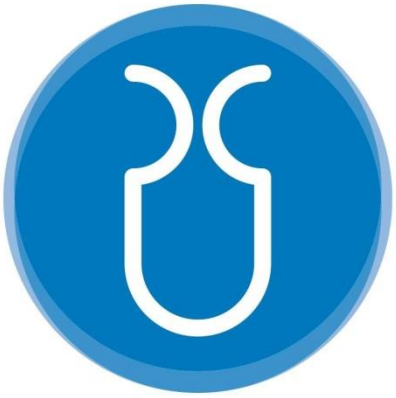

UNIVERSIDAD DEL AZUAY

FACULTAD DE CIENCIA Y TECNOLOGÍA

ESCUELA DE INGENIERÍA ELECTRÓNICA

Diseño e Implementación de un Sistema de Hardware Abierto Basado en Microcontroladores.

Trabajo de graduación previo a la obtención del título de Ingeniero Electrónico.

Autor: Josimar Aguilar

Director: Mst. Gabriel Delgado Oleas

Cuenca, Ecuador

2019

DEDICATORIA

El resultado de mi esfuerzo se lo dedico a Dios por haberme permitido llegar hasta este punto y concederme salud, sabiduría y perseverancia para lograr mis objetivos propuestos.

A mis padres y hermanos por su amor incondicional y por ser mi estímulo para seguir adelante. A mi esposa por apoyarme siempre, por su comprensión, por estar a mi lado respaldándome en el cumplimiento de esta meta; a mis hijos por su amor, por ser mi inspiración y el motivo para vivir.

Gracias a todos ustedes, inmensa gratitud por ser el puente para seguir avanzando hacia el éxito.

Josimar

AGRADECIMIENTO:

Al concluir con el presente proyecto, dejo constancia de mi profunda gratitud a Dios por iluminar mi mente y concederme sabiduría y entendimiento.

A mis padres por el apoyo y cariño que me han brindado, por cultivar e inculcar valores, por enseñarme que la perseverancia y la responsabilidad son el camino para alcanzar las metas propuestas; a mi esposa y mis hijos, que siempre estuvieron brindándome su apoyo, amor y comprensión. Con todo mi cariño les digo gracias de corazón.

A la Universidad del Azuay, por darme la oportunidad de convertirme en un profesional, en especial mi inmensa gratitud al Ing. Gabriel Delgado, Director de Tesis por su generosidad y sus conocimientos compartidos, sin los cuales no hubiese sido posible concluir con éxito este trabajo.

Josimar

RESUMEN

En el presente trabajo se realizó el diseño e implementación de un sistema de hardware abierto basado en microcontroladores, para facilitar la enseñanza de electrónica en otras carreras afines a la tecnología, reduciendo tiempos de desarrollo donde no se requiera conocimientos profundos de electrónica.

El diseño y programación se lo realizó en programa con códigos abiertos, lo cual es una ventaja al no representar costo alguno en su utilización. Igualmente, el equipo resultante es de bajo costo, además de ser compatibles con plataformas como Arduino. El cerebro del equipo es un PIC18F de 8 bits que integra comunicación USB nativa.

Palabras clave: microcontrolador, prototipo, plataforma de desarrollo, PIC18F.

Ing. Daniel Iturralde Piedra. Ph.D
Coordinador de la Escuela de Ingeniería Electrónica

Mst. Gabriel Delgado Oleas
Director de Trabajo de Titulación

Josimar Rogelio Aguilar Ordóñez
Autor

ABSTRACT

In this research, the design and implementation of an open hardware system based on microcontrollers was carried out to facilitate the teaching of electronics in other professions related to technology. The aim of the research is to reduce the development times, in which deep knowledge of electronics is not required. The design and programming was carried out in open source software, which is an advantage as it does not represent any cost in its use. The resulting equipment is low cost and compatible with platforms such as Arduino. The controller of the equipment is an 8-bit PIC18F that integrates native USB communication.

Keywords: microcontroller, prototype, development platform, PIC18F.

Ing. Daniel Iturralde Piedra. Ph.D

Electronic Engineering Faculty Coordinator

Mst. Gabriel Delgado Oleas

Thesis Director

Josimar Rogelio Aguilar Ordóñez

Author

UNIVERSIDAD DE AZUAY
Opto. Idiomas

Translated by
Ing. Paúl Arpi

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

Aguilar Josimar

Trabajo de Graduación

Ing.

Mayo del 2014

Diseño e implementación de un sistema de hardware abierto basado en microcontroladores.

INTRODUCCIÓN

El tiempo de desarrollo de prototipos y programación en áreas relacionadas a la tecnología o electrónica pueden llegar muy grandes, reduciendo los tiempos para la depuración de las aplicaciones finales, impidiendo el mejor desarrollo de los mismos. La evolución en la industria electrónica ha facilitado el desarrollo de prototipos electrónicos, por medio del desarrollo de tarjetas de desarrollo como Arduino, Raspberry, Olimex, etc. Sin embargo, la tarjeta debe ser elegida de acuerdo a las necesidades y requerimientos del proyecto.

Como solución se propone con fines estudiantiles para carreras afines a la electrónica el uso de una tarjeta de desarrollo basada en microcontroladores pic18F con USB nativo, los cuales son fáciles de adquirir en nuestro medio a un costo bajo, requiriendo de pocos componentes adicionales para su funcionamiento, esto basándose en el proyecto pingüino que mantiene una gran compatibilidad con sistemas como Arduino o Raspberry. Todo esto utilizando sistemas open software y open hardware lo cual evita costos adicionales de utilización y además manteniendo soporte, mejoras y actualizaciones en diseños y códigos.

El uso de este tipo de tarjetas facilitará la ejecución de aplicaciones de control electrónico, reduciendo significativamente el tiempo de desarrollo, así como reduce tareas dificultosas para estudiantes que no poseen conocimientos profundos de electrónica, permitiendo mejores desarrollos o facilitando hacer real las ideas para futuros proyectos.

CAPITULO 1

MARCO TEORICO

Introducción.

En esta sección se hace referencia a todo lo relacionado a la teoría del hardware y software utilizado en la elaboración del proyecto, además de conceptos básicos de técnicas utilizadas en el mismo, como son los bootloader, dentro de software tenemos a los entornos de programación, con sus respectivos compiladores, aquí se explicará el funcionamiento, especificaciones y características de los mismos.

Software

1.1.1 MPLABX

MPLAB X es un entorno de desarrollo integrado (IDE), es un software multiplataforma que puede ser utilizado en varios sistemas operativos (Windows, Mac OS, Linux), es utilizado para desarrollar aplicaciones para microcontroladores Microchip y controladores de señales digitales. Se llama un entorno de desarrollo integrado (IDE), porque proporciona un ambiente integrado único para desarrollar código para microcontroladores embebidos.

MPLAB X trae muchos cambios a la cadena de herramientas de desarrollo de microcontrolador PIC's. A diferencia de versiones anteriores de MPLAB que se desarrollaron completamente por Microchip, MPLAB X está basado en el código abierto NetBeans IDE de Oracle. Lo que ha permitido añadir muchas características solicitadas con frecuencia, proporcionando una arquitectura mucho más extensible a futuro.

Características:

- Proporciona una nueva interfaz gráfica para navegar por códigos más complejos.
- Admite varias configuraciones dentro del mismo proyecto.
- Soporta múltiples versiones del mismo compilador.
- Soporte para múltiples herramientas de depuración del mismo tipo
- Soporta análisis de datos en tiempo real.

- Permite importar proyectos existentes de MPLAB 8 y utilizar otro IDE utilizando el mismo código fuente.
- Soporta hipervínculos para una navegación rápida a las declaraciones e includes.
- MPLAB X puede controlar los cambios dentro del mismo sistema, utilizando históricos locales.
- Dentro MPLAB X, un usuario puede configurar su propio formato de Código.

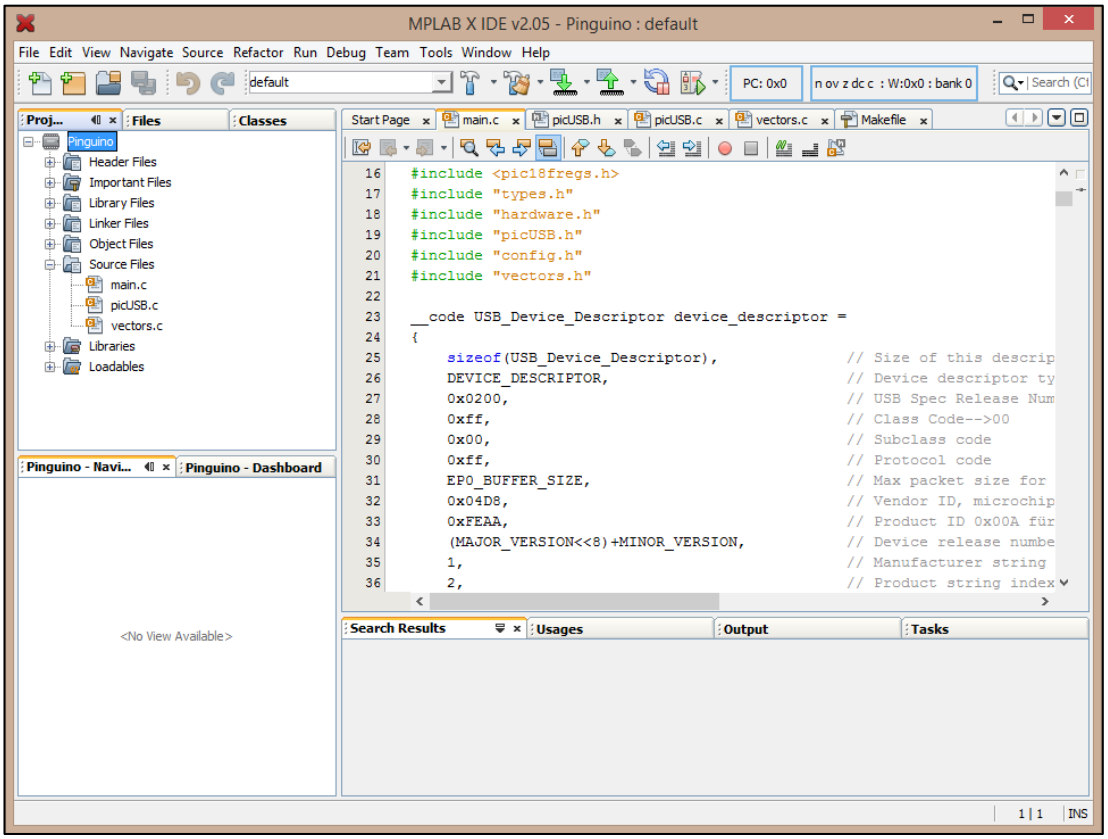

Figura 1. Entorno MPLABX

En la Figura 1 observamos el entorno de programación, en la parte superior tenemos los menús, y a continuación se encuentran los iconos de uso genérico, en la parte izquierda tenemos un navegador donde se puede seleccionar los archivos del proyecto, además de las funciones en la pestaña Classes.

En la Figura 2 observamos los iconos básicos de compilación, programación y depuración.

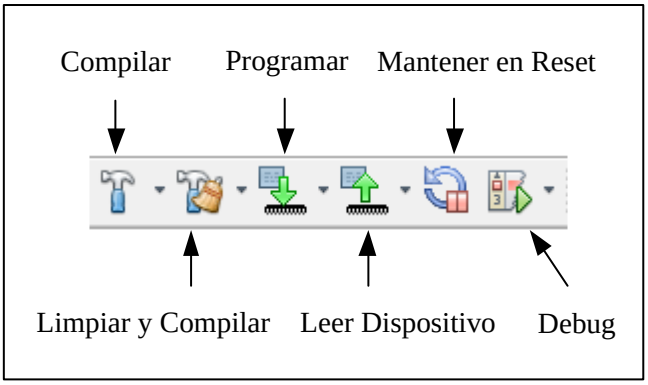

Figura 2. Iconos Básicos MPLABX

1.1.2 Compilador SDCC

Es un compilador optimizado de C, que soporta una gran variedad de microcontroladores, como los basados en Intel MCS51, Maxim(Dallas), Freescale(Motorola), los basados en HC08, los basados en Zilog Z80, y algunos modelos de Microchip de 8bits, principalmente PIC16 y PIC18. El código fuente del compilador está distribuido bajo licencia GPL.

Entre las principales características de optimización tenemos:

- Eliminación de subexpresiones globales
- Optimización de bucles o ciclos
- Eliminación de código muerto
- Salta tablas por declaraciones de estados (switch)

Hay que tener en cuenta algunas consideraciones para utilizar este compilador.

- Es necesario tener instalado el compilador SDCC en la raíz del disco, y debe estar incluido en el Path del sistema.
- El compilador SDCC requiere de GPUTILS(GNU PIC Utilities)
- Python (version 2) también es necesario, para compilar el proyecto, igualmente debe estar incluido en el Path.
- Para ser utilizado con MPLABX es necesario instalar en plugin, esto se realiza dentro de MPLABX, en el Menú Tools, dentro del cual tenemos la opción Plugins, dentro tenemos la Pestaña Available Plugins, donde seleccionamos SDCC Toolchain, y la opción instalar, como se observa en la Figura 3. Es necesario reiniciar MPLABX

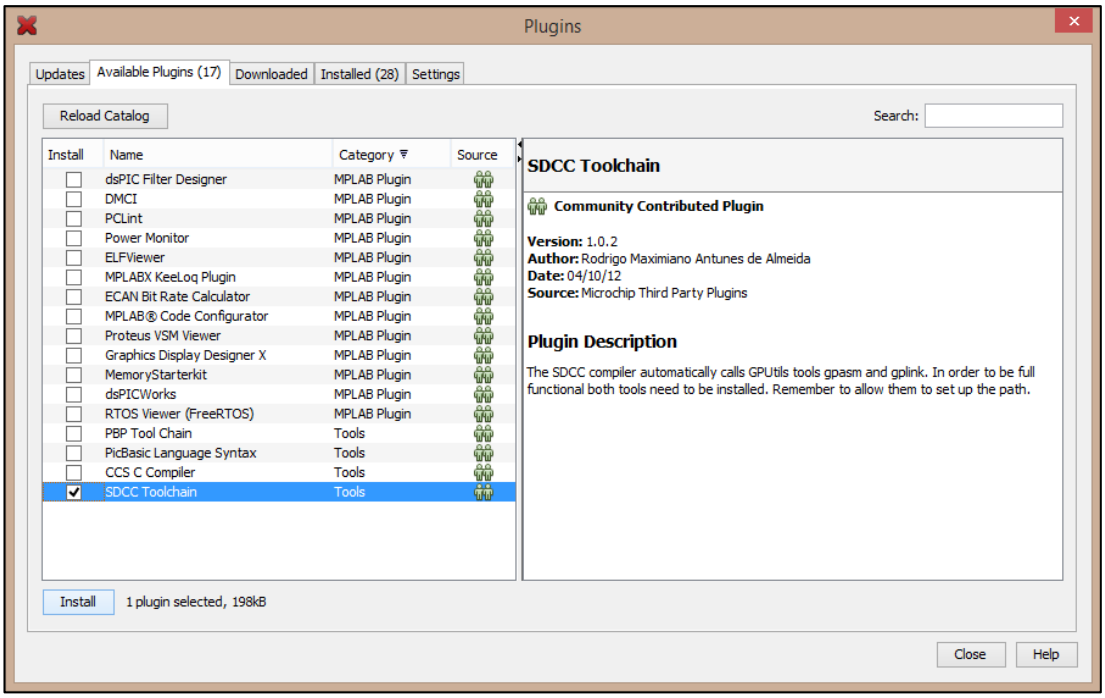

Figura 3**.** Instalación del plugin SDCC

Soporta los siguientes tipos de datos:

Tabla 1. Datos soportados en SDCC

Tipos	Ancho	Por defecto	Rango con signo	Rango sin signo
BOOL	8 bits, 1 byte	Sin signo	-	0, 1
char	8 bits, 1 byte	Con signo	-128, +127	0, +255
Short	16 bits, 2 bytes	Con signo	-32768, +32767	0, +65535
int	16 bits, 2 bytes	Con signo	-32768, +32767	0, +65535
long	32 bits, 4 bytes	Con signo	-2147483648, +2147483647	0, +4294967295
long long	64 bits, 8 bytes	Con signo		
float	4 bytes	Con signo		1.175494351E-38 3.402823466E+38
pointer	1, 2, 3 o 4 bytes	Genérico		

1.1.3 KiCad

KiCad es un software gratuito de diseño electrónico, facilita el diseño de esquemas para circuitos electrónicos y su conversión a circuitos impresos. Se puede crear y editar una gran cantidad de componentes, utilizarlos en esquemas, y en el diseño de

circuitos impresos, en varias capas con visualización en 3D.

KiCad es un software de código abierto (GNU GPL v2), multi-plataforma, corre en Windows, Linux, y Apple OSX, este proyecto fue iniciado por Jean-Pierre Charras

KiCad está organizado en siete partes:
Kicad: Administrador de proyectos.
Eeschema: Editor de esquemas.
Cvpcb: Selector de huellas de los componentes utilizados en el esquemático.
Pcbnew: Entorno de diseño para circuitos impresos (PCB).
Gerbview: Visualizador de archivos gerber.
Bitmap2Component: Crea componentes o huellas a partir de imagenes.
Pcb calculator: Calculadora para diseños electrónicos.

KiCad
Administrador de proyectos, permite cargar o crear nuevos proyectos, además permite ejecutar: Eeschema, Cvpcb, Pcbnew, Gerbview, Bitmap2Component, y Pcb calculator, los que se observan en la figura 4.

En la figura 5, se puede observar el administrador de proyectos, teniendo en la parte superior el menú de proyectos, a la izquierda los proyectos abiertos, y a la derecha los accesos a las aplicaciones individuales.

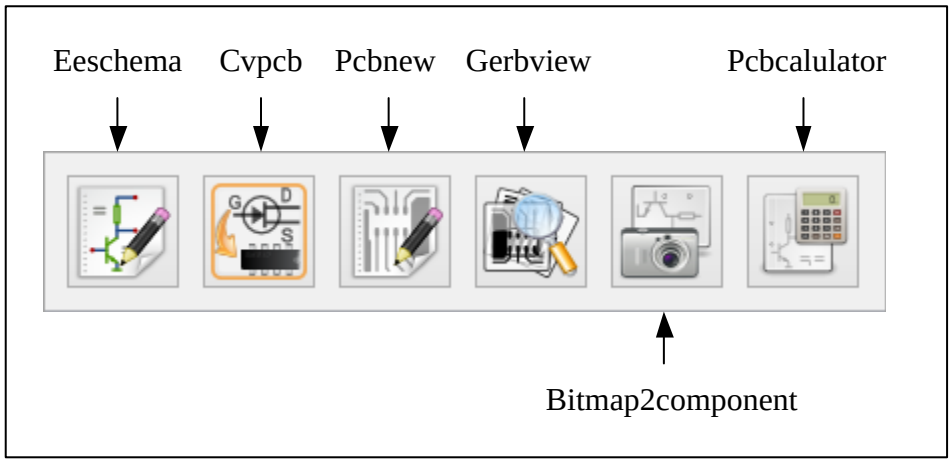

Figura 4**.** Aplicaciones KiCad

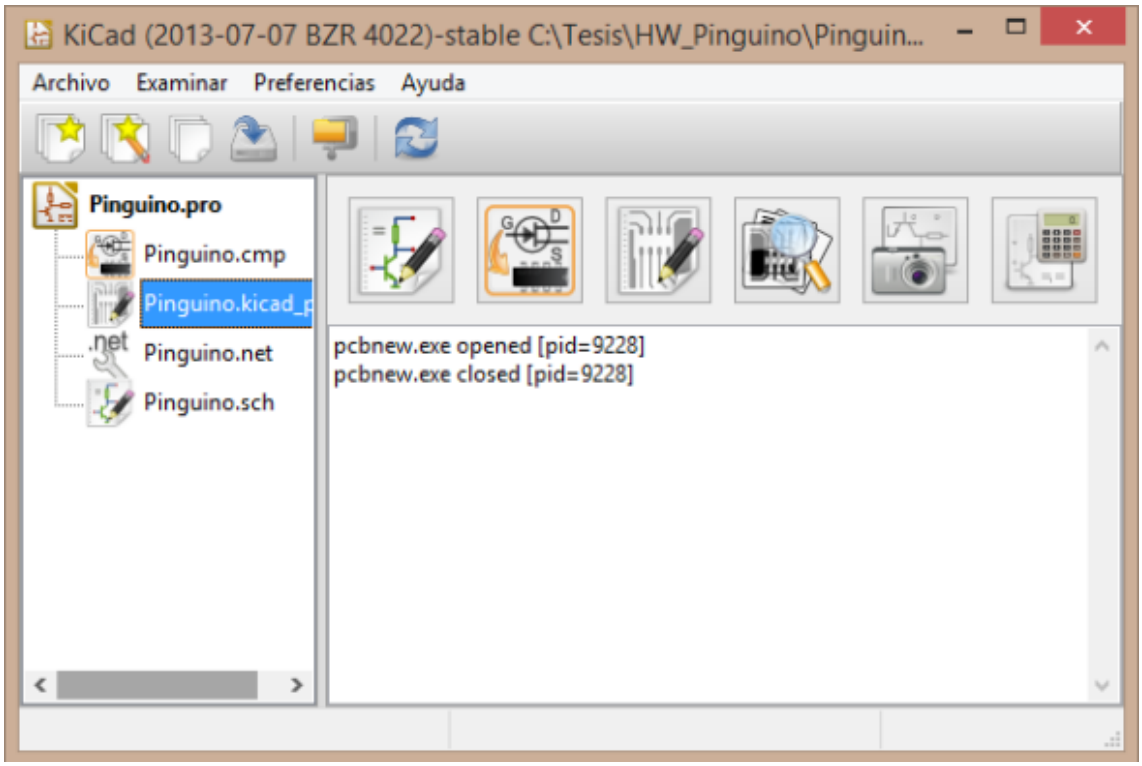

Figura 5**.** Administrador de proyectos Kicad

Eeschema

Permite crear esquemas simples o jerárquicos, ofrece pruebas de los esquemas con la herramienta de chequeo de reglas eléctricas (ERC). También se pueden crear netlists para Pcbnew, o para Spice.

Eeschema gestiona un acceso rápido y directo a la documentación de componentes. En la figura 6, se observa el entorno de diseño de esquemas, teniendo en la parte izquierda el control de las unidades de medida de la hoja, así como de las grillas, en la parte superior se observa la administración del circuito, y en la zona derecha los elementos de diseño.

Además Eeschema ofrece un editor de componentes y librerías, en el que podemos crear o editar componentes fácilmente, en la figura 7 se observa el editor.

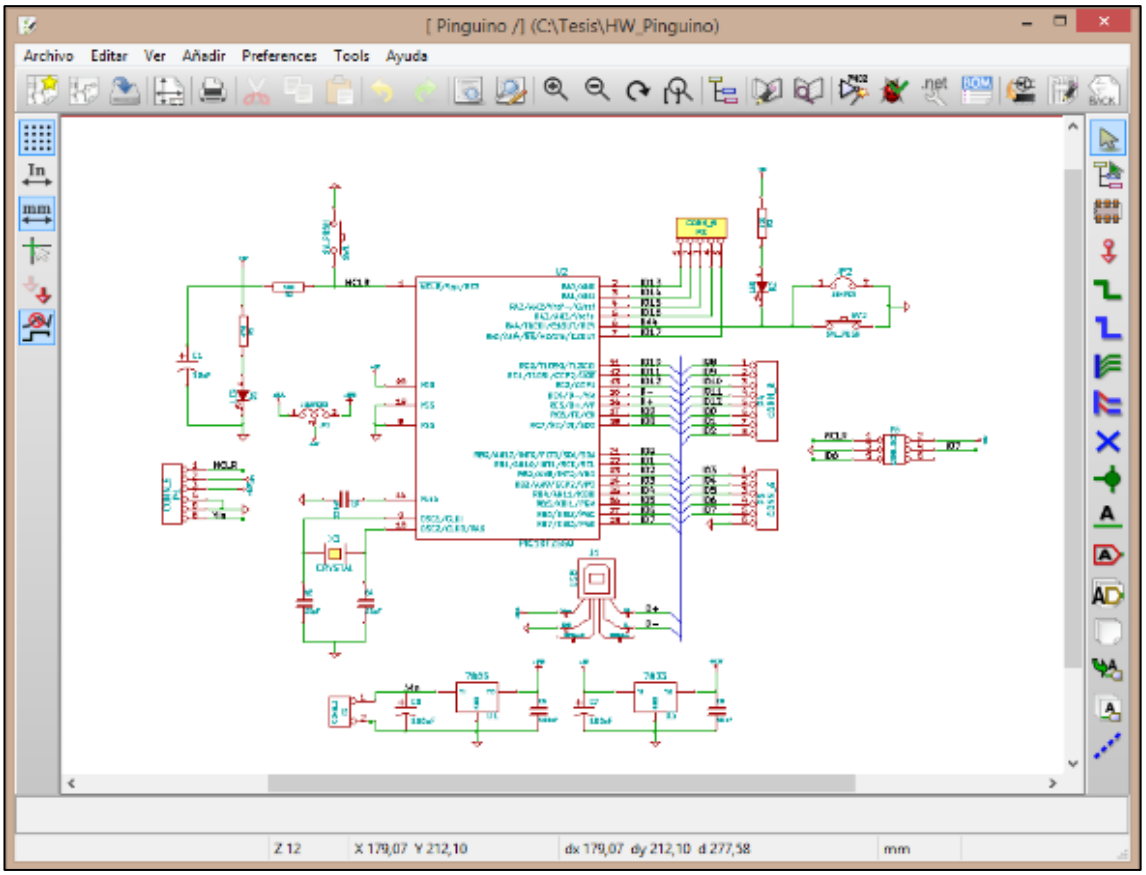

Figura 6**.** Eeschema Editor de esquemas.

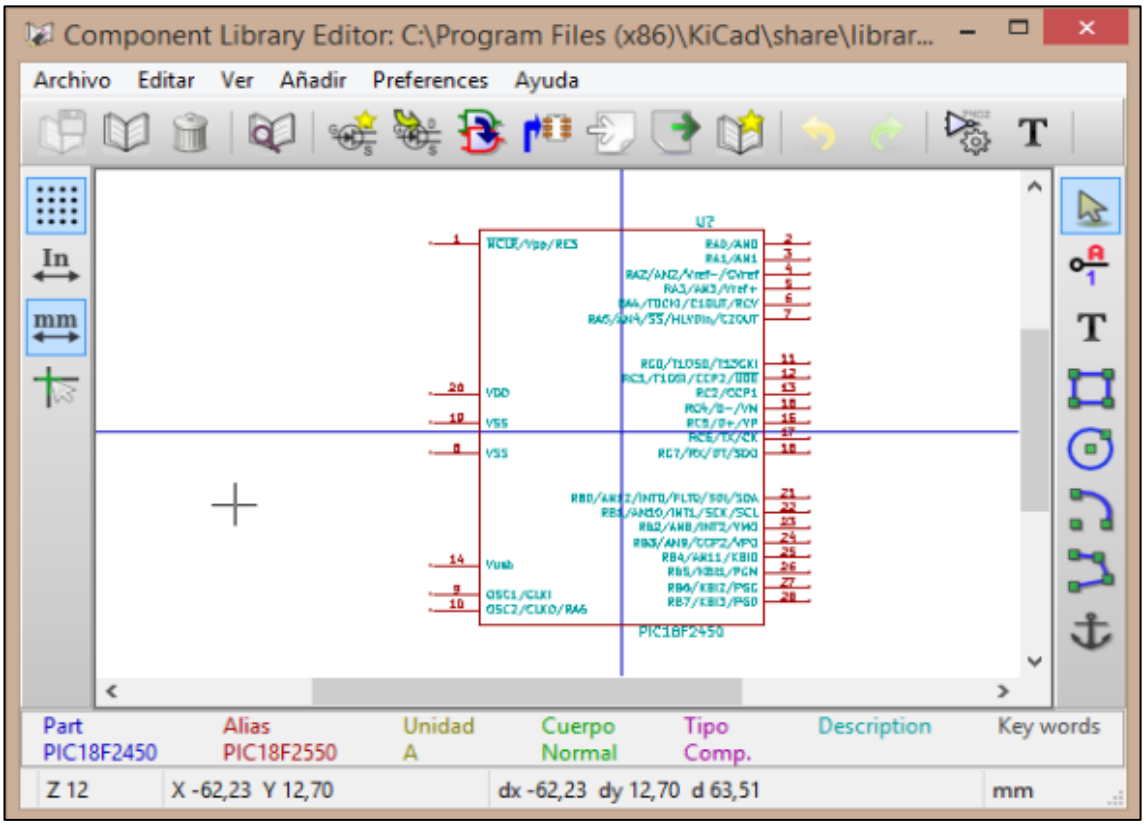

Figura 7**.** Editor de componentes y librerías

Cvpcb

Permite completar un archivo netlist, generado por aplicaciones de diseño de esquemas electrónicos, asociando cada componente a un módulo o encapsulado que será utilizado en la tarjeta de circuito impreso.

En la figura 8 se observa el convertidor de componentes a módulos, en la parte superior tenemos las opciones, para asociar componentes y generar documentación de los módulos.

En la parte derecha por defecto aparecen los encapsulados más probables para dicho componente, también se puede mostrar la lista completa de huellas, para elegir entre otras disponibles.

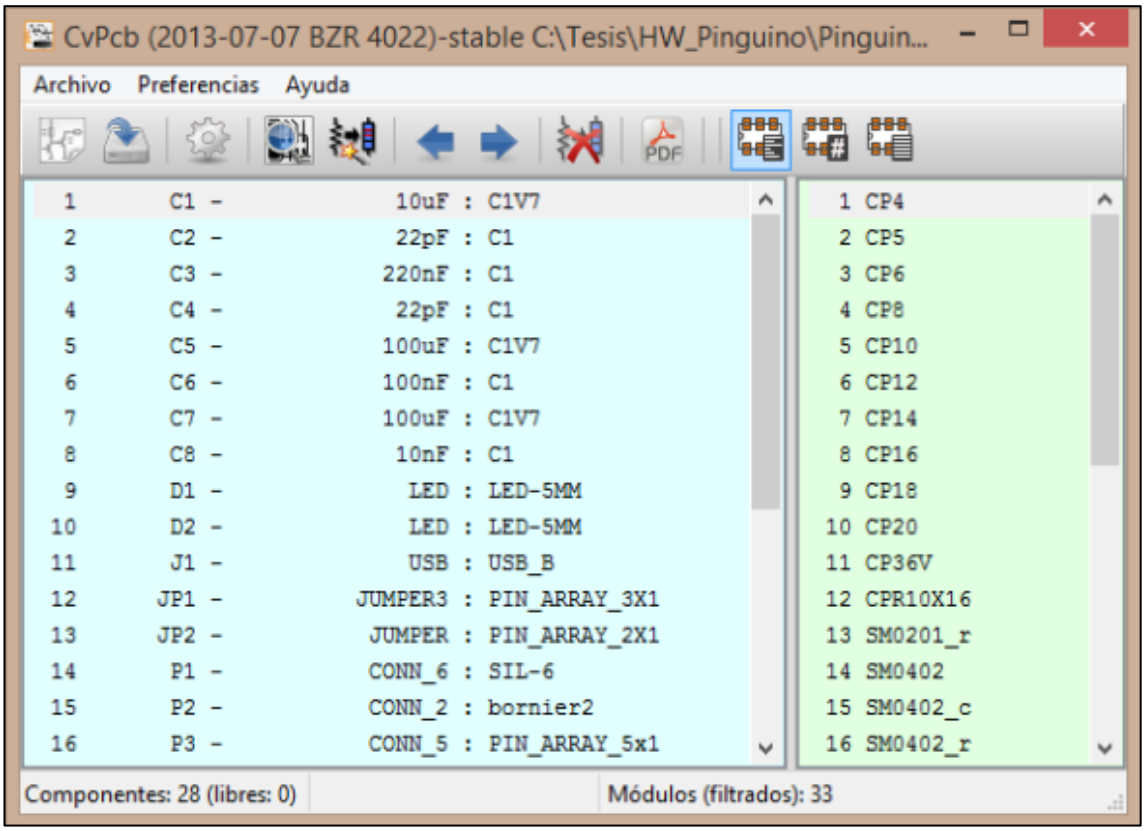

Figura 8. Cvpcb

Pcbnew

En Pcbnew se puede trabajar con hasta dieciséis capas de cobre y además doce capas técnicas, también genera los archivos necesarios para la fabricación de tarjetas de

circuito impreso (GERBER, de perforación y de ubicación de componentes). Pcbnew permite la visualización de los diseños en 3D, utilizando OpenGL.

En la figura 9 se observa el ambiente de trabajo de PCBnew, en la zona izquierda se tiene las opciones de visualización, en la parte superior la administración del diseño, en la zona derecha los elementos de ruteo, y además la visualización por capas del diseño. En la barra superior se puede seleccionar la capa que será ruteada.

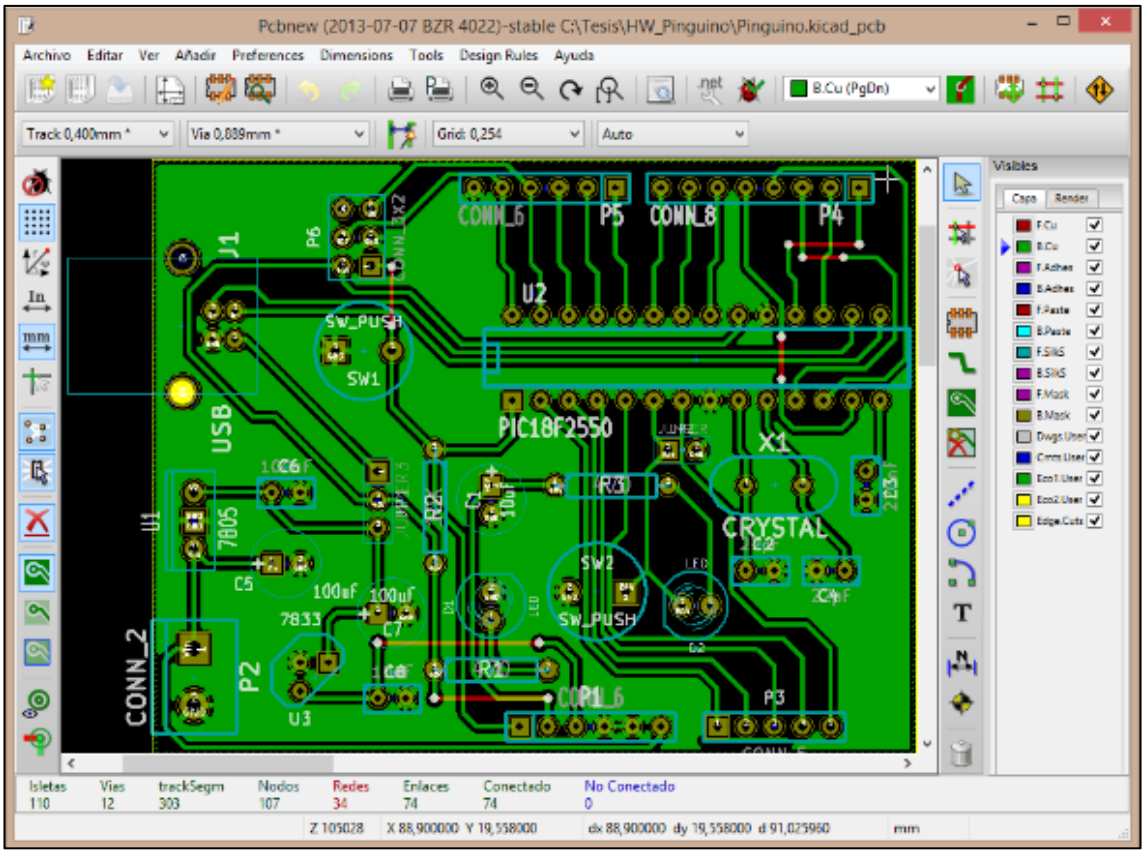

Figura 9**.** Pcbnew: editor de circuitos impresos

En la figura 10 vemos, el editor de módulos que nos permite editar las huellas de los componentes, que serán utilizados en el diseño del PCB, también permite la creación de nuevas huellas, en caso de que algún componente nuevo que no disponga de huellas, o requieran modificación con respecto a alguna existente.

En la figura 11, tenemos el visualizador 3D, en el que se puede apreciar cómo queda el diseño del circuito impreso con sus componentes. La visualización permite rotación en cualquiera de los ejes, lo que nos permite ver a detalle como quedara el

diseño al finalizar su realización.

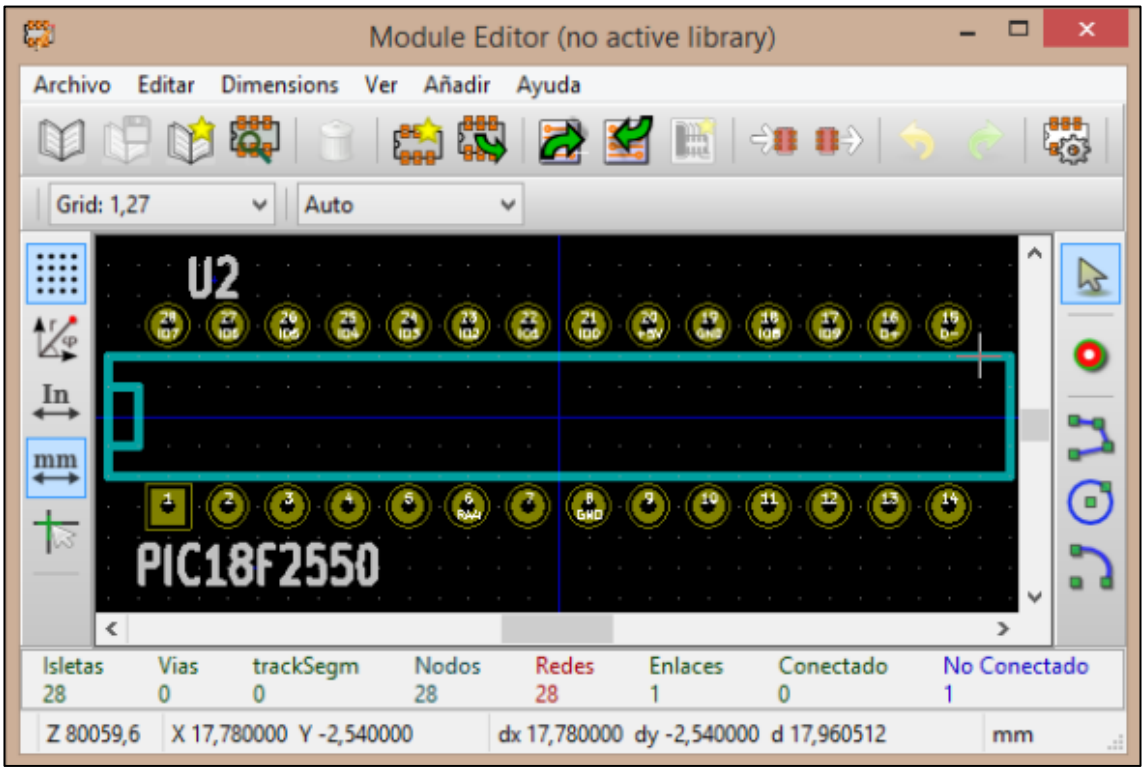

Figura 10. Editor de módulos

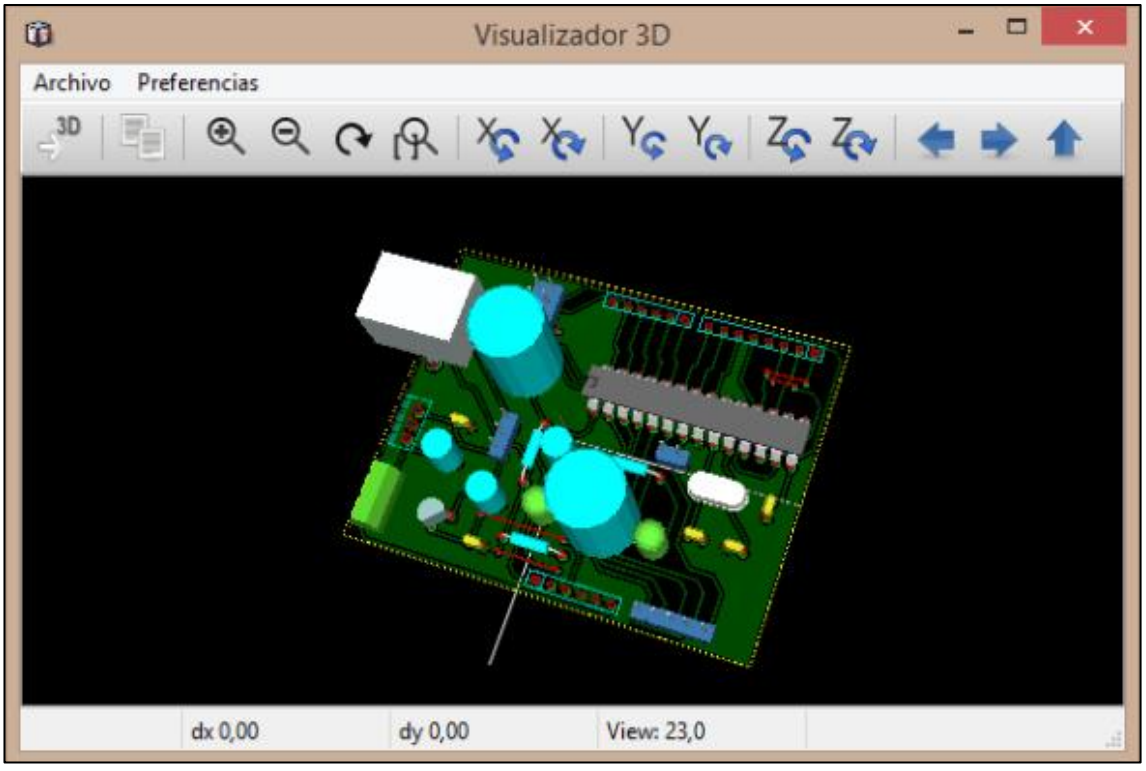

Figura 11. Visualizador 3D

Gerbview

Nos permite visualizar los archivos gerber de fabricación, así como los archivos drill, necesarios para la fabricación de circuitos impresos, además permite exportar archivos a Autodesk DXF, Adobe PDF y otros formatos.

En la figura 12, se observa el entorno de Gerberview, con las opciones de escala a la izquierda, en la parte superior las herramientas básicas, y a la derecha la selección por capas del PCB.Fig 8. Gerbview

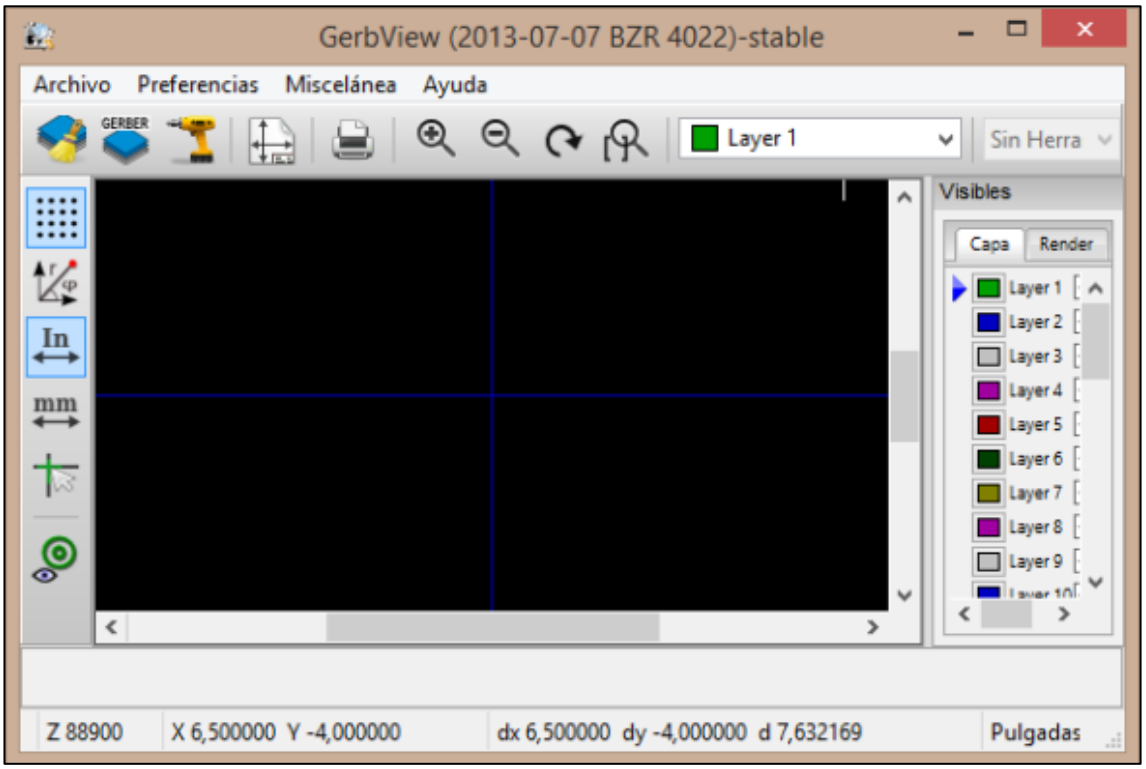

Figura 12**.** GerbView

Bitmap2component

Bitmap to component converter es una herramienta que permite crear un componente o una huella a partir de una imagen. En la figura 13 se observa una imagen que puede ser exportada a Eeschema o Pcbnew, pudiendo elegir si la imagen se utiliza a color, en escala de grises o en blanco y negro.

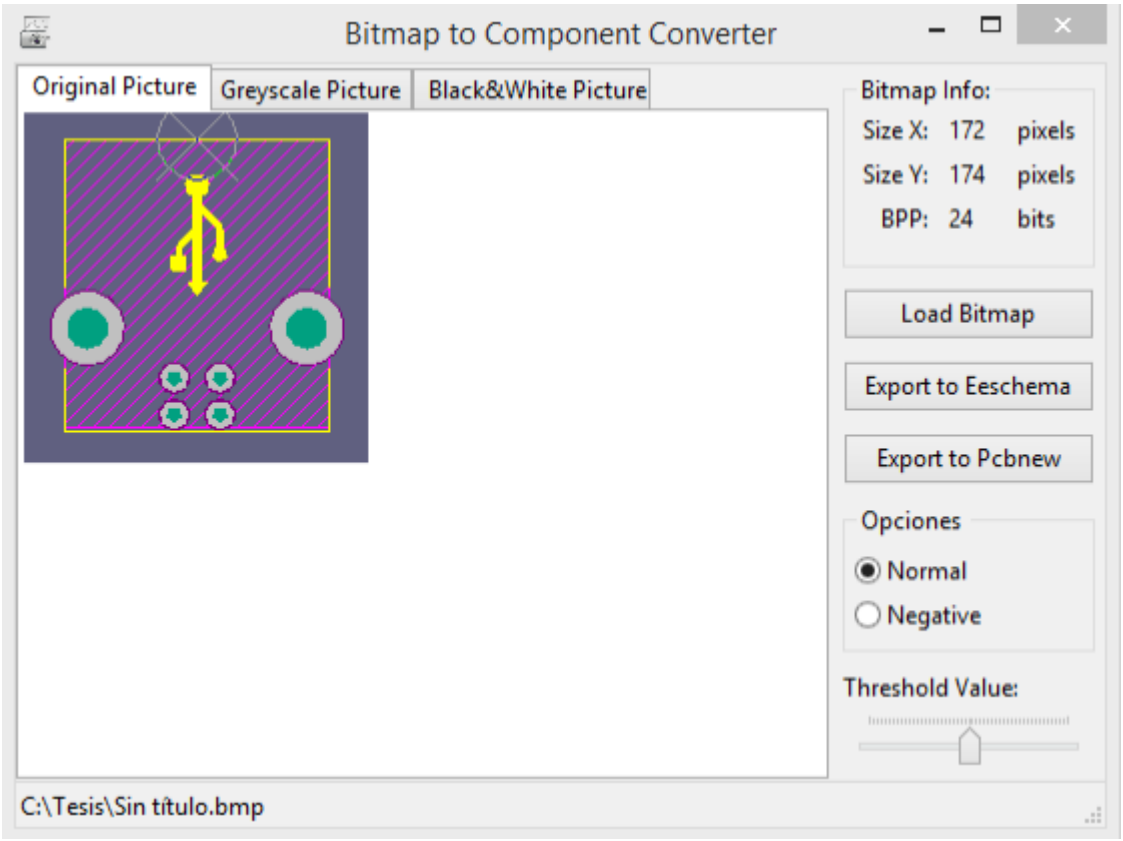

Figura 13. Bitmap2component

Pcb Calculator

Pcb Calculator es una herramienta que permite el cálculo de reguladores, ancho de pista, separación eléctrica, líneas de transmisión, atenuadores RF, además posee la tabla del código de colores y las clases de tarjetas.

En la figura 14 observamos, la calculadora de anchos de pista, la cual nos permite saber cuál será el ancho ideal para una pista de acuerdo a la capa, ya sea esta interna o externa, como podemos observar una pista interna requiere

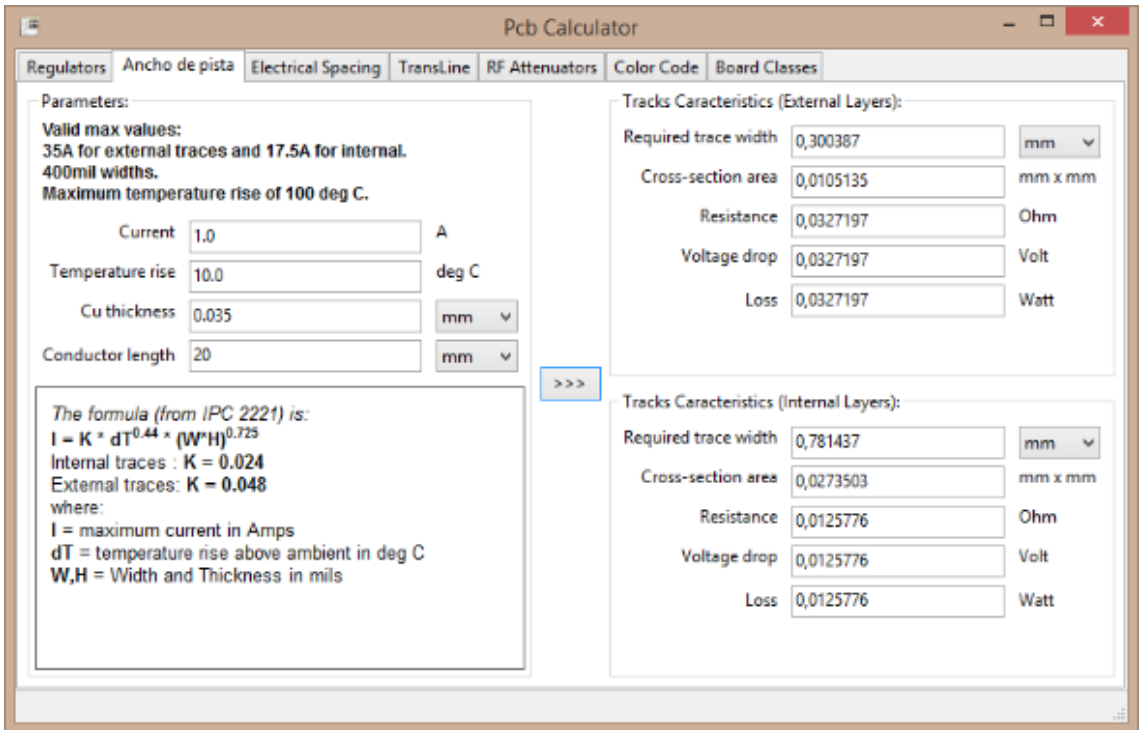

Figura 14**.** Pcb Calculator

Hardware

1.2.1. Microcontrolador PIC18F2550

Especificaciones

El PIC18F2550 es un microcontrolador de 8 bits de la serie 18, que posee un puerto USB mejorado, versión 2.0 soportando alta y baja velocidad con transmisión de datos de hasta 12Mb/s.

Posee una memoria de programa de 32Kbytes, soportando hasta 16384 instrucciones simples, una memoria SRAM de 2048 bytes. Su encapsulado es un DIP28, que posee 24 pines disponibles, soporta velocidades de hasta 48MHz, con un reloj interno de hasta 8MHz.

Posee 10 canales A/D de 10bits, 2 módulos CCP, un módulo SPI, un módulo I^2C, un módulo EUSART, también tiene 2 comparadores, la comunicación con este microcontrolador se puede realizar por medio de un puerto serial, un puerto SPI/I2C, además de un puerto USB.

El microcontrolador posee un convertidor analógico a digital de10 bits, que puede ser

utilizado hasta en 10 pines externos. Además, posee 2 puertos de captura/compara/PWM de 10bis.

Tiene un timer de 8 bits y 3 timers de 16 bits y soporta hasta tres interrupciones externas. En la figura 15 se observa la distribución de los pines, este microcontrolador posee 28 pines

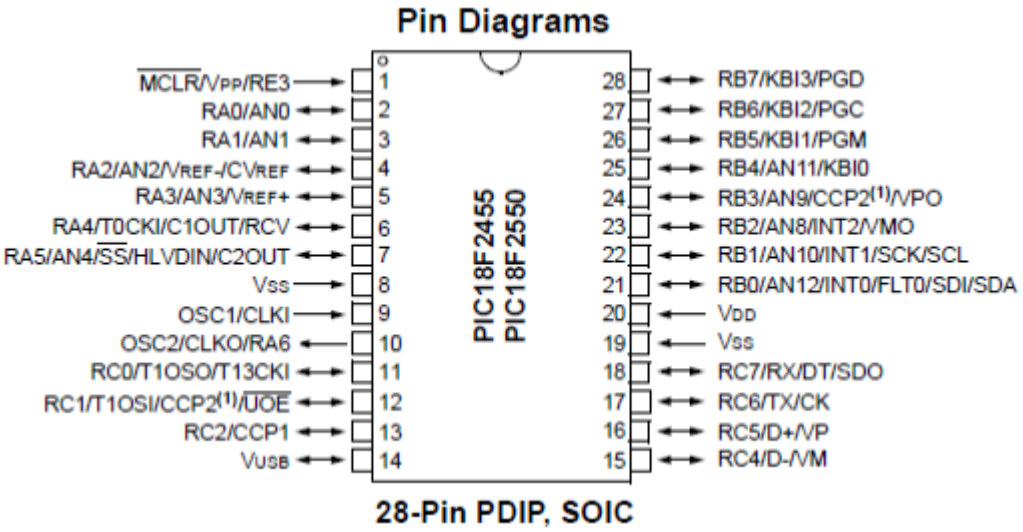

Figura 15**.** Diagrama de Pines PIC18F2550

Características Interface USB

- 1Kbyte RAM de doble puerto + 1Kbyte RAM GP
- Transceptor Full Speed
- 16 Endpoints(in/out)
- Resistencias de pull up internas (pines D+ y D-)
- Funcionamiento a 48MHz (12MIPS)
- Compatible pin a pin con los microcontroladores PIC16C7X5

1.2.2. Regulador L7805CV

Es un regulador de 5V a 1A, de la Serie L7800, que soporta una entrada máxima de 35VDC.

La regulación puede variar entre 4.8V y 5.2V. La distribución de los pines se muestra en la figura 16.

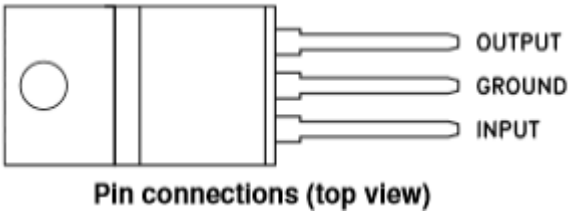

Figura 16**.** L7805CV

1.2.3. Regulador L1117AL

Es un regulador de 3.3V a 800mA de la Serie L7800, que soporta una entrada máxima de 15VDC.

La regulación puede variar entre 3.267V y 3.333V. La distribución de los pines se muestra en la figura 17.

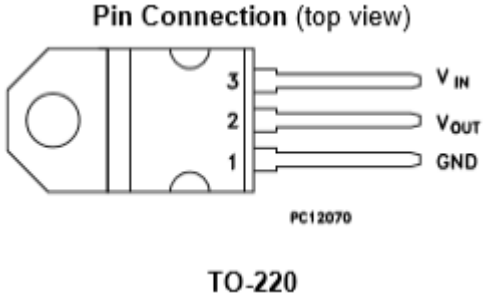

Figura 17. L1117AL

1.2.4. USB

USB (Universal Serial Bus) se basa en una arquitectura tipo serial, pero con una velocidad muy superior a los puertos seriales estándar. Se adoptó una arquitectura serial debido a dos razones principales:

- La arquitectura serial soporta una velocidad de reloj muy superior a comparación de una interfaz paralela, ya que esta genera errores en altas velocidades por los retrasos que se generan en cada hilo.
- Los cables para comunicación serial son más económicos.

Los dispositivos USB según su velocidad de transferencia de datos se clasifican en cuatro tipos:

- Baja velocidad (1.0): Tasa de transferencia de hasta 1,5 Mbit/s.
- Velocidad completa (1.1): Tasa de transferencia de hasta 12 Mbit/s.
- Alta velocidad (2.0): Tasa de transferencia de hasta 480 Mbit/s (60 MB/s) pero con una tasa real práctica máxima de 280 Mbit/s (35 MB/s).
- Superalta velocidad (3.0): Tiene una tasa de transferencia de hasta 4,8 Gbit/s (600 MB/s). La velocidad del bus es diez veces más rápida que la del USB 2.0.

Conectores USB

Existen dos tipos de conectores USB:

- Los conectores tipo A, cuya forma es rectangular, y se utilizan normalmente en dispositivos que no requieren de mucho ancho de banda.
- Los conectores tipo B tienen una forma cuadrada y se utilizan principalmente en dispositivos de alta velocidad
- Funcionamiento

Una de las características principales de USB es que puede utilizarse como fuente de alimentación para ciertos dispositivos, utiliza cuatro hilos (Alimentación del bus 5V, GND y dos para datos D+ y D-)

Las señales del USB se transmiten en un cable de par trenzado con impedancia característica de 90 Ω ± 15% (D+ y D-). Éstos utilizan señalización diferencial en half dúplex excepto el USB 3.0 que utiliza un segundo par de hilos para realizar una comunicación en full dúplex. La comunicación se realiza en modo diferencial porque reduce el efecto del ruido electromagnético en enlaces largos.

Este puerto sólo admite la conexión de dispositivos de bajo consumo, es decir, que tengan un consumo máximo de 100 mA por cada puerto; sin embargo, en caso de que estuviese conectado un dispositivo que permite 4 puertos por cada salida USB (extensiones de máximo 4 puertos), entonces la energía del USB se asignará en unidades de 100 mA hasta un máximo de 500 mA por puerto.

Descriptores USB

Todos los dispositivos USB tienen descriptores que permiten al anfitrión saber la información del dispositivo, fabricante, versión soportada de USB, formas de configurarse, numero de endpoints, tipos, etc.

Los más comunes son:

- Descriptores de dispositivo
- Descriptores de configuración
- Descriptores de interfaz
- Descriptores de endpoint
- Descriptores de cadena

Los dispositivos USB solamente pueden tener un descriptor de dispositivo, el cual contiene revisión del USB, ID del producto y vendedor, utilizados para poder asignar los drivers apropiados y el número de configuraciones posibles.

El descriptor de configuración especifica valores como el consumo de energía, si el dispositivo es alimentado por el bus o si es autoalimentado, y el número de interfaces que tiene, solo se puede utilizar una configuración a la vez.

El estándar USB dispone que los descriptores siempre sean 1:1, para cada función de un dispositivo existe un descriptor, y a la vez un driver para cada función (y una interfaz)

Bootloader

En microcontroladores, bootloader es un programa que se encuentra en el microcontrolador que permite cargar los programas de usuarios sin la necesidad de un programador.

El bootloader debe ser cargado al microcontrolador una sola vez a través de un programador externo como el ICD, Pickit2, etc. Una vez programado el bootloader ya no es necesario un programador externo, la carga de programas compilados por el usuario solamente se realizar a través de un canal de comunicación como el puerto usb, serie o paralelo de la computadora.

Para la transferencia del archivo compilado. hex es necesaria un aplicación de escritorio que permite la comunicación entre la PC y el firmware del microcontrolador, también existen aplicaciones que permite compilar y cargar aplicaciones a un bootloader, como Arduino, Chipkit o Pinguino.

Ventajas del Bootloader

Los bootloaders han sido utilizados varios años, principalmente por varios proyectos de software libre, como son Arduino, Pinguino, chipKIT, etc. Estos proyectos han logrado un gran alcance debido a que resultan económicos, ya que no es necesario utilizar un programador externo, y no necesitan de ningún hardware adicional para su correcto funcionamiento, además se presenta como una herramienta más sencilla, para el fácil aprendizaje de estudiantes y aficionados a la electrónica.

Inconvenientes

El principal inconveniente de utilizar un bootloader es el excesivo consumo de memoria, que muchas veces conlleva a la re-ubicación del vector o vectores de interrupción en memoria, por lo que será necesario ejecutar dos instrucciones adicionales, para re-ubicar los vectores de interrupción en las nuevas posiciones de memoria.

Consideraciones de diseño

Normalmente se considera que el bootloader debe ocupar el menor espacio posible en la memoria flash del microcontrolador, sin embargo, es relativo, además depende del canal de comunicación utilizado, por ejemplo, si se utiliza un bootloader que necesita conexión USB siempre ocupara más memoria que otro que utilice un puerto serial rs232.

Además, se debe considerar en el diseño del bootloader la protección de direcciones en donde está cargado el bootloader con el fin de evitar sobre-escritura, esto se puede hacer por dos métodos: por software o por hardware.

- Protección por software: El firmware se encarga de revisar que las direcciones implicadas en la escritura no pertenezcan a las direcciones del

propio bootloader, antes de escribir en la memoria flash un nuevo programa

- Protección por hardware: Algunos dispositivos de la familia PIC18 permiten la protección contra escritura, de un determinado número de registros en la parte baja de la memoria flash, a través de los bits de configuración del microcontrolador.

Empezando con el Bootloader

Un Bootloader para microcontroladores que cargue aplicaciones de usuario a través de un puerto de comunicación de la PC se divide en dos partes, por una parte se trata del propio bootloader que debe ser cargado en el microcontrolador y por otro lado está la aplicación de escritorio necesaria para poder enviarle el archivo .hex al bootloader.

El principio de funcionamiento de un bootloader para saber si debe ejecutar la aplicación que se encuentra en la memoria flash del microcontrolador o si debe cargar una nueva aplicación de usuario es la siguiente, luego de un reset espera por la llegada de un evento que determine la acción a ejecutarse, este evento puede ser provocado por hardware (ej. cambio de estado en algún pin del microcontrolador) o por software (ej. llegada de un comando por un canal de comunicación).

En la figura 18 se tiene de manera más detallada de cómo quedaría mapeada la memoria ROM del microcontrolador. Donde el vector de reset apunta al comienzo del bootloader.

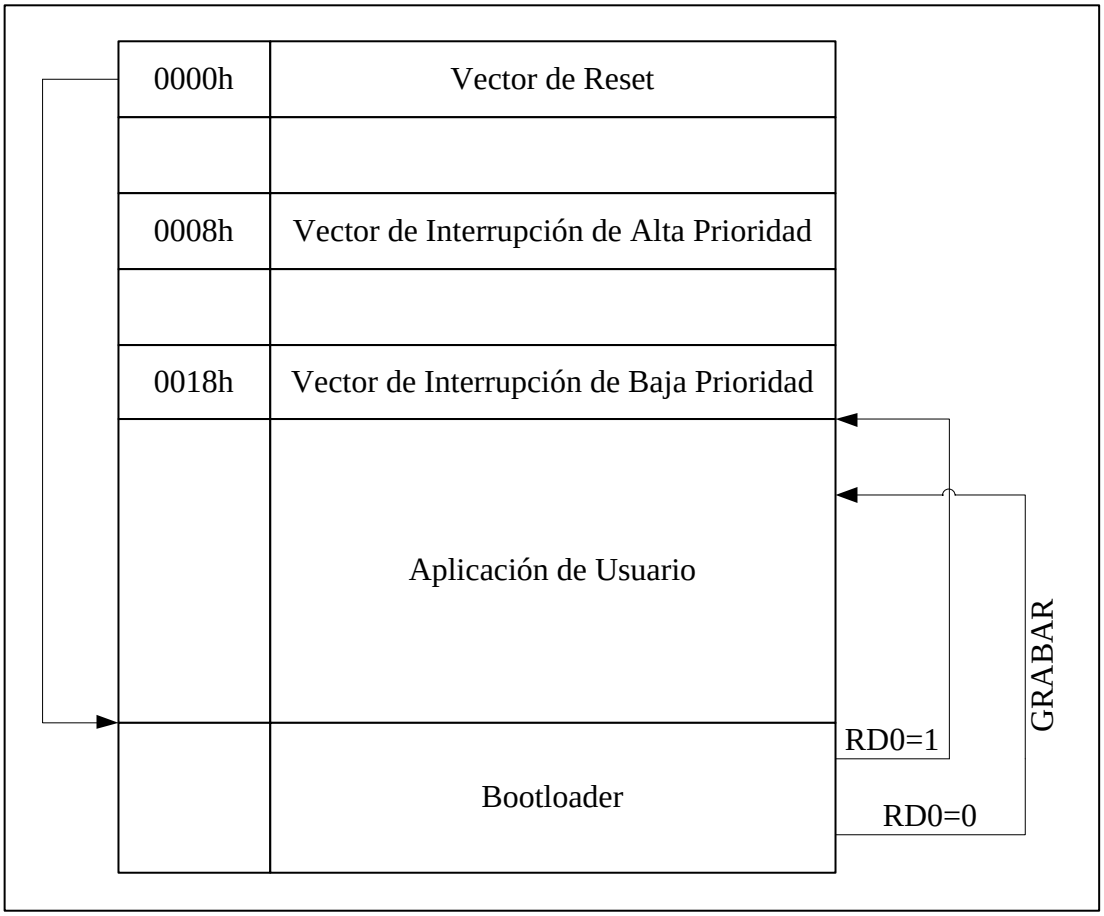

Figura 18**.** Memoria ROM de un microcontrolador con bootloader

Conclusiones

Luego de conocer los recursos disponibles con sus características, se determinó que son de fácil alcance y además son económicos, por lo que cumplen las características deseadas, al utilizar software libre, reduce costos económicos, para desarrollar el proyecto, lo que lo convertirá en una herramienta útil y económica.

Las aplicaciones como KiCAD poseen grandes ventajas, al ser gratuitas, pero no poseen las todas las ventajas que se tiene en aplicaciones de pago como Altium Designer.

CAPÍTULO 2

DISEÑO DE HARDWARE

Introducción.

En esta parte se considerarán las necesidades de una herramienta que pueda ser utilizada por un hobista y a la vez por un profesional, de una manera sencilla, con un diseño sencillo y practico, manteniendo en lo posible compatibilidad con una placa Arduino.

2.1 Consideraciones de Diseño

La alimentación puede ser obtenida desde un terminal de 2 pines que ira conectado con un diodo rectificador para evitar riesgo en caso de que se conecten al revés las polaridades, o también se puede alimentar desde el puerto USB, con el jumper P7 se elige la alimentación utilizada, en la posición superior se utiliza a través del USB, y en la inferior a través del regulador.

El PIC18F2550 requiere de una alimentación de 5VDC, por lo que se utilizara un regulador 7805, que debe ser alimentado con una fuente de al menos 8V, se utiliza un led rojo que indicara que están disponibles los 5V, además se utilizaran filtros para reducir ruido e interferencia que pueden afectar el correcto funcionamiento del módulo USB.

Para mantener compatibilidad con los módulos Arduino, se implementa un regulador de 3.3V que es utilizado por muchos componentes electrónicos, en este caso el regulador utilizado es un LD1117.

El microcontrolador utilizará un cristal para generar los ciclos, en este caso se utilizará uno de 20MHz, acompañado de dos condensadores de 22pF.

El circuito posee una entrada de pulsante (S1) que se utilizara como reset del microcontrolador.

Al pin RA4 están conectados un pulsante y un led verde, en el caso del led sirve como indicador de que se está cargando algún programa, luego pueden ser utilizados con diferentes propósitos.

En el pin VUSB se utiliza el condensador de 220nF que es el recomendado para la comunicación USB.

El diseño del PCB se basará en el modelo del Arduino Uno como se observa en la figura 19, manteniendo compatibilidad con la mayoría de sus pines.

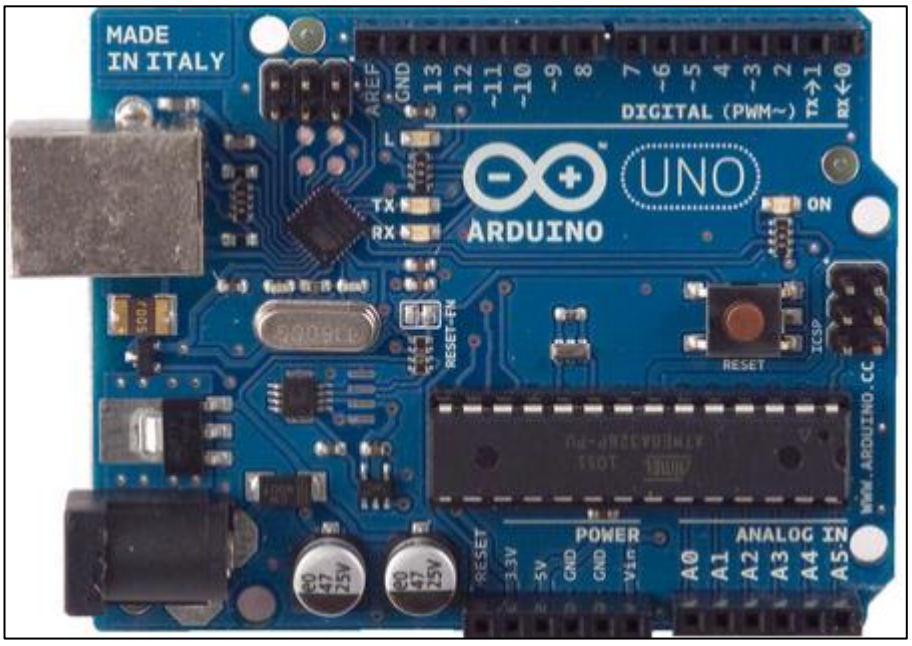

Figura 19. Arduino Uno

2.2 Diseño Electrónico

Las características de este circuito son:

- 18 entradas/salidas digitales, de las cuales 5 pueden ser usadas como entradas analógicas
- Puerto serial para comunicaciones
- Dos salidas PWM (3000Hz)
- 5 Entradas Analógicas.

Tiene un pulsante que servirá para poder resetear a la placa, al reiniciar se habilita el modo bootloader que esperará 5 segundos para ser cargado.

La placa podrá ser alimentada desde el conector USB, o desde un terminal externo, que ira con un regulador. La tabla 2 nos muestra cómo están siendo utilizados los pines del microcontrolador para este proyecto.

Tabla 2. Pines del Microcontrolador

Pin	I/O Digital	Entrada Analógica	Otros	Pin Físico	Microchip
0	Si	No[1]	I2C I/O - SPI SDI	21	RB0/AN12/INT0/FLT0/SDI/SDA
1	Si	No[1]	I2C SCL - SPI SCK	22	RB1/AN10/INT1/SCK/SCL
2	Si	No[1]	-	23	RB2/AN8/INT2/VMO
3	Si	No[1]	-	24	RB3/AN9/CCP2/VPO
4	Si	No[1]	-	25	RB4/AN11/KBI0
5	Si	-	-	26	RB5/KBI1/PGM
6	Si	-	ICSP PGC	27	RB6/KBI2/PGC
7	Si	-	ICSP PGD	28	RB6/KBI2/PGC
8	Si	-	Serial Tx	17	RC6/TX/CK
9	Si	-	Serial Rx – SPI SDO	18	RC7/RX/DT/SDO
10	Si	-	-	11	RC0/T1OSO/T13CK
11	Si	-	PWM	12	RC1/T1OSI/CCP2/UOE
12	Si	-	PWM	13	RC2/CCP1
13	Si	Si[2]	-	2	RA0/AN0
14	Si	Si[2]	-	3	RA1/AN1
15	Si	Si[2]	-	4	RA2/AN2/VREF-/CVREF
16	Si	Si[2]	-	5	RA3/AN3/VREF+
17	Si	Si[2]	-	7	RA5/AN4/SS/HLVDIN/C2OU
Run	Solo salida	-	Run Led	6	RA4/T0CKI/C1OUT/RCV
USB+	-	-	USB	16	RC5/D+/VP
USB-	-	-	USB	15	RC4/D-/VM
Vusb	-	-	USB	14	VUSB
Reset	-	-	Pulsante Reset	1	MCLR/VPP/RE3
Vdd(5V)	-	-	-	20	Vdd
Vss(GND)	-	-	-	8	Vss
Vss(GND)	-	-	-	19	Vss
OSC1	-	-	Cristal	9	OSC1/CLKI
OSC2	-	-	Cristal	10	OSC2/CLKO/RA6

Nota1: Pueden ser configuradas como analógicas, pero no está soportado por el software.

Nota 2: Si un pin entre el 14 y 17 es configurado como analógico, los demás también serán analógicos.

2.3 Diseño del PCB

En la figura 20, se observa el diseño del PCB, el cual se encuentra diseñado a 2 capas, manteniendo la forma del Arduino UNO, en la forma y posición de las peinetas, donde tenemos acceso externo a los pines del microcontrolador

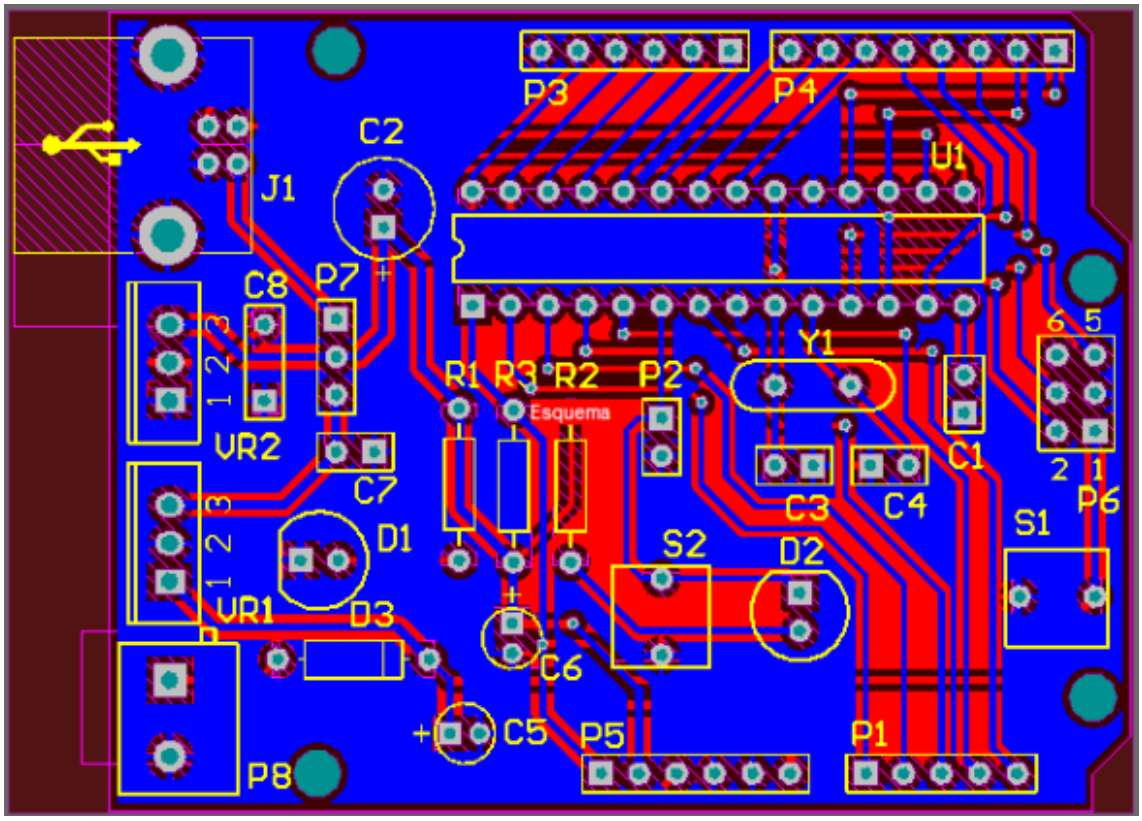

Figura 20**.** Visualización del PCB a 2 capas

Visualización 3D

En la figura 21 observamos el diseño en 3D lo que nos permite visualizar de mejor manera el diseño realizado, y como se lo vería en la realidad.

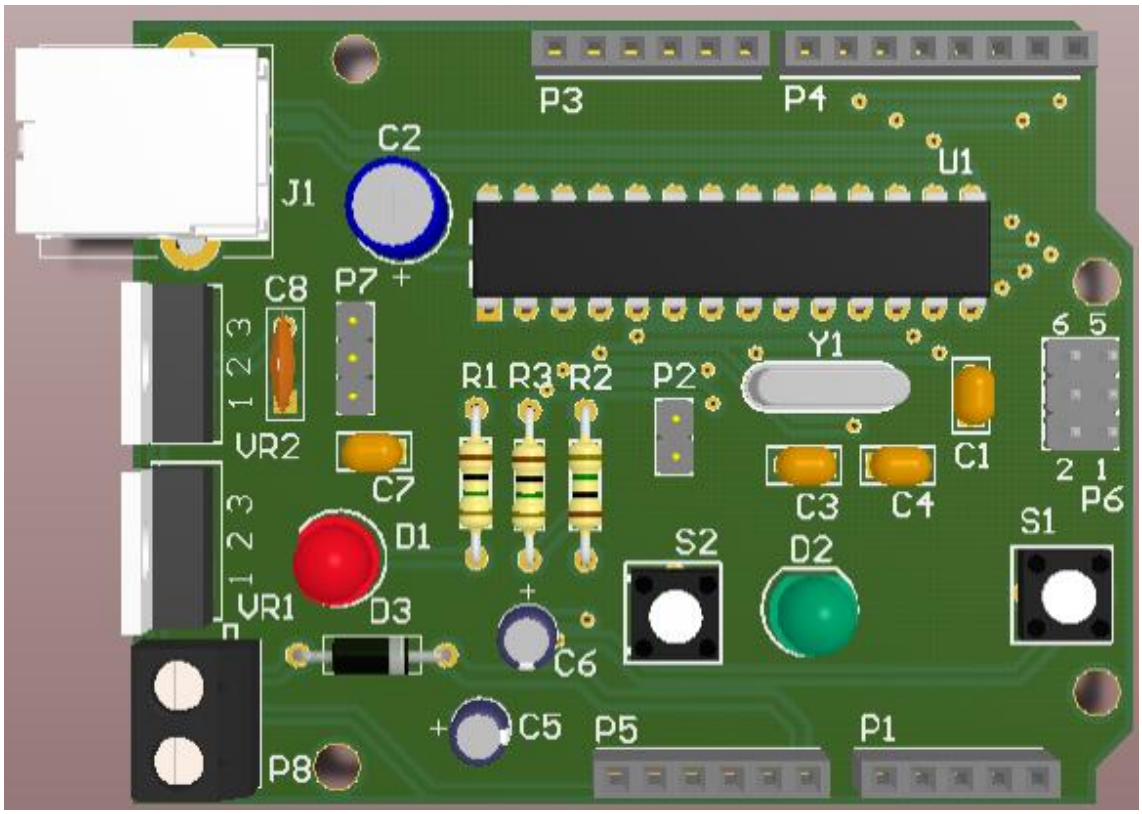

Figura 21**.** Diseño 3D

Conclusiones

El diseño se realizó de una manera compacta, únicamente con los componentes necesarios, cumpliendo con la compatibilidad en la mayoría de los pines con respecto a un Arduino Uno, al ser de un tamaño pequeño resulta ser muy económico, y fácil de armar incluso para un hobista.

CAPITULO 3

FIRMWARE

Introducción.

Este capítulo se enfoca en el firmware utilizado, el cual es una adaptación del firmware pingüino v4.8, el cual ha sido modificado para poder ser utilizado con el entorno de programación MPLABX, utilizando el compilador SDCC, el cual fue descrito en el Capítulo 1, se requiere una versión superior a la 3.2 del compilador.

4.1. Archivos de Cabecera (.h)

Los archivos de cabecera más conocidos como headers, son archivos que contienen declaraciones directas de funciones, variables, definiciones y otros identificadores, es muy común utilizar estos archivos al programar en C.

Se utilizan los siguientes archivos de cabecera: “pic18fregs.h”, “types.h”, “hardware.h”, “picUSB.h”, “config.h” y “vectors.h”, los cuales serán detallados a continuación.

3.1.1 “types.h”

El archivo “types.h” contiene definiciones genéricas de tipos de variable.

```
#ifndef TYPES_H_
#define TYPES_H_

typedef unsigned char byte;
typedef unsigned int  word;
typedef unsigned long dword;

#endif
```

3.1.2 “hardware.h”

En el archivo “hardware.h” se configura los ticks que utilizara el timer, el pin que se utilizara para el led, además de definiciones para el bootloader.

```
/********************************************************************
        Title:    USB Pinguino Bootloader
        File:     hardware.h
        Descr.: bootloader def. (version, speed, led, tempo.)
```

```
        Author:  RÃ©gis Blanchot <rblanchot@gmail.com>

        This file is part of Pinguino (http://www.pinguino.cc)
        Released under the LGPL license (http://www.gnu.org/licenses/lgpl.html)
*********************************************************************************/

#define LOW_SPEED                                                       1
#define HIGH_SPEED                                                      0

/*********************************************************************************
        boot blinking led
*********************************************************************************/

#if defined(__18f2455)  || defined(__18f4455)  || \
   defined(__18f2550)  || defined(__18f4550)  || \
   defined(__18f25k50) || defined(__18f45k50) || \
   defined(__18f13k50) || defined(__18f14k50)

        #define LED_PIN                 4
        #define LED_PORT        _LATA
        #define LED_TRIS        _TRISA

#endif

#if defined(__18f26j50) || defined(__18f46j50) || \
   defined(__18f26j53) || defined(__18f46j53) || \
   defined(__18f27j53) || defined(__18f47j53)

        #define LED_PIN                 2
        #define LED_PORT        _LATC
        #define LED_TRIS        _TRISC

#endif

#define LED_MASK                1 << LED_PIN

/*********************************************************************************
        boot timer delay (practical values between 1..10 seconds)
*********************************************************************************/
//#define BOOT_DELAY_IN_SECONDS                                 10

// timer ticks: clock / 4 / prescaler = 8  / 16 timer overflow
#if SPEED == LOW_SPEED
//      #define CPU_CLOCK               24000000
   #define BOOT_TIMER_TICS              114
#else
//      #define CPU_CLOCK               48000000
   #define BOOT_TIMER_TICS              229
#endif
//  #define TIMER_TICKS_PER_SECOND          CPU_CLOCK / 65536 / 8 / 4
//  #define BOOT_TIMER_TICS                 BOOT_DELAY_IN_SECONDS               *
TIMER_TICKS_PER_SECOND

//#if BOOT_TIMER_TICS > 256
//      #error "Boot delay too large"
//#endif

/*********************************************************************************
        BOOTLOADER COMMANDS
        General Data Packet Structure:
```

```
        .
        _____________________        RxBuffer fields
        |   COMMAND    | 0      [CMD]
        |    LEN       | 1      [LEN]
        |    ADDRL     | 2      [       ] [addrl]
        |    ADDRH     | 3      [ADR.pAdr]: [addrh]
        |    ADDRU     | 4      [       ] [addru]
        |              | 5      [DATA]
        |              |
        .     DATA     .
        .              .
        |              | 62
        |______________|  63

*****************************************************************************/

#define READ_VERSION        0x00
#define READ_FLASH          0x01
#define WRITE_FLASH         0x02
#define ERASE_FLASH         0x03
#define READ_EEDATA         0x04 // unused
#define WRITE_EEDATA                0x05 // unused
#define READ_CONFIG         0x06 // unused
#define WRITE_CONFIG                0x07 // unused
#define RESET               0xFF
```

3.1.3 “config.h”

El archivo “config.h” tiene las configuraciones de la velocidad del cristal, además de los bits de configuración del microcontrolador.

```
/*****************************************************************************
        Title:     USB Pinguino Bootloader
        File:      config.h
        Descr.: configuration bits for supported PIC18F
        Author:  RÃ©gis Blanchot <rblanchot@gmail.com>

        This file is part of Pinguino (http://www.pinguino.cc)
        Released under the LGPL license (http://www.gnu.org/licenses/lgpl.html)
*****************************************************************************/

#include "../../../../../Program Files/SDCC/include/pic16/pic18fregs.h"

/*****************************************************************************/
#if defined(__18f2550) || defined(__18f4550) || \
   defined(__18f2455) || defined(__18f4455)
/*****************************************************************************/

   #if (SPEED == LOW_SPEED)

      #error "    --------------------------------    "
      #error "    No low-speed mode on this device.    "
      #error "    --------------------------------    "
```

```
#elif (CRYSTAL == INTOSC)

   #error "    ---------------------------------    "
   #error "    Internal Crystal CAN NOT DRIVE USB Clock.     "
   #error "    ---------------------------------    "

#else

   #if   (CRYSTAL == 4)
      #pragma config PLLDIV = 1
   #elif (CRYSTAL == 8)
      #pragma config PLLDIV = 2
   #elif (CRYSTAL == 12)
      #pragma config PLLDIV = 3
   #elif (CRYSTAL == 16)
      #pragma config PLLDIV = 4
   #elif (CRYSTAL == 20)
      #pragma config PLLDIV = 5
   #elif (CRYSTAL == 24)
      #pragma config PLLDIV = 6
   #elif (CRYSTAL == 40)
      #pragma config PLLDIV = 10
   #elif (CRYSTAL == 48)
      #pragma config PLLDIV = 12
   #else
         #error "    ---------------------------------    "
         #error "    Crystal Frequency Not supported.     "
         #error "    ---------------------------------    "
   #endif

   #pragma config CPUDIV = OSC1_PLL2          // CPU_clk = PLL/2
   #pragma config USBDIV = 2             // USB_clk = PLL/2
   #pragma config FOSC = HSPLL_HS               // HS osc PLL

   #pragma config FCMEN = ON                    // Fail Safe Clock Monitor
   #pragma config IESO = OFF             // Int/Ext switchover mode
   #pragma config PWRT = ON              // PowerUp Timer
   #pragma config BOR = OFF              // Brown Out
   #pragma config VREGEN = ON                   // Int Voltage Regulator
   #pragma config WDT = OFF              // WatchDog Timer
   #pragma config MCLRE = ON                    // MCLR
   #pragma config LPT1OSC = OFF                 // Low Power OSC
   #pragma config PBADEN = OFF                  // PORTB<4:0> A/D
   #pragma config CCP2MX = ON                   // CCP2 Mux RC1
   #pragma config STVREN = ON                   // Stack Overflow Reset
   #if (LVP == 0)
      #pragma config LVP = OFF                  // High Voltage Programming
   #else
      #pragma config LVP = ON                   // Low Voltage Programming
   #endif
   #if defined(__18f4550)
      #pragma config ICPRT = OFF                // ICP
   #endif
   #pragma config XINST = OFF                   // Ext CPU Instruction Set
   #pragma config DEBUG = OFF                   // Background Debugging
   #pragma config CP0 = OFF              // Code Protect
   #pragma config CP1 = OFF
   #pragma config CP2 = OFF
   #pragma config CPB = OFF              // Boot Sect Code Protect
   #pragma config CPD = OFF              // EEPROM Data Protect
```

```
    #pragma config WRT0 = OFF                   // Table Write Protect
    #pragma config WRT1 = OFF
    #pragma config WRT2 = OFF
    #pragma config WRTB = OFF                   // Boot Table Write Protect
    #pragma config WRTC = OFF                   // CONFIG Write Protect
    #pragma config WRTD = OFF                   // EEPROM Write Protect
    #pragma config EBTR0 = OFF                  // Ext Table Read Protect
    #pragma config EBTR1 = OFF
    #pragma config EBTR2 = OFF
    #pragma config EBTRB = OFF                  // Boot Table Read Protect

  #endif

/*****************************************************************************/
#elif defined(__18f26j50) || defined(__18f46j50)
/*****************************************************************************/

  #if (CRYSTAL == INTOSC)

    #pragma config OSC = INTOSCPLL  // internal RC oscillator, PLL enabled, HSPLL used by
USB
    #pragma config PLLDIV = 2       // 8MHz/2 = The PLL requires a 4 MHz input and it produces a
96 MHz output.

  #else

    #pragma config OSC = HSPLL      // external oscillator, PLL enabled, HSPLL used by USB

    #if   (CRYSTAL == 4)
      #pragma config PLLDIV = 1
    #elif (CRYSTAL == 8)
      #pragma config PLLDIV = 2
    #elif (CRYSTAL == 12)
      #pragma config PLLDIV = 3
    #elif (CRYSTAL == 16)
      #pragma config PLLDIV = 4
    #elif (CRYSTAL == 20)
      #pragma config PLLDIV = 5
    #elif (CRYSTAL == 24)
      #pragma config PLLDIV = 6
    #elif (CRYSTAL == 40)
      #pragma config PLLDIV = 10
    #elif (CRYSTAL == 48)
      #pragma config PLLDIV = 12
    #else
        #error "    ---------------------------------    "
        #error "    Crystal Frequency Not supported.     "
        #error "    ---------------------------------    "
    #endif

  #endif

  // DS39931D-page 40 - 2011 Microchip Technology Inc.
  // PIC18F46J50 family core must run at 24 MHz in order for the USB module
  // to get the 6 MHz clock needed for low-speed USB operation.

  #if (SPEED == LOW_SPEED)
    #pragma config CPUDIV = OSC2_PLL2// 24 MHz (CPU system clock divided by 2)
  #else
    #pragma config CPUDIV = OSC1    // 48 MHz (No CPU system clock divide)
```

```
   #endif

   #pragma config WDTEN = OFF          // WDT disabled (enabled by SWDTEN bit)
   #pragma config STVREN = ON          // stack overflow/underflow reset enabled
   #pragma config XINST = OFF          // Extended instruction set disabled
   #pragma config CP0 = OFF            // Program memory is not code-protected
   #pragma config IESO = OFF           // Two-Speed Start-up disabled
   #pragma config FCMEN = OFF          // Fail-Safe Clock Monitor disabled
   #pragma config LPT1OSC = OFF        // high power Timer1 mode
   #pragma config T1DIG = ON           // Sec Osc clock source may be selected
   #pragma config WDTPS = 256          // 1:256 (1 second)
   #pragma config DSWDTPS = 8192       // 1:8192 (8.5 seconds)
   #pragma config DSWDTEN = OFF        // Disabled
   #pragma config DSBOREN = OFF        // Zero-Power BOR disabled in Deep Sleep
   #pragma config RTCOSC = INTOSCREF   // RTCC uses INTOSC as clock
   #pragma config DSWDTOSC = INTOSCREF // DSWDT uses INTOSC as clock
   #pragma config MSSP7B_EN = MSK7     // 7 Bit address masking
   #pragma config IOL1WAY = OFF        // IOLOCK bit can be set and cleared
   #pragma config WPCFG = OFF          // Write/Erase last page protect Disabled
   #pragma config WPEND = PAGE_0       // Start protection at page 0
   #pragma config WPFP = PAGE_1        // Write Protect Program Flash Page 0
   #pragma config WPDIS = OFF          // WPFP[5:0], WPEND, and WPCFG bits ignored

/*******************************************************************************/
#elif defined(__18f25k50) || defined(__18f45k50)// Config. Words for Internal Crystal (16 MHz) use
/*******************************************************************************/

   #pragma config CFGPLLEN = ON        // PLL Enable Configuration bit (PLL Enabled)
   #pragma config CPUDIV = NOCLKDIV    // 1:1 mode (for 48MHz CPU)

   #if (CRYSTAL == INTOSC)             // Internal 16 MHz Osc.
      #pragma config PLLSEL = PLL3X        // PLL Selection (3x clock multiplier) => 3 x 16 = 48
MHz
      #pragma config FOSC = INTOSCIO      // Oscillator Selection (Internal oscillator)

   #elif (CRYSTAL == 12)
      #pragma config PLLSEL = PLL4X        // PLL Selection (3x clock multiplier) => 3 x 16 = 48
MHz
      #pragma config FOSC = HSOSCIO      // Oscillator Selection (Internal oscillator)

   #elif (CRYSTAL == 16)
      #pragma config PLLSEL = PLL3X        // PLL Selection (3x clock multiplier) => 3 x 16 = 48
MHz
      #pragma config FOSC = HSOSCIO      // Oscillator Selection (Internal oscillator)

   #elif (CRYSTAL == 48)
      #pragma config PLLSEL = PLL3X      // Oscillator used directly
      #pragma config FOSC = HSOSCIO      // Oscillator Selection (Internal oscillator)

   #else
      #error "    ---------------------------------    "
      #error "    Crystal Frequency Not supported.     "
      #error "    ---------------------------------    "

   #endif

   #pragma config LS48MHZ = SYS24X4// USB Low-speed clock at 24 MHz, USB clock divider is
set to 4

   // CONFIG1H
```

```
   #pragma config PCLKEN = ON          // Primary Oscillator Shutdown (Primary oscillator enabled)
   #pragma config FCMEN = OFF          // Fail-Safe Clock Monitor (Fail-Safe Clock Monitor
disabled)
   #pragma config IESO = OFF           // Internal/External Oscillator Switchover (Oscillator
Switchover mode disabled)

   // CONFIG2L
   #pragma config nPWRTEN = OFF        // Power-up Timer Enable (Power up timer disabled)
   #pragma config BOREN = SBORDIS      // Brown-out Reset Enable (BOR enabled in hardware
(SBOREN is ignored))
   #pragma config BORV = 190           // Brown-out Reset Voltage (BOR set to 1.9V nominal)
   #pragma config nLPBOR = OFF         // Low-Power Brown-out Reset (Low-Power Brown-out
Reset disabled)

   // CONFIG2H
   #pragma config WDTEN = SWON         // Watchdog Timer Enable bits (WDT controlled by
firmware (SWDTEN enabled))
   #pragma config WDTPS = 32768        // Watchdog Timer Postscaler (1:32768)

   // CONFIG3H
   #pragma config CCP2MX = RC1         // CCP2 MUX bit (CCP2 input/output is multiplexed with
RC1)
   #pragma config PBADEN = OFF         // PORTB A/D Enable bit (PORTB<5:0> pins are
configured as digital I/O on Reset)
   #pragma config T3CMX = RC0          // Timer3 Clock Input MUX bit (T3CKI function is on RC0)
   #pragma config SDOMX = RC7          // SDO Output MUX bit (SDO function is on RC7)
   #pragma config MCLRE = ON           // Master Clear Reset Pin Enable (MCLR pin enabled; RE3
input disabled)

   // CONFIG4L
   #pragma config STVREN = ON          // Stack Full/Underflow Reset (Stack full/underflow will
cause Reset)
   #if (LVP == 0)
      #pragma config LVP = OFF         // High Voltage Programming
   #else
      #pragma config LVP = ON          // Low Voltage Programming
   #endif
   #pragma config XINST = OFF          // Extended Instruction Set Enable bit (Instruction set
extension and Indexed Addressing mode disabled)
   #if defined(__18f45k50)
   #pragma config ICPRT = OFF          // ICP
   #endif

   // CONFIG5L
   #pragma config CP0 = OFF            // Block 0 Code Protect (Block 0 is not code-protected)
   #pragma config CP1 = OFF            // Block 1 Code Protect (Block 1 is not code-protected)
   #pragma config CP2 = OFF            // Block 2 Code Protect (Block 2 is not code-protected)
   #pragma config CP3 = OFF            // Block 3 Code Protect (Block 3 is not code-protected)

   // CONFIG5H
   #pragma config CPB = OFF            // Boot Block Code Protect (Boot block is not code-protected)
   #pragma config CPD = OFF            // Data EEPROM Code Protect (Data EEPROM is not code-
protected)

   // CONFIG6L
   #pragma config WRT0 = OFF           // Block 0 Write Protect (Block 0 (0800-1FFFh) is not write-
protected)
   #pragma config WRT1 = OFF           // Block 1 Write Protect (Block 1 (2000-3FFFh) is not write-
protected)
   #pragma config WRT2 = OFF           // Block 2 Write Protect (Block 2 (04000-5FFFh) is not write-
```

```
protected)
   #pragma config WRT3 = OFF          // Block 3 Write Protect (Block 3 (06000-7FFFh) is not write-protected)

   // CONFIG6H
   #pragma config WRTC = OFF              // Configuration Registers Write Protect (Configuration registers (300000-3000FFh) are not write-protected)
   #pragma config WRTB = OFF           // Boot Block Write Protect (Boot block (0000-7FFh) is not write-protected)
   #pragma config WRTD = OFF          // Data EEPROM Write Protect (Data EEPROM is not write-protected)

   // CONFIG7L
   #pragma config EBTR0 = OFF         // Block 0 Table Read Protect (Block 0 is not protected from table reads executed in other blocks)
   #pragma config EBTR1 = OFF         // Block 1 Table Read Protect (Block 1 is not protected from table reads executed in other blocks)
   #pragma config EBTR2 = OFF         // Block 2 Table Read Protect (Block 2 is not protected from table reads executed in other blocks)
   #pragma config EBTR3 = OFF         // Block 3 Table Read Protect (Block 3 is not protected from table reads executed in other blocks)

   // CONFIG7H
   #pragma config EBTRB = OFF         // Boot Block Table Read Protect (Boot block is not protected from table reads executed in other blocks)

/*****************************************************************************/
#elif defined(__18f26j53) || defined(__18f46j53)|| \
     defined(__18f27j53) || defined(__18f47j53)
/*****************************************************************************/

   #if (CRYSTAL == INTOSC)            // Internal 8 MHz Osc.
      #pragma config OSC = INTOSCPLL  // Oscillator Selection (Internal oscillator)
      #pragma config PLLDIV = 2       // 8 MHz / 2 : The PLL requires a 4 MHz input and it produces a 96 MHz output.
   #else
      #pragma config OSC = HSPLL      // Oscillator Selection (External oscillator)

      #if   (CRYSTAL == 4)
         #pragma config PLLDIV = 1
      #elif (CRYSTAL == 8)
         #pragma config PLLDIV = 2
      #elif (CRYSTAL == 12)
         #pragma config PLLDIV = 3
      #elif (CRYSTAL == 16)
         #pragma config PLLDIV = 4
      #elif (CRYSTAL == 20)
         #pragma config PLLDIV = 5
      #elif (CRYSTAL == 24)
         #pragma config PLLDIV = 6
      #elif (CRYSTAL == 40)
         #pragma config PLLDIV = 10
      #elif (CRYSTAL == 48)
         #pragma config PLLDIV = 12
      #else
            #error "    ---------------------------------    "
            #error "    Crystal Frequency Not supported.     "
            #error "    ---------------------------------    "
      #endif
```

```
   #endif

   // CONFIG1L
   //#pragma config DEBUG = OFF                    // Background Debugging
   #pragma config XINST = OFF                    // Ext CPU Instruction Set
   #pragma config STVREN = ON            // Stack Full/Underflow Reset (Stack full/underflow will
cause Reset)
   #pragma config CFGPLLEN = ON        // PLL Enable Configuration bit (PLL Enabled)
   #pragma config WDTEN = OFF          // WDT disabled (enabled by SWDTEN bit)

   // CONFIG1H
   #pragma config CP0 = OFF           // Block 0 Code Protect (Block 0 is not code-protected)
   #pragma config CPUDIV = OSC1        // 48 MHz (No CPU system clock divide)

   // CONFIG2L
   #pragma config IESO = OFF                 // Internal/External Oscillator Switchover (Oscillator
Switchover mode disabled)
   #pragma config FCMEN = OFF          // Fail-Safe Clock Monitor disabled
   #pragma config CLKOEC = OFF         // CLKO output disabled on the RA6 pin
   //#pragma config SOSCSEL = HIGH       // Digital (SCLKI) mode selected

   // CONFIG2H
   #pragma config WDTPS = 32768        // Watchdog Timer Postscaler (1:32768)

   // CONFIG3L
   #pragma config DSWDTPS = 8192       // 1:8,192 (8.5 seconds)
   #pragma config DSWDTEN = OFF        // Disabled
   #pragma config DSBOREN = OFF        // Zero-Power BOR disabled in Deep Sleep
   #pragma config RTCOSC = INTOSCREF   // RTCC uses INTOSC as clock
   #pragma config DSWDTOSC = INTOSCREF // DSWDT uses INTOSC as clock

   // CONFIG3H
   #pragma config MSSP7B_EN = MSK7     // 7 Bit address masking
   #pragma config ADCSEL = BIT12       // 12-bit conversion mode is enabled
   #pragma config IOL1WAY = OFF        // IOLOCK bit can be set and cleared

   // CONFIG4L
   #pragma config WPCFG = OFF          // Write/Erase last page protect Disabled
   #pragma config WPFP = PAGE_0        // Write Protect Program Flash Page 0

   // CONFIG4H
   #pragma config LS48MHZ = SYS24X4// USB Low-speed clock at 24 MHz, USB clock divider is
set to 4
   #pragma config WPEND = PAGE_0       // Start protection at page 0
   #pragma config WPDIS = OFF          // WPFP[5:0], WPEND, and WPCFG bits ignored

/*
   static __code char __at __CONFIG1L _c1l = 0x8C;
   static __code char __at __CONFIG1H _c1h = 0x03;
   static __code char __at __CONFIG2L _c2l = 0x1A;
   static __code char __at __CONFIG2H _c2h = 0x08;
   static __code char __at __CONFIG3L _c3l = 0x65;
   static __code char __at __CONFIG3H _c3h = 0x08;
   static __code char __at __CONFIG4L _c4l = 0x80;
   static __code char __at __CONFIG4H _c4h = 0x0B; //0xF1;
*/

/*****************************************************************************/
#else
/*****************************************************************************/
```

```
   #error "    ------------------------------    "
   #error "    NO CONFIG. WORDS FOR YOUR CHIP.    "
   #error "    ------------------------------    "

#endif
```

3.1.4 "vectors.h"

El archivo "vectors.h" contiene las definiciones de las interrupciones que están definidas en el archivo "vectors.c", que son las de alta y baja prioridad.

```
/*******************************************************************************
        Title:     USB Pinguino Bootloader
        File:      vectors.h
        Descr.: move interrupt vectors
        Author:  RÃ©gis Blanchot <rblanchot@gmail.com>

        This file is part of Pinguino (http://www.pinguino.cc)
        Released under the LGPL license (http://www.gnu.org/licenses/lgpl.html)
*******************************************************************************/

#ifndef _VECTORS_H
#define _VECTORS_H

//void reset_isr(void) __naked __interrupt 0;
void high_priority_isr(void) __naked __interrupt 1;
void low_priority_isr(void) __naked __interrupt 2;

#endif
```

3.1.5 "pic18fregs.h"

En el archivo "pic18fregs.h" únicamente se hace un llamado al archivo de cabecera del microcontrolador que ha sido seleccionado.

3.1.6 "picUSB.h"

El archivo "picUSB.h" contiene las definiciones de variables, estructuras de datos, y de funciones del puerto USB que están definidas en el archivo "picUSB .c", que se utiliza como medio para descargar el nuevo firmware del microcontrolador.

```
// Firmware framework for USB I/O on PIC 18F2455 (and siblings)
// Copyright (C) 2005 Alexander Enzmann
// 2012 - modified for Pinguino by r.blanchot & a.gentric
//
```

```
// This library is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
//

#ifndef _PICUSB_H
#define _PICUSB_H

#include <<pic18fregs.h>
#include "types.h"

/**
        macros
**/

#define PTR16(x) ((unsigned int)(((unsigned long)x) & 0xFFFF))

/**
        definition for the initial ep0
        most are defined here since they are used by the user in the usb_config.h file
   NB: a transaction is limited in size to 8 bytes for low-speed devices
   * TODO
   * change uploader8.py to manage 8-bit transaction
**/

#if SPEED == LOW_SPEED
   #define MAX_PACKET_SIZE      8
#else
   #define MAX_PACKET_SIZE      64
#endif

#define EP0_BUFFER_SIZE             MAX_PACKET_SIZE
#define EP1_BUFFER_SIZE             MAX_PACKET_SIZE

// USB direction
#define OUT                     0
#define IN                      1

// How many different configuration should be available.
// At least configuration 1 must exist.
#define USB_MAX_CONFIGURATION    1

/* UEPn Initialization Parameters */
#define EP_CTRL                         0x06 // Cfg Control pipe for this ep
#define EP_OUT                  0x0C // Cfg OUT only pipe for this ep
#define EP_IN                   0x0A // Cfg IN only pipe for this ep
#define EP_OUT_IN               0x0E // Cfg both OUT & IN pipes for this ep
#define HSHK_EN                         0x10 // Enable handshake packet
                                 // Handshake should be disable for isoch
//
```

```
// Standard Request Codes USB 2.0 Spec Ref Table 9-4
//
#define GET_STATUS              0
#define CLEAR_FEATURE                   1
#define SET_FEATURE             3
#define SET_ADDRESS             5
#define GET_DESCRIPTOR                  6
#define SET_DESCRIPTOR                  7
#define GET_CONFIGURATION 8
#define SET_CONFIGURATION  9
#define GET_INTERFACE                   10
#define SET_INTERFACE                   11
#define SYNCH_FRAME             12

// Descriptor Types
#define DEVICE_DESCRIPTOR               0x01
#define CONFIGURATION_DESCRIPTOR                0x02
#define STRING_DESCRIPTOR               0x03
#define INTERFACE_DESCRIPTOR                    0x04
#define ENDPOINT_DESCRIPTOR                     0x05
#define DEVICE_QUALIFIER_DESCRIPTOR     0x06

// Standard Feature Selectors
#define DEVICE_REMOTE_WAKEUP                    0x01
#define ENDPOINT_HALT                           0x00

// Buffer Descriptor bit masks (from PIC datasheet)
#define BDS_COWN                0x00 // CPU Own Bit
#define BDS_UOWN                0x80 // USB Own Bit
#define BDS_DTS                         0x40 // Data Toggle Synchronization Bit
#define BDS_KEN                         0x20 // BD Keep Enable Bit
#define BDS_INCDIS              0x10 // Address Increment Disable Bit
#define BDS_DTSEN               0x08 // Data Toggle Synchronization Enable Bit
#define BDS_BSTALL              0x04 // Buffer Stall Enable Bit
#define BDS_BC9                         0x02 // Byte count bit 9
#define BDS_BC8                         0x01 // Byte count bit 8

// Device states (Chap 9.1.1)
#define DETACHED                0
#define ATTACHED                1
#define POWERED                         2
#define DEFAULT                         3
#define ADDRESS                         4
#define CONFIGURED              5

#define DEVICE_DESCRIPTOR_SIZE                  0x12
#define CONFIG_HEADER_SIZE              0x09
#define HID_DESCRIPTOR_SIZE                     0x20
#define HID_HEADER_SIZE                         0x09

// Buffer Descriptor Status Register
typedef union
{
   unsigned char uc;
   struct{
      unsigned BC8:1;
      unsigned BC9:1;
      unsigned BSTALL:1;          /* Buffer Stall Enable */
      unsigned DTSEN:1;          /* Data Toggle Synch Enable */
      unsigned INCDIS:1;         /* Address Increment Disable */
```

```
        unsigned KEN:1;              /* BD Keep Enable */
        unsigned DTS:1;              /* Data Toggle Synch Value */
        unsigned UOWN:1;             /* USB Ownership */
    };
    struct{
        unsigned :2;
        unsigned PID0:1;             /* Packet Identifier, bit 0 */
        unsigned PID1:1;             /* Packet Identifier, bit 1 */
        unsigned PID2:1;             /* Packet Identifier, bit 2 */
        unsigned PID3:1;             /* Packet Identifier, bit 3 */
        unsigned :2;
    };
    struct{
        unsigned :2;
        unsigned PID:4;              /* Packet Identifier */
        unsigned :2;
    };
} BDStat;

// Buffer Descriptor Table
typedef union
{
    struct
    {
        BDStat Stat;                 /* Buffer Descriptor Status Register */
        unsigned char Cnt;           /* Number of bytes to send/sent/(that can be )received */
        unsigned char ADRL;          /* Buffer Address Low */
        unsigned char ADRH;          /* Buffer Address High */
    };
    struct
    {
        unsigned :8;
        unsigned :8;
        __data unsigned int *ADDR;      /* Buffer Address */
    };
} BufferDescriptorTable;


// USB Device Descriptor
typedef struct
{
    unsigned char bLength;           /* Size of the Descriptor in Bytes (18 bytes = 0x12) */
    unsigned char bDescriptorType;      /* Device Descriptor (0x01) */
    unsigned int  bcdUSB;            /* USB Specification Number which device complies to. */
    unsigned char bDeviceClass;         /* Class Code (Assigned by USB Org).
                                If equal to Zero, each interface specifies itâ€ ™s own class code.
                                If equal to 0xFF, the class code is vendor specified.
                                Otherwise field is valid Class Code.*/
    unsigned char bDeviceSubClass;      /* Subclass Code (Assigned by USB Org) */
    unsigned char bDeviceProtocol;      /* Protocol Code (Assigned by USB Org) */
    unsigned char bMaxPacketSize0;      /* Maximum Packet Size for Zero Endpoint.
                                Valid Sizes are 8, 16, 32, 64 */
    unsigned int  idVendor;          /* Vendor ID (Assigned by USB Org).
                                Microchip Vendor ID is 0x04D8 */
    unsigned int  idProduct;         /* Product ID (Assigned by Manufacturer) */
    unsigned int  bcdDevice;          /* Device Release Number */
    unsigned char iManufacturer;        /* Index of Manufacturer String Descriptor */
    unsigned char iProduct;          /* Index of Product String Descriptor */
    unsigned char iSerialNumber;        /* Index of Serial Number String Descriptor */
    unsigned char bNumConfigurations;   /* Number of Possible Configurations */
```

```
} USB_Device_Descriptor;


// Configuration Descriptors
typedef struct
{
   unsigned char bLength;             /* Size of Descriptor in Bytes */
   unsigned char bDescriptorType;      /* Configuration Descriptor (0x02) */
   unsigned int   wTotalLength;              /* Total length in bytes of data returned (Configuration
Descriptor + Interface Descriptor + n* Endpoint Descriptor */
   unsigned char bNumInterfaces;       /* Number of Interfaces */
   unsigned char bConfigurationValue;  /* Value to use as an argument to select this configuration */
   unsigned char iConfiguration;       /* Index of String Descriptor describing this configuration */
   unsigned char bmAttributes;         /* D7 Reserved, set to 1.(USB 1.0 Bus Powered).
                                 D6 Self Powered.
                                 D5 Remote Wakeup.
                                 D4..0 Reserved, set to 0.
                                 0b10000000*/
   unsigned char bMaxPower;           /* Maximum Power Consumption in 2mA units */
} USB_Configuration_Descriptor_Header;

/**
Every device request starts with an 8 byte setup packet (USB 2.0, chap 9.3)
with a standard layout.  The meaning of wValue and wIndex will
vary depending on the request type and specific request.
See also: http://www.beyondlogic.org/usbnutshell/usb6.htm
TODO: split this Array up to be more precise
TODO: use word instead of LSB/MSB bytes
**/
typedef union
{
 struct {
   byte bmRequestType; /* D7 Data Phase Transfer Direction
                          0 = Host to Device
                          1 = Device to Host
                         D6..5 Type
                          0 = Standard
                          1 = Class
                          2 = Vendor
                          3 = Reserved
                         D4..0 Recipient
                          0 = Device
                          1 = Interface
                          2 = Endpoint
                          3 = Other */
   byte bRequest;      // Specific request
   byte wValue0;       // LSB of wValue
   byte wValue1;       // MSB of wValue
   byte wIndex0;       // LSB of wIndex
   byte wIndex1;       // MSB of wIndex
   word wLength;       // Number of bytes to transfer if there's a data stage
 };
 struct {
   byte extra[EP0_BUFFER_SIZE];     // Fill out to same size as Endpoint 0 max buffer
 };
} setupPacketStruct;


// Interface Descriptors
typedef struct
```

```
{
   unsigned char bLength;             /* Size of Descriptor in Bytes (9 Bytes) */
   unsigned char bDescriptorType;     /* Interface Descriptor (0x04) */
   unsigned char bInterfaceNumber;    /* Number of Interface */
   unsigned char bAlternateSetting;   /* Value used to select alternative setting */
   unsigned char bNumEndpoints;       /* Number of Endpoints used for this interface */
   unsigned char bInterfaceClass;             /* Class Code (Assigned by USB Org) e.g. HID,
communications, mass storage etc.*/
   unsigned char bInterfaceSubClass;  /* Subclass Code (Assigned by USB Org) */
   unsigned char bInterfaceProtocol;  /* Protocol Code (Assigned by USB Org) */
   unsigned char iInterface;          /* Index of String Descriptor Describing this interface */
} USB_Interface_Descriptor;

// Endpoint Descriptor
typedef struct
{
   unsigned char bLength;             /* Size of Descriptor in Bytes (7 bytes) */
   unsigned char bDescriptorType;     /* Endpoint Descriptor (0x05) */
   unsigned char bEndpointAddress;            /* Endpoint Address<ul><li>Bits 0..3b Endpoint
Number.</li><li>Bits 4..6b Reserved. Set to Zero</li><li>Bits 7 Direction 0 = Out, 1 = In (Ignored
for Control Endpoints)</li></ul> */
   unsigned char bmAttributes;        /* Bits 0..1 Transfer Type
                                      * 00 = Control
                                      * 01 = Isochronous
                                      * 10 = Bulk
                                      * 11 = Interrupt
                                      Bits 2..7 are reserved.
                                      If Isochronous endpoint,
                                      Bits 3..2 = Synchronisation Type (Iso Mode)
                                      * 00 = No Synchonisation
                                      * 01 = Asynchronous
                                      * 10 = Adaptive
                                      * 11 = Synchronous
                                      Bits 5..4 = Usage Type (Iso Mode)
                                      * 00 = Data Endpoint
                                      * 01 = Feedback Endpoint
                                      * 10 = Explicit Feedback Data Endpoint
                                      * 11 = Reserved */
   unsigned int  wMaxPacketSize;      /* Maximum Packet Size this endpoint is capable of sending or
receiving */
   unsigned char bInterval;           /* Interval for polling endpoint data transfers. Value in frame
counts. Ignored for Bulk & Control Endpoints. Isochronous must equal 1 and field may range from 1
to 255 for interrupt endpoints. */
} USB_Endpoint_Descriptor;

/**
   Example Composite of Configuration Header and Interface Descriptors for an ACM Header
   See USB CDC 1.1 Page 15
   Define your own Configuration Descriptor here.
**/
typedef struct
{   USB_Configuration_Descriptor_Header  Header;
   USB_Interface_Descriptor DataInterface;
//    USB_Endpoint_Descriptor ep_config;  // unused
   USB_Endpoint_Descriptor ep_out;
   USB_Endpoint_Descriptor ep_in;
} USB_Configuration_Descriptor;

// Packet structure sent by the uploader
typedef union
```

```
{
   struct
   {
      unsigned char cmd;
      unsigned char len;
      unsigned char addrl;
      unsigned char addrh;
      unsigned char addru;
      unsigned char xdat[EP1_BUFFER_SIZE-5];
   };
   unsigned char buffer[ EP1_BUFFER_SIZE ];
} allcmd;

extern volatile setupPacketStruct SetupPacket;
extern volatile byte controlTransferBuffer[EP0_BUFFER_SIZE];
extern volatile allcmd bootCmd;

// inPtr/OutPtr are used to move data from user memory (RAM/ROM/EEPROM) buffers
// from/to USB dual port buffers.
extern byte *outPtr;       // Address of buffer to send to host
extern byte *inPtr;        // Address of buffer to receive data from host
extern unsigned int wCount; // Total # of bytes to move

// USB Descriptors
extern __code USB_Device_Descriptor device_descriptor;
extern __code USB_Configuration_Descriptor configuration_descriptor;
#if (STRING == 1)
extern const char * const string_descriptor[];
#endif

// Global variables
extern byte deviceState;    // Visible device states (from USB 2.0, chap 9.1.1)
extern byte selfPowered;
extern byte currentConfiguration;

///
/// ep_bdt[4] changed to ep_bdt[32] par AndrÃ©, le 04-07-2012
///

#if   defined(__18f14k50) || defined(__18f14k50)  // Bank 2
extern volatile BufferDescriptorTable __at (0x200) ep_bdt[32];
#elif defined(__18f26j53) || defined(__18f46j53) || \
     defined(__18f27j53) || defined(__18f47j53)  // Bank 13
extern volatile BufferDescriptorTable __at (0xD00) ep_bdt[32];
#else                                    // Bank 4
extern volatile BufferDescriptorTable __at (0x400) ep_bdt[32];
#endif

// Out buffer descriptor of endpoint ep
#define EP_OUT_BD(ep) (ep_bdt[(ep << 1)])
// In buffer descriptor of endpoint ep
#define EP_IN_BD(ep)  (ep_bdt[(ep << 1) + 1])

// USB Functions
void EnableUSBModule(void);
void ProcessUSBTransactions(void);

#endif //_PICUSB_H
```

4.2. Archivos Fuente (.c)

Los archivos fuente son un conjunto líneas de texto, las cuales son las instrucciones que se ejecutan luego de ser compiladas y programadas, este no es ejecutado directamente, sino que debe ser traducido a otro lenguaje (ej. Lenguaje de maquina), que podrá ser ejecutado por el microcontrolador.

Este proyecto está compuesto por 3 archivos fuente: "picUSB.c", "vectors.c" y "main.c", que serán detallados a continuación.

3.2.1 "picUSB.c"

En el archivo "picUSB.c", se establece la dirección de memoria donde se ubicara el "BufferDescriptorTable", que para este microcontrolador sera la dirección 0x400.
Ademas tenemos la funciones para controlar el puerto USB, para estos microcontroladores, entre las cuales tenemos algunas como EnableUSBModule(), ProcessUSBTransactions() que habilitan el puerto y administran la comunicación USB entre la PC y el microcontrolador, estas dos funciones son globales, las demás son locales.

```
// Firmware framework for USB I/O on PIC 18F2455 (and siblings)
// Copyright (C) 2005 Alexander Enzmann
//
// This library is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
//

#include <pic18fregs.h>
#include "hardware.h"
#include "picUSB.h"
#include "types.h"

// Device and configuration descriptors.  These are used as the
// host enumerates the device and discovers what class of device
// it is and what interfaces it supports.
// TODO: remove below lines and replace with the apropriate device_desc.blength etc.
extern void usb_configure_endpoints();
//extern void usb_ep_data_out_callback(char end_point, word buffer_addr, unsigned char
```

```
byte_count);
extern void usb_ep_data_out_callback(char end_point);

// Global variables
byte deviceState;
byte selfPowered;
//byte remoteWakeup;
byte currentConfiguration;

byte deviceAddress;

// Control Transfer Stages - see USB spec chapter 5
#define SETUP_STAGE    0    // Start of a control transfer (followed by 0 or more data stages)
#define DATA_OUT_STAGE 1    // Data from host to device
#define DATA_IN_STAGE  2    // Data from device to host
#define STATUS_STAGE   3    // Unused - if data I/O went ok, then back to Setup

byte ctrlTransferStage;     // Holds the current stage in a control transfer

byte requestHandled;        // Set to 1 if request was understood and processed.

byte *outPtr;               // Data to send to the host
byte *inPtr;                // Data from the host
word wCount;                // Number of bytes of data


/**
   Buffer descriptors Table (see datasheet page 171)
   A RAM Bank (2, 4 or 13 depending on MCU) is used specifically for
   endpoint buffer control in a structure known as
   Buffer Descriptor Table (BDTZ).
   TODO: find something smarter to allocate the buffer,
   like in usbmmap.c of the microchip firmware.
   If not all endpoints are used the space in RAM is wasted.
**/

///
/// 2012-07-04 ep_bdt[4] -> ep_bdt[32] updated by AndrÃ©
///

#if   defined(__18f14k50) || defined(__18f14k50)  // Bank 2
   volatile BufferDescriptorTable __at (0x200) ep_bdt[32];
#elif defined(__18f26j53) || defined(__18f46j53) || \
     defined(__18f27j53) || defined(__18f47j53)  // Bank 13
   volatile BufferDescriptorTable __at (0xD00) ep_bdt[32];
#else                                  // Bank 4
   volatile BufferDescriptorTable __at (0x400) ep_bdt[32];
#endif

#pragma udata usbram5 SetupPacket controlTransferBuffer
volatile setupPacketStruct SetupPacket;
volatile byte controlTransferBuffer[EP0_BUFFER_SIZE];

volatile allcmd bootCmd;

//
// Start of code to process standard requests (USB chapter 9)
//

// Process GET_DESCRIPTOR
```

```
static void GetDescriptor()
{
	if(SetupPacket.bmRequestType == 0x80)
	{
		//byte descriptorType  = SetupPacket.wValue1;
		//byte descriptorIndex = SetupPacket.wValue0;

		if (SetupPacket.wValue1 == DEVICE_DESCRIPTOR)
		{
			requestHandled = 1;
			outPtr = (byte *)&device_descriptor;
			wCount = sizeof(USB_Device_Descriptor);
		}

		else if (SetupPacket.wValue1 == CONFIGURATION_DESCRIPTOR)
		{
			requestHandled = 1;
			outPtr = (byte *)&configuration_descriptor;
			wCount = configuration_descriptor.Header.wTotalLength;
		}
#if (STRING == 1)
		else if (SetupPacket.wValue1 == STRING_DESCRIPTOR)
		{
			requestHandled = 1;
			//outPtr = (byte *)&string_descriptor[SetupPacket.wValue0];
			outPtr = string_descriptor[SetupPacket.wValue0];
			wCount = *outPtr;
		}
#endif
/*
		else if (SetupPacket.wValue1 == DEVICE_QUALIFIER_DESCRIPTOR)
		{
			requestHandled = 1;
			// TODO: check if this is needed if not requestHandled is not set to 1 the
device will
			// stall later when the linux kernel requests this descriptor
			//outPtr = (byte *)&configuration_descriptor;
			//wCount = sizeof();
		}
*/
	}
}

void ProcessStandardRequest()
{
	//byte request = SetupPacket.bRequest;

	if((SetupPacket.bmRequestType & 0x60) != 0x00)
	// Not a standard request - don't process here.  Class or Vendor
	// requests have to be handled seperately.
	return;

	if (SetupPacket.bRequest == SET_ADDRESS)
	{
		// Set the address of the device.  All future requests
		// will come to that address.  Can't actually set UADDR
		// to the new address yet because the rest of the SET_ADDRESS
		// transaction uses address 0.
		requestHandled = 1;
		deviceState = ADDRESS;
```

```
            deviceAddress = SetupPacket.wValue0;
    }

    else if (SetupPacket.bRequest == GET_DESCRIPTOR)
    {
            GetDescriptor();
    }

    else if (SetupPacket.bRequest == SET_CONFIGURATION)
    {
            requestHandled = 1;
            usb_configure_endpoints();
            currentConfiguration = SetupPacket.wValue0;
            // TBD: ensure the new configuration value is one that
            // exists in the descriptor.
            if (currentConfiguration == 0)
                    // If configuration value is zero, device is put in
                    // address state (USB 2.0 - 9.4.7)
                    deviceState = ADDRESS;
            else
                    // Set the configuration.
                    deviceState = CONFIGURED;

            // Initialize the endpoints for all interfaces
            // TBD: Add initialization code here for any additional
            // interfaces beyond the one used for the HID
    }

    else if (SetupPacket.bRequest == GET_CONFIGURATION)
    {
            requestHandled = 1;
            outPtr = (byte*)&currentConfiguration;
            wCount = 1;
    }

    /*
    else if (SetupPacket.bRequest == GET_STATUS)
    {
    //    GetStatus();
    }

    else if ((SetupPacket.bRequest == CLEAR_FEATURE) || (request == SET_FEATURE))
    {
    //    SetFeature();
    }
    */

    else if (SetupPacket.bRequest == GET_INTERFACE)
    {
            // No support for alternate interfaces.  Send
            // zero back to the host.
            requestHandled = 1;
            controlTransferBuffer[0] = 0;
            outPtr = (byte *)&controlTransferBuffer;
            wCount = 1;
    }

    else if (SetupPacket.bRequest == SET_INTERFACE)
    {
            // No support for alternate interfaces - just ignore.
```

```
                requestHandled = 1;
        }

        /*
        else if (SetupPacket.bRequest == SET_DESCRIPTOR) {
        }
        else if (SetupPacket.bRequest == SYNCH_FRAME) {
        }
        else {
        }
        */
}


 /**
  Data stage for a Control Transfer that sends data to the host
 **/

void InDataStage(unsigned char ep)
{
        byte i;
        word bufferSize;

        // Determine how many bytes are going to the host
        if(wCount < EP0_BUFFER_SIZE)
                bufferSize = wCount;
        else
                bufferSize = EP0_BUFFER_SIZE;

        // Load the high two bits of the byte count into BC8:BC9
        // Clear BC8 and BC9
        EP_IN_BD(ep).Stat.uc &= ~(BDS_BC8 | BDS_BC9);
        EP_IN_BD(ep).Stat.uc |= (byte)((bufferSize & 0x0300) >> 8);
        EP_IN_BD(ep).Cnt = (byte)(bufferSize & 0xFF);
        EP_IN_BD(ep).ADDR = PTR16(&controlTransferBuffer);

        // Update the number of bytes that still need to be sent.  Getting
        // all the data back to the host can take multiple transactions, so
        // we need to track how far along we are.
        wCount = wCount - bufferSize;

        // Move data to the USB output buffer from wherever it sits now.
        inPtr = (byte *)&controlTransferBuffer;

        for (i=0;i<bufferSize;i++)
                *inPtr++ = *outPtr++;
}

 /**
  Data stage for a Control Transfer that reads data from the host
 **/

void OutDataStage(unsigned char ep)
{
        word i, bufferSize;

        bufferSize = ((0x03 & EP_OUT_BD(ep).Stat.uc) << 8) | EP_OUT_BD(ep).Cnt;

        // Accumulate total number of bytes read
        wCount = wCount + bufferSize;
```

```
        outPtr = (byte*)&controlTransferBuffer;

        for (i=0;i<bufferSize;i++)
                *inPtr++ = *outPtr++;
}

 /**
   Process the Setup stage of a control transfer.  This code initializes the
   flags that let the firmware know what to do during subsequent stages of
   the transfer.
   TODO:
   Only Ep0 is handled here.
 **/

void SetupStage()
{
        // Note: Microchip says to turn off the UOWN bit on the IN direction as
        // soon as possible after detecting that a SETUP has been received.
        EP_IN_BD(0).Stat.uc &= ~BDS_UOWN;
        EP_OUT_BD(0).Stat.uc &= ~BDS_UOWN;

        // Initialize the transfer process
        ctrlTransferStage = SETUP_STAGE;
        requestHandled = 0;             // Default is that request hasn't been handled
        wCount = 0;                     // No bytes transferred

        // See if this is a standard (as definded in USB chapter 9) request
        ProcessStandardRequest();

        // TBD: Add handlers for any other classes/interfaces in the device
        if (!requestHandled)
        {
                // If this service wasn't handled then stall endpoint 0
                EP_OUT_BD(0).Cnt = EP0_BUFFER_SIZE;
                EP_OUT_BD(0).ADDR = PTR16(&SetupPacket);
                EP_OUT_BD(0).Stat.uc = BDS_UOWN | BDS_BSTALL;
                EP_IN_BD(0).Stat.uc = BDS_UOWN | BDS_BSTALL;
        }

        else if (SetupPacket.bmRequestType & 0x80)
        {
                // Device-to-host
                if(SetupPacket.wLength < wCount)
                        wCount = SetupPacket.wLength;

                InDataStage(0);
                ctrlTransferStage = DATA_IN_STAGE;
                // Reset the out buffer descriptor for endpoint 0
                EP_OUT_BD(0).Cnt = EP0_BUFFER_SIZE;
                EP_OUT_BD(0).ADDR = PTR16(&SetupPacket);
                EP_OUT_BD(0).Stat.uc = BDS_UOWN;

                // Set the in buffer descriptor on endpoint 0 to send data
                EP_IN_BD(0).ADDR = PTR16(&controlTransferBuffer);
                // Give to SIE, DATA1 packet, enable data toggle checks
                EP_IN_BD(0).Stat.uc = BDS_UOWN | BDS_DTS | BDS_DTSEN;
        }

        else
```

```
        {
                // Host-to-device
                ctrlTransferStage = DATA_OUT_STAGE;

                // Clear the input buffer descriptor
                EP_IN_BD(0).Cnt = 0;
                EP_IN_BD(0).Stat.uc = BDS_UOWN | BDS_DTS | BDS_DTSEN;

                // Set the out buffer descriptor on endpoint 0 to receive data
                EP_OUT_BD(0).Cnt = EP0_BUFFER_SIZE;
                EP_OUT_BD(0).ADDR = PTR16(&controlTransferBuffer);
                // Give to SIE, DATA1 packet, enable data toggle checks
                EP_OUT_BD(0).Stat.uc = BDS_UOWN | BDS_DTS | BDS_DTSEN;
        }

        // Enable SIE token and packet processing
        UCONbits.PKTDIS = 0;
}


 // Configures the buffer descriptor for endpoint 0 so that it is waiting for
 // the status stage of a control transfer.
void WaitForSetupStage()
{
        ctrlTransferStage = SETUP_STAGE;
        EP_OUT_BD(0).Cnt = EP0_BUFFER_SIZE;
        EP_OUT_BD(0).ADDR = PTR16(&SetupPacket);
        // Give to SIE, enable data toggle checks
        EP_OUT_BD(0).Stat.uc = BDS_UOWN | BDS_DTSEN;
        EP_IN_BD(0).Stat.uc = 0x00;         // Give control to CPU
}

// This is the starting point for processing a Control Transfer.  The code directly
// follows the sequence of transactions described in the USB spec chapter 5.  The
// only Control Pipe in this firmware is the Default Control Pipe (endpoint 0).
// Control messages that have a different destination will be discarded.
void ProcessControlTransfer()
{
   // get encoded number of the last active Endpoint
        byte PID, end_point = USTAT >> 3;

        if (end_point == 0) // Endpoint 0
        {

                if (USTATbits.DIR == OUT)
                {
                        // Endpoint 0:out
                        // Pull PID from middle of BD0STAT
                        PID = (EP_OUT_BD(0).Stat.uc & 0x3C) >> 2;
                        if (PID == 0x0D)
                                // SETUP PID - a transaction is starting
                                SetupStage();

                        else if (ctrlTransferStage == DATA_OUT_STAGE)
                        {
                                // Complete the data stage so that all information has
                                // passed from host to device before servicing it.
                                OutDataStage(0);

                                // Turn control over to the SIE and toggle the data bit
```

```
                                        if(EP_OUT_BD(0).Stat.DTS)
                                                EP_OUT_BD(0).Stat.uc = BDS_UOWN | BDS_DTSEN;
                                        else
                                                EP_OUT_BD(0).Stat.uc = BDS_UOWN | BDS_DTS |
BDS_DTSEN;
                                }

                                else
                                {
                                        // Prepare for the Setup stage of a control transfer
                                        WaitForSetupStage();
                                }
                        }

                        else // if(USTATbits.DIR == IN)
                        {
                                // Endpoint 0:in
                                if ((UADDR == 0) && (deviceState == ADDRESS))
                                {
                                        // TBD: ensure that the new address matches the value of
                                        // "deviceAddress" (which came in through a SET_ADDRESS).
                                        UADDR = SetupPacket.wValue0;
                                        if(UADDR == 0)
                                                // If we get a reset after a SET_ADDRESS, then we need
                                                // to drop back to the Default state.
                                                deviceState = DEFAULT;
                                }

                                if (ctrlTransferStage == DATA_IN_STAGE)
                                {
                                        // Start (or continue) transmitting data
                                        InDataStage(0);

                                        // Turn control over to the SIE and toggle the data bit
                                        if(EP_IN_BD(0).Stat.DTS)
                                                EP_IN_BD(0).Stat.uc = BDS_UOWN | BDS_DTSEN;
                                        else
                                                EP_IN_BD(0).Stat.uc = BDS_UOWN | BDS_DTS |
BDS_DTSEN;
                                }

                                else
                                {
                                        // Prepare for the Setup stage of a control transfer
                                        WaitForSetupStage();
                                }

                        }
                }

                else //if (end_point == 1) // EndPoint 1
                {

                        if (!USTATbits.DIR) // If OUT
                                usb_ep_data_out_callback(end_point);
                }
}

void EnableUSBModule()
{
```

```
        // TBD: Check for voltage coming from the USB cable and use that
        // as an indication we are attached.
        if(UCONbits.USBEN == 0)
        {
                UCON = 0;               // USB Control Register
                UIE = 0;              // Disable USB Interrupt Register
                UCONbits.USBEN = 1;     // Enable USB module
                deviceState = ATTACHED;
        }
        // If we are attached and no single-ended zero is detected, then
        // we can move to the Powered state.
        if ((deviceState == ATTACHED) && !UCONbits.SE0)
        {
                UIR = 0;
                UIE = 0;              // Disable USB Interrupt Register
                UIEbits.URSTIE = 1;     // Enable USB Reset Interrupt
                UIEbits.IDLEIE = 1;     // Enable IDle Detect USB Interrupt
                deviceState = POWERED;
        }
}

// Main entry point for USB tasks.  Checks interrupts, then checks for transactions.
void ProcessUSBTransactions()
{
        // See if the device is connected yet.
        if(deviceState == DETACHED)
                return;

        // If the USB became active then wake up from suspend
        if(UIRbits.ACTVIF && UIEbits.ACTVIE)
  {
                // UnSuspend
    UCONbits.SUSPND = 0;
    UIEbits.ACTVIE = 0;
    UIRbits.ACTVIF = 0;
  }

        // If we are supposed to be suspended, then don't try performing any processing.
        if(UCONbits.SUSPND == 1)
                return;

        // Process a bus reset
        if (UIRbits.URSTIF && UIEbits.URSTIE)
  {
    UEIR  = 0x00;
    UIR   = 0x00;
    UEIE  = 0x9f;
    UIE   = 0x7b;
    UADDR = 0x00;

    // Set endpoint 0 as a control pipe
    UEP0 = EP_CTRL | HSHK_EN;

    // Flush any pending transactions
    while (UIRbits.TRNIF == 1)
      UIRbits.TRNIF = 0;

    // Enable packet processing
    UCONbits.PKTDIS = 0;
```

```
    // Prepare for the Setup stage of a control transfer
    WaitForSetupStage();

    //  remoteWakeup = 0;           // Remote wakeup is off by default
    selfPowered = 0;              // Self powered is off by default
    currentConfiguration = 0;     // Clear active configuration
    deviceState = DEFAULT;
  }

        /*
        if (UIRbits.IDLEIF && UIEbits.IDLEIE) {
        // No bus activity for a while - suspend the firmware
        Suspend();
        }
        if (UIRbits.SOFIF && UIEbits.SOFIE) {
        StartOfFrame();
        }
        if (UIRbits.STALLIF && UIEbits.STALLIE) {
        Stall();
        }
        */

        // TBD: See where the error came from.
        if (UIRbits.UERRIF && UIEbits.UERRIE)
                UIRbits.UERRIF = 0;// Clear errors

        // Unless we have been reset by the host, no need to keep processing
        if (deviceState < DEFAULT)  // DETACHED, ATTACHED or POWERED
                return;

        // A transaction has finished.  Try default processing on endpoint 0.
        if(UIRbits.TRNIF && UIEbits.TRNIE)
        {
                ProcessControlTransfer();
                // Turn off interrupt
                UIRbits.TRNIF = 0;
        }
}
```

3.2.2 “vectors.c”

El archivo “vectors.c” contiene las configuraciones de los vectores de interrupción.

```
/*********************************************************************************
        Title:    USB Pinguino Bootloader
        File:     vectors.c
        Descr.: move interrupt vectors
        Author:  RÃ©gis Blanchot <rblanchot@gmail.com>

        This file is part of Pinguino (http://www.pinguino.cc)
        Released under the LGPL license (http://www.gnu.org/licenses/lgpl.html)
*********************************************************************************/

// Never use --ivt-loc=$(ENTRY) to do the job
// as it will also move the Reset vector from 0 to ENTRY

/*
extern void main(void);
```

```
// 0x0000
void reset_isr(void) __naked __interrupt 0
{
   main();
}
*/

// 0x0008
void high_priority_isr(void) __naked __interrupt 1
{
   __asm
   goto   ENTRY + 0x08
   __endasm;
}

// 0x0018
void low_priority_isr(void) __naked __interrupt 2
{
   __asm
   goto   ENTRY + 0x18
   __endasm;
}
```

3.2.3 “main.c”

El archivo “main.c” contiene los descriptores y endpoint que serán utilizados para la comunicación USB, en este caso solo utiliza un endpoint(0).

En el bucle principal, al arrancar se configuran todos los puertos como digitales, luego de esto se habilitan las prioridades en interrupciones, luego se habilita la interrupción del timer 1 y se deshabilita la lectura y escritura tanto de la memoria EEPROM como de la Flash.

Para la comunicación USB es necesario la habilitación de resistencias de pull-up, que serán de acuerdo a la velocidad del bus.

Terminadas las configuraciones, el sistema entra a un bucle esperando hasta que el dispositivo se encuentre configurado, en caso de que el cable USB no se encuentre conectado, el bucle actúa de manera no bloqueante

```
/********************************************************************************
        Title:     USB Pinguino Bootloader
        File:      hardware.h
        Descr.: bootloader def. (version, led, tempo.)
        Author:  André Gentric
                        Régis Blanchot <rblanchot@gmail.com>
                        Based on Albert Faber's JAL bootloader
                        and Alexander Enzmann's USB Framework
        This file is part of Pinguino (http://www.pinguino.cc)
        Released under the LGPL license (http://www.gnu.org/licenses/lgpl.html)
*********************************************************************************/

//#pragma stack 0x300 255                   // Initializes stack of 255 bytes at RAM address 0x300

//#include <pic18fregs.h>
#include <pic18fregs.h>
#include "types.h"
#include "hardware.h"
#include "picUSB.h"
#include "config.h"
#include "vectors.h"

__code USB_Device_Descriptor device_descriptor =
{
   sizeof(USB_Device_Descriptor),            // Size of this descriptor in bytes
   DEVICE_DESCRIPTOR,                        // Device descriptor type
   0x0200,                                   // USB Spec Release Number in BCD format (0x0100 for USB 1.0,
0x0110 for USB1.1, 0x0200 for USB2.0)
   0xff,                                     // Class Code-->00
   0x00,                                     // Subclass code
   0xff,                                     // Protocol code
   EP0_BUFFER_SIZE,                          // Max packet size for EP0
   0x04D8,                                   // Vendor ID, microchip=0x04D8, generic=0x05f9, test=0x067b
   0xFEAA,                                   // Product ID 0x00A für CDC, generic=0xffff, test=0x2303
   (MAJOR_VERSION<<8)+MINOR_VERSION,         // Device release number in BCD format-->0
   1,                                        // Manufacturer string index (0=no string descriptor)
   2,                                        // Product string index (0=no string descriptor)
   0,                                        // Device serial number string index
   1                                         // Number of possible configurations
};

__code USB_Configuration_Descriptor configuration_descriptor =
{
   // Configuration Descriptor Header
   {sizeof(USB_Configuration_Descriptor_Header),// Size of this descriptor in bytes
   CONFIGURATION_DESCRIPTOR,                 // CONFIGURATION descriptor type
   sizeof(USB_Configuration_Descriptor),     // Total length of data for this configuration
   1,                                        // Number of interfaces in this configuration
   1,                                        // Index value of this configuration
   0,                                        // Configuration string index
   192,//DEFAULT | POWERED,                  // Attributes
   20},                                      // Maximum Power Consumption in 2mA units
   // Data Interface Descriptor with in and out EPs
   {sizeof(USB_Interface_Descriptor),        // Size of this descriptor in bytes
   INTERFACE_DESCRIPTOR,                     // Interface descriptor type
   0,                                        // Interface Number
   0,                                        // Alternate Setting Number
   2,                                        // Number of endpoints in this interface
   0xff,                                     // Class code
```

```
    0xff,                           // TODO: Subclass code
    0xff,                           // TODO: Protocol code
    0},                             // Index of String Descriptor Describing this interface-->2
    // Endpoint 1 Out
    {sizeof(USB_Endpoint_Descriptor),       // Size of Descriptor
    ENDPOINT_DESCRIPTOR,                    // Descriptor Type
    0x01,                           // Endpoint Address
    0x02,                           // Attribute = Bulk Transfer
    EP1_BUFFER_SIZE,                        // Packet Size
    0x00},                          // Poll Intervall
    // Endpoint 1 IN
    {sizeof(USB_Endpoint_Descriptor),       // Size of Descriptor
    ENDPOINT_DESCRIPTOR,                    // Descriptor Type
    0x81,                           // Endpoint Address
    0x02,                           // Attribute = Bulk Transfer
    EP1_BUFFER_SIZE,                        // Packet Size
    0x00}
};

#if (STRING == 1)
    const char lang[] = {sizeof(lang),  STRING_DESCRIPTOR,
        0x09,0x04}; // english = 0x0409
    const char manu[] = {sizeof(manu),  STRING_DESCRIPTOR,
        'R',0x00,'.',0x00,'B',0x00,'l',0x00,'a',0x00,'n',0x00,'c',0x00,'h',0x00,'o',0x00,'t',0x00,
//        '/',0x00,
//        'A',0x00,'.',0x00,'G',0x00,'e',0x00,'n',0x00,'t',0x00,'r',0x00,'i',0x00,'c',0x00
        };
    const char prod[] = {sizeof(prod),  STRING_DESCRIPTOR,
        'P',0x00,'i',0x00,'n',0x00,'g',0x00,'u',0x00,'i',0x00,'n',0x00,'o',0x00};
    const char * const string_descriptor[] = { lang, manu, prod};
#endif

/*  -------------------------------------------------------------------
    -------------------------------------------------------------------*/

void delay(void) //__naked
{
    word i;//=0xffff;
    while(i--);
/*
    __asm
        movlw       0xFF
        movwf       r0x00
        movlw       0xFF
        movwf       r0x01
    startup_loop:
        decfsz      r0x00, f
        bra startup_loop
        decfsz      r0x01, f
        bra startup_loop
    __endasm;
*/
}

/*  -------------------------------------------------------------------
    -------------------------------------------------------------------*/

void start_write(void) //__naked
{
    __asm
```

```
   #if defined(__18f2455)  || defined(__18f4455) || \
      defined(__18f2550)  || defined(__18f4550) || \
      defined(__18f25k50) || defined(__18f45k50)

      movlw   0x55
      movwf   _EECON2        ; EECON2 = 0x55;
      movlw   0xAA
      movwf   _EECON2        ; EECON2 = 0xAA;
      bsf     _EECON1, 1     ; EECON1bits.WR = 1; start flash/eeprom writing
                     ; CPU stall here for 2ms
      nop                    ; proc. can forget to execute the first operation

   #elif defined(__18f26j50) || defined(__18f46j50) || \
        defined(__18f26j53) || defined(__18f46j53) || \
        defined(__18f27j53) || defined(__18f47j53) || \
        defined(__18f13k50) || defined(__18f14k50)

      ;bsf     _EECON1, 5     ; EECON1bits.WPROG = 1;      Program  2  bytes  on  the  next  WR
command
      bsf     _EECON1, 2     ; EECON1bits.WREN = 1; allows write cycles to Flash program memory
      movlw   0x55
      movwf   _EECON2        ; EECON2 = 0x55;
      movlw   0xAA
      movwf   _EECON2        ; EECON2 = 0xAA;
      bsf     _EECON1, 1     ; EECON1bits.WR = 1; start flash/eeprom writing
      bcf       _EECON1,  2        ; EECON1bits.WREN  =  0;  inhibits  write  cycles  to  Flash  program
memory

   #endif

   __endasm;
}

/*  ------------------------------------------------------------------
   ------------------------------------------------------------------*/

void disable_boot(void) __naked
{
   // disable timer 1
   T1CON = 0x00;
   // disable USB
   UCON = 0x00;
   __asm
      bsf     LED_TRIS, LED_PIN   ; led input
      bcf     LED_PORT, LED_PIN   ; led off
      ;call    _delay             ; force timeout on USB
   __endasm;
   delay();
}

/*  ------------------------------------------------------------------
           UEP1bits.EPHSHK   = 1;              // EP handshaking on
           UEP1bits.EPCONDIS = 1;                      // control transfers off
           UEP1bits.EPOUTEN  = 1;                      // EP OUT enabled
           UEP1bits.EPINEN   = 1;              // EP IN enabled
   ------------------------------------------------------------------*/

void usb_configure_endpoints()
{
```

```
   UEP1 = 0b00011110;

   // for IN
   // set DTS bit, turn on data togle sync TOGGLE
   EP_IN_BD(1).Stat.uc  = 0b01000000;

   // for OUT
   EP_OUT_BD(1).Cnt  = EP1_BUFFER_SIZE;
   EP_OUT_BD(1).ADDR = PTR16(&bootCmd);
   // set UOWN bit, SIE owns the buffer
   EP_OUT_BD(1).Stat.uc = 0b10000000;
}

/*  -----------------------------------------------------------------
   -----------------------------------------------------------------*/

void usb_ep_data_out_callback(char end_point)
{
        byte counter;

        // whatever the command, keep LED high
        __asm
        bsf             LED_PORT, LED_PIN    ; led on
        __endasm;

   // Number of byte(s) to return
        EP_IN_BD(end_point).Cnt = 0;

   // load table pointer
   TBLPTRU = bootCmd.addru;
   TBLPTRH = bootCmd.addrh;
   TBLPTRL = bootCmd.addrl;

/*************************************************************************/
   if (bootCmd.cmd ==  RESET)
/*************************************************************************/
        {
                disable_boot();
                __asm
                goto    ENTRY           ; start user app
                __endasm;
        }
/*************************************************************************/
        else if (bootCmd.cmd == READ_VERSION)
/*************************************************************************/
        {
                bootCmd.buffer[2] = MINOR_VERSION;
                bootCmd.buffer[3] = MAJOR_VERSION;
      // Number of byte(s) to return
                EP_IN_BD(end_point).Cnt = 4;
                //T1CON = 0b00000000; // disable timer 1 ???
        }

/*************************************************************************/
        else if (bootCmd.cmd == READ_FLASH)
/*************************************************************************/
        {
                for (counter=0; counter < bootCmd.len; counter++)
                {
                        __asm TBLRD*+ __endasm;
```

```
                        bootCmd.xdat[counter] = TABLAT;
                }

        // Number of byte(s) to return
                EP_IN_BD(end_point).Cnt = 5 + bootCmd.len;
        }

/*****************************************************************************/
        else if (bootCmd.cmd == WRITE_FLASH)
/*****************************************************************************/
        {
    #if defined(__18f2550)  || defined(__18f4550)  || \
       defined(__18f2455)  || defined(__18f4455)  || \
       defined(__18f25k50) || defined(__18f45k50) || \
       defined(__18f13k50) || defined(__18f14k50)

    /// The programming block is 32 bytes for all chips except x5k50
    /// The programming block is 64 bytes for x5k50.
    /// It is not necessary to load all holding register before a write opreration
    /// Word or byte programming is not supported by these chips.
    /// NB:
    /// * High Speed USB has a Max. packet size of 64 bytes
    /// * Uploader (uploader8.py) sends 32-byte Data block + 5-byte Command block

    //TBLPTRL = (TBLPTRL & 0xF0);   // Force 16-byte boundary
    //TBLPTRL = (TBLPTRL & 0xE0);   // Force 32-byte boundary
    // Load max. 32 holding registers
                for (counter=0; counter < bootCmd.len; counter++)
                {
                        TABLAT = bootCmd.xdat[counter];     // present data to table latch
                        __asm TBLWT*+ __endasm;                 // write data in TBLWT holding
register
    }
    __asm TBLRD*- __endasm;                 // to be inside the 32 bytes range
    // issue the block
                EECON1 = 0b10000100; // allows write (WREN=1) in flash (EEPGD=1)
                start_write();

    #elif defined(__18f26j50) || defined(__18f46j50) || \
         defined(__18f26j53) || defined(__18f46j53) || \
         defined(__18f27j53) || defined(__18f47j53)

    /// blocks must be erased before written
    /// the whole memory is erased at the begining of upload
    /// we write 32 bytes not 64 bytes
    /// n   : write [address]      + 64 bytes
    /// n+1 : write [address + 32] + 64 bytes
    /// 32 bytes are written 2 times
    /// that's why we use 2-byte write instead of 64-byte write

                EECON1bits.WPROG = 1; // Enable single-word write
                for (counter=0; counter < bootCmd.len; counter+=2)
                {
                        TBLPTRL =  bootCmd.addrl + counter;
                        TABLAT = bootCmd.xdat[counter];
      // TBLPTR is incremented after the write
                        __asm TBLWT*+ __endasm;
                        TABLAT = bootCmd.xdat[counter + 1];
      // TBLPTR is NOT incremented after the write
                        __asm TBLWT* __endasm;
```

```
                        start_write();
                }
                EECON1bits.WPROG = 0;           // Disable single-word write

    #endif

    // Number of byte(s) to return
                EP_IN_BD(end_point).Cnt = 1;
        }

/*****************************************************************************/
        else if (bootCmd.cmd ==  ERASE_FLASH)
/*****************************************************************************/
        {

    #if defined(__18f2550)  || defined(__18f4550)  || \
       defined(__18f2455)  || defined(__18f4455)  || \
       defined(__18f25k50) || defined(__18f45k50) || \
       defined(__18f14k50)

                // The erase block is 64 bytes
    // bootCmd.len = num. of 64-byte block to erase

                EECON1 = 0b10010100; // allows erase (WREN=1, FREE=1) in flash (EEPGD=1)
                for (counter=0; counter < bootCmd.len; counter++)
                {
       EECON1bits.FREE = 1;    // allow a program memory erase operation
                        start_write();
       EECON1bits.FREE = 0;    // inhibit program memory erase operation

       // next block (TBLPTR = TBLPTR + 64)
                        __asm
                        movlw   0x40            ; 0x40 + (TBLPTRL) -> TBLPTRL
                        addwf   _TBLPTRL, 1                     ;     (W)   +   (TBLPTRL)   ->
TBLPTRL
                       ;  (C) is affected
                        movlw   0x00                            ; 0x00 + (TBLPTRH) + (C) ->
TBLPTRH
                        addwfc  _TBLPTRH, 1                     ;   (W) + (TBLPTRH) + (C) ->
TBLPTRH
                        __endasm;
                }
                // TBLPTRU = 0

    #elif defined(__18f26j50) || defined(__18f46j50) || \
         defined(__18f26j53) || defined(__18f46j53) || \
         defined(__18f27j53) || defined(__18f47j53)

                // The erase block is 1024 bytes
    // bootCmd.len = num. of 1024-byte blocks to erase

                for (counter=0; counter < bootCmd.len; counter++)
                {
       EECON1bits.FREE = 1;    // allow a program memory erase operation
       start_write();
       EECON1bits.FREE = 0;    // inhibit program memory erase operation

       // next block (TBLPTR = TBLPTR + 1024)
                        __asm
                        movlw   0x04            ; 0x04 + (TBLPTRH) -> TBLPTRH
```

```
                        addwf   _TBLPTRH, 1                     ;     (W)  +  (TBLPTRH)  ->
TBLPTRH
                      ;  (C) is affected
                        movlw   0x00                            ; 0x00 + (TBLPTRU) + (C) ->
TBLPTRU
                        addwfc  _TBLPTRU, 1                     ;  (W) + (TBLPTRU) + (C) ->
TBLPTRU
                        __endasm;
                }

    #endif

    // Number of byte(s) to return
                EP_IN_BD(end_point).Cnt = 1;
        }

/*********************************************************************/
  // Is there something to return ?
        if (EP_IN_BD(end_point).Cnt > 0)
        {
    // Data packet toggle
                if (EP_IN_BD(1).Stat.DTS)
                        EP_IN_BD(1).Stat.uc = 0b10001000; // UOWN 1 DTS 0 DTSEN 1
                else
                        EP_IN_BD(1).Stat.uc = 0b11001000; // UOWN 1 DTS 1 DTSEN 1
        }

        // reset size
  EP_OUT_BD(end_point).Cnt = EP1_BUFFER_SIZE;
        // set to UOWN
  EP_OUT_BD(end_point).Stat.uc = 0x80;      // UOWN 1
}

/*  -------------------------------------------------------------
  Main loop
  -------------------------------------------------------------*/

void main(void) //__naked
{
  //dword i = 0;
  byte t1_count = 0;
  word led_counter = 0;

  __asm

  //bcf     _PIR2, 4          ; Clear USB Interrupt Flag

/*********************************************************************/
  #if defined(__18f14k50) || \
     defined(__18f2455)  || defined(__18f4455)  || \
     defined(__18f2550)  || defined(__18f4550)
/*********************************************************************/
     movlw   0x0F
     movwf   _ADCON1            ; all I/O to Digital mode
     movlw   0x07
     movwf   _CMCON             ; all I/O to Digital mode
/*********************************************************************/
  #elif defined(__18f25k50) || defined(__18f45k50)
/*********************************************************************/
     movlw   0x70               ; 0b01110000 : 111 = HFINTOSC (16 MHz)
```

```
      movwf   _OSCCON            ; enable the 16 MHz internal clock
   wait_hfintosc:
      btfss   _OSCCON, 2         ; HFIOFS: HFINTOSC Frequency Stable bit
      bra     wait_hfintosc      ; wait HFINTOSC frequency is stable (HFIOFS=1)
      clrf    _ANSELA            ; all I/O to Digital mode
      clrf    _ANSELB            ; all I/O to Digital mode
      clrf    _ANSELC            ; all I/O to Digital mode
      #if defined(__18f45k50)
      clrf    _ANSELD            ; all I/O to Digital mode
      clrf    _ANSELE            ; all I/O to Digital mode
      #endif
/*****************************************************************************/
   #elif defined(__18f26j50) || defined(__18f46j50)
/*****************************************************************************/
      bsf     _OSCTUNEbits, 6    ; Enable the PLL (PLLEN=bit6)
      ;call    _delay            ; Wait 2+ms until the PLL locks
                                 ; before enabling USB module
   __endasm;
      delay();
   __asm
      movlw   0xFF
      movwf   _ANCON0            ; all I/O to Digital mode
      movlw   0x1F
      movwf   _ANCON1            ; all I/O to Digital mode
/*****************************************************************************/
   #elif defined(__18f26j53) || defined(__18f46j53)|| \
        defined(__18f27j53) || defined(__18f47j53)
/*****************************************************************************/
      movlw   0x70               ; 0b01110000 : 111 = INTOSC (8 MHz)
      movwf   _OSCCON            ; enable the 8 MHz internal clock
   wait_intosc:
      btfss   _OSCCON, 2         ; FLTS: Frequency Lock Tuning Status bit
      bra     wait_intosc        ; wait INTOSC frequency is stable (FLTS=1)
;      banksel _ANCON0            ; ANCONx registers are not in access bank
      movlw   0xFF
      movwf   _ANCON0            ; all Analog pins to Digital mode
;      banksel _ANCON1            ; ANCONx registers are not in access bank
;      movlw   0x1F
      movwf   _ANCON1            ; all I/O to Digital mode
;      movlw   0x07
;      movwf   _ADCON0            ; all I/O to Digital mode
;      movlw   0x07
;      movwf   _CMCON             ; all I/O to Digital mode
/*****************************************************************************/
   #else
/*****************************************************************************/
      #error "    -------------------------    "
      #error "    PIC NO YET SUPPORTED !       "
      #error "    Please contact developers.   "
      #error "    -------------------------    "

   #endif
/*****************************************************************************/

      bsf     _RCON, 7           ; enable priority levels on interrupts

      bcf     LED_TRIS, LED_PIN   ; led output
      bsf     LED_PORT, LED_PIN   ; led on

      movlw    b'00110001'       ; prescaler 8 (0b11)
```

```
        movwf    _T1CON            ; timer 1 on,

        clrf    _EECON1           ; EECON1=0

        #if (SPEED == LOW_SPEED)
        movlw   b'00010000'       ; (0x10) Enable pullup resistors and low speed mode
        #else
        movlw   b'00010100'       ; (0x14) Enable pullup resistors and full speed mode
        #endif

        banksel _UCFG
        movwf   _UCFG, b

    __endasm;

    // Initialize USB

    EP_IN_BD(1).ADDR = PTR16(&bootCmd);
    currentConfiguration = 0x00;
    deviceState = DETACHED;

    // Non-blocking loop if USB cable is not connected (ext. supply)

    do {
        EnableUSBModule();
        ProcessUSBTransactions();
        //i = i + 1;
        //if (i == 0xFFFFF) break;
    } while (deviceState != CONFIGURED);

    // If no USB cable then start user app. now
/*
    if (deviceState != CONFIGURED)
    {
        t1_count = BOOT_TIMER_TICS;
        __asm
            bcf     LED_PORT, LED_PIN   ; led on
        __endasm;
    }
*/
    while (1)
    {
        //if (i!=0)
            ProcessUSBTransactions();

        // strobing LED
        if (led_counter == 0)
        {
            __asm
            movlw   LED_MASK    ; toggle
            xorwf   LED_PORT, f ; the led
            __endasm;
        }
        led_counter++;

        // timeout ?
        if (PIR1bits.TMR1IF == 1)
        {
            t1_count++;
            PIR1bits.TMR1IF = 0;
```

```
        // if expired, then jump to user location
        if (t1_count > BOOT_TIMER_TICS)
        {
           disable_boot();
           __asm
           goto    ENTRY   ; start user app
           __endasm;
        }
      }
   }
}
```

4.3. Makefile

Los archivos makefile, especifican como se realiza la compilación, su función principal es determinar que partes del programa son necesarias. La ventaja de este estilo de programación es que nos permite utilizar diferentes microcontroladores con diferentes configuraciones, y cristales de diferentes frecuencias.

```
#########################################################################
#                                       #
#       Pinguino Bootloader v4.x                        #
#       8-bit USB Bootloader                            #
#       Author:  Régis Blanchot <rblanchot@gmail.com>           #
#      André Gentric                            #
#                                       #
#       Usage: make --makefile=Makefile PROC=18f26j50 OSC=20 STRINGDESC=0 LVP=0
#      to compile the bootloader                        #
#                                       #
#       This file is part of Pinguino Project (http://www.pinguino.cc)  #
#       Released under the LGPL license (www.gnu.org/licenses/lgpl.html)  #
#                                       #
#########################################################################
#       Supported CPU's                                 #
#########################################################################
#                                       #
#  18f2455              18f4455                         #
#       18f2550         18f4550                         #
#       18f25k50        18f45k50                         #
#       18f26j50        18f46j50                        #
#       18F26J53        18F46J53                         #
#       18F27J53        18F47J53                         #
#                                       #
#########################################################################
#       Work in progress                                #
#########################################################################
#                                       #
#       16f1455                                 #
#       18f13k50        18f14k50                         #
#                                       #
#########################################################################

############CONFIG############
PROC=18f2550
```

```
OSC=20
STRINGDESC=0
LVP=0

#########################################################################3
LOW_SPEED    = 1
HIGH_SPEED   = 0
LVP          = 0

#############################################################################
#          CONFIGURATION OPTIONS                                            #
#############################################################################

CPU     = $(PROC)
CRYSTAL         = $(OSC)

# usb speed (HIGH_SPEED or LOW_SPEED)
USBSPEED = $(HIGH_SPEED)

# SDCC's toolchain directory
SDCCDIR         = C:\SDCC
# Pinguino directory
#PINGUINODIR = /dvpt/pinguino/local/branches/x.4

#############################################################################
#          DO NOT CHANGE FOLLOWINGS WITHOUT CARE                            #
#############################################################################

# bootloader version (cf. changelog.txt)
MAJ_VER         = 4
MIN_VER         = 8

# end of bootloader / start of pinguino application
ENTRY = 0x0C00

# string descriptor flag
STRING          = $(STRINGDESC)

#ifeq ($(USBSPEED), $(LOW_SPEED))
#CRYSTAL        = 8
#endif

#FAM                        = $(shell echo $(CPU) | cut -c 1-3)
FAM     = $(findstring 18f, $(CPU))

ifeq ($(FAM), 18f)
ARCH   = pic16
OPTIMIZ         = --optimize-df --optimize-cmp --obanksel=9 --denable-peeps
else
ARCH   = pic14
endif

ifeq ($(CRYSTAL), INTOSC)
PRJ      = Bootloader_v$(MAJ_VER).$(MIN_VER)_$(CPU)_$(CRYSTAL)
else
PRJ      = Bootloader_v$(MAJ_VER).$(MIN_VER)_$(CPU)_X$(CRYSTAL)MHz
endif

# files
# startup files (cf. sdcc/sources/sdcc/device/lib/pic16/startup)
```

```
# crt0      minimal initialisation routine
# crt0i     initialisation of variables (default)
# crt0iz RAM cleanup and initialisation of variables
CRTB   = crt0iBoot4
#CRTP  = crt0iPinguino
SRCS   = $(wildcard src/*.c)
OBJS   = $(SRCS:src/%.c=obj/%.o)

# directories
BINDIR          = $(SDCCDIR)\bin
#INCLUDEDIR  = $(SDCCDIR)/share/sdcc/non-free/include/$(ARCH)
#LIBDIR                 = $(SDCCDIR)/share/sdcc/lib/$(ARCH)

# flags
CC      = $(BINDIR)\sdcc

CFLAGS          = -V \
                  -m$(ARCH) \
                  -p$(CPU) \
                  $(OPTIMIZ) \
                  --fomit-frame-pointer \
                  --use-non-free \
                  -D ENTRY=$(ENTRY) \
                  -D STRING=$(STRING) \
                  -D SPEED=$(USBSPEED) \
                  -D CRYSTAL=$(CRYSTAL) \
                  -D LVP=$(LVP) \
                  -D MAJOR_VERSION=$(MAJ_VER) \
                  -D MINOR_VERSION=$(MIN_VER)

AFLAGS          = -Wa-w2

# we use our own linker script and startup code (work)
LDFLAGS         = -Wl,"--map -w -s lkr/boot4.$(CPU).lkr" \
                  --use-crt=obj/$(CRTB).o
#                           --no-crt

# we use default linker script and own startup code (doesn't work)
#LDFLAGS                = --no-crt

########################################################################
#       RULES                                                  #
########################################################################

#all: start clean $(CRTP).o $(PRJ).hex size
all: start $(CRTB).o $(PRJ).hex size

# Start message
start:
        @echo $(PRJ) ...

# Deleting all the debug files generated

clean:
        @rm -f obj/*.o  obj/*.asm obj/*.lst
        @rm -f hex/*.cod hex/*.map hex/*.lst

# make obj/crt0iBoot4.o
$(CRTB).o:
        @$(CC) $(CFLAGS) $(AFLAGS) -c obj/$(CRTB).c -o obj/$(CRTB).o
```

```
# make obj/crt0iPinguino.o
#$(CRTP).o:
#	@echo Compiling $(CRTP).o ...
#	@$(CC) $(CFLAGS) --ivt-loc=$(ENTRY) -c obj/$(CRTP).c -o obj/crt0i$(CPU).o
#	@cp obj/crt0i$(CPU).o $(PINGUINODIR)/p8/obj/

# Compiling and assembling all the src/.c in obj/.o
obj/%.o: src/%.c
	@$(CC) $(CFLAGS) $(AFLAGS) -c $< -o $@

# Linking the modules all together
$(PRJ).hex: $(OBJS)
	@$(CC) $(CFLAGS) $(AFLAGS) $(LDFLAGS) -o hex/$@ $^

# Calculating the whole code size
size:
	@tools/codesize.py hex/$(PRJ)

# Programming the Chip
#burn:
#	tools/picpgm -pic PIC$(CPU) hex/$(PRJ).hex
```

Para este proyecto se utiliza la siguiente configuración:

- PROC=18f2550
- OSC=20
- STRINGDESC=0
- LVP=0

Donde PROC es el microcontrolador que será utilizado, actualmente soporta los siguientes microcontroladores:

- 18F2550
- 18F4550
- 18F2455
- 18F4455
- 18F26J50
- 18F46J50
- 18F25K50
- 18F45K50
- 18F26J53
- 18F46J53
- 18F27J53

- 18F47J53

OSC es la frecuencia del Cristal, las siguientes frecuencias son soportadas (pueden variar de acuerdo al microcontrolador):

- 4MHz
- 8MHz
- 12Mhz
- 16MHz
- 20MHz
- 24MHz
- 40MHz
- 48MHz

STRINGDESC sirve para determinar el tipo de descriptor:

- 0 el descriptor no es un string, se rellena con ceros.
- 1 el descriptor es de tipo string(cadena de caracteres).

LVP (Low Voltage Programming)

- 0 LVP deshabilitado
- 1 LVP habilitado

4.4. IDE Pinguino

Esta es una aplicación independiente hecha con Python, tiene un preprocesador que convierte instrucciones específicas de Arduino directamente a C, este preprocesador reduce el tamaño del código, y además incrementa la velocidad en la ejecución.

Posee un compilador de C(SDCC para microcontroladores de 8bits y GCC-mips-elf para microcontroladores de 32bits), ensamblador y enlazador(GPUTILS para microcontroladores de 8bits y BINUTILS para microcontroladores de 32bits), y un bootloader USB(basado en Microchip USB HID para microcontroladores de 32bits)

Para programar las aplicaciones de usuarios a través del bootloader, se necesita el

IDE Pingüino, el cual requiere tener instalado lo siguiente:

- Python: Es el intérprete para el lenguaje de programación en que está escrito el IDE de Pinguino.
- wxPython: paquete para Python que permite crear aplicaciones con entorno gráfico.
- LibUSB: biblioteca para acceder al Puerto USB.
- Pinguino Drivers: De Microchip modificados para ser usados en el proyecto Pinguino.

La librería LibUSB y los drivers tienen que ser ejecutados como administrador. Luego de instalar los drivers y conectar el equipo, en el administrador de dispositivos aún no está reconocido el dispositivo, por lo que aparece en la lista de otros dispositivos, en este caso será necesario asociar el dispositivo con los drivers previamente instalados.

En la figura 22 se observa, como aparece con una advertencia, debido a que aún no está configurado el driver por completo. Para terminar la configuración debemos dar clic derecho en el dispositivo, y elegir la opción actualizar drivers, los cuales tendrán que ser buscados manualmente en el computador, en la carpeta donde se instaló.

Aparecerá un mensaje, advirtiendo que Windows no puede verificar los drivers, seleccionar la opción de instalar de todas formas, terminado esto estará listo para ser usado.

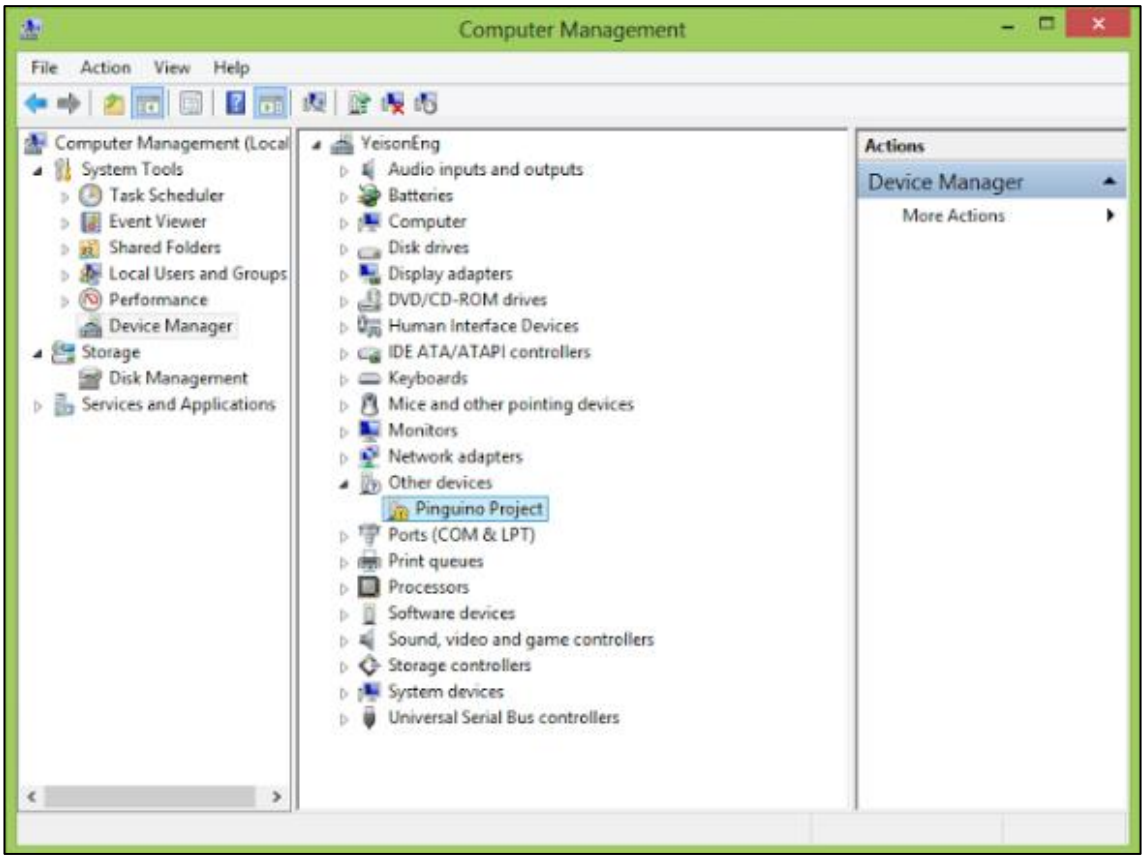

Figura 22. Administrador de dispositivos

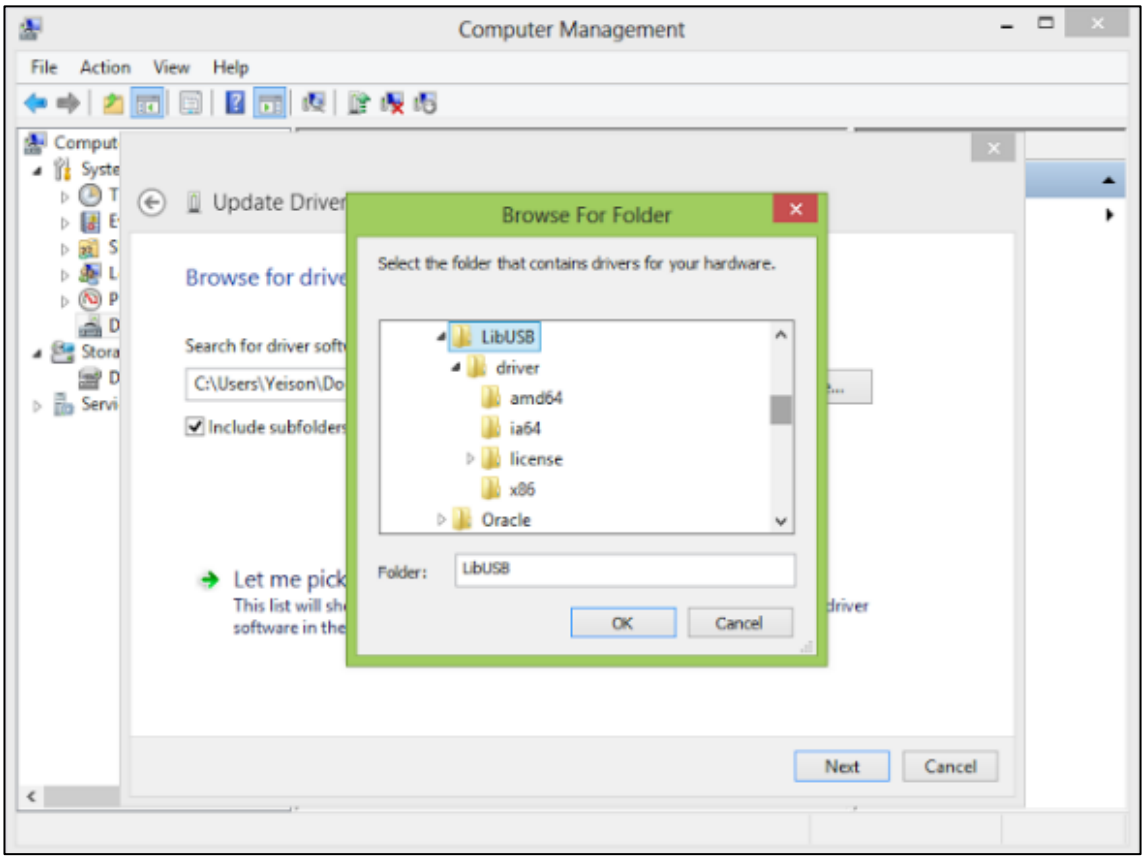

Figura 23**.** Instalación de Drivers

En Windows 8 es necesario cambiar la configuración de Windows para poder instalar drivers no firmados, caso contrario no podrá ser utilizado el dispositivo.

El IDE viene con varios ejemplos, listos para ser compilados y descargados a la placa, como se observa en el gráfico al abrir cualquier ejemplo es necesario compilarlo, (icono de engranes).
Realizado esto, es necesario dar un reset a la placa con un pulsante ubicado a la derecha, después del reset es admitida la carga de aplicaciones a la placa

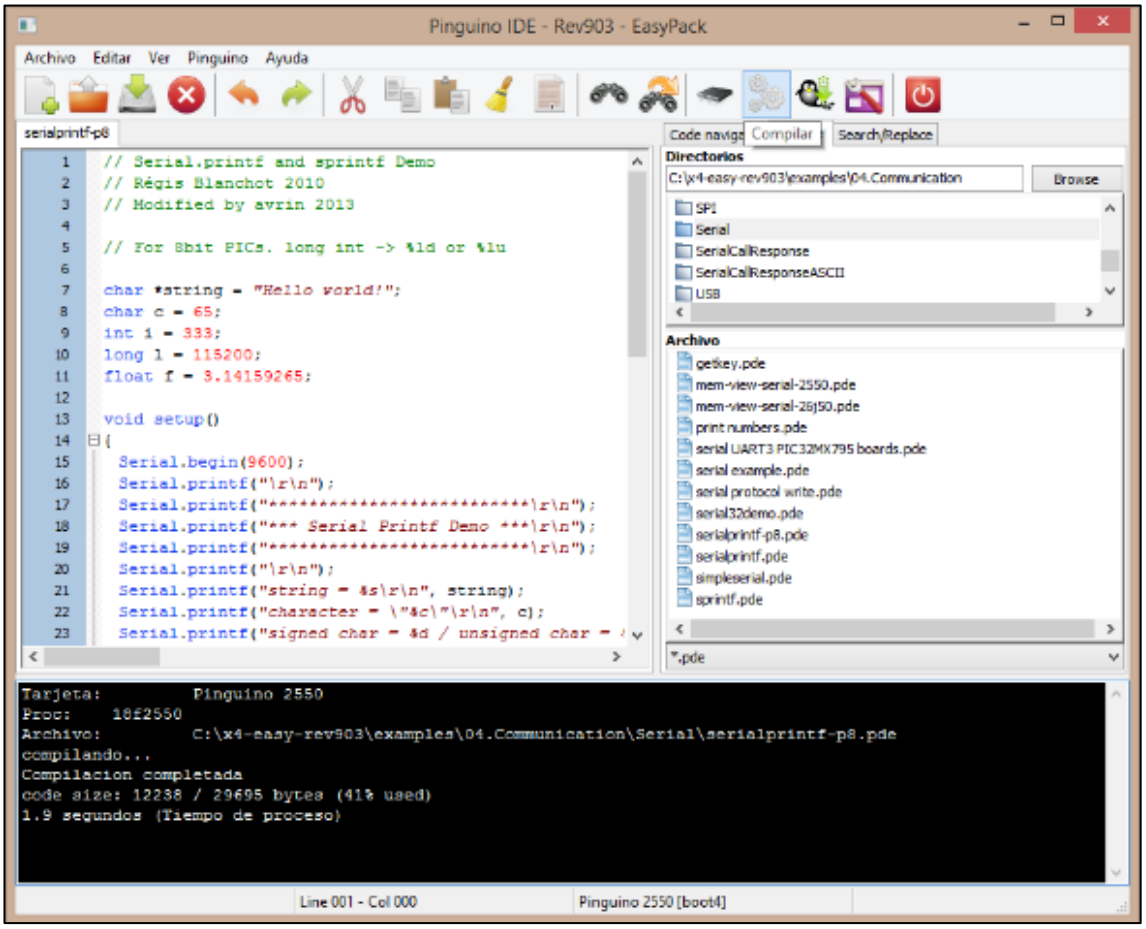

Figura 24. Compilación de un proyecto

Realizada la compilación, la descarga se realiza dando clic en el botón en forma de un pinguino.

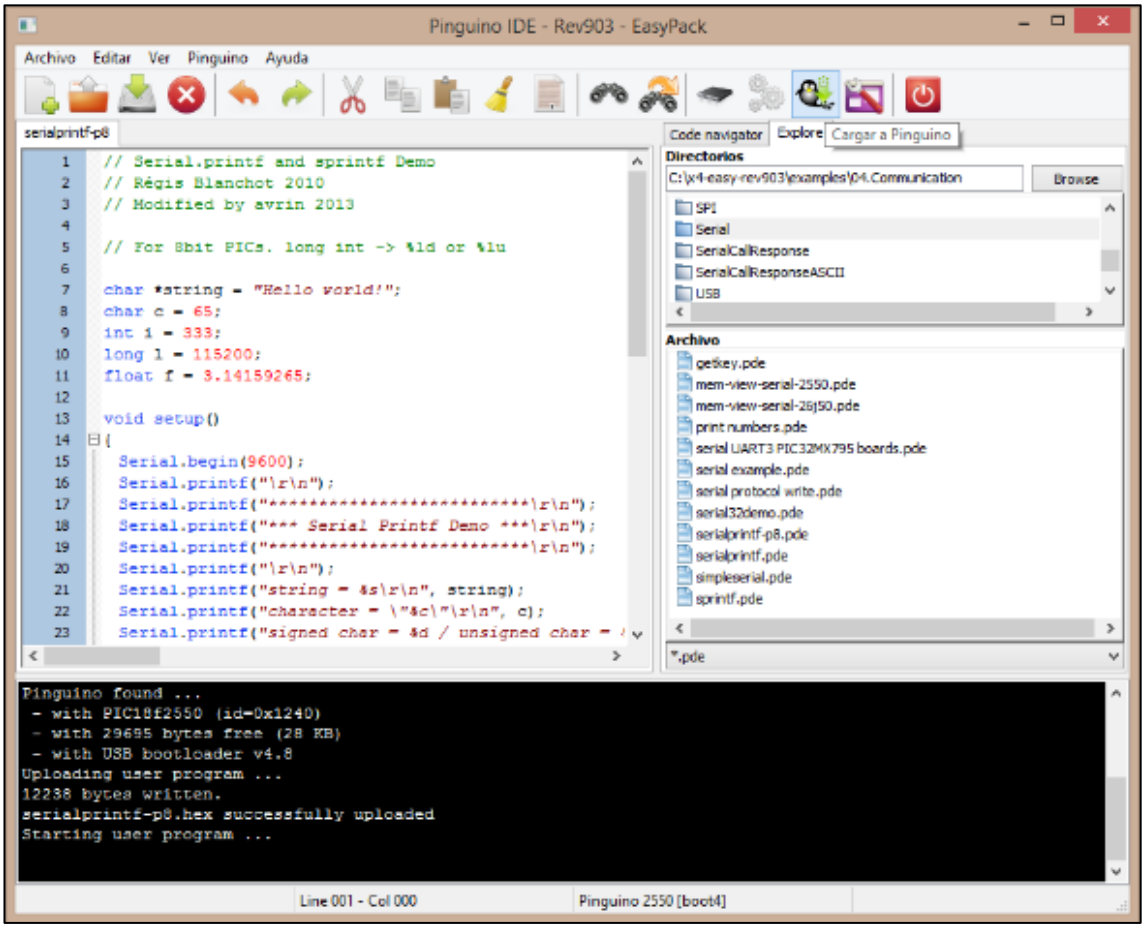

Figura 25. Descarga de una aplicación.

Conclusiones

Como se observa la aplicación debe ser lo más sencilla posible, debido a que el bootloader debe ocupar la cantidad menor de memoria posible, y no debe reducir los recursos para la aplicación final, que será ejecutada, además hay que tener en cuenta todas las consideraciones necesarias al instalar la aplicaciones y drivers para no tener problemas, al ejecutar, compilar o descargar aplicaciones.

CAPÍTULO 4

PRUEBAS DE FUNCIONAMIENTO

Introducción.

En este capítulo se detallan las pruebas realizadas para determinar el correcto funcionamiento del hardware desarrollado, así como de las librerías proporcionadas en el IDE pingüino, las pruebas consisten en la utilización de varios periféricos, como son puertos digitales, analógicos y de comunicación.

4.1. Prueba de Reconocimiento con la PC

Si los drivers se encuentran correctamente instalados, y el hardware está correcto, en el administrador de dispositivos aparecerá en la lista dentro de “libusb-win32 devices” como Pinguino, al abrir sus propiedades dentro de “Estado del dispositivo” tendremos “Este dispositivo funciona correctamente” (Ejemplo en Windows 8, 64 bits), como se observa en la Figura 26.

4.2. Prueba de comunicación con la PC

Esto se realiza desde el IDE Pinguino, la prueba consiste en descargar una aplicación, lo cual se observó en la Figura 25.

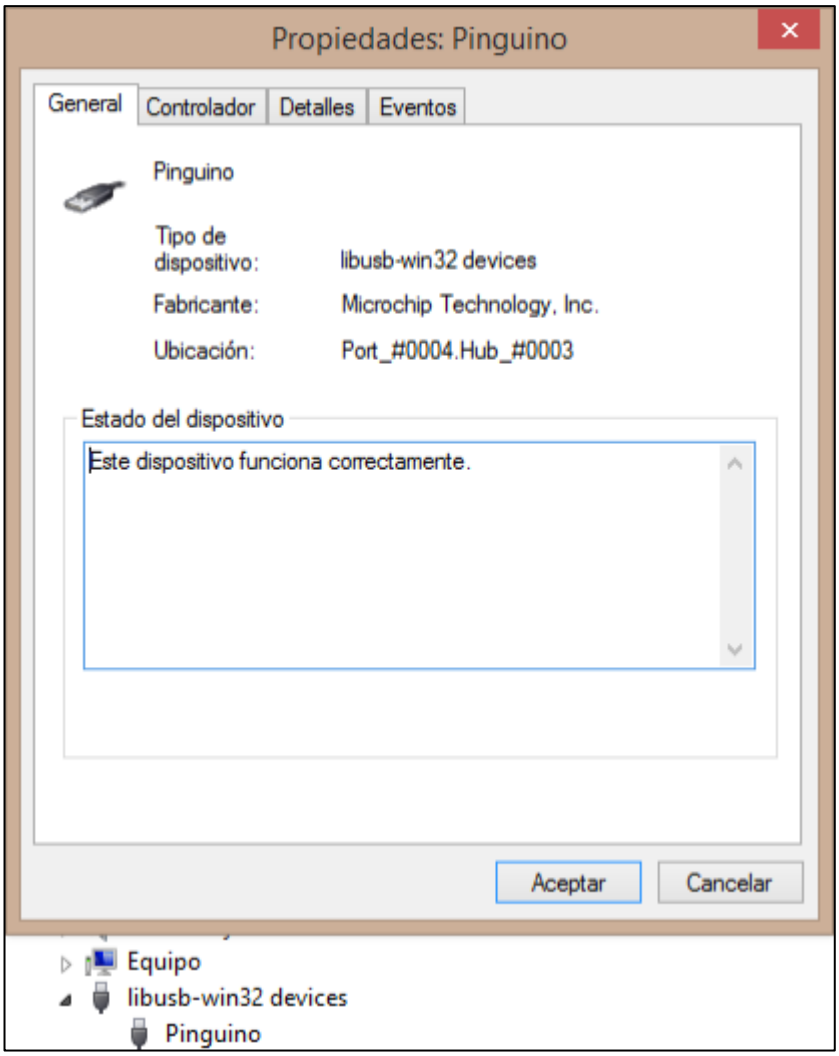

Figura 26**.** Comprobación de los drivers

4.3. Pruebas funcionamiento

A continuación, observamos los resultados de las pruebas realizadas con la comunicación serial, transmitiendo y recibiendo datos hacia y desde la PC.

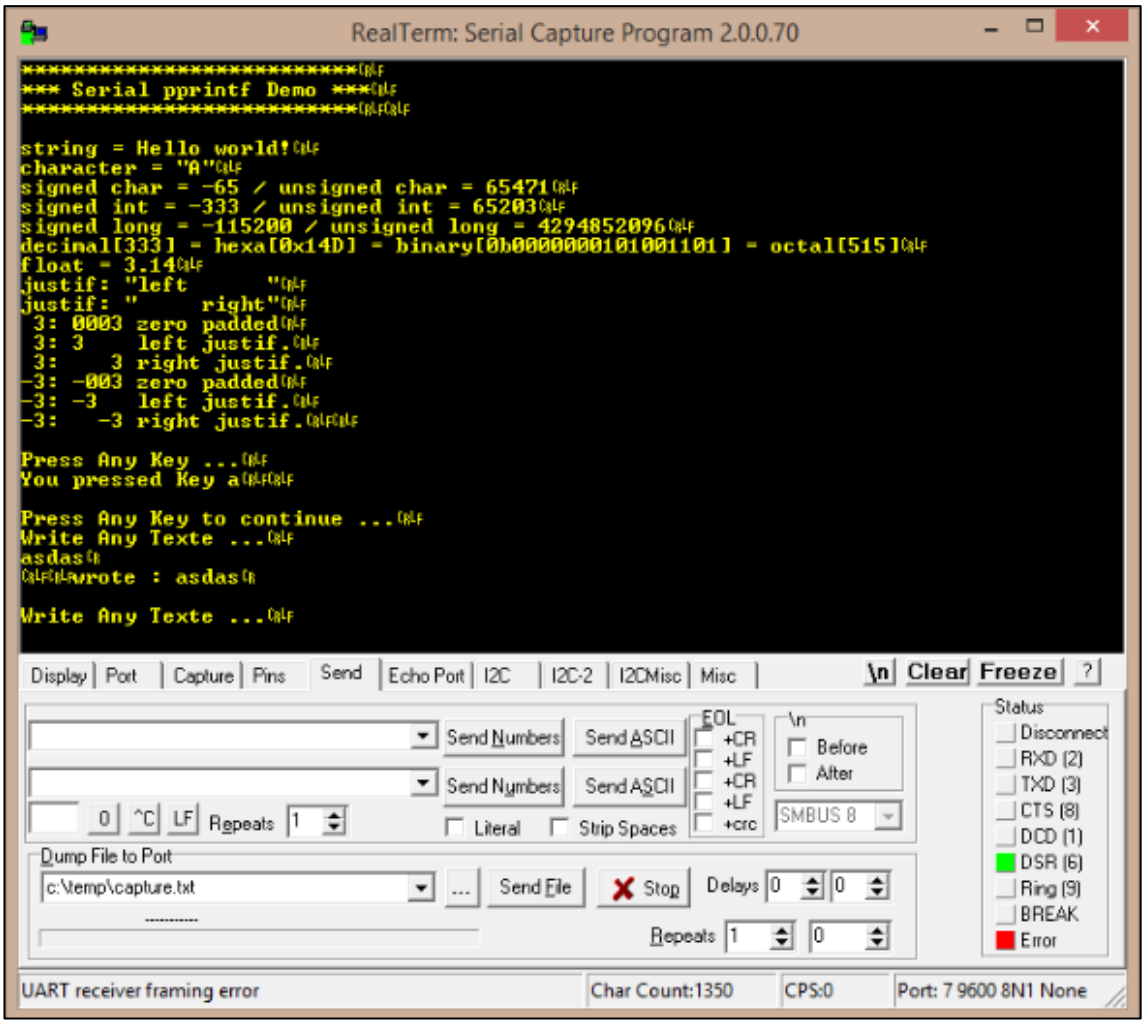

Figura 27**.** Pruebas comunicación serial.

Esta prueba se realizó utilizando el software RealTerm el cual permite, leer datos a partir de diferentes puertos de comunicación, en este caso se utiliza el puerto serial, a 9600 baudios.

La aplicación utilizada es serialprintf-p8 que se encuentra en los ejemplos que vienen incluidos con el IDE Pinguino

Prueba PWM, se utilizar un led, el cual al variar el PWM aparentemente variara su intensidad luminosa.

También existen varias librerías para diferentes usos, como por ejemplo el uso de servomotores

CONCLUSIONES Y RECOMENDACIONES.

Luego de revisar las necesidades de realizar prototipos electrónicos, ya sea de personas con conocimiento de electrónica, tanto como para personas que lo hacen por hobbie, se determinó que estos dispositivos de código abierto son las mejores opciones para dichos prototipos, incluyendo a estudiantes de otras carreras con lo que se les facilitaría realizar varios proyectos, sin la necesidad de grandes conocimientos en electrónica, además las personas que ya tienen conocimiento en electrónica también les resultará de manera más rápida y eficiente realizar sus proyectos en el menor tiempo posible, el limitante es que su uso debe darse principalmente para desarrollo y pruebas de equipos, no es muy recomendable su uso para productos finales, debido a que estos deben tener un hardware especifico mientras que placas como Arduino, o pingüino son un hardware genérico para desarrollo y pruebas.

Sin embargo, hay mucha gente que utiliza estos dispositivos para aplicaciones específicas porque les resulta más rápido y sencillo realizar las mismas, en carreras como sistemas o diseño resultaría muy útil, por ejemplo en la carrera de sistemas les facilitaría su utilización para realizar puertos de comunicación con elementos externos a una computadora, o en diseño para la realización de sistemas de iluminación controlados por PWM en los que se puede obtener cualquier color utilizando led’s RGB

BIBLIOGRAFÍA

Referencias Electrónicas:

1. Esquemas genéricos 18F2550 http://pinguino.cc/download/schematics/Generic2550 [Consulta 10 Julio 2013]
2. Código fuente de pinguino https://code.google.com/p/pinguino32/source/browse/#svn%2Fbootloaders [Consulta 18 Septiembre 2013]
3. MPLABX https://www.microchip.com/pagehandler/en-us/family/mplabx/ [Consulta 5 Enero 2014]
4. SDCC http://sdcc.sourceforge.net/doc/sdccman.pdf [Consulta 10 Enero 2014]
5. Constantes en microcontroladores 18Fx550 y 18Fx5J50 http://wiki.pinguino.cc/index.php/PIC18Fx550_and_PIC18Fx5J50_Constants [Consulta 18 Febrero 2014]
6. Instalación IDE Pinguino y drivers http://yeisoneng.appspot.com/post/pinguino_svn_windows/ [Consulta 16 Febrero 2014]