

PIC18F2550 y USB

Desarrollo de aplicaciones

INTRODUCCIÓN AL PUERTO USB:

USB es una especificación de las empresas Compaq, Intel, Microsoft y NEC, que describe un canal serie que soporta una gran variedad de periféricos de media y baja velocidad, con soporte integral para transferencias en tiempo real (isócronas) como voz, audio y vídeo comprimido, y que permite mezclar dispositivos y aplicaciones isócronas y asíncronas. Por lo tanto, entre los dispositivos USB más característicos se pueden citar teclados, ratones, joysticks, tabletas gráficas, monitores, modems, impresoras, escáneres, CD-ROMs, dispositivos de audio (como micrófonos o altavoces digitales), cámaras digitales y otros dispositivos multimedia.

La versión 1.1 (**La que soporta el PIC18F2550**) establece:

- Un acceso al bus gestionado directamente por el Controlador USB, para permitir transferencias isócronas y eliminar los tiempos de arbitración.
- Una velocidad de 12 Mbps (Full Speed o FS) y un subcanal de 1,5 Mbps (Low Speed o LS) para los dispositivos más lentos, como ratones y joysticks. La coexistencia en un mismo sistema de dispositivos FS y LS se maneja mediante conmutación automática y dinámica de velocidad entre unas transferencias y otras.
- Una conectividad excepcional, ya que puede manejar hasta 127 dispositivos simultáneamente que se pueden conectar y desconectar en caliente, sin tener que reiniciar el sistema.
- Una configuración automática de dispositivos, que elimina la necesidad de realizar configuraciones manuales por medio de puentes o conmutadores.
- La coexistencia de dispositivos isócronos y asíncronos. Los dispositivos isócronos se atienden en función del ancho de banda y latencia requeridos, y los asíncronos se atienden durante el tiempo restante no consumido por los dispositivos isócronos.
- Una distribución de alimentación desde el Controlador USB, que permite la conexión tanto de dispositivos alimentados desde el bus como autoalimentados.
- Una arquitectura fácilmente escalable para permitir la existencia de varios Controladores USB en un sistema.
- La versión 1.1 es soportada por los siguientes sistemas operativos: Windows 98\Windows 2000\Windows XP\Windows Vista\Windows 7 y además los siguientes OS ajenos a windows: Linux\Mac OS.

NIVEL FISICO:

A nivel físico, USB utiliza un cable de 4 conductores para transmitir una señal diferencial (D+ y D-) y alimentación (VBus = 5V y GND) por medio de conexiones punto a punto. Los dispositivos LS van obligatoriamente equipados con un cable de longitud adecuada (hasta unos 3m, dependiendo de sus características eléctricas), mientras que los FS pueden ir equipados con un cable o utilizar cables independientes de hasta 5m (también dependiendo de sus características eléctricas).

La comunicación es bidireccional y semi-dúplex, y utiliza codificación autoreloj NRZI (la línea cambia de nivel si se transmite un 0 y no cambia si transmite un 1) con "bit stuffing" (inserción de un cero tras la transmisión de 6 unos, para asegurar transiciones en la línea y permitir que la PLL del receptor se mantenga sincronizada). Los dispositivos disponen de un transmisor diferencial, receptores diferencial y S/E y resistencias de terminación con los que pueden transmitir y detectar varios estados eléctricos distintos en la línea:

- Transmisión/Recepción diferencial de bits: Estados DIFF0 y DIFF1, denominados también estados J y K.
- SE0 (Single-Ended 0): Ambas señales D+ y D- a 0V. Se utiliza para detectar la conexión/desconexión de dispositivos, para indicar el EOP (fin de paquete) y para generar reset.
- IDLE: reposo o línea en alta impedancia, necesario para permitir transferencias semi-dúplex, detectar la conexión y desconexión de dispositivos y discriminar entre dispositivos FS y LS.
- El SOP (principio de paquete) se indica mediante una transición IDLE a K.
- El EOP (fin de paquete) se indica mediante una secuencia SE0 (2 bits) + J (1 bit) + IDLE.
- Detección de dispositivo y discriminación FS/LS: cuando el transmisor deja la línea en IDLE, si hay un dispositivo conectado su polarización fuerza un estado J (DIFF0 si LS ó DIFF1 si FS), y si no lo hay, la polarización del transmisor fuerza un estado SE0.
- Reset: transmisión de SE0 durante ≥ 10 ms.

HUBS:

Dentro de la arquitectura USB, unos elementos esenciales y especiales son los Hubs (concentradores), que proveen conectividad (los dispositivos no se conectan entre sí directamente, sino cada uno a un hub), detectan la conexión y desconexión de dispositivos y si son FullSpeed o LowSpeed, generan alimentación hacia los dispositivos e incorporan la terminación de las líneas. Los Hubs disponen de una conexión "Upstream" hacia el ordenador y una o varias conexiones "Downstreams" hacia dispositivos u otros Hubs (concentrando efectivamente varios puntos de conexión en uno sólo), de forma que se pueden encadenar varios Hubs para formar una topología en varios niveles. Como a los Hub se conectan los dispositivos en estrella, la topología USB se denomina Estrella en Niveles. USB permite hasta 6 niveles, y en el nivel 0 (Raíz o Root) se encuentra el Controlador USB, que controla todo el tráfico de información en el bus. Los Hubs podrán ir integrados en algunos dispositivos (del estilo de teclados, impresoras y monitores), y también estarán disponibles como elementos independientes. Normalmente los Hubs serán autoalimentados, aunque bajo ciertas restricciones topológicas podrían utilizarse Hubs alimentados desde el bus. El hub dispone de un Repetidor, para pasar información entre el puerto Upstream y los puertos Downstreams, y de un Controlador, que incorpora un juego de registros a través de los cuales el Controlador USB configura el hub y controla y monitoriza los puertos Downstream.

CONEXIÓN / DESCONEXIÓN EN CALIENTE:

USB permite conectar un nuevo dispositivo sin tener que reiniciar el sistema. Cuando un Hub detecta una nueva conexión se lo comunica al Controlador USB, el sistema interroga al nuevo dispositivo, determina sus propiedades y posibilidades y lo configura. Adicionalmente el sistema carga el manejador adecuado al tipo de dispositivo con lo que el usuario puede empezar a trabajar con el mismo inmediatamente. Algunos sistemas operativos llaman a este proceso "Bus Enumeration".

PROTOCOLO USB:

El protocolo de nivel físico se basa en tokens (testigos). El controlador USB transmite tokens que incluyen la dirección del dispositivo destino, y el dispositivo que detecta su dirección en el Token responde y lleva a cabo la transferencia de datos con el controlador. De esta manera, el Controlador USB maneja la parte más compleja del protocolo, generando los tokens de transferencias de datos a 12 Mbps o a 1,5 Mbps, y controlando la conexión lógica entre el sistema y las funciones internas de cada dispositivo. El controlador USB también maneja el consumo en el bus a través de las funciones Suspend/Continuar, por medio de las cuales controla los modos Reposo/Activo de los dispositivos. Esta arquitectura permite el diseño de dispositivos extremadamente simples y de bajo coste.

USB divide el tiempo en espacios de 1ms denominados Tramas, durante las cuales se llevan a cabo las comunicaciones a través de Transacciones, las cuales se componen a su vez de Paquetes. Las transacciones se componen de 3 fases: Token, Dato y Validación (Handshake):

- La fase de Token se compone de un paquete de Token enviado por el Controlador USB, y siempre está presente en toda transacción. El paquete contiene los campos:
 - PID (identifica el tipo de paquete). Todos los PIDs van protegidos por bits redundantes,
 - Dirección del elemento destino (7 bits de dispositivo + 4 bits de elemento interno al dispositivo), y CRC5.
- La fase de Datos (opcional) se compone de los paquetes de datos que se transfieren entre el Controlador USB y el dispositivo. Cada paquete se compone de los campos PID, Datos, y CRC16.
- La fase de Validación (opcional) se usa para indicar el resultado de la transacción. Se compone sólo de un campo PID.

Adicionalmente, el Controlador USB indica el principio de cada Trama y la transmisión hacia dispositivos LS mediante tokens especiales.

TIPOS DE TRANSFERENCIA DE DATOS:

USB soporta 4 tipos de transferencias de datos:

- Control, para configuración y control de dispositivos y para manejo del bus.
- Isócrono, para transmisión de información con ancho de banda y latencia garantizadas, necesario para aplicaciones como audio, telefonía y vídeo. Permite una comunicación periódica y continua entre el sistema y el dispositivo.
- Interrupción, para transferencias de pocos datos, no periódicas, de baja frecuencia pero con unos ciertos límites de latencia.
- Bulk, para transferencias de grandes cantidades de datos con dispositivos asíncronos, como impresoras, escáneres, cámaras de fotos (foto fija), etc.

El PIC18F2550 soporta la transferencia interruptiva (Mouse, teclado y cualquier dispositivo HID) y transferencias tipo Bulk (Paquetes) en dispositivos como por ejemplo osciloscopios USB.

Desarrollaremos entonces la transferencia de control, interruptiva y Bulk.

TRANSFERENCIA DE CONTROL:

- Se desarrollan en 3 Transacciones:
- Transacción de Configuración (Setup), en la que se envía al dispositivo un paquete que especifica la operación a ejecutar. Ocupa 8 bytes.
- Cero o más Transacciones de Datos, en las que se transfieren los paquetes de datos en el sentido indicado por la Transacción de Configuración. La información útil por paquete puede ser de 8, 16, 32 ó 64 bytes para Endpoints FS, y de 8 bytes para Endpoints LS.
- Transacción de Estado, en la que el receptor informa del estado final de la operación.
- Se procesan por medio de un mecanismo "best effort", según el cual el Controlador USB las va procesando en función del tiempo disponible en cada Trama. Como mínimo se reserva el 10% del tiempo de Trama, y se puede utilizar tiempo adicional siempre que las necesidades de los tráfico isócrono y de interrupción lo permitan.
- Incorporan mecanismos de detección de errores (CRC) y de recuperación/retransmisión de datos.

TRANSFERENCIAS DE INTERRUPCIÓN:

- Aseguran una transacción (paquete) dentro de un periodo máximo (los dispositivos FS pueden solicitar entre 1 y 255ms, y los LS entre 10 y 255ms de periodo máximo de servicio).
- Incorpora detección de errores y retransmisión de datos.
- La información útil por paquete puede oscilar entre 1 y 64 bytes para dispositivos FS y entre 1 y 8 bytes para dispositivos LS.
- El sistema puede asignar como máximo el 90% del tiempo de Trama para transferencias isócronas y de interrupción. Si el sistema no puede garantizar tiempo suficiente como para manejar una nueva conexión de interrupción (transmitir un nuevo paquete dentro del periodo máximo requerido), simplemente no se establece la conexión.

TRANSFERENCIAS BULK:

- Sólo son utilizables por dispositivos FS.
- Se procesan por medio de un mecanismo "good effort", en el que el sistema aprovecha cualquier ancho de banda disponible y en el momento en que esté disponible (en otras palabras, no se garantiza una latencia ni un ancho de banda mínimos). Se puede utilizar el tiempo de Trama reservado y no consumido por transferencias de Control (10%).
- Incorporan mecanismos de control de errores para garantizar la entrega de datos.
- La información útil por paquete puede ser de 8, 16, 32 ó 64 bytes.

Estos 4 tipos de transferencias están disponibles como interfaces software que el sistema pone a disposición de los manejadores de dispositivo, estando los manejadores obligados a comunicarse con los dispositivos única y exclusivamente a través de estos 4 interfaces de programación. Esto viene a significar que un manejador de dispositivo USB jamás accede directamente al hardware del dispositivo, y por otro lado significa que todos los dispositivos USB deben cumplir necesariamente unas especificaciones básicas comunes, ya que deben gestionar adecuadamente los tipos de transferencias que soportan. Adicionalmente, los dispositivos USB se agrupan en Clases, de forma que todos los dispositivos de una misma Clase cumplen además con las especificaciones de dicha Clase, ya que la Clase incide directamente en la manera en que el software interactúa con el dispositivo.

MODELO LÓGICO:

Los dispositivos USB pueden tener una o más Configuraciones posibles, que definen distintas formas de funcionamiento. A nivel lógico, una determinada Configuración es un conjunto de Interfaces, donde cada Interfaz especifica qué partes del hardware del dispositivo se comunican con el sistema, donde cada una de estas partes de hardware se denomina Endpoints. En resumen, cada posible Configuración de un dispositivo USB es un conjunto de Interfaces y cada Interfaz es un conjunto de Endpoints. Los Endpoints son unidireccionales, y se direccionan por un número y por el sentido en que transfieren la información (IN (entrada) si transfieren información hacia el sistema, y OUT (salida) si transfieren información hacia el dispositivo).

La comunicación entre una aplicación y los distintos Endpoints de un dispositivo se realiza a través de USB por medio de unos caminos lógicos de transferencias de datos denominados Pipes, de forma que cada Pipe comunica la aplicación con un determinado Endpoint en el dispositivo. Los Pipes pueden ser de tipo Control (también denominadas de Mensaje), que son bidireccionales y con formato especificado por la norma, y de tipo Stream, que son unidireccionales (tipo FIFO) y con formato libre no especificado por la norma. Los Pipes de Control conectan la aplicación con un Endpoint de Control (formado por una pareja de Endpoints uno IN y otro OUT) para realizar transferencias bidireccionales de Control. Los Pipes Stream conectan la aplicación con un Endpoint para realizar transferencias unidireccionales Isócronas, Interrupción y Bulk.

Todos los dispositivos USB deben implementar los dos Endpoints 0 (IN y OUT) para permitir que el sistema pueda establecer el Pipe de Control por Defecto y pueda acceder a información de identificación y requisitos de configuración y pueda configurar el dispositivo. Adicionalmente, USB permite direccionar otros 15 Endpoints IN y 15 Endpoints OUT por dispositivo FS y otros 2 Pipes de Control y/o Interrupción por dispositivo LS. Estos Endpoints adicionales son opcionales y dependientes de los requisitos de implementación del dispositivo.

CLASES USB:

Una Clase USB es un grupo de dispositivos (o interfaces) con atributos o características similares. Las especificaciones para cada Clase permiten el desarrollo de dispositivos que pueden controlarse por medio de un manejador adaptativo, es decir, que se configura según la Clase reportada por el dispositivo. Dos dispositivos (o interfaces) pertenecen a la misma Clase si por ejemplo utilizan una misma forma de comunicarse con el sistema, o si por ejemplo utilizan el mismo formato de datos.

Las Clases USB se usan principalmente para describir la manera en que los dispositivos (o interfaces) se comunican con el sistema, incluyendo los mecanismos de control y datos, y adicionalmente algunas Clases se usan para identificar en todo o en parte la funcionalidad del dispositivo (o interfaz). En este caso, la Clase se puede utilizar para identificar qué manejador debe controlar dicho dispositivo (o interfaz).

Adicionalmente, los dispositivos de una Clase pueden agruparse en Subclases, lo que facilita aún más el que los manejadores puedan explorar el bus y seleccionar todos aquellos dispositivos que pueda controlar.

Antes de seguir con nuestro curso vamos a describir lo que hasta ahora hemos Visto:

Host: Dispositivo maestro que inicia la comunicación (Generalmente la computadora).

Hub: Dispositivo que contiene uno o mas conectores o conexiones internas hacia otros dispositivos usb, el cual habilita la comunicación entre el host y con diversos dispositivos. Cada conector representa un puerto USB.

Dispositivo compuesto: Es aquel dispositivo con múltiples interfaces independientes. Cada una tiene una dirección sobre el bus para cada interfase puede tener un diferente driver device en el host.

Puerto USB: Cada host soporta solo un bus, cada conector en el bus representa un puerto USB por lo tanto sobre el bus puede haber un varios conectores, pero solo existe una ruta y solo un dispositivo puede transmitir información a un tiempo.

Driver: es un programa que habilita aplicaciones para poderse comunicar con el dispositivo. Cada dispositivo sobre el bus debe tener un driver, algunos periféricos utilizan los drivers que trae Windows.

Puntos terminales (Endpoints): Es una localidad específica dentro del dispositivo. El Endpoint es un buffer que almacena múltiples bytes, típicamente es un bloque de la memoria de datos o un registro dentro del microcontrolador. Todos los dispositivos deben soportar el punto terminal 0. Este punto terminal es el que recibe todo el control y las peticiones de estado durante la enumeración cuando el dispositivo está sobre el bus.

Tuberías (Pipes): Es un enlace virtual entre el host (la PC) y el dispositivo USB, este enlace configura los parámetros asociados con el ancho de banda que tipo de transferencia se va a utilizar (**Control, Bulk, Isócrona o Interrupt**) dirección del flujo de datos y el máximo y/o mínimo tamaño de los paquetes/buffers.

Cada enlace está caracterizado por su banda de paso (Token), su tipo de servicio, el número de punto terminal (End Point) y el tamaño de los paquetes. Estos enlaces se definen y crean durante la inicialización del USB. Siempre existe un enlace virtual 0 que permite tener acceso a la información de configuración del periférico USB (estado, control e información). La norma USB define 2 tipos de enlaces virtuales (pipe); *stream* y *message*.

Stream Pipes: se trata de un flujo sin formato USB definido, esto significa que se puede enviar cualquier tipo de dato. Este tipo de pipe soporta las transferencias **bulk, isócronas, e interrupt**. Además tanto el host como el dispositivo USB pueden controlar.

Message Pipes: este tipo de enlace virtual si tiene un formato USB definido y solo puede soportar la transferencia **Control**.

Cuando se conecta un dispositivo USB a la PC se produce el **Proceso de Enumeración**, el cual consiste en que el host le pregunta al dispositivo que se presente y le diga cuales son sus parámetros, tales como:

- Consumo de energía expresada en unidades de Carga.
- Numero y tipos de Puntos terminales.
- Clase del producto.
- Tipo de transferencia.
- Razón de escrutinio, etc.

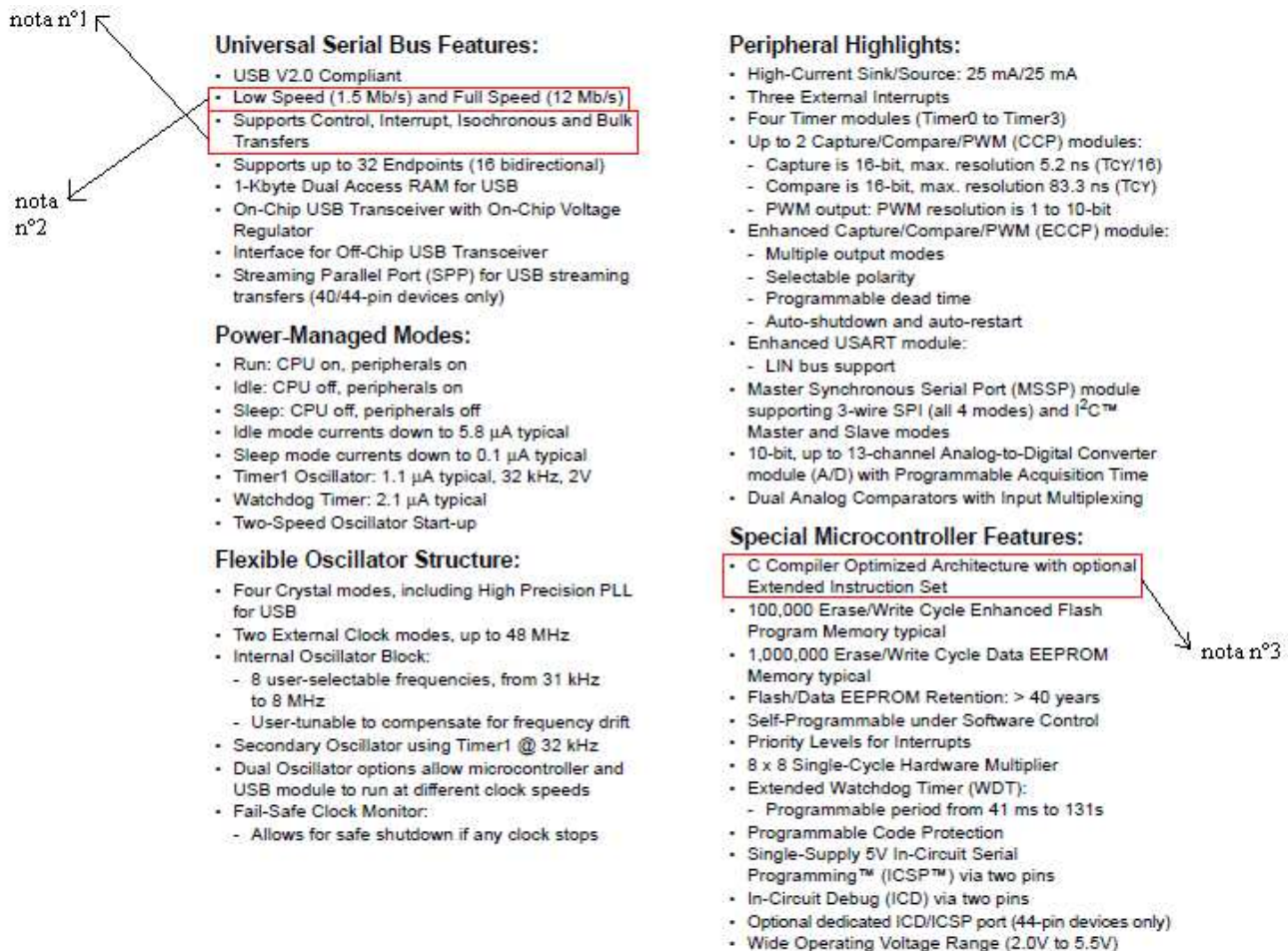
El proceso de enumeración es inicializado por el host cuando detecta que un nuevo dispositivo que ha sido adjuntado al Bus. El host le asigna una dirección al dispositivo adjuntado al bus y habilita su configuración permitiendo la transferencia de datos sobre el bus.

Ahora que ya tenemos una noción sobre como trabaja el puerto USB, vamos a ver un poco el PIC18F2550, su estructura interna, la etapa osciladora, registros internos, etc.

A medida que vallamos avanzando en el estudio de este microcontrolador veremos sus características pero principalmente como comunicarnos con el puerto USB y el control de dispositivos mediante interfaces hechas en VB.net 2008.

PIC18F2550: Características principales

Dentro de la hoja de datos del microcontrolador encontraremos las siguientes características.




Nota n° 1: Soporta solo Full Speed y Low Speed * 1

Nota n° 2: Soporta modos interruptivo, isócrono y bulk transfer.

Nota n° 3: Al tener una arquitectura optimizada para C utilizaremos un compilador de C como puede ser C de CCS para nuestros programas.

* 1: Veamos el siguiente cuadro:

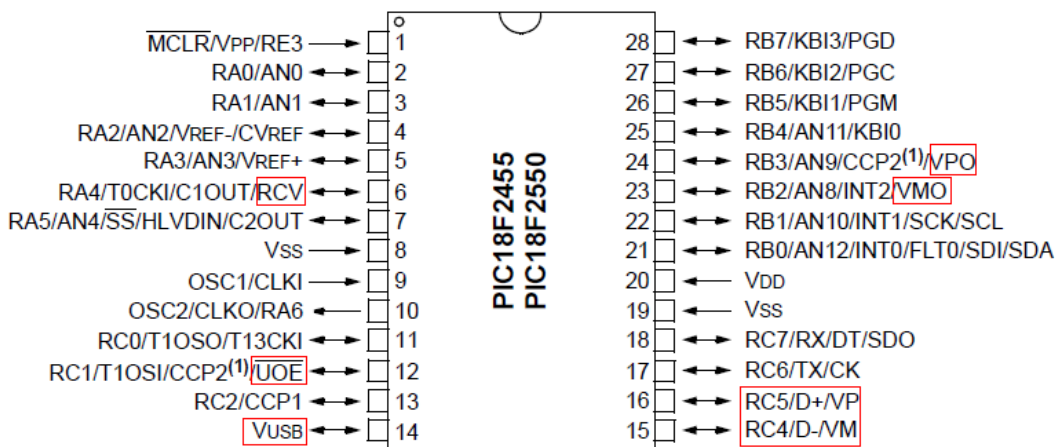


Biggest Myth

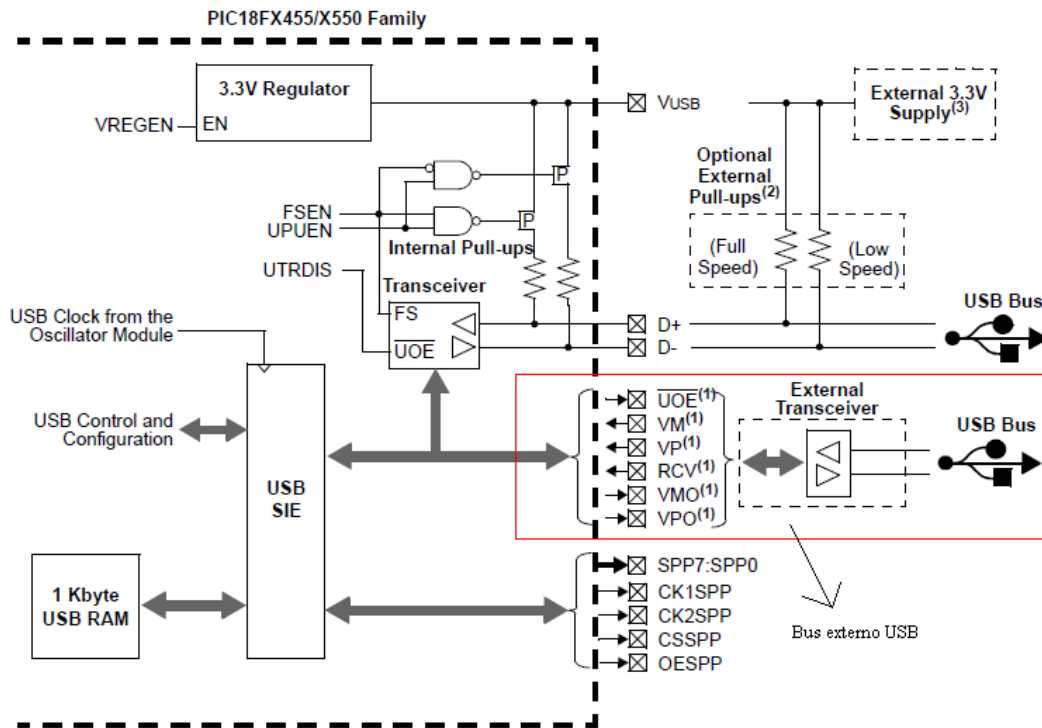
- **MYTH:** A Full-Speed USB peripheral can transfer data up to 1.5 MB/s (12 Mb/s)
- **FACT:** Impossible, 1.5 MB/s is the total bus bandwidth
 - Must be shared among peripherals
 - Protocol overhead
 - Protocol restrictions
 - Realistic raw data throughput to a single peripheral is ~1.0 MB/s
 - Only 64 KB/s in many cases

Según este cuadro vemos que 1.5Mb/seg. Es el total de ancho de banda que posee el bus por lo cual es imposible que el micro se comunique con el Host a esa velocidad ya que hay perdidas derivadas de restricciones del protocolo, otros periféricos conectados al Host, etc. [En la mayoría de los casos nos vamos a poder comunicar a 64KB/seg.](#)

La distribución de pines del PIC18F2550 es la siguiente:



Los pines marcados de rojo son los que le dan al PIC la capacidad de conectarse con un controlador USB externo. El mismo se conectaría de la siguiente manera:



Note 1: This signal is only available if the internal transceiver is disabled (UTRDIS = 1).

2: The internal pull-up resistors should be disabled (UPUEN = 0) if external pull-up resistors are used.

3: Do not enable the internal regulator when using an external 3.3V supply.

Esta función del microcontrolador no la estudiaremos ya que vamos a utilizar el transceptor USB interno del microcontrolador.

El PIC18F4550, también tiene la posibilidad de conectarse al puerto paralelo mediante USB, es decir puede convertir los datos seriales provenientes del USB y convertirlos en un dato de 8 bits de forma paralela utilizando la función Streaming pararell Port. Esta función es utilizada en transmisiones isócronas donde hay que transmitir grandes volúmenes de datos como por ejemplo la comunicación con memorias externas.

Al protocolo USB también lo llaman la pila USB: en las capas superiores tenemos las funciones básicas que el usuario puede realizar (comunicación lógica). esto a su vez va a parar a la segunda capa y luego a la tercera capa (comunicación física) que involucra el aspecto eléctrico. En nuestro caso estaríamos directamente metidos en la capa superior, pero algunas veces entrando en las otras dos:

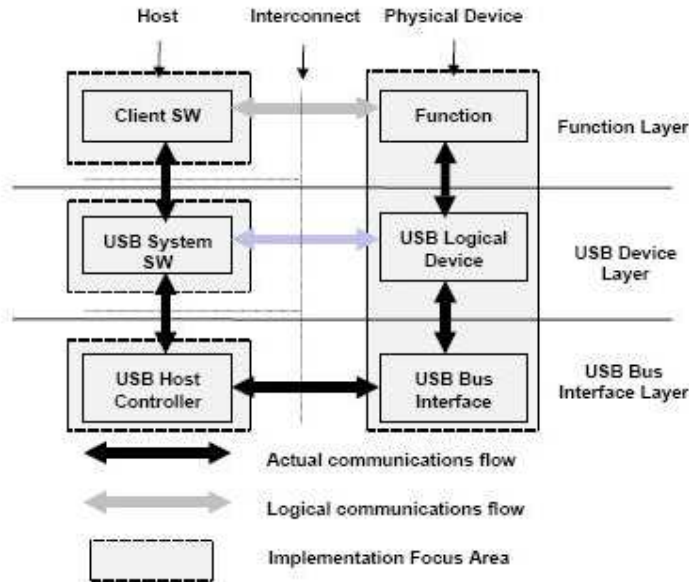


Figure 5-2. USB Implementation Areas

Nosotros trabajaremos programando en C y las capas se distribuirán de la siguiente manera:

primera capa (superior): programación básica en C.

segunda capa (intermedio): llamados a los drivers que trae el compilador de C.

tercera capa (inferior): llamados a los drivers que trae el compilador de C (procesos dentro de los drivers) y conexión del módulo USB al HOST.

Volviendo al tema de los pipes y los Endpoint, los mismos se pueden graficar de la siguiente manera:

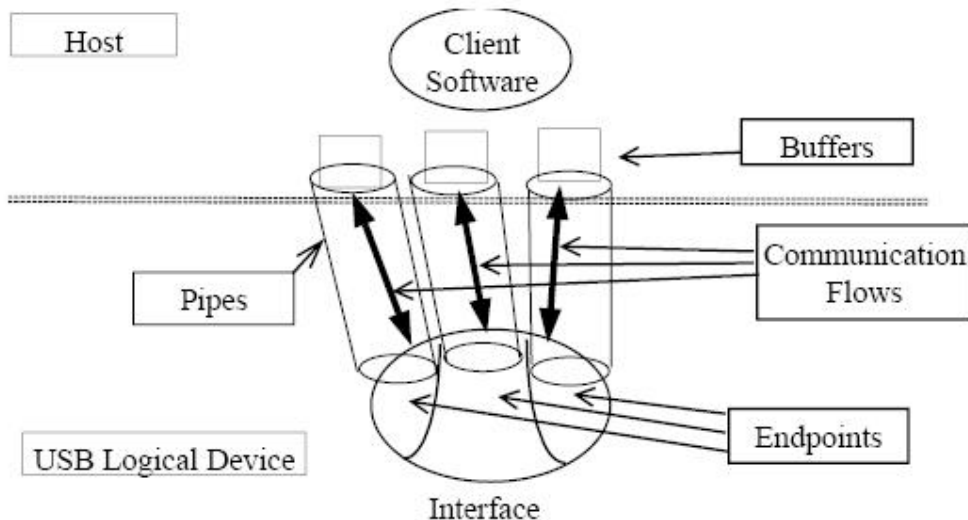


Figure 5-10. USB Communication Flow

Como parte de su protocolo, nos encontraremos entre otras cosas que USB maneja la transmisión de datos por paquetes, llamados TOKEN en la cuál el HOST es el iniciador de todas las transferencias que se producen en el BUS.

8.4.4 Data Packets

A data packet consists of a PID, a data field containing zero or more bytes of data, and a CRC as shown in Figure 8-15. There are four types of data packets, identified by differing PIDs: DATA0, DATA1, DATA2 and MDATA. Two data packet PIDs (DATA0 and DATA1) are defined to support data toggle synchronization (refer to Section 8.6). All four data PIDs are used in data PID sequencing for high bandwidth high-speed isochronous endpoints (refer to Section 5.9). Three data PIDs (MDATA, DATA0, DATA1) are used in split transactions (refer to Sections 11.17-11.21).

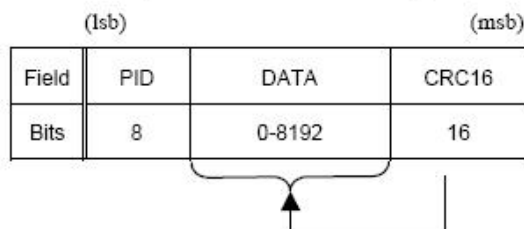


Figure 8-15. Data Packet Format

En la parte de transmisión de datos USB, los paquetes de datos se encuentran en grupos de paquetes de datos, y dentro de estos, existen unos llamados DATA0, DATA1. hay un proceso llamado sincronización del data toggle. A grandes rasgos esto no es mas que un método de validación de paquetes, y lo que hace es enviar alternadamente a DATA0 y DATA1 en una secuencia seguido de su ACK respectivo. todo con el objetivo de mantener la sincronización transmisor <-> receptor.

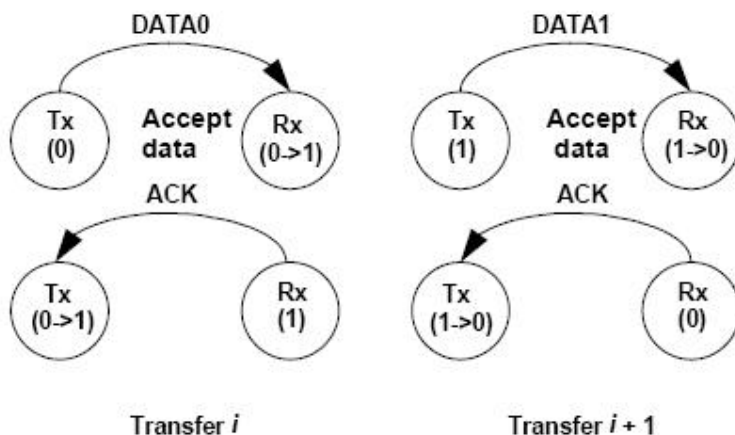
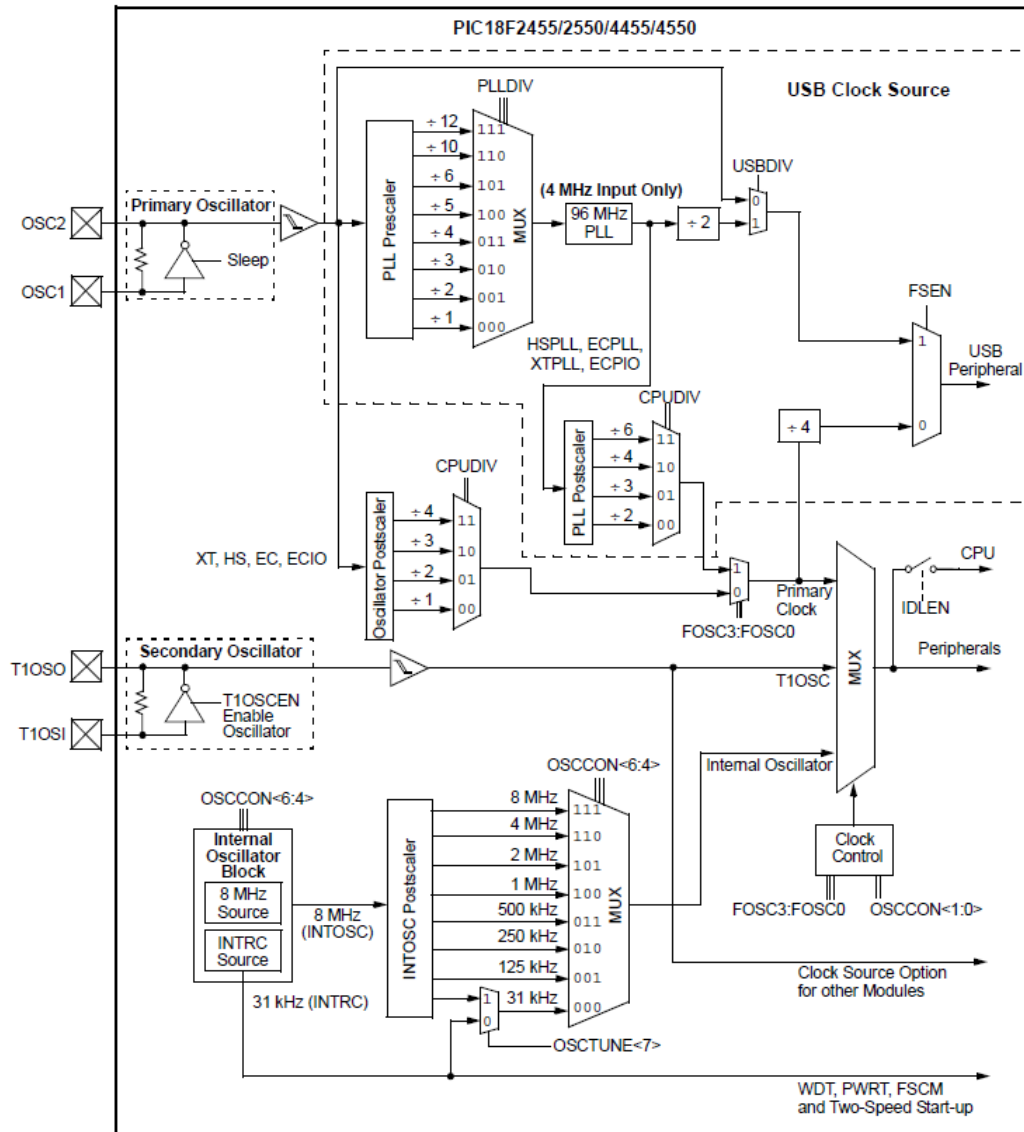


Figure 8-45. Consecutive Transactions

Hasta ahora esta es toda la teoría que necesitamos para comenzar a realizar nuestros ejemplos.

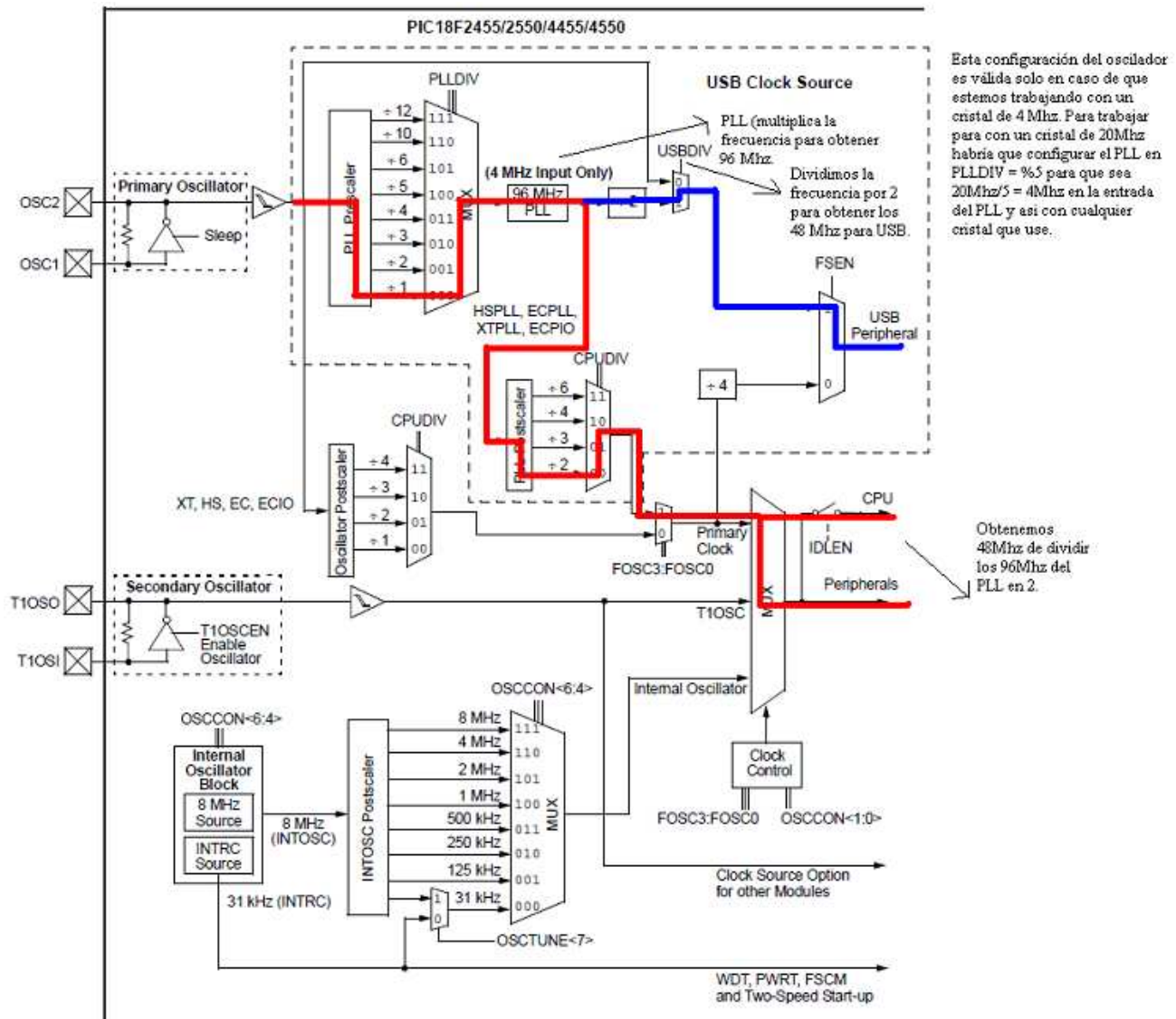
CONFIGURACIÓN DEL MÓDULO OSCILADOR:

El módulo oscilador del PIC18F2550 viene dado de la siguiente manera:



El oscilador tiene varias configuraciones según el cristal usado y que dispositivos utilizarán el oscilador.

Las configuraciones para los diferentes cristales se detalla en la próxima imagen.



Veremos ahora como quedan configurados nuestros fuses para trabajar.

Si trabajamos con un cristal de 4.00Mhz:

```
#fuses MCLR,XTPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL1,CPUDIV1,VREGEN,NOPBADEN
```

Detallaremos cada uno de los fuses:

MCLR: Acá le decimos al compilador que usaremos la función MCLR del Pin 1 del microcontrolador. También podemos poder **NOMCLR** con lo cual dejamos libre el pin RE3 que lo podemos configurar como entrada. Veamos en el cuadro como se configura este bit.

REGISTER 25-5: CONFIG3H: CONFIGURATION REGISTER 3 HIGH (BYTE ADDRESS 300005h)

R/P-1	U-0	U-0	U-0	U-0	R/P-0	R/P-1	R/P-1
MCLRE	—	—	—	—	LPT1OSC	PBADEN	CCP2MX
bit 7							bit 0

Legend:

R = Readable bit

P = Programmable bit

U = Unimplemented bit, read as '0'

-n = Value when device is unprogrammed

u = Unchanged from programmed state

bit 7

MCLRE: MCLR Pin Enable bit

1 = MCLR pin enabled, RE3 input pin disabled

0 = RE3 input pin enabled, MCLR pin disabled

El pin /MCLR se puede activar o desactivar en caso que necesitemos el pin para otras funciones en caso de no utilizarlo podremos usar el pin RE3 como entrada solamente.

bit 6-3

Unimplemented: Read as '0'

bit 2

LPT1OSC: Low-Power Timer1 Oscillator Enable bit

1 = Timer1 configured for low-power operation

0 = Timer1 configured for higher power operation

bit 1

PBADEN: PORTB A/D Enable bit

(Affects ADCON1 Reset state. ADCON1 controls PORTB<4:0> pin configuration.)

1 = PORTB<4:0> pins are configured as analog input channels on Reset

0 = PORTB<4:0> pins are configured as digital I/O on Reset

bit 0

CCP2MX: CCP2 MUX bit

1 = CCP2 input/output is multiplexed with RC1

0 = CCP2 input/output is multiplexed with RB3

El puerto B tiene entradas analógicas que se pueden o no desactivar. Como nosotros usaremos el PORTB como I/O digital tendremos que dejar el bit PBADEN en 0.

Por lo visto anteriormente dejaremos el fuses del puerto A/D del PORTB como **NOPADEN**, por lo que dejaremos configurado el puerto B del microcontrolador como I/O digital.

XTPLL: Con esto le indicamos al compilador que usaremos un cristal en conjunto con el PLL para generar 48Mhz. Si estamos usando un cristal de 4Mhz no ara falta usar división en el postcaler. En caso de que usemos un cristal de 20Mhz tendríamos que poner **HSPLL** y un divisor de postcaler de 5 para obtener los 4Mhz en la entrada del PLL. Veamos en el cuadro que opciones tenemos.

REGISTER 25-2: CONFIG1H: CONFIGURATION REGISTER 1 HIGH (BYTE ADDRESS 300001h)

R/P-0	R/P-0	U-0	U-0	R/P-0	R/P-1	R/P-0	R/P-1
IESO	FCMEN	—	—	FOSC3 ⁽¹⁾	FOSC2 ⁽¹⁾	FOSC1 ⁽¹⁾	FOSC0 ⁽¹⁾
bit 7				bit 0			

Legend:

R = Readable bit

P = Programmable bit

U = Unimplemented bit, read as '0'

-n = Value when device is unprogrammed

u = Unchanged from programmed state

bit 7 **IESO:** Internal/External Oscillator Switchover bit

1 = Oscillator Switchover mode enabled

0 = Oscillator Switchover mode disabled

bit 6 **FCMEN:** Fail-Safe Clock Monitor Enable bit

1 = Fail-Safe Clock Monitor enabled

0 = Fail-Safe Clock Monitor disabled

El compilador lo activa así por defecto.

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **FOSC3:FOSC0:** Oscillator Selection bits⁽¹⁾

111x = HS oscillator, PLL enabled (HSPLL)

110x = HS oscillator (HS)

1011 = Internal oscillator, HS oscillator used by USB (INTHS)

1010 = Internal oscillator, XT used by USB (INTXT)

1001 = Internal oscillator, CLKO function on RA6, EC used by USB (INTCKO)

1000 = Internal oscillator, port function on RA6, EC used by USB (INTIO)

0111 = EC oscillator, PLL enabled, CLKO function on RA6 (ECPLL)

0110 = EC oscillator, PLL enabled, port function on RA6 (ECPIO)

0101 = EC oscillator, CLKO function on RA6 (EC)

0100 = EC oscillator, port function on RA6 (ECIO)

001x = XT oscillator, PLL enabled (XTPLL)

000x = XT oscillator (XT)

Selecciono este debido a que utilizo un cristal de 4Mhz.

Note 1: The microcontroller and USB module both use the selected oscillator as their clock source in XT, HS and EC modes. The USB module uses the indicated XT, HS or EC oscillator as its clock source whenever the microcontroller uses the internal oscillator.

NOWDT: No utilizamos el Whachdog o perro guardián.
Fijémonos en el siguiente cuadro que opciones tenemos:

REGISTER 25-4: CONFIG2H: CONFIGURATION REGISTER 2 HIGH (BYTE ADDRESS 300003h)

U-0	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	WDTEN
bit 7			bit 0				

Legend:

R = Readable bit

P = Programmable bit

U = Unimplemented bit, read as '0'

-n = Value when device is unprogrammed

u = Unchanged from programmed state

bit 7-5

Unimplemented: Read as '0'

bit 4-1

WDTPS3:WDTPS0: Watchdog Timer Postscale Select bits

1111 = 1:32,768

1110 = 1:16,384

1101 = 1:8,192

1100 = 1:4,096

1011 = 1:2,048

1010 = 1:1,024

1001 = 1:512

1000 = 1:256

0111 = 1:128

0110 = 1:64

0101 = 1:32

0100 = 1:16

0011 = 1:8

0010 = 1:4

0001 = 1:2

0000 = 1:1

Opciones del WDT
configurables según los 3
bits: WDTPS3 a WDTPS0

bit 0

WDTEN: Watchdog Timer Enable bit

1 = WDT enabled

0 = WDT disabled (control is placed on the SWDTEN bit)

Habilitamos o
deshabilitamos el WDT
mediante el bit WDTEN

NOPROTECT: Memoria de programa no protegida contra lecturas.**NOLVP:** Modo de programación a bajo voltaje desactivado.**NODEBUG:** No utilizaremos el modo debug.

REGISTER 25-6: CONFIG4L: CONFIGURATION REGISTER 4 LOW (BYTE ADDRESS 300006h)

R/P-1	R/P-0	R/P-0	U-0	U-0	R/P-1	U-0	R/P-1
DEBUG	XINST	ICPRT ⁽¹⁾	—	—	LVP	—	STVREN
bit 7							bit 0

Legend:

R = Readable bit

P = Programmable bit

U = Unimplemented bit, read as '0'

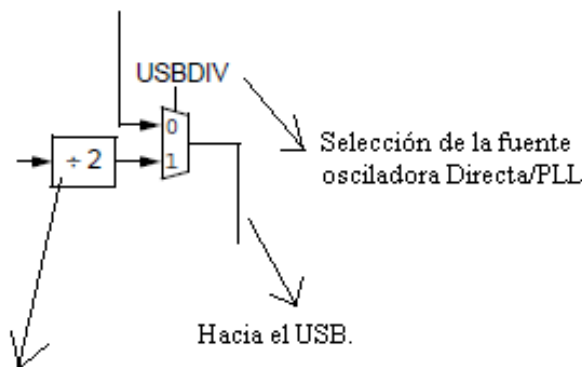
-n = Value when device is unprogrammed

u = Unchanged from programmed state

- bit 7 **DEBUG:** Background Debugger Enable bit
1 = Background debugger disabled, RB6 and RB7 configured as general purpose I/O pins
0 = Background debugger enabled, RB6 and RB7 are dedicated to In-Circuit Debug
- bit 6 **XINST:** Extended Instruction Set Enable bit
1 = Instruction set extension and Indexed Addressing mode enabled
0 = Instruction set extension and Indexed Addressing mode disabled (Legacy mode)
- bit 5 **ICPRT:** Dedicated In-Circuit Debug/Programming Port (ICPORT) Enable bit⁽¹⁾
1 = ICPORT enabled
0 = ICPORT disabled
- bit 4-3 **Unimplemented:** Read as '0'
- bit 2 **LVP:** Single-Supply ICSP™ Enable bit
1 = Single-Supply ICSP enabled
0 = Single-Supply ICSP disabled
- bit 1 **Unimplemented:** Read as '0'
- bit 0 **STVREN:** Stack Full/Underflow Reset Enable bit
1 = Stack full/underflow will cause Reset
0 = Stack full/underflow will not cause Reset

Note 1: Available only on PIC18F4455/4550 devices in 44-pin TQFP packages. Always leave this bit clear in all other devices.

USBDIV: Este bit puede ser 1 o 0 y con el seleccionamos la fuente de oscilación del periférico USB o directa utilizando la frecuencia del cristal seleccionado o del PLL/2.



Dividimos por 2 la frecuencia del PLL que es de 96Mhz para obtener los 48Mhz necesarios para trabajar con USB.

Legend:

R = Readable bit

P = Programmable bit

U = Unimplemented bit, read as '0'

-n = Value when device is unprogrammed

u = Unchanged from programmed state

bit 7-6 **Unimplemented:** Read as '0'

bit 5 **USBDIV:** USB Clock Selection bit (used in Full-Speed USB mode only; UCFG:FSEN = 1)
 1 = USB clock source comes from the 96 MHz PLL divided by 2
 0 = USB clock source comes directly from the primary oscillator block with no postscale

bit 4-3 **CPUDIV1:CPUDIV0:** System Clock Postscaler Selection bits

For XT, HS, EC and ECIO Oscillator modes:

- 11 = Primary oscillator divided by 4 to derive system clock
- 10 = Primary oscillator divided by 3 to derive system clock
- 01 = Primary oscillator divided by 2 to derive system clock
- 00 = Primary oscillator used directly for system clock (no postscaler)

For XTPLL, HSPLL, ECPLL and ECPIO Oscillator modes:

- 11 = 96 MHz PLL divided by 6 to derive system clock
- 10 = 96 MHz PLL divided by 4 to derive system clock
- 01 = 96 MHz PLL divided by 3 to derive system clock
- 00 = 96 MHz PLL divided by 2 to derive system clock

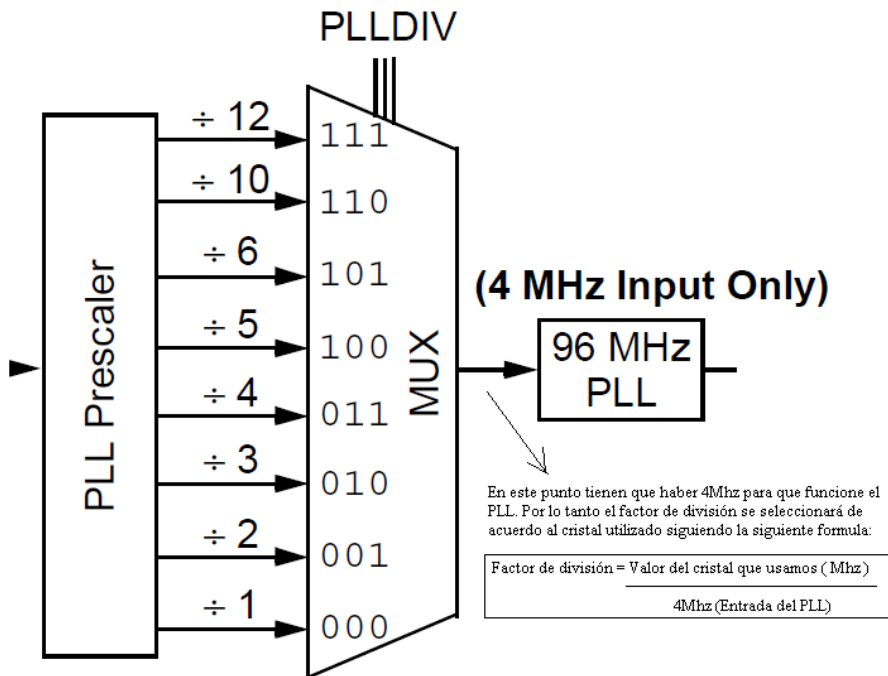
Nosotros utilizaremos el PLL por lo que tendremos que seleccionar 96Mhz del PLL / 2 para obtener 48Mhz para trabajar con USB.

bit 2-0 **PLLDIV2:PLLDIV0:** PLL Prescaler Selection bits

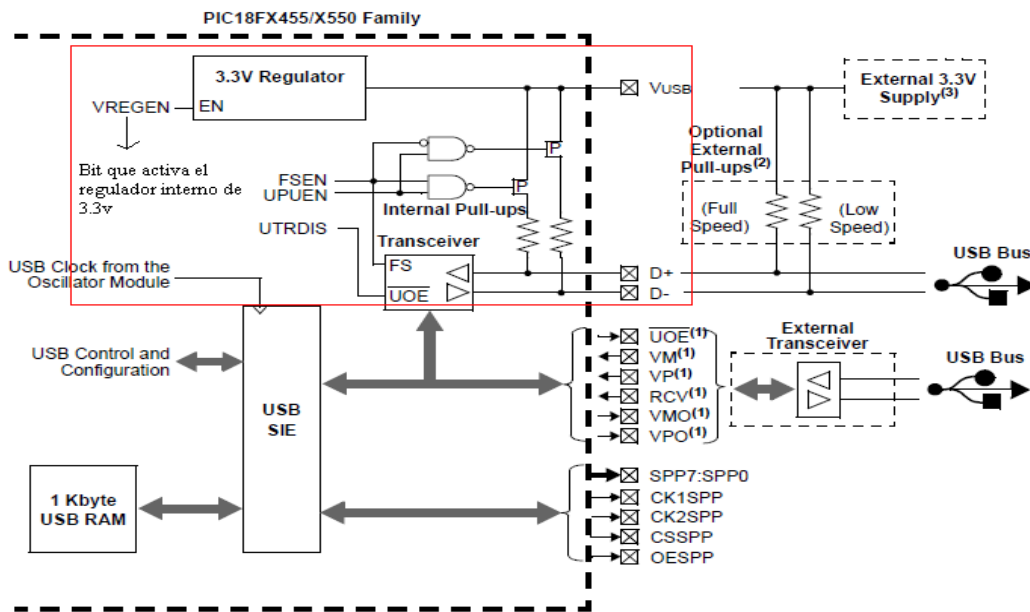
- 111 = Divide by 12 (48 MHz oscillator input)
- 110 = Divide by 10 (40 MHz oscillator input)
- 101 = Divide by 6 (24 MHz oscillator input)
- 100 = Divide by 5 (20 MHz oscillator input)
- 011 = Divide by 4 (16 MHz oscillator input)
- 010 = Divide by 3 (12 MHz oscillator input)
- 001 = Divide by 2 (8 MHz oscillator input)
- 000 = No prescale (4 MHz oscillator input drives PLL directly)

Al utilizar un cristal de 4 Mhz no tendremos que usar preescalamiento.

PLL1: Aquí seleccionamos el factor de división del postcaler, el mismo se seleccionará teniendo en cuenta el valor del cristal que se ha utilizado. Siempre se tiene que tener la premisa que se necesitan 4Mhz en la entrada del PLL para que este genere 96Mhz. Nosotros utilizaremos un cristal de 4Mhz por lo que el factor de división va a ser % 1.



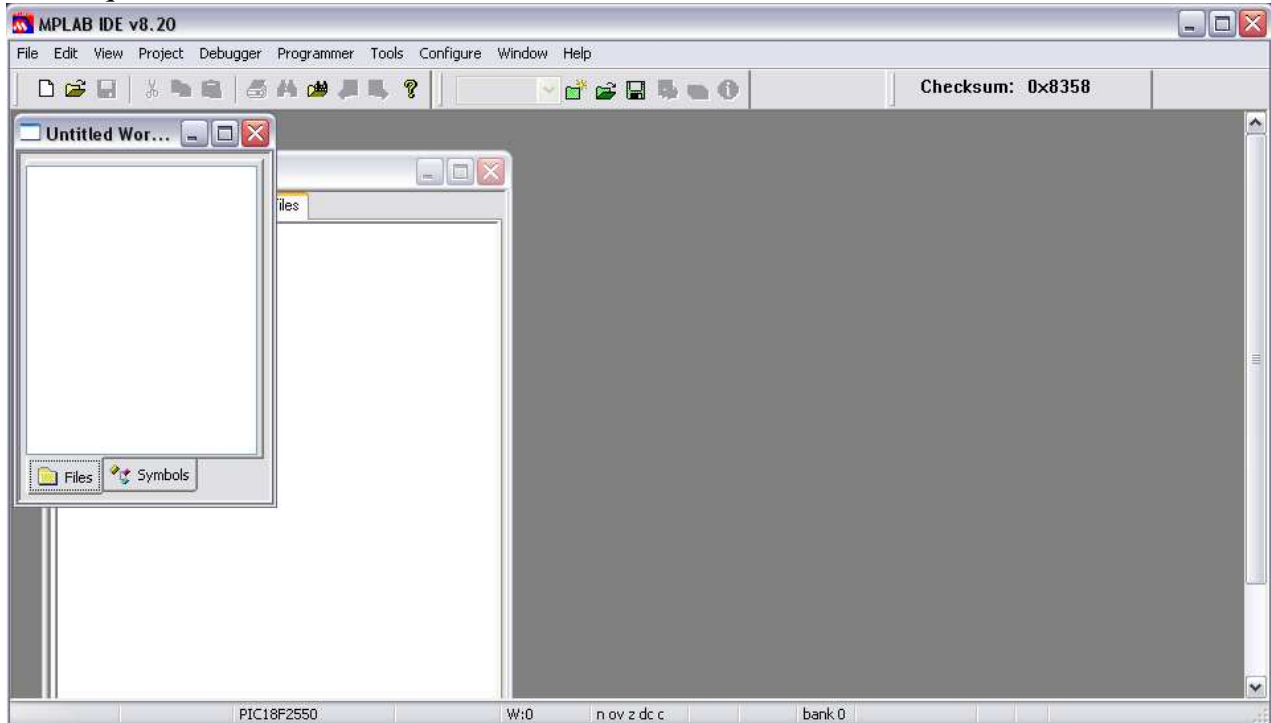
Por último vemos el bit de configuración **VRGEN** que es utilizado para habilitar el regulador interno de 3.3v para el puerto USB, en caso de que estemos usando el USB interno pero en caso de que estemos usando un controlador externo desactivaremos este bit y usaremos una fuente externa. La siguiente inmágen nos muestra esto detalladamente.




- Note 1: This signal is only available if the internal transceiver is disabled (UTRDIS = 1).
- 2: The internal pull-up resistors should be disabled (UPUEN = 0) if external pull-up resistors are used.
- 3: Do not enable the internal regulator when using an external 3.3V supply.

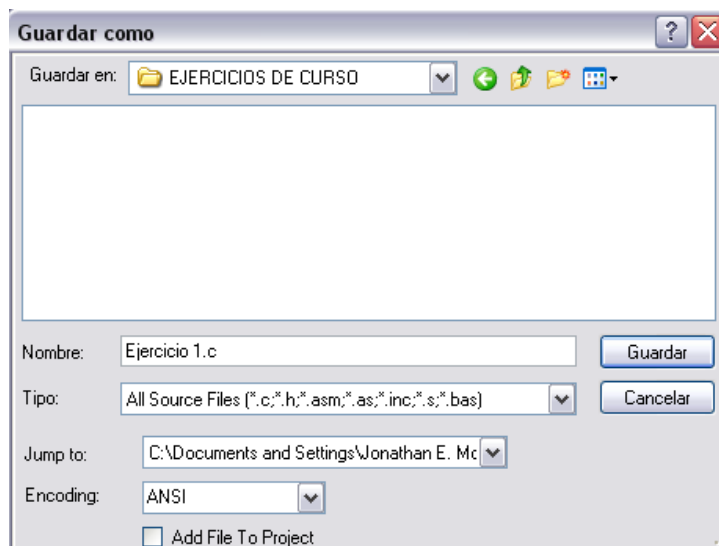
Ahora que ya sabemos configurar los fuses, vamos a nuestra primera aplicación Que es el encendido y apagado de 8 led's utilizando la PLACA de desarrollo PIC28USB.

Antes que nada abrimos una ventana en el MPLAB:

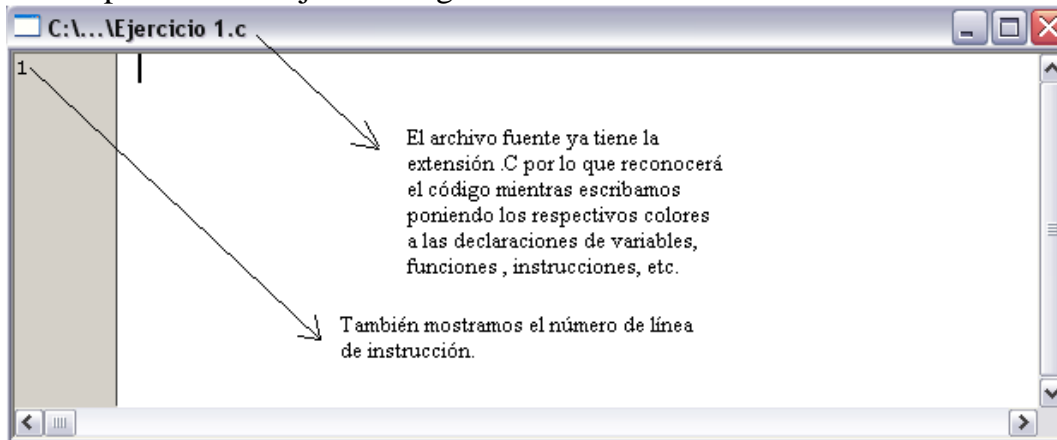


Luego vamos a la pestaña new  y creamos un nuevo archivo de trabajo el cual guardaremos con el nombre de **ejercicio 1.C**, con lo cuál crearemos un archivo fuente de C.

Tendría que quedar algo así:



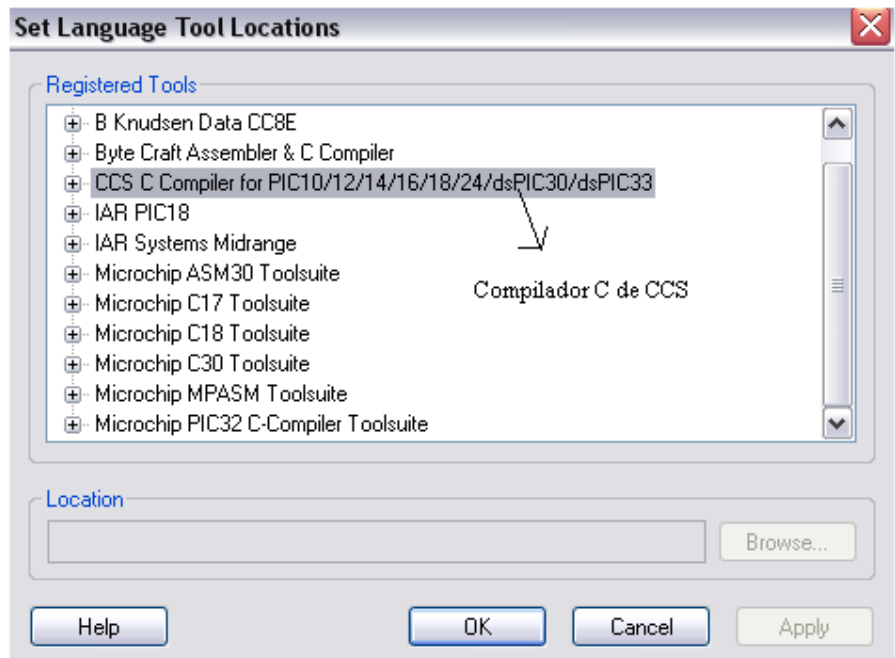
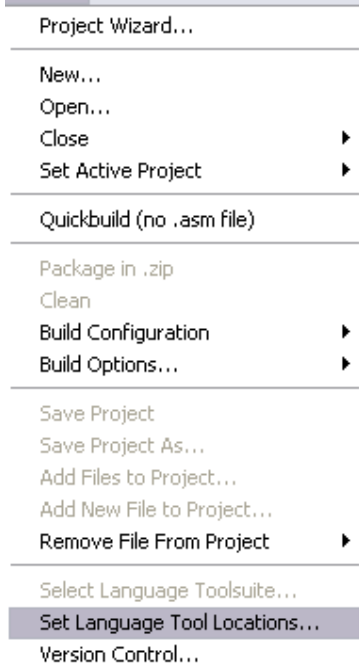
Y el espacio de trabajo de la siguiente de forma:



Antes de ponernos a trabajar tenemos que configurar ciertos parámetros:

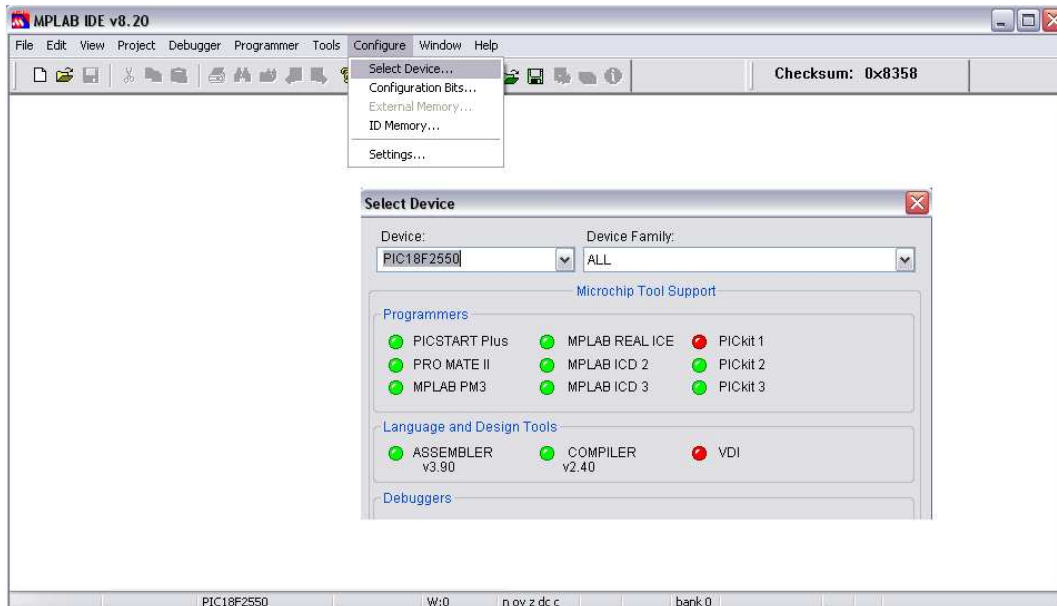
1º - Configurar el compilador de C con el trabajaremos. Como vamos a usar C de CCS, primero tendremos que instalar este software y luego tendremos que instalar el plugin que trae para MPLAB. Luego de esto vamos a:

Project Debugger Programmer To y configuramos lo siguiente:

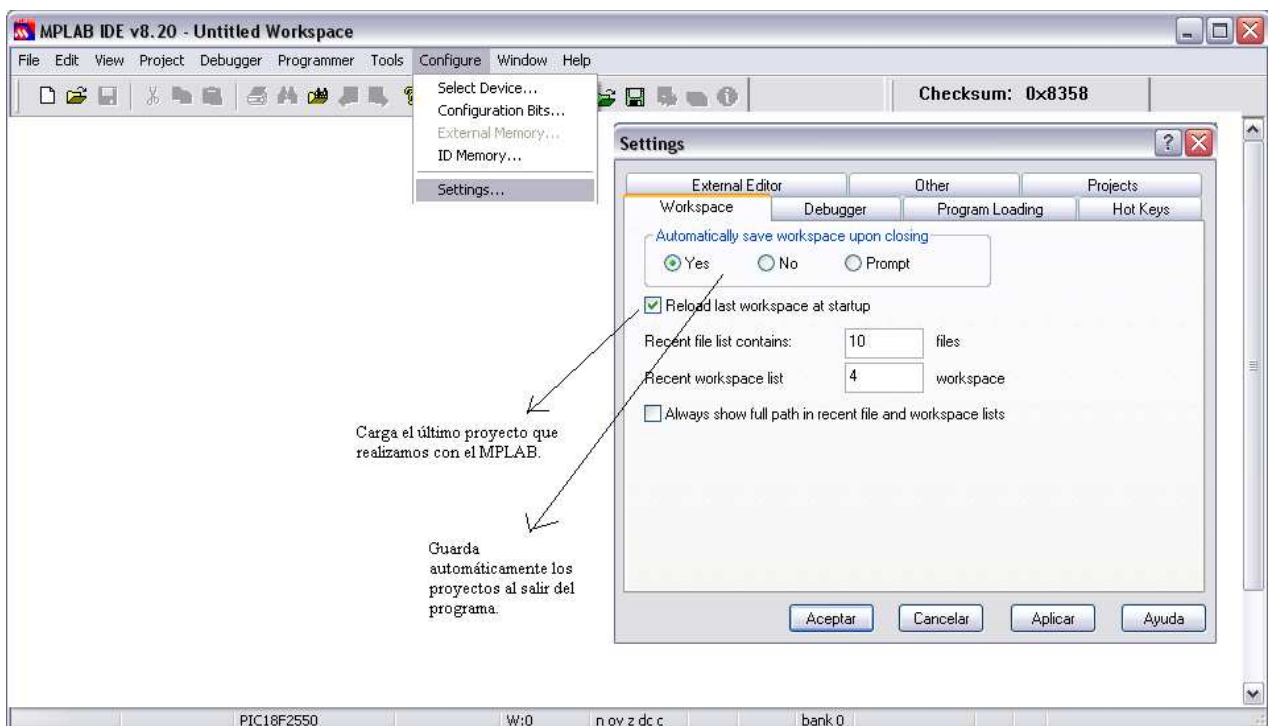


Con esto detecta automáticamente que vamos a trabajar con ese compilador.

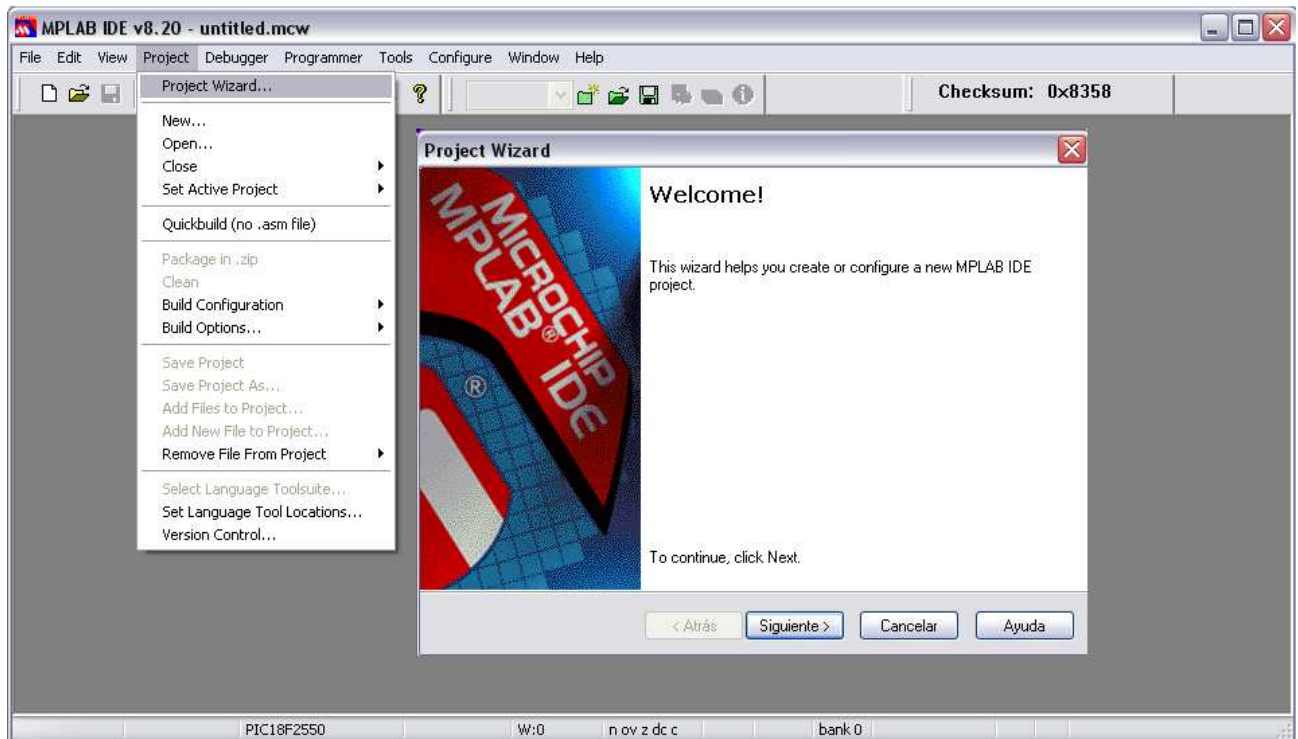
2º - Seleccionamos el dispositivo con el cual trabajaremos en este caso el PIC18F2550.



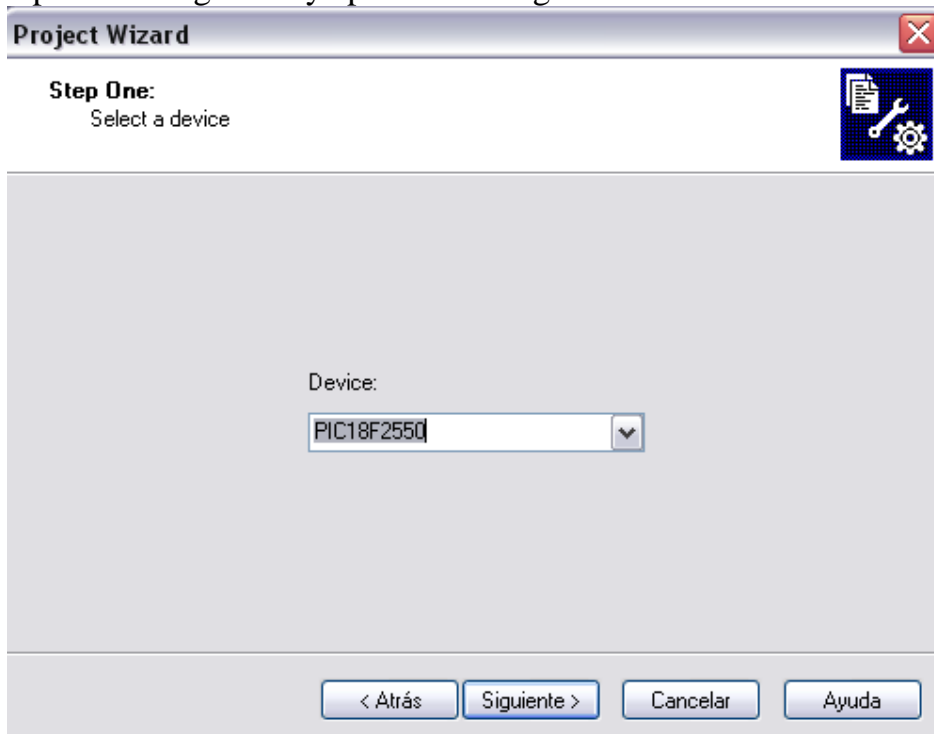
3º - Es útil que cada vez que abramos el MPLAB se cargue el proyecto en el que estábamos trabajando para esto hacemos lo siguiente:



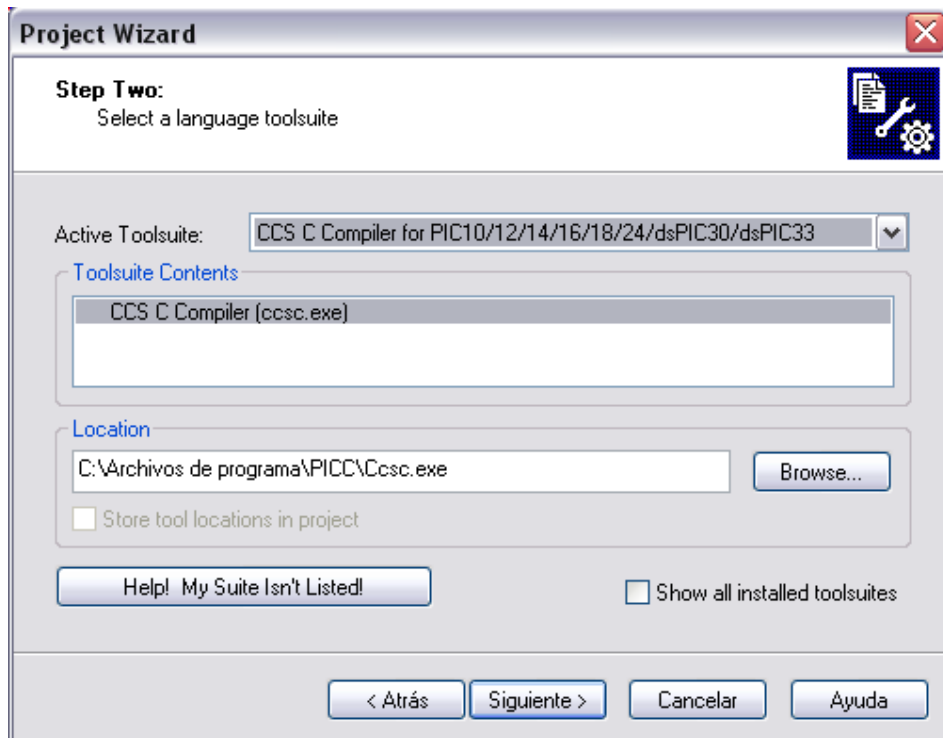
4º - Una vez que tenemos configurado el MPLAB hay que crear nuestro proyecto y para este fin podemos usar el project Wizard del programa.



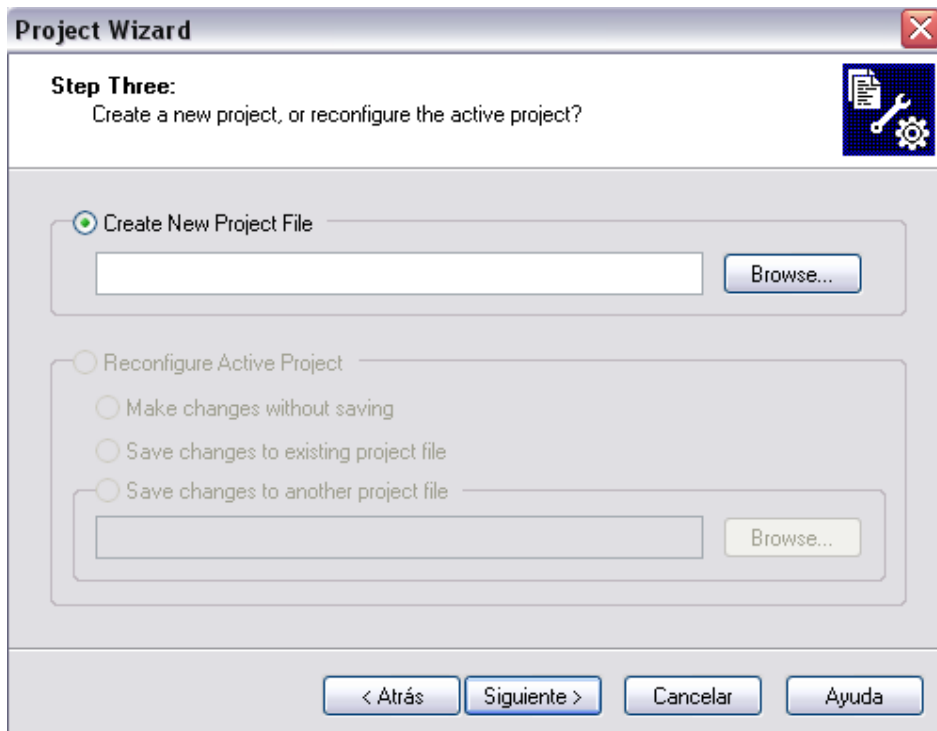
Apretamos siguiente y aparecerá la siguiente ventana:



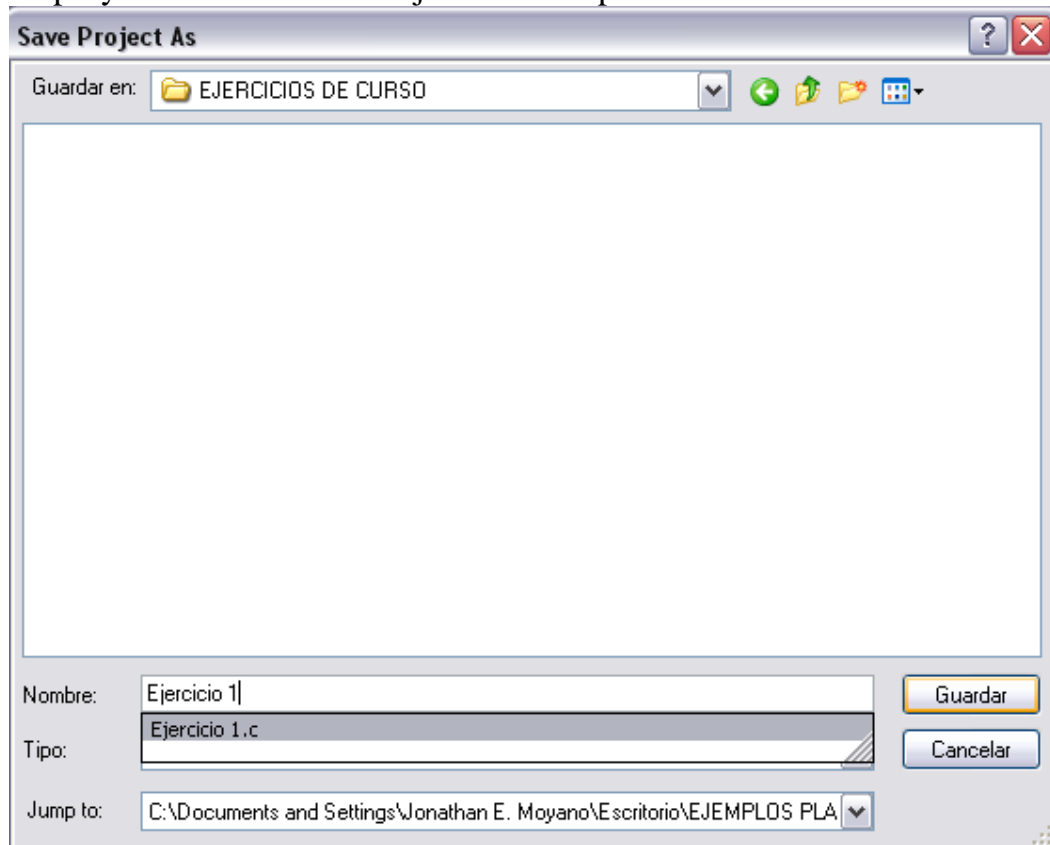
Aquí configuraremos el microcontrolador utilizado en este caso el PIC18F2550. Luego se selecciona el compilador utilizado C de CCS.



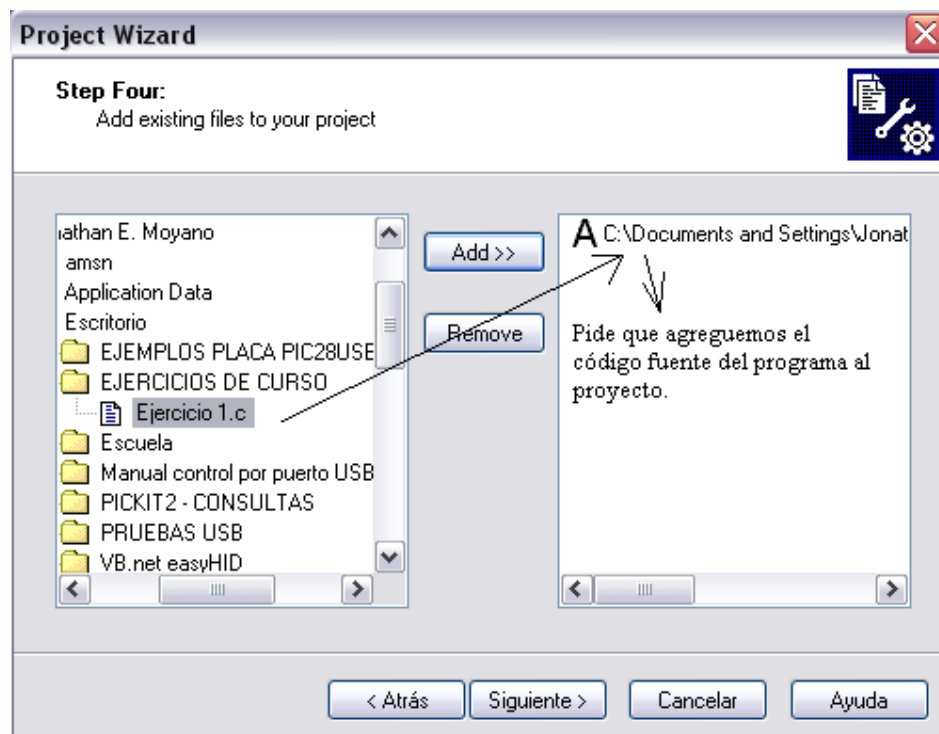
Luego nos va a pedir que creamos un nuevo archivo de proyecto y nos dirá que lo guardemos.



Al proyecto lo llamaremos Ejercicio 1.mcp



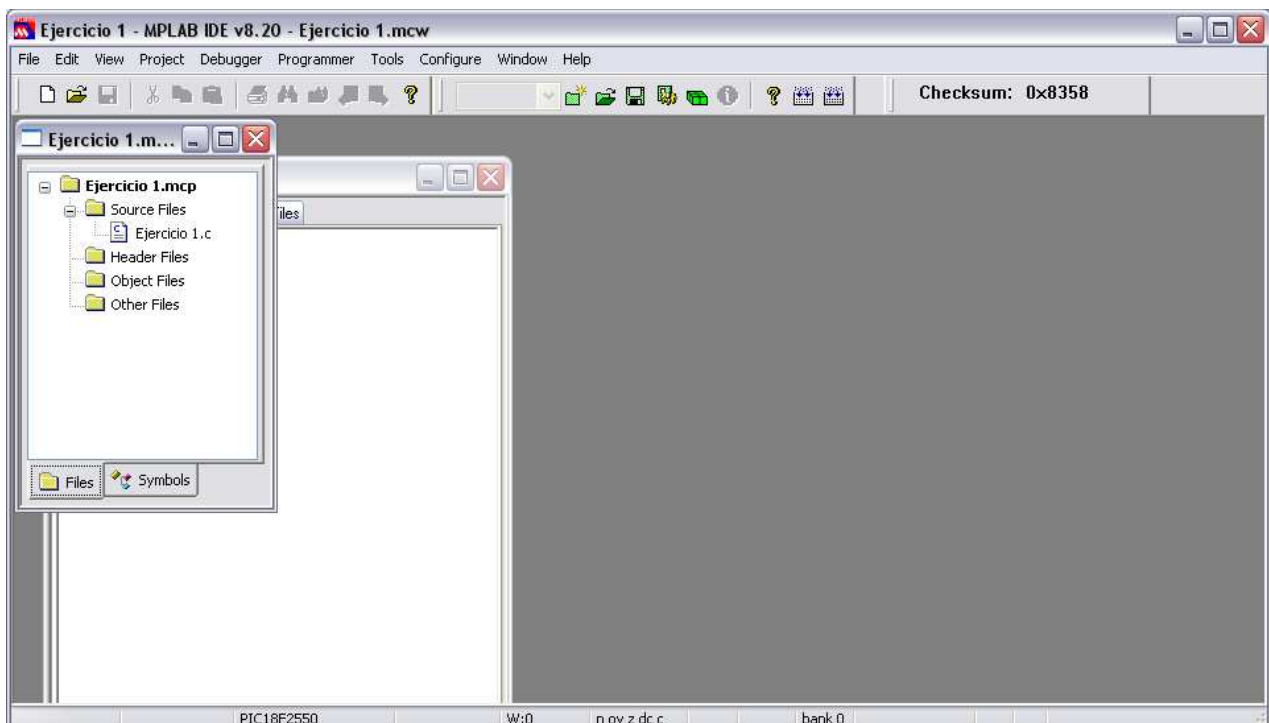
Luego nos va a pedir que enlacemos el código fuente del programa con el proyecto.



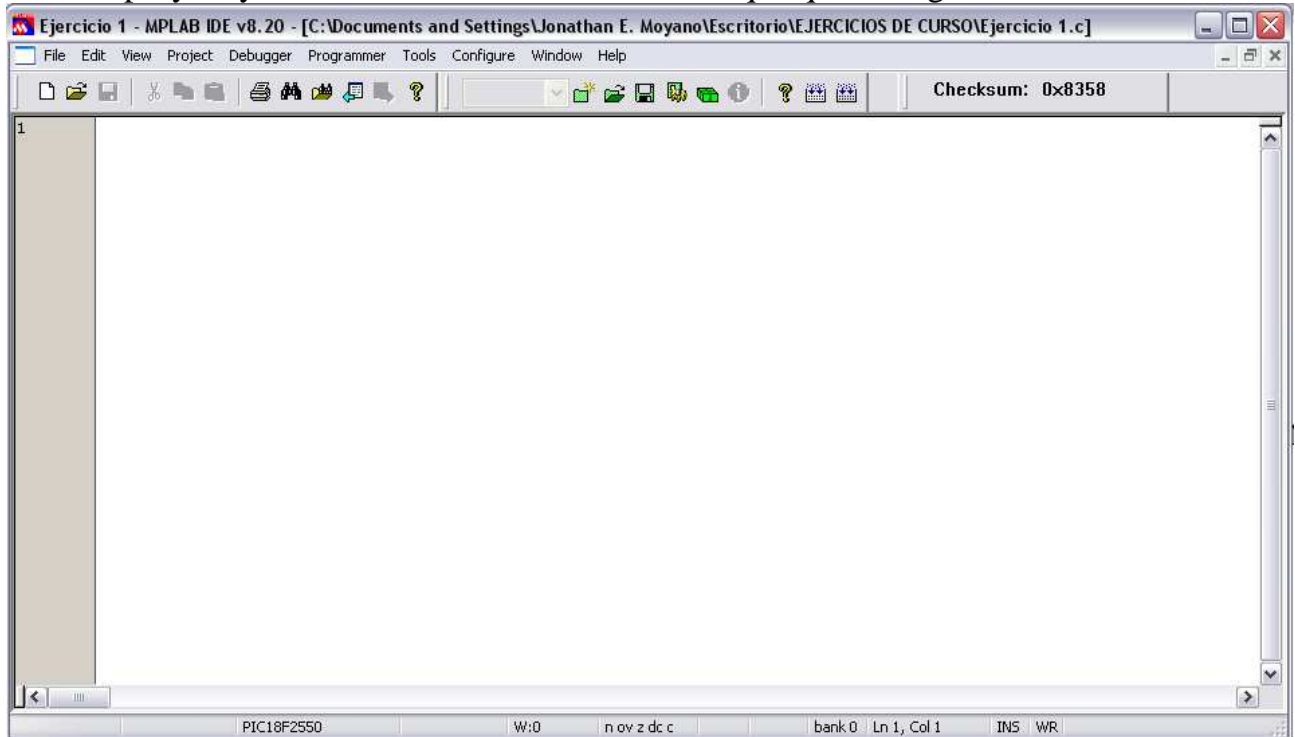
Por último nos muestra una pantalla indicándonos la finalización de la creación del proyecto.



Por último se nos abrirá una pantalla donde aparecerá una ventana como la que se muestra a continuación indicándonos que ya podemos empezar a trabajar.

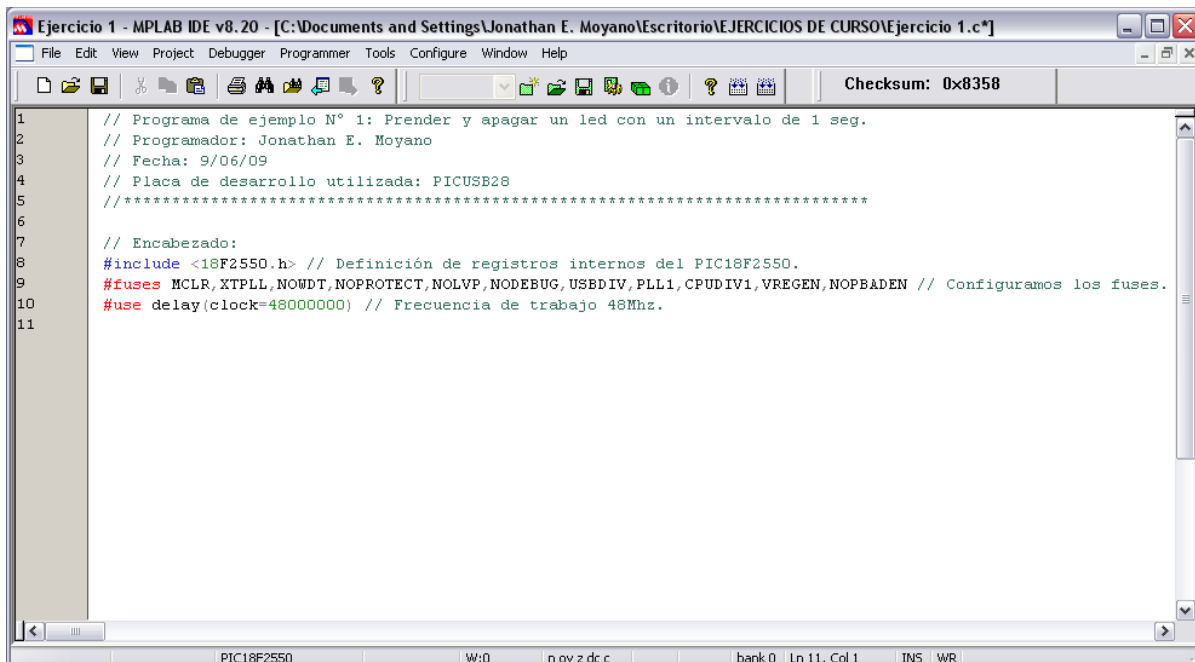


Ahora maximizamos la pantalla de trabajo del MPLAB y comenzamos a escribir nuestro programa para esto seleccionamos el archivo Ejercicio 1.c que está en la ventana project y le damos maximizar. Nos tendría que quedar algo como esto:



Con esto ya estamos en condiciones de empezar a introducir código.

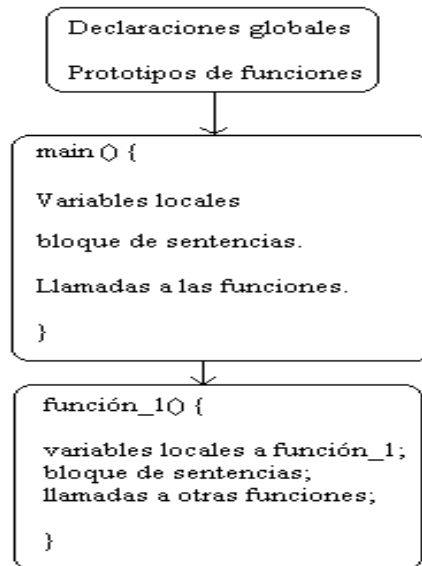
Nota: Todos mis programas se desarrollarán bajo la plataforma de desarrollo PICUSB28 ya que es la única herramienta con la que cuento para poder hacer mis experimentos.



En la página anterior vimos el encabezado del programa donde pusimos los siguientes datos:

- El archivo cabecera con la definición de los registros internos del PIC.
- Los fusibles configurados.
- La frecuencia de trabajo expresada en Hz.

Antes de seguir veremos un poco la estructura de programación en C, para ello veamos el siguiente diagrama:



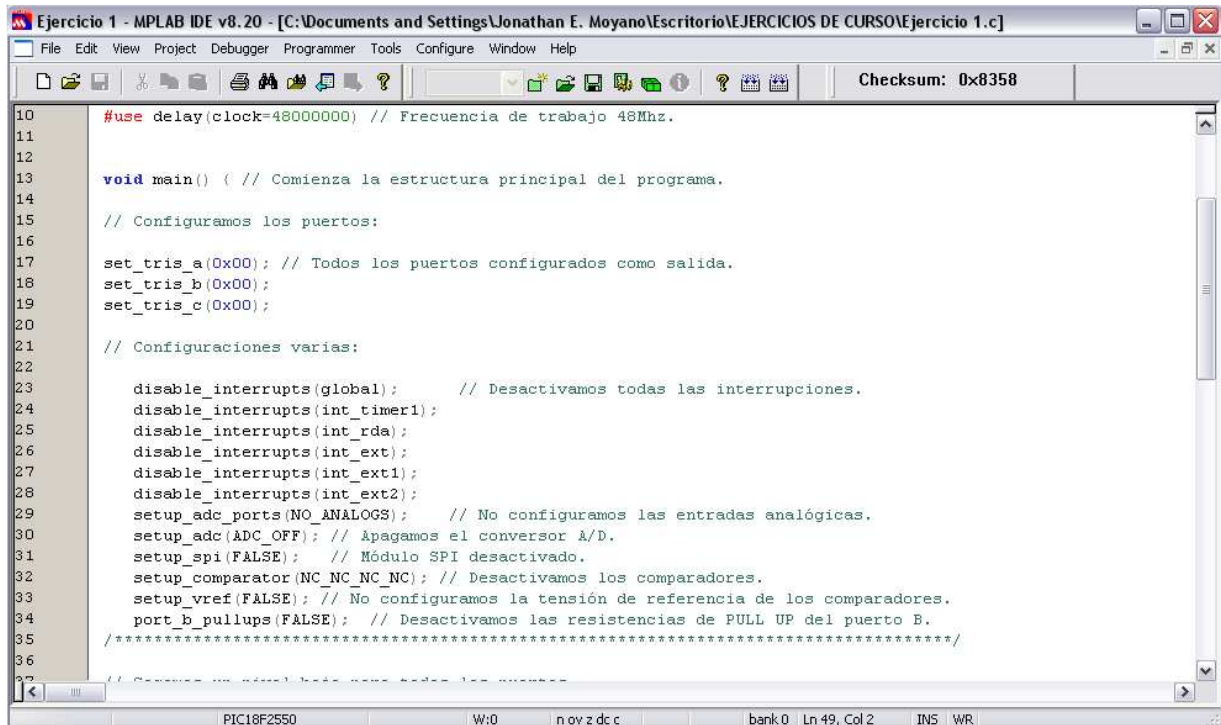
Lo primero que definimos son las variables globales. Estas variables pueden ser usadas en todo el programa y son registros dentro del PIC donde se pueden guardar datos. Luego definimos las funciones, aquí definimos el tipo de función si va a devolver un resultado o no.

Luego en el bloque main o bloque principal del programa definimos las funciones que se utilizarán dentro del main las instrucciones propias del programa principal, configuración de dispositivos internos del microcontrolador, configuración de puertos, etc. Y por último desde el main llamamos a las funciones previamente definidas.

Ya dentro de la función en si tenemos las respectivas variables de la función, el bloque de sentencias que dicta que va a hacer la función, es decir cual va a ser la función en si. Y dentro de la misma función podemos llamar a otras funciones.

Algo a tener muy en cuenta a la hora de programar en C es dividir las tareas en funciones para que el programa quede más modular y entendible y a su vez el mantenimiento sea más fácil.

Dicho lo anterior podemos seguir con nuestro programa de ejemplo Ejercicio 1.c

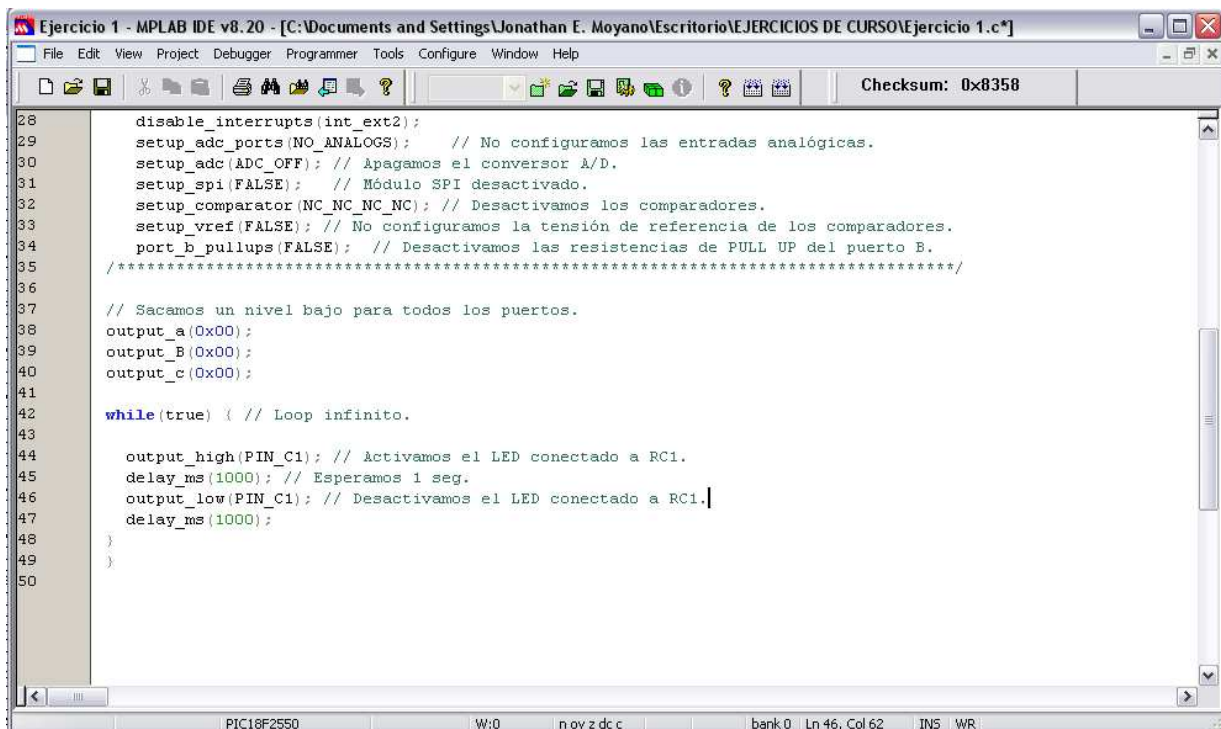


The screenshot shows the MPLAB IDE interface with the file 'Ejercicio 1.c' open. The code defines the main function and performs various hardware configurations for the PIC18F2550. The status bar at the bottom indicates the target device is PIC18F2550, with a checksum of 0x8358.

```

10 #use delay(clock=48000000) // Frecuencia de trabajo 48Mhz.
11
12
13 void main() { // Comienza la estructura principal del programa.
14
15     // Configuramos los puertos:
16
17     set_tris_a(0x00); // Todos los puertos configurados como salida.
18     set_tris_b(0x00);
19     set_tris_c(0x00);
20
21     // Configuraciones varias:
22
23     disable_interrupts(global); // Desactivamos todas las interrupciones.
24     disable_interrupts(int_timer1);
25     disable_interrupts(int_rda);
26     disable_interrupts(int_ext);
27     disable_interrupts(int_ext1);
28     disable_interrupts(int_ext2);
29     setup_adc_ports(NO_ANALOGS); // No configuramos las entradas analógicas.
30     setup_adc(ADC_OFF); // Apagamos el conversor A/D.
31     setup_spi(FALSE); // Módulo SPI desactivado.
32     setup_comparator(NC_NC_NC_NC); // Desactivamos los comparadores.
33     setup_vref(FALSE); // No configuramos la tensión de referencia de los comparadores.
34     port_b_pullups(FALSE); // Desactivamos las resistencias de PULL UP del puerto B.
35     /*****
36     // Sacamos un nivel bajo para todos los puertos.
37
38     output_a(0x00);
39     output_b(0x00);
40     output_c(0x00);
41
42     while(true) { // Loop infinito.
43
44         output_high(PIN_C1); // Activamos el LED conectado a RC1.
45         delay_ms(1000); // Esperamos 1 seg.
46         output_low(PIN_C1); // Desactivamos el LED conectado a RC1.
47         delay_ms(1000);
48     }
49 }
50

```



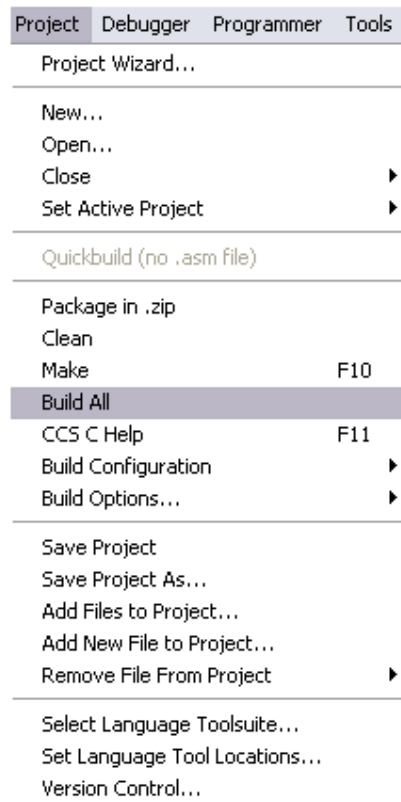
This screenshot continues the code from the previous one, showing the initialization of the output pins and the start of an infinite loop that toggles the state of pin C1 every 1000 milliseconds. The status bar at the bottom shows the cursor is at line 46, column 62.

```

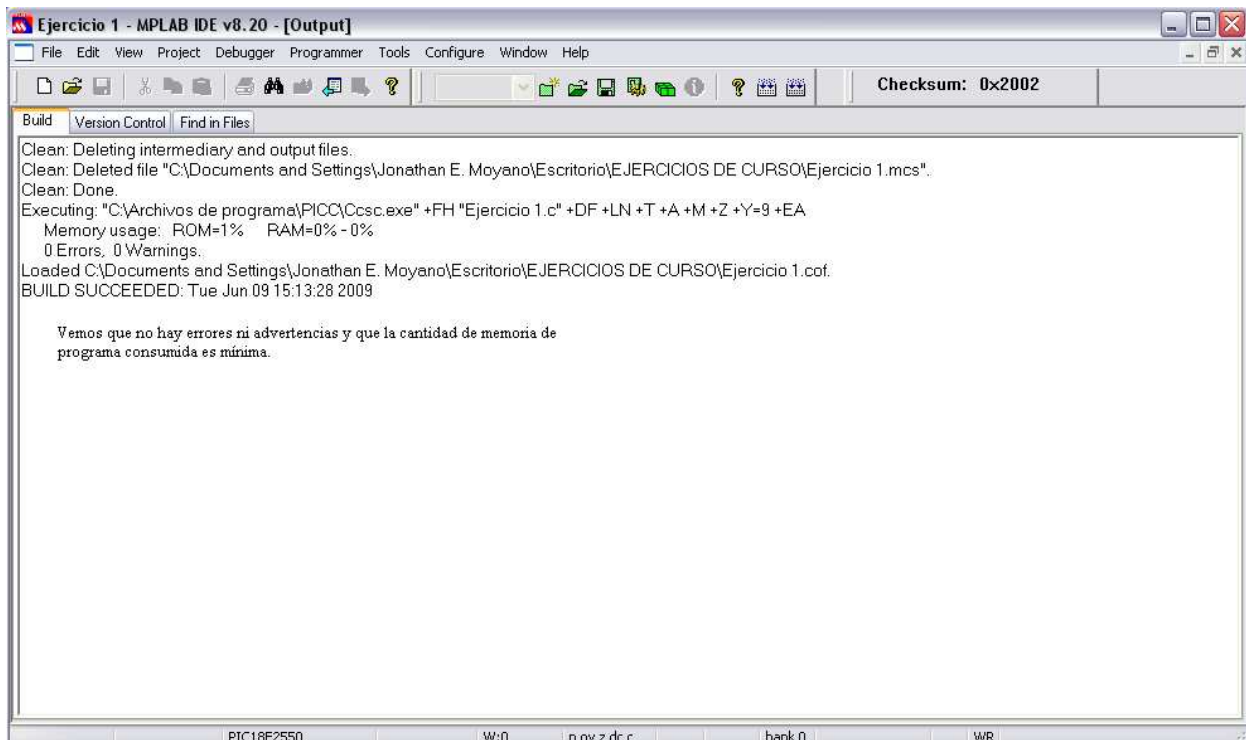
28     disable_interrupts(int_ext2);
29     setup_adc_ports(NO_ANALOGS); // No configuramos las entradas analógicas.
30     setup_adc(ADC_OFF); // Apagamos el conversor A/D.
31     setup_spi(FALSE); // Módulo SPI desactivado.
32     setup_comparator(NC_NC_NC_NC); // Desactivamos los comparadores.
33     setup_vref(FALSE); // No configuramos la tensión de referencia de los comparadores.
34     port_b_pullups(FALSE); // Desactivamos las resistencias de PULL UP del puerto B.
35     /*****
36
37     // Sacamos un nivel bajo para todos los puertos.
38     output_a(0x00);
39     output_b(0x00);
40     output_c(0x00);
41
42     while(true) { // Loop infinito.
43
44         output_high(PIN_C1); // Activamos el LED conectado a RC1.
45         delay_ms(1000); // Esperamos 1 seg.
46         output_low(PIN_C1); // Desactivamos el LED conectado a RC1.
47         delay_ms(1000);
48     }
49 }
50

```

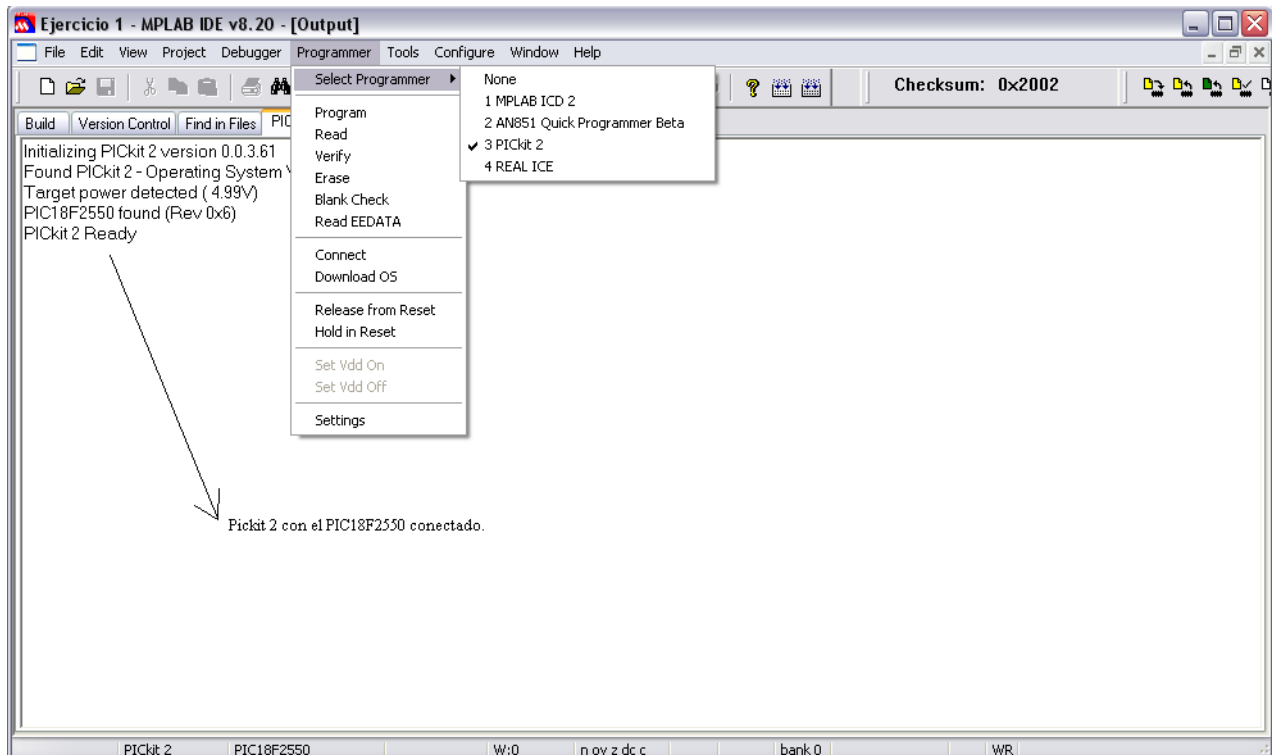
Ahora compilamos el programa para esto hay que hacer lo siguiente:



Luego de que el compilador vea que no hay ningún error nos mostrará la siguiente pantalla:



El próximo paso es programar nuestra aplicación en la memoria del microcontrolador de la placa PICUSB28. Para esto seleccionamos en la pestaña programmer del MPLAB el PICKIT 2 que es el programador USB que yo poseo, en caso de tener otro programador compatible seleccionar de la misma forma.



Como la placa es autoalimentada hay que poner en OFF la tensión de programación.



Con esto el programa comenzará a correr y el LED parpadeará cada 1 segundo.

CLASE CDC: Communication device class *Desarrollo de aplicaciones en VB.net*

Por fin hemos llegado al primer tema importante del manual, aquí veremos como crear un puerto COM virtual a través del PIC18F2550 programando en C de CCS y crearemos aplicaciones en VB.net para comunicarnos con la PC.

En este apartado se desarrollarán los siguientes temas:

- Detección del puerto COM generado.
- Conexión / Desconexión en caliente.
- Envío y recepción de mensajes.
- prender y apagar un LED.
- Control de 8 relés.
- Control de 8 entradas analógicas y 8 relés.
- Lectura de 3 canales analógicos.
- Lectura completa de 8 entradas digitales, 3 analógicas y control de 8 relés.
- Lectura / escritura de una memoria EEPROM a través de una interfaz en VB.net.

Para comenzar con los ejemplos lo primero que hay que instalar es el Visual Basic 2008 Express Edition, en caso de tener Visual Studio 2008 mucho mejor.

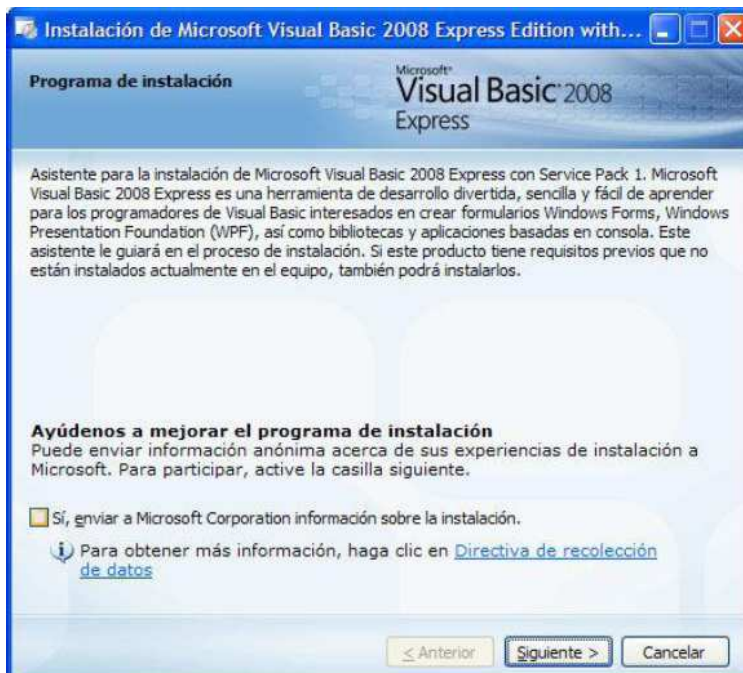
Para poder bajarlo hay que meterse a la página de Microsoft en la parte de descargas.



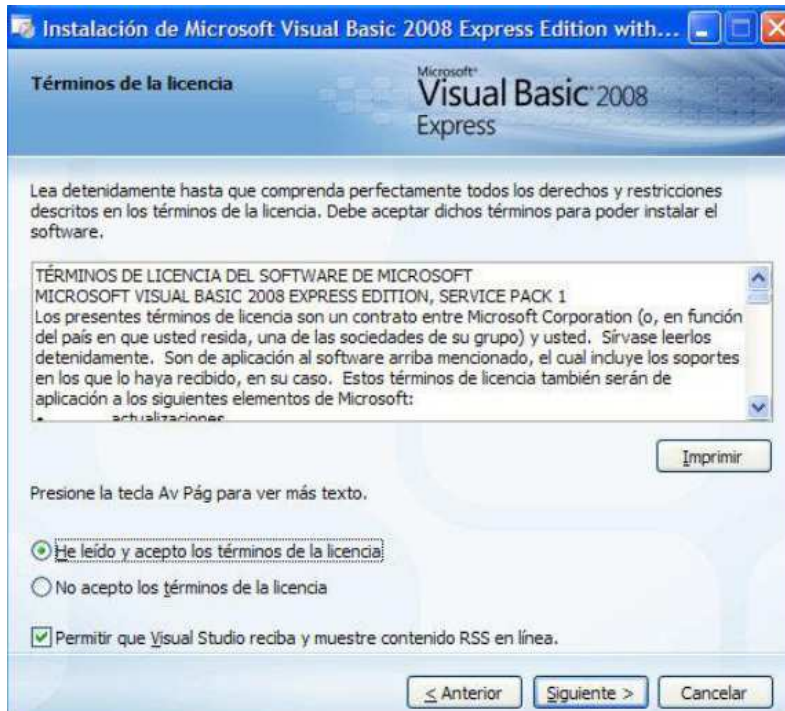
Una vez seleccionado el idioma, presionamos el link "**Download**" y se nos preguntará si queremos descargar el archivo "**vbsetup.exe**" de 2.6MB:



Luego de que lo descarguemos comenzará la descarga del programa que durará en función a la velocidad de conexión que tengamos.



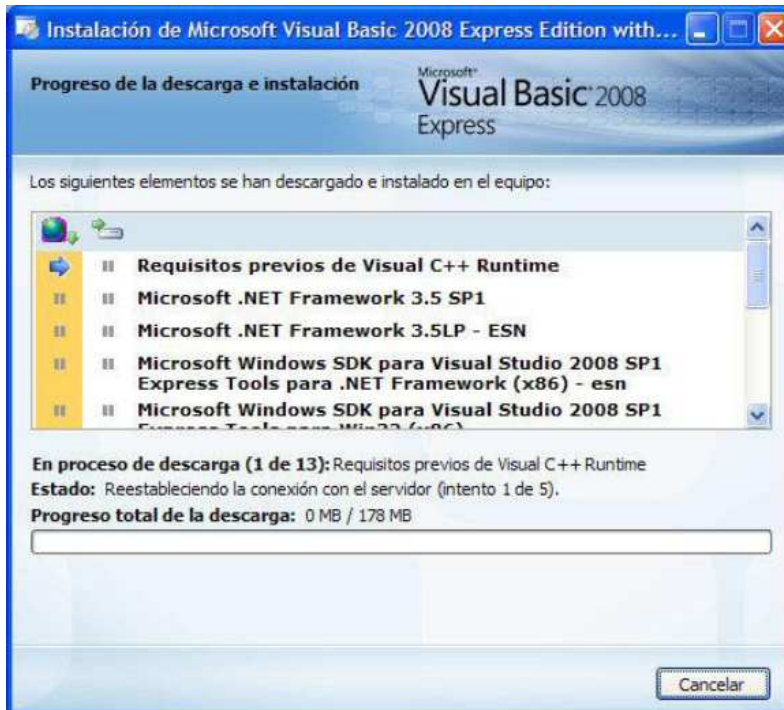
Nos pedirá que aceptemos la licencia de uso:



Luego nos preguntará donde guardaremos los archivos, y le damos a todos la misma ruta por defecto.



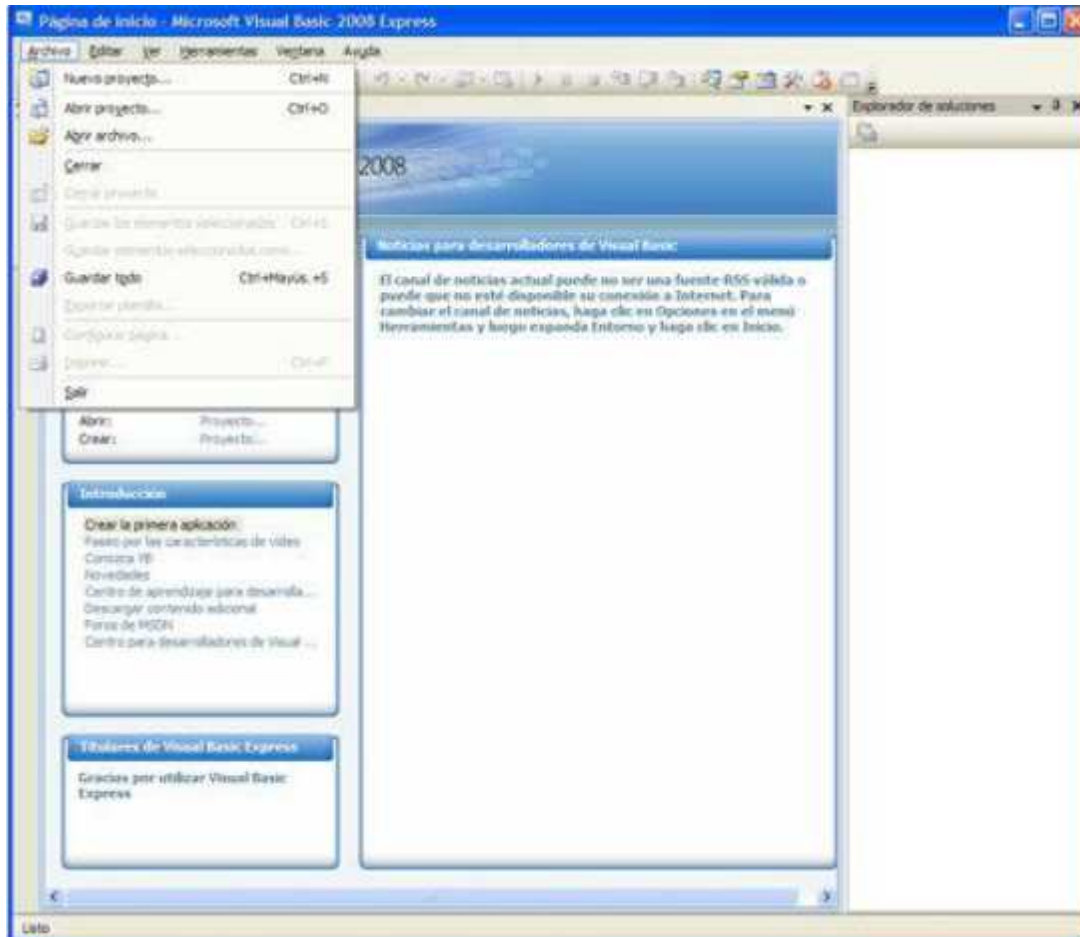
Por último comenzará a descargarse y a instalarse:



Luego de que se haya descargado por completo el programa, pedirá que reiniciemos la máquina. Luego vamos a donde tenemos instalado el programa y lo iniciamos por primera vez, esto llevará un para de minutos ya que se tienen que configurar unos parámetros iniciales antes de comenzar a programar.

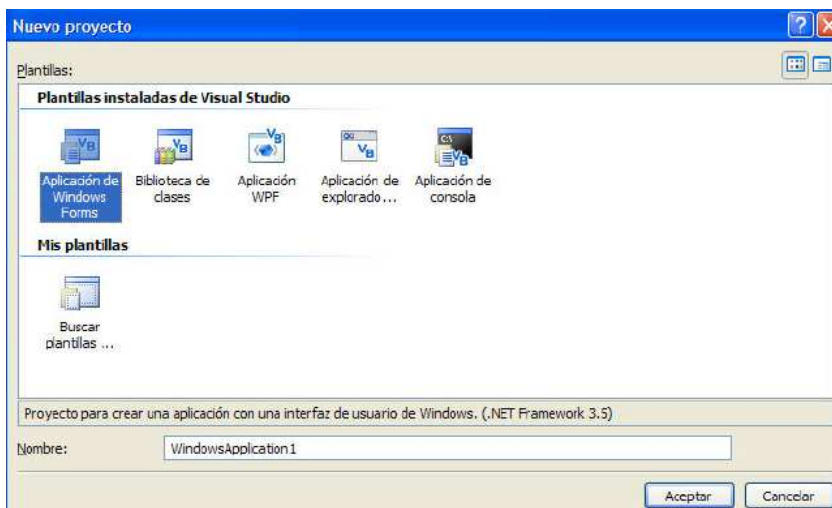


Luego de que se cargan todos los parámetros, el programa se inicia y muestra la siguiente pantalla:



Este es el IDE de desarrollo, las iniciales corresponden a:
Integrated Development Environment, o entorno de desarrollo integrado.

Luego vamos a la pestaña archivo --> nuevo proyecto y nos tiene que aparecer la siguiente pantalla:



Desde aquí podemos elegir que tipo de proyecto vamos a utilizar, en este curso trabajaremos con 2 tipos únicamente *Aplicación de Windows form Y Biblioteca de clases*.

Definamos cada una de ellas para tener una idea más clara de que es cada cosa:

Aplicación de Windows form: Es una aplicación .exe donde nosotros tenemos el marco o estructura al que le llamamos formulario donde nosotros le agregamos los controles tales como botones , cuadros de texto, temporizadores, menús, etc.

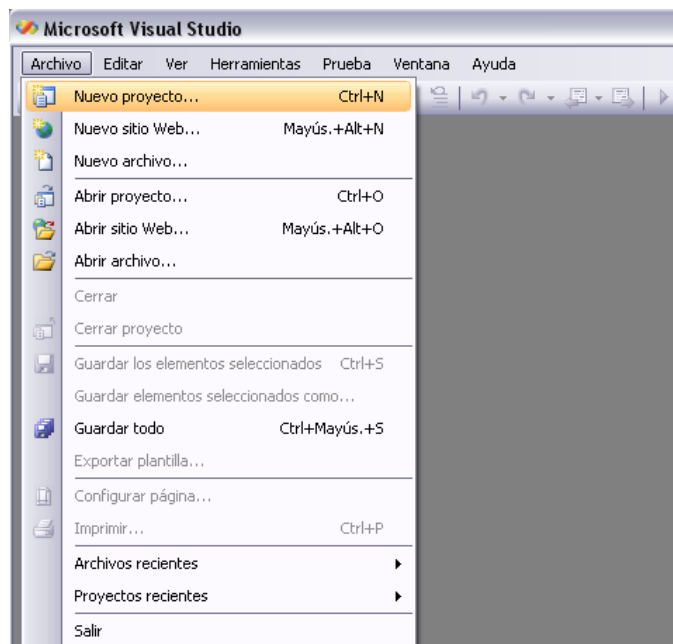
Biblioteca de clases: La biblioteca de clases de .NET Framework está constituida por espacios de nombres. Cada espacio de nombres contiene tipos que se pueden utilizar en el programa: clases, estructuras, enumeraciones, delegados e interfaces.

Cuando se crea un proyecto de Visual Basic o Visual C# en Visual Studio, se sigue haciendo referencia a las DLL más comunes de la clase base (ensamblados). No obstante, si necesita utilizar un tipo incluido en una DLL a la que aún no se hace referencia, deberá agregar la referencia de esa DLL

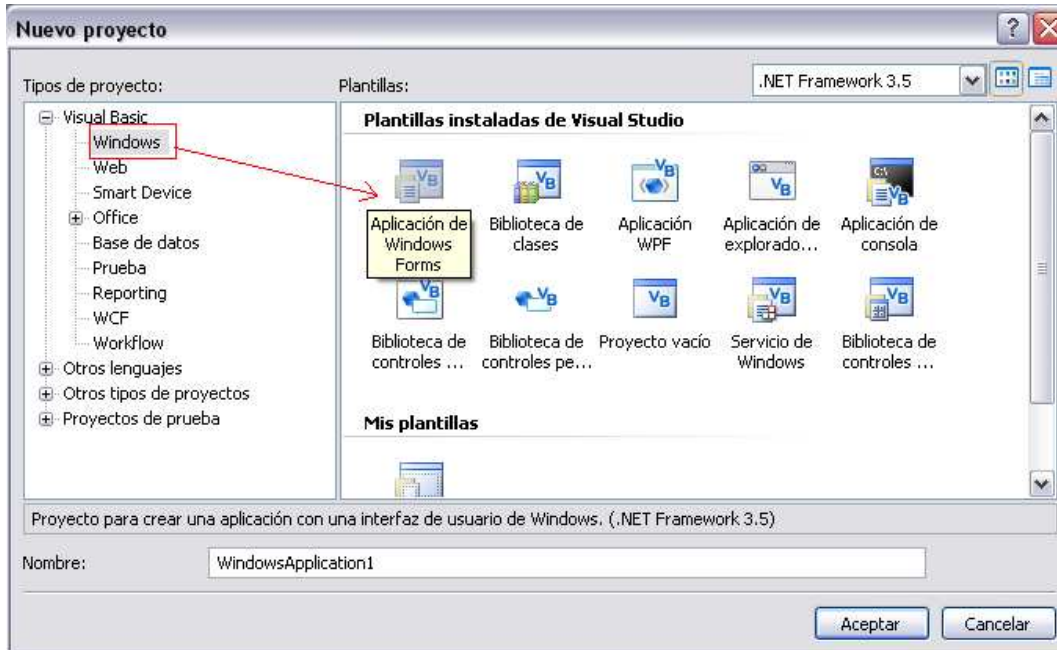
Dentro de nuevo proyecto se le da clic a aplicación de Windows form.

Nota: Todos los proyectos de la clase CDC se programarán en Visual Basic .NET 2008 bajo la suite Visual Studio.

Por lo tanto mi pantalla queda así:



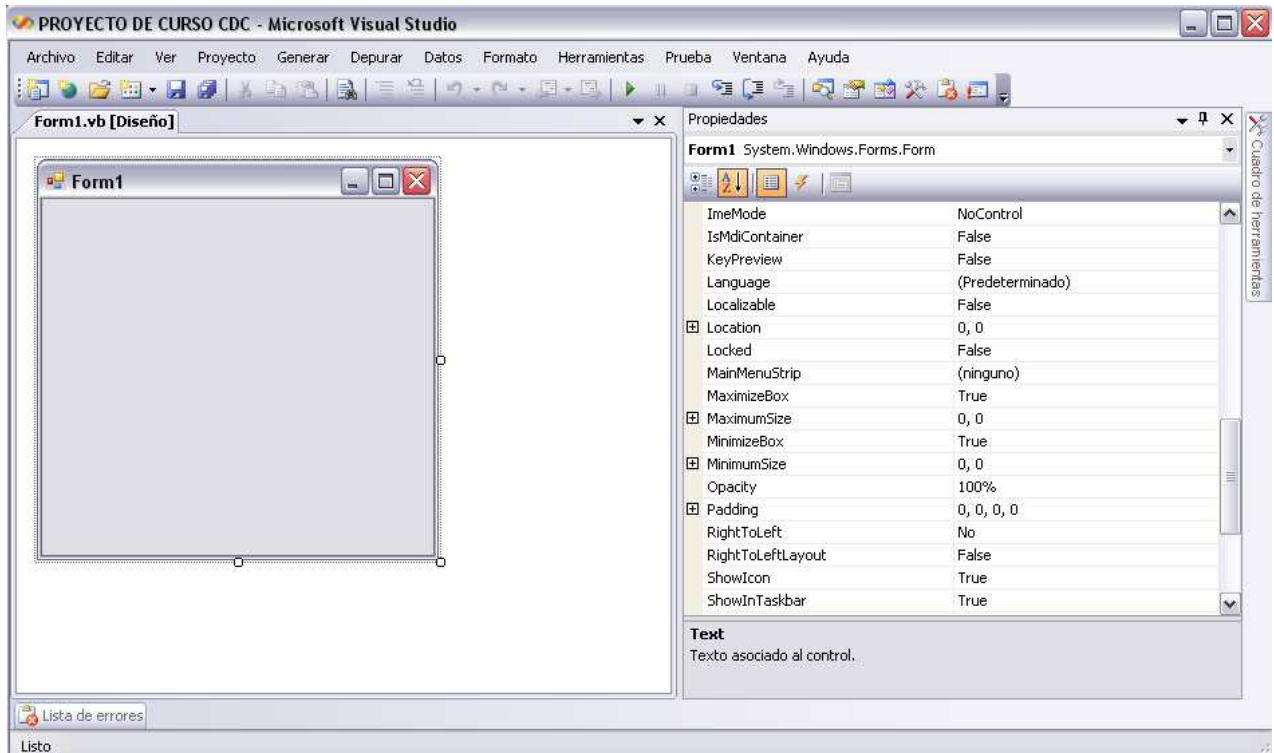
Y seleccionamos:



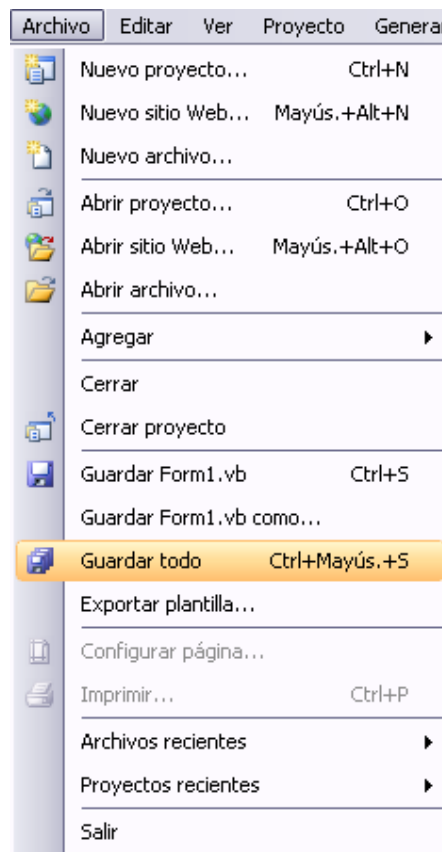
Una vez elegido el tipo de proyecto le damos un nombre, en este caso PROYECTO DE CURSO CDC ya que vamos a escribir un solo ejemplo y se le van a ir añadiendo controles a medida que vallamos aprendiendo a trabajar con ellos y según valla avanzando la complejidad de la programación.



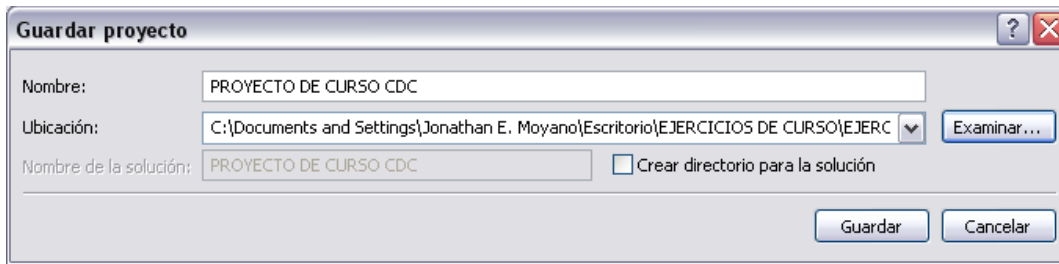
La pantalla para empezar a añadir controles les tendría que quedar como la siguiente:



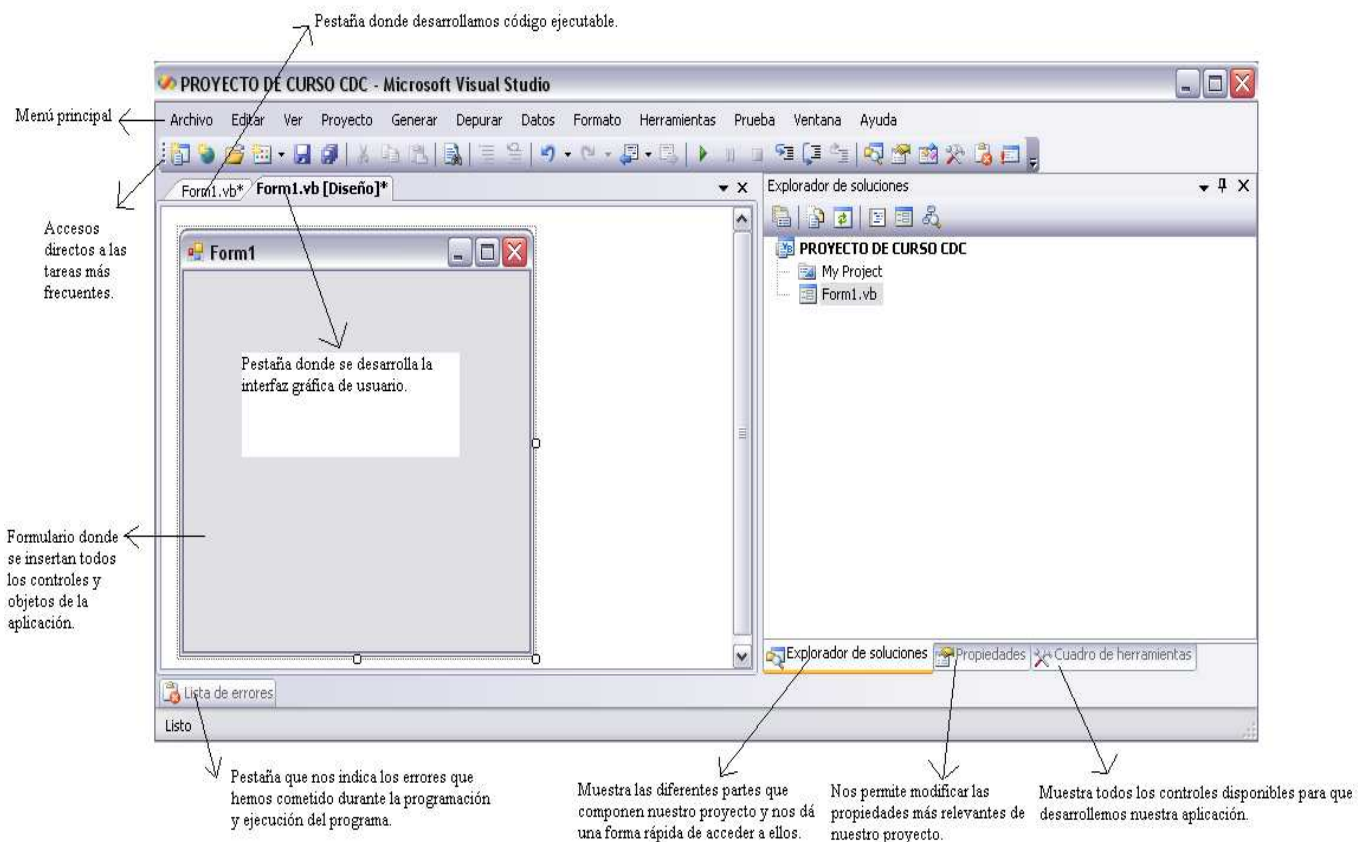
Antes de seguir trabajando es conveniente guardar nuestro proyecto:



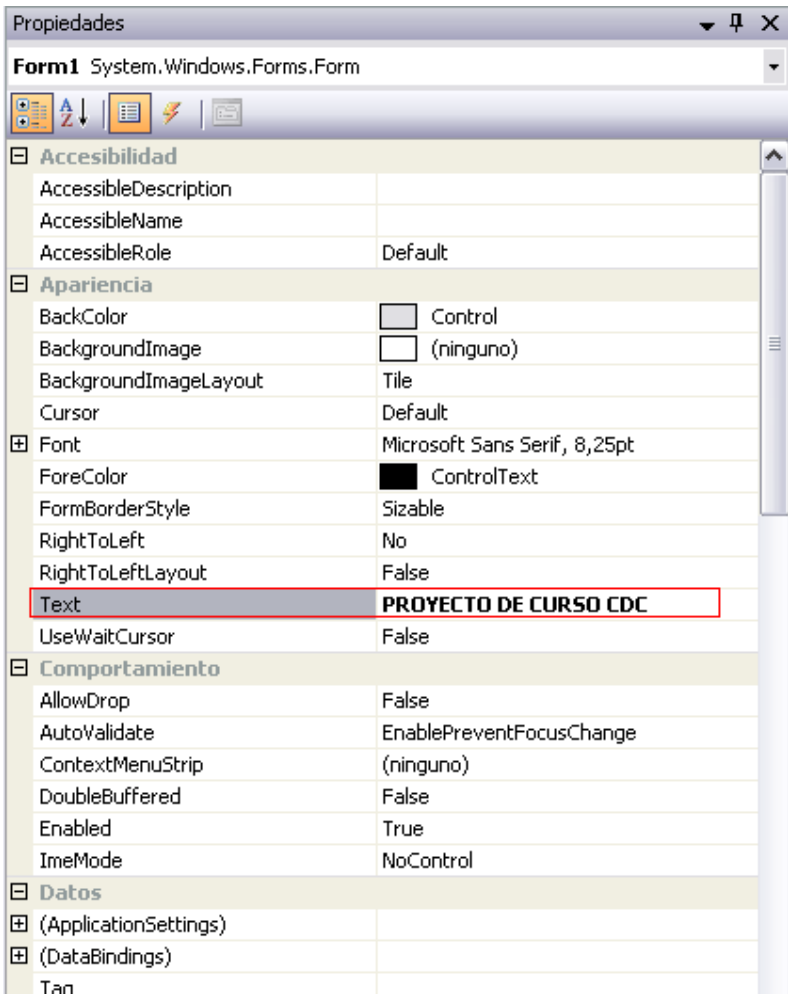
Nos pedirá un nombre para el proyecto que se lo habíamos designado antes y una ruta donde se guardará el proyecto.



Antes de comenzar a trabajar el IDE nos tiene que quedar de la siguiente forma:



Para dar comienzo a nuestro trabajo en VB.net lo primero que tenemos que hacer es modificar las características de nuestro formulario para que quede acorde a lo que nosotros necesitamos, para esto nos vamos a la pestaña propiedades y modificamos los siguientes parámetros:



Y luego cambiamos el name del proyecto, quedando algo así:

(Name)	Form_proyecto
AutoScaleMode	Font
AutoScroll	False
AutoScrollMargin	0, 0
AutoScrollMinSize	0, 0
AutoSize	False

Luego cambiamos la posición inicial del formulario a *center screen* al ejecutarse, esto nos servirá cuando estemos depurando nuestro programa.

StartPosition	CenterScreen
WindowState	Manual
Estilo de ventana	CenterScreen
ControlBox	WindowsDefaultLocation
HelpButton	WindowsDefaultBounds
Icon	CenterParent

Luego de los cambios la ventana del formulario quedaría de la siguiente manera:



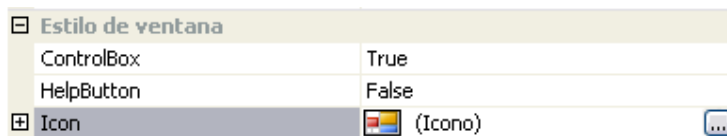
Y en la ventana de programación nos tendría que quedar de la siguiente manera:

```
Public Class Form_proyecto
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    End Sub
End Class
```

Para ejecutar nuestra aplicación presionaremos F5 y el formulario tendrá que aparecer en el centro de la pantalla de la PC.

Otra cosa que podemos modificar en nuestro formulario es el icono del mismo (*Que puede ser o no el mismo que el icono de la aplicación*).

Para modificar el icono de la aplicación se procede de la siguiente manera:

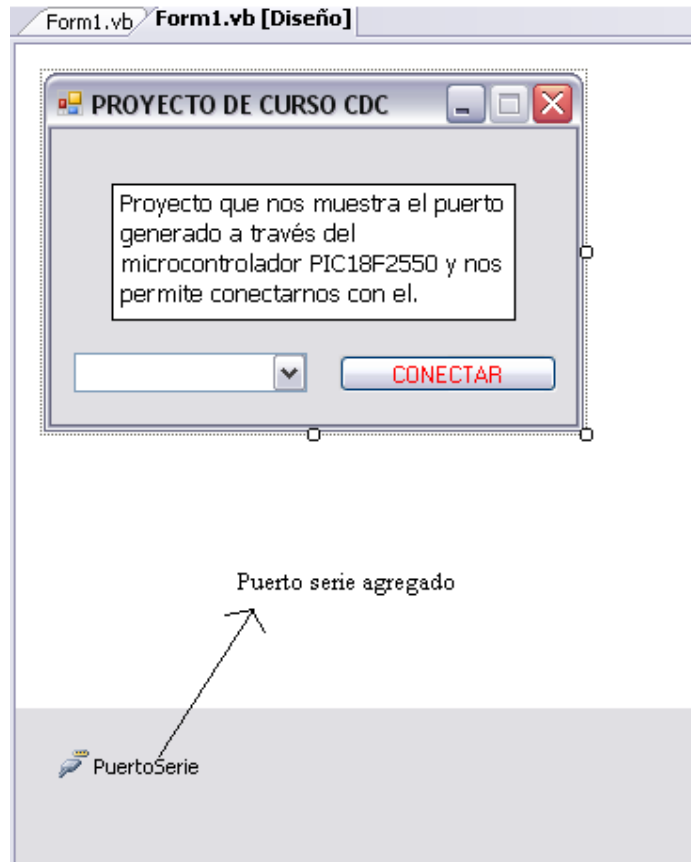


Aquí nos enviará a un explorador donde podremos seleccionar el icono que deseemos.

Bueno hasta aquí llegó la configuración básica de nuestro proyecto, con lo que comenzaremos con nuestro primer ejemplo que se basa en detectar el puerto COM generado por el dispositivo conectado al host.

Comenzaremos a añadir algunos controles que utilizará nuestra aplicación, estos serán un puerto serie y un label para añadir texto a la aplicación y un Combobox para identificar el puerto serie generado por el microcontrolador y por último un botón para conectar la aplicación al dispositivo.

El formulario debería quedarnos de la siguiente manera:




Ahora veremos cuales son las propiedades de los diferentes objetos agregados hasta ahora en el formulario.

Dentro del formulario en la opción *MaximizeBox* pondremos de opción **False** ya que el tamaño del formulario va a variar según la cantidad de controles.



El cuadro de texto donde hemos puesto la descripción del programa tiene las siguientes características.

Apariencia	
BackColor	<input type="checkbox"/> Window
BorderStyle	FixedSingle
Cursor	IBeam
Font	Tahoma, 9,75pt
ForeColor	<input checked="" type="checkbox"/> WindowText
Lines	Matriz String[]
RightToLeft	No
ScrollBars	None
Text	Proyecto que nos muestra el p 
TextAlign	Left
UseWaitCursor	False

Aquí escribimos que es lo que queremos que aparezca en el TextBox

Para poder escribir con múltiples líneas, hay que activar la opción *Multiline* dentro de las propiedades:

Multiline	True
PasswordChar	
ReadOnly	False

Las propiedades del *ComboBox* son las siguientes:

Apariencia	
BackColor	<input type="checkbox"/> Window
Cursor	Default
DropDownStyle	DropDownList
FlatStyle	Standard
Font	Microsoft Sans Serif, 8,25pt
ForeColor	<input checked="" type="checkbox"/> WindowText
RightToLeft	No
Text	
UseWaitCursor	False
Comportamiento	
AllowDrop	False
ContextMenuStrip	(ninguno)
DrawMode	OwnerDrawVariable

Luego las propiedades del botón son las siguientes:

Font	Microsoft Sans Serif, 8,25pt
ForeColor	<input checked="" type="checkbox"/> Red
Image	<input type="checkbox"/> (ninguno)
ImageAlign	MiddleCenter
ImageIndex	<input type="checkbox"/> (ninguno)
ImageKey	<input type="checkbox"/> (ninguno)
ImageList	(ninguno)
RightToLeft	No
Text	CONECTAR
TextAlign	MiddleCenter
TextImageRelation	Overlay

(Name) **CONEXION** Nombre del objeto.

Y por último tenemos las propiedades del puerto serie que vamos a utilizar:

Varios	
BaudRate	9600
DataBits	8
DiscardNull	False
DtrEnable	False
Handshake	None
Parity	None
ParityReplace	63
PortName	COM1
ReadBufferSize	4096
ReadTimeout	-1
ReceivedBytesThreshold	1
RtsEnable	False
StopBits	Two
WriteBufferSize	2048
WriteTimeout	-1

Ya tenemos definidas todas las características de la aplicación y de sus controles ahora nos evocaremos un momento al estudio de la programación del lado del PIC18F2550, explicando como se realizará la comunicación y explicando paso a paso el código fuente y sus funciones.