible a través de numerosos microcontroladores de microchip y familias de procesadores de señal digital. El entorno de desarrollo de MPLAB abarca toda la gama de microchip de más de 600 microcontroladores [1].

## 3.1.1. Soporte de herramientas de desarrollo de microchip

- MPLAB IDE
- MPLAB C Compiler
- Emulador REAL ICE  en circuito MPLAB
- MPLAB ICD 3 en circuito depurador
- Programador de PICkit 3
- Programador de dispositivos universal MPLAB PM3

## 3.1.2. Funciones integradas de MCU

- Ethernet 10/100, CAN2.0b, USB host/dispositivo/OTG
- Controlador de interrupciones de vector anidados
- ADC de 10 bits, 1Msps y +/-1 LSB
- puerto principal paralelo de 16-bit para agregar QVGA & memoria
- POR, BOR, LVD, Pull-ups
- 2.3 - 3.6v operación, tolerante 5v de E/s

### 3.1.3. Recursos de software de microchip (código de fuente libre)

- Pilas USB host y dispositivos
- Biblioteca de gráficos y audio
- Pila de TCP/IP con SSL
- Sistema de archivos de 16 y 32 bits
- CAN software


***Figura 3.1.-*** *PIC 32*


***Figura 3.2.-*** *StarterKit PIC 32*

## 3.2. Expansión Board

La tarjeta de expansión I/O de PIC32 proporciona al Starter Kit un acceso completo a las señales MCU (Multipoint Control Unit), debugers adicionales y una conexión para tarjetas Plus de PICtail. Las señales MCU están disponibles para la conexión de circuitos prototipo o monitorizar señales.

El Starter Kit del PIC 32 es necesario para ejecutar el código de la aplicación. La tarjeta de expansión no dispone de MCU, la cual es la encargada de gestionar la comunicación de datos entre diferentes terminales.

El Starter Kit del PIC32 puede proporcionar energía a la tarjeta de expansión mediante el conector J1, ya que se alimenta a través del puerto USB que dispone. En nuestro caso colocaremos una alimentación adicional para su funcionamiento autónomo con una fuente de alimentación de 9V.

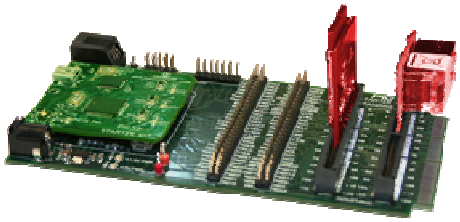


***Figura 3.3.-*** *PIC 32 I/O Expansion Board*

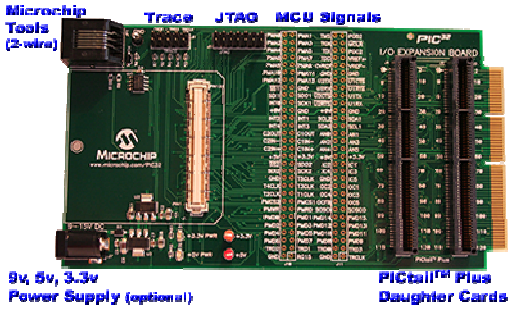



***Figura 3.4..-*** *Conexiones Expansion Board*

# Capítulo 4.

# MPLAB

## 4.1. Introducción

MPLAB es un editor IDE gratuito, destinado a productos de la marca Microchip. Este editor es modular, permite seleccionar los distintos microprocesadores, además de permitir la grabación de estos circuitos integrados directamente al programador.

Es un programa que corre bajo Windows y como tal, presenta las clásicas barras de programa, de menú, de herramientas de estado, etc.

Permite escribir el programa para los PIC en lenguaje ensamblador (assembler) o en C, crear proyectos, ensamblar o compilar, simular el programa y finalmente programar el componente.

En la compilación el MPLAB nos generara un archivo de extensión .hex el cual es completamente entendible para el PIC. Es decir, solo resta grabarlo al PIC por medio de una interfaz. Una vez completado esto, se alimenta al mismo y el programa ya se estará ejecutando.

Una vez realizado esto, se está en condiciones de empezar a escribir el programa respetando las directivas necesarias y la sintaxis para luego compilarlo y grabarlo en el PIC.

MPLAB incorpora todas las utilidades necesarias para la realización de cualquier proyecto y, para los que no dispongan de un emulador, el programa permite editar el archivo fuente en lenguaje ensamblador de nuestro proyecto, además de ensamblarlo y simularlo en pantalla, pudiendo ejecutarlo posteriormente en modo paso a paso y ver como evolucionarían de forma real tanto sus registros internos, la memoria RAM y/o EEPROM de usuario como la memoria de programa, según se fueran ejecutando las instrucciones. Además el entorno que se utiliza es el mismo que si se estuviera utilizando un emulador [2].

Las funciones principales de MPLAB-IDE son:

- **EDITOR:** Editor incorporado que permite escribir y editar programas u otros archivos de texto.
- **PROJECT MANAGER:** Organiza los distintos archivos relacionados con un programa en un proyecto. Permite crear un proyecto, editar y simular un programa. Además crea archivos objetos y permite bajar archivos hacia emuladores (MPLAB-ICE) o simuladores de hardware (SIMICE).

- **SIMULADOR:** Simulador de eventos discretos que permite simular programas con ilimitados breakpoint, examinar/modificar registros, observar variables, tiempos y simular estímulos externos.
- **ENSAMBLADOR:** Genera varios tipos de archivos objetos y relacionados, para programadores Microchip y universales.
- **LINKER:** Permite unir varios archivos objetos en uno solo, generados por el ensamblador o compiladores C.
- **PROGRAMADOR:** Mplab-IDE puede trabajar con varios tipos de programadores. El usuario debe seleccionar con cual trabajará, haciendo click en opción Programmer/ Select programmer, se pueden seleccionar 4 programadores distintos, y para nuestro proyecto utilizaremos el primero de ellos:
    - PICSTART Plus
    - MPLAB ICD 2
    - MPLAB PM 3
    - PRO MATE II

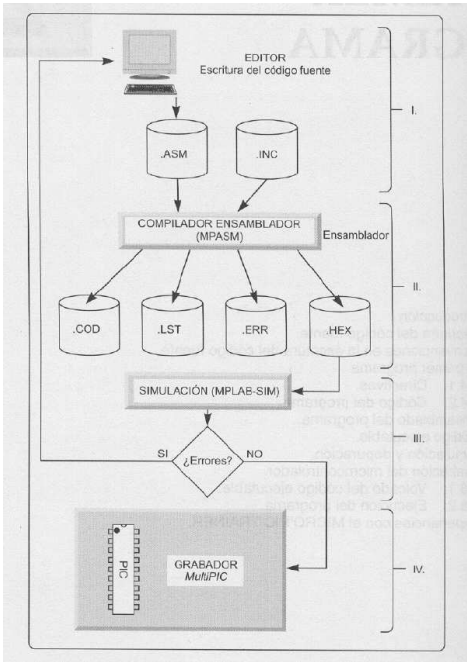



***Figura 4.1.**-Esquema del MPLAB*

En este apartado del proyecto se muestra paso a paso como crear un proyecto mediante el programa MPLAB para PIC32, como compilarlo con el compilador C32 y descargar y ejecutar nuestro programa en la placa Starter Kit [12].

## 4.2. Crear un proyecto

1. El primer paso a realizar para crear un proyecto es abrir el MPLAB y seleccionar en la barra de herramientas "Project --> Project Wizard..."

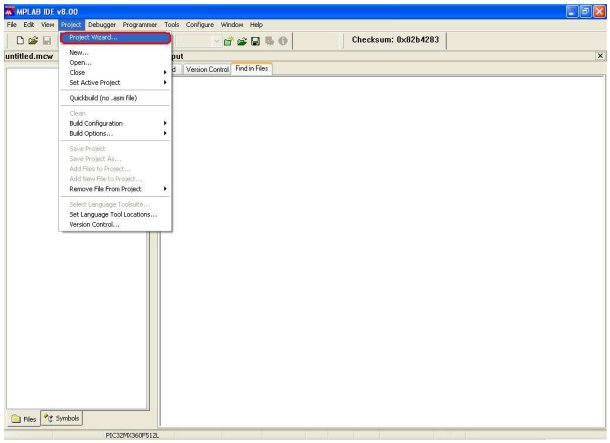



***Figura 4.2.-*** *Pantalla principal MPLAB*

2. A continuación se pulsa "Siguiente" para iniciar la creación del proyecto con el asistente.

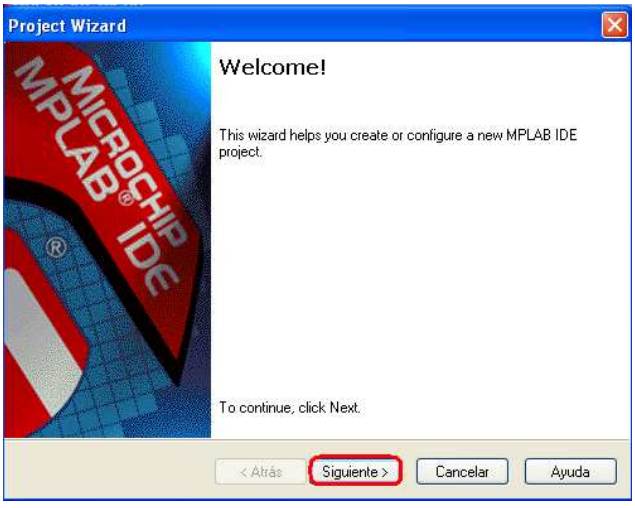



***Figura 4.3.-*** *Ventana inicial para crear proyecto*

3. En la ventana de la figura X Se selecciona el tipo de PIC, en este caso "PIC32MX360F512L".

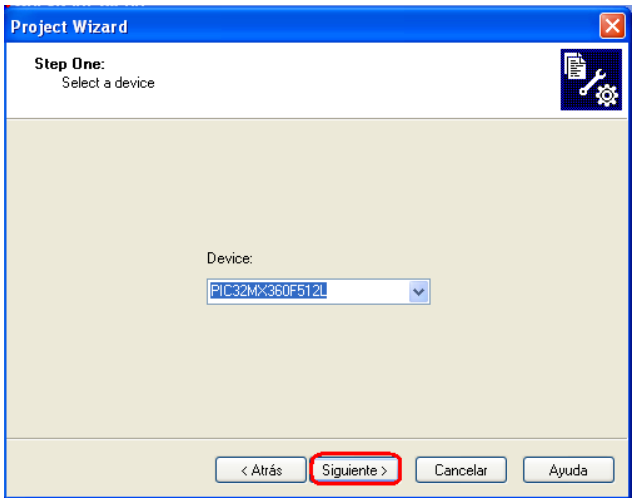



***Figura 4.4.-*** *Ventana de selección de PIC*

4. Se selecciona el compilador C32 como se observa en la figura X y pinchamos en "Siguiente".

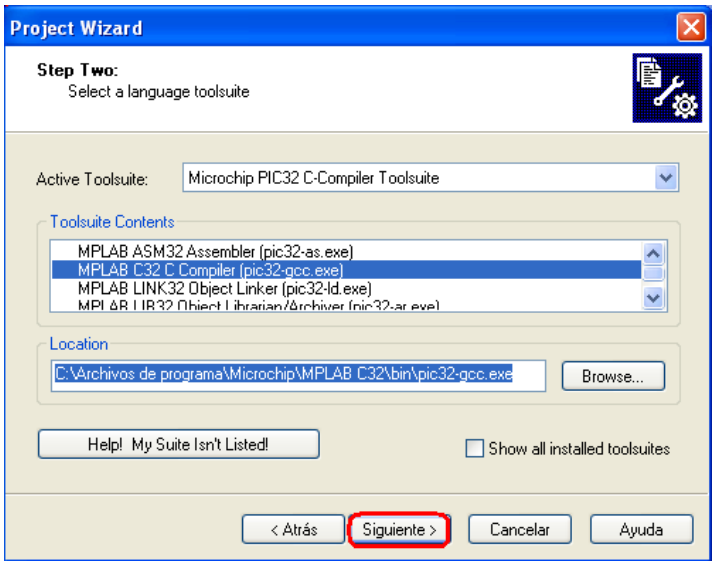



***Figura 4.5.-*** *Ventana de selección de compilador*

5. El siguiente paso a realizar es escribir el nombre del proyecto y escoger el directorio donde se quiere guardar.

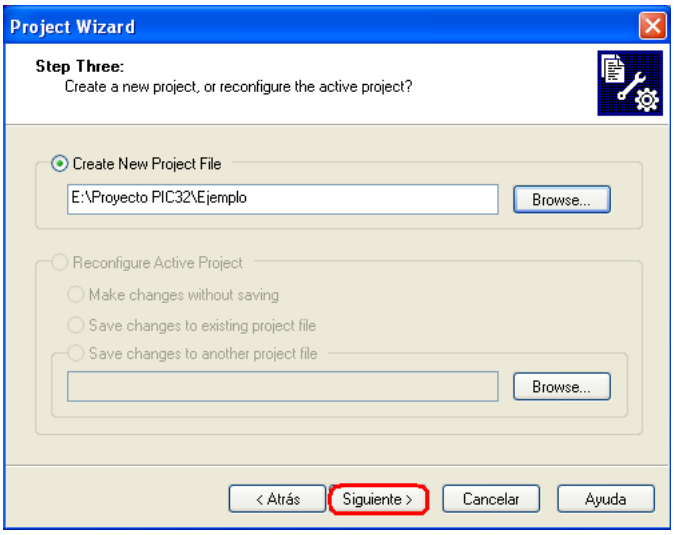



***Figura 4.6.-*** *Ventana de guardado de proyecto*

6. Se añaden los archivos existentes al proyecto. Como se va a crear un proyecto desde cero, no se añade ninguno y se pulsa sobre "Siguiente".

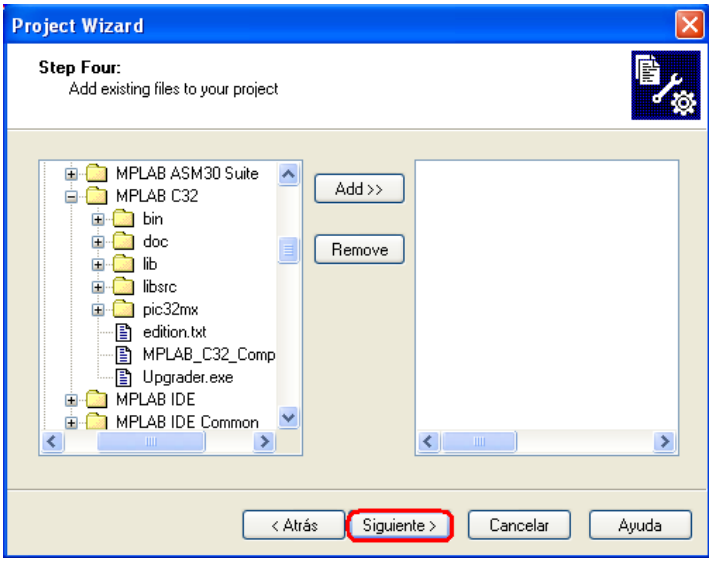



***Figura 4.7.-*** *Ventana para añadir archivos*

7. Pulsamos en “Finalizar” para que el MPLAB genere los archivos del proyecto.

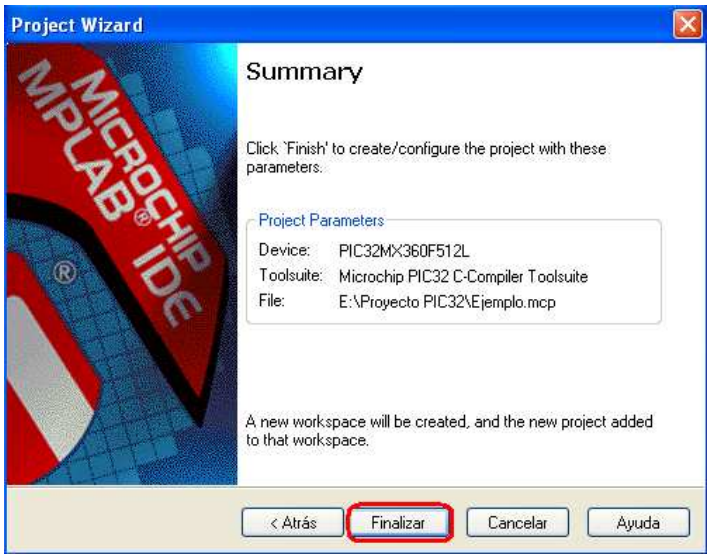



***Figura 4.8.-*** *Ventana de resumen*

## 4.3. Escribir y compilar el programa

1. El siguiente paso es escribir el código del programa, para ello desde el MPLAB se selecciona en la barra de herramientas “New File” para crear un nuevo archivo según se muestra en la figura X.

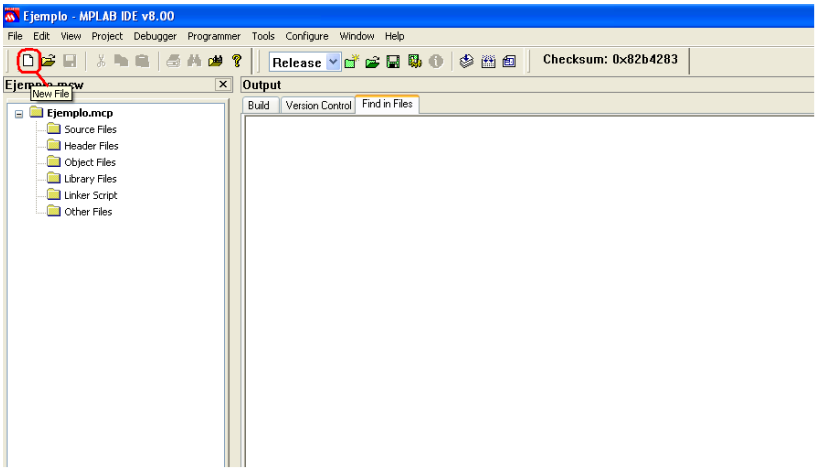



***Figura 4.9.-*** *Crear un nuevo archivo*

2. A continuación aparecerá el cuadro de edición de programas, donde se escribe el código fuente de la aplicación.

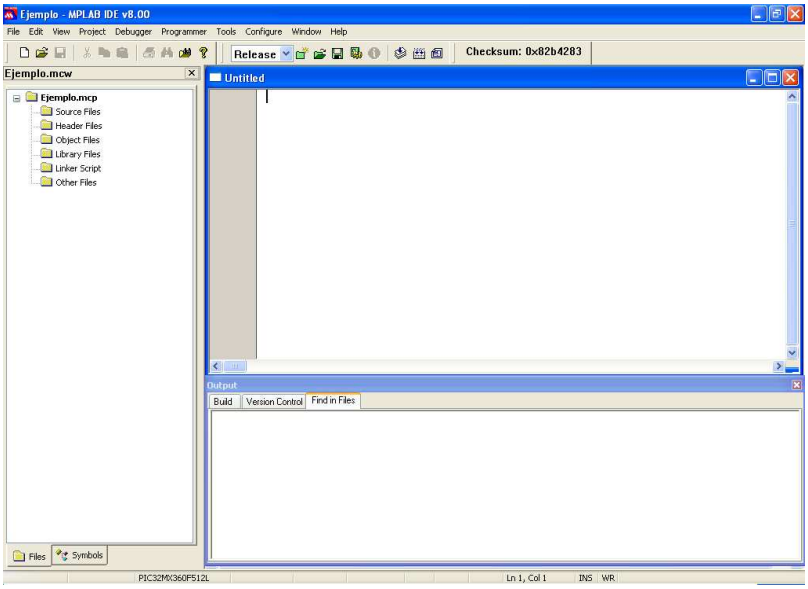



***Figura 4.10.-*** *Ventana de edición de programas*

3. Una vez escrito el programa en la ventana de edición, se debe agregar el archivo al proyecto, para ello se selecciona “File--> Save” para guardar el programa, en la ventana que aparece se escribe un nombre para el programa, deberá tener una extensión .c. No hay que olvidarse de marcar la casilla "añadir archivo al proyecto", como se muestra en la figura X.

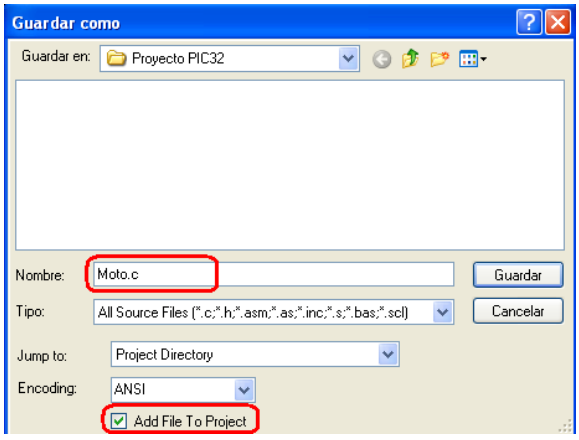



***Figura 4.11.-*** *Ventana para guardar el programa*

4. MPLAB nos permite depurar el programa a través de varias herramientas, seleccionamos “Debugger --> Select Tool --> MPLAB SIM” como se observa en la figura X.

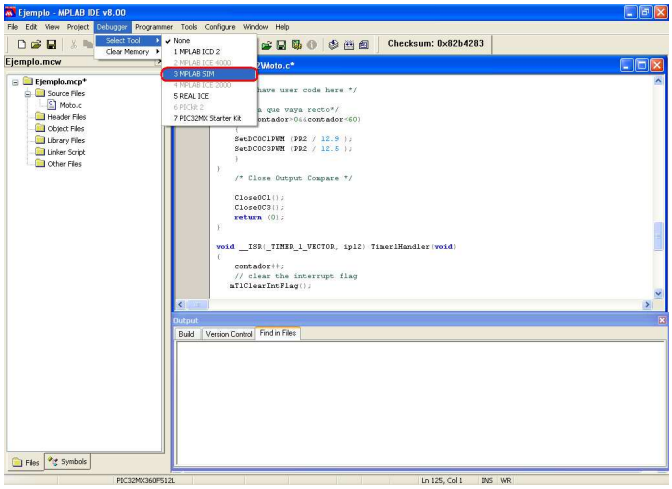



***Figura 4.12.-*** *Ventana para depurar el programa*

5. Una vez está todo preparado, llega el momento de compilar para comprobar que el programa no tiene ningún error. Se selecciona la opción "Build All" para que MPLAB nos genere todos los archivos necesarios para ejecutar y depurar nuestro programa.

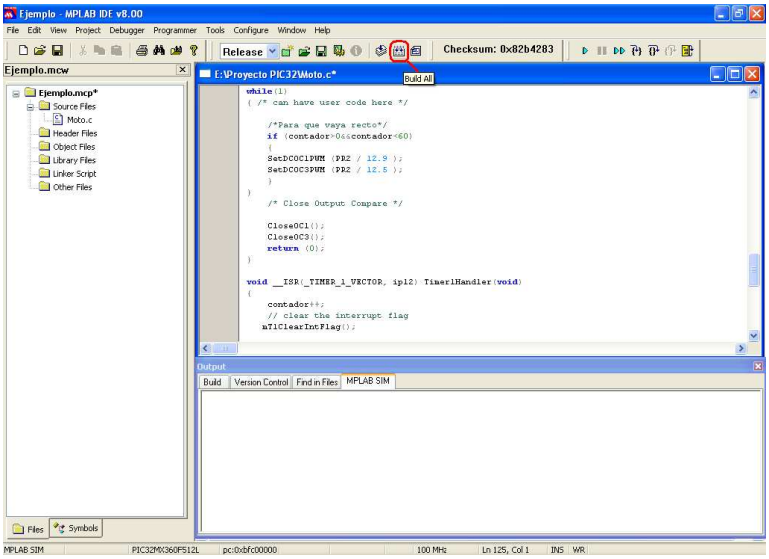



***Figura 4.13.-*** *Ventana de compilación del programa*

6. Si el programa se ha compilado sin errores, aparecerá en la ventana de salida que el proyecto se ha construido satisfactoriamente y a partir de ahora estarán habilitados los botones de simulación. MPLAB permite realizar acciones como simular el programa paso a paso, crear Break Point y ver el estado de los registros del PIC.

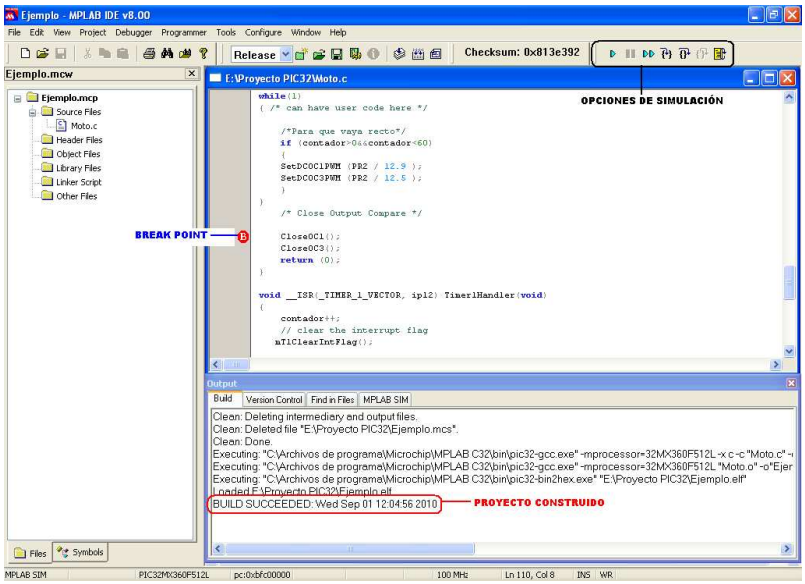



***Figura 4.14.-*** *Proyecto compilado*

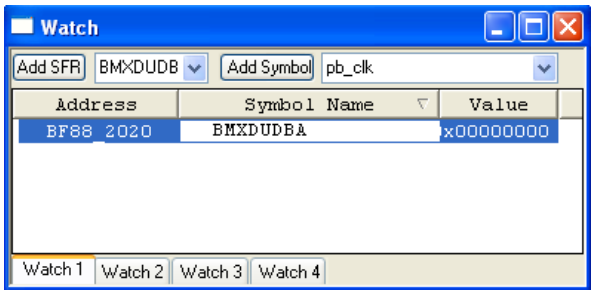



***Figura 4.15.-*** *Ventana de estado de registros del PIC*

## 4.4. Descargar y ejecutar el programa

1. El último a realizar es conectar el Starter Kit de PIC32 a través del cable USB al ordenador. Desde el MPLAB se selecciona “Debugger-> Select tool -> PIC32MX Starter Kit”

2. Si la conexión ha tenido éxito aparecerá la barra de herramientas para programar y depurar el PIC, donde pulsaremos sobre “Program All Memories” para cargar el programa en el PIC. Si todo ha ido bien aparecerá “Programming Done” en la ventana de salida como podemos observar en la figura X.

***Figura 4.16.-*** *Barra de herramientas1*

3. A partir de ahora se podrá ejecutar y simular paso a paso el programa desde el interior del PIC pulsando sobre “Run”.

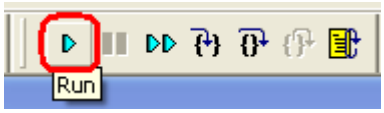



***Figura 4.17.-*** *Barra de herramientas2*

# Capítulo 5.

# Osciladores y timers

## 5.1. Osciladores

Una de las primeras cosas que hay que tener claro cuando trabajamos con Microcontroladores es la configuración de los diferentes osciladores que tiene nuestro micro, si esa configuración no es correcta el PIC no funcionará o lo hará a un ritmo diferente al deseado [11].

Los PIC32 presentan bastantes novedades en los relojes y en los osciladores disponibles para su uso, lo primero que hay que tener en cuenta es que en esta arquitectura existen dos buses diferentes para la comunicación de instrucciones y datos entre los diferentes componentes del PIC. El principal (Matrix), utilizado por el núcleo del Microcontrolador y por unos pocos periféricos privilegiados como el Acceso Directo a Memoria (DMA) y el controlador del servicio de interrupciones que demandan una frecuencia elevada en el bus de comunicaciones. El resto de componentes se comunica a través del bus de periféricos, que trabaja a una frecuencia menor que el bus principal, ambos buses se enlazan a través de un puente llamado "peripheral Bridge" similar al puente Norte y puente Sur que disponen las placas base de los ordenadores personales.

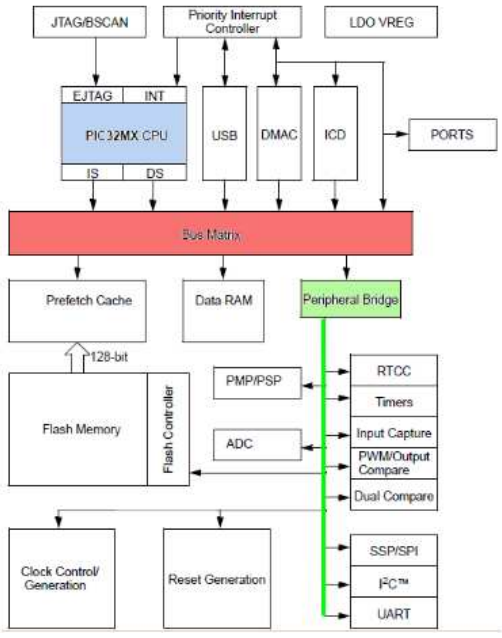



***Figura 5.1.****-Esquema de osciladores y buses.*

Cada uno de estos buses funciona con una señal de reloj diferente, los relojes disponibles en un PIC32 son:

1. El reloj del bus principal (SYSCLK)
2. EL reloj del bus de periféricos (PBCLK)
3. El reloj USB (USBCLK), utilizado expresamente por los periféricos USB.

Cada una de estas señales de reloj pueden ser producidas por diferentes fuentes de oscilación:

- Oscilador externo primario (POSC) conectado entre los pines OSCI y OSCO del PIC. El starter kit tiene conectado un cristal de 8 Mhz.
- Oscilador externo secundario (SOSC) conectado en los pines SOSCI y SOSCO. EL starter kit viene preparado para conectarle un cristal externo de 32.768 Hz que puede ser utilizado como oscilador principal ó como fuente para el Timer1 o los módulos RTCC. Es un oscilador de precisión que se puede utilizar en aplicaciones donde se necesite un cronometraje de tiempo preciso.
- Oscilador RC interno (FRC), proporciona una frecuencia de reloj de 8 MHz y no requiere ningún componente externo.
- Oscilador RC interno de baja potencia (LPRC), proporciona una frecuencia de reloj base de 32 KHz, sin necesidad de ningún componente externo también pero con una precisión baja.

Cada una de las fuentes de reloj tiene sus propias opciones de configuración, como multiplicador y divisor de entrada y salida del PLL, que nos permite obtener la frecuencia de reloj que queramos en cada uno de los buses.

El esquema de bloques de los diferentes osciladores de que disponen los PIC32 se observa en la siguiente figura. Los cálculos de las frecuencias son calculados en el apartado 5.2.3.

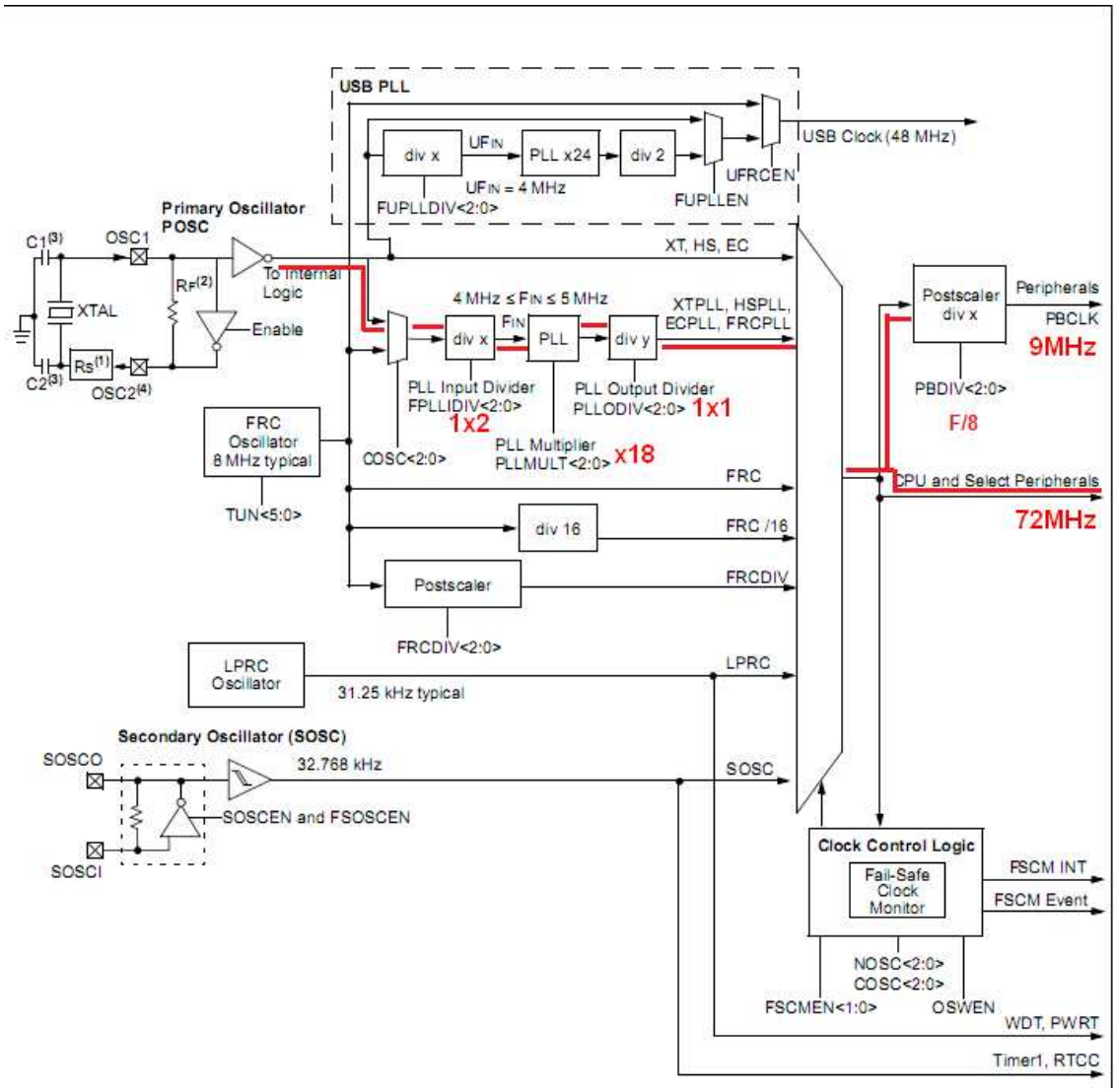



***Figura 5.2.-*** *Diagrama de bloques de los osciladores*

## 5.2. Timers

### 5.2.1. Pines disponibles

El PIC 32 dispone de 5 pines para utilizarlos como PWM de entrada (OC1, OC2, OC3, OC4 y OC5) y 2 pines de salida (EPOC y OCFB). La función de estos pines de salida es como protección contra fallos.

### 5.2.2. Códigos del timer

La PWM se puede configurar mediante algún cambio directo de las funciones especiales de registro o usando las funciones en outcompare.h (un archivo principal incluido en la biblioteca periférica (plib.h). Este último caso es más sencillo y fácil.

Hay cuatro funciones principales que se utilizan para PWM.

- void OpenOCX (config, valor1, valor2)
- void OpenTimerX (config y período)
- unsigned int SetDCOCXPWM (dutycycle)
- void CloseOCX ();

Donde X es el módulo (1-5) o el timer (2-3) que desea utilizar. Cada una de estas funciones se describen a continuación.

- OpenOCX configura el módulo de OCX y carga el R (valor2) y RS (valor1) siendo registros con valores predeterminados.
- El registro OCxR sirve solo como lectura del registro del ciclo de trabajo.
- OCxRS es un registro de memoria que está escrito por el usuario para actualizar el ciclo de trabajo de la señal PWM. Al final de un período de PWM, OCxR se carga con el contenido de OCxRS. Un ejemplo es el siguiente:

```
OpenOC1 (OC_ON | OC_TIMER2_SRC | OC_PWM_FAULT_PIN_DISABLE, 0, 0);
```

Las constantes de configuración diferente se muestran a continuación. Si no se especifica, la configuración se utiliza por defecto. Estas constantes son mutuamente excluyentes para cada categoría (es decir, sólo se puede usar una de ellas).

| Constante de configuración | Descripción |
|---|---|
| OC_ON | Activa el módulo |
| OC_OFF | Por defecto - Desactiva el módulo |
| Control de parada en el modo inactivo | |
| OC_IDLE_STOP | Parada en el modo inactivo |
| OC_IDLE_CON | Por defecto - Continuar la operación en modo inactivo |
| Selección del modo 16/32 bit | |
| OC_TIMER_MODE32 | Utiliza el modo de 32 bits |
| OC_TIMER_MODE16 | Por defecto - Utiliza el modo de 16 bits |
| Selección del timer | |
| OC_TIMER3_SRC | Selección del Timer 3 |
| OC_TIMER2_SRC | Por defecto – Selección del Timer 2 |
| Selección del modo a operar | |
| OC_PWM_FAULT_PIN_ENABLE | PWM en modo OCx activado por defecto |
| OC_PWM_FAULT_PIN_DISABLE | PWM en modo OCx desactivado por defecto |
| OC_CONTINUE_PULSE | Tren de pulsos de salida en el pin OCx |
| OC_SINGLE_PULSE | Un único pulso de salida en el pin OCx |
| OC_TOGGLE_PULSE | Cambio de estado en el pin OCx |
| OC_HIGH_LOW | |
| OC_LOW_HIGH | |
| OC_MODE_OFF | Por defecto - Output compare desactivado |

***Tabla 5.1.-*** *Constantes de configuración*

En el modo de funcionamiento, para usar una PWM una de las primeras dos constantes debe ser utilizada.

El módulo de comparador de salida utiliza el Timer 2 por defecto o el Timer 3. Estos deben crearse utilizando la función OpenTimerX, donde X es 2 ó 3. El periodo puede ir desde 0 hasta 0xFFFF, ambos inclusive. El OC puede ser configurado para usar un timer de 32 bits, siendo el timer 2 y 3 los utilizados.

Un ejemplo es el siguiente:

OpenTimer2 (T2_ON | T2_PS_1_1 | T2_SOURCE_INT, 0xFFFF)

La primera entrada establece el temporizador y la segunda variable es el valor del periodo. El periodo de PWM depende del valor del periodo (PR), el periodo de buses periféricos y el pre-escalar, como podemos ver a continuación:

Periodo PWM = [(PR + 1) TPB (TMR_Prescaler_Value)]

En la siguiente tabla se muestran las constantes de configuración para el Timer 2.

| **Constante de configuración** | **Descripción** |
|---|---|
| T2_ON | Activa el temporizador |
| T2_OFF | Por defecto - Desactiva el temporizador |
| Control de parada en el modo inactivo | |
| T2_IDLE_STOP | Parada en el modo inactivo |
| T2_IDLE_CON | Por defecto - Continuar la operación en modo inactivo |
| Control timer de entrada | |
| T2_GATE_ON | Activa la entrada del Timer 2 |
| T2_GATE_OFF | Por defecto – Desactiva la entrada del Timer 2 |
| Valores del prescaler | |
| T2_PS_1_256 | 1:256 |
| T2_PS_1_64 | 1:64 |
| T2_PS_1_32 | 1:32 |
| T2_PS_1_16 | 1:16 |
| T2_PS_1_8 | 01:08 |
| T2_PS_1_4 | 01:04 |
| T2_PS_1_2 | 01:02 |
| T2_PS_1_1 | Por defecto - 01:01 |
| Selección del modo 16/32 bit | |
| T2_32BIT_MODE_ON | Activa el modo 32-bit |

| | |
|---|---|
| T2_32BIT_MODE_OFF | Por defecto – Desactiva el modo 32-bit |
| Selección del reloj | |
| T2_SOURCE_EXT | Reloj externo |
| T2_SOURCE_INT | Reloj interno |

***Tabla 5.2.-*** *Constantes de configuración Timer 2*

El ciclo de trabajo se puede actualizar mediante el uso de SetDCOCXPWM (dutycycle) donde X es el módulo. El ciclo de trabajo se actualiza en el ciclo siguiente. El ciclo de trabajo debe ser menor o igual al valor del periodo definido en el registro Timer 2 o 3 del timer. Un ejemplo es el siguiente:

SetDCOC1PWM (PR2 / 2);

La salida del Output Compare se puede cerrar como se muestra a continuación:

CloseOC1 ();

## 5.2.3. Cálculos del timer

En este proyecto vamos a utilizar el timer1 y el timer2, que son dos de los cinco temporizadores de 16 bits de que dispone el PIC32, hay que decir que aunque los temporizadores son de 16 bits, el Timer2 y Timer3 se pueden agrupar para formar un temporizador de 32 bits, lo mismo ocurre para el Timer4 y Timer5, eso da un margen de tiempos muy amplio.

Los registros principales para controlar el timer2 son:

- **TMR2**--> donde se guarda el valor del contador de 16 bits.
- **T2CON** --> Registro de configuración del TIMER2
- **PR2** --> valor a cargar para producir el reseteo del Timer

Lo que queremos conseguir en nuestro proyecto es una señal PWM de 20ms de periodo. Podremos comprobar la precisión de la señal obtenida con el timer con la ayuda de un osciloscopio. Para obtener dicha señal utilizaremos la función de interrupción del Timer2.

Además necesitamos un contador que cambie de pulso cada segundo. Para ello utilizaremos el Timer1.

### 5.2.3.1. Frecuencia del bus principal

Podemos decir que la frecuencia del Bus principal vendrá dada por la siguiente fórmula:

$$SYSCLK = FNOSC \times FPLLIDIV \times FPLLMUL \times FPLLODIV$$

Y según la configuración de nuestro ejemplo, obtendremos:

$$SYSCLK = 8\ MHz\ x\ \frac{1}{2}\ x\ 18\ x\ \frac{1}{1} = 72\ MHz$$

#### 5.2.3.2. Frecuencia del bus de periféricos

La fórmula para calcular la frecuencia del Bus de periféricos, es bien sencilla y será proporcional a la frecuencia que tengamos en el bus principal, en nuestro ejemplo valdrá lo siguiente:

$$PBCLK = SYSCLK \times \frac{1}{FPBDIV} = 72\ MHz \times \frac{1}{8} = 9\ MHz$$

#### 5.2.3.3. Cálculo del PR2

La fórmula para calcular los valores de los Timers del PIC32 en su modo simple (16 bits) es la siguiente:

$$Time = \frac{1}{PBCLK} \times Prescaler \times PRx$$

En donde:

- **Time** --> Tiempo en segundos para el desbordamiento del timer.
- **PBCLK** --> Frecuencia del bus de periféricos.
- **Prescaler** --> Puede tomar los valores (1:1, 1:2, 1:4, 1:8, 1:16, 1:32, 1:64, 1:256)
- **PR2** --> Valor del registro PR2, es el valor a calcular en función de los otros parámetros.

Si utilizamos un preescaler de 4 el valor a cargar en el PR2 para que se produzca un desbordamiento cada 20ms será:

$$20 \times 10^{-3} = \frac{1}{9 \times 10^{6}} \times 4 \times PR2$$

$$PR2 = 45000 = 0xAFC8 \text{ en Hexadecimal}$$

# Capítulo 6.

# Código del programa

# 6.1. Código

```
/********************************************************************
 *
 *              OCMP Simple PWM Application
 *
 ********************************************************************
 * FileName:       ocmp_simple_pwm.c
 * Dependencies:
 * Processor:      PIC32
 *
 * Complier:       MPLAB C32
 *                 MPLAB IDE
 * Company:        Microchip Technology, Inc.
 *
 * Software License Agreement
********************************************************************/
#include <plib.h>
#pragma config FPLLMUL = MUL_18, FPLLIDIV = DIV_2, FPLLODIV = DIV_1, FWDTEN =
OFF
#pragma config POSCMOD = HS, FNOSC = PRIPLL, FPBDIV = DIV_2
// Let compile time pre-processor calculate the PR1 (period)
#define FOSC                    72E6
#define PB_DIV                  8
#define PRESCALE                256
#define TOGGLES_PER_SEC             1
#define T1_TICK                 (FOSC/PB_DIV/PRESCALE/TOGGLES_PER_SEC)
int contador=0;
int main(void)
{
        SYSTEMConfigPerformance(FOSC);
   OpenTimer1(T1_ON | T1_SOURCE_INT | T1_PS_1_256, T1_TICK);

    ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_2);

    mOSCSetPBDIV(OSC_PB_DIV_8);

    INTEnableSystemMultiVectoredInt();
```

```
    mPORTDSetPinsDigitalOut(BIT_0);

    OpenTimer2(T2_ON | T2_PS_1_4 | T2_SOURCE_INT, 0xAFC8);

    OpenOC1( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE |  OC_LOW_HIGH ,
0xAFC8, 0x0000 );

    OpenOC3( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE | OC_LOW_HIGH ,
0xAFC8, 0x0000 );

while(1)
{
        if (contador>0&&contador<1)
        {
        SetDCOC1PWM (PR2 / 12.15 );
        SetDCOC3PWM (PR2 / 13.5 );
        }
        if (contador>=1&&contador<2)
        {
        SetDCOC1PWM (PR2 / 12.15 );
        SetDCOC3PWM (PR2 / 13.4 );
        }
        if (contador>=2&&contador<3)
        {
        SetDCOC1PWM (PR2 / 12.15 );
        SetDCOC3PWM (PR2 / 13.3 );
        }
        if (contador>=3&&contador<4)
        {
        SetDCOC1PWM (PR2 / 12.15 );
        SetDCOC3PWM (PR2 / 13.2 );
        }
        if (contador>=4&&contador<5)
        {
        SetDCOC1PWM (PR2 / 12.15 );
        SetDCOC3PWM (PR2 / 13.1 );
        }
        if (contador>=5&&contador<6)
        {
        SetDCOC1PWM (PR2 / 12.15 );
```

```
            SetDCOC3PWM (PR2 / 12.9 );
            }
            if (contador>=6&&contador<7)
            {
            SetDCOC1PWM (PR2 /12.15);
            SetDCOC3PWM (PR2 / 12.7 );
            }

            if (contador>=7&&contador<60)
            {
            SetDCOC1PWM (PR2 /12.15);
            SetDCOC3PWM (PR2 / 12.5 );
            }
}
            CloseOC1();
            CloseOC3();
            return (0);
}

void __ISR(_TIMER_1_VECTOR, ipl2) Timer1Handler(void)
{
            contador++;
   // clear the interrupt flag
  mT1ClearIntFlag();

   // .. things to do
   // .. in this case, toggle the LED
   mPORTDToggleBits(BIT_0);
}
```

## 6.2. Significado de las funciones

### 6.2.1. Inicialización de los valores

```
#pragma config FPLLMUL = MUL_18, FPLLIDIV = DIV_2, FPLLODIV =
DIV_1, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRIPLL, FPBDIV = DIV_2
```

Lo primero que hay que configurar son los fusibles o bits de configuración, para ello tenemos dos opciones configurarlos, a través del IDE de MPLAB: menú -->Configurar--> Configuración Bits.... y Una vez establecidos los valores se guardarán en el archivo del proyecto (.MCW), volcándose a la memoria Flash del PIC en la programación del mismo o podemos configurarlos como en este ejemplo en el propio código a través de la directiva #**pragma**. Nosotros utilizaremos está última.

Los PIC32 disponen de un amplio número de fusibles con diferentes parámetros la mayoría de ellos, pero se pueden configurar los básicos e imprescindibles y dejar por defecto los otros, está configuración básica nos servirá de plantilla en otra ocasión para futuros ejemplos. Empecemos a describir cada uno de los fusibles:

- #**pragma config FNOSC = PRIPLL**//Con FNOSC seleccionamos el oscilador a utilizar, seleccionaremos el Oscilador Principal con PPL (Multiplicador), recordar que en el Starter Kit está alimentado por un cristal de 8MHz
- #**pragma config POSCMOD = HS** //Modo reloj principal, como la frecuencia del reloj es menor de 10 MHz seleccionamos HS.
- #**pragma config FPBDIV = DIV_2**//divisor Bus Periféricos, el Bus de Periféricos será un submúltiplo de la frecuencia del Bus principal, en este caso dividimos por 2 la frecuencia principal.
- #**pragma config FWDTEN = OFF**//Wachdog deshabilitado //Configuración del Multiplicador
- #**pragma config FPLLODIV = DIV_1** //Pos-Divisor PPL 1/1
- #**pragma config FPLLIDIV = DIV_2** //Pre-Divisor PPL ½
- #**pragma config FPLLMUL = MUL_18** // 18Xppl

### 6.2.2. Función de optimización y configuración de cache

```
SYSTEMConfigPerformance(FOSC);
```

Esta arquitectura de microcontroladores incorpora entre otras novedades un módulo de memoria cache, este tipo de memoria es muy rápida pero también costosa por lo que por cuestiones de precio no se puede poner la que se quiera, a muchos de vosotros por lo menos les sonará el nombre ya que las computadoras personales las incorporan desde hace tiempo. El concepto es el mismo, cuando el microprocesador solicita una instrucción o un dato de la memoria Flash, primero comprueba si está en la memoria cache, si está lo coge de ahí, si no lo va a buscar a la memoria Flash y guarda una copia en la cache para la próxima vez que el micro necesite ejecutar la misma instrucción, con esto se reducen los tiempos de espera en que el micro tiene que esperar en recuperar una instrucción o un dato de la memoria Flash.

La memoria cache esta deshabilitada por defecto en los PIC32 y lo mismo que en un ordenador personal hay que entrar en el setup de la Bios, y activarla para sacarle el máximo rendimiento al ordenador, en el PIC 32 tenemos que hacer lo mismo, salvo que aquí no tenemos ninguna BIOS. Para hacerlo fácil Microchip ha incorporado una función de biblioteca del tipo todo en uno, que tendremos que llamar al inicio de nuestro programa:

```
SYSTEMConfigPerformance (72000000L);
```

Esta función recibirá como parámetro la frecuencia de trabajo de nuestro Bus Principal y no solamente activará la memoria cache sino que configurará el resto de parámetros necesarios para que nuestro PIC trabaje al máximo rendimiento.

### 6.2.3. Iniciación Timer 1

```
OpenTimer1(T1_ON | T1_SOURCE_INT | T1_PS_1_256, T1_TICK);
```

La biblioteca plib.h nos proporciona una función para configurar fácilmente el timer1.

Esta función recibe una serie de parámetros en los que:

- **T1_ON** --> activa el TIMER1
- **T1_PS_1_256** --> establece el prescaler a 1:256
- **T1_SOURCE_INT**-->Establece como fuente el reloj interno
- **T1_TICK** --> carga el valor calculado previamente en el registro PR1

Hay que tener en cuenta que el Timer1 en su modo de trabajo simple es un contador de 16 bits por lo que el valor del contador estará comprendido entre 0 y 65535.

### 6.2.4. Configuración de Interrupción por desbordamiento TMR1, Prioridad 2

```
ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_2);
```

Esta instrucción habilita el modo de múltiples vectores de interrupción y la interrupción por desbordamiento del TMR1 con un nivel de prioridad de 2.

### 6.2.5. Variación del PBDIV

```
mOSCSetPBDIV(OSC_PB_DIV_8);
```

La llamada a esta función lo que hace es sobrescribir el divisor del bus de periféricos al valor que habíamos establecido previamente en los bits de configuración, esto es necesario ya que la función de optimización anterior modifica por su cuenta el divisor del bus de periféricos, supongo que para obtener la máxima frecuencia también en este bus, pero que si lo dejamos así, los cálculos que realicemos para el timer con los parámetros que hemos elegido nosotros no serán correctos.

Una vez configurados los fusibles y optimizado los recursos de nuestro PIC para obtener su máximo rendimiento, nos queda hacer los cálculos necesarios sobre el Timer2 para obtener una interrupción cada 20 milisegundos.

### 6.2.6. Habilitar múltiples vectores de interrupción

```
INTEnableSystemMultiVectoredInt();
```

Los PIC32 disponen de un controlador para gestionar las interrupciones muy flexible, se puede programar para que trabaje con un solo vector de interrupción como lo hacen los PIC de inferior categoría o para aumentar el rendimiento, utilizar un sistema con múltiples vectores de interrupción, la familia PIC32MX dispone, debido al núcleo (MIPS), de un máximo 64 vectores de interrupción, sin embargo se dispone de hasta 96 fuentes diferentes de interrupción, por lo que los diseñadores han establecido que determinadas fuentes de interrupción pertenecientes a un mismo periférico compartan el mismo vector de interrupción. Si queremos habilitar la opción de Múltiples vectores de interrupción y utilizamos para ello la librería plib de Microchip, simplemente tenemos que incluir la llamada a la función **INTEnableSystemMultiVectoredInt()**. En el modo Multi-vector, a cada vector de interrupción se le puede asigna una prioridad determinada que va del 1 (menor prioridad) al 7 (máxima prioridad) un valor 0 indica que el vector de interrupción está deshabilitado (valor por defecto). Las interrupciones de mayor prioridad prevalecen a las de menor prioridad, además cada vector admite hasta cuatro niveles de sub-prioridad diferentes, de esta manera se puede programar diferentes vectores

con la misma prioridad pero con una sub-prioridad diferente y prevalecerá la que tenga una sub-prioridad mayor.

### 6.2.7. Configuración puerto de salida

```
mPORTDSetPinsDigitalOut(BIT_0);
.....
.....
.....
void __ISR(_TIMER_1_VECTOR, ipl2) Timer1Handler(void)
{
        contador++;
        mT1ClearIntFlag();
        mPORTDToggleBits(BIT_0);
}
```

La siguiente instrucción configura el pin RD0 del puerto D como salida:

**mPORTDSetPinsDigitalOut(BIT_0);**

Dentro ya de la función de interrupción hacemos solo dos cosas: Borrar el flag de interrupción, bit T1IF del registro IFS0 por medio de la llamada a la función

**mT1ClearIntFlag();**

y hacer el toggle del led conectado en el pin RD0

**mPORTDToggleBits(BIT_0);**

Una vez compilado el ejemplo y ejecutado en la starter kit podemos comprobar si son

correctos los cálculos realizados, viendo en el osciloscopio el periodo de la señal obtenida en el pin RD0.

### 6.2.8. Iniciación Timer 2

```
OpenTimer2(T2_ON | T2_PS_1_4 | T2_SOURCE_INT, 0xAFC8);
```

La biblioteca plib.h nos proporciona una función para configurar fácilmente el timer2. Esta función recibe una serie de parámetros en los que:

- **T2_ON** --> activa el TIMER2
- **T2_PS_1_4** --> establece el prescaler a 1:4
- **T2_SOURCE_INT**-->Establece como fuente el reloj interno
- **0xAFC8** --> carga el valor calculado previamente en el registro PR2

Hay que tener en cuenta que el Timer2 en su modo de trabajo simple es un contador de 16 bits por lo que el valor del contador estará comprendido entre 0 y 65535, cualquier valor que intentemos cargar en el PR2 superior a este será truncado y el resultado obtenido será inesperado. Para obtener temporizaciones mayores a las permitidas por este rango, siempre podemos utilizar el Timer2 junto al Timer3 trabajando en modo de 32 bits.

### 6.2.9. Apertura de los puertos PWM

```
OpenOC1( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE |
OC_LOW_HIGH , 0xAFC8, 0x0000 );

OpenOC3( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE |
OC_LOW_HIGH , 0xAFC8, 0x0000 );
```

OpenOCX configura el módulo de OCX y carga el R (valor2) y RS (valor1) registros con valores predeterminados. El registro OCxR es una lectura única obligación de registro de esclavos ciclo. OCxRS es un registro de memoria que está escrito por el usuario para actualizar el ciclo de trabajo PWM.Al final de un periodo de PWM, OcxR se carga con el contenido de OcxRS. Como en ambos motores necesitamos una PWM con la misma frecuencia y periodo, ambas funciones serán las mismas pero en diferentes puertos; OC1 para el servomotor y OC3 para el motor DC.

Esta función recibe una serie de parámetros en los que:

- **OC_ON** --> activa el OUTPUT COMPARE.
- **OC_TIMER2_SRC** --> la fuente de reloj es la del timer 2.
- **OC_CONTINUE_PULSE** --> genera a la salida un tren de pulsos continuo en el pin OCx
- **OC_LOW_HIGH** --> tren de pulsos a nivel alto.
- **0xAFC8** --> carga el valor calculado previamente en el registro PR2.
- **0x0000** --> se auto recarga a este valor.

# Capítulo 7.

# Simulaciones

## 7.1. Simulación 1. Moto en sentido recto.

### 7.1.1. Trayectoria a realizar

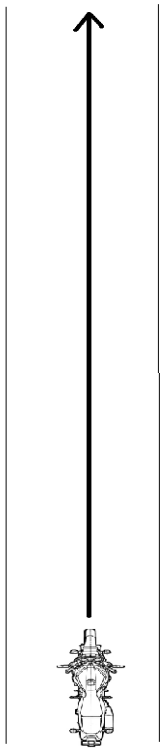


***Figura 7.1.****-Trayectoria recta*

### 7.1.2. Programación.

```
/*********************************************************************
 *
 *              OCMP Simple PWM Application
 *
 *********************************************************************
 * FileName:        ocmp_simple_pwm.c
 * Dependencies:
 * Processor:       PIC32
 *
 * Complier:        MPLAB C32
 *              MPLAB IDE
```

```
**********************************************************************/
#include <plib.h>
#pragma config FPLLMUL = MUL_18, FPLLIDIV = DIV_2, FPLLODIV = DIV_1, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRIPLL, FPBDIV = DIV_2
// Let compile time pre-processor calculate the PR1 (period)
#define FOSC                    72E6
#define PB_DIV                  8
#define PRESCALE                256
#define TOGGLES_PER_SEC             1
#define T1_TICK                 (FOSC/PB_DIV/PRESCALE/TOGGLES_PER_SEC)

int contador=0;
int main(void)
{
        SYSTEMConfigPerformance(FOSC);

   OpenTimer1(T1_ON | T1_SOURCE_INT | T1_PS_1_256, T1_TICK);

   ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_2);

    mOSCSetPBDIV(OSC_PB_DIV_8);
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   INTEnableSystemMultiVectoredInt();

   mPORTDSetPinsDigitalOut(BIT_0);

   OpenTimer2(T2_ON | T2_PS_1_4 | T2_SOURCE_INT, 0xAFC8);

   OpenOC1( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE | OC_LOW_HIGH , 0xAFC8,
0x0000 );
    OpenOC3( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE | OC_LOW_HIGH , 0xAFC8,
0x0000 );

while(1)

        if (contador>0&&contador<1)
        {
        SetDCOC1PWM (PR2 / 12.1);
```

```
        SetDCOC3PWM (PR2 / 13.5);
        }
        if (contador>=1&&contador<2)
        {
        SetDCOC1PWM (PR2 / 12.1);
        SetDCOC3PWM (PR2 / 13.4);
        }
        if (contador>=2&&contador<3)
        {
        SetDCOC1PWM (PR2 / 12.1);
        SetDCOC3PWM (PR2 / 13.3);
        }
        if (contador>=3&&contador<4)
        {
        SetDCOC1PWM (PR2 / 12.1);
        SetDCOC3PWM (PR2 / 13.2);
        }
        if (contador>=4&&contador<5)
        {
        SetDCOC1PWM (PR2 / 12.1);
        SetDCOC3PWM (PR2 / 13.1);
        }
        if (contador>=5&&contador<6)
        {
        SetDCOC1PWM (PR2 / 12.1);
        SetDCOC3PWM (PR2 / 12.9);
        }
        if (contador>=6&&contador<7)
        {
        SetDCOC1PWM (PR2 /12.1);
        SetDCOC3PWM (PR2 / 12.7);
        }

        if (contador>=7&&contador<60)
        {
        SetDCOC1PWM (PR2 /12.15);
        SetDCOC3PWM (PR2 / 12.5);
        }
}
```

```
        CloseOC1();
        CloseOC3();
        return (0);
}

void __ISR(_TIMER_1_VECTOR, ipl2) Timer1Handler(void)
{
        contador++;
  mT1ClearIntFlag();
  mPORTDToggleBits(BIT_0);
}
```

## 7.2. Simulación 2. Moto en sentido de giro a la derecha.

### 7.2.1. Trayectoria a realizar

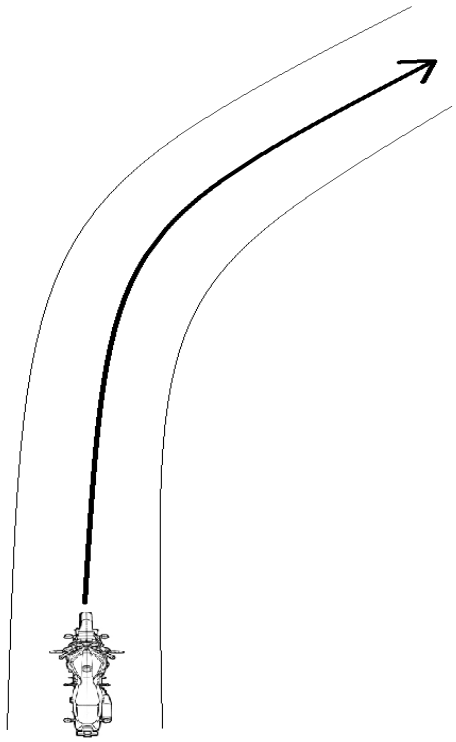


***Figura 7.2.****-Trayectoria curva derecha*

### 7.2.2. Programación

```
/*********************************************************************
 *
 *              OCMP Simple PWM Application
 *
 *********************************************************************
 * FileName:        ocmp_simple_pwm.c
 * Dependencies:
 * Processor:       PIC32
 *
 * Complier:        MPLAB C32
 *                  MPLAB IDE
```

```
**********************************************************************/
#include <plib.h>
#pragma config FPLLMUL = MUL_18, FPLLIDIV = DIV_2, FPLLODIV = DIV_1, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRIPLL, FPBDIV = DIV_2
// Let compile time pre-processor calculate the PR1 (period)
#define FOSC                    72E6
#define PB_DIV                  8
#define PRESCALE                256
#define TOGGLES_PER_SEC             1
#define T1_TICK                 (FOSC/PB_DIV/PRESCALE/TOGGLES_PER_SEC)

int contador=0;
int main(void)
{
        SYSTEMConfigPerformance(FOSC);

   OpenTimer1(T1_ON | T1_SOURCE_INT | T1_PS_1_256, T1_TICK);

   ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_2);

    mOSCSetPBDIV(OSC_PB_DIV_8);
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   INTEnableSystemMultiVectoredInt();

   mPORTDSetPinsDigitalOut(BIT_0);

   OpenTimer2(T2_ON | T2_PS_1_4 | T2_SOURCE_INT, 0xAFC8);

   OpenOC1( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE | OC_LOW_HIGH , 0xAFC8,
0x0000 );
    OpenOC3( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE | OC_LOW_HIGH , 0xAFC8,
0x0000 );

while(1)

        if (contador>0&&contador<1)
        {
        SetDCOC1PWM (PR2 / 12.6);
        SetDCOC3PWM (PR2 / 13.5);
```

```
        }
        if (contador>=1&&contador<2)
        {
        SetDCOC1PWM (PR2 / 12.6);
        SetDCOC3PWM (PR2 / 13.4);
        }
        if (contador>=2&&contador<3)
        {
        SetDCOC1PWM (PR2 / 12.6);
        SetDCOC3PWM (PR2 / 13.3);
        }
        if (contador>=3&&contador<4)
        {
        SetDCOC1PWM (PR2 / 12.6);
        SetDCOC3PWM (PR2 / 13.2);
        }
        if (contador>=4&&contador<5)
        {
        SetDCOC1PWM (PR2 / 12.6);
        SetDCOC3PWM (PR2 / 13.1);
        }
        if (contador>=5&&contador<6)
        {
        SetDCOC1PWM (PR2 / 12.6);
        SetDCOC3PWM (PR2 / 12.9);
        }
        if (contador>=6&&contador<7)
        {
        SetDCOC1PWM (PR2 /12.6);
        SetDCOC3PWM (PR2 / 12.7);
        }

        if (contador>=7&&contador<60)
        {
        SetDCOC1PWM (PR2 /12.6);
        SetDCOC3PWM (PR2 / 12.5);
        }
}

        CloseOC1();
```

```
        CloseOC3();
        return (0);
}

void __ISR(_TIMER_1_VECTOR, ipl2) Timer1Handler(void)
{
        contador++;
  mT1ClearIntFlag();
  mPORTDToggleBits(BIT_0);
}
```

## 7.3. Simulación 3. Moto en sentido de giro a la izquierda

### 7.3.1. Trayectoria a realizar

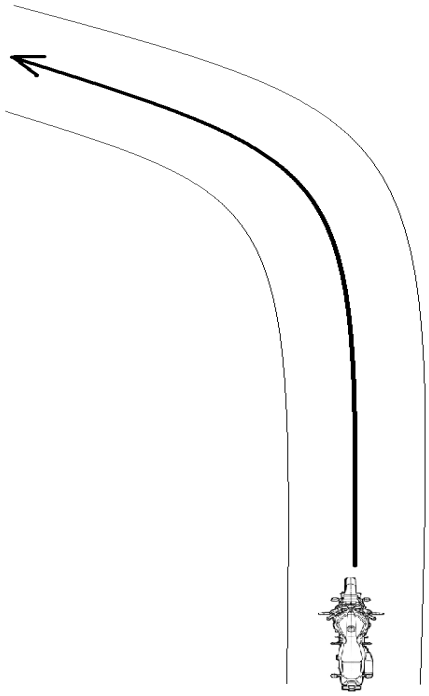


***Figura 7.3.****-Trayectoria recta*

### 7.3.2. Programación

```
/********************************************************************
 *
 *              OCMP Simple PWM Application
 *
 ********************************************************************
 * FileName:       ocmp_simple_pwm.c
 * Dependencies:
 * Processor:      PIC32
 *
 * Complier:       MPLAB C32
 *                 MPLAB IDE
```

```
**********************************************************************/
#include <plib.h>
#pragma config FPLLMUL = MUL_18, FPLLIDIV = DIV_2, FPLLODIV = DIV_1, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRIPLL, FPBDIV = DIV_2
// Let compile time pre-processor calculate the PR1 (period)
#define FOSC                    72E6
#define PB_DIV                  8
#define PRESCALE                256
#define TOGGLES_PER_SEC             1
#define T1_TICK                 (FOSC/PB_DIV/PRESCALE/TOGGLES_PER_SEC)

int contador=0;
int main(void)
{
        SYSTEMConfigPerformance(FOSC);

   OpenTimer1(T1_ON | T1_SOURCE_INT | T1_PS_1_256, T1_TICK);

   ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_2);

    mOSCSetPBDIV(OSC_PB_DIV_8);
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   INTEnableSystemMultiVectoredInt();

   mPORTDSetPinsDigitalOut(BIT_0);

   OpenTimer2(T2_ON | T2_PS_1_4 | T2_SOURCE_INT, 0xAFC8);

   OpenOC1( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE | OC_LOW_HIGH , 0xAFC8,
0x0000 );
    OpenOC3( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE | OC_LOW_HIGH , 0xAFC8,
0x0000 );

while(1)

        if (contador>0&&contador<1)
        {
        SetDCOC1PWM (PR2 / 11.6);
        SetDCOC3PWM (PR2 / 13.5);
```

```
        }
        if (contador>=1&&contador<2)
        {
        SetDCOC1PWM (PR2 / 11.6);
        SetDCOC3PWM (PR2 / 13.4);
        }
        if (contador>=2&&contador<3)
        {
        SetDCOC1PWM (PR2 / 11.6);
        SetDCOC3PWM (PR2 / 13.3);
        }
        if (contador>=3&&contador<4)
        {
        SetDCOC1PWM (PR2 / 11.6);
        SetDCOC3PWM (PR2 / 13.2);
        }
        if (contador>=4&&contador<5)
        {
        SetDCOC1PWM (PR2 / 11.6);
        SetDCOC3PWM (PR2 / 13.1);
        }
        if (contador>=5&&contador<6)
        {
        SetDCOC1PWM (PR2 / 11.6);
        SetDCOC3PWM (PR2 / 12.9);
        }
        if (contador>=6&&contador<7)
        {
        SetDCOC1PWM (PR2 / 11.6);
        SetDCOC3PWM (PR2 / 12.7);
        }

        if (contador>=7&&contador<60)
        {
        SetDCOC1PWM (PR2 / 11.6);
        SetDCOC3PWM (PR2 / 12.5);
        }
}

        CloseOC1();
```

```
        CloseOC3();
        return (0);
}

void __ISR(_TIMER_1_VECTOR, ipl2) Timer1Handler(void)
{
        contador++;
  mT1ClearIntFlag();
  mPORTDToggleBits(BIT_0);
}
```

## 7.4. Simulación 4. Moto en sentido recto y con cambio de velocidad.

### 7.4.1. Trayectoria a realizar (ver 7.1.1)

### 7.4.2. Programación

```
/********************************************************************
 *
 *              OCMP Simple PWM Application
 *
 ********************************************************************
 * FileName:       ocmp_simple_pwm.c
 * Dependencies:
 * Processor:      PIC32
 *
 * Complier:       MPLAB C32
 *              MPLAB IDE

********************************************************************/
#include <plib.h>
#pragma config FPLLMUL = MUL_18, FPLLIDIV = DIV_2, FPLLODIV = DIV_1, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRIPLL, FPBDIV = DIV_2
// Let compile time pre-processor calculate the PR1 (period)
#define FOSC                    72E6
#define PB_DIV                  8
#define PRESCALE                256
#define TOGGLES_PER_SEC             1
#define T1_TICK                 (FOSC/PB_DIV/PRESCALE/TOGGLES_PER_SEC)

int contador=0;
int main(void)
{
        SYSTEMConfigPerformance(FOSC);

   OpenTimer1(T1_ON | T1_SOURCE_INT | T1_PS_1_256, T1_TICK);
```

```
ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_2);

 mOSCSetPBDIV(OSC_PB_DIV_8);
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

INTEnableSystemMultiVectoredInt();

mPORTDSetPinsDigitalOut(BIT_0);

OpenTimer2(T2_ON | T2_PS_1_4 | T2_SOURCE_INT, 0xAFC8);

OpenOC1( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE | OC_LOW_HIGH , 0xAFC8, 0x0000 );
 OpenOC3( OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE | OC_LOW_HIGH , 0xAFC8, 0x0000 );

while(1)

        if (contador>0&&contador<1)
        {
        SetDCOC1PWM (PR2 / 12.1);
        SetDCOC3PWM (PR2 / 13.5);
        }
        if (contador>=1&&contador<2)
        {
        SetDCOC1PWM (PR2 / 12.1);
        SetDCOC3PWM (PR2 / 13.3);
        }
        if (contador>=2&&contador<3)
        {
        SetDCOC1PWM (PR2 / 12.1);
        SetDCOC3PWM (PR2 / 13);
        }
        if (contador>=3&&contador<4)
        {
        SetDCOC1PWM (PR2 / 12.1);
        SetDCOC3PWM (PR2 / 12.7);
        }
        if (contador>=4&&contador<10)
        {
```

```
            SetDCOC1PWM (PR2 / 12.1);
            SetDCOC3PWM (PR2 / 12.5);
            }
            if (contador>=10&&contador<20)
            {
            SetDCOC1PWM (PR2 / 12.1);
            SetDCOC3PWM (PR2 / 12.9);
            }
            if (contador>=20&&contador<30)
            {
            SetDCOC1PWM (PR2 / 12.1);
            SetDCOC3PWM (PR2 / 12.5);
            }

            if (contador>=30&&contador<60)
            {
            SetDCOC1PWM (PR2 / 12.1);
            SetDCOC3PWM (PR2 / 13.5);
            }
}

            CloseOC1();
            CloseOC3();
            return (0);
}

void __ISR(_TIMER_1_VECTOR, ipl2) Timer1Handler(void)
{
            contador++;
  mT1ClearIntFlag();
  mPORTDToggleBits(BIT_0);
}
```

## 7.5. Simulaciones en osciloscopio.

### 7.5.1. Servomotor.

En esta imagen se observa la PWM que realiza el servomotor, la cual debe tener 20ms de periodo para su correcto funcionamiento ya que el servomotor funciona adecuadamente a 50 Hz.

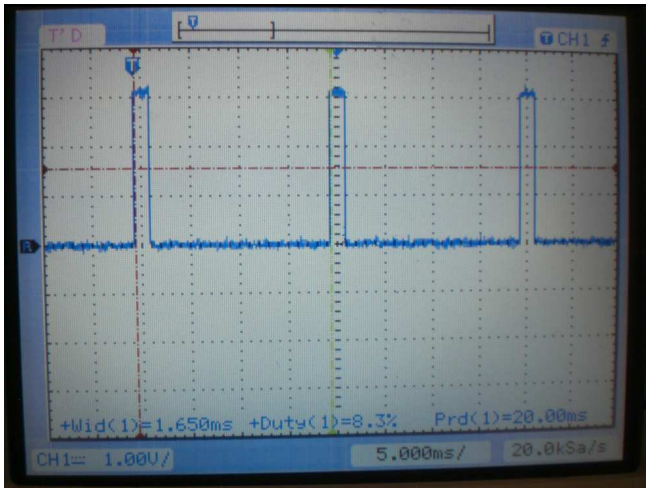



***Figura 7.4 .-*** *Servomotor*

#### 7.5.1.1. Servomotor centrado.

Esta es la posición en la que la moto consigue realizar una línea recta con la mínima velocidad necesaria que se índica en la figura.
Ton = 1.652ms T = 20ms f = 50Hz

$$d = \frac{\text{Ton x } 100}{T} = 8{,}26\,\%$$

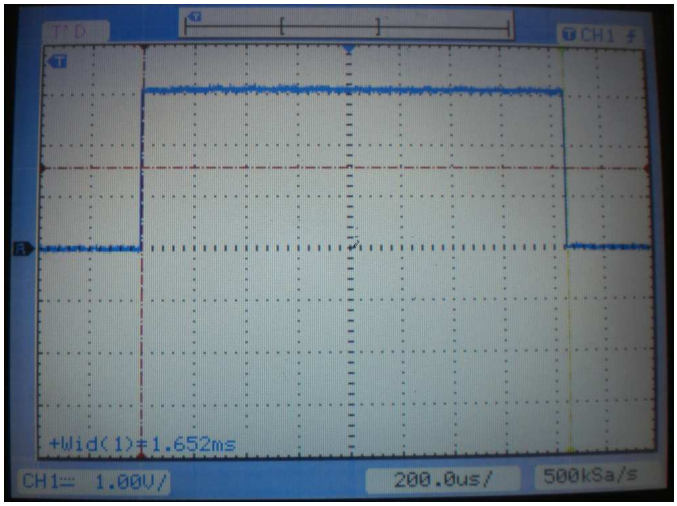



***Figura 7.5.-*** *Servomotor centrado*

#### 7.5.1.2. Servomotor para giro a la derecha.

Esta es la posición en la que la moto consigue realizar giros hacia la derecha. Con este valor la moto no caerá y permanecerá inclinada.
Ton = 1.710ms T = 20ms f = 50Hz

$$d = \frac{Ton \times 100}{T} = 8{,}55\%$$

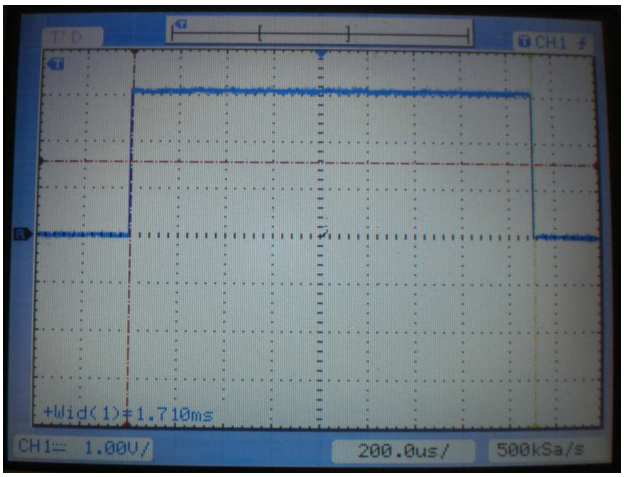



***Figura 7.6.-*** *Servomotor giro a la derecha*

#### 7.5.1.3. Servomotor para giro a la izquierda.

Esta es la posición en la que la moto consigue realizar giros hacia la derecha. Con este valor la moto no caerá y permanecerá inclinada.
Ton = 1.710ms T = 20ms f = 50Hz

$$d = \frac{Ton \times 100}{T} = 7{,}81\%$$

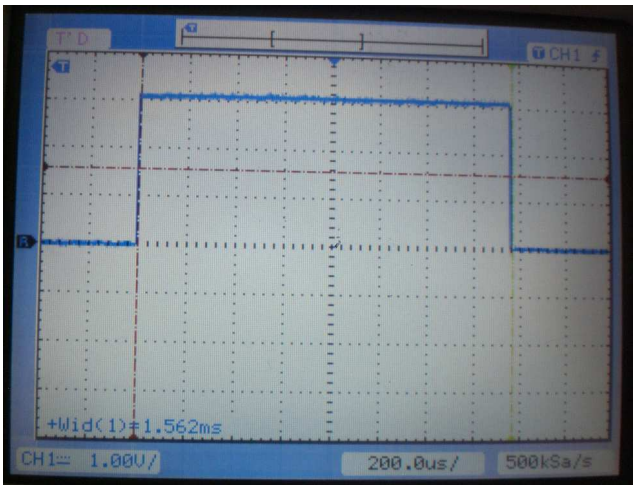



***Figura 7.7.-*** *Servomotor giro a la izquierda*

## 7.5.2. Motor DC.

En esta imagen se observa la PWM que realiza el motor brushless, la cual debe tener 20ms de periodo para su correcto funcionamiento ya que el motor DC en este caso funciona como un servomotor a 50 Hz.

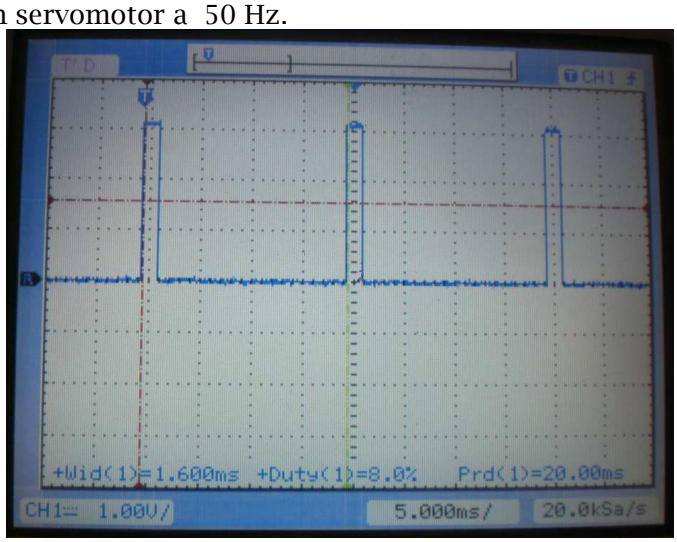



***Figura 7.8.-*** *Motor DC*

### 7.5.2.1. Motor DC en parado.

Esta es la posición del motor DC en la que la moto se mantiene parada. Con este valor la moto se mantendrá sin ninguna velocidad y se utiliza para realizar el frenado total de la moto.

Ton = 1.480ms T = 20ms f = 50Hz

$$d = \frac{\text{Ton x 100}}{\text{T}} = 7{,}40\,\%$$

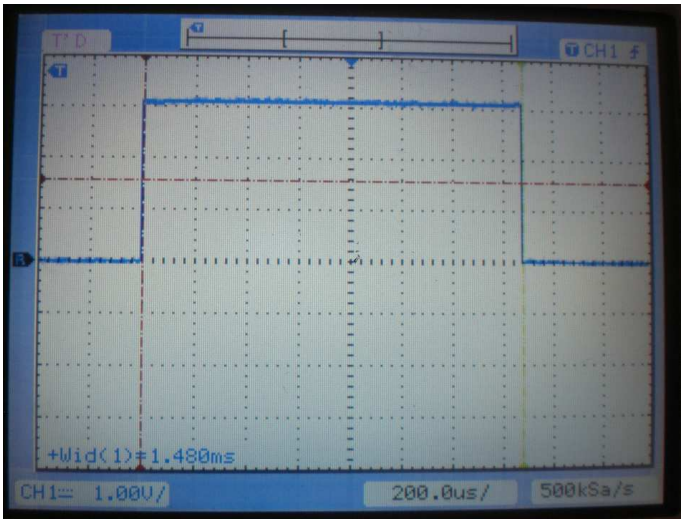



***Figura 7.9.-*** *Motor DC parado*

### 7.5.2.2. Motor DC mínima velocidad para tener estabilidad.

Esta es la posición del motor DC en la que la moto tendrá velocidad más baja para mantenerse estable. A menos velocidad la moto tendera a caer hacia el suelo y a mayor velocidad la estabilidad aun será mayor.
Ton = 1.600ms T = 20ms f = 50Hz

$$d = \frac{Ton \times 100}{T} = 8{,}00\,\%$$

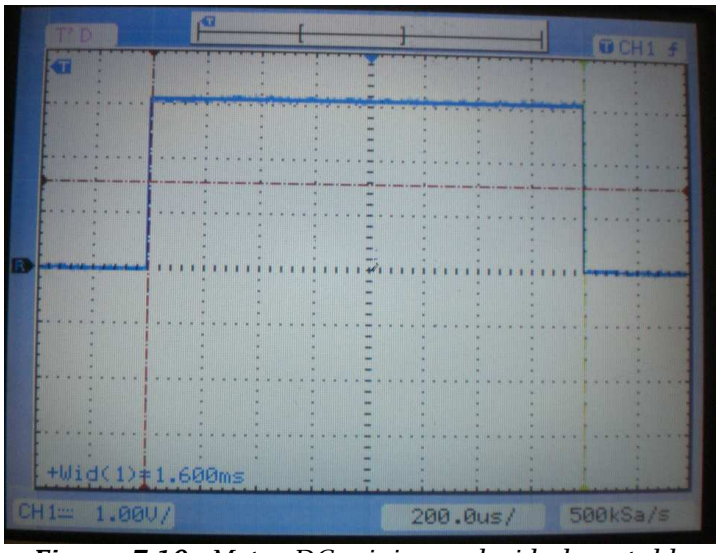



***Figura 7.10.***- *Motor DC minima velocidad y estable*

# Capítulo 8.- Conclusiones y trabajos futuros

## 8.1. Conclusiones

El proyecto realizado tenía como principal objetivo el desarrollo de una moto de automodelismo capaz de ser controlada por computador, manteniendose estable a velocidades medias-bajas. Para ello se instaló un microcontrolador PIC 32 el cual será el encargado de controlar tanto la velocidad de giro del motor brushless como la posición del servomotor. Este control se realizará de manera autónoma mediante la descarga de datos de un computador al microcontrolador mediante una conexion USB.

Una vez finalizado el proyecto se puede concluir que se han alcanzado los siguientes objetivos:

- Construcción de una moto de automodelismo por piezas realizando la inserción y conexionado de los motores con su respectiva puesta a punto.

- Dotación de la electrónica y mecánica necesarias para proporcionarle estabilidad a velociadades de trabajo medias y bajas.

- Sustitución de la parte de radiocontrol por un microcontrolador encargado de controlar la moto de manera autónoma.

- Programación del microcontrolador siendo capaz de gobernar los motores mediante la técnica PWM.

Los problemas y dificultades surgidas durante la realización del proyecto fueron las siguientes:

- Se dieron unas pautas para la realización del proyecto y se tuvo que realizar una investigación previa de como solventar los problemas planteados, ya que no se tenía ningún tipo de conocimiento acerca de este tema.

- Al no tener un control en tiempo real sobre la moto hará que sea imposible evitar cualquier imprevisto que pueda surgir durante la ruta programada.

- Es necesario lanzar de manera manual con un pequeño impulso la moto ya que al ser un robot móvil de dos ruedas carece de estabilidad inicial.

- Se necesita de un espacio abierto lo suficientemente extenso y sin ningún obstáculo intermedio para poder realizar las pruebas de campo pertinentes.

## 8.2. Trabajos futuros

Como desarrollos futuros pueden incluirse las siguientes líneas de estudio:

- Desarrollar un sistema que permita el control del microcontrolador sobre el funcionamiento y la velocidad de giro del giróscopo situado en la rueda trasera para variar la estabilidad de la moto según su necesidad.

- Instalar un encoder en la rueda delantera para registrar la velocidad de la moto en el microcontrolador en función del número de vueltas que dé la rueda por segundo.

- Colocar sensores infrarrojos en los laterales de la moto para medir y registrar en el microcontrolador la distancia al suelo e inclinación de la misma, para mediante una programación adecuada, conseguir que el servo cambie su posición durante el trayecto para lograr recuperar la estabilidad de la moto.

- Dotar al sistema con sensores de presencia en el chasis para evitar el choque contra obstáculos.

# Capítulo 9.
# Costes del proyecto

En este capítulo se presentan de forma aproximada los costes del proyecto que han sido necesarios. Para ello se desglosan las tareas que se realizaron ordenándolas temporalmente.

En la Tabla 7.1 se observan los costes de tiempo para cada fase del proyecto, mientras que en la Tabla 7.2 se detallan los costes de material como de tiempo en términos monetarios.

| FASES | TIEMPO (h) |
|---|---|
| Estudio y comprensión del problema a resolver | 60 |
| Montaje del sistema mecánico | 25 |
| Montaje del sistema electrónico | 20 |
| Iniciación a MPLAB | 15 |
| Programación del código a implementar | 25 |
| Pruebas a pie de campo | 50 |
| Mejoras para optimizar el código (ajuste de parámetros, ajustes iniciales, etc.) | 10 |
| | **SUBTOTAL 1 = 205 h** |
| Memoria del proyecto (realización paralela al mismo) | 150 |
| | **SUBTOTAL 2 = 150 h** |
| | ***TOTAL = 355 h*** |

***Tabla 9.1.-*** *Tiempo invertido en cada fase del proyecto*

Suponiendo un precio horario aproximado de 25 €/hora, el coste total sería:

| CONCEPTO | COSTE |
|---|---|
| Coste asociado a las horas ingeniero | 25 €/h x 355 h = 8875 € |
| Moto automodelismo ARX-540 | 430 € |
| Giróscopo eléctrico | 100 € |
| Microcontrolador PIC 32 | 55 € |
| Expansion Board PIC 32 | 68 € |
| Software MPLAB | 0 € |
| Circuito puente H | 10 € |
| Carcasa | 15 € |
| | ***TOTAL = 9553 €*** |

***Tabla 9.2.-*** *Costes económicos del proyecto*

# Bibliografía

# MANUALES

[1]PIC32MX Starter Kit User's Guide

[2]MPLAB IDE User's Guide

# PROYECTOS FIN DE CARRERA

[3] Mini-moto controlada por computador: arquitectura Hardware. *Espinosa Polo, Rubén*

[4] Sistema de control de un vehículo de inspección remota mediante marcación por tonos. *Sánchez Donaire, Sergio.*

[5] Diseño de un robot móvil para la realización de tareas de transporte de cargas en un hospital. *Fidalgo Rodríguez, Damián.*

[6] Micro-robot móvil autónomo rastreador con control remoto basado en un enlace radio. *Rísquez López, Óliver.*

# PÁGINAS WEB

[7] Introducción a la robótica:

< http://fundamentosinformaticosjl.wordpress.com/2008/01/27/las-computadoras/ >

[8] Clasificación de los robots:

< http://www.monografias.com/trabajos6/larobo/larobo.shtml >

[9] Robots móviles: < http://redalyc.uaemex.mx/redalyc/pdf/784/78460301.pdf >

[10] Historia de las motos RC: < http://automodelismo.com/ >

[11] Breve Tutorial - PIC32MX
<http://www.migsantiago.com/index.php?option=com_content&view=article&id=17&Itemid=20 >

[12]Iniciación a la programación

< www.aquihayapuntes.com/curso-pic32.html >

# Anexos

# ANEXO A.

# PIC32MX Family Data Sheet

## 64/100-Pin General Purpose, 32-Bit Flash Microcontrollers

© 2007 Microchip Technology Inc. Advance Information DS61143A

**Note the following details of the code protection feature on Microchip devices:**

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

**Trademarks**

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, Linear Active Thermistor, Migratable Memory, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2007, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

*Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*

MICROCHIP

# PIC32MX FAMILY

## 64/100-Pin General Purpose, 32-Bit Flash Microcontrollers

### High-Performance RISC CPU:

- MIPS32® M4K™ 32-Bit Core with 5-Stage Pipeline
- Single-Cycle Multiply and High-Performance Divide Unit
- MIPS16e™ Mode for Up to 40% Smaller Code Size
- User and Kernel Modes to Enable Robust Embedded System
- Two 32-Bit Core Register Files to Reduce Interrupt Latency
- Prefetch Cache Module to Speed Execution from Flash

### Special Microcontroller Features:

- Operating Voltage Range of 2.5V to 3.6V
- 32-512K Flash and 8-32K Data Memory
- Additional 12 KB of Boot Flash Memory
- Pin-Compatible with most PIC24/dsPIC® Devices
- Multiple Power Management Modes
- Multiple Interrupt Vectors with Individually Programmable Priority
- Fail-Safe Clock Monitor Mode
- Configurable Watchdog Timer with On-Chip, Low-Power RC Oscillator for Reliable Operation
- Two Programming and Debugging Interfaces:
  - 2-wire interface with unintrusive access and real-time data exchange with application
  - 4-wire MIPS standard enhanced JTAG interface
- Unintrusive Hardware-Based Instruction Trace
- IEEE Std 1149.2 Compatible (JTAG) Boundary Scan

### Analog Features:

- Up to 16-Channel 10-Bit Analog-to-Digital Converter:
  - 400 ksps conversion rate
  - Conversion available during Sleep, Idle
- Two Analog Comparators

### Peripheral Features:

- Atomic SET, CLEAR and INVERT Operation on Select Peripheral Registers
- Up to 4-Channel Hardware DMA Controller with Automatic Data Size Detection
- Two I²C™ Modules
- Two UART Modules with:
  - RS-232, RS-485 and LIN 1.2 support
  - IrDA® with on-chip hardware encoder and decoder
- Parallel Master and Slave Port (PMP/PSP) with 8-Bit and 16-Bit Data and Up to 16 Address Lines
- Hardware Real-Time Clock/Calendar (RTCC)
- Five 16-Bit Timers/Counters (two 16-bit pairs combine to create two 32-bit timers)
- Five Capture Inputs
- Five Compare/PWM Outputs
- Five External Interrupt pins
- 5V Tolerant Input Pins
- 8 mA Sink/Source on Select I/O Pins
- Configurable Open-Drain Output on Digital I/O Pins

| General Purpose | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Device | Pins | Program/ Data Memory (KB) | Timers/ Capture/ Compare | DMA Channels | VREG | Prefetch Cache | Trace | EUART/ SPI/ I²C™ | 10-Bit A/D (ch) | Comparators | PMP/PSP | JTAG |
| PIC32MX300F032H | 64 | 32/8 | 5/5/5 | 0 | Yes | No | No | 2/2/2 | 16 | 2 | Yes | Yes |
| PIC32MX320F064H | 64 | 64/16 | 5/5/5 | 0 | Yes | Yes | No | 2/2/2 | 16 | 2 | Yes | Yes |
| PIC32MX320F128H | 64 | 128/16 | 5/5/5 | 0 | Yes | Yes | No | 2/2/2 | 16 | 2 | Yes | Yes |
| PIC32MX340F256H | 64 | 256/32 | 5/5/5 | 4 | Yes | Yes | No | 2/2/2 | 16 | 2 | Yes | Yes |
| PIC32MX320F128L | 100 | 128/16 | 5/5/5 | 0 | Yes | Yes | No | 2/2/2 | 16 | 2 | Yes | Yes |
| PIC32MX360F256L | 100 | 256/32 | 5/5/5 | 4 | Yes | Yes | Yes | 2/2/2 | 16 | 2 | Yes | Yes |
| PIC32MX360F512L | 100 | 512/32 | 5/5/5 | 4 | Yes | Yes | Yes | 2/2/2 | 16 | 2 | Yes | Yes |

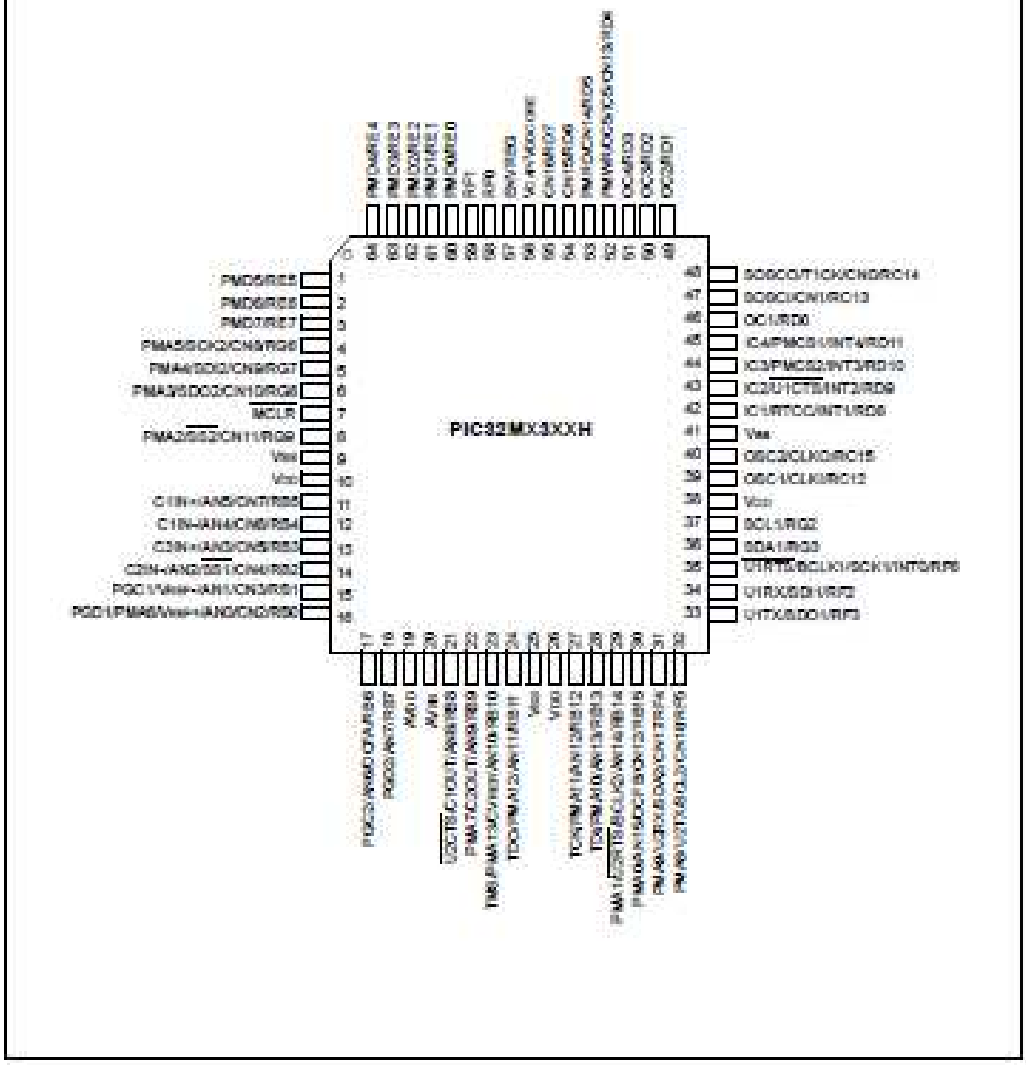
© 2007 Microchip Technology Inc.

**Advance Information**

# PIC32MX FAMILY

**Pin Diagram (64-Pin General Purpose)**

64-Pin TQFP (General Purpose)

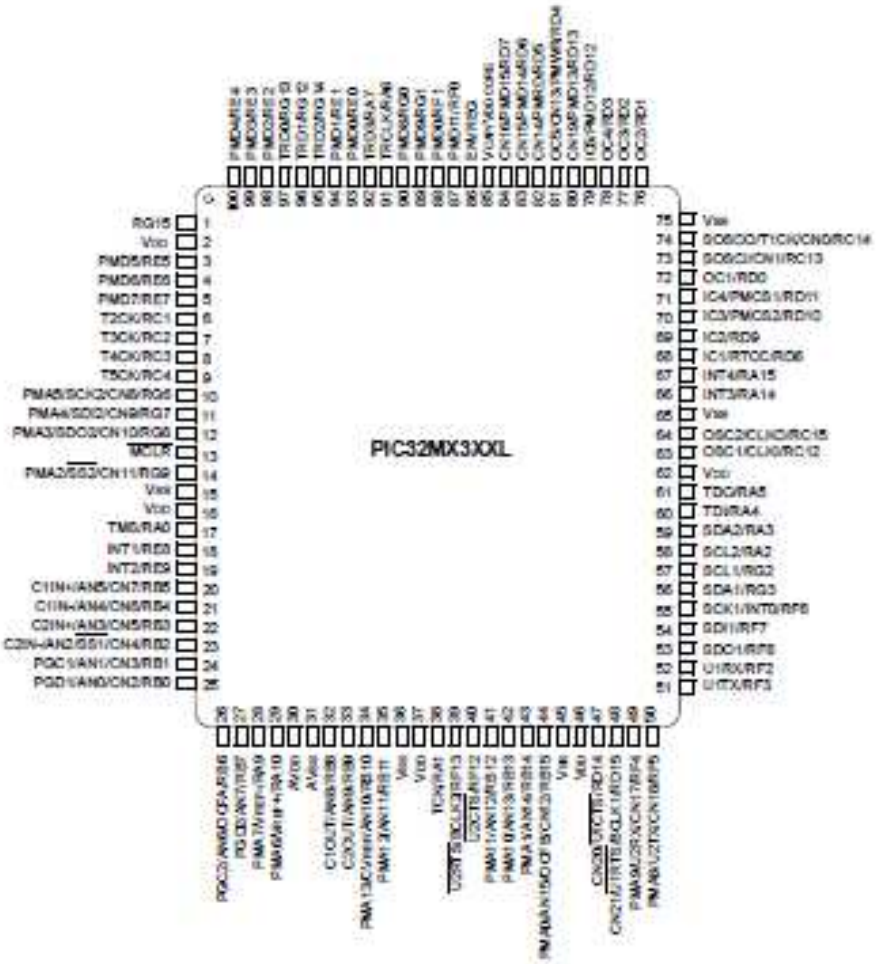




© 2007 Microchip Technology Inc.

Pin Diagram (100-Pin General Purpose)

100-Pin TQFP (General Purpose)



## 16.0 OUTPUT COMPARE

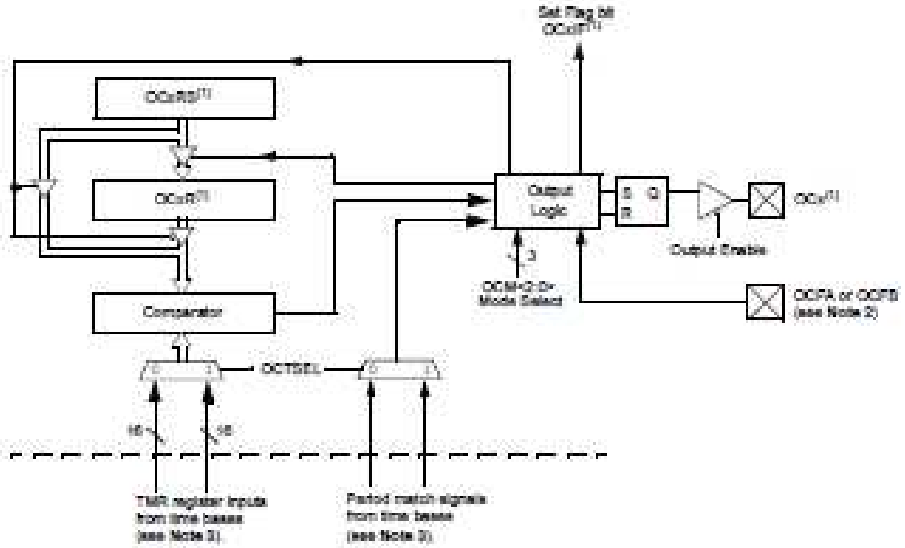
| Note: | This data sheet summarizes the features of the PIC32MX family of devices. It is not intended to be a comprehensive reference source. Refer to the *PIC32MX Family Reference Manual* (DS61132) for a detailed description of this peripheral. |
|---|---|

The Output Compare module (OCMP) is used to generate a single pulse or a train of pulses in response to selected time base events. For all modes of operation, the OCMP module compares the values stored in the OCxR and/or the OCxRS registers to the value in the selected timer. When a match occurs, the OCMP module generates an event based on the selected mode of operation.

The following are some of the key features:

- Multiple output compare modules in a device
- Programmable interrupt generation on compare event
- Single and Dual Compare modes
- Single and continuous output pulse generation
- Pulse-Width Modulation (PWM) mode
- Hardware-based PWM Fault detection and automatic output disable
- Programmable selection of 16-bit or 32-bit time bases.
- Can operate from either of two available 16-bit time bases or a single 32-bit time base.

FIGURE 16-1: OUTPUT COMPARE MODULE BLOCK DIAGRAM




Note 1: Where 'x' is shown, reference is made to the registers associated with the respective output compare channels 1 through 5.
2: The OCFA pin controls the OC1-OC3 channels. The OCFB pin controls the OC4-OC5 channels.
3: Each output compare channel can use one of two selectable 16-bit time bases or a single 32-bit timer base.

# PIC32MX FAMILY

TABLE 16-1: OUTPUT COMPARE SFR SUMMARY

| Virtual Address | Name | | Bit 31/23/15/7 | Bit 30/22/14/6 | Bit 29/21/13/5 | Bit 28/20/12/4 | Bit 27/19/11/3 | Bit 26/18/10/2 | Bit 25/17/9/1 | Bit 24/16/8/0 |
|---|---|---|---|---|---|---|---|---|---|---|
| BF80_3000 | OC1CON | 31:24 | — | — | — | — | — | — | — | — |
| | | 23:16 | — | — | — | — | — | — | — | — |
| | | 15:8 | ON | FRZ | SIDL | — | — | — | — | — |
| | | 7:0 | — | — | OC32 | OCFLT | OCTSEL | OCM<2:0> | | |
| BF80_3004 | OC1CONCLR | 31:0 | Write clears selected bits in OC1CON, Read yields an undefined value. | | | | | | | |
| BF80_3008 | OC1CONSET | 31:0 | Write sets selected bits in OC1CON, Read yields an undefined value. | | | | | | | |
| BF80_300C | OC1CONINV | 31:0 | Write inverts selected bits in OC1CON, Read yields an undefined value. | | | | | | | |
| BF80_3010 | OC1R | 31:24 | OC1R<31:24> | | | | | | | |
| | | 23:16 | OC1R<23:16> | | | | | | | |
| | | 15:8 | OC1R<15:8> | | | | | | | |
| | | 7:0 | OC1R<7:0> | | | | | | | |
| BF80_3014 | OC1RCLR | 31:0 | Write clears selected bits in OC1R, Read yields an undefined value. | | | | | | | |
| BF80_3018 | OC1RSET | 31:0 | Write sets selected bits in OC1R, Read yields an undefined value. | | | | | | | |
| BF80_301C | OC1RINV | 31:0 | Write inverts selected bits in OC1R, Read yields an undefined value. | | | | | | | |
| BF80_3020 | OC1RS | 31:24 | OC1RS<31:24> | | | | | | | |
| | | 23:16 | OC1RS<23:16> | | | | | | | |
| | | 15:8 | OC1RS<15:8> | | | | | | | |
| | | 7:0 | OC1RS<7:0> | | | | | | | |
| BF80_3024 | OC1RSCLR | 31:0 | Write clears selected bits in OC1RS, Read yields an undefined value. | | | | | | | |
| BF80_3028 | OC1RSSET | 31:0 | Write sets selected bits in OC1RS, Read yields an undefined value. | | | | | | | |
| BF80_302C | OC1RSINV | 31:0 | Write inverts selected bits in OC1RS, Read yields an undefined value. | | | | | | | |
| BF80_3200 | OC2CON | 31:24 | — | — | — | — | — | — | — | — |
| | | 23:16 | — | — | — | — | — | — | — | — |
| | | 15:8 | ON | FRZ | SIDL | — | — | — | — | — |
| | | 7:0 | — | — | OC32 | OCFLT | OCTSEL | OCM<2:0> | | |
| BF80_3204 | OC2CONCLR | 31:0 | Write clears selected bits in OC2CON, Read yields an undefined value. | | | | | | | |
| BF80_3208 | OC2CONSET | 31:0 | Write sets selected bits in OC2CON, Read yields an undefined value. | | | | | | | |
| BF80_320C | OC2CONINV | 31:0 | Write inverts selected bits in OC2CON, Read yields an undefined value. | | | | | | | |
| BF80_3210 | OC2R | 31:24 | OC2R<31:24> | | | | | | | |
| | | 23:16 | OC2R<23:16> | | | | | | | |
| | | 15:8 | OC2R<15:8> | | | | | | | |
| | | 7:0 | OC2R<7:0> | | | | | | | |
| BF80_3214 | OC2RCLR | 31:0 | Write clears selected bits in OC2R, Read yields an undefined value. | | | | | | | |
| BF80_3218 | OC2RSET | 31:0 | Write sets selected bits in OC2R, Read yields an undefined value. | | | | | | | |
| BF80_321C | OC2RINV | 31:0 | Write inverts selected bits in OC2R, Read yields an undefined value. | | | | | | | |
| BF80_3220 | OC2RS | 31:24 | OC2RS<31:24> | | | | | | | |
| | | 23:16 | OC2RS<23:16> | | | | | | | |
| | | 15:8 | OC2RS<15:8> | | | | | | | |
| | | 7:0 | OC2RS<7:0> | | | | | | | |
| BF80_3224 | OC2RSCLR | 31:0 | Write clears selected bits in OC2RS, Read yields an undefined value. | | | | | | | |
| BF80_3228 | OC2RSSET | 31:0 | Write sets selected bits in OC2RS, Read yields an undefined value. | | | | | | | |
| BF80_322C | OC2RSINV | 31:0 | Write inverts selected bits in OC2RS, Read yields an undefined value. | | | | | | | |
| BF80_3400 | OC3CON | 31:24 | — | — | — | — | — | — | — | — |
| | | 23:16 | — | — | — | — | — | — | — | — |
| | | 15:8 | ON | FRZ | SIDL | — | — | — | — | — |
| | | 7:0 | — | — | OC32 | OCFLT | OCTSEL | OCM<2:0> | | |
| BF80_3404 | OC3CONCLR | 31:0 | Write clears selected bits in OC3CON, Read yields an undefined value. | | | | | | | |
| BF80_3408 | OC3CONSET | 31:0 | Write sets selected bits in OC3CON, Read yields an undefined value. | | | | | | | |
| BF80_340C | OC3CONINV | 31:0 | Write inverts selected bits in OC3CON, Read yields an undefined value. | | | | | | | |
| BF80_3410 | OC3R | 31:24 | OC3R<31:24> | | | | | | | |
| | | 23:16 | OC3R<23:16> | | | | | | | |
| | | 15:8 | OC3R<15:8> | | | | | | | |
| | | 7:0 | OC3R<7:0> | | | | | | | |
| BF80_3414 | OC3RCLR | 31:0 | Write clears selected bits in OC3R, Read yields an undefined value. | | | | | | | |
| BF80_3418 | OC3RSET | 31:0 | Write sets selected bits in OC3R, Read yields an undefined value. | | | | | | | |
| BF80_341C | OC3RINV | 31:0 | Write inverts selected bits in OC3R, Read yields an undefined value. | | | | | | | |


© 2007 Microchip Technology Inc.

TABLE 16-1: OUTPUT COMPARE SFR SUMMARY (CONTINUED)

| Virtual Address | Name | | Bit 31/23/15/7 | Bit 30/22/14/6 | Bit 29/21/13/5 | Bit 28/20/12/4 | Bit 27/19/11/3 | Bit 26/18/10/2 | Bit 25/17/9/1 | Bit 24/16/8/0 |
|---|---|---|---|---|---|---|---|---|---|---|
| BF80_3020 | OC3RS | 31:24 | OC3RS<31:24> | | | | | | | |
| | | 23:16 | OC3RS<23:16> | | | | | | | |
| | | 15:8 | OC3RS<15:8> | | | | | | | |
| | | 7:0 | OC3RS<7:0> | | | | | | | |
| BF80_3024 | OC3RSCLR | 31:0 | Write clears selected bits in OC3RS, Read yields an undefined value | | | | | | | |
| BF80_3028 | OC3RSSET | 31:0 | Write sets selected bits in OC3RS, Read yields an undefined value | | | | | | | |
| BF80_302C | OC3RSINV | 31:0 | Write inverts selected bits in OC3RS, Read yields an undefined value | | | | | | | |
| BF80_3000 | OC4CON | 31:24 | — | — | — | — | — | — | — | — |
| | | 23:16 | — | — | — | — | — | — | — | — |
| | | 15:8 | ON | FRZ | SIDL | — | — | — | — | — |
| | | 7:0 | — | — | OC32 | OCFLT | OCTSEL | OCM<2:0> | | |
| BF80_3004 | OC4CONCLR | 31:0 | Write clears selected bits in OC4CON, read yields an undefined value | | | | | | | |
| BF80_3008 | OC4CONSET | 31:0 | Write sets selected bits in OC4CON, read yields an undefined value | | | | | | | |
| BF80_300C | OC4CONINV | 31:0 | Write inverts selected bits in OC4CON, read yields an undefined value | | | | | | | |
| BF80_3010 | OC4R | 31:24 | OC4R<31:24> | | | | | | | |
| | | 23:16 | OC4R<23:16> | | | | | | | |
| | | 15:8 | OC4R<15:8> | | | | | | | |
| | | 7:0 | OC4R<7:0> | | | | | | | |
| BF80_3014 | OC4RCLR | 31:0 | Write clears selected bits in OC4R, read yields an undefined value | | | | | | | |
| BF80_3018 | OC4RSET | 31:0 | Write sets selected bits in OC4R, read yields an undefined value | | | | | | | |
| BF80_301C | OC4RINV | 31:0 | Write inverts selected bits in OC4R, read yields an undefined value | | | | | | | |
| BF80_3020 | OC4RS | 31:24 | OC4RS<31:24> | | | | | | | |
| | | 23:16 | OC4RS<23:16> | | | | | | | |
| | | 15:8 | OC4RS<15:8> | | | | | | | |
| | | 7:0 | OC4RS<7:0> | | | | | | | |
| BF80_3024 | OC4RSCLR | 31:0 | Write clears selected bits in OC4RS, read yields an undefined value | | | | | | | |
| BF80_3028 | OC4RSSET | 31:0 | Write sets selected bits in OC4RS, read yields an undefined value | | | | | | | |
| BF80_302C | OC4RSINV | 31:0 | Write inverts selected bits in OC4RS, read yields an undefined value | | | | | | | |
| BF80_3000 | OC5CON | 31:24 | — | — | — | — | — | — | — | — |
| | | 23:16 | — | — | — | — | — | — | — | — |
| | | 15:8 | ON | FRZ | SIDL | — | — | — | — | — |
| | | 7:0 | — | — | OC32 | OCFLT | OCTSEL | OCM<2:0> | | |
| BF80_3004 | OC5CONCLR | 31:0 | Write clears selected bits in OC5CON, read yields an undefined value | | | | | | | |
| BF80_3008 | OC5CONSET | 31:0 | Write sets selected bits in OC5CON, read yields an undefined value | | | | | | | |
| BF80_300C | OC5CONINV | 31:0 | Write inverts selected bits in OC5CON, read yields an undefined value | | | | | | | |
| BF80_3010 | OC5R | 31:24 | OC5R<31:24> | | | | | | | |
| | | 23:16 | OC5R<23:16> | | | | | | | |
| | | 15:8 | OC5R<15:8> | | | | | | | |
| | | 7:0 | OC5R<7:0> | | | | | | | |
| BF80_3014 | OC5RCLR | 31:0 | Write clears selected bits in OC5R, read yields an undefined value | | | | | | | |
| BF80_3018 | OC5RSET | 31:0 | Write sets selected bits in OC5R, read yields an undefined value | | | | | | | |
| BF80_301C | OC5RINV | 31:0 | Write inverts selected bits in OC5R, read yields an undefined value | | | | | | | |
| BF80_3020 | OC5RS | 31:24 | OC5RS<31:24> | | | | | | | |
| | | 23:16 | OC5RS<23:16> | | | | | | | |
| | | 15:8 | OC5RS<15:8> | | | | | | | |
| | | 7:0 | OC5RS<7:0> | | | | | | | |
| BF80_3024 | OC5RSCLR | 31:0 | Write clears selected bits in OC5RS, read yields an undefined value | | | | | | | |
| BF80_3028 | OC5RSSET | 31:0 | Write sets selected bits in OC5RS, read yields an undefined value | | | | | | | |
| BF80_302C | OC5RSINV | 31:0 | Write inverts selected bits in OC5RS, read yields an undefined value | | | | | | | |
| 0xBF881000 | INTCON | 31:24 | IPTMR<31:24> | | | | | | | |
| | | 23:16 | IPTMR<23:16> | | | | | | | |
| | | 15:8 | — | FRZ | — | — | IPRST | TPC<2:0> | | |
| | | 7:0 | — | — | — | INT4EP | INT3EP | INT2EP | INT1EP | INT0EP |
| 0xBF881010 | IFS0 | 31:24 | I2CMIF | I2CSIF | U1EIF | U1TXIF | U1RXIF | SPI1RXIF | SPI1TXIF | SPI1EIF |
| | | 23:16 | CNIF | CS1IF | ICSIF | T5IF | INT4IF | OC4IF | IC4IF | T4IF |
| | | 15:8 | INT3IF | OC3IF | IC3IF | T3IF | INT2IF | OC2IF | IC2IF | T2IF |
| | | 7:0 | INT1IF | OC1IF | IC1IF | T1IF | INT0IF | CS0IF | CS0IF | CTIF |

# PIC32MX FAMILY

TABLE 16-1: OUTPUT COMPARE SFR SUMMARY (CONTINUED)

| Virtual Address | Name | | Bit 31/23/15/7 | Bit 30/22/14/6 | Bit 29/21/13/5 | Bit 28/20/12/4 | Bit 27/19/11/3 | Bit 26/18/10/2 | Bit 25/17/9/1 | Bit 24/16/8/0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0xBF881040 | IEC0 | 31:24 | I2CMIE | I2CSIE | U1EIE | U1TXIE | U1RXIE | SPI1RXIE | SPI1TXIE | SPI1EIE |
| | | 23:16 | CNIE | CMIE | IC5IE | T5IE | INT4IE | OC4IE | IC4IE | T4IE |
| | | 15:8 | INT3IE | OC3IE | IC3IE | T3IE | INT2IE | OC2IE | IC2IE | T2IE |
| | | 7:0 | INT1IE | OC1IE | IC1IE | T1IE | INT0IE | CS1IE | CS0IE | CTIE |
| 0xBF881050 | IPC1 | 31:24 | — | — | — | INT1IP<2:0> | | | INT1IS<1:0> | |
| | | 23:16 | — | — | — | OC1IP<2:0> | | | OC1IS<1:0> | |
| | | 15:8 | — | — | — | IC1IP<2:0> | | | IC1IS<1:0> | |
| | | 7:0 | — | — | — | T1IP<2:0> | | | T1IS<1:0> | |
| 0xBF881060 | IPC2 | 31:24 | — | — | — | INT2IP<2:0> | | | INT2IS<1:0> | |
| | | 23:16 | — | — | — | OC2IP<2:0> | | | OC2IS<1:0> | |
| | | 15:8 | — | — | — | IC2IP<2:0> | | | IC2IS<1:0> | |
| | | 7:0 | — | — | — | T2IP<2:0> | | | T2IS<1:0> | |
| 0xBF881070 | IPC3 | 31:24 | — | — | — | INT3IP<2:0> | | | INT3IS<1:0> | |
| | | 23:16 | — | — | — | OC3IP<2:0> | | | OC3IS<1:0> | |
| | | 15:8 | — | — | — | IC3IP<2:0> | | | IC3IS<1:0> | |
| | | 7:0 | — | — | — | T3IP<2:0> | | | T3IS<1:0> | |
| 0xBF881080 | IPC4 | 31:24 | — | — | — | INT4IP<2:0> | | | INT4IS<1:0> | |
| | | 23:16 | — | — | — | OC4IP<2:0> | | | OC4IS<1:0> | |
| | | 15:8 | — | — | — | IC4IP<2:0> | | | IC4IS<1:0> | |
| | | 7:0 | — | — | — | T4IP<2:0> | | | T4IS<1:0> | |
| 0xBF881090 | IPC5 | 31:24 | — | — | — | CNIP<2:0> | | | CNIS<1:0> | |
| | | 23:16 | — | — | — | OC5IP<2:0> | | | OC5IS<1:0> | |
| | | 15:8 | — | — | — | IC5IP<2:0> | | | IC5IS<1:0> | |
| | | 7:0 | — | — | — | T5IP<2:0> | | | T5IS<1:0> | |
| BF80_0800 | T2CON | 31:24 | — | — | — | — | — | — | — | — |
| | | 23:16 | — | — | — | — | — | — | — | — |
| | | 15:8 | ON | FRZ | SIDL | — | — | — | — | — |
| | | 7:0 | TGATE | TCKPS<2:0> | | | T32 | — | TCS | — |
| BF80_0A00 | T3CON | 31:24 | — | — | — | — | — | — | — | — |
| | | 23:16 | — | — | — | — | — | — | — | — |
| | | 15:8 | ON | FRZ | SIDL | — | — | — | — | — |
| | | 7:0 | TGATE | TCKPS<2:0> | | | — | — | TCS | — |
| BF80_0820 | PR2 | 31:24 | PR2<31:24> | | | | | | | |
| | | 23:16 | PR2<23:16> | | | | | | | |
| | | 15:8 | PR2<15:8> | | | | | | | |
| | | 7:0 | PR2<7:0> | | | | | | | |
| BF80_0A20 | PR3 | 31:24 | PR3<31:24> | | | | | | | |
| | | 23:16 | PR3<23:16> | | | | | | | |
| | | 15:8 | PR3<15:8> | | | | | | | |
| | | 7:0 | PR3<7:0> | | | | | | | |


© 2007 Microchip Technology Inc.

**REGISTER 16-1: OCxCON: OUTPUT COMPARE x CONTROL REGISTER**

| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |
| bit 31 | | | | | | | bit 24 |

| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |
| bit 23 | | | | | | | bit 16 |

| R/W-0 | R/W-0 | R/W-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
|---|---|---|---|---|---|---|---|
| ON | FRZ | SIDL | — | — | — | — | — |
| bit 15 | | | | | | | bit 8 |

| U-0 | U-0 | R/W-0 | R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| — | — | OC32 | OCFLT | OCTSEL | OCM<2:0> | | |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | P = Programmable bit | r = Reserved bit |
| U = Unimplemented bit | -n = Bit Value at POR: ('0', '1', x = Unknown) | | |

bit 31-16 **Unimplemented:** Read as '0'

bit 15 **ON:** Output Compare Peripheral On bit
1 = Output compare peripheral is enabled. The status of other bits in the register are not affected by setting this bit
0 = Output compare peripheral is disabled and not drawing current. SFR modifications are allowed. The status of other bits in this register are not affected by clearing this bit

bit 14 **FRZ:** Freeze in Debug Exception Mode bit(1)
1 = Freeze operation when CPU enters in Debug Exception mode
0 = Continue operation when CPU enters in Debug Exception mode

bit 13 **SIDL:** Stop in Idle Mode bit
1 = Discontinue operation when CPU enters in Idle mode
0 = Continue operation in Idle mode

bit 12-6 **Unimplemented:** Read as '0'

bit 5 **OC32:** 32-Bit Compare Mode bit
1 = OCxR<31:0> and/or OCxRS<31:0> are used for comparisons to the 32-bit timer source
0 = OCxR<15:0> and OCxRS<15:0> are used for comparisons to the 16-bit timer source

bit 4 **OCFLT:** PWM Fault Condition Status bit(2)
1 = PWM Fault condition has occurred (cleared in HW only)
0 = No PWM Fault condition has occurred
**Note:** (This bit is only used when OCM<2:0> = 111.

bit 3 **OCTSEL:** Output Compare Timer Select bit
1 = Timer3 is the clock source for compare x
0 = Timer2 is the clock source for compare x
**Note:** OCTSEL must be set to '1' when using 32-bit mode (OC32 = 1)

**Note 1:** FRZ is writable in Debug Exception mode only, it is forced to read '0' in Normal mode.
**2:** Reads as '0' in modes other than PWM mode.

bit 2-0 **OCM<2:0>**: Output Compare Mode Select bits
111 = PWM mode on OCx, Fault pin enabled
110 = PWM mode on OCx, Fault pin disabled
101 = Initialize OCx pin low, generate continuous output pulses on OCx pin
100 = Initialize OCx pin low, generate single output pulse on OCx pin
011 = Compare event toggles OCx pin
010 = Initialize OCx pin high, compare event forces OCx pin low
001 = Initialize OCx pin low, compare event forces OCx pin high
000 = Output compare peripheral is disabled

**Note 1:** FRZ is writable in Debug Exception mode only, it is forced to read '0' in Normal mode.
**2:** Reads as '0' in modes other than PWM mode.

 © 2007 Microchip Technology Inc.

**Register 16-1: OCxR: OUTPUT COMPARE x COMPARE PRIMARY REGISTER**

| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
|---|---|---|---|---|---|---|---|
| OCR<31:24> | | | | | | | |
| bit 31 | | | | | | | bit 24 |

| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
|---|---|---|---|---|---|---|---|
| OCR<23:16> | | | | | | | |
| bit 23 | | | | | | | bit 16 |

| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
|---|---|---|---|---|---|---|---|
| OCR<15:8> | | | | | | | |
| bit 15 | | | | | | | bit 8 |

| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
|---|---|---|---|---|---|---|---|
| OCR<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

| Legend: | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | P = Programmable bit | r = Reserved bit |
| U = Unimplemented bit | -n = Bit Value at POR: ('0', '1', x = Unknown) | | |

bit 31-16 OCxR<31:16>: Upper 16 bits of 32-bit compare value when OC32 (OCxCON<5>) = 1

bit 15-0 OCxR<15:0>: Lower 16 bits of 32-bit compare value or entire 16 bits of 16-bit compare value

# PIC32MX FAMILY

**Register 16-2: OCxRS: OUTPUT COMPARE x COMPARE SECONDARY REGISTER**

| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
|---|---|---|---|---|---|---|---|
| OCRS<31:24> | | | | | | | |
| bit 31 | | | | | | | bit 24 |

| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
|---|---|---|---|---|---|---|---|
| OCRS<23:16> | | | | | | | |
| bit 23 | | | | | | | bit 16 |

| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
|---|---|---|---|---|---|---|---|
| OCRS<5>8> | | | | | | | |
| bit 15 | | | | | | | bit 8 |

| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
|---|---|---|---|---|---|---|---|
| OCRS<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

| Legend: | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | P = Programmable bit | r = Reserved bit |
| U = Unimplemented bit | -n = Bit Value at POR: ('0', '1', x = Unknown) | | |

bit 31-16 OCxRS<31:16>: Upper 16 bits of 32-bit compare value, when OC32 (OCxCON<5>) = 1

bit 15-0 OCxRS<15:0>: Lower 16 bits of 32-bit compare value or entire 16 bits of 16-bit compare value


© 2007 Microchip Technology Inc.

### 16.1 Setup for Single Output Change

There are three modes of operation that change the state of the output pin; these modes can be referred to as drive high, drive low and toggle. The configuration for these modes is identical, the mode is selected by the OCM bits. For this example, Tx will represent Timer2.

Drive High: When the OCM control bits (OCxCON<2:0>) are set to '001', the selected output compare channel initializes the OCx pin to the low state and drives the output pin high when a compare event occurs.

Drive Low: When the OCM control bits (OCxCON<2:0>) are set to '010', the selected output compare channel initializes the OCx pin to the high state and drives the output pin low when a compare event occurs.

Toggle: When the OCM control bits (OCxCON<2:0>) are set to '011', the selected output compare channel OCx pin is not initialized. The OCx pin is driven to the opposite state when a compare event occurs.

To generate a output change signal, the following steps are required (these steps assume the timer source is initially turned off, but this is not a requirement for the module operation):

1. Determine the timer clock cycle time. Take into account the frequency of the external clock to the timer source (if one is used) and the timer prescaler settings.
2. Calculate time to the rising edge of the output pulse relative to the timer start value (0000h).
3. Determine if the output compare module will be used in 16 or 32-bit mode based on the previous calculations.
4. Configure the timer to be used as the time base for 16 or 32-bit mode by writing to the T32 bit (TxCON<T32>).
5. Configure the output compare channel for 16 or 32-bit operation by writing to the OC32 bit (OCxCON<5>).
6. Write the value computed in step 2 above into the Compare register, OCxR.
7. Set Timer Period register, PRx, to the value equal to or greater than the value in OCxRS, the Secondary Compare register.
8. Set the OCM bits to the desired mode of operation and the OCTSEL (OCxCON<3>) bit to the desired timer source. The OCx pin state will now be driven low.
9. Set the ON (TxCON<15>) bit to '1' which enables the compare time base to count.
10. Upon the first match between TMRx and OCxR, the OCx pin will be driven high.
11. When the incrementing timer, TMRx, matches the Secondary Compare register, OCxRS, the second and trailing edge (high-to-low) of the pulse is driven onto the OCx pin. No additional pulses are driven onto the OCx pin and it remains at low. As a result of the second compare match event, the OCxIF interrupt flag bit is set, which will result in an interrupt if it is enabled, by setting the OCxIE bit. For further information on peripheral interrupts, refer to Section 8.0 "Interrupts".
12. To initiate another single pulse output, change the Timer and Compare register settings, if needed, and then issue a write to set the OCM bits to the desired mode of operation. Disabling and re-enabling of the timer and clearing the Timer register are not required, but may be advantageous for defining a pulse from a known event time boundary.

### 16.2 Setup for Single Output Pulse Generation

When the OCM control bits (OCxCON<2:0>) are set to '100', the selected output compare channel initializes the OCx pin to the low state and generates a single output pulse.

To generate a single output pulse, the following steps are required (these steps assume the timer source is initially turned off, but this is not a requirement for the module operation): For this example Tx will represent Timer2.

1. Determine the timer clock cycle time. Take into account the frequency of the external clock to the timer source (if one is used) and the timer prescaler settings.
2. Calculate time to the rising edge of the output pulse relative to the timer start value (0000h).
3. Calculate the time to the falling edge of the pulse based on the desired pulse width and the time to the rising edge of the pulse.
4. Determine if the output compare module will be used in 16 or 32-bit mode based on the previous calculations.
5. Configure the timer to be used as the time base for 16 or 32-bit mode by writing to the T32 bit (TxCON<T32>).
6. Configure the output compare channel for 16 or 32-bit operation by writing to the OC32 bit (OCxCON<5>).
7. Write the values computed in steps 2 and 3 above into the Compare register, OCxR, and the Secondary Compare register, OCxRS, respectively.
8. Set Timer Period register, PRx, to the value equal to or greater than the value in the OCxRS, the Secondary Compare register.

9. Set the OCM bits to '100' and the OCTSEL (OCxCON<3>) bit to the desired timer source. The OCx pin state will now be driven low.
10. Set the ON (TxCON<15>) bit to '1' which enables the compare time base to count.
11. Upon the first match between TMRx and OCxR, the OCx pin will be driven high.
12. When the incrementing timer matches the Secondary Compare register, OCxRS, the second and trailing edge (high-to-low) of the pulse is driven onto the OCx pin. No additional pulses are driven onto the OCx pin and it remains at low. As a result of the second compare match event, the OCxIF interrupt flag bit is set, which will result in an interrupt if it is enabled, by setting the OCxIE bit. For further information on peripheral interrupts, refer to Section 8.0 "Interrupts".
13. To initiate another single pulse output, change the Timer and Compare register settings, if needed, and then issue a write to set the OCM bits to '100'. Disabling and re-enabling of the timer and clearing the TMRx register are not required, but may be advantageous for defining a pulse from a known event time boundary.

**EXAMPLE 16-1: EXAMPLE CODE**

```
// The following code example will set the Output Compare 1 module
// for interrupts on the single pulse event and select Timer 2
// as the clock source for the compare time base.

    T2CON = 0x0010;             // Configure Timer 2 for a prescaler of 2

    OC1CON = 0x0000;            // Turn off OC1 while doing setup.
    OC1CON = 0x0004;            // Configure for single pulse mode
    OC1R = 0x3000;              // Initialize primary Compare Register
    OC1RS = 0x3003;             // Initialize secondary Compare Register
    PR2 = 0x3003;               // Set period (PR2 is ever 32-bits wide)

                                // configure int
    IFS0CLR = 0x00000080;       // Clear the OC1 interrupt flag
    IEC0SET = 0x00000080;       // Enable OC1 interrupt
    IPC1SET = 0x0030000C;       // Set OC1 interrupt subpriority to 3,
                                // the highest level
    IPC1SET = 0x00000003;       // Set subpriority to 3, maximum

    T2CONSET = 0x8000;          // Enable timer2
    OC1CONSET = 0x8000;         // Enable the OC1 module


    // Example code for Output Compare 1 ISR:

#pragma interrupt OC1IntHandler ipl3 vector 6
void Cmp1IntHandler(void)
{
    // insert user code here
    IFS0CLR = 0x00000080;  // Clear the OC1 interrupt flag
}
```

 © 2007 Microchip Technology Inc.

### 16.3 Setup for Continuous Output Pulse Generation

When the OCM control bits (OCxCON<2:0>) are set to '101', the selected output compare channel initializes the OCx pin to the low state and generates output pulses on each and every compare match event.

For the user to configure the module for the generation of a continuous stream of output pulses, the following steps are required (these steps assume the timer source is initially turned off, but this is not a requirement for the module operation). For this example, Tx will represent Timer2.

1. Determine the timer clock cycle time. Take into account the frequency of the external clock to the timer source (if one is used) and the timer prescaler settings.
2. Calculate time to the rising edge of the output pulse relative to the TMRx start value (0000h).
3. Calculate the time to the falling edge of the pulse based on the desired pulse width and the time to the rising edge of the pulse.
4. Determine if the output compare module will be used in 16 or 32-bit mode based on the previous calculations.
5. Configure the timer to be used as the time base for 16 or 32-bit mode by writing to the T32 bit (TxCON<T32>).
6. Configure the output compare channel for 16 or 32-bit operation by writing to the OC32 bit (OCxCON<5>).
7. Write the values computed in step 2 and 3 above into the Compare register, OCxR, and the Secondary Compare register, OCxRS, respectively.
8. Set Timer Period register, PRx, to the value equal to or greater than the value in OCxRS, the Secondary Compare register.
9. Set the OCM bits to '101' and the OCTSEL bit to the desired timer source. The OCx pin state will now be driven low.
10. Enable the compare time base by setting the ON (TxCON<15>) bit to '1'.
11. Upon the first match between TMRx and OCxR, the OCx pin will be driven high.
12. When the compare time base, TMRy, matches the Secondary Compare register, OCxRS, the second and trailing edge (high-to-low) of the pulse is driven onto the OCx pin.
13. As a result of the second compare match event, the OCxIF interrupt flag bit set.
14. When the compare time base and the value in its respective Period register match, the TMRx register resets to 0x0000 and resumes counting.
15. Steps 8 through 11 are repeated and a continuous stream of pulses is generated, indefinitely. The OCxIF flag is set on each OCxRS-TMRx compare match event.

### 16.4 Pulse-Width Modulation Mode

There are two modes of PWM operation for this device: PWM and PWM with Fault input. The configuration of both modes is identical with the exception of the value written to the OCM bits to select the desired mode.

The following steps should be taken when configuring the output compare module for PWM operation:

1. Calculate the PWM period.
2. Calculate the PWM duty cycle.
3. Determine if the Output Compare module will be used in 16 or 32-bit mode based on the previous calculations.
4. Configure the timer to be used as the time base for 16 or 32-bit mode by writing to the T32 bit (TxCON<T32>).
5. Configure the output compare channel for 16 or 32-bit operation by writing to the OC32 bit (OCxCON<5>).
6. Set the PWM period by writing to the selected Timer Period register (PR).
7. Set the PWM duty cycle by writing to the OCxRS register.
8. Write the OCxR register with the initial duty cycle.
9. Enable interrupts, if required, for the timer and output compare modules. The output compare interrupt is required for PWM Fault pin utilization.
10. Configure the output compare module for one of two PWM operation modes by writing to the Output Compare mode bits OCM<2:0> (OCxCON<2:0>).
11. Set the TMRx prescale value and enable the time base by setting ON (TxCON<15>) = 1.

| Note: | The OCxR register should be initialized before the output compare module is first enabled. The OCxR register becomes a read-only Duty Cycle register when the module is operated in the PWM modes. The value held in OCxR will become the PWM duty cycle for the first PWM period. The contents of the Duty Cycle Buffer register, OCxRS, will not be transferred into OCxR until a time base period match occurs. |
|---|---|

# PIC32MX FAMILY

### 16.4.1 PWM PERIOD

The PWM period is specified by writing to PR, the Timer Period register. The PWM period can be calculated using Equation 16-1.

**EQUATION 16-1: CALCULATING THE PWM PERIOD**

PWM Period = [(PRy) + 1] • TPB • (TMRy Prescale Value)]

PWM Frequency = 1/[PWM Period]

**Note:** A PRy value of N will produce a PWM period of N + 1 time base count cycles. For example, a value of 7 written into the PRy register will yield a period consisting of 8 time base cycles.

### 16.4.2 PWM DUTY CYCLE

The PWM duty cycle is specified by writing to the OCxRS register. The OCxRS register can be written to at any time, but the duty cycle value is not latched into OCxR until a match between the PR and Timer occurs (i.e., the period is complete). This provides a double buffer for the PWM duty cycle and is essential for glitch-less PWM operation. In the PWM mode, OCxR is a read-only register.

Some important boundary parameters of the PWM duty cycle include:

- If the Duty Cycle register, OCxR, is loaded with 0000h, the OCx pin will remain low (0% duty cycle).
- If OCxR is greater than PR (Timer Period register), the pin will remain high (100% duty cycle).
- If OCxR is equal to PR, the OCx pin will be low for one time base count value and high for all other count values.

See Example 16-2 for PWM mode timing details. Table 16-2 shows example PWM frequencies and resolutions for a device peripheral bus operating at 10 MHz.

**EQUATION 16-2: CALCULATION FOR MAXIMUM PWM RESOLUTION**

$$\text{Maximum PWM Resolution (bits)} = \frac{\log_{10}\left(\frac{F_{PB}}{F_{PWM} \cdot \text{TMRy Prescaler}}\right)}{\log_{10}(2)} \text{ bits}$$

**EXAMPLE 16-2: PWM PERIOD AND DUTY CYCLE CALCULATION**

Desired PWM frequency is 52.08 kHz,
FPB = 10 MHz
Timer 2 prescale setting: 1:1

1/52.08 kHz = (PR2 + 1) • FPB • (Timer2 prescale value)
19.20 µs = (PR2 + 1) • 0.1 µs • (1)
PR2 = 191

Find the maximum resolution of the duty cycle that can be used with a 52.08 kHz PWM frequency and a 10 MHz peripheral bus clock rate:

1/52.08 kHz = $2^{\text{PWM RESOLUTION}}$ • 1/10 MHz • 1
19.20 µs = $2^{\text{PWM RESOLUTION}}$ • 100 ns • 1
192 = $2^{\text{PWM RESOLUTION}}$
$\log_{10}(192)$ = (PWM Resolution) • $\log_{10}(2)$
PWM Resolution= 7.6 bits

**Note:** If the PR value exceeds 16-bits the module must be used in 32-bit mode to maintain the calculated PWM resolution. If reduced resolution is acceptable the Timer prescaler may be increased and the calculation repeated until the result is a 16-bit value. Increasing the Timer prescaler to allow operation in 16-bit mode may result in reduced PWM resolution.


© 2007 Microchip Technology Inc.

**EXAMPLE 16-3: PWM MODE PULSE SETUP AND INTERRUPT SERVICING (32-BIT MODE)**

```
// The following code example will set the Output Compare 1 module
// for PWM mode with FAULT pin disabled, a 50% duty cycle and a
// PWM frequency of 52.08 kHz at Fpb = 40 MHz. Timer2 is selected as
// the clock for the PWM time base and Output Compare 1 interrupts
// are enabled.

    OC1CON = 0x0000;            // Turn off OC1 while doing setup.
    OC1R = 0x00600000;          // Initialize primary Compare Register
    OC1RS = 0x00600000;         // Initialize secondary Compare Register
    OC1CON = 0x0006;            // Configure for PWM mode, Fault pin Disabled
    PR2 = 0x00600000;           // Set period

                                // configure int
    IFS0 &= ~0x00000080;        // Clear the OC1 interrupt flag
    IEC0 |= 0x00000080;         // Enable OC1 interrupt
    IPC1 |= 0x001C0000;         // Set OC1 interrupt priority to 7,
                                // the highest level
    IPC1 |= 0x00000003;         // Set subpriority to 3, maximum

    T2CON |= 0x8000;            // Enable timer2
    OC1CON |= 0x8000;           // turn on OC1 module


// Example code for Output Compare 1 ISR:
#pragma interrupt OC1IntHandler ipl4 vector 36
void OC1IntHandler(void)
{
              // insert user code here
                  IFS0CLR = 0x00000080; // Clear the interrupt flag
}
```

# PIC32MX FAMILY

TABLE 16-2: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS WITH PERIPHERAL BUS CLOCK OF 10 MHZ (16-BIT MODE)

| PWM Frequency | 19.6 Hz | 153 Hz | 305 Hz | 2.44 kHz | 9.77 kHz | 78.1 kHz | 313 kHz |
|---|---|---|---|---|---|---|---|
| Timer Prescaler Ratio | 8 | 1 | 1 | 1 | 1 | 1 | 1 |
| Period Register Value (hex) | 0xFA65 | 0xFF4E | 0x8011 | 0x1001 | 0x03FE | 0x007F | 0x001E |
| Resolution (bits) (decimal) | 16 | 16 | 15 | 12 | 10 | 7 | 5 |

TABLE 16-3: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS WITH PERIPHERAL BUS CLOCK OF 30 MHZ (16-BIT MODE)

| PWM Frequency | 58 Hz | 458 Hz | 916 Hz | 7.32 kHz | 29.3 kHz | 234 kHz | 938 kHz |
|---|---|---|---|---|---|---|---|
| Timer Prescaler Ratio | 8 | 1 | 1 | 1 | 1 | 1 | 1 |
| Period Register Value (hex) | 0xFC8E | 0xFFDD | 0x7FEE | 0x1001 | 0x03FE | 0x007F | 0x001E |
| Resolution (bits) (decimal) | 16 | 16 | 15 | 12 | 10 | 7 | 5 |

TABLE 16-4: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS WITH PERIPHERAL BUS CLOCK OF 50 MHZ (16-BIT MODE)

| PWM Frequency | 58 Hz | 458 Hz | 916 Hz | 7.32 kHz | 29.3 kHz | 234 kHz | 938 kHz |
|---|---|---|---|---|---|---|---|
| Timer Prescaler Ratio | 64 | 8 | 1 | 1 | 1 | 1 | 1 |
| Period Register Value (hex) | 0x349C | 0x354D | 0xD538 | 0x1AAD | 0x06A9 | 0x00D4 | 0x0034 |
| Resolution (bits) (decimal) | 13.7 | 13.7 | 15.7 | 12.7 | 10.7 | 7.7 | 5.7 |

TABLE 16-5: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS WITH PERIPHERAL BUS CLOCK OF 50 MHZ (16-BIT MODE)

| PWM Frequency | 100 Hz | 200 Hz | 600 Hz | 1 kHz | 2 kHz | 6 kHz | 10 kHz |
|---|---|---|---|---|---|---|---|
| Timer Prescaler Ratio | 8 | 8 | 8 | 1 | 8 | 1 | 1 |
| Period Register Value (hex) | 0xF423 | 0x7A11 | 0x30D3 | 0xC34F | 0x0C34 | 0x270F | 0x1387 |
| Resolution (bits) (decimal) | 15.9 | 14.9 | 13.6 | 15.6 | 11.6 | 13.3 | 12.3 |

TABLE 16-6: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS WITH PERIPHERAL BUS CLOCK OF 50 MHZ (16-BIT MODE)

| PWM Frequency | 100 Hz | 200 Hz | 600 Hz | 1 kHz | 2 kHz | 6 kHz | 10 kHz |
|---|---|---|---|---|---|---|---|
| Timer Prescaler Ratio | 8 | 4 | 2 | 1 | 1 | 1 | 1 |
| Period Register Value (hex) | 0xF423 | 0xF423 | 0xC34F | 0x0C34F | 0x61A7 | 0x270F | 0x1387 |
| Resolution (bits) (decimal) | 15.9 | 15.9 | 15.6 | 15.6 | 14.6 | 13.3 | 12.3 |

TABLE 16-7: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS WITH PERIPHERAL BUS CLOCK OF 50 MHZ (32-BIT MODE)

| PWM Frequency | 100 Hz | 200 Hz | 600 Hz | 1 kHz | 2 kHz | 6 kHz | 10 kHz |
|---|---|---|---|---|---|---|---|
| Period Register Value (hex) | 1 | 1 | 1 | 1 | 1 | 8 | 1 |
| Resolution (bits) (decimal) | 0x0007A11F | 0x0003D08F | 0x00019B9F | 0x0000C34F | 0x000061A7 | 0x000004E1 | 0x00001387 |
| Resolution (bits) | 18.9 | 17.9 | 16.6 | 15.6 | 14.6 | 10.3 | 12.3 |


© 2007 Microchip Technology Inc.

### 16.5 Output Compare Register I/O Pin Control

When the output compare module is enabled, the I/O pin direction is controlled by the compare module. The compare module returns the I/O pin control back to the appropriate pin LAT and TRIS control bits when it is disabled.

When the PWM with Fault Protection Input mode is enabled, the OCFx Fault pin must be configured as an input by setting the respective TRIS SFR bit. The OCFx Fault input pin is not automatically configured as an input when PWM with Fault Input mode is selected.

TABLE 16-8: PINS ASSOCIATED WITH OUTPUT COMPARE MODULES 1- 5

| Pin Name | Module Control | Controlling Bit Field | Required TRIS bit Setting | Pin Type | Buffer Type | Description |
|---|---|---|---|---|---|---|
| OC1 | ON(2) | OCM<2:0>(1,3) | — | D, O | — | Output Compare/PWM Channel 1 |
| OC2 | ON(2) | OCM<2:0>(1,3) | — | D, O | — | Output Compare/PWM Channel 2 |
| OC3 | ON(2) | OCM<2:0>(1,3) | — | D, O | — | Output Compare/PWM Channel 3 |
| OC4 | ON(2) | OCM<2:0>(1,3) | — | D, O | — | Output Compare/PWM Channel 4 |
| OC5 | ON(2) | OCM<2:0>(1,3) | — | D, O | — | Output Compare/PWM Channel 5 |
| OCFA | ON(2) | OCM<2:0>(1,3) | Input | D, I | ST | PWM Fault Protection A Input (For Channels 1-3)(4) |
| OCFB | ON(2) | OCM<2:0>(1,3) | Input | D, I | ST | PWM Fault Protection B Input (For Channels 4 -5)(4) |

**Legend:** ST = Schmitt Trigger input with CMOS levels, I = Input, O = Output, A = Analog, D = Digital

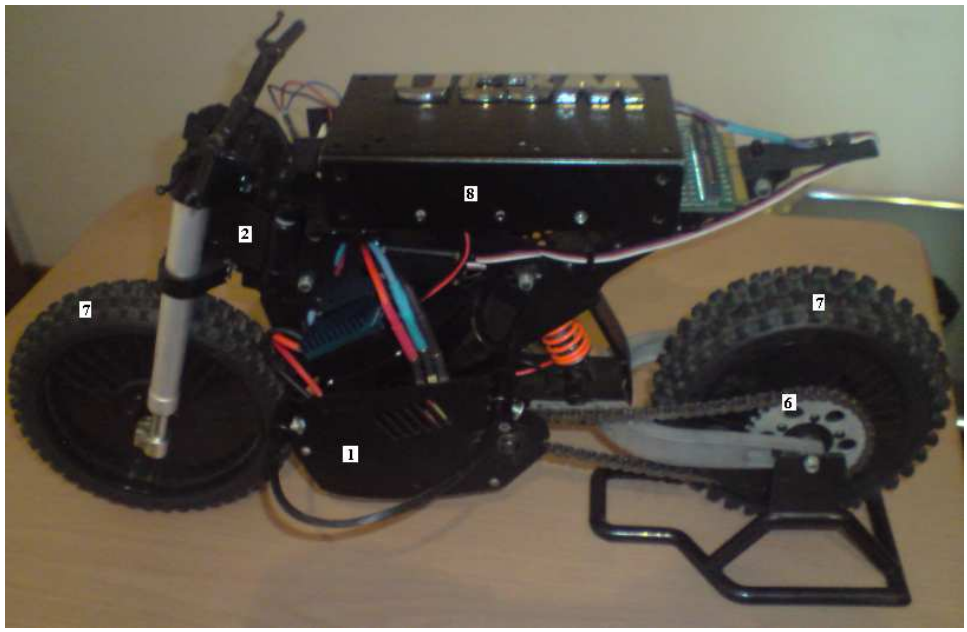
**Note 1:** All pins are subject to device pin priority control.

**2:** ON (OCxCON<15>) = 1. When the module is turned off, pins controlled by the module are released.

**3:** Mode select bits OCM<2:0> (CMxCON<2:0>).

**4:** Use of PWM Fault input is optional and is controlled by the OCM bits.

# ANEXO B.




1.- Motor brushless
2.- Servomotor
3.- Variador
4.- Giróscopo
5.- Puente H
6.- Batería 7.2 V
7.- Batería 9 V
8.- Microcontrolador