Universidad Politécnica de Cartagena

industriales

etsii UPCT

Periférico de acceso al PC para discapacitados de diseño hardware/software abierto.

Titulación: Ingeniería Técnica Industrial, esp. Electrónica Industrial
Intensificación: Automática
Alumno/a: José Antonio Sánchez García
Director/a/s:Joaquín Roca González

Cartagena, 5 de Julio de 2013

ÍNDICE

1.Introducción. 3
1.1 Objetivo 4
2.Revisión del estado del Arte 7
2.1 Ratones 7
2.2 Hardware Libre 12
2.2.1 Definición de Software Libre 12
2.2.2 Libertades del software libre 14
2.2.3 Hardware libre 16
2.2.4 Ejemplos de Hardware Libre 21
3. Materiales y métodos. 27
3.1 Bus Serie Universal 27
3.2 PIC18F14K50 y Entorno Low Pin Count 33
3.3 SDK y lecciones 36
3.3.1 MPLAB IDE y C18 36
3.3.2 Lecciones 37
3.4 Software propio 48
3.4.1 Microchip HID Bootloader 49
3.4.2 Implementación del software propio. 50
4. Hardware 89
4.1 Actuadores 89
4.2 Elaboración del PCB. 90
4.2.1 Software utilizado para el diseño del PCB. 90
4.2.2 Diseño del esquemático. 92
4.2.3 Elaboración del PCB 95
4.3. Banco de pruebas. 107
5. Conclusiones y futuros proyectos 109
6. Bibliografía 111
7. Enlaces. 112
8. Anexos 113
8.1 Presupuesto 113
8.2 Planos 116
8.3 Código 123
8.4 Índice de figuras 136

1.Introducción

Existen diversos elementos de carácter tecnológico que facilitan el acceso al PC a las personas con discapacidad, de venta tanto en mercados públicos como otros más privados y menos accesibles al público general. Estos dispositivos suelen tener un coste muy alto, no suelen ser reconfigurables y tampoco son eficientes para algunos casos concretos de discapacidad.

En algunos casos de discapacidad, como puede ser el caso de la invidencia, se han creado multitud de accesos al PC totalmente funcionales que permiten no solo la interactuación del usuario con el ordenador a nivel de usuario estándar, sino que le permite desarrollar una actividad laboral con un rendimiento más que aceptable.

La Organización Nacional de Ciegos Españoles y la fundación ONCE mejoran la vida de las personas ciegas invirtiendo en I+D+I y ayudan tanto a los invidentes como a las empresas que los contratan, proporcionando el equipamiento necesario para su desarrollo personal y laboral.

Entre los dispositivos de hardware que proporciona la ONCE desde el Centro de Investigación, Desarrollo y Aplicación Tiflotécnica (CIDAT), se pueden encontrar por ejemplo, las líneas braille. Estos dispositivos permiten que el discapacitado sea capaz de leer en braille una línea de texto subrayada en un procesador de texto, en un documento o en una página web, existiendo distintos modelos de la misma.

Figura 1.1 - Línea Braille SuperVario 80

1.1 Objetivo

Dentro del acceso al ordenador mediante ratón para personas con discapacidad, se puede encontrar una gran dispersión de productos con diversas formas y tecnologías. Estos modelos presentan un diseño hardware y software muy rígido, no permitiendo su reconfiguración para adaptar el producto a las necesidades de cada individuo.

Esta característica produce que el precio de venta de estos producto sea muy elevado debido a las bajas ventas de cada modelo, ya que el público que satisfacen cada dispositivo por separado es bastante reducido.

Se pueden encontrar ratones para mentón, ratones con joysticks de diversas formas, con pulsadores y modelos convencionales que pueden ser aptos para algunas discapacidades, como ciertos modelos de trackballs, como se puede observar en la figura:

Figura 1.2 - Ratones para discapacitados

El objetivo principal del proyecto es la fabricación de un dispositivo de acceso hardware al PC del tipo ratón que cumpla con las siguientes características:

- Basado en Software y Hardware Libre.
- Altamente adaptable.
- Altamente configurable.
- Altamente modificable.
- Compatible con los sistemas operativos más comunes.
- Económico.
- Fácilmente reproducible.
- Programación en lenguaje de alto nivel.
- Basado en el protocolo USB estándar de Dispositivo de Interfaz Humana.

Dentro de las múltiples opciones que proporciona el mercado (microprocesadores, microcontroladores, FPGA y ASIC), se ha procedido a la utilización de tecnología basada en microcontroladores para el desarrollo del proyecto.

La elección de los microcontroladores está fundamentada por los siguientes hechos:

- **Económicos**: Siempre que el nivel de producción no sea excesivamente grande.
- **Facilidad de implementación**: Debido a que incorporan en el mismo encapsulado casi todos los elementos necesarios para su funcionamiento (memoria ROM del programa, memoria RAM, periféricos, ect).
- **Interfaz USB integrada:** Existen en el mercado modelos que llevan incorporado la interfaz de comunicaciones USB, y permiten la programación del mismo utilizando lenguaje de alto nivel.

El proyecto se desarrollará siguiendo el siguiente esquema:

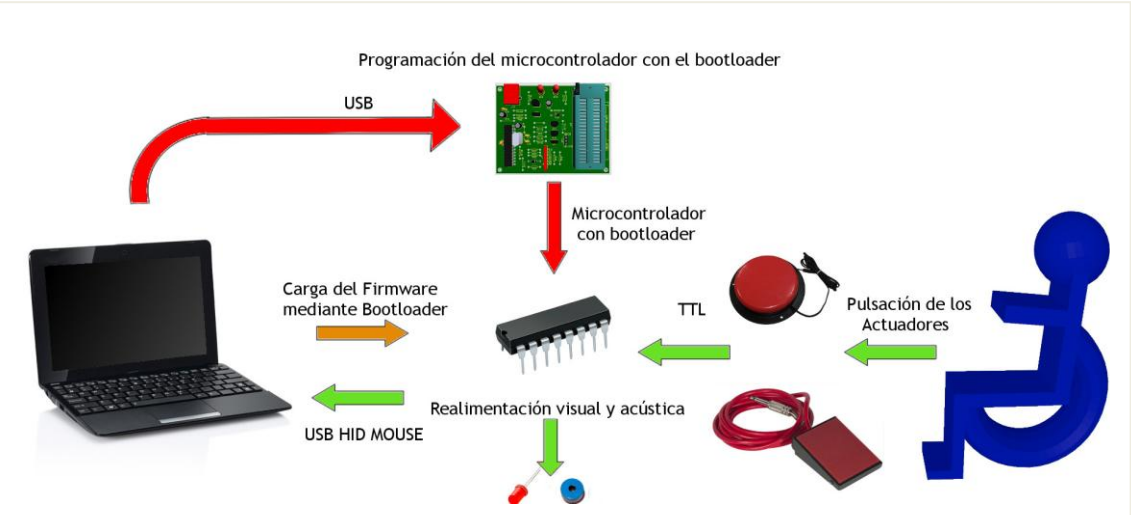


Figura 1.3 - Esquema de funcionamiento del proyecto

En el esquema se puede ver como el discapacitado , mediante una acción voluntaria sobre un actuador, envía señales eléctricas tipo TTL al microcontrolador. Éste a su vez interpreta estas señales y envía al PC la acción correspondiente utilizando el puerto USB bajo el protocolo de Dispositivo de Interfaz Humana (Human Interface Device o HID). Notar que, aunque en el esquema los actuadores estén representados por un gran pulsador y un pulsador de pedal, se puede conectar cualquier tipo de pulsador o dispositivo que envíe señales tipo TTL al microcontrolador, aumentando con ello su adaptabilidad.

El microcontrolador envía dos señales de realimentación hacia el usuario cada vez que éste interactúa con los pulsadores; una señal de realimentación lumínica mediante el encendido de un diodo LED y una señal acústica generada tras la excitación de un piezoeléctrico. El ordenador, como es habitual en este tipo de dispositivos, moverá el cursor en la dirección deseada o generará acciones de clicado, creando también una realimentación visual y, en el caso del clicado, realimentación acústica (si estuviese configurado para ello).

Si el usuario lo desea, puede modificar el firmware del dispositivo mediante la carga de otro más apropiado a sus necesidades utilizando el bootloader incluido en el microcontrolador, sin necesidad de recurrir a hardware externo de programación. El origen del nuevo firmware puede ser propio (si el usuario obtiene el proyecto y lo reprograma y recompila) o externo (mediante la descarga de otros archivos compilados por otros usuarios).

Para la instalación previa del bootloader, el usuario precisa disponer del hardware y software necesario para la programación del microcontrolador en cuestión. Otra alternativa para evitar la compra del hardware programador es la adquisición de microcontroladores previamente programados con el bootloader, en el caso de que el proveedor permita esa opción a la hora de realizar el pedido.

La tarjeta controladora debe de tener los puertos de entrada de los periféricos accesibles y fácilmente modificables, tanto a nivel hardware como a nivel software, para facilitar la conexión del periférico que más convenga en cada caso, obteniendo así la alta capacidad de configuración del dispositivo.

2.Revisión del estado del Arte

2.1 Ratones

El ratón de ordenador fue creado por Douglas Engelbart y Bill English en los años 60 en el Stanford Research Institute (California) y mejorado posteriormente por Xerox en sus laboratorios de Palo Alto. El dispositivo surgió como parte de un proyecto que buscaba aumentar el intelecto humano mejorando la comunicación entre el hombre y la máquina. La aparición de estos dispositivos propició las interfaces gráficas de usuario, ya que permitía desplazar un puntero por la pantalla, facilitando la accesibilidad de estos entornos gráficos.

Figura 2.1 - Réplica de prototipo de ratón

Aunque su aspecto puede resultar arcaico, la primera maqueta de ratón, bautizada con el nombre de *X-Y Position Indicator for a Display System*, posee un funcionamiento básico similar a los ratones de hoy en día. Este prototipo poseía dos ruedas metálicas que giraban al desplazarse por la superficie y, con su movimiento, proporcionaba al ordenador los desplazamientos en los ejes X e Y del puntero. Disponía también de un pulsador que proporcionaba la acción de click al pc.

Desde su creación han aparecido en el mercado multitud de modelos de ratones para ordenador, bajo distintas tecnologías de funcionamiento y distintas formas, aunque todos con la misma finalidad.

Ratones mecánicos.

El ratón mecánico es el modelo clásico de ratón para PC, que poco difiere en su funcionamiento del primer prototipo de mouse.

Los ratones mecánicos suelen tener una esfera de goma en su interior que actúa sobre dos ejes al mover el mouse sobre una superficie plana. Estos dos ejes están acoplados a dos ruedas dentadas que al girar transmiten el movimiento al circuito del ratón mediante encoders.

Figura 2.2 - Ratón mecánico.

El principal problema de estos dispositivos es la increíble facilidad con la se ensucia el mecanismo, debido a que la bola de goma atrapa una gran cantidad de suciedad en el desplazamiento del dispositivo.

Esta acumulación de suciedad impide que la esfera de goma transmita fielmente el movimiento del ratón, dando lugar a fallos de precisión del puntero en la pantalla. Por este motivo, la mayoría de los ratones mecánicos permiten acceder al mecanismo de rodillos para limpiarlos y restaurar el funcionamiento normal del dispositivo.

.

Ratones ópticos.

Los ratones ópticos son los más extendidos actualmente en el mercado, debido a su alta inmunidad a la suciedad y su buen funcionamiento, así como su bajo precio.

Figura 2.3 - Ratón óptico.

El funcionamiento de los ratones ópticos se basa en la captación de fotografías de la superficie sobre la que se desplaza el dispositivo. Un sensor óptico, encargado de la captura, toma dos fotografías entre dos instantes consecutivos con ayuda de un haz de luz óptica y, tras realizar una comparación de los datos obtenidos, decide cual ha sido la dirección en la que se ha desplazado el ratón y traslada esta dirección al puntero del mouse.

La gran ventaja de los ratones ópticos con respecto de los mecánicos es que no les afecta prácticamente la suciedad del entorno, así como el aumento considerable de la sensibilidad del dispositivo, que no suele ser inferior a 300 dpi.

El mayor inconveniente de la utilización de este tipo de ratones es que el sensor óptico suele fallar en superficies brillantes o pulidas, por lo que en algunos casos el usuario se ve obligatorio a utilizar alfombrillas para el ratón.

Ratones láser.

El funcionamiento de este tipo de ratones es similar al de los ratones ópticos. Tan sólo cambia el haz de luz óptica por un haz de luz láser, lo que aumenta considerablemente la sensibilidad de este tipo de ratones.

Figura 2.4 - Ratón láser.

Es el ratón más utilizado en diseño gráfico y para gaming, debido a que son dispositivos de altas prestaciones. Gracias a la utilización de la tecnología láser, permite que estos dispositivos lleguen a alcanzar precisiones superiores a los 5000 dpi. Debido a esto, este tipo de dispositivos suelen contar con un selector de precisión incorporado para variarla mientras se está utilizando.

Trackball

Las trackball difieren en su funcionamiento básico con respecto de los ratones convencionales, ya que a diferencia de éstos, las trackballs no se arrastran sobre una superficie, sino que el movimiento del puntero se realiza moviendo una esfera con la mano. Hay distintos modelos de trackballs en el mercado, aunque el más convencional es el trackball de pulgar.

Figura 2.5 - Trackball.

La principal ventaja de este dispositivo es el espacio reducido que se necesita para su manejo, debido a que no hay que desplazarlo. Por consiguiente, este tipo de ratones prevén las molestias y enfermedades causadas por los dispositivos desplazables.

Aunque no son comunes en entornos domésticos, este tipo de ratones están muy extendidos en industrias y entornos no convencionales, como la industria naval.

Figura 2.6 - Teclado con Trackball.

2.2 Hardware Libre

2.2.1 Definición de Software Libre

Para poder hablar del hardware libre, primero debemos introducir el concepto de software libre, ya que el hardware libre tiene su base en los mismos principios.

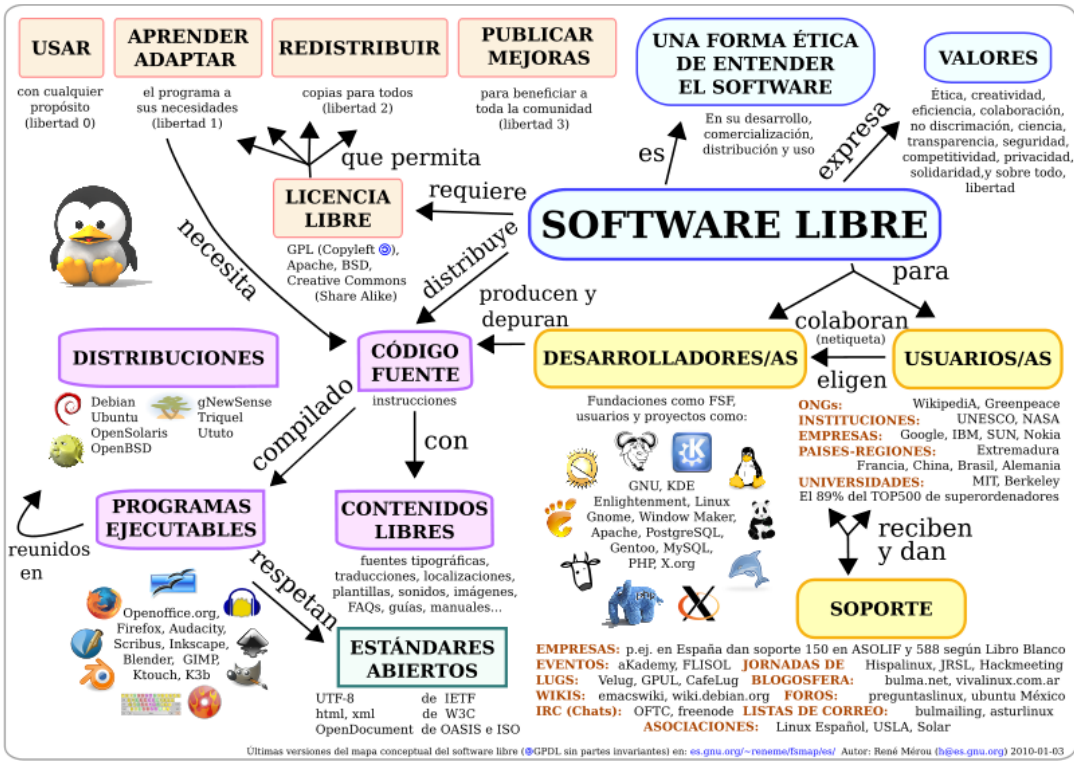


Figura 2.7 - Mapa conceptual del software libre

En los orígenes de la informática, el software no se consideraba un producto, sino tan sólo un añadido que los vendedores de equipos informáticos distribuían para que los usuarios pudieran trabajar con las computadoras, lo que fomentó que el software se creara y compartiera sin restricciones.

En la década de los 80 las computadoras empezaron a incorporar sistemas operativos privativos, lo cual forzaba a los usuarios a aceptar las condiciones restrictivas en cuanto a la modificación del software. En el caso de que un usuario encontrase un error en el programa, tan sólo podía comunicarlo a la empresa generadora de dicho software, para que ésta lo solucionara, aunque el usuario tuviese los conocimientos suficientes como para modificar el software y reparar el mismo el error.

Este fue el caso que le ocurrió a Richard Matthew Stallman, padre del concepto de Software Libre y precursor de *GNU* y *Free Software Foundation* .

En el laboratorio en el cual trabajaba recibió una impresora donada por una empresa externa cuyo funcionamiento no era del todo correcto, ya que el papel se atascaba. Esto generaba una gran pérdida de tiempo debido a que la impresora estaba conectada en red y cuando se atascaba no generaba ningún tipo de alarma que avisara a los usuarios. Mientras tanto, los usuarios seguían mandando trabajo a la impresora, generando un gran caos.

Figura 2.8 - Richard Matthew Stallman

Richard Stallman intentó implementar una mejora en el software de la impresora, creando un algoritmo que enviara un mensaje de error de atasco de papel por red cuando esto ocurriera, poniéndose en contacto con el fabricante de la impresora para obtener el permiso de modificación del software, recibiendo una negativa por parte del mismo.

Con este antecedente, en 1984 Richard Stallman comenzó a trabajar en el proyecto *GNU*, y un año más tarde fundo la *Free Software Foundation*, introduciendo el concepto de *copyleft*, que otorgaba libertad a los usuarios y restringía las posibilidades de apropiación del software.

2.2.2 Libertades del software libre

La definición de software libre indica que para que un determinado software se pueda calificar como software libre tienen que cumplir las siguientes libertades:

1) *Libertad de usar el programa con cualquier propósito.*
2) *Libertad de estudiar el funcionamiento del programa y su modificación para adaptarlo a las necesidades pertinentes.*
3) *Libertad de distribuir copias del programa.*
4) *Libertad de mejorar el programa y hacer públicas esas mejoras, de modo que la comunidad se beneficie.*

A continuación desglosaremos las distintas libertades con su correspondiente definición.

1. Libertad de usar el programa con cualquier propósito.

El usuario del programa puede utilizar el mismo para el propósito que le convenga, aunque sea distinto para el que fuera diseñado el software en su origen.

2. Libertad de estudiar el funcionamiento del programa y su modificación para adaptarlo a las necesidades pertinentes.

El usuario puede aprender cómo funciona el programa y puede modificarlo libremente, pero para ello necesita tener acceso al código fuente del mismo, por lo que el programador inicial tiene el deber, en principio, de proporcionar el código fuente del programa para ello.

3. Libertad de distribuir copias del programa.

El usuario puede crear copias del programa y distribuirlas libremente.

4. Libertad de mejorar el programa y hacer públicas esas mejoras, de modo que la comunidad se beneficie.

Esta libertad, similar a la libertad número tres, permite la copia y distribución de las distintas modificaciones creadas por el usuario.

En resumen, todo software libre se podrá usar para lo que el usuario crea conveniente, podrá estudiarse y modificarse y se podrán realizar copias del programa y subprogramas generados con el código fuente.

Como se puede observar en las libertades del software libre, ninguna especifica cómo se ha de tratar el precio del software. Al contrario de lo que se puede creer, el software libre no es software gratis. Un eslogan frecuentemente usado para la desambiguación del término libre es "*Libre como en libertad, no como en cerveza gratis*" (en Inglés, *"Free as in freedom, not as in free beer"*, debido al múltiple sentido que tiene el término *free* en la lengua anglosajona), por lo tanto, los desarrolladores pueden no ofrecer el acceso al código sin antes haber facturado por ello.

Aunque estas son las bases del software libre, existen multiples licencias basadas en este tipo de definición. Las más conocidas son:

- Licencia Pública General de GNU(GNU GPL)

Figuras 2.9 - Logotipo de GPL V.3

El autor conserva los derechos de autor (copyright) del software, y permite la redistribución y modificación bajo unos términos que aseguren que todas las versiones modificadas del software permanecen bajo los términos más restrictivos de GNU GPL.

- Licencia Pública General de Affero(AGPL)

Figuras 2.10 - Logotipo de AGPL V.3

Licencia copyleft derivada de GNU GPL diseñada para asegurar la cooperatividad de la comunidad para software utilizados en servidores de red.

- Licencia estilo BSD

Figura 2.11 - Logotipo de freeBSD

El programador mantiene la protección de copyright únicamente para la renuncia de garantía y para obtener la atribución de autoría en trabajos derivados, y permite la libre redistribución y modificación, incluso si dichos trabajos tienen propietario.

-Licencia Mozilla Public License (MPL)

Figura 2.12 - Logotipo de Mozilla

La licencia MPL promueve la colaboración en el desarrollo de software evitando el efecto del contagio de licencias que tiene las licencias GPL.

- Copyleft

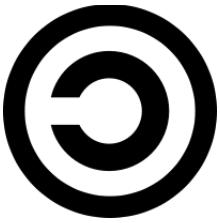


Figura 2.13 - Logotipo de Copyleft

El titular de los derechos de autor (Copyright) que haya licenciado su producto con licencia Copyleft, puede distribuir cualquier subproducto de su obra con la licencia que escoja, pudiendo ser licencia libre o no.

- Creative Commons

Figura 2.14 - Logotipo de Creative Commons

La licencia Creative Commons ofrecen al autor una forma simple y estandarizada de otorgar permiso al público en general de compartir y usar su trabajo bajo los términos y condiciones elegidos por el autor.

2.2.3 Hardware libre

Figura 2.15 - Logotipo del Open Source Hardware

Aunque el concepto de hardware libre intenta apoyarse en las mismas bases que el software de las mismas características, no existe una definición exacta en cuanto a este término se refiere.
Richard Stallman afirma que las cuatro libertades del software libre se pueden aplicar, en cuanto a hardware se refiere, tan sólo a los ficheros necesarios para el diseño y las especificaciones técnicas (esquemáticos, diseño de PCB etc.) pero no al circuito físico en sí.
La directa aplicación de este término al hardware, si se adopta tal y como lo entendemos para el software, conlleva distintos inconvenientes.

Problemas del Hardware Libre.

Como se ha comentado anteriormente, no se puede aplicar las cuatro libertades del software libre de manera directa ya que la naturaleza del hardware es , por lo general, distinta a la del software. Si se intenta aplicar las cuatro libertades del software libre surgirán los siguientes problemas:

Diseño físico único.

Cuando se ultima el diseño de un circuito hardware, se realiza una placa de circuito impreso con unos componentes y unas características concretas. La capacidad de compartir ese diseño con exactitud no es tarea sencilla y dependerá, entre otros factores, de la facilidad de reproducción del mismo.

Costes de reproducción.

El usuario que quiera utilizar el hardware debe fabricarlo, para lo cual tendrá que adquirir los componentes y construirlo, lo que conlleva unos costes adicionales y una destreza en su creación inexistentes en el software. En algunos casos se puede adquirir el hardware ya montado (Arduino, por ejemplo), pero aun así sigue existiendo el coste adicional de la compra.

Disponibilidad de los componentes.

Existe la posibilidad que en el momento de comenzar con la reproducción de un diseño no sea posible encontrar todos los componentes del mismo. En algunas ocasiones esos componentes pueden estar obsoletos o en desuso y en otras ocasiones ciertos componentes pueden estar restringidos en algún país, o existan barreras burocráticas para la adquisición de estos componentes por internet o catálogo, etc.

Existencia de patentes.

Algunos de los diseños pueden utilizar, o bien componentes patentados, o bien se pueden apoyar en circuitos patentados. En ese caso no existirá la total libertad de poder modificar esas partes del hardware ni de poder utilizarlas para otro fin que no sea el original, y el usuario tendrá que tener en cuenta la patente de esas partes del circuito en la utilización y rediseño del mismo.

Limitación de procesos de producción.

Debido a la infraestructura necesaria para la reproducción de los diseños (estaciones de soldadura, ácidos, placas insoladoras, hornos para soldadura en SMD...), existe la posibilidad de que no todos los usuarios puedan reproducir los circuitos por falta de acceso a estos medios, y se vean obligados a la compra del circuito o a la fabricación del mismo por una empresa externa.

Definiciones del Hardware Libre.

Debido a no existir una definición estricta del término hardware libre, se han creado dos clasificaciones del hardware libre, una de ellas dependiente de su naturaleza y la otra dependiente de su filosofía.

Según su naturaleza

Existen multitud de naturalezas diferentes para los diseños hardware, dependientes de múltiples factores. Desde el punto de vista del hardware libre, los distintos modelos de hardware se pueden englobar en dos grandes grupos.

Hardware reconfigurable.

El hardware reconfigurable es aquel que viene descrito por un lenguaje de descripción hardware (*Hardware Description Language o HDL)*. Debido a que en este tipo de hardware los diseños son archivos de texto que contienen la configuración del circuito en forma de código fuente, se pueden aplicar directamente las mismas licencias del software libre a estos archivos de configuración.

Sin embargo, estos archivos HDL se deben de montar sobre dispositivos electrónicos que sean capaces de cambiar su configuración hardware tras interpretar el código de programación (un claro ejemplo son las FPGA's o *Field Programmable Gate Array*).

Estos circuitos, en su gran mayoría, cuentan con patentes y licencias privativas, por lo que no se puede considerar el circuito físico al completo (hardware y software) como libre. Por lo tanto se puede considerar estos archivos de lenguaje de descripción hardware un caso especial de software libre ya que, al igual que pasa con éste, el objeto que se puede considerar libre es el software, no la máquina en la cual se ejecuta el mismo.

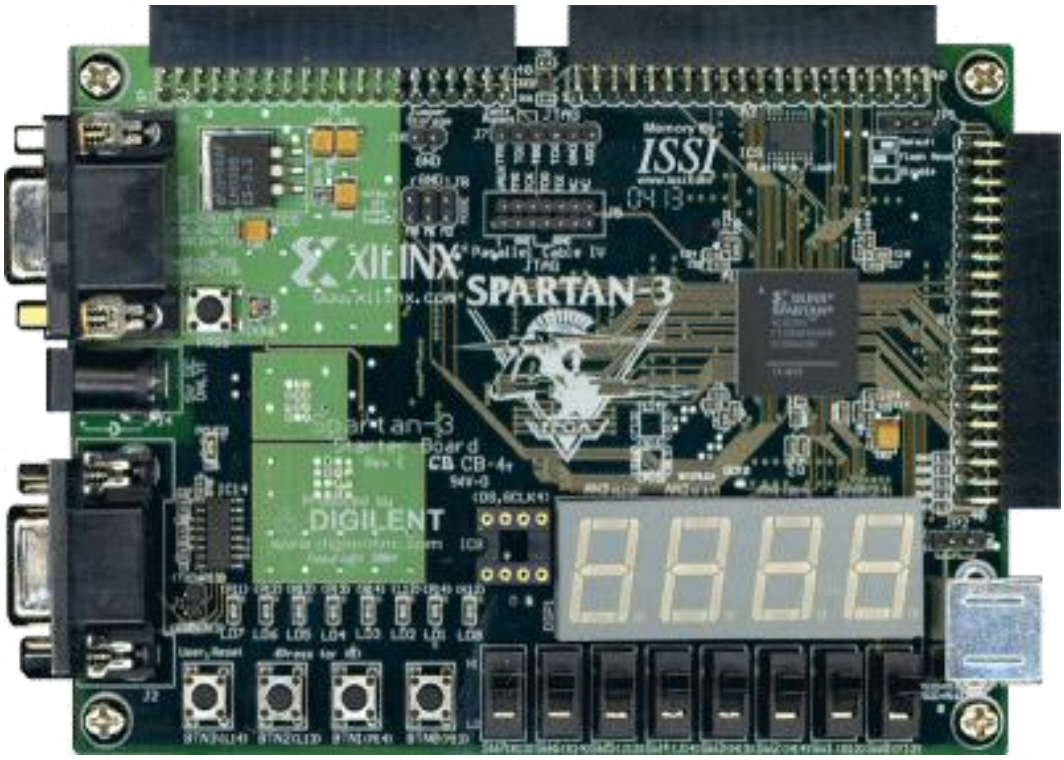


Figura 2.16. Xilinx Spartan 3, ejemplo de hardware reconfigurable.

Hardware estático.

Se considera hardware estático al conjunto de componentes y materiales de los sistemas electrónicos (lo que comúnmente es definido en informática como hardware).

Mientras que en el software los elementos que lo componen son intangibles (programas), el hardware estático tiene elementos físicos tangibles(componentes, placas de circuito impreso...). Esta característica dificulta la aplicación directa de las licencias del software libre a este tipo de circuitos, debido a los diversos problemas explicados anteriormente.

Figura 2.17 Beepic. ejemplo de hardware estático.

Según su filosofía.

De la misma manera que ha ocurrido con las licencias del software libre, se han creado un gran conjunto de definiciones de hardware libre, cada una con sus características especiales, que enumeraremos a continuación:

Free Hardware Design.

El diseño puede ser copiado, distribuido, modificado y fabricado libremente, sin implicar que el diseño no pueda ser vendido o que la puesta en práctica del diseño esté libre de coste.

Libre Hardware Design.

Idéntico al Free Hardware Design, pero matizando con la palabra libre la libertad, no al precio.

Open Source Hardware.

Aquel en el cual la información del diseño se pone a la disposición del público en general. Se puede basar en una Free Hardware Design o puede estar restringido.

Open Hardware:

Forma limitada de Open Source Hardware para cual el requisito es que:

"La suficiente documentación del dispositivo debe estar disponible para que un programador competente pueda escribir un controlador del dispositivo. La documentación debe cubrir todas las características de la interfaz del dispositivo - controlador que se espera que cualquier usuario emplee. Esto incluye funciones de entrada-salida, de control y funciones auxiliares como medidas de funcionamiento o diagnósticos de autoprueba. Los detalles de soporte de firmware on-board y de la puesta en práctica de hardware no necesitan ser divulgados excepto cuando son necesarios para permitir programar un controlador para el dispositivo".

Free Hardware:

Término ambiguo que intenta buscar el paralelismo entre el hardware libre y el software libre, aún implicando el estado físico del hardware, por lo que puede resultar muy confuso.

2.2.4 Ejemplos de Hardware Libre

A continuación expondremos algunos proyectos de Hardware Libre.

Figura 2.18 Logotipo de OpenEEG

OpenEEG

El proyecto OpenEEG permite al usuario que lo desee la creación de una interfaz de electroencefalograma (EEG). El objetivo final del proyecto es conseguir leer las señales eléctricas del cerebro para poder utilizarlas y estudiarlas. Estas señales pueden utilizarse para realizar interfaces cerebro-ordenador, para el entrenamiento cerebral (debido a que el equipo hace que el usuario sea más consciente del funcionamiento de su cerebro), o simplemente por mera curiosidad.

Debido a que el proyecto OpenEEG se desarrolla sin ánimo de lucro y que el proyecto se está construyendo a través de una gran comunidad, los desarrolladores del proyecto remarcan que no pueden ofrecer ningún tipo de garantía de uso, informando al usuario que, si decide realizar el proyecto, este correrá bajo su cuenta y riesgo.

La página web del proyecto cuenta con una extensa sección de preguntas frecuentes que intenta resolver las dudas sobre el funcionamiento del equipamiento y sobre señales de electroencefalograma en general. También podemos encontrar toda la información del desarrollo y montaje del equipo y el software asociado al mismo.

Para favorecer la transmisión del proyecto y su desarrollo por el usuario, los creadores del proyecto facilitan la compra de las distintas placas de circuito impreso a través de Olimex, así como los electrodos necesarios para su funcionamiento.

Figuras 2.19 Partes del proyecto disponibles en Olimex.

-Arduino

Figura 2.20 Logo de Arduino.

El proyecto Arduino se creó en el Instituto Italiano de Diseño Interactivo Ivrea por Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino y David Mellis. En su diseño se buscaba crear una plataforma de desarrollo de proyectos basada en hardware y software flexible. Desde su creación en el 2005, se ha convertido en el centro de múltiples desarrollos de proyectos DIY (*Do It Yourself* o *Hazlo Tu Mismo*) debido a su fácil utilización.

Las múltiples placas de circuito integrado Arduino se basan en microcontroladores de la marca Atmel. En concreto, la tarjeta Arduino Duemilanove tiene las siguientes características:

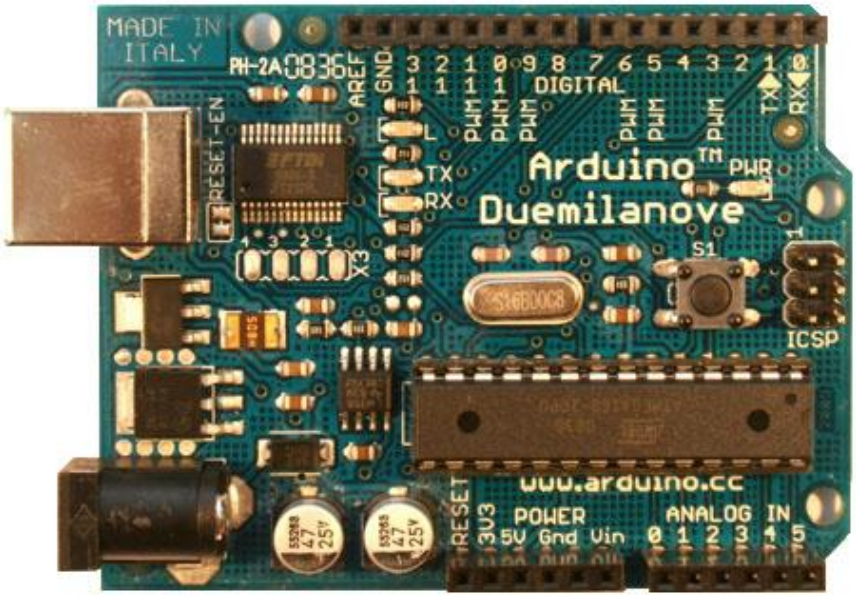


Figura 2.21 Arduino Duemilanove

-Procesadores ATmega168 o ATmega328.
-6 entradas analógicas.
-14 pines de entrada/salida digitales (6 se pueden utilizar como PWM).
-Cristal oscilador de 16 Mhz.
-Conexión USB.
-Contiene bootloader.
-Protección contra sobretensiones de USB.

La programación para Arduino se realiza a través de un software propio de programación en C, con multitud de ejemplos útiles para comprender el funcionamiento del dispositivo. La carga del firmware generado se lleva a cabo sin necesidad de ningún dispositivo de programación externo, ya que Arduino cuenta con un cargador de arranque (bootloader) que permite la grabación del programa simplemente conectando la placa al ordenador mediante el puerto USB.

Debido al diseño de la placa de circuito impreso de Arduino, que permite la conexión de otras placas sobre ella misma mediante pines, han proliferado en el mercado distintos diseños de placas que aumentan las posibilidades de Arduino.

Figura 2.22 Distintas placas de expansión de Arduino.

Como se puede observar en la figura, encrontramos placas de expansión de carga solar, de conexión a Ethernet, ect. Esta característica de Arduino, junto con la gran comunidad que respalda y utiliza el proyecto, así como la gran facilidad de adquisición en el mercado, ha hecho de Arduino la herramienta a utilizar por múltiples aficionados a la electrónica.

-Pinguino

Figura 2.23 Logo de Pinguino

Tras el éxito logrado por Arduino , ha aparecido recientemente en la red multitud de plataformas electrónicas de desarrollo de hardware libre con características similares, como por ejemplo Pinguino. Esta plataforma tiene la peculiaridad de que los microcontroladores utilizados son de la marca Microchip (más concretamente el PIC18F2550 ,PIC18F4550, PIC18F26J50 de 8 bits, y el PIC32MX250F128B, de 32 bits) .

Al contrario que Atmel, Microchip ya incorpora la interfaz USB dentro del microcontrolador, por lo que abarata y facilita la reproducción del diseño.

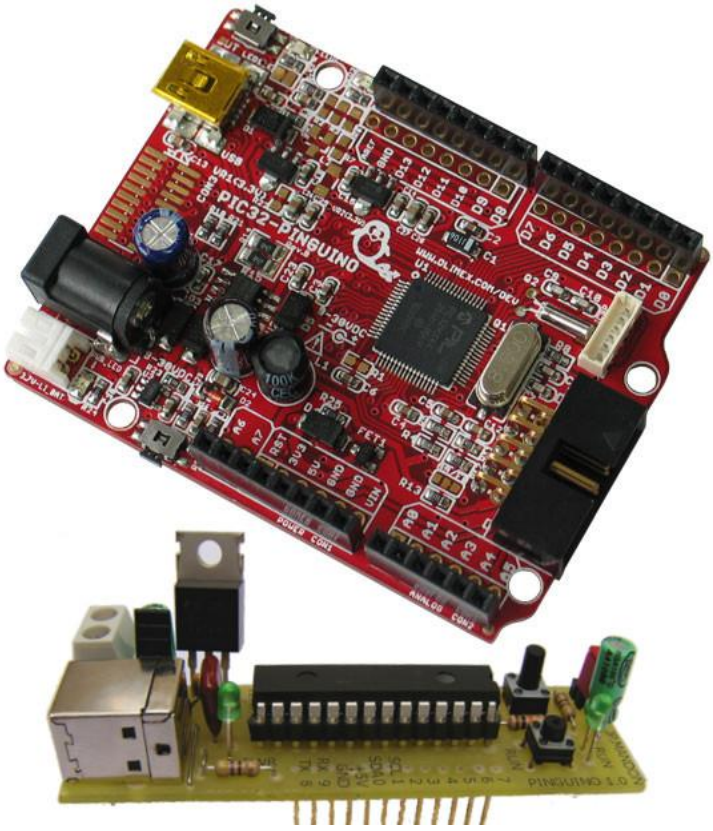


Figura 2.24 Pinguino versión 32 bits y versión de 8 bits

Aunque pinguino todavía no cuenta con una gran comunidad que lo respalde debido a su reciente aparición en la red, empieza a abrirse hueco en el mercado del hardware libre. Pinguino no tiene placas de expansión propias como le ocurre a Arduino, pero esto no es ningún problema, ya que es prácticamente compatible con casi todas las placas de expansión de Arduino.

Pinguino se está expandiendo rápidamente por Sudamérica, debido a que es mucho mas sencilla la adquisición de microcontroladores de Microchip que los fabricados por Atmel en esta zona, pudiendo desbancar en un futuro próximo a Arduino como placa más utilizada para desarrollo de proyectos electrónicos.

-Raspberry Pi

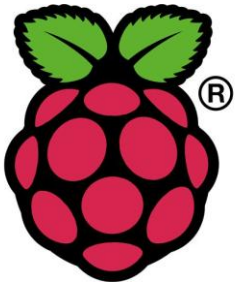


Figura 2.25 Logo de Raspberry Pi

A diferencia de Arduino y Pinguino, Raspberry Pi no está basado en microcontroladores, sino que basa su núcleo en microprocesadores tipo ARM (Advanced RISC Machine), usados comúnmente en dispositivos móviles debido a su bajo consumo energético y su alto rendimiento.

Raspberry Pi es, mas que una placa de desarrollo de proyectos hardware, una pequeño ordenador, en el cual podemos acceder fácilmente a pines de entrada/salida. Aunque soporta distintos tipos de sistemas operativos, el más utilizado debido a su versatilidad es Raspbian (una distribución Debian enfocada para Raspberry Pi).

Actualmente existen en el mercado dos Raspberry Pi, la versión de 256 Mb de memoria y la versión de 512 Mb, aunque ambas poseen el siguiente hardware:

- Conexión Ethernet.
- Lector de tarjetas MicroSD (almacenamiento masivo).
- Puertos USB V 2.0.
- Salida de vídeo HDMI.
- Salida de vídeo compuesto (RCA).
- Salida de audio (mini jack).
- Pines de entrada/salida accesibles.
- Puerto micro USB para alimentación.
- Buses de expansión.

Figura 2.26 Raspberry Pi

La versatilidad del sistema operativo permite al usuario utilizar la Raspberry Pi como Media Center, como servidor FTP de discos duros, o simplemente le permite experimentar con el sistema. A pesar de su corta vida en el mercado, existen multiples proyectos desarrollados sobre Raspberry Pi y cada día surgen nuevos gadgets y aplicaciones para el dispositivo, como cámaras web incluidas directamente en los buses de expansión de la placa.

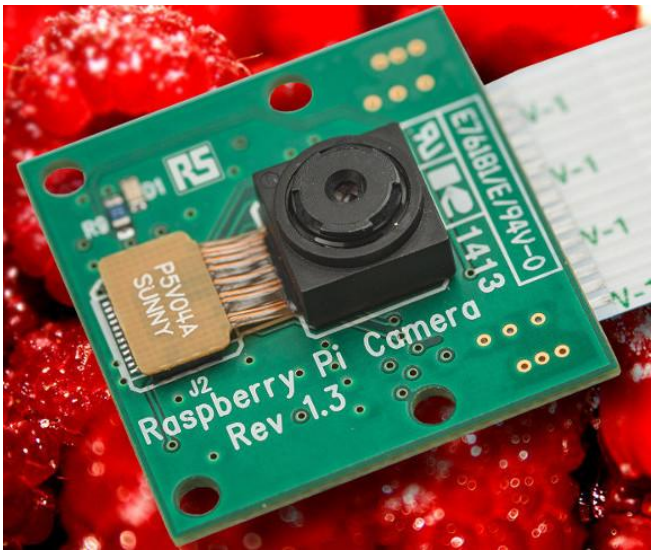


Figura 2.27 Módulo de cámara de Raspberry Pi.

3. Materiales y métodos.

3.1 Bus Serie Universal

El bus serie universal, comúnmente conocido como USB, surgió en la década de los 90 debido a la necesidad de aumentar la velocidad en las comunicaciones entre dispositivos.

Antes de que se desarrollara el USB, los ordenadores tan sólo disponían de puerto serie RS-232 o el famoso puerto paralelo. Estos puertos estaban muy extendido en el mundo de la informática y de la industria, siendo posible encontrar todavía equipos y maquinaria industrial compatibles con estos estándares de comunicación.

Figura 3.1 Puertos paralelo y serie.

El éxito de estos puertos es debido a que sus protocolos son fáciles de entender y de implementar y que en el momento de su creación solucionaban la mayoría de los problemas. Debido a esto, la extensión de estos puestos en los dispositivos fue muy rápida, y lo podíamos encontrar en la mayoría de los periféricos (ratones, modems, impresoras, etc.).

Como consecuencia de que estos puertos ya estaban asentados en la mayoría de los dispositivos, se retardó la creación del bus serie universal, debido al coste que supondría tener que adaptar todos los dispositivos a este nuevo formato. Fue tan sólo cuando las necesidades de velocidad de comunicación entre dispositivos era insostenible cuando se impulsó la creación del USB.

Tras su lanzamiento, los tres puertos estuvieron conviviendo hasta que con la aparición de Windows 98, que mejoraba la compatibilidad para dispositivos USB. Este hecho junto con la acuñación del término Plug and Play, marcó el descenso de la utilización de los puertos paralelo y serie, aunque a día de hoy todavía se sigue utilizando el puerto serie en muchos dispositivos debido a su facilidad de implementación y a los adaptadores USB-RS232, que permiten su utilización en los dispositivos que carecen de este bus.

Desde que en Enero de 1996 saliera al mercado la primera versión del USB, han salido distintas versiones del dispositivo:

USB 1.0
Primera versión de USB. Versión limitada y con errores, poco extendida en su lanzamiento. Hubo que esperar al lanzamiento de Windows 98 para que este bus fuera popular.

USB 1.1
Similar al USB 1.0. Añade un nuevo tipo de transferencia de datos (interrupt OUT).

USB 2.0
Vio la luz en Abril del 2000. En esta versión del bus se incluye la nueva alta velocidad de 480 Mbps. Totalmente compatible con los dispositivos USB 1.0, es capaz de ajustar su velocidad entre la velocidad de éste y la nueva incluida en el.

USB 3.0
De reciente aparición, incluye una nueva velocidad de hasta 5 Gbps. Incluye además la novedad de que la conexión es full duplex, en vez de half duplex (modelos 1.0,1.1 y 2.0).

USB On-The-Go
Por definición del funcionamiento del protocolo USB estándar, los dispositivos tan sólo pueden comunicarse con el host y viceversa, por lo que no puede existir la comunicación entre dos dispositivos. Para solventar la necesidad de comunicación entre dos dispositivos, se diseñó USB On-The-Go. Este tipo de configuración permite poder conectar varios dispositivos entre sí, con bastantes limitaciones. Por ejemplo, podríamos conectar una cámara de fotos digital a una impresora e imprimir las fotos directamente, entre muchos otros usos.

Wireless USB
Permite la comunicación USB sin cables, aunque se encuentra algo restringido en cuanto a velocidad (hasta 480 Mbps) y a tecnología (suelen apoyarse en otros métodos de comunicación inalámbrica, como zigbee o wifi).

La comunicación estándar en el USB se debe realizar entre un host (PC) y uno o mas dispositivos (ratón, teclado, impresora, etc), permitiendo conectar hubs para incluir mas dispositivos al mismo bus. La topología del bus es en estrella por niveles y parte siempre desde el host, con un máximo de 5 niveles de hubs encadenados.

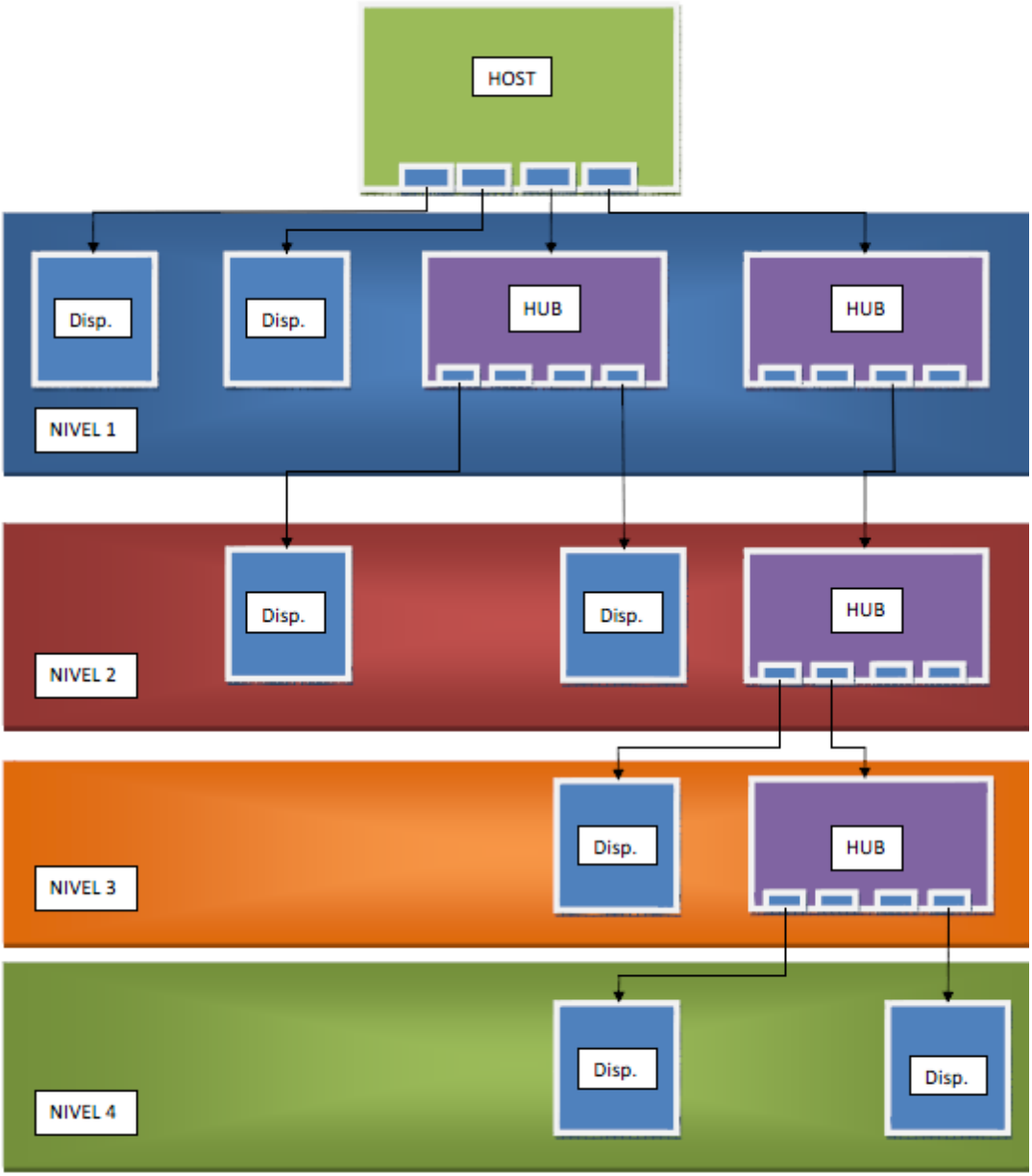


Figura 3.2 Arquitectura de red de estrella por niveles usada por el USB

La comunicación USB se inicia automáticamente cuando conectamos un dispositivo en el host. En este momento, el host detecta el dispositivo conectado y comienza un intercambio de información entre el dispositivo y el host necesario para el correcto funcionamiento del dispositivo. A este proceso se le conoce como enumeración. En la enumeración el host detecta la velocidad del dispositivo, le asigna una dirección y recoge información adicional. Tras la enumeración, si desconectamos el dispositivo del host, éste lo tendrá en cuenta y quedará a la espera de volver a enumerar cualquier dispositivo que se conecte en ese puerto.

Dentro de la información adicional que es enviada al host por el dispositivo encontramos los descriptores. Los descriptores indican que comportamiento va a tener el dispositivo, y siguen la siguiente estructura:

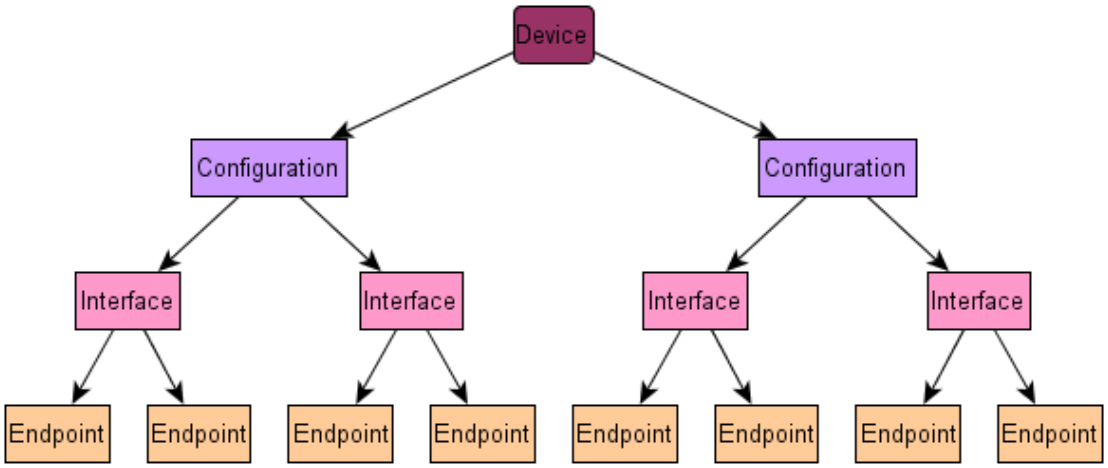


Figura 3.3. Tipos de descriptores y relación entre ellos.

Se puede observar en la figura como encontramos cuatro tipos de descriptores distintos: El descriptor de dispositivo, el descriptor de configuración, el descriptor de interfaz y el descriptor de punto final o endpoint.

El descriptor de punto final le proporciona al host los tipos de endpoints que va a tener el dispositivo (tipo bulk, asíncrono o interrupción), la dirección (IN o OUT) de estos endpoints, el intervalo de votación y el número máximo de paquetes. Estos dependen de un descriptor de interfaz y todos requieren un descriptor salvo el descriptor 0, ya que es el punto final de control por defecto y, por norma, no necesita de descriptor.

Los descriptores de interfaz indica la cantidad de endpoints que va a tener el dispositivo y especifica las interfaces de clase soportadas por el dispositivo.

Los descriptores de configuración le proporcionan al host la información necesaria sobre los requisitos de alimentación del dispositivo y el número de interfaces de las que dispone.

El host tan sólo leerá un descriptor de configuración a la vez, lo que permite que un mismo dispositivo pueda comportarse de manera distinta configurándolo previamente antes de conectarlo al host.

El ejemplo más claro podemos encontrarlos en los teléfonos Android, que pueden conectarse al PC como dispositivo de almacenamiento masivo o como módem, utilizando la misma interfaz USB y cambiando tan sólo su configuración.

Por último, el descriptor de dispositivo provee al host de información más global del dispositivo, como son el Vendor ID, el Product ID, el número de serie, la clase del dispositivo, el número de configuraciones, ect. Por norma, un dispositivo tan sólo tiene un descriptor de dispositivo.

El proceso de enumeración de dispositivos se desarrolla según el siguiente esquema:

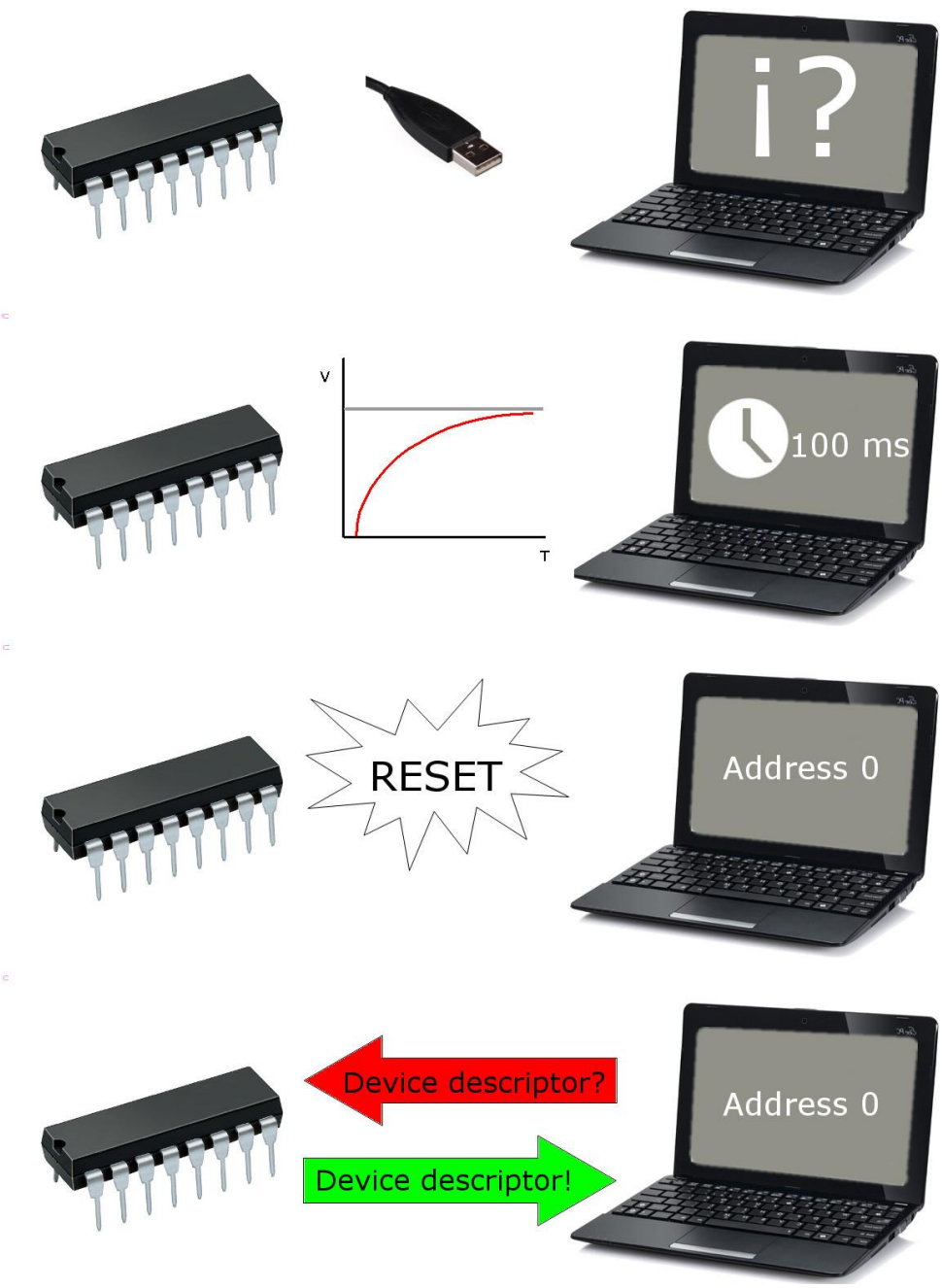


Figura 3.4 Esquema gráfico de la enumeración de un dispositivo USB 1

El proceso comienza cuando se conecta el dispositivo por primera vez al host. Éste detecta que hay cambios de nivel en el bus de comunicación y espera un mínimo de 100ms hasta que la tensión en el dispositivo se ha estabilizado correctamente.

Tras esperar los 100 ms, el host manda una señal de reset (forzando el bus a nivel bajo más de 10ms) al dispositivo y le asigna la dirección 0, (dirección específica para realizar el proceso de enumeración). Tras el reinicio del dispositivo, el host pide el descriptor de dispositivo y vuelve a reiniciarlo, asignándole una dirección comprendida entre 1 y 128 que esté libre.

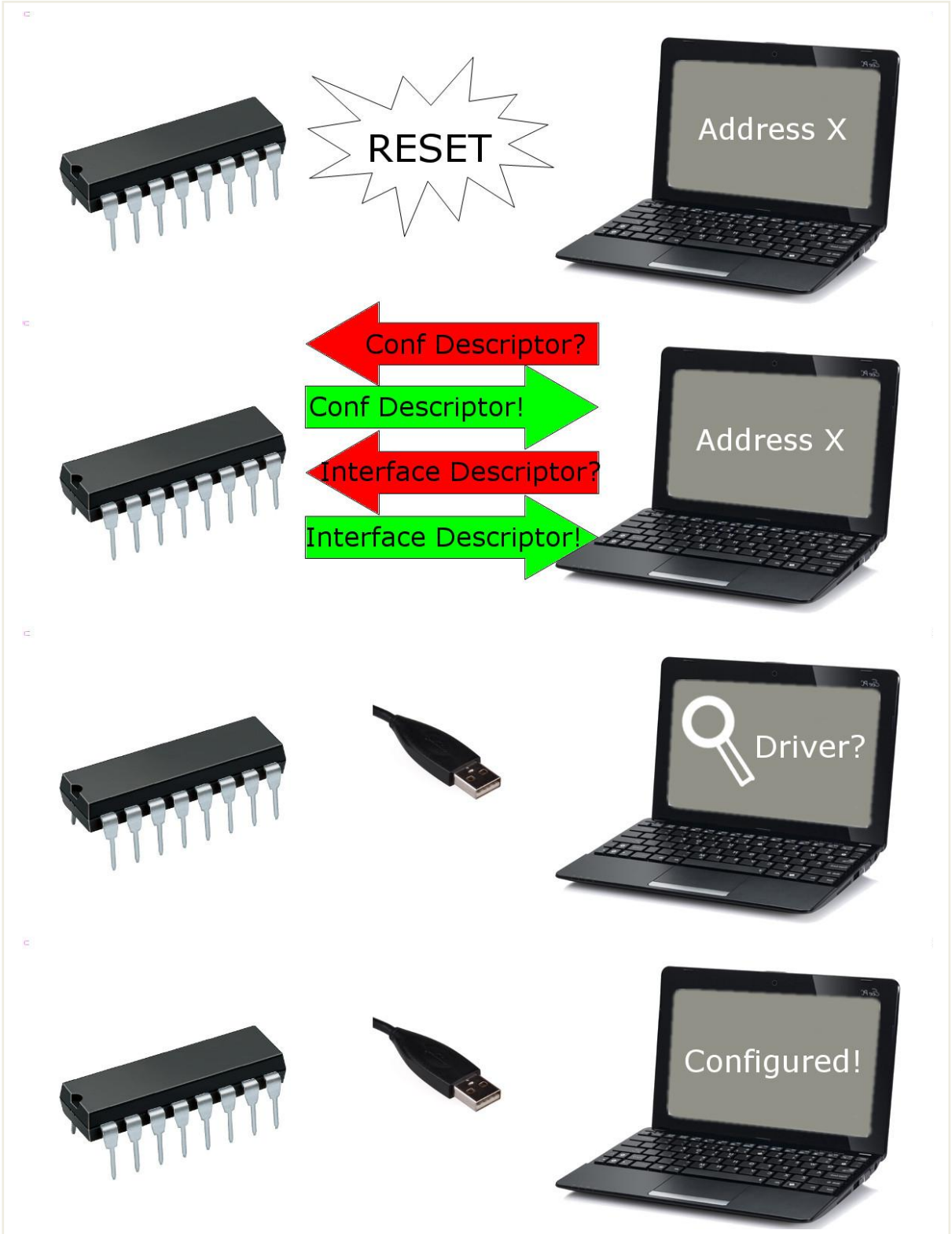


Figura 3.5 Esquema gráfico de la enumeración de un dispositivo USB 2

Tras el reset, el host empieza a pedirle los descriptores al dispositivo sistemáticamente. Cuando el host acaba la petición de los descriptores, busca un driver para el dispositivo que funcione correctamente y se lo asigna, terminando la configuración correctamente. En el caso de no encontrar un driver funcional, el sistema operativo correspondiente enviará al usuario un mensaje de error.

Terminada la enumeración, tanto el dispositivo como el host ya están listos para intercambiar datos. Podemos encontrar principalmente cuatro tipos de flujo de datos en la comunicación USB:

Control
Todos los dispositivos utilizan este tipo de flujo de datos, al menos en la enumeración. Se puede utilizar por cualquier tipo de dispositivo para cualquier propósito concreto.

Síncrono
Para tipos de dispositivos que necesitan del envío de información periódica, como tarjetas de sonido o capturadoras de vídeo.

Bulk
Este tipo de flujo de datos está preparado para el volcado de gran cantidad de información. Muy común en unidades de almacenamiento.

Interrupt
Ideal para dispositivos que envían datos en momentos puntuales. Utilizado mayoritariamente por teclados, ratones, y dispositivos de acceso al PC.

3.2 PIC18F14K50 y Entorno Low Pin Count

El microcontrolador que se ha elegido para la realización del proyecto es el PIC18F14K50, de la casa Microchip. Las características principales del microcontrolador son las siguientes:

Compatibilidad con USB
- USB 2.0.
- Soporta transferencias del tipo Control, Interrupción, Asíncrona y Bulk.
- 256 bytes de RAM de doble acceso para USB.
- Detección de conexión por USB mediante D+/D-.

CPU tipo RISC de altas prestaciones
-Arquitectura optimizada para compilación en C.
- Instrucciones extendidas opcionales para la optimización del código.
- 256 bytes de memoria EEPROM.

Estructura de oscilador flexible.
-Bloque oscilador interno de 16 Mhz.
-Permite osciladores extrenos de hasta 48 Mhz.

-4x Phase Lock Loop (PLL).
-Oscilador secundario utilizando Timer1 a 32Khz.

Características Especiales del Microcontrolador
- Auto programable bajo Software.
- Programación por ICSP (In-Circuit Serial Programming).
- Alimentación hasta 5.5 V (modelo PIC18F14K50).
- Alimentación desde 1.8V a 3.6V (modelo PIC18LF14K50).

Características Analógicas
- Módulo Analógico/Digital (ADC).
- Resolución de 10 bits, 9 canales.
- Capacidad de auto-adquisición.
- Permite la convesión en modo sleep.
- Referencia de voltaje interna.
- Comparador dual.
- Comparación rail-to-rail.
- Programable (% de VDD), con 16 posiciones.

Periféricos
- 14 pines de entrada/salida, 1 pin de solo entrada.
- Pines de alta corriente (25 mA).
- 7 resistencias de pull-ups programables.
- 3 interrupciones externas programables.
- Módulo ECCP (Enhances Capture/Compare/PWM).
- Hasta 4 salidas PWM.
- Módulo MSSP (Master Syncronous Serial Port).
- Protocolo SPI de 3 hilos.
- Protocolo I2C con modos Maestro y Esclavo.

-Módulo EUSART (Enhanced Universal Synchronous Asynchronous Receiver Transmitter).
-Soporta RS-232, RS-485 y LIN 2.0.

Para la toma de contacto con el microcontrolador y el diseño del programa se ha utilizado el kit de desarrollo proporcionado por Microchip para el PIC18F14K50, el Low Pin Count USB Development Kit.

El kit contiene dos placas de circuito impreso, una implementada con componentes, lista para funcionar, y otra sin poblar, para realizar la implementación que prefiera el usuario. El kit también contiene un programador Pickit 2 y una pequeña placa de programación del microcontrolador, así como un CD con la información necesaria para empezar a utilizar la placa de desarrollo.

Figura 3.6 Kit de desarrollo Low Pin Count USB Developement Kit

La placa implementada consta de:

1- Conector USB tipo mini-B.
2- Conector de corriente regulada de 5 V.
3- Selector de alimentación, bien vía USB o alimentación externa por el conector 2.
4- Conexión de programación del Pickit 2.
5- LEDs conectados a las patillas del puerto C (RC0,RC1,RC2,RC3).
6- Conexión del analizador serie PICkit.
7- Chip de adaptación para RS232 MAX3232.
8- Conector RS-232.
9- Numero de serie .
10- Potenciometro .
11- Jumper de conexión de VUSB al microcontrolador.
12- Pulsador.
13- Cristal de 12 Mhz.
14- Microcontrolador PIC18F14K50.
15- Área de prototipado.
16- Cabezal de placa de expansión PICtail.
17- Expansión SSOP.

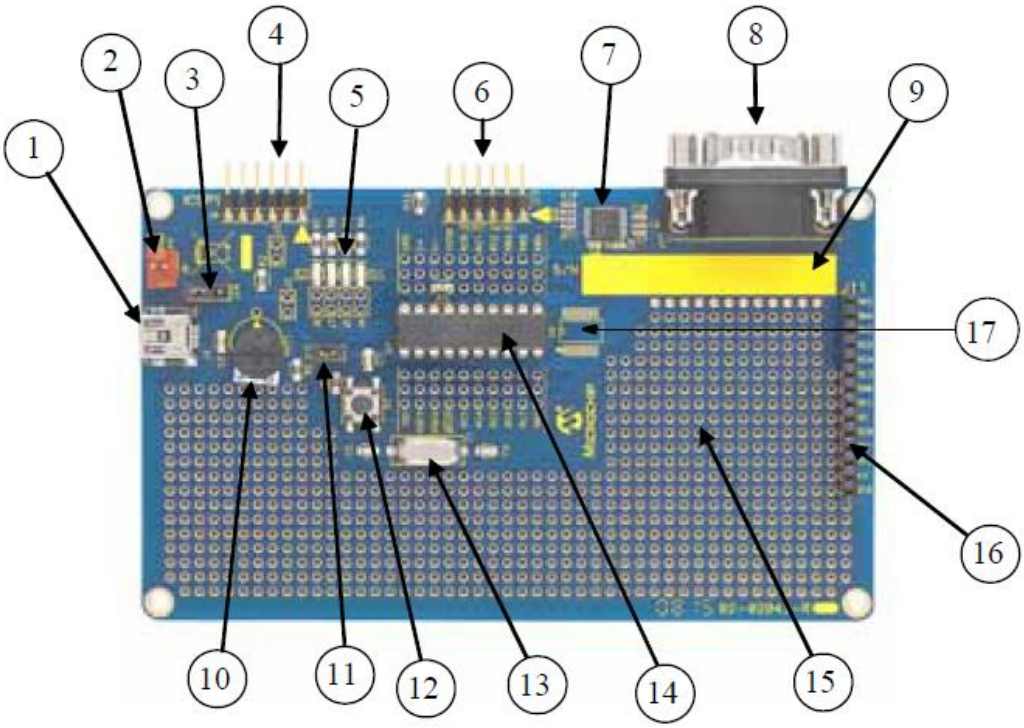


Figura 3.7 Tarjeta implementada del kit

3.3 SDK y lecciones

En la guía de uso que proporciona Microchip para el kit de desarrollo se encuentran distintas lecciones que permiten realizar la primera toma de contacto con el hardware y el aprendizaje práctico de la utilización del USB.

A la hora de la disposición de realizar las lecciones, y tras un estudio previo del funcionamiento del USB, se ha tenido que realizar la descarga de las SDK`s de trabajo de Microchip, conocidas comúnmente como Microchip Solutions. Para la ejecución tanto de las prácticas como del proyecto se ha utilizado la Microchip Solutions v2010-08-04.

Dentro de los múltiples proyectos semi-desarrollados que se pueden encontrar dentro de las Microchip Solutions, existen un gran número relacionadas con USB (dispositivos de audio diversos, dispositivos de interfaz humana, almacenamiento masivo, ect.). Dentro de cada uno de ellos se puede encontrar un proyecto completo de MPLAB y C18 listo para abrir para cada uno de los dispositivos compatibles con este tipo de interfaz.

3.3.1 MPLAB IDE y C18

Para el desarrollo del proyecto se ha utilizado el software proporcionado por Microchip para la programación de sus microcontroladores, MPLAB IDE, con la API

C18, para compilar en lenguaje C para los microcontroladores de la serie 18. Las versiones de software utilizadas en el proyecto han sido MPLAB IDE V 8.50 y C18 v3.10.

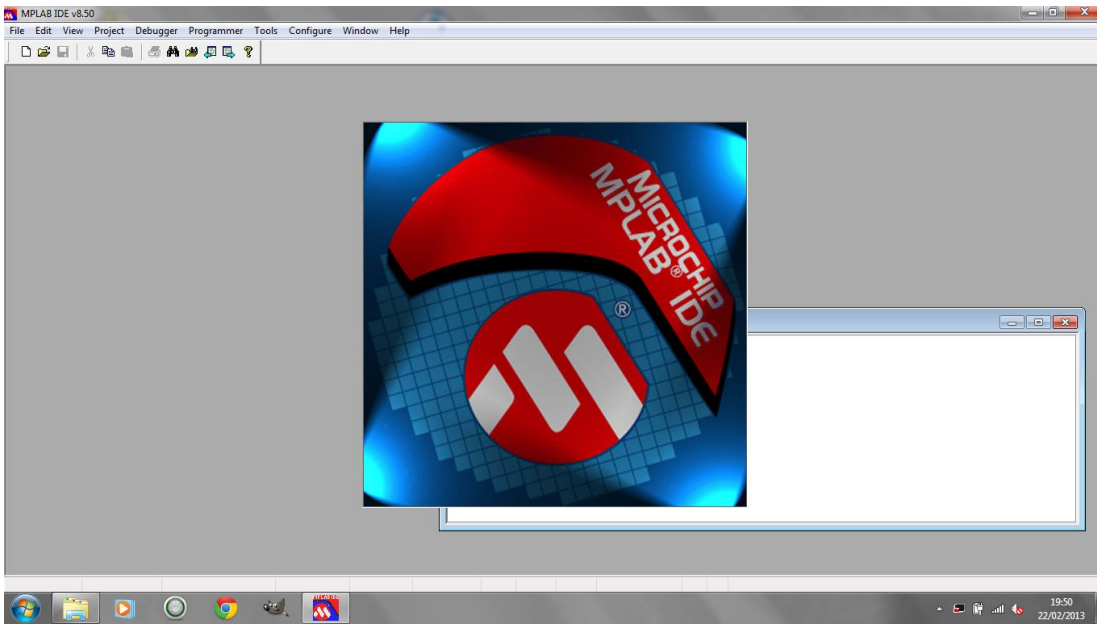


Figura 3.8 - MPLAB IDE V 8.50

3.3.2 Lecciones

Como primera toma de contacto con el entorno de programación y con el microcontrolador, se han realizado los tres primeros ejemplos de la guía de uso del kit de desarrollo.

Estas consisten en la enumeración del dispositivo, la simulación en círculos del ratón y una simulación a un solo botón de un teclado. Con éstas prácticas se pretende aprender a utilizar las librerías que proporciona Microchip para el desarrollo de firmware con compatibilidad USB, ya que se utilizarán para el desarrollo del proyecto (comúnmente conocidas como USB Stack).

3.3.2.1 Enumeración

En este ejemplo se pretende que el ordenador reconozca al microcontrolador en la fase de enumeración y la familiarización por parte del programador de la configuración del descriptor. Para la creación del proyecto se han seguido los siguientes pasos:

1- Se ha creado el framework de trabajo, añadiendo en un nuevo proyecto los archivos necesarios para el desarrollo del ejemplo, así como se han cambiado las rutas de las librerías de MPLAB dentro de sus menús de configuración, las ruta del compilador, el linker y la carpeta de salida. Parte de los archivos agregados al proyecto proceden de las librerías de aplicación de Microchip (concretamente de Microchip Solutions v2010-08-04).

Tras agregar los archivos al proyecto, se recibió un error de compilación en el entorno de programación y se decidió repetir la práctica, cambiando la fuente de cuatro archivos que estaban incluidos en el CD del kit por los incorporados en las librerías de aplicación de Microchip (enumeration.c, usb_descriptors.c, HardwareProfile.h y usb_config.h), ya que se había llegado a la conclusión de que el error de debía a un problema de compatibilidad de estos archivos.

Tras la realización del cambio de estos archivos y la compilación del programa, éste terminó de compilar satisfactoriamente.

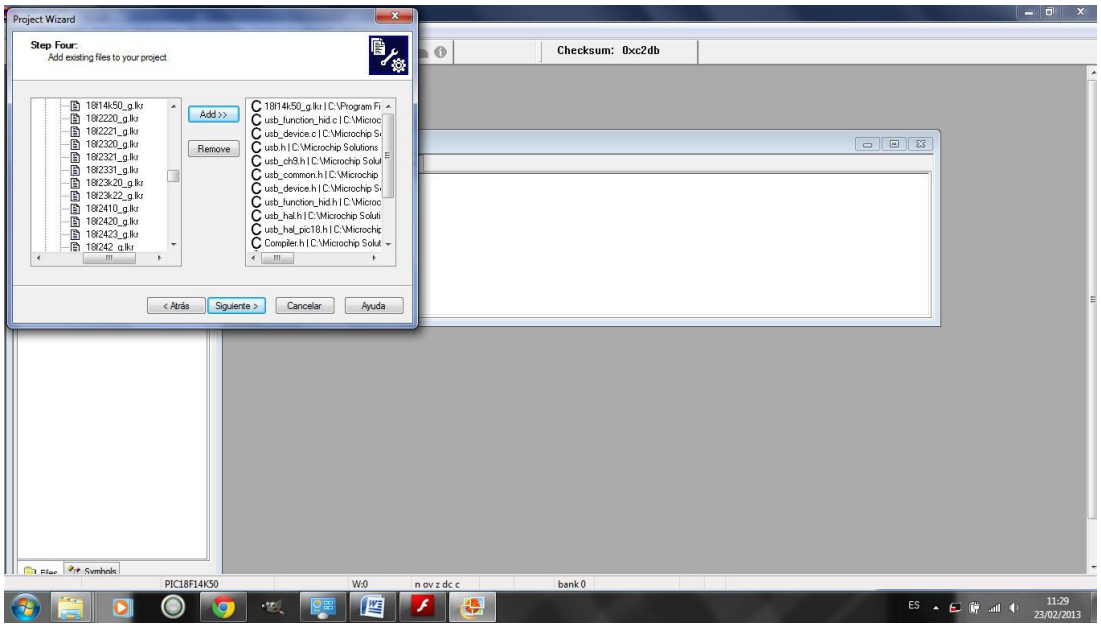


Figura 3.9 Añadiendo los archivos necesarios para el proyecto.

2- Tras la creación del framework se ha modificado el archivo usb_descriptor.h , insertando el código incluido en la guía.

usb_descriptors.h

En este archivo se han insertado los códigos del descriptor del dispositivo, el de configuración, el de clase, el de interfaz y el de informe, así como
las cadenas de caracteres del dispositivo y del fabricante.

Tras realizar este paso, se ha compilado el proyecto sin ningún error, se ha cargado el firmware generado en el microcontrolador y se ha conectado este al PC, comprobando que el dispositivo era reconocido correctamente.

```
ROM       struct{BYTE      bLength;BYTE      bDscType;WORD
string[25];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{
     //ADD MANUFACTURER STRING DESCRIPTOR CODE HERE
     //INICIO DEL PEGADO DEL CÓDIGO
'M','i','c','r','o','c','h','i','p',
' ','T','e','c','h','n','o','l','o','g','y',
' ','I','n','c','.'

}};
//Product string descriptor

     //FIN DEL PEGADO DEL CÓDIGO
ROM       struct{BYTE      bLength;BYTE      bDscType;WORD
string[22];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{
     //ADD PRODUCT STRING DESCRIPTOR CODE HERE
     //INICIO DEL PEGADO DEL CÓDIGO

'M','o','u','s','e',
' ','E','n','u','m','e','r','a','t','i','o','n ',
' ','D','e','m','o'
}};

     //FIN DEL PEGADO DEL CÓDIGO
```

```
//Class specific descriptor - HID mouse
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
    {
          //ADD REPORT DESCRIPTOR CODE HERE
          //INICIO DEL PEGADO DEL CÓDIGO

0x05, 0x01, /* Usage Page (Generic Desktop)*/
0x09, 0x02, /* Usage (Mouse)*/
0xA1, 0x01, /* Collection (Application)*/
0x09, 0x01, /* Usage (Pointer)*/
0xA1, 0x00, /* Collection (Physical)*/
0x05, 0x09, /* Usage Page (Buttons) */
0x19, 0x01, /* Usage Minimum (01)*/
0x29, 0x03, /* Usage Maximum (03)*/
0x15, 0x00, /* Logical Minimum (0)*/
0x25, 0x01, /* Logical Maximum (0)*/
0x95, 0x03, /* Report Count (3)*/
0x75, 0x01, /* Report Size (1)*/
0x81, 0x02, /* Input (Data, Variable, Absolute)*/
0x95, 0x01, /* Report Count (1)*/
0x75, 0x05, /* Report Size (5)*/
0x81, 0x01, /* Input (Constant) ;5 bit padding */
0x05, 0x01, /* Usage Page (Generic Desktop)*/
0x09, 0x30, /* Usage (X)*/
0x09, 0x31, /* Usage (Y)*/
0x15, 0x81, /* Logical Minimum (-127)*/
0x25, 0x7F, /* Logical Maximum (127)*/
0x75, 0x08, /* Report Size (8)*/
0x95, 0x02, /* Report Count (2)*/
0x81, 0x06, /* Input (Data, Variable, Relative)*/
0xC0, 0xC0

     //FIN DEL PEGADO DEL CÓDIGO
     }
};/* End Collection,End Collection            */
```

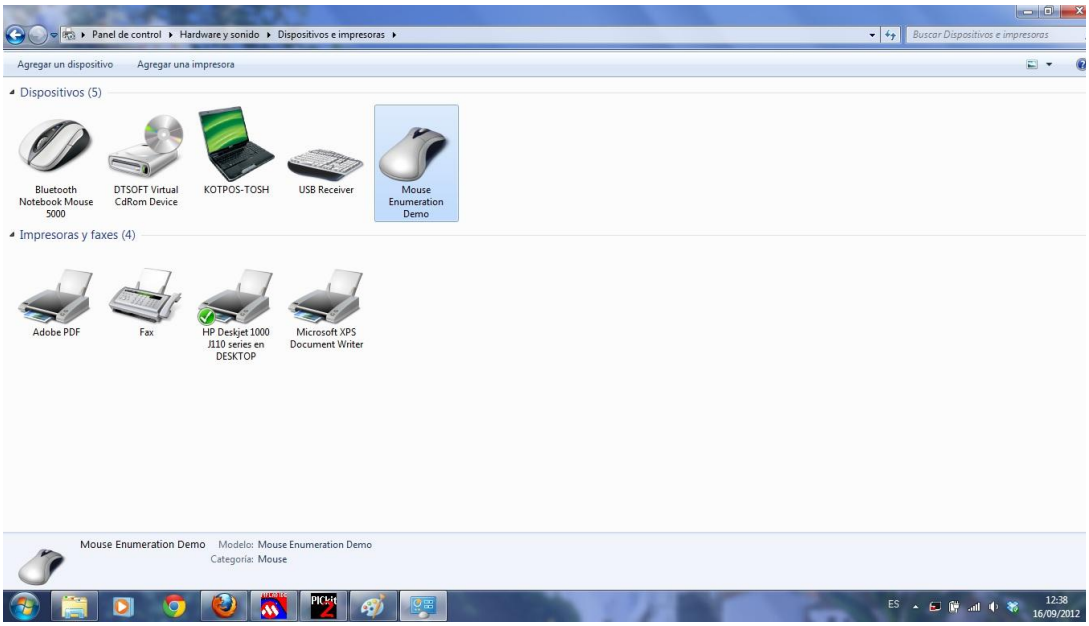


Figura 3.10 Resultado tras la enumeración del dispositivo.

3.3.2.2 Ratón en círculos.

En este ejemplo se pretende configurar el microcontrolador para simular el movimiento del ratón en círculos, así como la familiarización del programador con los dispositivos de interfaz humana USB de tipo ratón.

Para el desarrollo de la lección se han seguido los siguientes pasos:

1- Se ha procedido a la preparación del framework de trabajo de manera similar a como se realizó en la práctica anterior, con la excepción de la carga de los archivos referentes a la segunda práctica. Igual que en el caso anterior, se han escogido dichos archivos de las librerías de aplicación en vez de las proporcionadas en el CD.

2- Tras este paso, se ha procedido a modificar el archivo mouse.c, incorporando el código que proporciona Microchip en la guía y cambiando distintas líneas para el correcto funcionamiento del programa.

mouse.c

En este archivo se ha descomentado la línea de la definición de la función *Emulate_Mouse(),* así como las líneas *emulate_mouse=true* de la función *UserInit()* y *emulate_mode=!emulate_mode* de la función *ProcessIO().*

Tras este paso se ha incluido el código de la figura dentro de la función *Emulate_Mouse().*Con éste último paso el proyecto reconocerá la función y será capaz de ejecutarla.

```
void Emulate_Mouse(void)
{
     //ADD EMULATE MOUSE CODE HERE
     //INICIO DEL PEGADO DEL CÓDIGO
if (emulate_mode == TRUE)
{
//go 14 times in the same direction before changing
if (movement_length > 14)
{
buffer[0] = 0;
buffer[1] = dir_table[vector & 0x07];
//X-Vector
buffer[2] = dir_table [(vector+2) & 0x07];
//Y-Vector
// go to the next direction in the table
vector++;
//reset the counter for when to change again
movement_length = 0;
}//end
if (movement_length > 14)
}
else
{
//don't move the mouse
buffer[0] = buffer[1] = buffer[2] = 0;
}
if(HIDTxHandleBusy(lastTransmission) == 0)
{
//copy over the data to the HID buffer
hid_report_in[0] = buffer[0];
hid_report_in[1] = buffer[1];
hid_report_in[2] = buffer[2];
//Send the 3 byte packet over USB to the host.
lastTransmission = HIDTxPacket(HID_EP,
(BYTE*)hid_report_in,0x03);
//increment the counter to change the data sent
movement_length++;
}
}//end Emulate_Mouse
     //FIN DEL PEGADO DEL CÓDIGO
```

El código insertado en la función Emulate_Mouse() se ha programado siguiendo el siguiente flujograma:

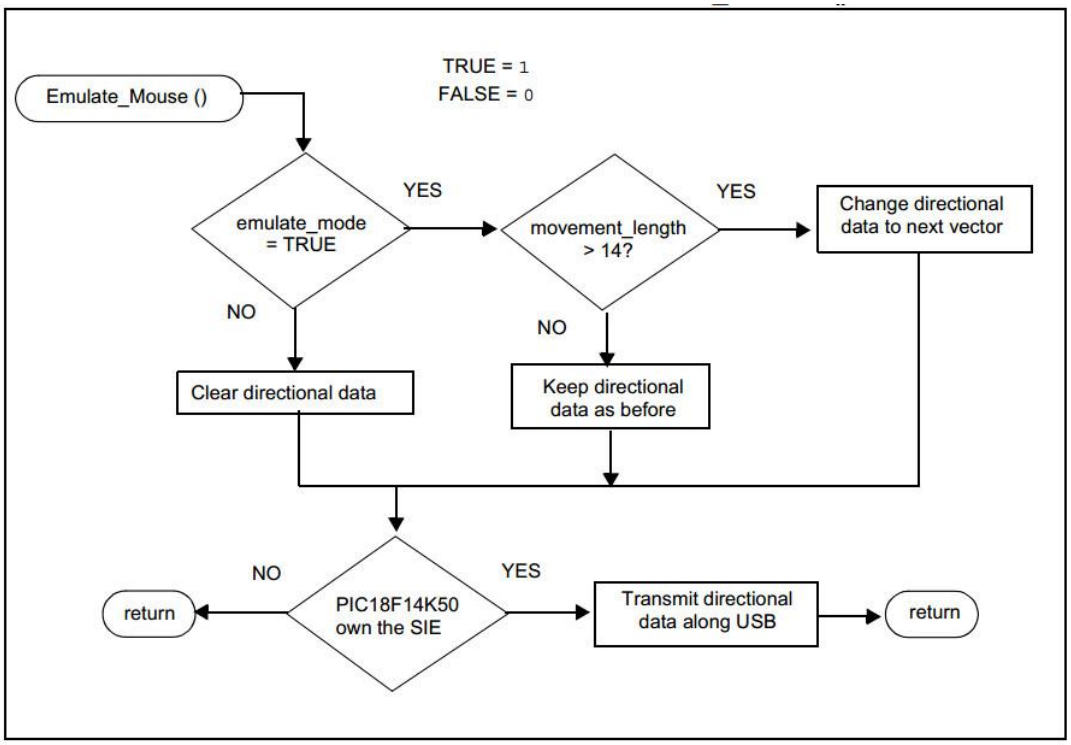


Figura 3.11 Flujograma del programa.

En primer lugar, el programa comprobará que el modo de emulación del ratón está activo. En el caso de que este modo esté desactivado borrará los datos de dirección y, si el dispositivo está listo para enviar los datos, los enviará a través del USB y, en el caso de que el dispositivo no esté preparado para enviar los datos, volverá a empezar el ciclo.

Si el modo de emulación esta activo, el programa recorrerá un array de direcciones y cargará los valores de éste en dos variables temporales que posteriormente se cargarán en el buffer de salida. En el caso de que el dispositivo esté preparado para enviar datos por USB, hará lo propio y, si no lo estuviera, volvería a empezar el programa.

Notar que el array de direcciones contiene 14 posiciones y, si el índice que recorre el array fuera superior al número de posiciones, el índice se reseteará, volviendo a cargar el valor inicial.

Tras la inserción del código, se ha compilado con éxito el proyecto y se ha cargado el programa en el microcontrolador, comprobando el correcto funcionamiento del programa.

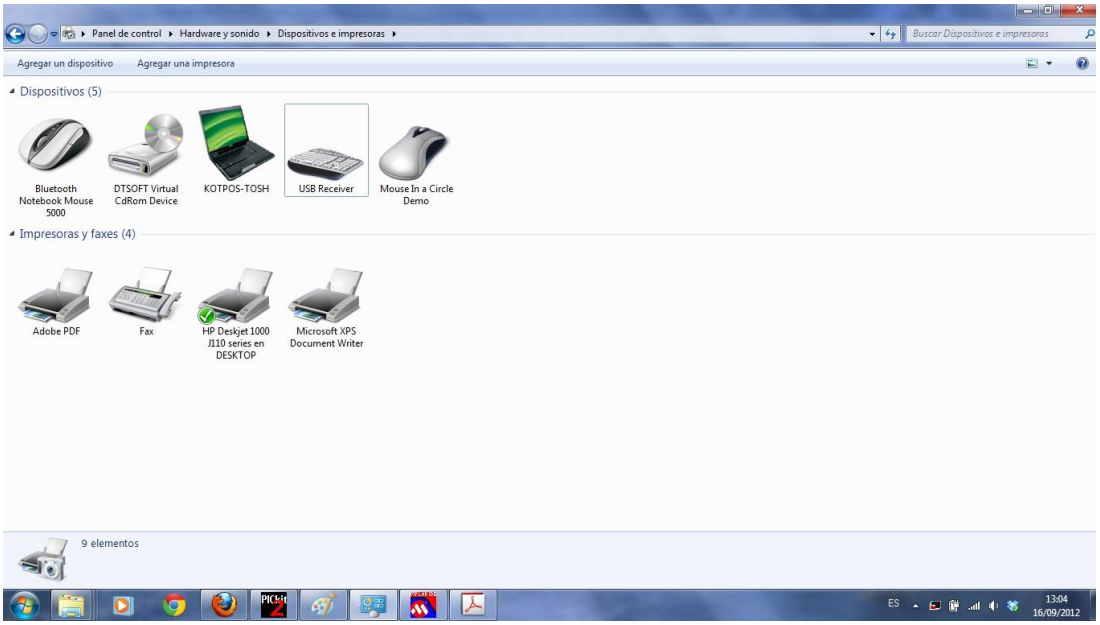


Figura 3.12 Detección del microcontrolador en modo ratón en círculos.

3.3.2.3 Teclado.

En este ejemplo se pretende configurar el microcontrolador para simular el movimiento del ratón en círculos, así como la familiarización del programador con los dispositivos de interfaz humana USB de tipo teclado.

El microcontrolador emulará un teclado de ordenador y enviará una letra por USB cada vez que el botón de reset sea pulsado. El valor que se enviará por el interfaz USB viene dado por un número entre 4 y 29, que corresponden a un valor numérico de los caracteres comprendidos entre la "a" y la "z", ambas inclusive. Este valor se corresponderá con el valor de la lectura del conversor analógico digital del microcontrolador, según como se muestra en el siguiente diagrama de estados:

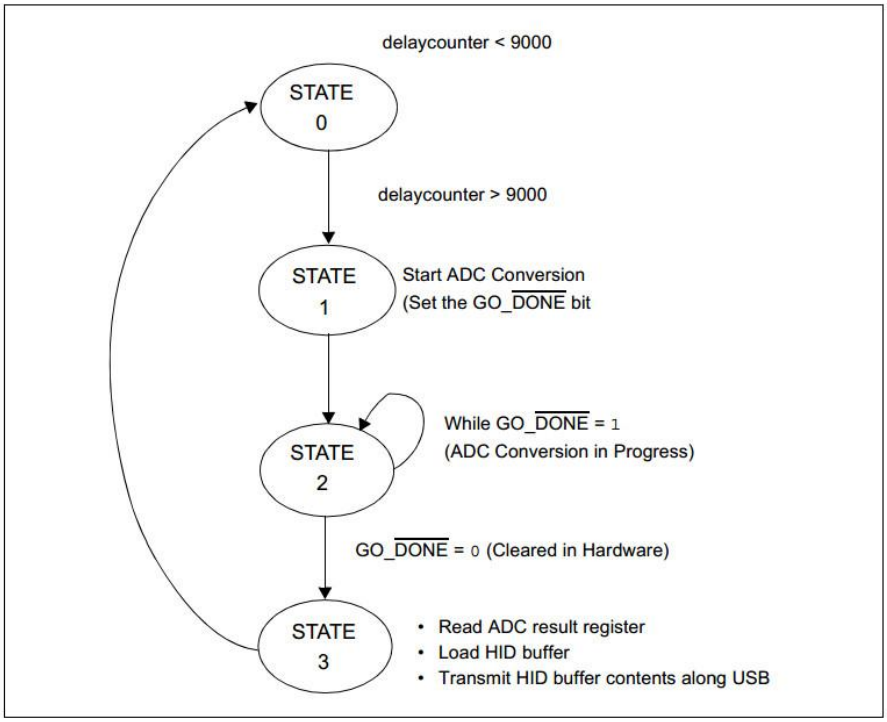


Figura 3.13 Diagrama de estados que refleja el funcionamiento del dispositivo.

En el diagrama se puede observar que, mientras que el delaycounter sea menor que 9000, el programa esperará en el estado 0, cuando esta variable sea superior a 9000, el programa saltará al estado 1 y mandará la orden de inicio de conversión analógico-digital y pasará al estado 2, en el cual quedará a la espera de que se haya terminado de realizar la conversión. Tras este paso almacenará el valor del registro del conversor en el buffer HID y lo enviará a través del puerto USB.

Para la realización de la práctica se han seguido los siguientes pasos:

1- Se ha procedido a la preparación del framework de trabajo de la misma manera que los dos ejemplos anteriores, cargando los archivos referentes a esta práctica desde las librerías de aplicación.

2- Tras este paso, se han modificado diversos archivos insertando código de la guía y modificando el existente, para conseguir el buen funcionamiento del programa. Los archivos modificados han sido los siguientes.

usb_descriptor.c

Se han descomentado las líneas HID_PROTOCOL_KEYBOARD del descriptor de la interfaz y la línea DESC_CONFIG_WORD(63) del descriptor de clase. Se ha añadido el siguiente código extraído de la guía de uso.

```
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
    {

          //ADD REPORT DESCRIPTOR HERE
          //INICIO DEL PEGADO DEL CÓDIGO

0x05, 0x01, // USAGE_PAGE (Generic Desktop)
0x09, 0x06, // USAGE (Keyboard)
0xa1, 0x01, // COLLECTION (Application)
0x05, 0x07, // USAGE_PAGE (Keyboard)
0x19, 0xe0, // USAGE_MINIMUM (Keyboard LeftControl)
0x29, 0xe7, // USAGE_MAXIMUM (Keyboard Right GUI)
0x15, 0x00, // LOGICAL_MINIMUM (0)
0x25, 0x01, // LOGICAL_MAXIMUM (1)
0x75, 0x01, // REPORT_SIZE (1)
0x95, 0x08, // REPORT_COUNT (8)
0x81, 0x02, // INPUT (Data,Var,Abs)
0x95, 0x01, // REPORT_COUNT (1)
0x75, 0x08, // REPORT_SIZE (8)
0x81, 0x03, // INPUT (Cnst,Var,Abs)
0x95, 0x05, // REPORT_COUNT (5)
0x75, 0x01, // REPORT_SIZE (1)
0x05, 0x08, // USAGE_PAGE (LEDs)
0x19, 0x01, // USAGE_MINIMUM (Num Lock)
0x29, 0x05, // USAGE_MAXIMUM (Kana)
0x91, 0x02, // OUTPUT (Data,Var,Abs)
0x95, 0x01, // REPORT_COUNT (1)
0x75, 0x03, // REPORT_SIZE (3)
0x91, 0x03, // OUTPUT (Cnst,Var,Abs)
0x95, 0x06, // REPORT_COUNT (6)
0x75, 0x08, // REPORT_SIZE (8)
0x15, 0x00, // LOGICAL_MINIMUM (0)
0x25, 0x65, // LOGICAL_MAXIMUM (101)
0x05, 0x07, // USAGE_PAGE (Keyboard)
0x19, 0x00, // USAGE_MINIMUM (Reserved (no event
indicated))
0x29, 0x65, // USAGE_MAXIMUM (Keyboard Application)
0x81, 0x00, // INPUT (Data,Ary,Abs)
0xc0
     //FIN DEL PEGADO DEL CÓDIGO
    }
};/* End Collection,End Collection            */
```

keyboard.c

Se han descomentado las líneas ADCON0=0x29, ADCON1 = 0X00 y ADCON2=0x3F de la función UserInit(). Se ha insertado el código de la figura en la zona señalada en el código.

```
void Keyboard(void)
{
     static unsigned char state = 0;    //declare a state
variable for the keyboard state machine
     static unsigned long delaycounter = 0;
     static unsigned char HIDoutput;    //will  store  an
altered version of the ADRESH result

//State machine
                switch (state)
                {
                     //ADD STATE MACHINE CODE HERE
                     //INICIO DEL PEGADO DEL CÓDIGO

//delay to allow the hold capacitor on the ADC to charge
case 0: if(++delaycounter>9000)
{
delaycounter = 0;
state = 1;
}
break;
case 1: ADCON0bits.GO_DONE = 1; //Start an ADC conversion
state = 2;
break;
case 2: if(ADCON0bits.GO_DONE == 0) //Check if conversion
is
//completed
{
state = 3;
}
break;
case  3:  HIDoutput  =  ADRESH>>3;//shift  the  result  in
ADRESH
//left by three
if(HIDoutput<=4) HIDoutput = 4;
if(HIDoutput>=29) HIDoutput = 29;
//Can the SIE transmit?
if((HIDTxHandleBusy(lastTransmission) == 0))
{
//Load the HID buffer
hid_report_in[0] = 0;
hid_report_in[1] = 0;
hid_report_in[2] = HIDoutput;
hid_report_in[3] = 0;
hid_report_in[4] = 0;
hid_report_in[5] = 0;
hid_report_in[6] = 0;
hid_report_in[7] = 0;
```

```
     //Send the 8 byte packet over USB to the host.
lastTransmission = HIDTxPacket(HID_EP,
(BYTE*)hid_report_in, 0x08);
state = 0;
                                }
                            break;

                default: state = 0;
                            break;
                }

}//end keyboard()
//FIN DEL PEGADO DEL CÓDIGO
```

Tras realizar estos pasos, se ha compilado el programa, se ha cargado el firmware en el dispositivo y se ha comprobado el funcionamiento del microcontrolador tras conectarlo al ordenador.

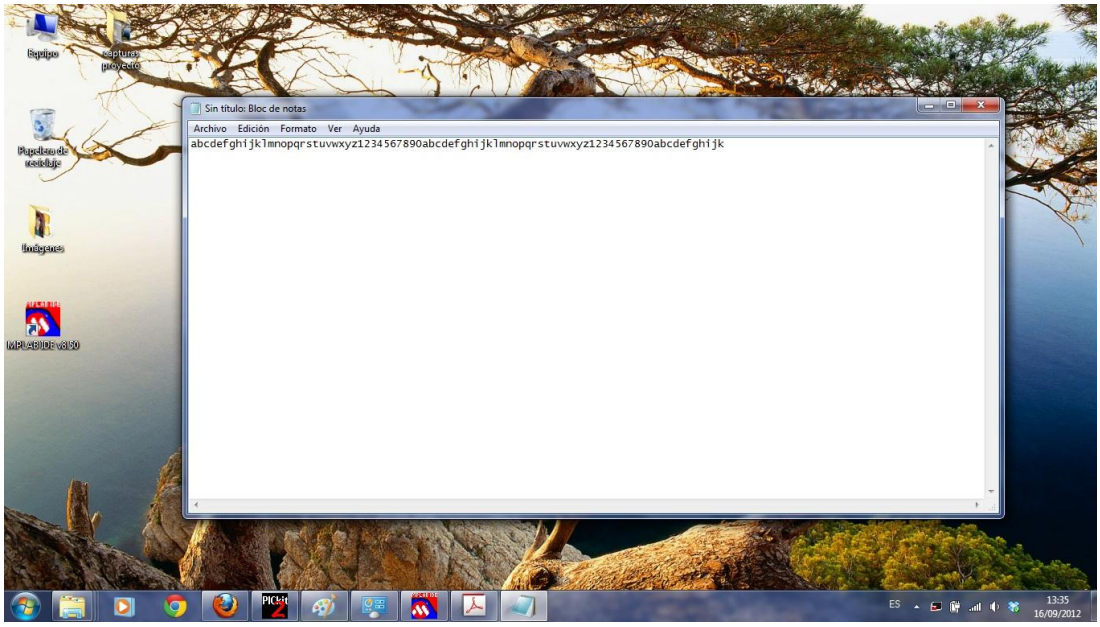


Figura 3.14 Línea de texto escrita con el dispositivo.

3.4 Software propio

Tras haber realizado las lecciones que provee Microchip en el manual de uso del kit de desarrollo, se ha iniciado la implementación del código del proyecto. Como paso previo a la programación del código, se ha programado el microcontrolador con el bootloader HID proporcionado por Microchip, que permite prescindir del programador PicKit2 para las instalaciones sucesivas de firmwares, cumpliendo con esto uno de los objetivos del proyecto.

3.4.1 Microchip HID Bootloader

Microchip proporciona principalmente dos métodos distintos para la carga de los programas realizados para los microcontroladores de la serie 18 en los respectivos dispositivos.

La más común es la programación por ICSP (In Circuit Serial Programming). Para realizar este tipo de programación, es necesario disponer de una tarjeta programadora para realizar la grabación del dispositivo (tipo PicKit 2 o similar), y software dedicado para la comunicación del PC con la tarjeta programadora.

Sin embargo, si queremos realizar continuas cargas de software en nuestro dispositvo o queremos prescindir del hardware programador, Microchip nos proporciona la alternativa a la programación por ICSP mediante la carga en el microcontrolador de un bootloader.

Un bootloader, o cargador de arranque, no es más que un pequeño programa que permite al microcontrolador su auto-programación, utilizando uno de los puertos de comunicaciones de los que dispone el dispositivo para comunicarse con el PC que contiene el firmware.

Microchip proporciona dos bootloaders USB diferentes para los microcontroladores de la serie 18. El más rápido de ellos está basa en un protocolo propio y no en un estándar de comunicación USB por lo que, para utilizarlo, es necesario instalar una serie de controladores en el PC con el que se vaya a realizar la programación para el reconocimiento del dispositivo, así como software dedicado para realizar la programación.

La otra opción que proporciona Microchip está basada en el protocolo USB HID (Human Interface Device). A diferencia del bootloader de protocolo propio, éste es estándar y es entendible por todos los computadores sin necesidad de recurrir a la instalación de ningún driver para su funcionamiento. Aunque este protocolo es más lento que el anterior (aunque en la práctica no es apreciable), se ha elegido este bootloader para facilitar la programación del mismo por parte del usuario que quiera reproducir el proyecto.

Al igual que ocurría con el primer bootloader, también es necesario recurrir a un software específico en el ordenador para realizar la programación. Este software es proporcionado por Microchip, que no sólo facilita el ejecutable, sino todo el proyecto completo para modificar en Visual Studio.

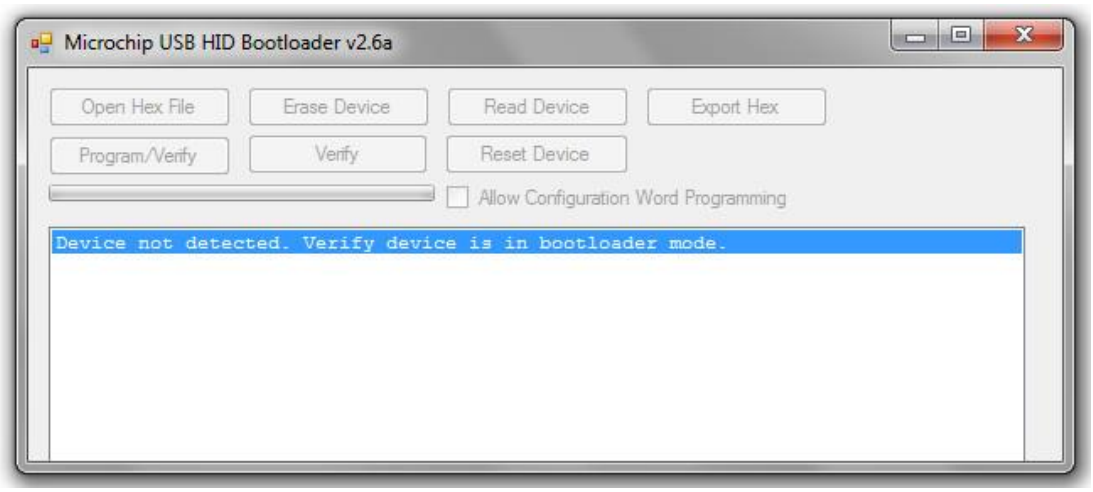


Figura 3.15 Microchip HID Bootloader v2.6a

3.4.2 Implementación del software propio.

3.4.2.1 Preparación del proyecto

Para la implementación el software se ha utilizado el proyecto *USB Device - HID - Mouse* que proporciona Microchip en el *Microchip Solutions v2010-08-04*. Este proyecto está preparado para la programación de firmware de ratones USB y está compilado para las distintas plataformas de desarrollo de Microchip, por lo que se pueden encontrar dentro de la carpeta del proyecto distintos entornos de trabajo.

Este proyecto implementa el mismo software que la lección de ratón en círculos, tan sólo que a diferencia del proyecto que se ha desarrollado en la lección, ofrece un árbol de carpetas mucho más ordenado.

Como se puede comprobar en la figura, los archivos referentes al USB que no se deben de manipular por parte del programador (al menos, en teoría) y que dan el soporte necesario para su utilización están incluidos en una carpeta llamada USB Stack, encontrando los archivos manipulables relacionados con el USB en el árbol principal del proyecto.

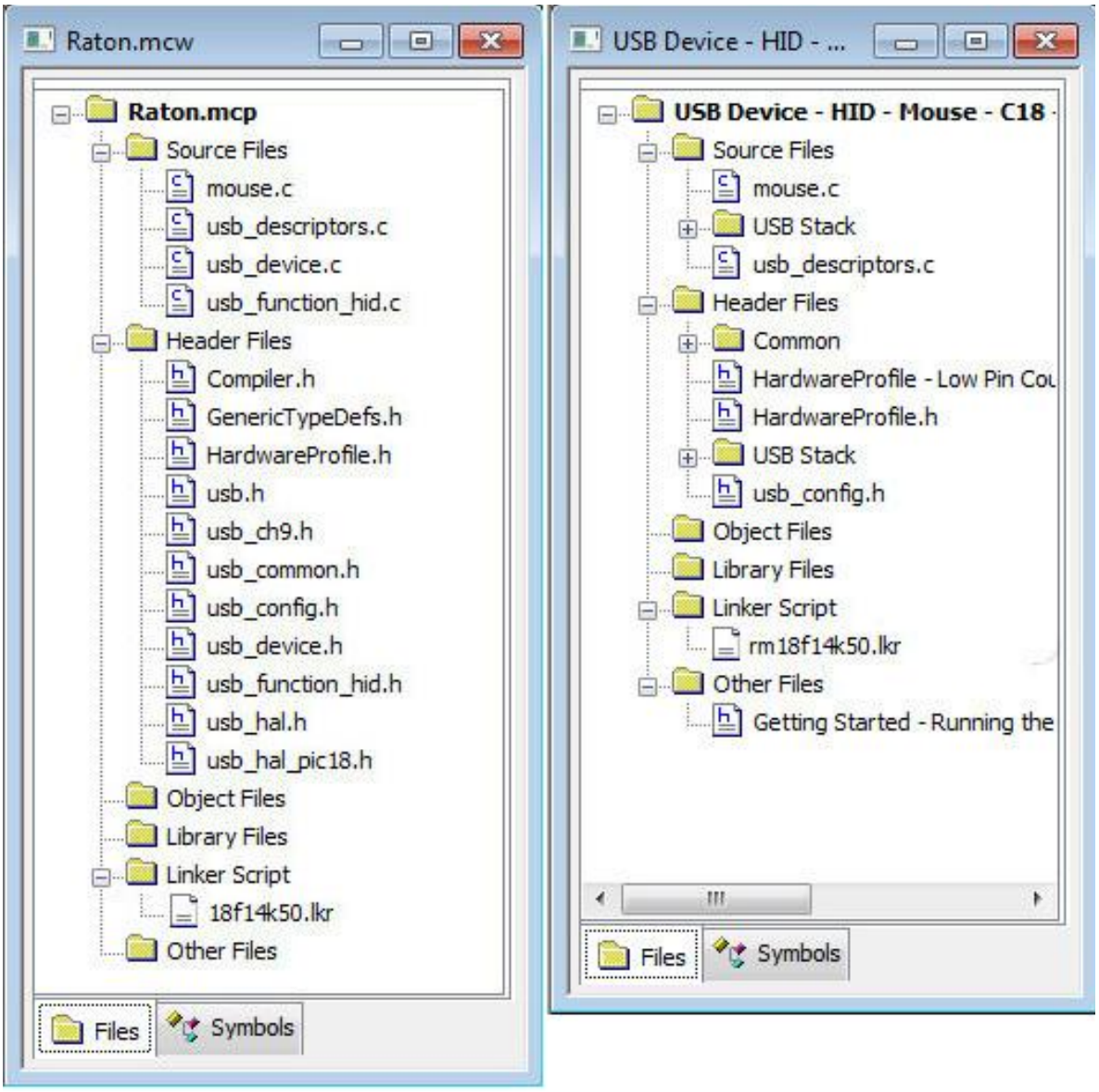


Figura 3.16 Árbol del proyecto de la lección y árbol del proyecto del ejemplo

Se ha escogido el entorno de trabajo para el Low Pin Count USB Developement Kit de entre todas las disponibles, debido a que este proyecto en concreto está compilado para nuestro microcontrolador y ya estamos familiarizados con el entorno de desarrollo.

El proyecto tiene la siguiente estructura de programación:

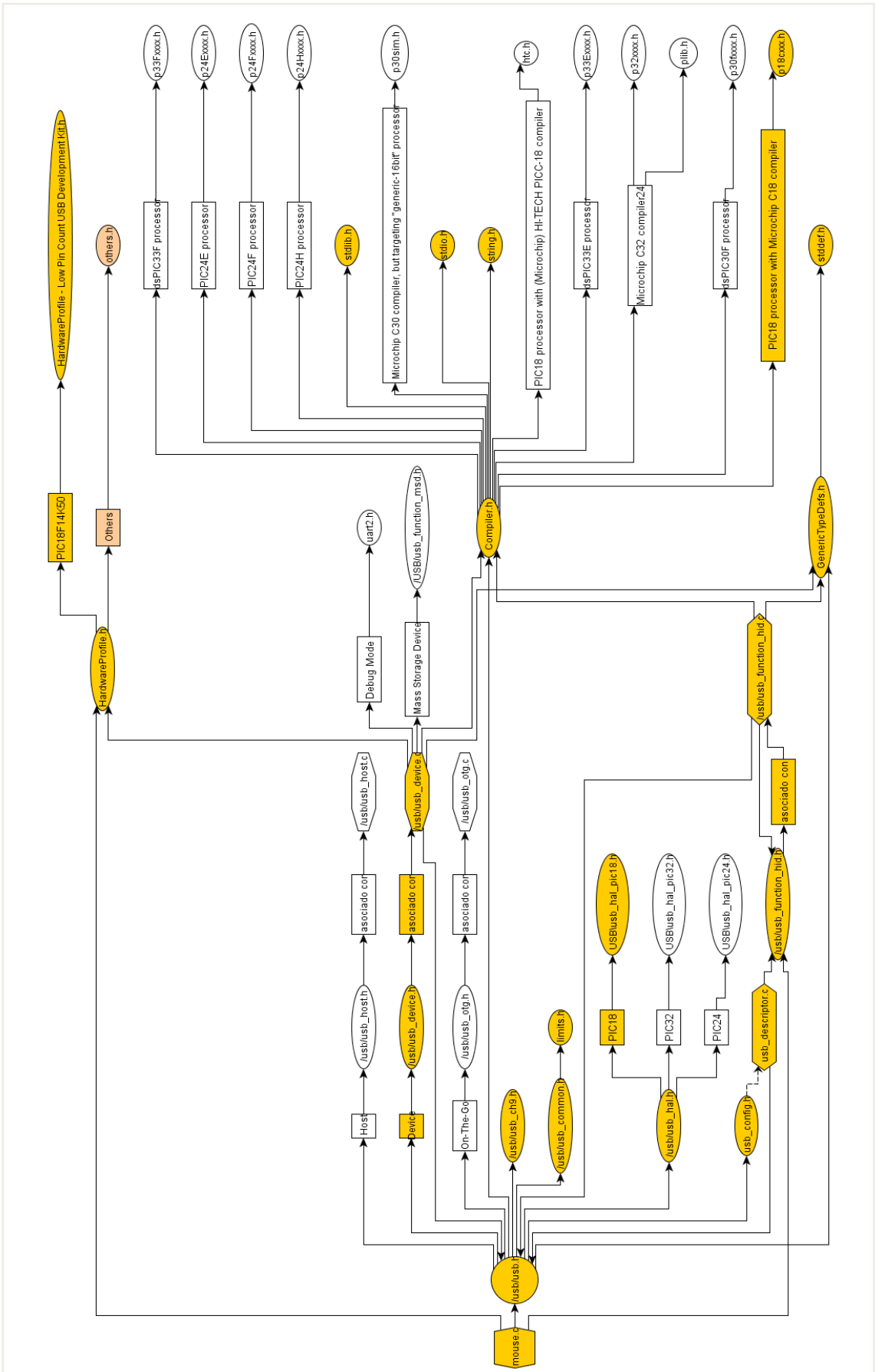


Figura 3.17 Estructura de programación del USB Stack de Microchip

Como se puede observar en la figura, el proyecto incluye multitud de archivos necesarios para el correcto funcionamiento del dispositivo. Parte de ellos son archivos de cabecera o de programa que se deben de modificar para programar correctamente nuestro dispositivo y otros forman parte del USB Stack de Microchip y no se deberían de modificar.

El programa principal *mouse.c* comienza con una serie de configuraciones específicas para cada placa entrenadora, utilizándose en el proyecto la correspondiente a la placa en uso. Tras la configuración, inicializa las variables globales y crea los prototipos de las funciones que se van a utilizar a lo largo del programa.

El siguiente paso que realiza el código es la preparación del programa en el caso de usar bootloader. Si se ha decidido que el microcontrolador utilizara un bootloader como método de carga de firmware, se tendría que descomentar la línea correspondiente al bootloader elegido en esta parte del código.

En el caso de haber descomentado alguna de las líneas, el programa se reconfigurará para poder compartir los recursos con el bootloader y no interferir en su funcionamiento, direccionando las interrupciones mediante la definición de tres etiquetas (una para el mapeo de la nueva dirección de memoria para las interrupciones de baja prioridad, otra para las de alta prioridad y otra para la dirección del vector de reset) y llamando a posteriori a dos funciones, que saltarán cuando las interrupciones sean llamadas, y redireccionarán el programa a las líneas antes descritas en las etiquetas.

A continuación, el programa prepara dos funciones relacionadas con los eventos de interrupción(una para las interrupciones de alta prioridad y otra para las interrupciones de baja prioridad), que permiten el manejo de las mismas.

El programa *main()*, programado a posteriori de estas funciones, comienza llamando a la función *InitializeSystem(),* y dependiendo si el dispositivo ha sido configurado para funcionar por interrupciones o por votación (*polling*), llamará a *USBDeviceAttach()* o a *USBDeviceTasks()*. Tras este paso, el programa llamará al programa *ProcessIO()* y finalizará.

La función *InitializeSystem()* inicializa el sistema, detectando que tipo de microcontrolador es el que estamos utilizando y realiza su configuración, detectando la fuente de alimentación del dispositivo observando cual de las dos etiquetas (*USE_USB_BUS_SENSE_IO* o *USE_SELF_POWER_SENSE_IO*) está definida, y configurando *INPUT_PIN* según lo conveniente y según venga descrito en el archivo de cabecera *HardwareProfile.h*, del cual hablaremos más adelante.

Tras estos pasos, llamará a la función *UserInit()* y tras ella, a la función *USBDeviceInit()* del programa *usb_device.c*, del cual también hablaremos más adelante.

La función *UserInit()* incluye todas las funciones y acciones que el usuario desea que se realicen a la hora de inicializar el sistema. En nuestro caso el programa llama a las

funciones *mInitAllLEDs()* y *mInitAllSwitches(),* inicializa el buffer de movimientos del mouse a 0, pone a verdadero el flag de *emulate_mode*, inicializa la variable del manejador de la última transmisión a cero y llama a la función *Emulate_Mouse()* (la cual es similar a la de la práctica anterior y huelga su explicación).

El programa también pasa los valores de *sw2* y *sw3* (los valores de dos de los switches) a *old_sw2* y *old_sw3*. Este paso de valores de los switches a dos variables temporales forman parte de dos rutinas anti rebotes de los pulsadores, *Switch2IsPressed* y *Switch3IsPressed*. Estas rutinas comprueban si ha habido un cambio de flanco en la señal. Si esto ha sido así, comprueban si el valor de la entrada es 0. En el caso de que sea correcto, devuelve TRUE y en el caso contrario, devuelven FALSE.

A continuación de estas dos funciones, el programa incluye otra función, *BlinkUSBStatus()* que se encarga de hacer que los leds se enciendan y apaguen según el estado en el que se encuentre el USB en cada momento.

Tras esta función, el programa nos proporciona dos funciones vacías, *USBCBSuspend()* y *USBCBWakeFromSuspend(),* con la intención de que se utilicen si el programa hiciera que el microcontrolador entrara en suspensión o modo de bajo consumo.

La primera de las funciones permite escribir el código que se desee que se ejecute en el momento anterior de entrar en suspensión y la segunda de las funciones permite escribir el código que se ejecutará cuando el microcontrolador salga del estado de suspensión.

Tras estas funciones, el programa incluye la función *USBCB_SOF_Handler(),* necesaria para que el dispositivo, en el momento de la enumeración, comunique al host que se trata de un dispositivo de alta velocidad.

A continuación, el programa proporciona la función *USBCBErrorHandler(),* para poder insertar en ella el código necesario a la hora de realizar la depuración del programa.

Después de esta última función, el programa incluye varias funciones necesarias para que la enumeración y detección del dispositivo se realice de manera correcta. Estas son las funciones *USBCBCheckOtherReq()*,*USBCBStdSetDscHandler(),USBCBInitEP(), USBCBSendResume(),* y *USER_USB_CALLBACK_EVENT_HANDLER(),* tras lo cual el programa principal termina.

Para poder ejecutar todo el código escrito en el programa *mouse.c*, éste incluye tres archivos al inicio del programa del cual irá utilizando su código cuando sea conveniente.

El primero de ellos es el archivo HardwareProfile.h. En este archivo de cabecera se encuentran las redirecciones a los distintos archivos de configuración de los microcontroladores para los cuales el proyecto es compatible, incluyendo en el proyecto el archivo el necesario en cada caso.

Como la lista de microcontroladores compatibles con el proyecto es demasiado extensa, y con el fin de no ensuciar el árbol del esquema, tan sólo se ha incluido en el mismo el archivo referente al PIC18F14K50, incluyendo el resto de manera gráfica en el archivo others.h, remarcado con fondo rosa.

En este caso, el archivo que HardwareProfile.h incluye es HardwareProfile - Low Pin Count USB Development Kit.h. Este archivo incluye parte del código necesario para el correcto funcionamiento de la placa de desarrollo con los proyectos proporcionados por Microchip:

```
/********************************************************
************/
    /******** USB stack hardware selection options
*********************/

/********************************************************
************/
    //This section is the set of definitions required by
the MCHPFSUSB
    //  framework.  These definitions tell the firmware
what mode it is
    //  running in, and where it can find the results to
some information
    //  that the stack needs.
    //These definitions are required by every
application developed with
    //  this revision of the MCHPFSUSB framework.
Please review each
    //  option carefully and determine which options are
desired/required
    //  for your application.

    //#define USE_SELF_POWER_SENSE_IO
    #define tris_self_power     TRISAbits.TRISA2    //
Input
    #if defined(USE_SELF_POWER_SENSE_IO)
    #define self_power          PORTAbits.RA2
    #else
    #define self_power          1
    #endif
    //#define USE_USB_BUS_SENSE_IO
    #define tris_usb_bus_sense  TRISAbits.TRISA1    //
Input
```

```
    #if defined(USE_USB_BUS_SENSE_IO)
    #define USB_BUS_SENSE        PORTAbits.RA1
    #else
    #define USB_BUS_SENSE        1
    #endif
```

Como se puede leer en el comentario de cabecera, esta parte del código es necesario para el correcto funcionamiento del stack. Aunque se puede modificar, Microchip recomienda hacerlo con cuidado, ya que un mal funcionamiento de este código puede provocar un mal funcionamiento del stack.

Su funcionamiento es simple, tan sólo define las patillas que van a decidir si el microcontrolador está autoalimentado y si existe la comunicación USB.

La siguiente parte del código programa las definiciones necesarias que definen el layout de la placa, así como la configuración hardware del microcontrolador.

```
    //Uncomment the following line to make the output
HEX of this
    //  project work with the HID Bootloader
    //#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER

    /** Board definition
*********************************************/
    //These defintions will tell the main() function
which board is
    //  currently selected.  This will allow the
application to add
    //  the correct configuration bits as wells use the
correct
    //  initialization functions for the board.  These
defitions are only
    //  required in the stack provided demos.  They are
not required in
    //  final application design.

    #define DEMO_BOARD LOW_PIN_COUNT_USB_DEVELOPMENT_KIT
    #define LOW_PIN_COUNT_USB_DEVELOPMENT_KIT
    #define CLOCK_FREQ 48000000

    /** LED
*****************************************************
****/
    #define mInitAllLEDs()      LATC &= 0xF0; TRISC &=
0xF0;
```

```
    #define mLED_1              LATCbits.LATC0
    #define mLED_2              LATCbits.LATC1
    #define mLED_3              LATCbits.LATC2
    #define mLED_4              LATCbits.LATC3

    #define mGetLED_1()         mLED_1
    #define mGetLED_2()         mLED_2
    #define mGetLED_3()         mLED_3
    #define mGetLED_4()         mLED_4

    #define mLED_1_On()         mLED_1 = 1;
    #define mLED_2_On()         mLED_2 = 1;
    #define mLED_3_On()         mLED_3 = 1;
    #define mLED_4_On()         mLED_4 = 1;

    #define mLED_1_Off()        mLED_1 = 0;
    #define mLED_2_Off()        mLED_2 = 0;
    #define mLED_3_Off()        mLED_3 = 0;
    #define mLED_4_Off()        mLED_4 = 0;

    #define mLED_1_Toggle()     mLED_1 = !mLED_1;
    #define mLED_2_Toggle()     mLED_2 = !mLED_2;
    #define mLED_3_Toggle()     mLED_3 = !mLED_3;
    #define mLED_4_Toggle()     mLED_4 = !mLED_4;

/** SWITCH
*****************************************************************
/
    #define mInitSwitch2()      //TRISAbits.TRISA3=1
        //only one switch available so double duty
    #define mInitSwitch3()      //TRISAbits.TRISA3=1
    #define sw2                 PORTAbits.RA3
    #define sw3                 PORTAbits.RA3
    #define mInitAllSwitches()  mInitSwitch2();

/** I/O pin definitions
*********************************************/
    #define INPUT_PIN 1
    #define OUTPUT_PIN 0

#endif
//HARDWARE_PROFILE_LOW_PIN_COUNT_USB_DEVELOPMENT_KIT_H
```

Blah Blah

Al igual que en el archivo mouse.c, este archivo de cabecera también tiene una parte dedicada al bootloader HID y, en el caso de que se utilizase, sería necesario descomentar la línea `#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER` para que el proyecto fuera compatible con el bootloader.

Tras estas líneas se encuentran las distintas definiciones necesarias para que los ejemplos proporcionados por Microchip funcionen correctamente, pudiendo ser eliminadas o modificadas en el caso de no utilizarse en el programa definitivo.

Entre las definiciones se encuentran la configuración del reloj a 48Mhz, la definición de las patillas de los leds junto con varios tratamientos básicos con las mismas, las patillas asignadas a los switches y dos asignaciones a las variables INPUT_PIN y OUTPUT_PIN con las constantes 1 y 0.

Otro de los archivos de cabecera que incluye el programa principal es el archivo *usb.h*. Este archivo de cabecera agrupa al resto de archivos de cabecera necesarios para el funcionamiento del stack USB, así como ciertos archivos de cabecera para cada tipo de rol USB distinto que puede presentar el microcontrolador (dispositivo, host o On-The-Go). También incluye al final del archivo definiciones referentes al número de versión del programa.

Debido a que el microcontrolador sólo va a tomar el rol de dispositivo, el proyecto solo incluye los archivos de cabecera requeridos para este tipo de funcionamiento. El archivo incluido, por lo tanto, es *usb_device.h* y su asociado, *usb_device.c.*

En el interior del archivo *usb_device.h* se encuentra la inicialización de las funciones que va a manejar todas las acciones referentes con la comunicación USB, así como la definición de valores fijos para ciertos mensajes clave en las comunicaciones. Las funciones están simplemente definidas, para permitir al usuario configurarlas desde el programa principal (como *USBDeviceInit(),* por ejemplo) para evitar que el usuario tenga que escribir código dentro de un archivo incluido en el USB Stack.

Dentro del archivo *usb_device.c* encontramos un galimatías de código relacionado con la comunicación USB en modo dispositivo, como por ejemplo la configuración del modo ping-pong en sus cuatro formas de funcionamiento, y el comportamiento del dispositivo a la hora de enviar información, ya sea en la enumeración o en la comunicación como dispositivo.

A su vez, el archivo *usb_device.c* incluye los archivos *HardwareProfile.h* (ya explicado con anterioridad), *GenericTypeDefs.h* y *Compiler.h*. En el caso de que estuviese el modo debug activo también incluiría el archivo *uart2.h*, encargado del manejo de las comunicaciones asíncronas. Si estuviese activo el modo de dispositivo de almacenamiento masivo, incluiría el archivo de cabecera *usb_function_msd.h.* Como ninguno de estos dos modos está activo, no podemos encontrar estos archivos dentro del proyecto.

El archivo *GenericTypeDefs.h* permite al proyecto poder utilizar los tipo de variable genéricos de C, incluyendo multitud de definiciones para los tipos de datos más comunes. A su vez, *GenericTypeDefs.h* incluye el archivo *stddef.h*, un archivo de cabecera estándar de C que complementa con más definiciones al archivo anterior.

El archivo *Compiler.h* incluye los archivos necesarios para la compilación del proyecto para cada tipo de familia de microcontroladores de Microchip, así como los archivos de cabecera *stdio.h, string.h* y *stdlib.h,* que permiten utilizar un sistema estándar de entrada/salida de información, la utilización de cadena de caracteres y las librerías estándar de C. En este caso tan sólo está incluido el archivo *p18cxxx.h,* ya que tan sólo se va a compilar el proyecto para un microcontrolador de la familia 18, quedando el resto de archivos fuera de nuestro proyecto.

Otro de los archivos incluidos en el programa principal es el archivo de cabecera *usb_ch9.h*, que incluye todas las definiciones y las funciones necesarias para cumplir correctamente con el protocolo definido en el capítulo 9 de la definición de comunicaciones USB.

El archivo *usb_hal.h*, también incluido en el archivo de cabecera *usb.h*, permite al proyecto abstraer la interfaz hardware. Para ello el archivo incluye rutinas y etiquetas que tratan rutinas de la capa física del USB, como tratamiento de errores, detección de tensiones en el bus... así como la inclusión de otros archivos de cabecera concretos para las distintas familias de microcontroladores compatibles con USB (PIC18,PIC24 y PIC32). En este caso tan sólo está incluido el archivo *usb_hal_pic18.h*, el cual contiene más rutinas y etiquetas, pero ya específicas de esta familia de microcontroladores.

Otro de los archivos de cabecera a los que llama el archivo *usb.h* es *usb_common.h.* Este archivo de cabecera define las constantes y variables comunes en la comunicación por usb, como la designación de errores, valores de retorno en las comunicaciones, flags de comunicaciones ect. A su vez, este archivo incluye otro archivo , *limits.h*, un archivo de cabecera clásico en compiladores de C que define los límites numéricos de las variables de entorno.

El último de los archivos que es incluido por *mouse.c* es *usb_config.h*. En este archivo se encuentra todo el código referente a la configuración del dispositivo USB, como el tamaño del buffer de comunicaciones, el tipo de rebote de señal, los números de identificación de producto y de fabricante, el tipo de clase usado, y otras características del USB.

Aunque no está incluido como tal, el archivo de cabecera define una etiqueta llamada *USB_USER_DEVICE_DESCRIPTOR*, que asocia a la variable *device_dsc*. Esta variable viene descrita en el archivo *usb_descriptor.c* y es necesario que exista con ese mismo nombre, o el programa dejaría de funcionar correctamente, ya que es una especificación obligatoria de Microchip para el uso del stack USB.

En el interior del archivo *usb_descriptor.c* encontramos el siguiente código:

```
#ifndef __USB_DESCRIPTORS_C
#define __USB_DESCRIPTORS_C

/** INCLUDES *******************************************************/
#include "./USB/usb.h"
#include "./USB/usb_function_hid.h"

/** CONSTANTS ******************************************************/
#if defined(__18CXX)
#pragma romdata
#endif

/* Device Descriptor */
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
    0x12,    // Size of this descriptor in bytes
    USB_DESCRIPTOR_DEVICE,                // DEVICE descriptor type
    0x0200,                 // USB Spec Release Number in BCD format
    0x00,                   // Class Code
    0x00,                   // Subclass code
    0x00,                   // Protocol code
    USB_EP0_BUFF_SIZE,            // Max packet size for EP0, see usb_config.h
    MY_VID,                 // Vendor ID
    MY_PID,                 // Product ID: Mouse in a circle fw demo
    0x0003,                 // Device release number in BCD format
    0x01,                   // Manufacturer string index
    0x02,                   // Product string index
    0x00,                   // Device serial number string index
    0x01                                // Number of possible configurations
};
```

Se puede observar cómo, en esta parte del código, guarda en la memoria ROM la variable etiquetada con anterioridad en el archivo usb_config.h. Dentro de esa variable se encuentran los distintos valores que son necesarios escribir para que el descriptor de dispositivo esté correctamente creado, según el capítulo 9 de las especificaciones del protocolo USB.

Tras el valor en hexadecimal se encuentra bajo una línea de comentario a qué código corresponde el valor señalado. Algunos de estos valores no vienen escritos en

hexadecimal y hay una etiqueta en su lugar, debido a que se han definido con anterioridad en el archivo usb_config.h.

```
ROM BYTE configDescriptor1[]={
    /* Configuration Descriptor */
    0x09,//sizeof(USB_CFG_DSC),    // Size of this
descriptor in bytes
    USB_DESCRIPTOR_CONFIGURATION,               //
CONFIGURATION descriptor type
    DESC_CONFIG_WORD(0x0022),   // Total length of data
for this cfg
    1,                      // Number of interfaces in
this cfg
    1,                      // Index value of this
configuration
    0,                      // Configuration string
index
    _DEFAULT | _SELF,               // Attributes, see
usb_device.h
    50,                     // Max power consumption (2X
mA)

 /* Interface Descriptor */
    0x09,//sizeof(USB_INTF_DSC),   // Size of this
descriptor in bytes
    USB_DESCRIPTOR_INTERFACE,               // INTERFACE
descriptor type
    0,                      // Interface Number
    0,                      // Alternate Setting Number
    1,                      // Number of endpoints in
this intf
    HID_INTF,               // Class code
    BOOT_INTF_SUBCLASS,     // Subclass code
    HID_PROTOCOL_MOUSE,     // Protocol code
    0,                      // Interface string index

    /* HID Class-Specific Descriptor */
    0x09,//sizeof(USB_HID_DSC)+3,    // Size of this
descriptor in bytes RRoj hack
    DSC_HID,                // HID descriptor type
    DESC_CONFIG_WORD(0x0111),               // HID
Spec Release Number in BCD format (1.11)
    0x00,                   // Country Code (0x00 for
Not supported)
    HID_NUM_OF_DSC,         // Number of class
descriptors, see usbcfg.h
    DSC_RPT,                // Report descriptor type
```

```
    DESC_CONFIG_WORD(50),   //sizeof(hid_rpt01),      //
Size of the report descriptor

    /* Endpoint Descriptor */
    0x07,/*sizeof(USB_EP_DSC)*/
    USB_DESCRIPTOR_ENDPOINT,    //Endpoint Descriptor
    HID_EP | _EP_IN,            //EndpointAddress
    _INTERRUPT,                       //Attributes
    DESC_CONFIG_WORD(3),                  //size
    0x01                        //Interval
};
```

Tras la inicialización del descriptor de dispositivo, llega el turno de los descriptores de configuración, de interfaz (incluyendo el de clase específica), y el de endpoint. Como este dispositivo tan sólo tiene un perfil de dispositivo (Mouse HID), tan sólo existe un descriptor de configuración, de interfaz y de endpoint. Se puede ver como se configuran todos los descriptores en el código anterior, con sus respectivos comentarios de aclaración.

```
//Language code string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD
string[1];}sd000={
sizeof(sd000),USB_DESCRIPTOR_STRING,{0x0409
}};

//Manufacturer string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD
string[25];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{'M','i','c','r','o','c','h','i','p',' ',
'T','e','c','h','n','o','l','o','g','y',' ',
'I','n','c','.'
}};

//Product string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD
string[22];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{'M','o','u','s','e',' ','I','n',' ','a',' ',
'C','i','r','c','l','e',' ','D','e','m','o'
}};
```

```
//Class specific descriptor - HID mouse
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
    {0x05, 0x01, /* Usage Page (Generic Desktop)
*/
    0x09, 0x02, /* Usage (Mouse)
*/
    0xA1, 0x01, /* Collection (Application)
*/
    0x09, 0x01, /*  Usage (Pointer)
*/
    0xA1, 0x00, /*  Collection (Physical)
*/
    0x05, 0x09, /*      Usage Page (Buttons)
*/
    0x19, 0x01, /*      Usage Minimum (01)
*/
    0x29, 0x03, /*      Usage Maximum (03)
*/

    0x15, 0x00, /*      Logical Minimum (0)
*/
    0x25, 0x01, /*      Logical Maximum (0)
*/
    0x95, 0x03, /*      Report Count (3)
*/
    0x75, 0x01, /*      Report Size (1)
*/
    0x81, 0x02, /*      Input (Data, Variable, Absolute)
*/
    0x95, 0x01, /*      Report Count (1)
*/
    0x75, 0x05, /*      Report Size (5)
*/
    0x81, 0x01, /*      Input (Constant)    ;5 bit
padding  */
    0x05, 0x01, /*      Usage Page (Generic Desktop)
*/
    0x09, 0x30, /*      Usage (X)
*/
    0x09, 0x31, /*      Usage (Y)
*/
    0x15, 0x81, /*      Logical Minimum (-127)
*/
```

```
    0x25, 0x7F, /*      Logical Maximum (127)
*/
    0x75, 0x08, /*      Report Size (8)
*/
    0x95, 0x02, /*      Report Count (2)
*/
    0x81, 0x06, /*      Input (Data, Variable, Relative)
*/
    0xC0, 0xC0}
};/* End Collection,End Collection            */
//Array of configuration descriptors
ROM BYTE *ROM USB_CD_Ptr[]=
{
    (ROM BYTE *ROM)&configDescriptor1
};

//Array of string descriptors
ROM BYTE *ROM USB_SD_Ptr[]=
{
    (ROM BYTE *ROM)&sd000,
    (ROM BYTE *ROM)&sd001,
    (ROM BYTE *ROM)&sd002
};

/** EOF usb_descriptors.c
*************************************************/

#endif
```

Blah Blah

Como se puede observar en el código, tras configurar el idioma del dispositivo, el programa configura dos de las etiquetas que ha utilizado en su descripción, los identificadores de producto y fabricante (Vendor ID y Product ID).

Tras esta configuración, el programa configura el descriptor de clase específico y tras esto, configura el array de descriptores que va a utilizar (en nuestro caso, tan sólo utiliza uno) y el array de strings de descriptores, insertando el string de idioma y el resto de strings necesarios.

El archivo *usb_descriptor.c* también incluye dos archivos de cabecera, el archivo *usb_function_hid.h* y el archivo *usb.h*, ambos incluidos también por el archivo del programa principal, *mouse.c* . Como el archivo *usb.h* ya ha sido explicado con anterioridad, pasaremos a explicar el archivo *usb_function_hid.h.*

El archivo *usb_function_hid.h* configura los distintos registros del microcontrolador para prepararlo para la función de dispositivo de interfaz humana, así como la configuración del formato de los mensajes enviados a través del bus.

Se puede encontrar la configuración de las cabeceras de los mensajes, la configuración de los paquetes de comunicación etc.

Si en el archivo de cabecera *usb_function_hid.h* se han configurado los registros, en el archivo de programa asociado *usb_function_hid.c* se utilizan estos registros y se programan las funciones necesarias para el correcto funcionamiento del microcontrolador como dispositivo HID. El archivo *usb_function_hid.c* a su vez incluye los archivos *GenericTypeDefs.h* y *Compiler.h*, ya explicados anteriormente.

Como resumen del funcionamiento de los archivos incluidos en el proyecto, se ha creado la siguiente tabla, con el nombre de los archivos más relevantes, su funcionamiento, y la pertenencia al Stack USB de Microchip.

Nombre del archivo	Descripción del funcionamiento	¿Microchip USB Stack?
mouse.c	- Prepara proyecto para bootloader. - Maneja las interrupciones. - Inicializa el sistema. - Emula movimiento del ratón en círculos. - Maneja la comunicación USB.	NO
usb.h	- Incluye los archivos necesarios para el funcionamiento del USB Stack.	SI
HardwareProfile.h	- Incluye los archivos de configuración de todos los microcontroladores compatibles.	NO
HardwareProfile - Low Pin Count USB Development Kit.h	- Archivo de configuración hardware para la placa de desarrollo USB Low Pin Count.	NO
usb_device.h **usb_device.c**	- Permiten configurar el microcontrolador con el perfil de dispositivo.	SI
usb_ch9.h	- Incluye las definiciones y funciones necesarias para cumplir con el Capítulo 9 del protocolo USB.	SI
usb_common.h	- Integra las funciones y definiciones necesarias para las comunicaciones USB comunes.	SI
usb_hal.h	- Incluye las capas de abstracción hardware de todos los microcontroladores compatibles.	SI
usb_hal_pic18.h	- Capa de abstracción hardware de la familia de microcontroladores PIC18.	SI
usb_config.h	- Archivo de configuración USB.	NO
usb_descriptor.c	- Descriptor USB.	NO
usb_function_hid.h **usb_function_hid.c**	- Permiten al microcontrolador comportarse como un Dispositivo de Interfaz Humana.	SI

Antes de empezar a programar el código, se ha realizado una limpieza del proyecto, tanto a nivel de código como de archivos, debido a que tan sólo vamos a compilar el proyecto para el microcontrolador PIC18F14K50 y la placa hardware que vamos a utilizar no posee el mismo layout que la placa de circuito impreso del kit.

Como primera medida, se han borrado todos los archivos innecesarios referidos a otros microcontroladores de la carpeta principal del proyecto:

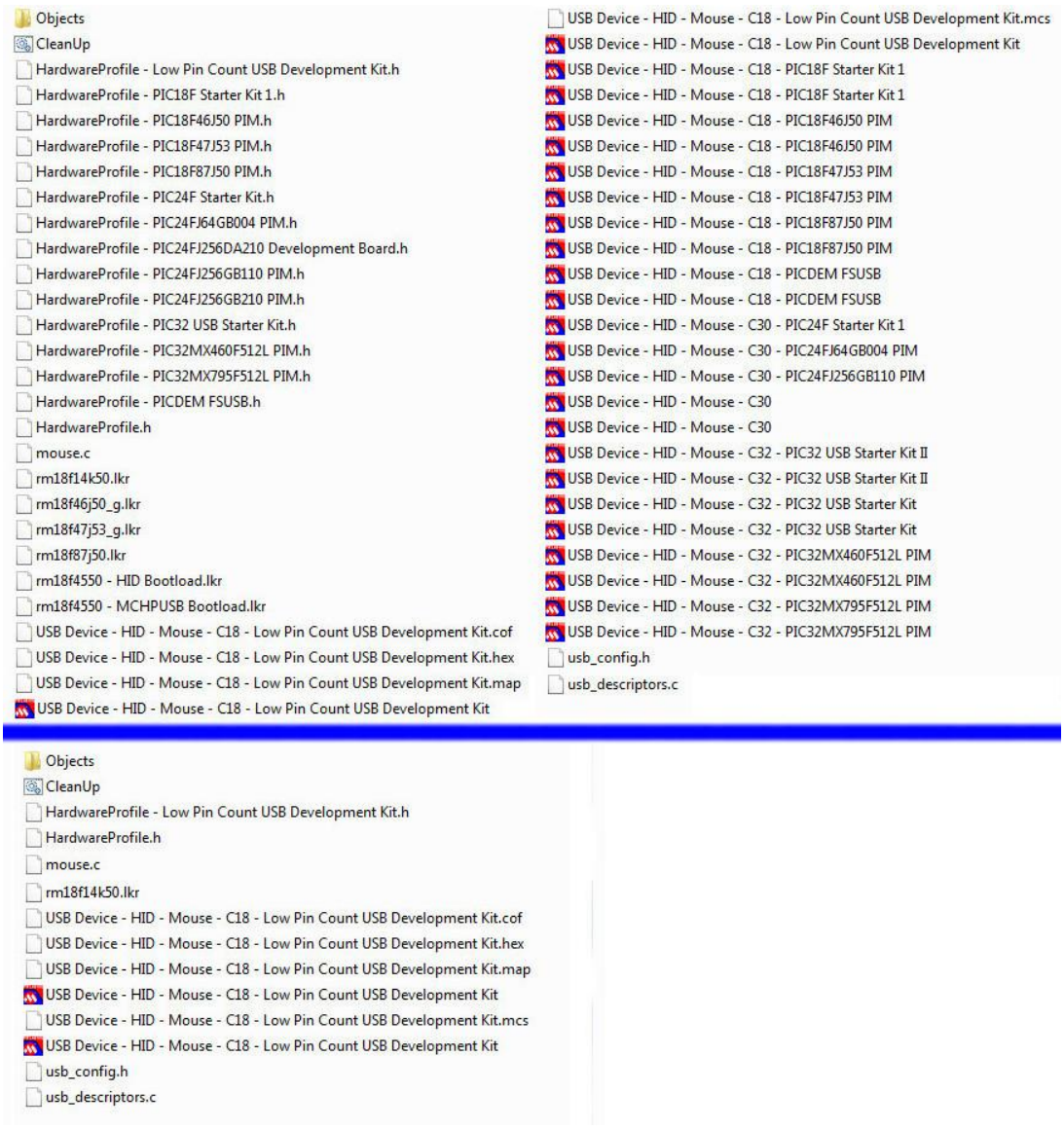


Figura 3.18 Carpeta del proyecto antes y después de eliminar archivos innecesarios

Se puede observar en la figura como se han eliminado una gran cantidad de archivos, y que no podemos encontrar los archivos referentes al USB Stack en la carpeta pricipal del proyecto. Esto es debido a que estos archivos se encuentran un nivel por encima en el árbol de archivos, y se ha tenido en cuenta a la hora de incluirlos en el proyecto.

El siguiente paso realizado ha sido la eliminación de todas las inclusiones que no se van a realizar de los archivos de proyecto restantes, así como de los comentarios innecesarios.

El primer archivo al que se le ha realizado la limpieza ha sido el archivo *mouse.c.*
Se han eliminado todas las inicializaciones de los registros de configuración de otros microcontroladores, dejando tan sólo los del PIC18F14K50.

Se han eliminado las variables *old_sw2*, *old_sw3*, *emulate_mode*, *movement_length*, *vector* y *dir_table[]* y las inicializaciones de los prototipos de las funciones *BlinkUSBStatus(), Switch2IsPressed()* y *Switch3IsPressed(),* así como toda referencia a ellas en el programa, debido a que no se van a utilizar en el proyecto definitivo.

En la función *InitializeSystem()* se han eliminado todas las inicializaciones de los microcontroladores que no se van a utilizar, quedándonos tan sólo con la inicialización del PIC18F14K50.

En la función *Emulate_Mouse()* se han eliminado todo el código que genera el movimiento en círculos ya que no se va a utilizar, dejando tan sólo el código de envío de la información al report HID.

En el archivo *Hardware_Profile.h* tan sólo se ha dejado la línea que incluye el archivo *Hardware_Profile - Low Pin Count USB Development Kit.h*.

En éste último se ha eliminado todo excepto las detecciones de auto alimentación y de utilización de la interfaz USB, la definición de la utilización del bootloader hid y la configuración de la frecuencia del reloj.

El montaje hardware definitivo del proyecto con el que va a interactuar el microcontrolador es una placa de circuito impreso propia, con un layout distinto al de la PCB del kit del desarrollo.

Como consecuencia de este cambio en la configuración hardware, hay que añadir un nuevo perfil hardware a nuestro dispositivo.

Para ello se ha modificado el archivo *HardwareProfile.h* y, en vez de añadir un archivo externo de perfil hardware, se ha escrito en el mismo archivo la configuración del dispositivo, borrando por lo tanto el archivo de configuración de la placa de desarrollo de Microchip.

El código del archivo *HardwareProfile.h* queda como se muestra en la figura.

```
#ifndef HARDWARE_PROFILE_H
#define HARDWARE_PROFILE_H

#define self_power          1
#define USB_BUS_SENSE       1
#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER
#define CLOCK_FREQ 48000000
```

```
//INICIALIZACIÓN DE LOS PUERTOS
#define Inicializar_Hardware() TRISC &= 0xFC; TRISB &=
0X80; ANSEL &= 0X00; ANSELH &= 0X00;

//ETIQUETAS DE PUERTOS
#define joystickArriba        PORTBbits.RB7    //Asignar
patilla rb7 para joystick arriba
#define joystickAbajo         PORTCbits.RC6    //Asignar
patilla rc6 para joystick abajo
#define joystickDerecha       PORTCbits.RC3    //Asignar
patilla rc3 para joystick derecha
#define joystickIzquierda     PORTCbits.RC7    //Asignar
patilla rc7 para joystick izquierda
#define botonDerecho          PORTCbits.RC4    //Asignar
patilla rc4 para boton derecho
#define botonIzquierdo       PORTCbits.RC5     //Asignar
patilla rc5 para boton izquierdo
#define salidaLed             PORTCbits.RC0    //Asignar
patilla rc0 para salida de led
#define salidaPiezo           PORTCbits.RC1    //Asignar
patilla rc1 para salida de piezoelectrico
```

Como se puede ver en el código, se han fijado las variables de auto alimentación y de sensor de USB a 1, debido a que se conoce que el dispositivo tomará la alimentación del USB y que siempre va a funcionar como dispositivo USB.

Se han incluido la línea de programación con bootloader HID y se ha configurado el reloj a 48 Mhz. Tras esto, se ha creado una etiqueta de inicialización de los puertos, que configura las patillas RC0, RC1, RB0, RB1, RB2, RB3, RB4, RB5, y RB6 como salidas digitales y las patillas RC2, RC3, RC4, RC5, RC6, RC7, y RB7 como entradas digitales.

Para que los puertos se inicialicen correctamente, se ha añadido en la función *UserInit()* la llamada a la etiqueta para que los puertos se inicialicen cuando el programa principal llame a la función que Microchip ha reservado para las inicializaciones de usuario.

A continuación , se han etiquetado las patillas utilizadas según la conexión que van a tener en la placa de circuito impreso. Se ha decidido utilizar etiquetas para los puertos en vez de utilizar los nombres directamente para, por un lado, facilitar la lectura del programa principal y para poder realizar cualquier cambio en la conexión hardware sin tener que modificar el programa principal.

Tras la preparación de los puertos, se ha modificado el archivo usb_descriptor para personalizar los descriptores del proyecto.

```
//Manufacturer string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD
string[37];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{'U','n','i','v','e','r','s','i','d','a','d','
','P','o','l','i','t','e','c','n','i','c','a ',' ',
'd','e',' ','C','a','r','t','a','g','e','n','a','.',
}};

//Product string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD
string[26];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{'R','a','t','o','n',' ','p','a','r','a',' ',
'D','i','s','c','a','p','a','c','i','t','a','d','o','s',
'.'
}};
```

El cambio de las cadenas de caracteres de los descriptores de fabricante y producto es tan solo a modo de ejemplo, y no con motivos comerciales. Aunque Microchip permite la utilización del Stack USB sin pago de royalties, la utilización del Vendor ID no es gestionado por ellos, sino por el foro de implementadores de USB (USB - IF).

Microchip permite la utilización de su Vendor ID de manera gratuita, previa aceptación de la aplicación. Si la aplicación fuese aceptada, Microchip nos asignaría un Product ID de manera gratuita mientas que no se superase un volumen de ventas de más de 10000 unidades. En el caso de superar este volumen de ventas, se tendría que adquirir un Vendor ID propio para comercializar el producto.

3.4.2.2 Implementación del código.

Terminada la preparación del código, se ha procedido a la programación de la función *Emulate_Mouse(),* que deberá de seguir el siguiente flujograma.

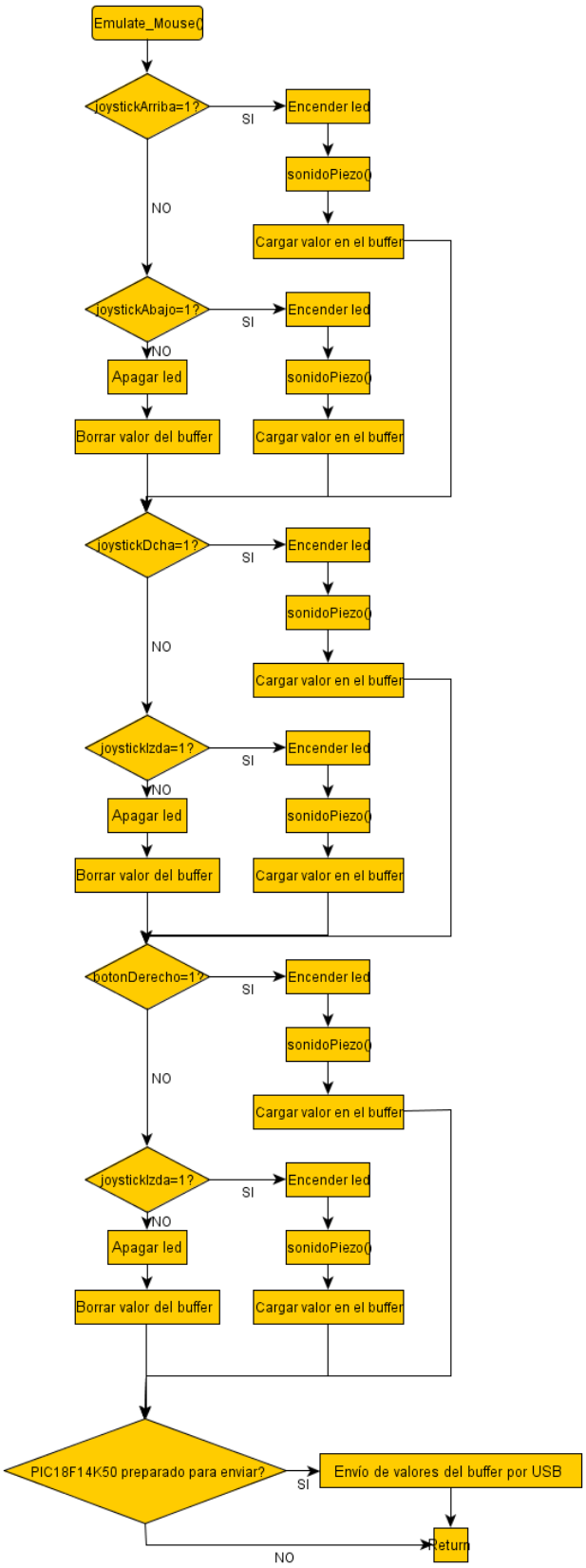


Figura 3.19 Flujograma del programa

Como se puede observar en el flujograma, el programa evaluará si el usuario ha actuado sobre el pulsador de movimiento ascendente del cursor. Si esto fuese afirmativo, el programa encenderá el led y llamará a la función sonidoPiezo(), que generará una señal cuadrada que excitará el piezoeléctrico, que describiremos más adelante.

Tras éste último paso el programa preparará en el buffer del USB el valor que corresponde con el movimiento ascendente del ratón. El buffer USB está compuesto por un array de tres componentes, cada uno de ellos relacionado con un gesto del ratón.

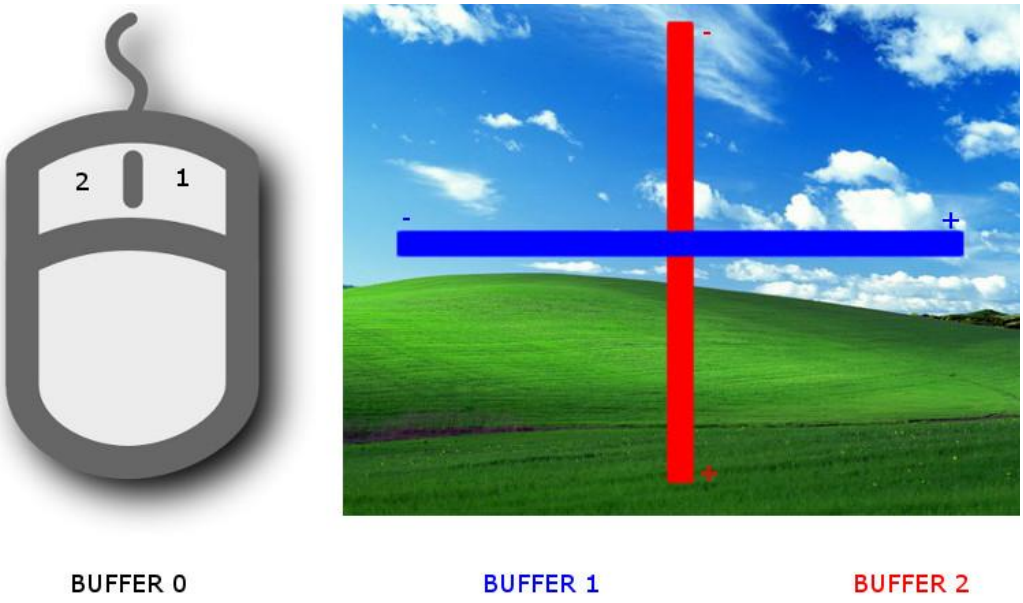


Figura 3.20 Correspondencia física del buffer de acciones del ratón.

El buffer 0 se corresponde con los botones derecho e izquierdo del ratón. El valor 1 corresponde con el botón derecho y el valor 2 corresponde con el botón izquierdo.

El buffer 1 se corresponde con el movimiento en el eje X del cursor del mouse. Un valor positivo en el buffer desplazará el cursor a la derecha y un valor negativo lo desplazará a la izquierda.

El buffer 2 se corresponde con el movimiento en el eje Y del cursor del mouse. Un valor positivo en el buffer desplazará el cursor hacia abajo y un valor negativo lo desplazará hacia arriba.

Buffer[2]	**Buffer[1]**	**Buffer[0]**
ΔY	**ΔX**	**B_L B_R**

El desplazamiento del cursor del ratón por el escritorio viene dado por los valores del buffer 1 y buffer 2, y se desplazará por coordenadas incrementales con respecto de su situación actual, moviéndose tanto más deprisa cuanto mayor sea el valor que hayamos cargado en el buffer.

Debido a esta configuración en los buffers, el cursor no se podrá desplazar en direcciones contrarias a la misma vez, ni podrá realizar la acción de pulsar los botones derecho e izquierdo simultáneamente. Por lo tanto el programa irá evaluando alternativamente los niveles de señal correspondientes a direcciones contrarias.

Es decir, en el caso de que se haya activado la señal de joystickArriba, el programa encenderá el led y cargará el valor correspondiente en el buffer, y no evaluará si la señal joystickAbajo está activa, ya que esta combinación no es posible. En el caso de que no esté la señal de joystickArriba encendida, deberá comprobar el estado de la señal joystickAbajo.

Si el programa detectara que la señal de joystickAbajo está activa actuará de la misma manera que con la señal anterior y, si no estuviese activa, apagará el led y borrará el buffer correspondiente poniéndolo a 0.

Tras esta evaluación, el programa analizará las señales de joystickDerecha y joystickIzquierda, de idéntica manera a la anterior y, tras terminar esta evaluación, pasará a evaluar las señales botonDerecho y botonIzquierdo.

Tras terminar la evaluación de todas las señales de entrada, comprobará si el microcontrolador está preparado para enviar los datos por USB y, en caso afirmativo, hará lo propio y en caso negativo terminará la función. El código de la función quedaría como sigue:

```
void Emulate_Mouse(void)
{
      if(joystickArriba == 1)
     //ESTRUCTURA IF-ELSE QUE DISCRIMINA QUE ENTRADAS
ESTÁN ACTIVAS
           {
           salidaLed = 1;
     //ENCIENDE EL LED
           sonidoPiezo();
     //LLAMADA AL SUBPROGRAMA SONIDOPIEZO()
           buffer [2] = -1;
           }
     else if(joystickAbajo ==1)
           {
           salidaLed = 1;
     //ENCIENDE EL LED
           sonidoPiezo();
     //LLAMADA AL SUBPROGRAMA SONIDOPIEZO()
           buffer [2] = 1;}
```

```
     else
          {
          salidaLed = 0;                              //APAGA
EL LED
          buffer [2] = 0;
          }
     if(joystickDerecha == 1)
          {
          salidaLed  = 1;
     //ENCIENDE EL LED
          sonidoPiezo();
     //LLAMADA AL SUBPROGRAMA SONIDOPIEZO()
          buffer [1] = 1;
          }
     else if(joystickIzquierda ==1)
          {
          salidaLed = 1;
     //ENCIENDE EL LED
          sonidoPiezo();
     //LLAMADA AL SUBPROGRAMA SONIDOPIEZO()
          buffer [1] = -1;
          }
     else
          {
          salidaLed = 0;                              //APAGA
EL LED
          buffer [1] = 0;
          }

     if(botonDerecho == 1)
          {
          salidaLed  = 1;
     //ENCIENDE EL LED
          sonidoPiezo();
     //LLAMADA AL SUBPROGRAMA SONIDOPIEZO()
        buffer [0] = 1;                               //MANDA
AL BUFFER EL CLICK DERECHO
          }
     else if(botonIzquierdo ==1)
          {
          salidaLed = 1;
     //ENCIENDE EL LED
          sonidoPiezo();
     //LLAMADA AL SUBPROGRAMA SONIDOPIEZO()
          buffer [0] = 2;
     //MANDA AL BUFFER EL CLICK IZQUIERDO
          }
```

```
else
          {
          salidaLed = 0;                              //APAGA
EL LED
          buffer [0] = 0;
          }
    if(HIDTxHandleBusy(lastTransmission) == 0)
    {
        //copy over the data to the HID buffer
        hid_report_in[0] = buffer[0];
        hid_report_in[1] = buffer[1];
        hid_report_in[2] = buffer[2];

        lastTransmission      =      HIDTxPacket(HID_EP,
(BYTE*)hid_report_in, 0x03);
    }
}//end Emulate_Mouse
```

Como se puede observar, el código podemos dividirlo en dos partes. La primera parte evalúa los valores de las entradas y la segunda parte se encarga de enviar la información por USB.

La programación de la primera parte del código se ha realizado con condicionales de tipo if-else, cargando en una variable de tipo vector los valores correspondientes a las acciones del ratón. En la segunda parte del código, el programa prepara el buffer USB con los valores del array preparado anteriormente y , si el dispositivo está listo, los enviará por el USB.

Cada vez que se active una de las entradas el programa debe de generar una realimentación acústica y visual. La realimentación visual se basa en el encendido de un diodo led que está conectado directamente a una patilla de salida, por lo que simplemente será necesario la activación a nivel alto de esa patilla para provocar su encendido.

La realimentación acústica viene dada por un altavoz piezoeléctrico que tendrá que ser excitado por una señal alterna. Esta señal alterna viene generada por la función *sonidoPiezo(),* cuyo flujograma es el siguiente:

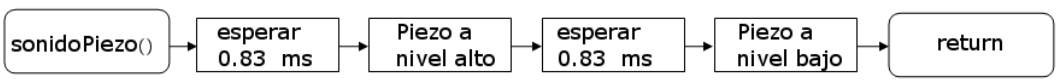


Figura 3.21 Flujograma de funcionamiento de la función sonidoPiezo()

Como se puede observar en el flujograma, el programa simplemente pone a nivel alto la patilla a la cual está conectada el piezoeléctrico tras esperar 0,83 milisegundos y, tras esperar otros 0.83 milisegundos, la vuelve a conectar a nivel bajo, generando una onda cuadrada de 600 Hz aproximadamente.

Debido a que esta función es llamada cada vez que el programa entra en el bucle *Emulate_Mouse(),* generará una onda lo suficientemente larga en el tiempo como para que pueda ser percibida por el oído humano. Esta onda tendrá la siguiente forma aproximada:

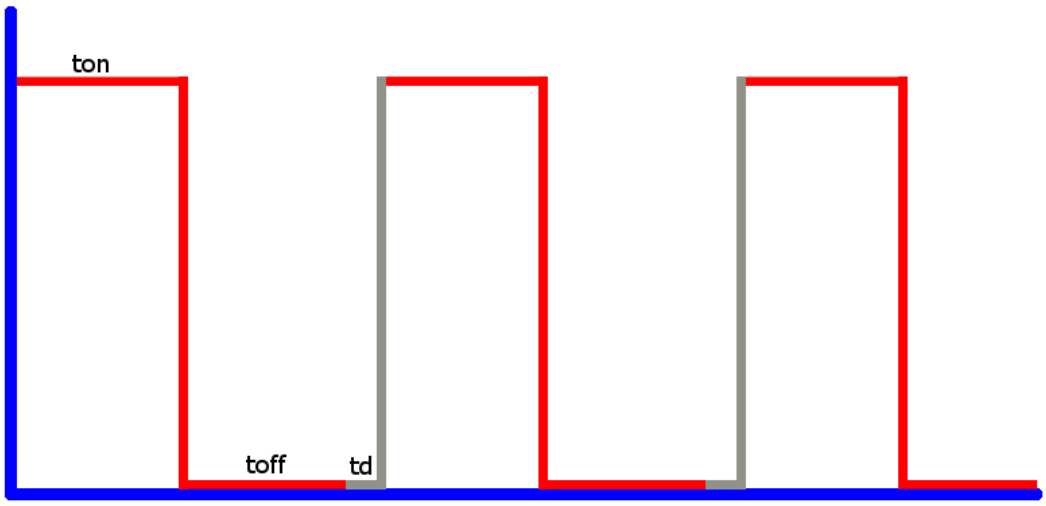


Figura 3.22 Forma de onda teórica de alimentación del piezoeléctrico.

Como se puede observar en la figura, se podrá tener la señal a nivel alto y a nivel bajo el tiempo que se le indique en la función (marcadas con *ton* y *toff*), apareciendo un tiempo de retardo (*td*) relacionado con el tiempo que tarda el programa principal a volver a ejecutar la función.

Debido a que la velocidad del microcontrolador es muy alta comparada con la frecuencia que se desea generar, este tiempo de retardo será tan pequeño que se puede despreciar, debido a que no se busca una gran precisión en la frecuencia de salida sino una mera señal auditiva de realimentación. El código de la función quedará como sigue.

```
void sonidoPiezo()//SUBPROGRAMA QUE ENVÍA AL PIEZOELECTRICO
UNA SEÑAL CUADRADA

{
                Delay1KTCYx(10);//ESPERA 0.83 MILISEGUNDOS
                salidaPiezo=1;//ENVÍA   UN   NIVEL   ALTO   AL
PIEZOELECTRICO
                Delay1KTCYx(10);//ESPERA 0.83 MILISEGUNDOS
                salidaPiezo=0;//ENVÍA   UN   NIVEL   BAJO   AL
PIEZOELECTRICO

}
```

Se puede observar que para el conteo el tiempo se utiliza la función Delay1KTCYx(). Esta función viene proporcionada por el entorno de programación en el archivo de cabecera delays.h y se puede utilizar con tan sólo incluirla.

La función Delay1KTCYx() mantiene el programa en espera un cierto número de instrucciones, que viene dado por el índice que le pasamos en la función multiplicado por 1000. En este caso, se conoce que el microcontrolador esperará 10000 instrucciones y , debido a que la frecuencia de trabajo es de 48 Mhz, se puede calcular que el tiempo de espera del microcontrolador será de:

$$t = 4(\frac{1}{f}) * 1000 * n$$

donde:

t: tiempo de espera (en segundos)
f: frecuencia de trabajo del microcontrolador(en hercios)
n: índice de la función

Por lo que, sustituyendo, nos da un tiempo de:

$$t = 4(\frac{1}{48 * 10^6}) * 1000 * 10 = 0.83\ mS$$

Por lo tanto, esperará 0.83 milisegundos antes de cambiar a nivel bajo y volverá a esperar otros 0.83 milisegundos antes de volver a pasar a nivel alto. Por lo tanto, la frecuencia de la señal cuadrada que estamos generando será de:

$$f = \frac{1}{ton + toff}$$

donde:

ton: tiempo a nivel alto(en segundos)
toff: tiempo a nivel bajo(en segundos)

Por lo tanto, sustituyendo:

$$f = \frac{1}{0.83 * 10^{-3} + 0.83 * 10^{-3}} = 600\ Hz$$

Con lo cual, se están generando ciclos de onda cuadrada de 600 Hz, que se irán repitiendo junto con un tiempo de retardo prácticamente nulo.

Tras la escritura del código, se ha compilado el proyecto y, tras la correción de pequeños errores de programación (cierre de paréntesis, finales de línea ect), se ha cargado el programa en la placa de pruebas y se han encontrado varios fallos en el funcionamiento consistentes en que el sonido generado por el sistema y la velocidad del cursor varía si se combinan dos o más botones a la vez.

Esto ocurre por la razón de que el programa, en vez de repetir un ciclo de la onda mandada al piezoeléctrico por cada vez que se envían datos por el buffer, repite dos o más veces este ciclo, dependiendo la cantidad de entradas activas a la vez.

A consecuencia de ello, el tiempo que debe de pasar hasta que se envían los datos por el buffer es mayor, lo que repercute en una mayor lentitud del desplazamiento del cursor por la pantalla (tanto más lento cuanto más veces se repita el ciclo).

La diferencia apreciable en el sonido generado por el piezoeléctrico viene dada por el tiempo de retardo generado por el ciclo de programa.

En el caso de que solamente haya una señal activa, la onda generada por el microcontrolador tendrá la forma anteriormente descrita.

En el caso de que haya dos entradas activas, se generarán dos ciclos de onda consecutivos y, tras ellos, el tiempo de retardo y, en el caso de que existan tres entradas activas, se generaran tres ciclos de onda antes del tiempo de retardo, como se puede ver en las siguientes figuras:

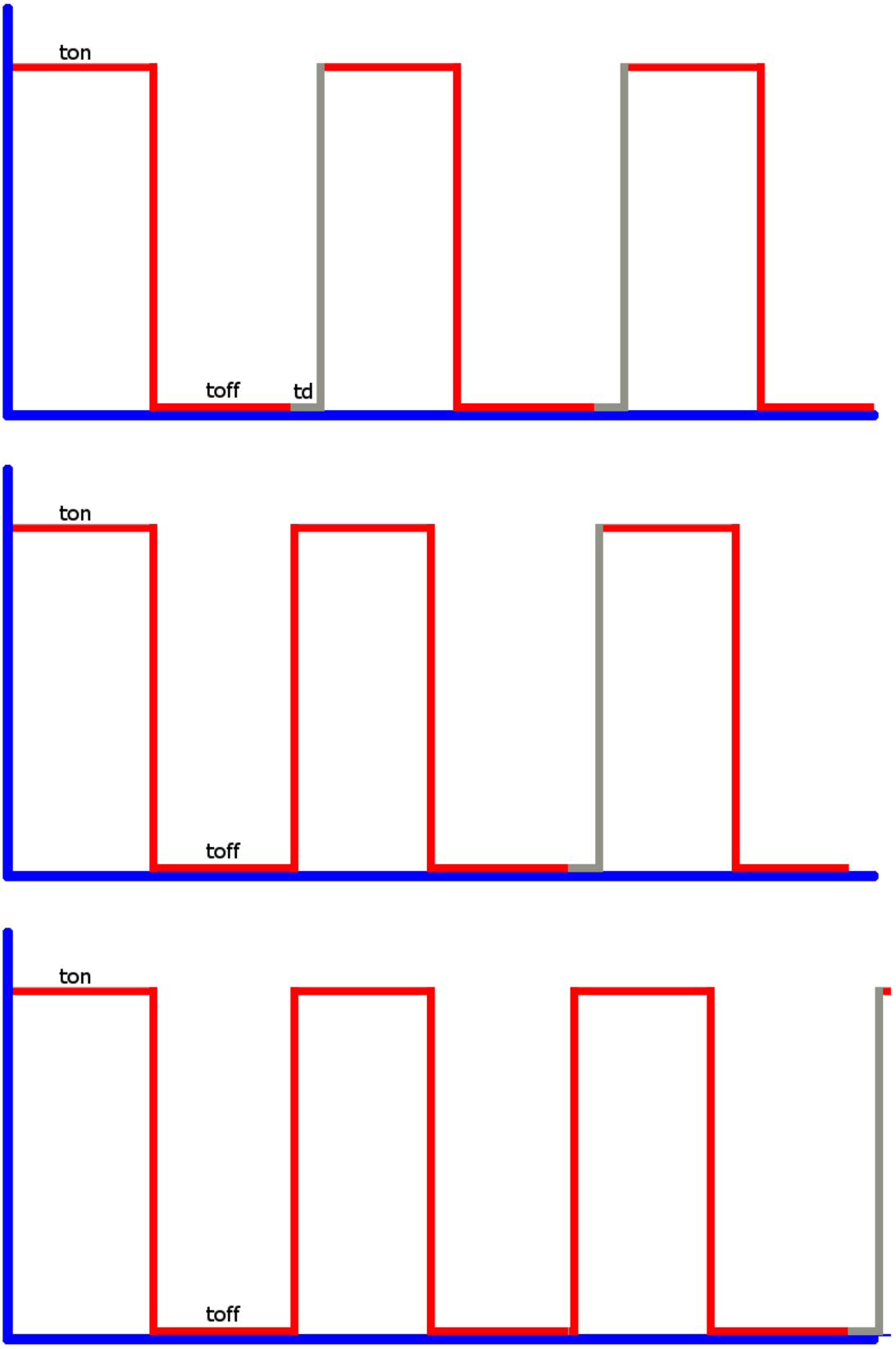
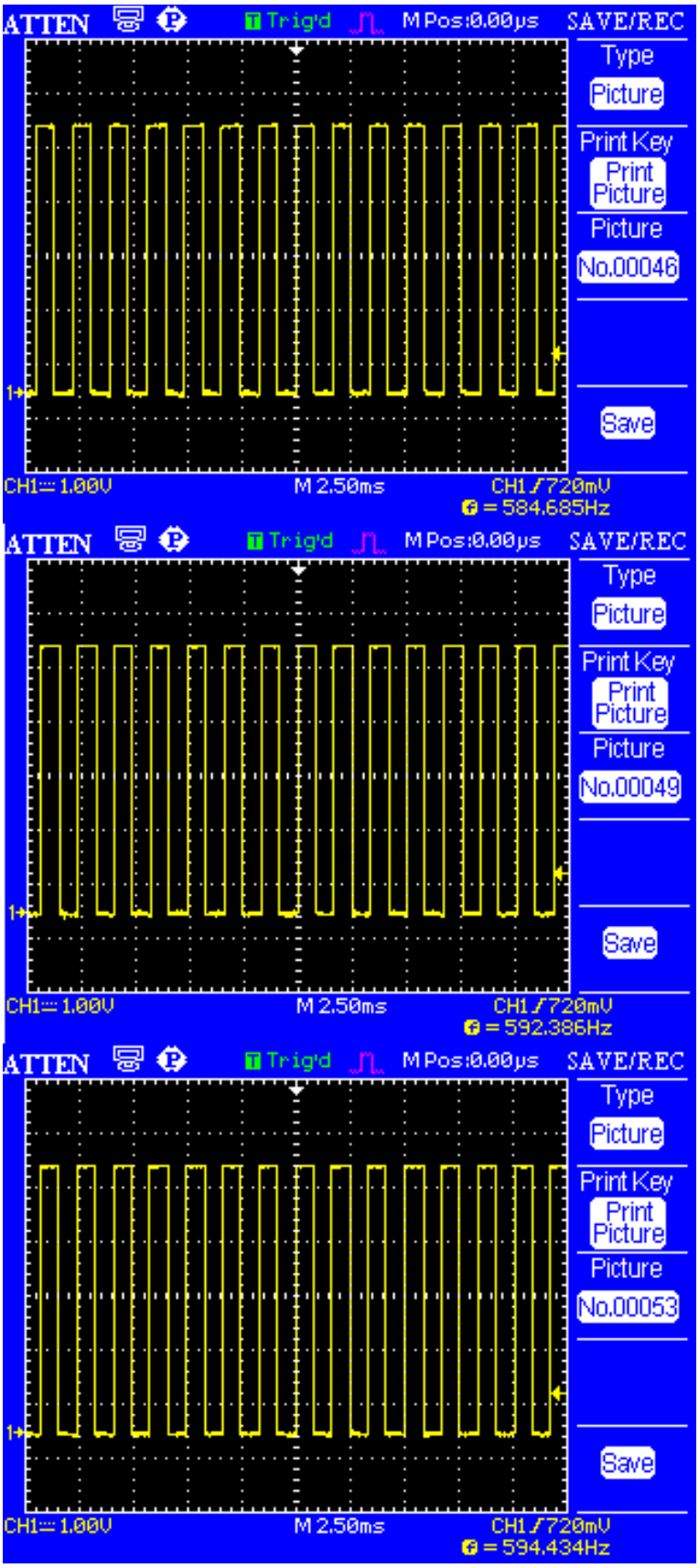
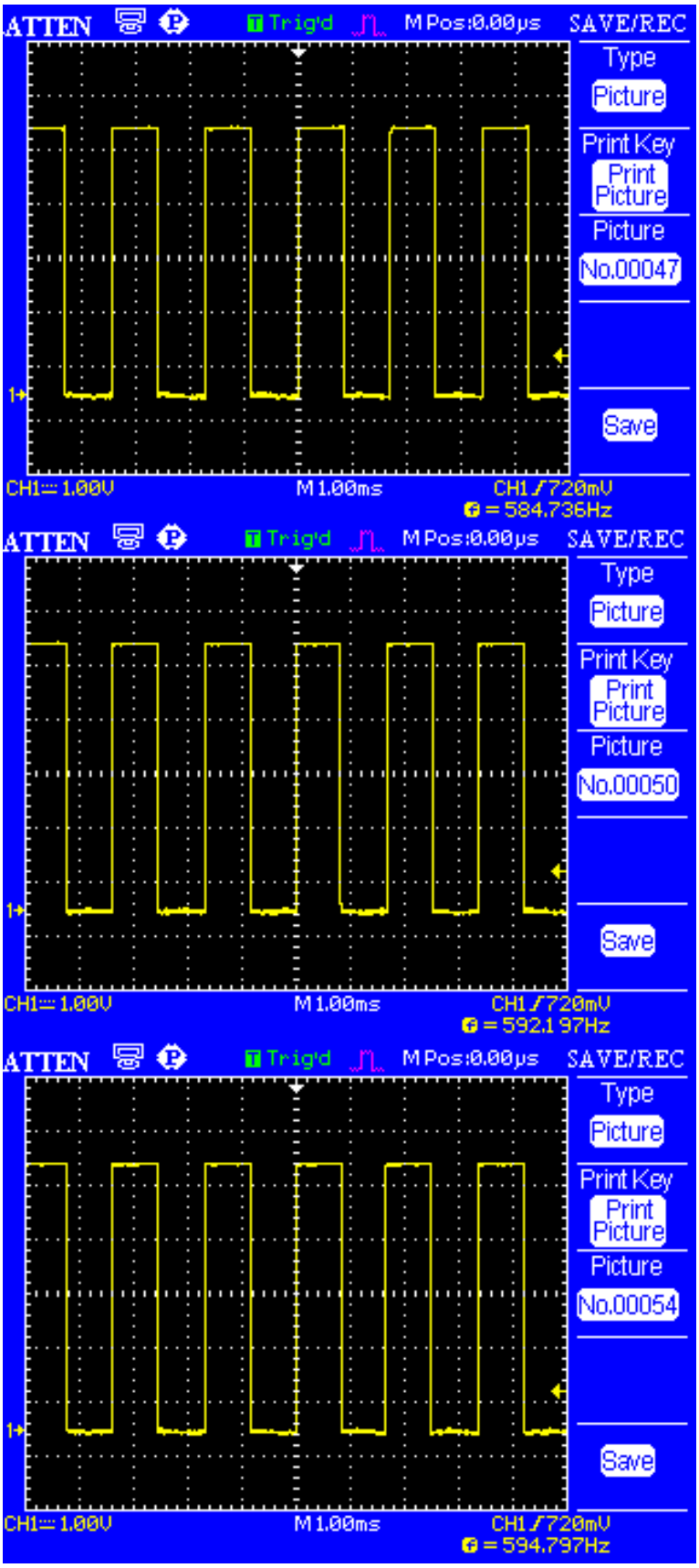
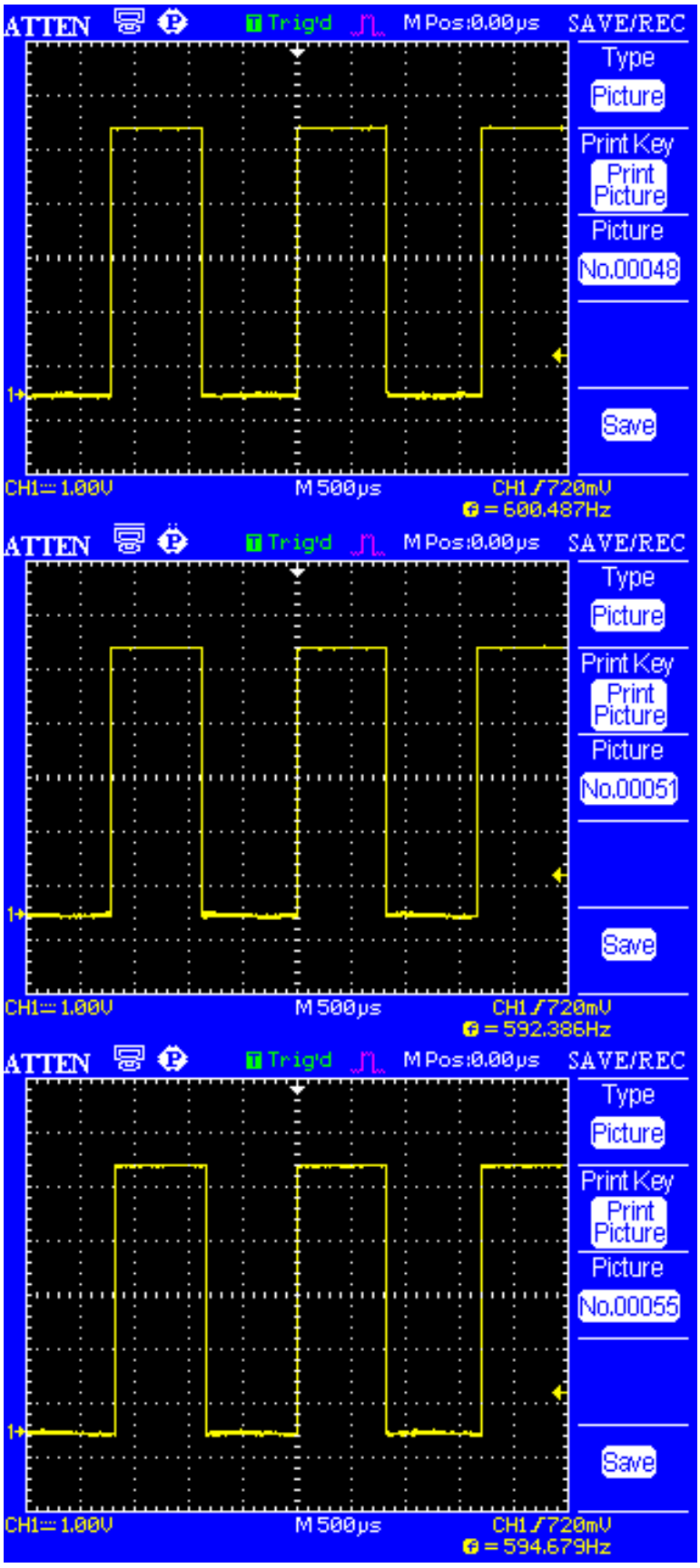


Figura 3.23 Formas de onda teóricas de alimentación del piezoelectrico


ATTEN
Trig'd
M Pos:0.00μs
SAVE/REC
Type
Picture
Print Key
Print Picture
Picture
No.00046
Save
CH1 1.00V
M 2.50ms
CH1 720mV
f = 584.685Hz
ATTEN
Trig'd
M Pos:0.00μs
SAVE/REC
Type
Picture
Print Key
Print Picture
Picture
No.00049
Save
CH1 1.00V
M 2.50ms
CH1 720mV
f = 592.386Hz
ATTEN
Trig'd
M Pos:0.00μs
SAVE/REC
Type
Picture
Print Key
Print Picture
Picture
No.00053
Save
CH1 1.00V
M 2.50ms
CH1 720mV
f = 594.434Hz


ATTEN
Trig'd
M Pos:0.00µs
SAVE/REC
Type
Picture
Print Key
Print Picture
Picture
No.00047
Save
CH1 1.00V
M 1.00ms
CH1 720mV
= 584.736Hz
ATTEN
Trig'd
M Pos:0.00µs
SAVE/REC
Type
Picture
Print Key
Print Picture
Picture
No.00050
Save
CH1 1.00V
M 1.00ms
CH1 720mV
= 592.197Hz
ATTEN
Trig'd
M Pos:0.00µs
SAVE/REC
Type
Picture
Print Key
Print Picture
Picture
No.00054
Save
CH1 1.00V
M 1.00ms
CH1 720mV
= 594.797Hz



Figuras 3.24, 3.25 y 3.26 Lecturas del osciloscopio de la señal del piezoeléctrico.

Como se pueden observar en las ondas tomadas con el osciloscopio, la frecuencias con la que se excita el piezoeléctrico son prácticamente idénticas, pero la pequeña diferencia existente entre estas genera la ligera diferencia de sonido apreciable por el oído humano.

Para corregir estos errores, se han independizado el código referente a la realimentación, creando un pequeño bloque al final de la evaluación de las entradas.

El flujograma de la función Emulate_Mouse() con el nuevo bloque queda como sigue:

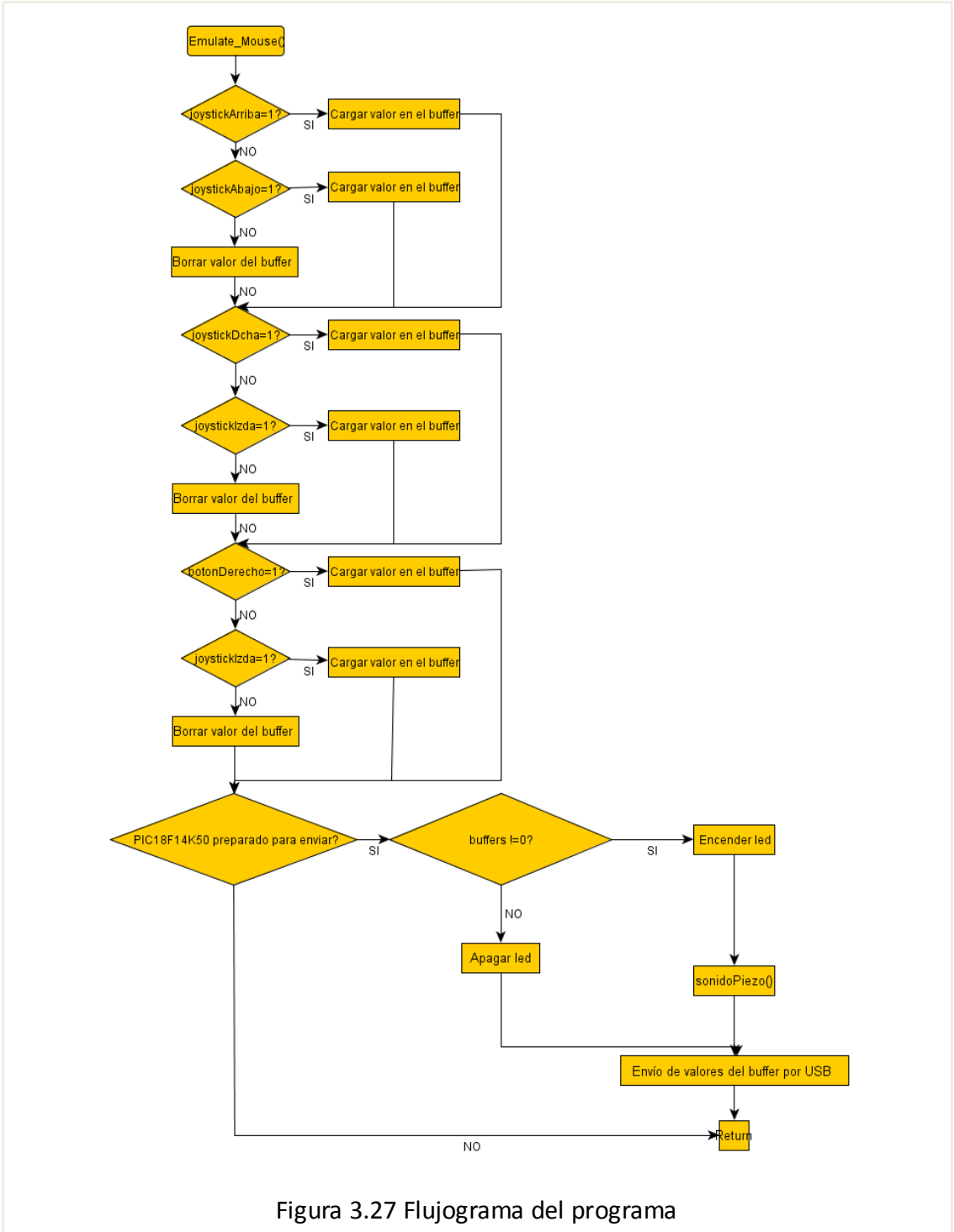


Figura 3.27 Flujograma del programa

Este nuevo bloque, colocado después del envío de valores por el buffer, evaluará si hay alguna de las entradas activas observando el buffer de direcciones, y lanzará las señales de realimentación en caso afirmativo. En el caso de que no se haya pulsado ninguna, no se lanzará la función de generación de la onda cuadrada y se desactivará la salida del led.

De esta manera, se ha optimizado el programa, eliminando código de una acción repetitiva, aligerando el código y haciéndolo más legible.

El nuevo código de la función Emulate_Mouse() queda como sigue:

```
void Emulate_Mouse(void)
{
      if(joystickArriba == 1)
           {
           buffer [2] = -1;
           }
     else if(joystickAbajo ==1)
           {
           buffer [2] = 1;
           }
     else
           {
           buffer [2] = 0;
           }
     if(joystickDerecha == 1)
           {
           buffer [1] = 1;
           }
     else if(joystickIzquierda ==1)
           {
           buffer [1] = -1;
           }
     else
           {
           buffer [1] = 0;
           }

     if(botonDerecho == 1)
           {
        buffer [0] = 1;
           }
```

```
     else if(botonIzquierdo ==1)
          {
          buffer [0] = 2;
          }
     else
          {
          buffer [0] = 0;
          }

    if(HIDTxHandleBusy(lastTransmission) == 0)
    {
        //copy over the data to the HID buffer
        hid_report_in[0] = buffer[0];
        hid_report_in[1] = buffer[1];
        hid_report_in[2] = buffer[2];

        lastTransmission = HIDTxPacket(HID_EP,
(BYTE*)hid_report_in, 0x03);

if(buffer[0]||buffer[1]||buffer[2] != 0)
                {
                salidaLed =1;
                sonidoPiezo();
                }
          else
                {
                salidaLed=0;
                }
    }
}//end Emulate_Mouse
```

Tras la programación de las modificaciones , se ha compilado el proyecto, se ha cargado en el microcontrolador y se ha procedido a la comprobación del funcionamiento del mismo, comportándose de la manera esperada.

Aunque en este caso no se han tenido problemas encontrando rebotes en los interruptores, no significa que no se pueda dar algún tipo de rebote con el desgaste de los pulsadores o a la hora de que un usuario reproduzca el proyecto, por lo que se ha realizado una sencilla rutina anti-rebotes similar a la que Microchip implementaba en el ejemplo para los pulsadores sw2 y sw3, adaptada para la evaluación de todas las entradas, cómo se puede observar en el flujograma.

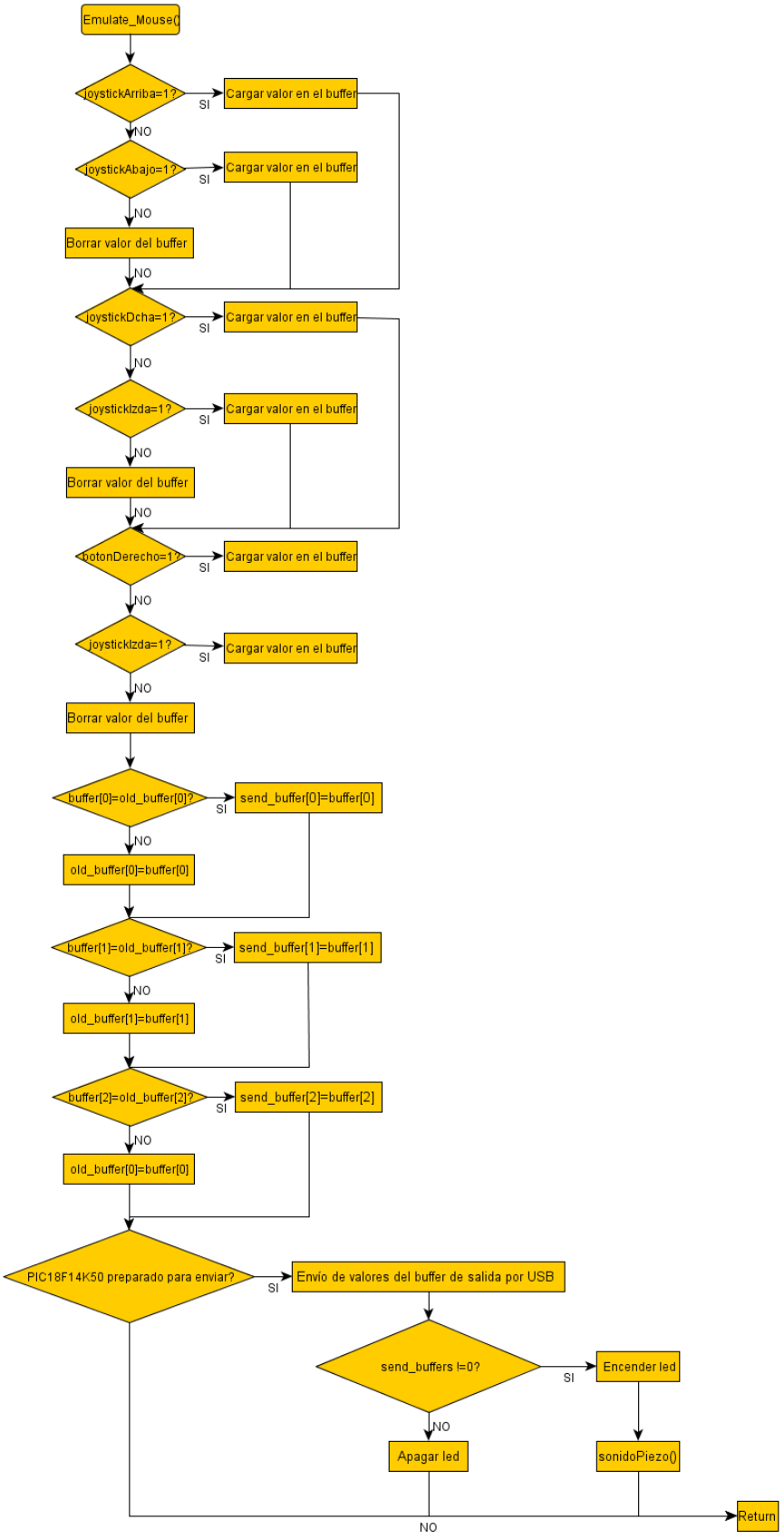


Figura 3.28 Flujograma del programa

En este caso, se han creado dos nuevas variables, *old_buffer[]* y *send_buffer[].* La variable *old_buffer[]* se encargará de guardar los valores de las entradas en el instante anterior y la variable send_buffer[] almacenará los valores que se enviarán por USB.

El programa comparará los valores de las componentes de buffer[] y old_buffer[]. En el caso de que alguna de las componentes de las variables no coincidan , se guardará este componente de buffer[] en su lugar correspondiente de old_buffer[] y, en el caso de que coincidan, se cargará el valor de esa componente de la variable buffer en send_buffer().

De esta manera, se detectan los típicos cambios de flanco rápidos en el nivel de entrada de las patillas producidos por los rebotes del pulsador y tan sólo se enviará el valor que detectamos en la entrada cuando se ha comprobado que éste es estable.

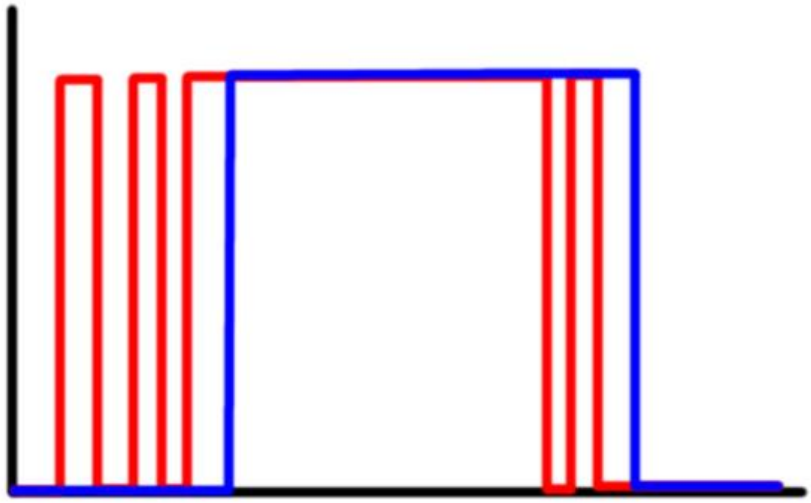


Figura 3.29 Señal causada por los rebotes y señal de la función anti rebotes

Como se puede observar en la figura, la señal enviada por el buffer (línea azul) tan sólo cambia de nivel cuando la señal de entrada (línea roja) se mantiene estable en el tiempo.

El nuevo código de la función Emulate_Mouse() queda como se muestra a continuación:

```
void Emulate_Mouse(void)
{
      if(joystickArriba == 1)
     //ESTRUCTURA IF-ELSE QUE DISCRIMINA QUE ENTRADAS
ESTÁN ACTIVAS
          {
          buffer [2] = -1;
          }
```

```
else if(joystickAbajo ==1)
          {
          buffer [2] = 1;
          }
     else
          {
          buffer [2] = 0;
          }
     if(joystickDerecha == 1)
          {
          buffer [1] = 1;
          }
     else if(joystickIzquierda ==1)
          {
          buffer [1] = -1;
          }
     else
          {
          buffer [1] = 0;
          }

     if(botonDerecho == 1)
          {
        buffer [0] = 1;
          }
     else if(botonIzquierdo ==1)
          {
          buffer [0] = 2;
          }
     else
          {
          buffer [0] = 0;
          }
//RUTINA ANTI REBOTES
     if(buffer[0]==old_buffer[0])
          {
          send_buffer[0]=buffer[0];
          }
     else
          {
          old_buffer[0]=buffer[0];
          }
     if(buffer[1]==old_buffer[1])
          {
          send_buffer[1]=buffer[1];
          }
     else
          {
          old_buffer[1]=buffer[1];
          }
```

```
     if(buffer[2]==old_buffer[2])
          {
          send_buffer[2]=buffer[2];
          }
     else
          {
          old_buffer[2]=buffer[2];
          }
    if(HIDTxHandleBusy(lastTransmission) == 0)
    {
        //copy over the data to the HID buffer
        hid_report_in[0] = send_buffer[0];
        hid_report_in[1] = send_buffer[1];
        hid_report_in[2] = send_buffer[2];

        lastTransmission      =      HIDTxPacket(HID_EP,
(BYTE*)hid_report_in, 0x03);
    }
     if(send_buffer[0]||send_buffer[1]||send_buffer[2]
!= 0)
          {
          salidaLed =1;
          sonidoPiezo();
          }
     else
          {
          salidaLed=0;
          }
}//end Emulate_Mouse
```

Se puede observar como se ha realizado la implementación de la rutina anti rebotes utilizando condicionales tipo *if-else* que pasan el valor de la variable *buffer[]* a *old_buffer[]* o a *send_buffer[]* según corresponda.

Tras la programación del código y su compilación se ha cargado en el microcontrolador, funcionando correctamente. Con éste último paso, se da por terminada la tarea de programación del código.

4. Hardware

4.1 Actuadores

Para facilitar la realización del prototipo, y debido a que no se disponen de todos los actuadores existentes en el mercado, los actuadores que serán utilizados en el proyecto son un joystick y dos pulsadores de máquinas recreativas, debido a su bajo coste (en comparación con algunos interruptores para discapacitados), fácil adquisición en el mercado, alta durabilidad y rápida reparación.

Figura 4.1 Actuadores tipo Joystick y Pulsador, junto con un microrruptor, base de su funcionamiento.

Como se puede observar en la figura, se ha utilizado un joystick de 8 posiciones muy robusto y desmontable. El joystick consta de una palanca de movimiento y 4 microrruptores alojados en la base.

El joystick acciona uno o dos de sus microrruptores al moverse, dependiendo si la dirección del movimiento es en los ejes X e Y o si es una combinación de los mismos. Utilizaremos el joystick para generar las señales TTL referidas al movimiento del cursor por la pantalla.

Para captar los dos botones del ratón, se han utilizado dos pulsadores de máquinas tragaperras, que también accionan un microrruptor al ser pulsados. Aunque estos pulsadores llevan incorporadas unas lámparas en su interior, se ha decidido prescindir de las mismas como realimentación luminosa, debido a que son lámparas convencionales de filamento de 12 V y su uso iba a elevar el nivel de complejidad de la placa para conseguir este voltaje, sin contar el incremento de consumo del dispositivo por usar estas lámparas, lo que llevaría a comprometer la alimentación por USB de la misma.

En el caso de rotura de los periféricos de entrada, la reparación de todos los componentes es muy sencilla, tan sólo hay que desmontar el microrruptor roto y sustituirlo por uno en perfectas condiciones.

4.2 Elaboración del PCB.

Para el desarrollo del proyecto se ha decidido fabricar una placa de pruebas previa a la elaboración de la placa de circuito impreso final, para corregir posibles errores en la distribución del circuito y realizar las pruebas del software necesarias en ella.

Para la elaboración de la placa de pruebas, se ha optado por la implementación sobre placa de circuito impreso perforada. La implementación en la placa perforada se ha realizado según el diseño del esquemático que se iba a utilizar para la realización del PCB definitivo.

4.2.1 Software utilizado para el diseño del PCB.

Figura 4.2 Logotipo de Eagle

Dentro de los múltiples programas de diseño de circuitos y placas electrónicas existentes en el mercado, se ha optado por la utilización de EAGLE (*Easily Applicable Graphical Layout Editor*), debido a que se distribuye con una licencia freeware para placas de reducido tamaño con dos layouts como máximo para diseños no comerciales, con el fin de facilitar la modificación libre del diseño por parte del usuario.

Características de EAGLE 6.4.0

-Fácil aprendizaje.

- Idénticas interfaces de usuario para el editor de esquemático, de librerías y de layout.
- Soporte gratuito.
- Compatible con Windows, Linux y Mac OS.

-Fácil de comprar.

- Compra online o a través de distribuidores autorizados.
- Sin cuota de mantenimiento.

-Fácil de usar.

- Incorpora un lenguaje de programación de usuario (ULP) flexible(lenguaje tipo C para la importación y exportación de datos para la realización de comandos autodefinidos).
- Exportación a archivos Gerber (formato de creación de PCB`s industrial).
- Importación on-line de productos e información de precios de los mismos mediante DesignLink(software de conexión con la base de datos de Premier Farnell).
- Foros de usuarios activos.

-Requerimientos del sistema

- *Windows:* Windows Xp, Windows Vista o Windows 7.
- *Linux:* Basados en el kernel 2.6 para procesadores Intel, X11 con una profundidad de color mínima de 8 bpp y con tiempo de ejecución de 32 bits para las siguientes librerías: libpng14.so.14, libssl.so.1.0.0, libcrypto.so.1.0.0 y libjpeg.so.8.
- *Mac OS:* Mac OS X versión 10.6 o 10.7, (con compatibilidad para la 10.8 en breve).

Restricciones de la licencia Eagle Light Edition.

- Área de diseño de placa limitada a 100 x 80 mm.
- Sólo dos capas de diseño (top y bottom).
- Tan sólo se puede crear un esquema por proyecto.
- Soporte limitado vía e-mail o mediante la utilización de los foros.
- Uso limitado a aplicaciones que no produzcan beneficios o para placas de evaluación.
- Existe el permiso de distribución de EAGLE con distribuciones de Linux con colecciones de software o CD-ROMs, mediante información a la empresa desarrolladora.

4.2.2 Diseño del esquemático.

Como paso previo a la realización del esquemático, se han diseñado los layouts del microcontrolador en Eagle, debido a que no existe este componente por defecto en las librerías del programa en el momento de la elaboración del esquemático.

Se ha tomado como referencia para la creación de la representación para el esquemático el pineado del encapsulado de 20 pines tipo PDIP que Microchip proporciona en la hoja de características, debido a que resulta más sencilla la implementación del PCB de este tipo de encapsulados por parte de un aficionado.

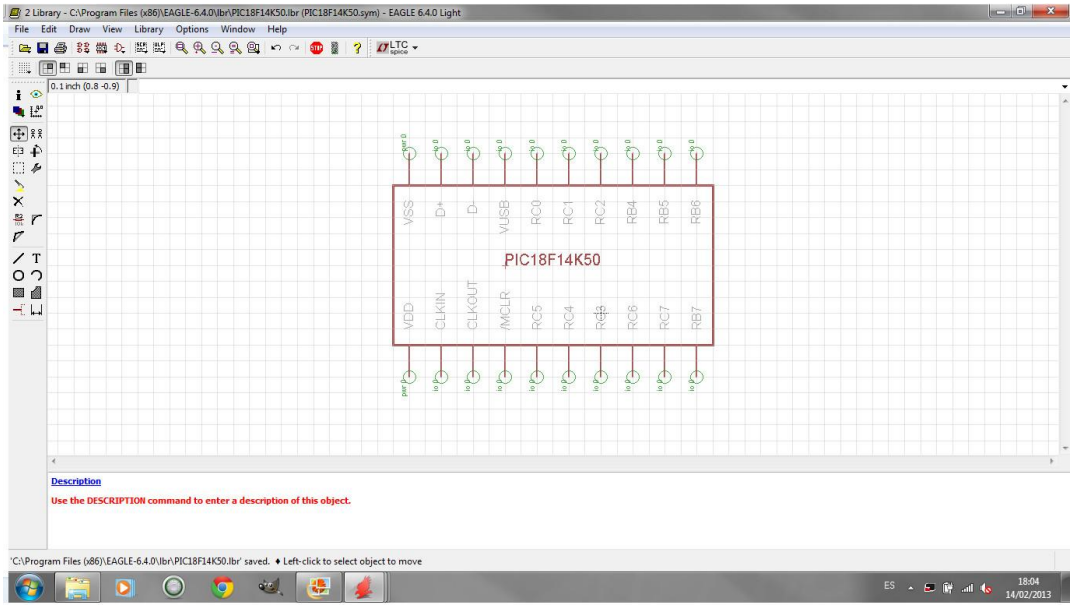


Figura 4.3 - Símbolo de esquemático

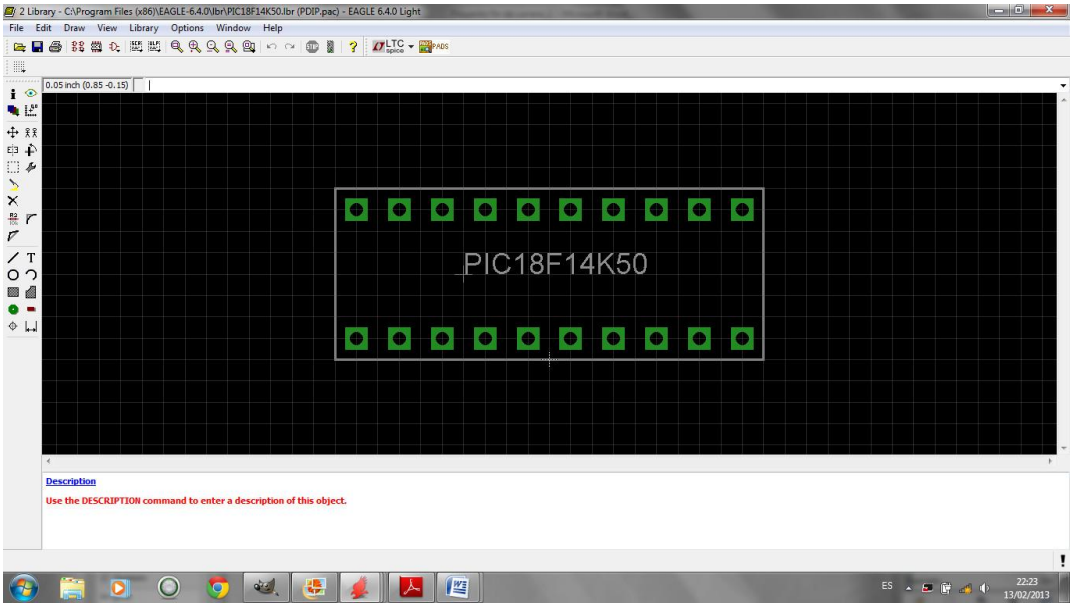


Figura 4.4 - Símbolo del layout

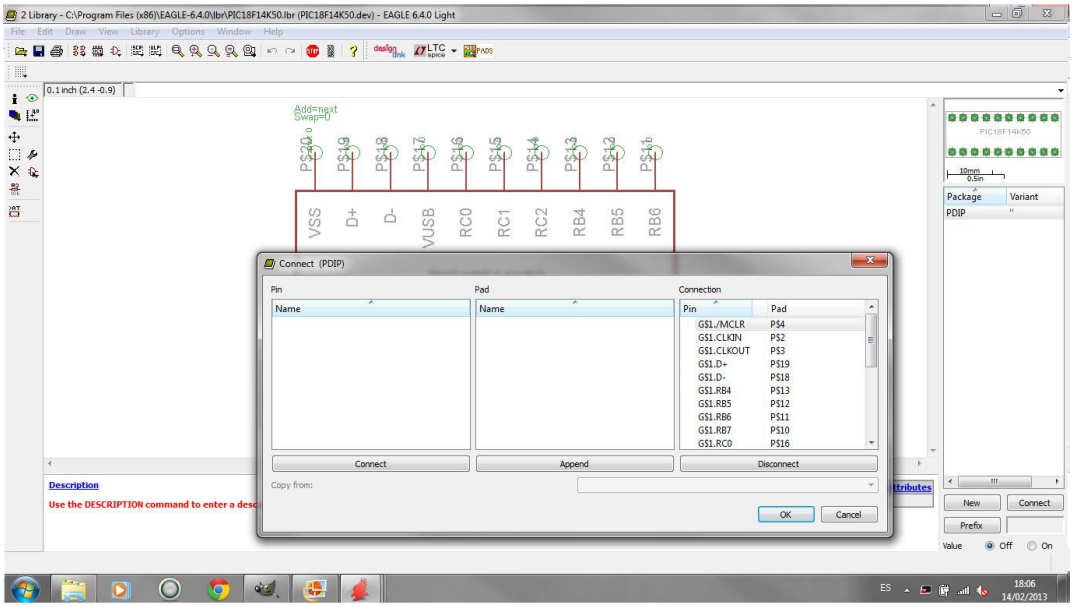


Figura 4.5 - Componente terminado, con asignación de pines realizada

Terminado el modelado del microcontrolador, se ha procedido al diseño del esquemático.

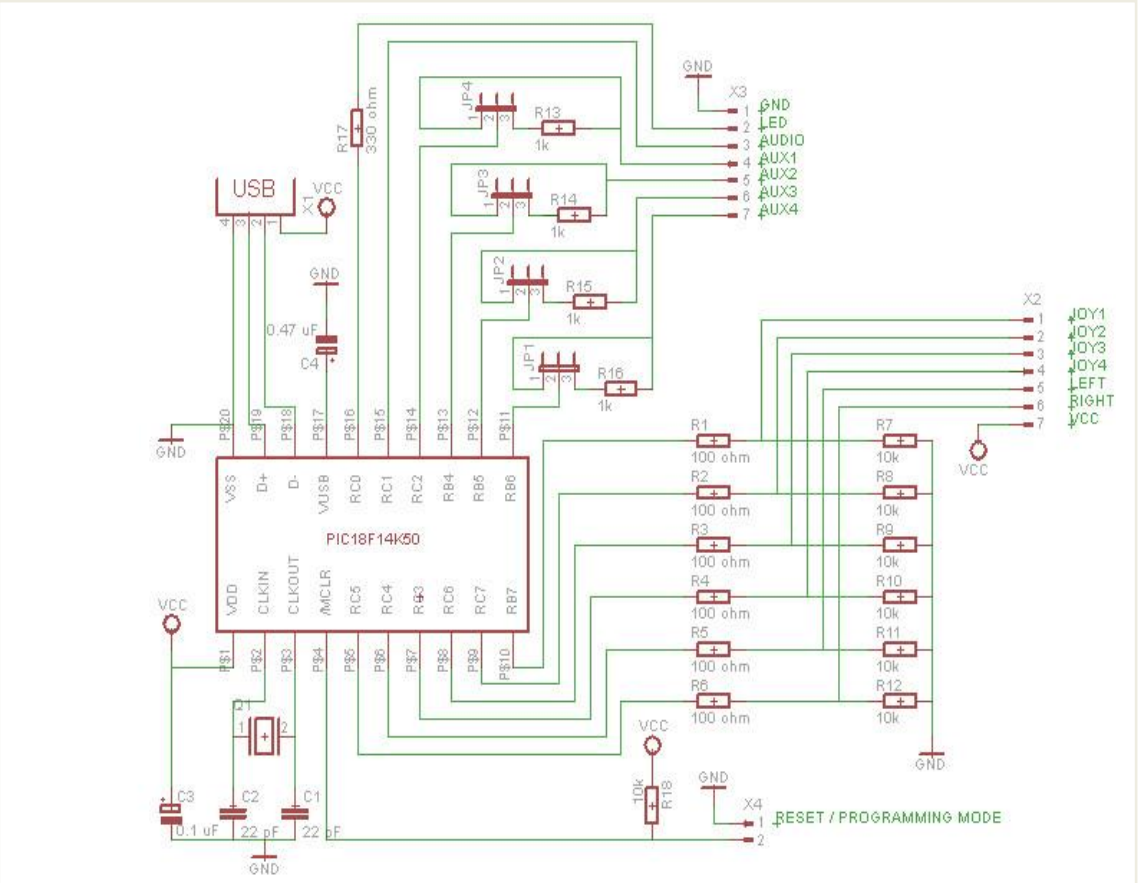


Figura 4.6 - Esquemático

Partlist

Exported from usb mouse.sch at 10/06/2013 10:01:35

EAGLE Version 6.4.0 Copyright (c) 1988-2013 CadSoft

Assembly variant:

Part	Value	Device	Package	Library	Sheet
C1	22 pF	C-EU025-024X044	C025-024X044	rcl	1
C2	22 pF	C-EU025-024X044	C025-024X044	rcl	1
C3	0.1 uF	CPOL-EUE1.8-4	E1,8-4	rcl	1
C4	0.47 uF	CPOL-EUE1.8-4	E1,8-4	rcl	1
JP1		JP2E	JP2	jumper	1
JP2		JP2E	JP2	jumper	1
JP3		JP2E	JP2	jumper	1
JP4		JP2E	JP2	jumper	1
Q1		XTAL	QS	special	1
R1	100 ohm	R-EU_0204/7	0204/7	rcl	1
R2	100 ohm	R-EU_0204/7	0204/7	rcl	1
R3	100 ohm	R-EU_0204/7	0204/7	rcl	1
R4	100 ohm	R-EU_0204/7	0204/7	rcl	1
R5	100 ohm	R-EU_0204/7	0204/7	rcl	1
R6	100 ohm	R-EU_0204/7	0204/7	rcl	1
R7	10k	R-EU_0204/7	0204/7	rcl	1
R8	10k	R-EU_0204/7	0204/7	rcl	1
R9	10k	R-EU_0204/7	0204/7	rcl	1
R10	10k	R-EU_0204/7	0204/7	rcl	1
R11	10k	R-EU_0204/7	0204/7	rcl	1
R12	10k	R-EU_0204/7	0204/7	rcl	1
R13	1k	R-EU_0204/7	0204/7	rcl	1
R14	1k	R-EU_0204/7	0204/7	rcl	1
R15	1k	R-EU_0204/7	0204/7	rcl	1
R16	1k	R-EU_0204/7	0204/7	rcl	1
R17	330 ohm	R-EU_0204/7	0204/7	rcl	1
R18	10k	R-EU_0204/7	0204/7	rcl	1
U$1	PIC18F14K50	PIC18F14K50	PDIP	PIC18F14K50	1
X1		PN61729	PN61729	con-berg	1
X2		180G-7	180G-7	con-weidmueller-sl35	1
X3		180G-7	180G-7	con-weidmueller-sl35	1
X4		180G-2	180G-2	con-weidmueller-sl35	1

Listado de componentes

Como se puede observar en el esquema, el microcontrolador tiene conectado un cristal de 12 Mhz, que marcará la frecuencia principal de trabajo, con sus respectivos condensadores.

Se ha configurado la patilla /MCLR para poder realizar en tiempo de ejecución del programa un reinicio del mismo y para indicar al microcontrolador en el arranque que entre en modo programación USB, cargando su bootloader.

Se ha conectado una regleta de 2 pines para realizar el montaje externo de un pulsador normalmente abierto, que tras ser pulsado forzará un cero lógico en esta patilla que provocará la acción en el dispositivo. Para ello, se ha dispuesto una resistencia de 10k entre alimentación y la patilla, que mantendrá la misma forzada a un uno lógico hasta que se active el pulsador.

En cuanto a la configuración de los pines de entrada, se ha decidido utilizar seis pines, que corresponderán cada uno de ellos con un movimiento del ratón (arriba, abajo, derecha, izquierda, botón derecho y botón izquierdo).

Estos pines están conectados a una serie de resistencias de pull down conectadas a unas regletas que facilitarán la conexión de los periféricos a la placa. Para conseguir una mayor limpieza en la distribución de los componentes en la placa de circuito impreso, se han escogido los pines RC5, RC4, RC3,RC6,RC7 y RB7 por estar contiguos en el integrado.

Se ha colocado una regleta adicional conectada a alimentación, para poder enviar señal de alimentación en el caso de que fuera necesario.

La patilla RC1 estará conectada a una regleta en donde irá conectado un altavoz piezoeléctrico que generará los pulsos auditivos de realimentación acústica.

La patilla RC0 estará conectada a una regleta en donde irá conectado un diodo led rojo estándar que generará las señales de realimentación luminosas. Para ajustar la tensión de la patilla a la necesaria para encender el led ocasionar su destrucción, se ha colocado una resistencia de 330 ohm.

Las patillas RC2,RB4,RB5 y RB6 se han dejado como patillas auxiliares, pensando en una posible conexión futura de mas periféricos de entrada o para otros fines. Se han dispuesto unos jumpers que permiten conectar las patillas, bien al aire, bien a una resistencia de 1k, antes de conectarse a las regletas.

Tanto las regletas auxiliares como las regletas de realimentación están unidas junto con otra regleta auxiliar que estará conectada a masa para facilitar la conexión de los dispositivos de realimentación.

Junto con los condensadores necesarios para el correcto funcionamiento tanto del USB como del cristal oscilador y como la alimentación, el esquemático del circuito se da por terminado.

4.2.3 Elaboración del PCB

Como se puede observar en la figura, se han seguido tres métodos distintos en la elaboración del PCB. El primero que se ha realizado ha sido el trabajo en placa perforada, para comprobar el correcto funcionamiento tanto del diseño hardware como software.

A continuación se ha procedido a la elaboración de una placa de circuito impreso siguiendo el método de insolación por fotolito. El tercer y último método de fabricación del PCB ha sido su creación mediante una fresadora de prototipado.

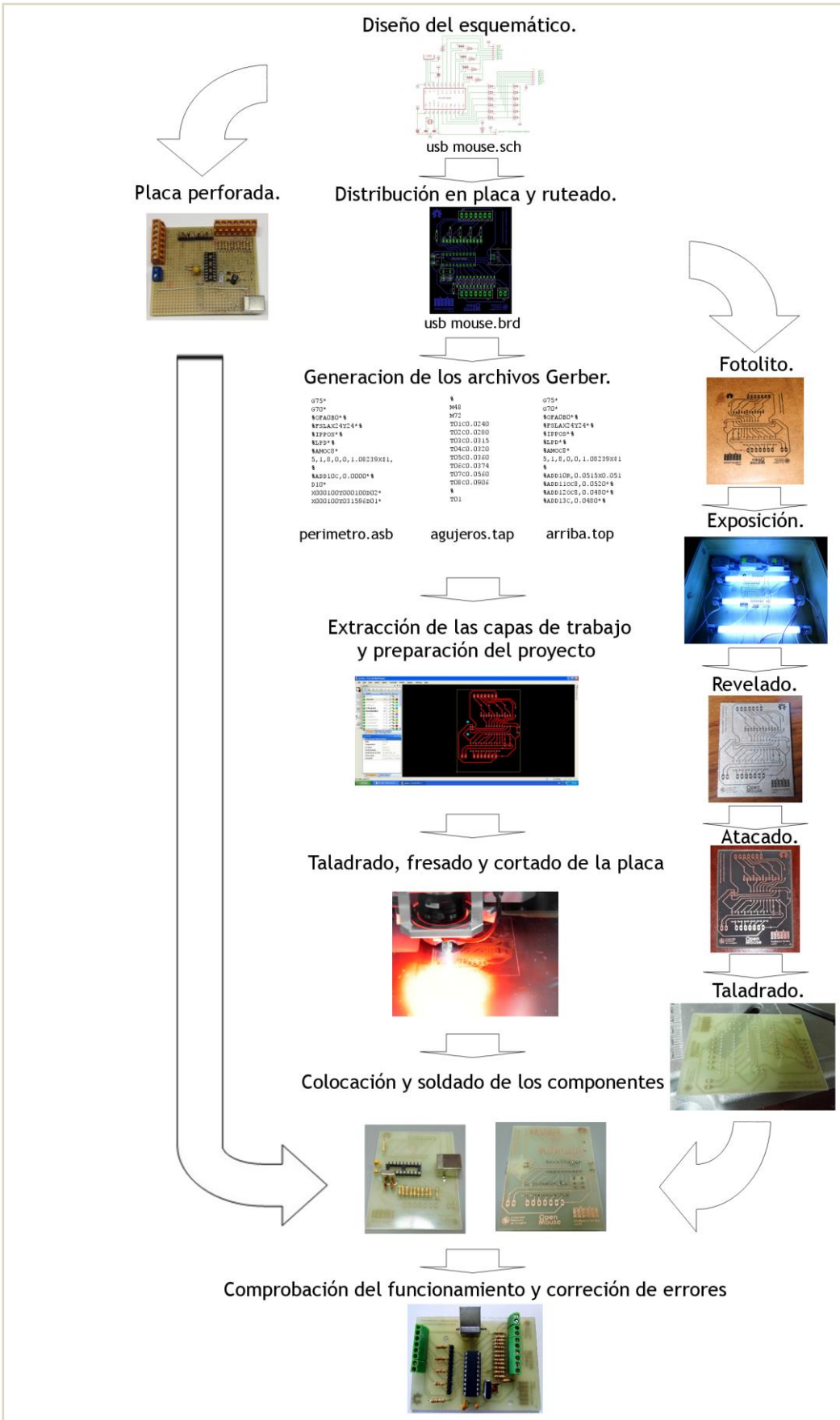


Figura 4.7 Elaboración de las distintas PCB`s

Tan sólo se han poblado las dos primeras placas, la primera para realizar las pruebas y la segunda para comprobar que el layout creado ha sido correcto. Debido a que el layout de la PCB creada por insolación y el de la PCB creada mediante la fresadora tienen el mismo diseño, no se ha considerado necesario la población de esta última.

4.2.3.1 Elaboración de la placa de circuito perforada.

Siguiendo el esquemático e intentando aprovechar el espacio de la placa para que no dificultara la creación de las pistas, se ha montado el circuito sobre una placa perforada de prototipado, con la intención de realizar las pruebas necesarias y hacer llegar el proyecto a aquellas personas de escasos recursos técnicos para la elaboración de placas.

Se han tenido que realizar varios puentes de pistas en la cara superior, así como un puente con hilo de wrapping en la cara inferior. Por sobrecalentamiento de la placa a la hora de realizar las soldaduras, se han despegado varios pads, por lo que se ha tenido que realizar otro puente en la cara inferior de la placa.

Tras el montaje y testeo de la placa, se ha insertado un microcontrolador, programado previamente con el bootloader, y se han cargado los diversos programas, funcionando a la perfección en cuanto a hardware se refiere.

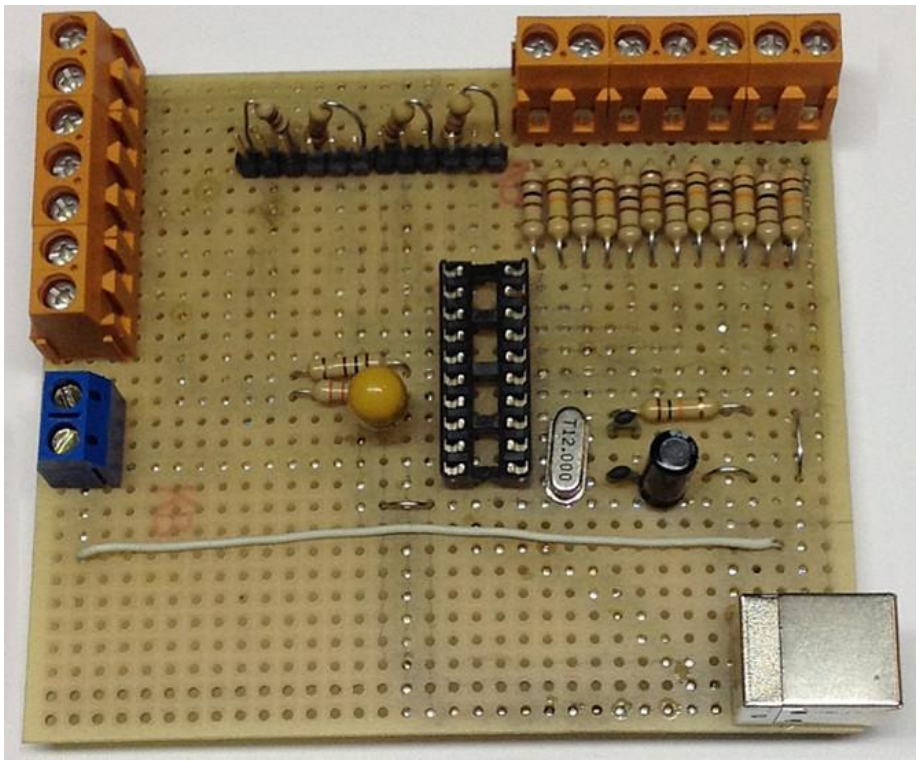


Figura 4.8 - Cara superior de la tarjeta perforada de pruebas.

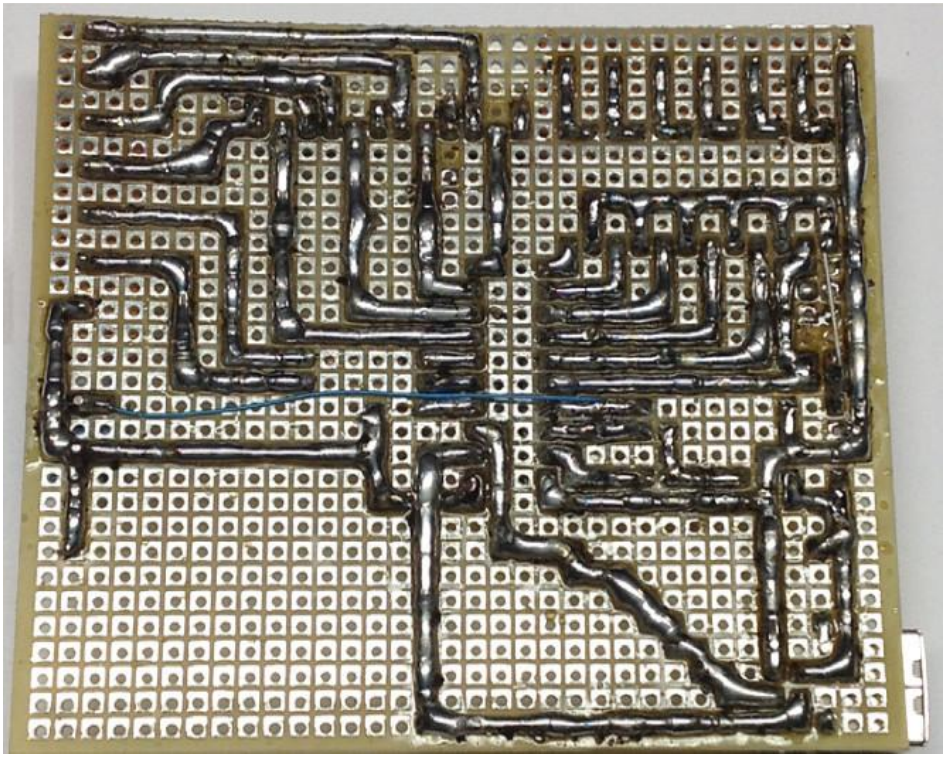


Figura 4.9 - Cara inferior espejada de la tarjeta perforada de pruebas.

4.2.3.2 Elaboración del layout de las placas de circuito impreso

Tras comprobar que el diseño de la placa y los programas funcionan correctamente, se ha procedido a diseñar el layout para realizar las placas de circuito impreso mediante insolación y mediante fresadora de prototipado rápido.

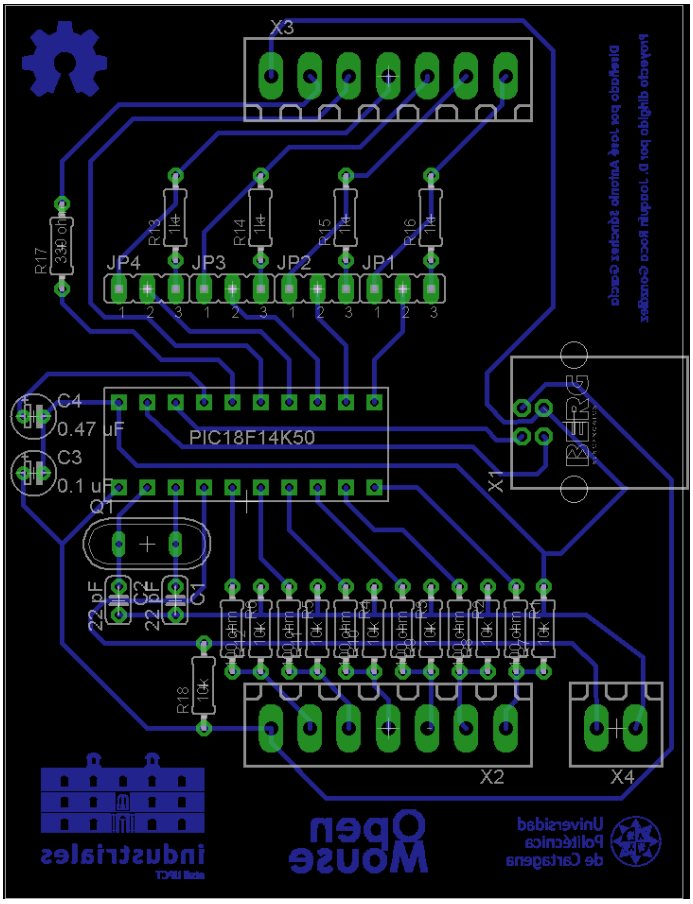


Figura 4.10 - Layout del PCB

Como se puede observar en la figura, se han dispuesto todos los componentes buscando bastante similitud con el esquemático, a fin de facilitar la comprensión entre ambos.

Se ha intentado utilizar el suficiente espacio como para que la distribución de los componentes sea compacta pero limpia y el ruteado de la placa se ha realizado a una sola cara, para facilitar la reproducción de la misma.

Debido a que sobraba el suficiente espacio en la placa, se han dispuesto distintas islas de cobre con la forma de los logotipos de la Universidad Politécnica de Cartagena, el escudo de la Escuela Técnica Superior de Ingeniería Industrial y el logotipo de Open Hardware, así como el rotulado del diseñador y del director del proyecto y de un supuesto logotipo comercial de la placa.

4.2.3.3 Creación del PCB en placa fotosensible.

El primer paso realizado para la impresión en la placa fotosensible ha sido la impresión en fotolito del layout de la placa.

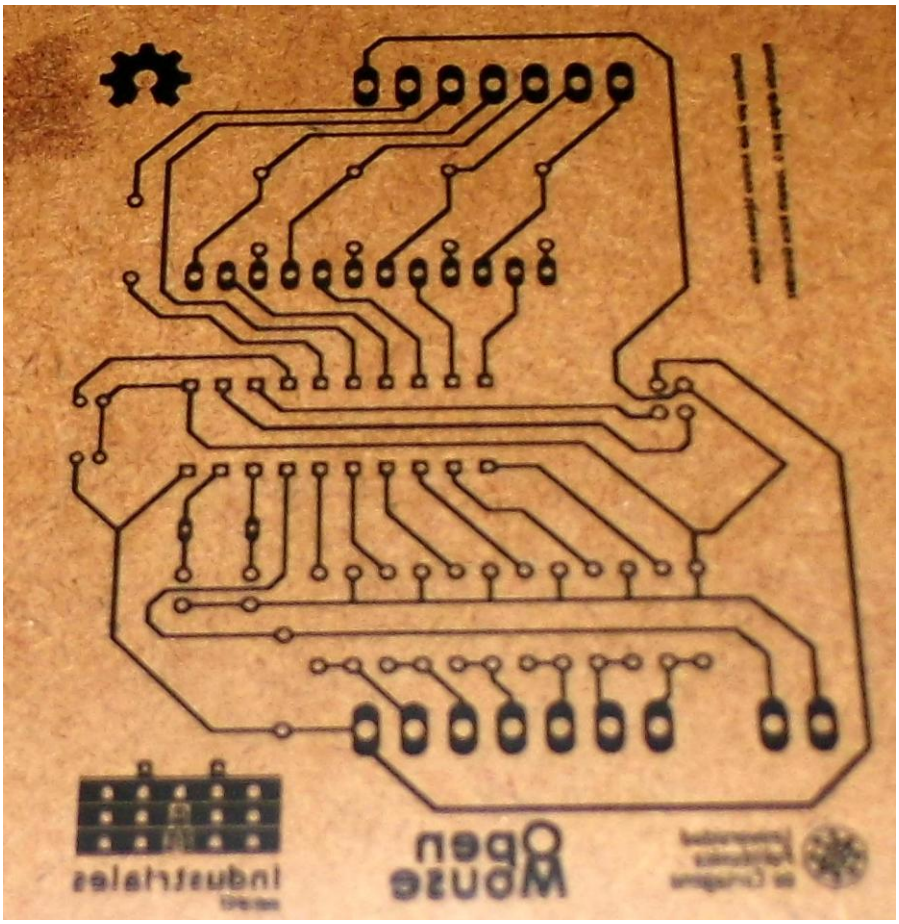


Figura 4.11 Fotolito

Tras la impresión del fotolito, se ha preparado la insoladora de rayos ultravioleta con el mismo y se ha colocado el PCB fotosensible para su exposición. Al ser una insoladora casera, el tiempo de insolación es bastante prolongado comparándolo con insoladoras profesionales. El tiempo de insolación para esta placa de circuito impreso ha sido de 8 minutos y medio.

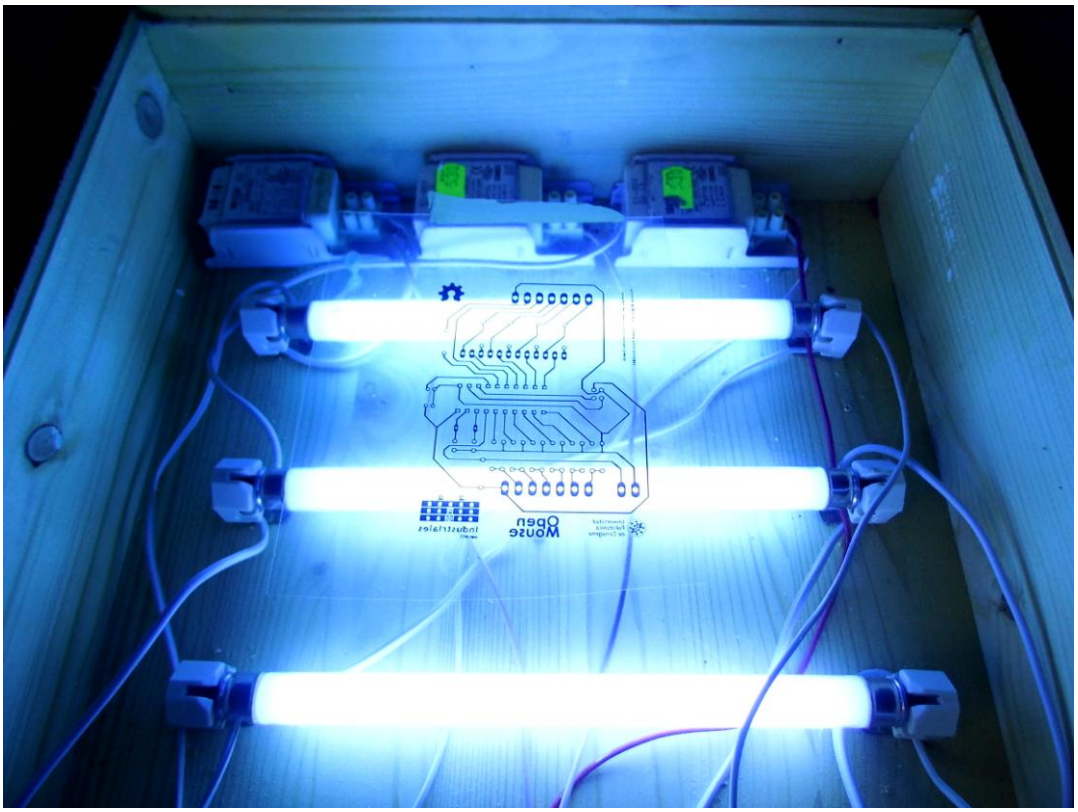


Figura 4.12 Fotolito preparado en la insoladora

Tras este paso se ha revelado el PCB con una solución de agua y sosa cáustica.

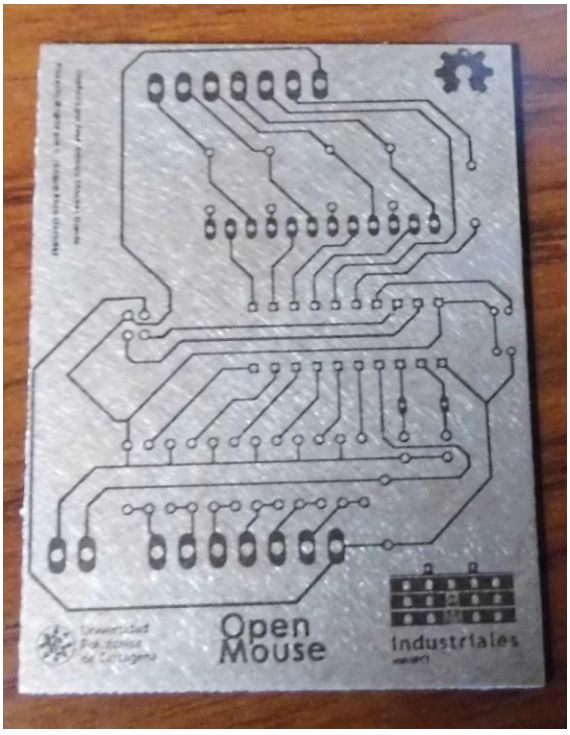


Figura 4.13 PCB tras el proceso de revelado

Tras el revelado, se ha procedido a el PCB con una solución de ácido sulfúrico, peróxido de hidrógeno y agua

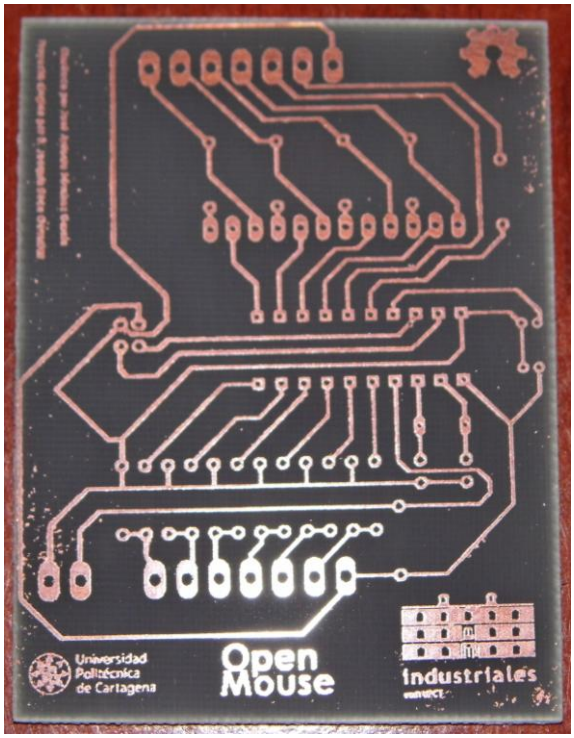


Figura 4.14 PCB tras el proceso de atacado

Tras el atacado, se ha eliminado de la placa de circuito impreso los restos de material fotosensible con alcohol y se ha procedido al perforado del PCB

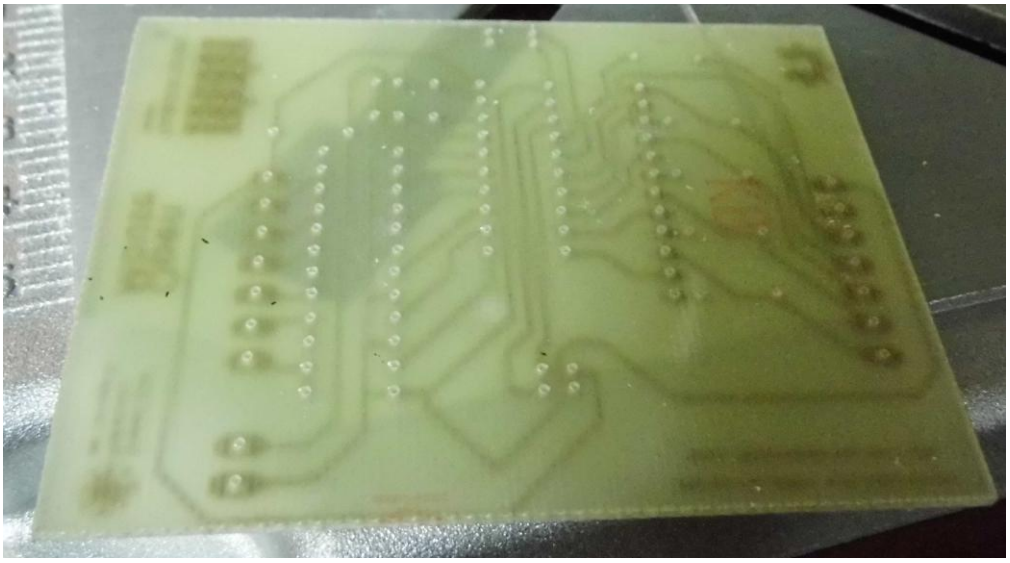


Figura 4.15 PCB tras el proceso de taladrado

Tras este paso, se ha procedido al soldado de los componentes y a la comprobación de su funcionamiento. En el paso de soldadura de la placa, se rompió uno de los pads de una resistencia, teniendo que añadir más estaño y dejar la patilla de la resistencia un poco más larga para solventar el problema.

Tras las pruebas de funcionamiento, se comprobó que no funcionaba correctamente debido a que, tras terminar la soldadura de la resistencia R12 (concretamente, la patilla conectada a masa) la pista se rompió, y no forzaba la patilla RC5 a masa, por lo que el dispositivo, al conectarlo, desplazaba aleatoriamente el cursor a la derecha. Tras desmontar la placa y reparar la soldadura, se volvieron a repetir las pruebas, funcionando el circuito de manera satisfactoria.

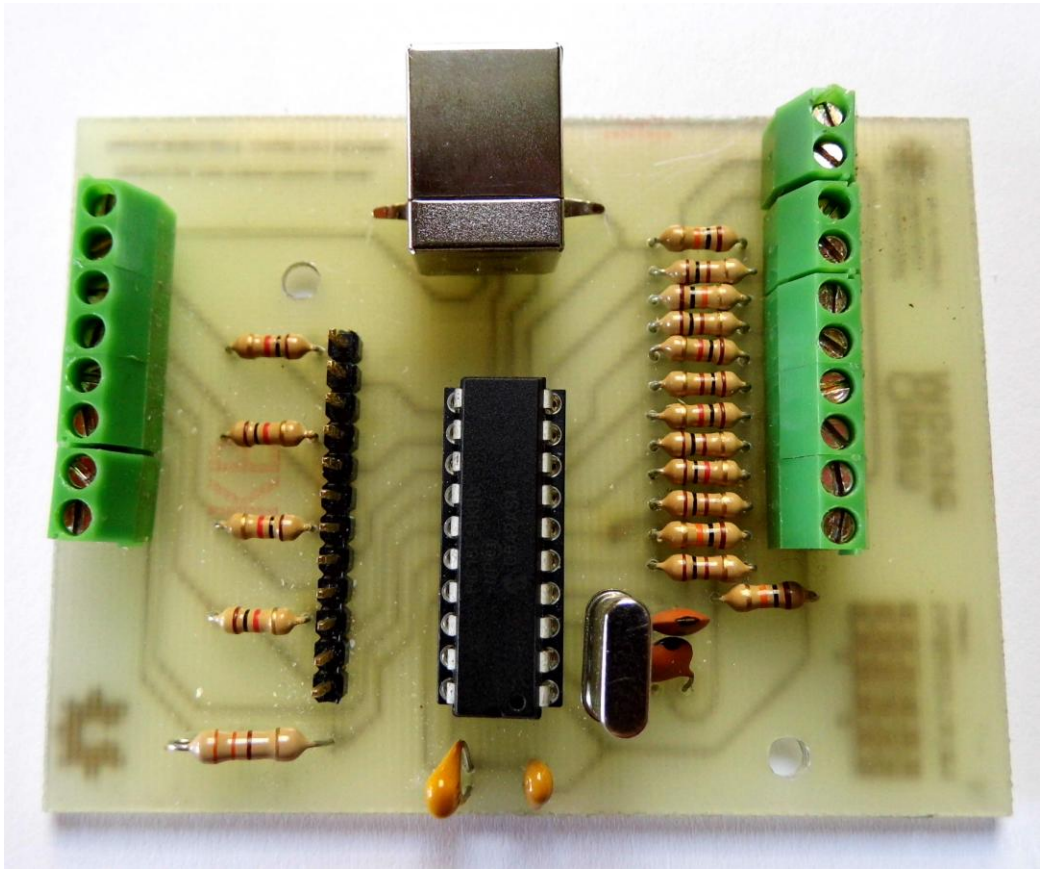


Figura 4.16 PCB Terminado cara componentes

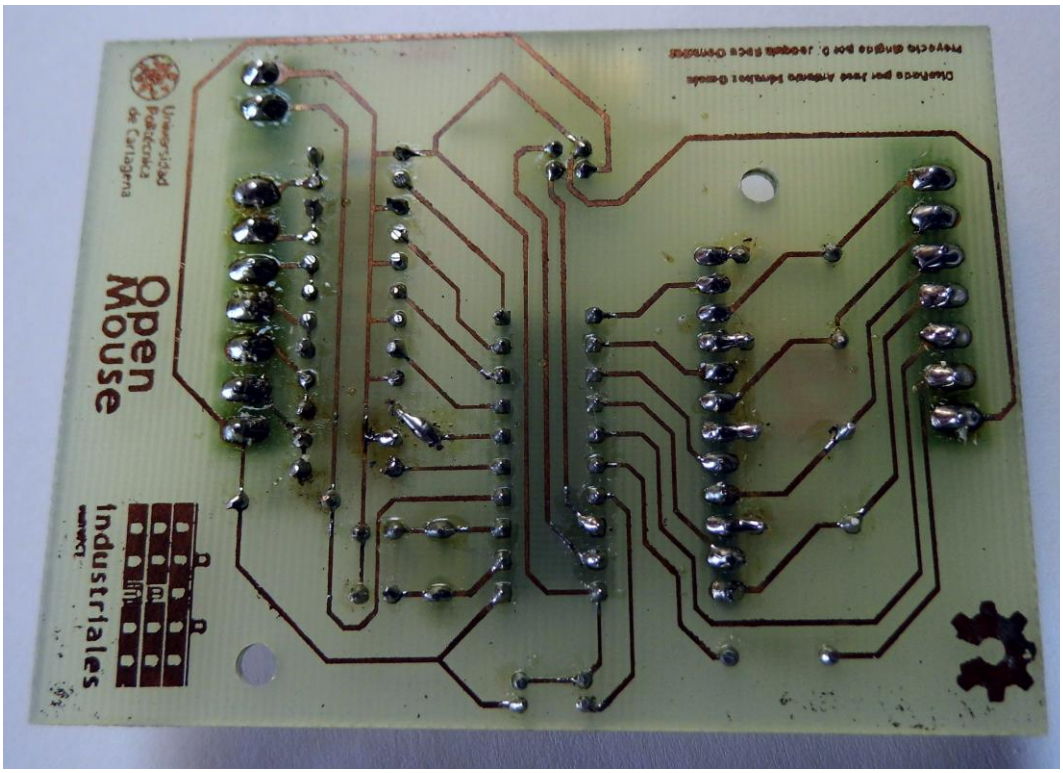


Figura 4.17 PCB terminado cara pistas

4.2.3.4 Elaboración del PCB en placa fresada.

Para realizar la placa fresada, se han generado con EAGLE tres archivos Gerber. Uno de ellos contiene el contorno de la placa, otro de ellos contiene el layout de las pistas y otro contiene el layout de las perforaciones.

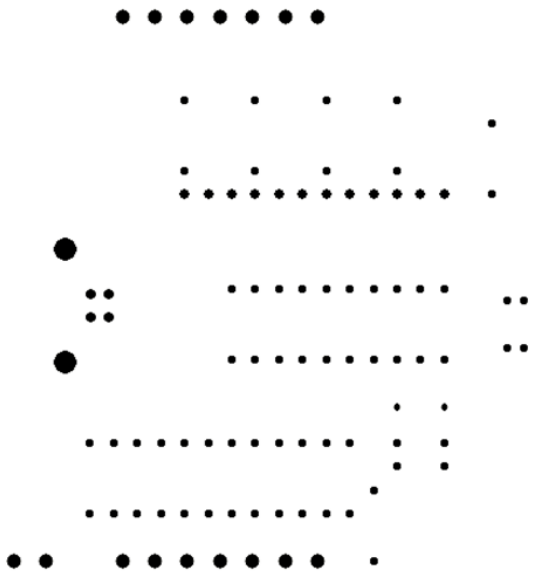


Figura 4.18 Layout Gerber de las perforaciones

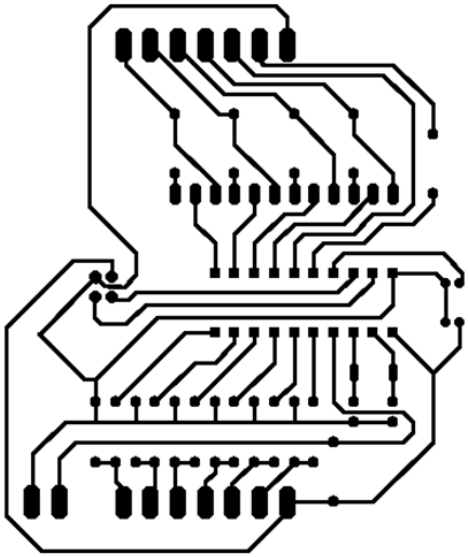


Figura 4.19 Layout Geber de las pistas

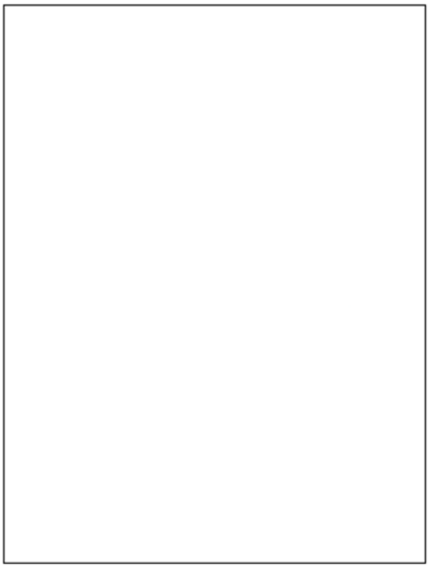


Figura 4.20 Layout Gerber del borde de placa

Tras la generación de los gerber, se ha procedido a introducirlos por capas en el programa CircuitCam, software servido por LPKF para la preparación de los proyectos antes de enviarlos al software controlador de la máquina de prototipado rápido.

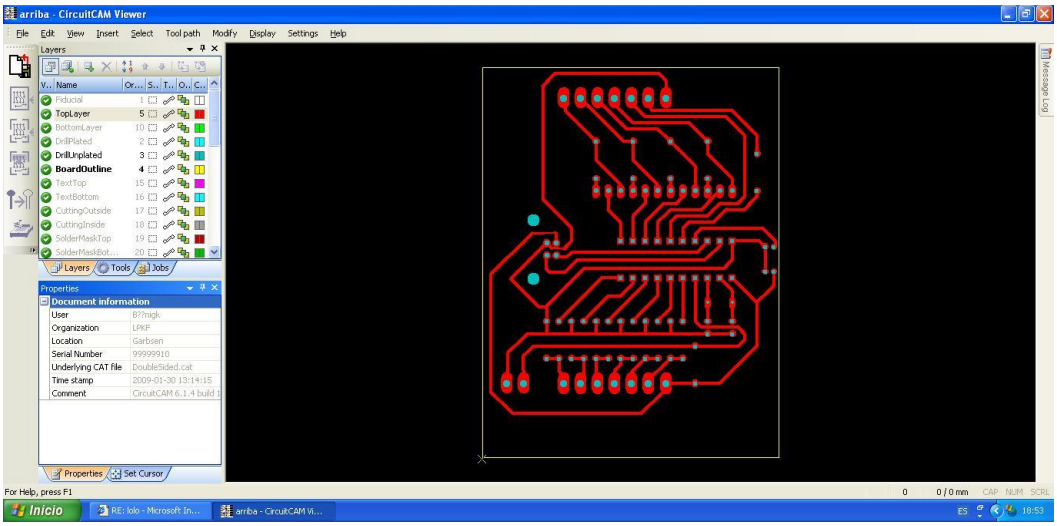


Figura 4.21 Carga de los archivos gerber en CircuitCam

Tras preparar los archivos por capas, se ha mandado el proyecto al programa controlador de la fresadora y , tras calibrarla y preparar el portaherramientas con las herramientas adecuadas, se ha iniciado el fresado de la placa.

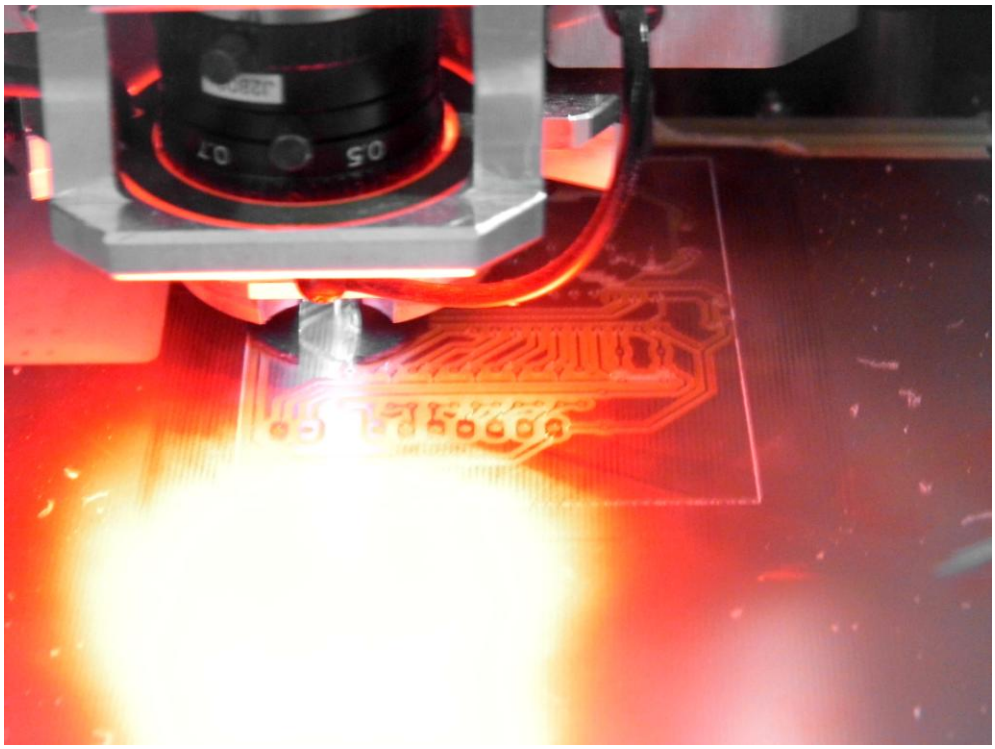


Figura 4.22 Fresado de placa

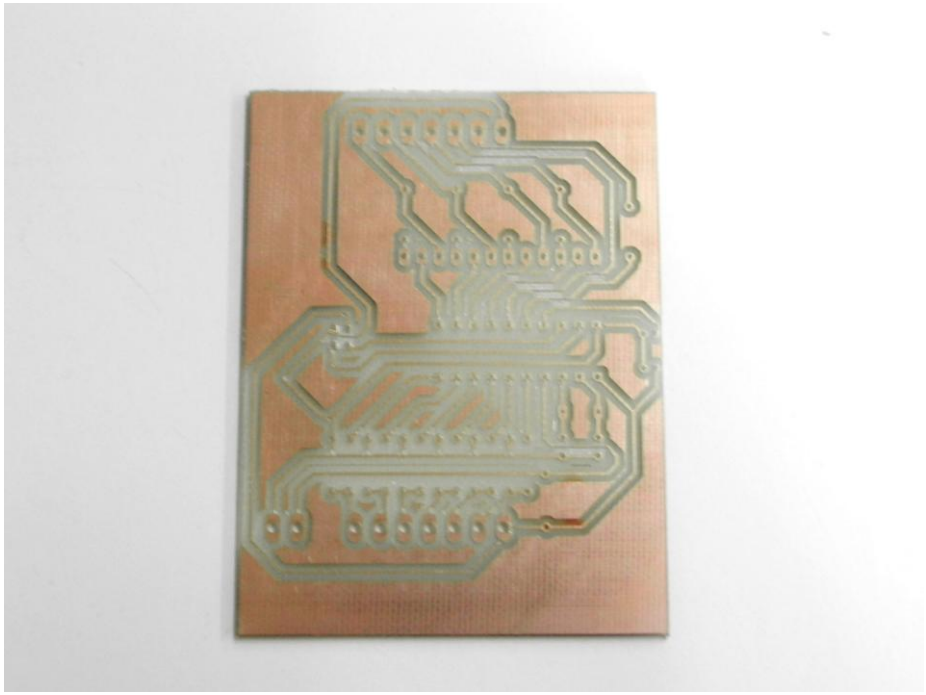


Figura 4.23 Placa fresada terminada

4.3. Banco de pruebas.

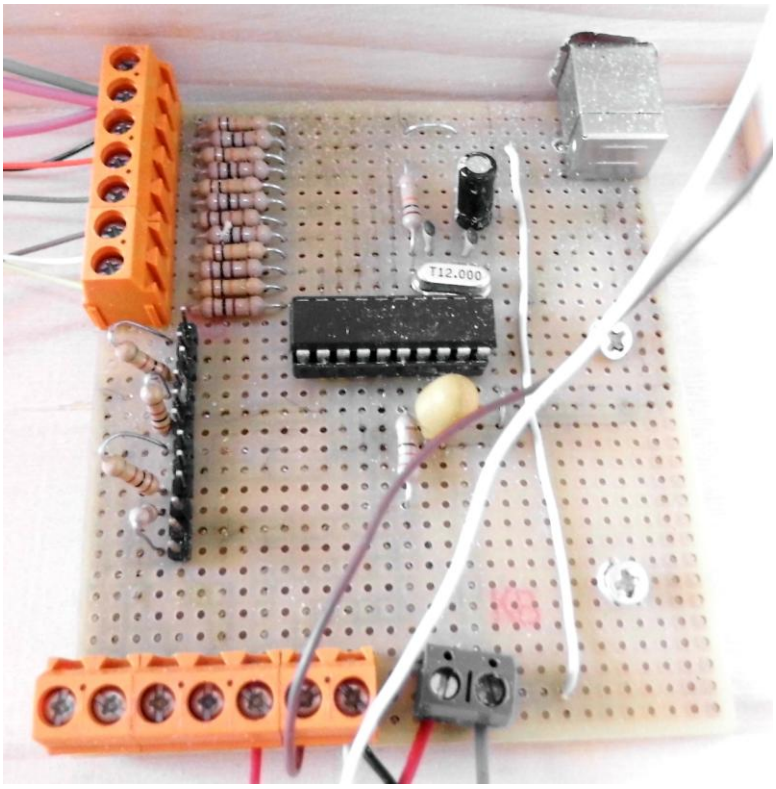


Figura 4.24 Placa prototipo montada en el banco de pruebas

Para la elaboración del banco de pruebas, se ha optado por realizar el montaje de los actuadores, de la placa controladora y de los elementos de realimentación acústicos y visuales, en una caja de madera que permitiese el rápido acceso al circuito para realizar modificaciones y que fuese lo suficientemente grande como para que los elementos del proyecto no se tocaran entre sí.

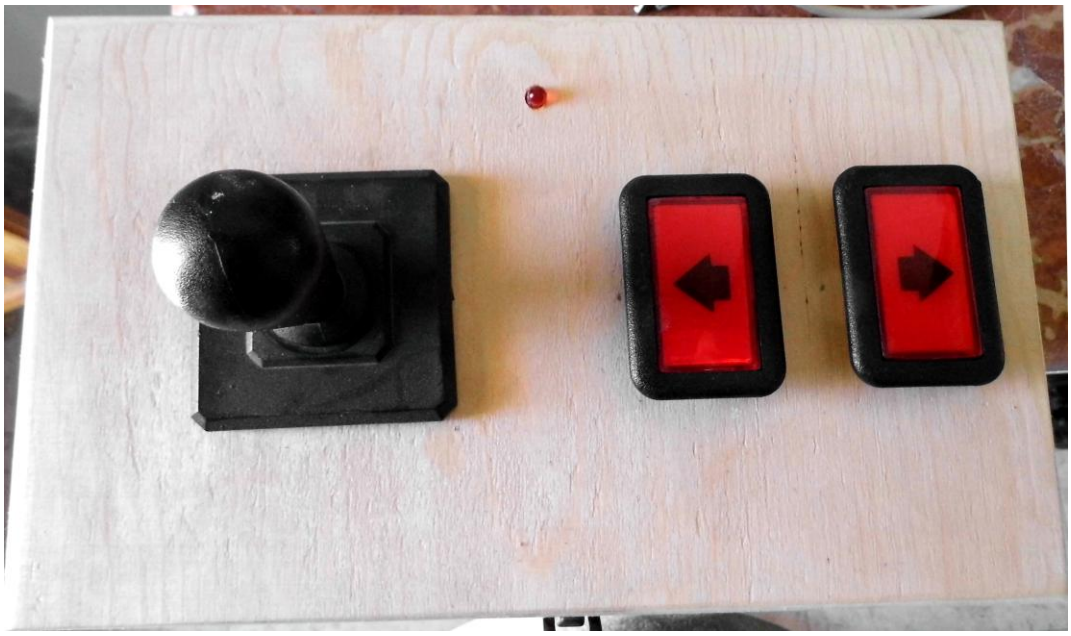


Figura 4.25 Actuadores montados en el banco de pruebas

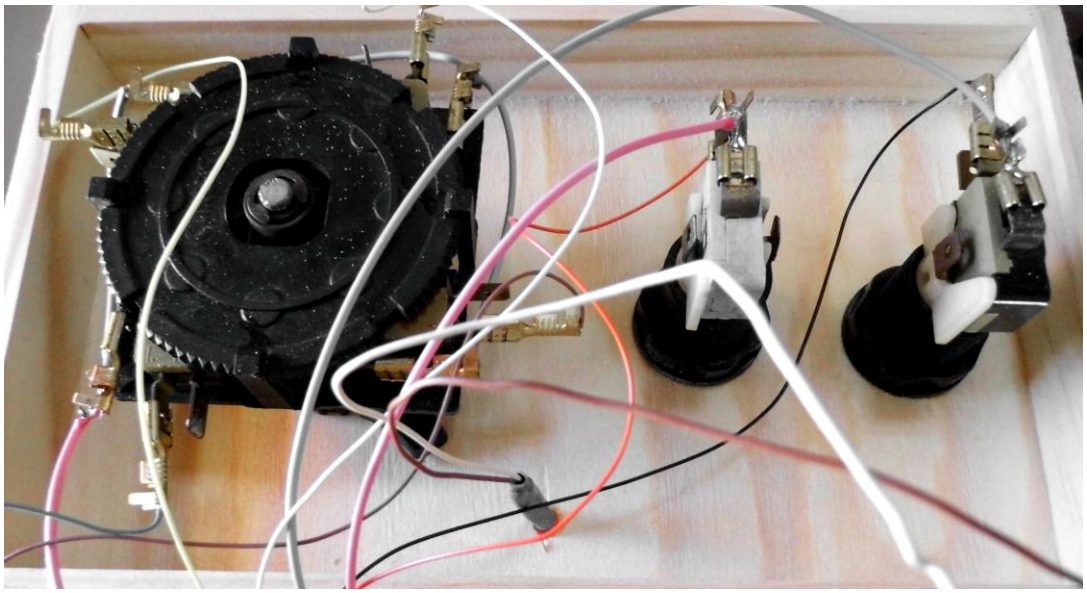


Figura 4.26 Cableado de los actuadores

5. Conclusiones y futuros proyectos

Tras la realización del proyecto se han llegado a diferentes conclusiones expuestas a continuación:

El entorno de trabajo MPLAB aunque gratuito, es un entorno demasiado profesional para desarrollos sencillos de software y necesita la instalación adicional del compilador de pago C18 para permitir la programación en C (aunque la versión de evaluación se puede seguir utilizando tras los 60 días de prueba sin optimización del código de salida, lo cual aumenta el tamaño del archivo de programa que hay que cargar en el microcontrolador).

La utilización del Framework USB y el Stack USB de Microchip no permiten una fácil comprensión del interfaz USB del dispositivo, debido a que implementa todo el protocolo tal y como lo viene reflejado en las especificaciones técnicas del USB. Esto conlleva que el empleo del USB deje de ser transparente como en otros entornos de desarrollo (CCS, MikroC...).

Estas dos razones parecen llevar a la conclusión de que sería mas adecuada la utilización de otro entorno de trabajo que permitese utilizar el interfaz USB de manera más transparente.

Sin embargo esto no es así, debido a que los entornos de trabajo alternativos a MPLAB no son gratuitos, dificultando la difusión del proyecto y, aunque la curva de aprendizaje de la utilización de estos entornos es mucho más inmediata que la de MPLAB, no permiten utilizar toda la potencia del protocolo USB de los microcontroladores de Microchip debido a que no implementan todo el protocolo de comunicaciones para todos los microcontroladores.

Para diseños de una complejidad media/alta, no se deberían utilizar estos entornos de desarrollo, siendo más adecuada la utilización del entorno MPLAB. Debido a que se ha elegido trabajar con el Framework USB de Microchip, será posible reutilizar el código generado para compilar el proyecto para la gran mayoría de microcontroladores de Microchip con soporte USB.

En cuanto a la elección de la creación de un entorno físico específico para el desarrollo final del proyecto se ha llegado a la conclusión que el hecho de crear una plataforma propia puede despertar la inquietud de los aficionados a la electrónica y de los usuarios cuya única motivación sea el uso del producto final, bien para uso propio o para uso ajeno (familiares u asociaciones de discapacitados).

Se ha creado otra alternativa más a las existentes actualmente en el mercado (consistentes en hardware privativo o desarrollos con otras plataformas hardware abiertas, como Arduino), que proporciona un bajo coste de fabricación para el caso de la elaboración propia en cantidades medianas de producto, si contamos con un coste de mano de obra nulo. Por lo tanto, aunque el gasto producido por el desarrollo del

proyecto a pequeña escala es igualable al de la adquisición de algunas plataformas comerciales, el proyecto ofrece una solución más económica para aquellas asociaciones de discapacitados que quieran llevarlo a cabo para sus asociados a gran escala.

Como futuros proyectos a desarrollar, se proponen los expuestos a continuación:

A pesar de que se ha pretendido que el proyecto fuese fácil de reproducir, es más que probable que esto no pueda resultar viable para cierta parte de los usuarios interesados en ello. Como solución a este problema se propone la venta a precio de coste mas tasas de las placas acabadas, con el microcontrolador programado con el bootloader y con su respectivo encapsulado en caja, que permitiera al usuario la conexión rápida de los actuadores a la placa mediante conectores tipo jack o equivalentes, para una rápida puesta en marcha del dispositivo.

Para ello, sería conveniente abaratar costes de producción proyectando el layout para componentes SMD , consiguiendo la reduciendo el tamaño del PCB y con ello, los costes.

Por último, para mejorar la visibilidad del producto, se propone la subida del mismo a Internet, con una explicación breve de los pasos a seguir para elaborar el hardware y una guía de modificación del software del proyecto, en el caso de que alguien deseara modificar el firmware del dispositivo para ajustarlo a sus necesidades, dando la posibilidad de compartir el nuevo firmware generado para el beneficio común de los usuarios.

6. Bibliografía

- AXELSON,JAN. *USB Complete. The Developer`s Guide.*4ª ed. Lakeview Research LLC, 2009. ISBN13 978-1-931448-08-6.

- MICROCHIP. *PIC18F/LF1XK50 Data Sheet. 20-Pin USB Flash Microcontrollers with nanoWatt XLP Technology.* Microchip Technology Incorporated, 2010. ISBN: 978-1-60932-624-1

- MICROCHIP. *Low Pin Count USB Development Kit User's Guide.* Microchip Technology Incorporated, 2009.

- MICROCHIP. *Getting Started with Microchip's Low Pin Count USB solutions.* Microchip Technology Incorporated, 2009.

- UNIVERSIDAD DE VALENCIA. *MICROCHIP MPLAB C18 C COMPILER GUÍA DEL ESTUDIANTE*[En línea].Departamento de Informática, 2005 [citado Julio 2, 2013]. Disponible en World Wide Web http://informatica.uv.es/iiguia/SBM/manual.pdf .

- WIKIPEDIA. *Software Libre*[En línea],2013[citado Julio 2,2013] .Disponible en World Wide Web http://es.wikipedia.org/wiki/Software_libre .

- WIKIPEDIA. *Hardware Libre*[En línea],2013[citado Julio 2,2013] .Disponible en World Wide Web http://es.wikipedia.org/wiki/Hardware_libre .

- PANIAGUA, SORAYA. *Arduino, la revolución silenciosa del hardware libre.*[En línea],2011 [citado Julio 2,2013]. Disponible en World Wide Web http://www.sorayapaniagua.com/2011/03/14/arduino-la-revolucion-silenciosa-del-hardware-libre/ .

- VELASCO, JUAN J. *12 usos creativos para tu Raspberry Pi*[En línea] ALT1040, 2013 [citado Julio 2, 2013]. Disponible en World Wide Web http://alt1040.com/2013/04/usos-de-raspberry-pi .

7. Enlaces.

- Eagle
  http://www.cadsoftusa.com/download-eagle/freeware/?language=en

- Microchip MPLAB
  http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002

- Microchip C18
  http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en010014

- Microchip USB Framework
  http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en537044

- Circuito Abierto - Open Mouse
  http://circuitoabierto.es/?p=173

8. Anexos

8.1 Presupuesto

Costes de materiales

Cant.	Descripción	Proveedor	Precio unitario	Precio total
2	CONDENSADOR 22 PF	FARNELL	0.26€	0.52€
1	CONDENSADOR 0,1UF	FARNELL	0.22€	0.22€
1	CONDENSADOR 0,47UF	FARNELL	0.53€	0.53€
1	TIRA DE PINES 12 VIAS	FARNELL	0.78€	0.78€
1	CRISTAL 12.0 MHZ	FARNELL	0.42€	0.42€
6	RESISTENCIA 100 OHM	FARNELL	0.026€	0.16€
7	RESISTENCIA 10K	FARNELL	0.085€	0.6€
4	RESISTENCIA 1K	FARNELL	0.1€	0.4€
1	RESISTENCIA, 330 OHM	FARNELL	0.7€	0.7€
1	PIC18F14K50	FARNELL	2.71€	2.71€
5	ZÓCALO DIL 20VÍAS	FARNELL	0.19€	0.95€
1	USB B	FARNELL	0.69€	0.69€
10	REGLETA PCB 2 VIAS	FARNELL	0.25€	2.5€
			TOTAL MATERIALES	11.18€

Costes ejecución mano de obra.

CANTIDAD	TRABAJO	PRECIO UNITARIO	TOTAL
60 min	PREPARACIÓN PCB PROTOTIPO INCLUYENDO: -MECANIZADO -RUTEADO	30€/HORA	30€
45 min	PREPARACIÓN PCB FOTOSENSIBLE 80X60MM INCLUYENDO: -Exposición -Revelado -Atacado -Taladrado -Limpieza	30€/HORA	22.5€
45 min	FRESADO PCB 80X60MM INCLUYENDO: -Taladrado -Fresado -Cortado -Limpieza	40€/HORA	30€
60 min	MECANIZADO CAJA INCLUYENDO: -Taladrado -Lijado -Acabado -Montaje -Cableado	20€/HORA	20€
45 min	SOLDADURA COMPONENTES Y CABLEADO	30€/HORA	22.5€

Opciones

PCB PROTOTIPO	
MATERIALES	11.18€
PLACA CIRCUITO PERFORADO	3.64€
MANO DE OBRA PCB PROTOTIPO	30€
MANO DE OBRA MECANIZADO CAJA	20€
MANO DE OBRA SOLDADURA Y CABLEADO	22.5€
PROGRAMACIÓN/PRUEBAS	30€
TOTAL 1 UNIDAD PROTOTIPO	117.32€
TOTAL 100 UNIDADES PROTOTIPO (40% DTO)	4692.8€ 46'93€/UNIDAD

PCB FOTOSENSIBLE	
MATERIALES	11.18€
PLACA CIRCUITO FOTOSENSIBLE	6.98€
MANO DE OBRA PCB FOTOSENSIBLE	22.5€
MANO DE OBRA MECANIZADO CAJA	20€
MANO DE OBRA SOLDADURA Y CABLEADO	22.5€
PROGRAMACIÓN/PRUEBAS	30€
TOTAL 1 UNIDAD FOTOSENSIBLE	113.16€
TOTAL 100 UNIDADES FOTOSENSIBLE (40% DTO)	4526.4€
	45'27€/UNIDAD

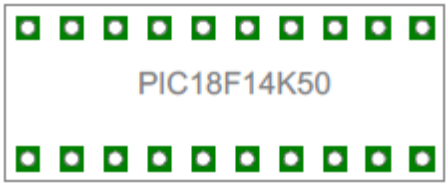
PCB FRESADO	
MATERIALES	11.18€
PLACA PCB COBRE	3.08€
MANO DE OBRA PCB FRESADO	30€
MANO DE OBRA MECANIZADO CAJA	20€
MANO DE OBRA SOLDADURA Y CABLEADO	22.5€
PROGRAMACIÓN/PRUEBAS	30€
TOTAL 1 UNIDAD PCB FRESADO	116.76€
TOTAL 100 UNIDADES PCB FRESADO (40% DTO)	4670.4€
	46'70€/UNIDAD

1 PCB POR ENCARGO	
MATERIALES	11.18€
PLACA PCB POR ENCARGO	69.81€
MANO DE OBRA MECANIZADO CAJA	20€
MANO DE OBRA SOLDADURA Y CABLEADO	22.5€
PROGRAMACIÓN/PRUEBAS	30€
TOTAL 1 UNIDAD PCB ENCARGO	152.77€

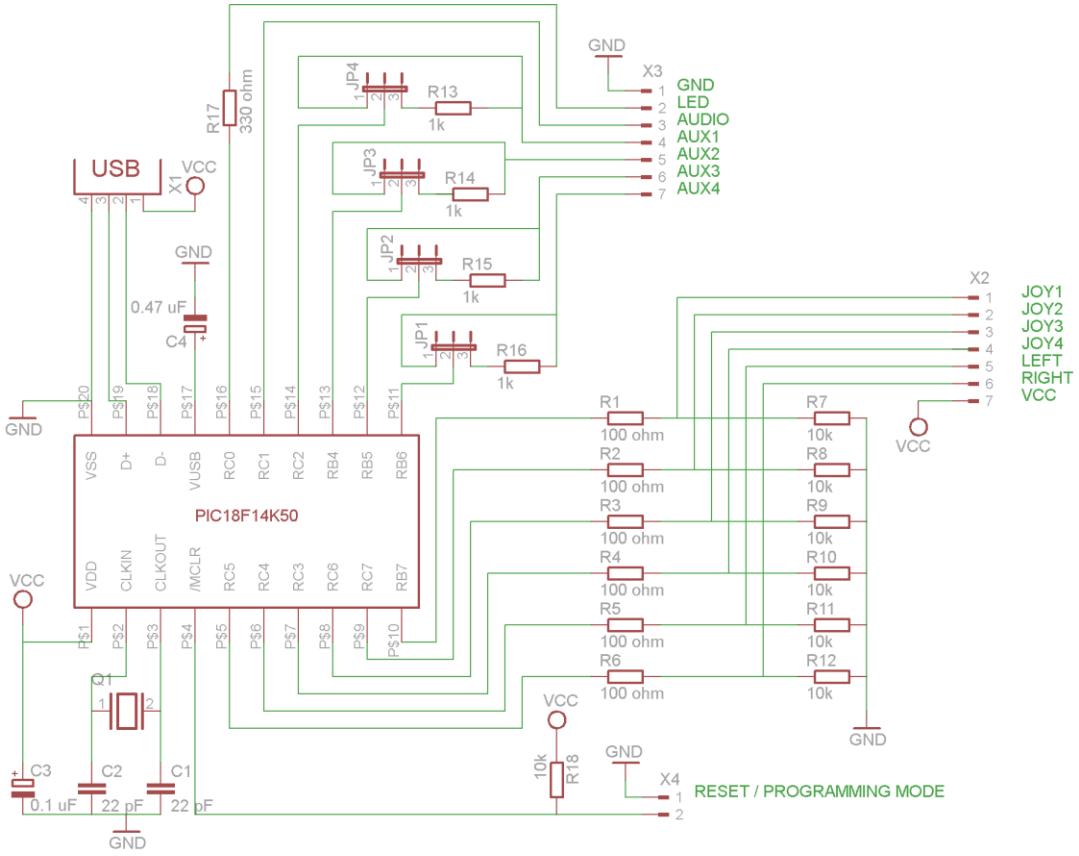
100 PCB POR ENCARGO	
MATERIALES	11.18€
PLACA PCB POR ENCARGO	3.42€
MANO DE OBRA MECANIZADO CAJA	20€
MANO DE OBRA SOLDADURA Y CABLEADO	22.5€
PROGRAMACIÓN/PRUEBAS	30€
TOTAL 1 UNIDAD PCB ENCARGO	87.1€
TOTAL 100 UNIDADES PCB ENCARGO(40% DTO)	3484€
	34'84€

8.2 Planos

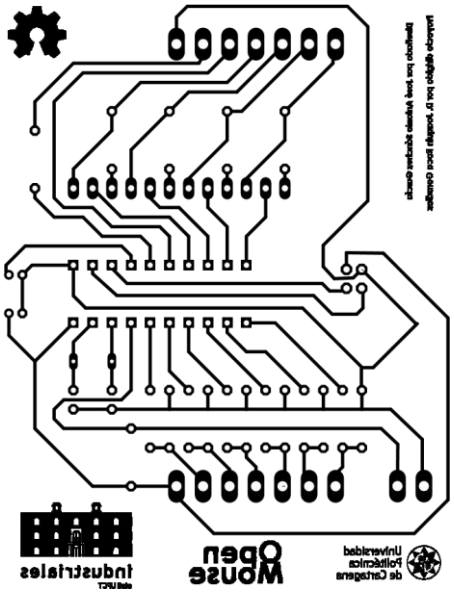
Autor:	José Antonio Sánchez García	Universidad Politécnica de Cartagena
Título:	Layout esquemático PIC18F14K50	
Fecha:	03/07/2013	
UNIVERSIDAD POLITÉCNICA DE CARTAGENA		



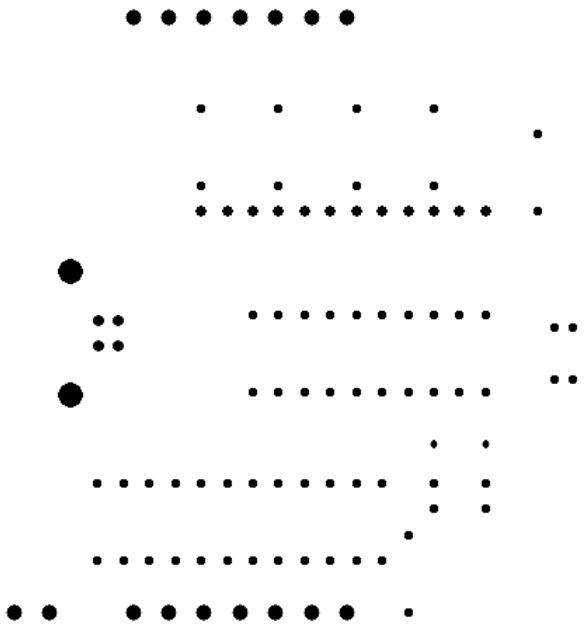
Autor:	José Antonio Sánchez García	Universidad Politécnica de Cartagena
Título:	Layout PCB PIC18F14K50	
Fecha:	03/07/2013	
UNIVERSIDAD POLITÉCNICA DE CARTAGENA		



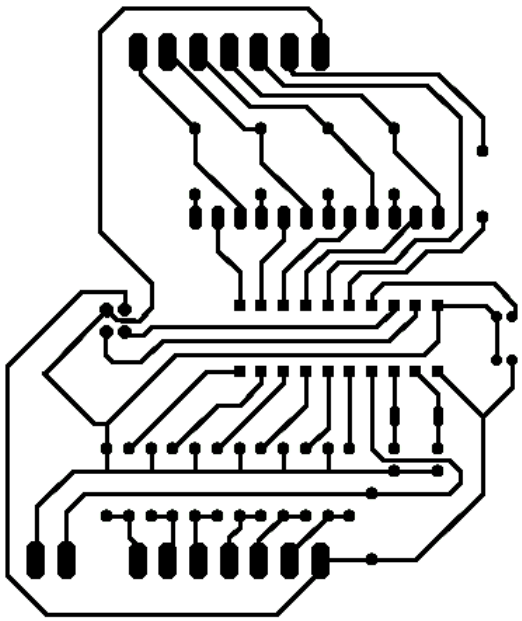
Autor:	José Antonio Sánchez García	Universidad Politécnica de Cartagena
Título:	Esquemático	
Fecha:	03/07/2013	
UNIVERSIDAD POLITÉCNICA DE CARTAGENA		



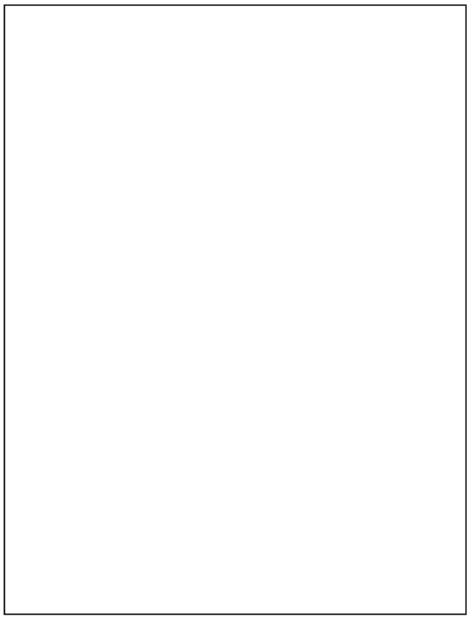
Autor:	José Antonio Sánchez García	Universidad Politécnica de Cartagena
Título:	Layout PCB	
Fecha:	03/07/2013	
UNIVERSIDAD POLITÉCNICA DE CARTAGENA		



Autor:	José Antonio Sánchez García	Universidad Politécnica de Cartagena
Título:	Impresión Gerber - Taladrado	
Fecha:	03/07/2013	
UNIVERSIDAD POLITÉCNICA DE CARTAGENA		



Autor:	José Antonio Sánchez García	Universidad Politécnica de Cartagena
Título:	Impresión Gerber - Fresado	
Fecha:	03/07/2013	
UNIVERSIDAD POLITÉCNICA DE CARTAGENA		



Autor:	José Antonio Sánchez García	Universidad Politécnica de Cartagena
Título:	Impresión Gerber - Contorno	
Fecha:	03/07/2013	
UNIVERSIDAD POLITÉCNICA DE CARTAGENA		

8.3 Código

usb_descriptor.c

```
#ifndef __USB_DESCRIPTORS_C
#define __USB_DESCRIPTORS_C

/** INCLUDES
*******************************************************/
#include "./USB/usb.h"
#include "./USB/usb_function_hid.h"

/** CONSTANTS
*******************************************************/
#if defined(__18CXX)
#pragma romdata
#endif

/* Device Descriptor */
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
    0x12,    // Size of this descriptor in bytes
    USB_DESCRIPTOR_DEVICE,                // DEVICE
descriptor type
    0x0200,                 // USB Spec Release Number in
BCD format
    0x00,                   // Class Code
    0x00,                   // Subclass code
    0x00,                   // Protocol code
    USB_EP0_BUFF_SIZE,          // Max packet size for EP0,
see usb_config.h
    MY_VID,                 // Vendor ID
    MY_PID,                 // Product ID: Mouse in a
circle fw demo
    0x0003,                 // Device release number in BCD
format
    0x01,                   // Manufacturer string index
    0x02,                   // Product string index
    0x00,                   // Device serial number string
index
    0x01                    // Number of possible
configurations
};

/* Configuration 1 Descriptor */
ROM BYTE configDescriptor1[]={
    /* Configuration Descriptor */
    0x09,//sizeof(USB_CFG_DSC),    // Size of this
descriptor in bytes
```

```
    USB_DESCRIPTOR_CONFIGURATION,               //
CONFIGURATION descriptor type
    DESC_CONFIG_WORD(0x0022),   // Total length of data for
this cfg
    1,                      // Number of interfaces in this
cfg
    1,                      // Index value of this
configuration
    0,                      // Configuration string index
    _DEFAULT | _SELF,               // Attributes, see
usb_device.h
    50,                     // Max power consumption (2X
mA)

    /* Interface Descriptor */
    0x09,//sizeof(USB_INTF_DSC),   // Size of this
descriptor in bytes
    USB_DESCRIPTOR_INTERFACE,               // INTERFACE
descriptor type
    0,                      // Interface Number
    0,                      // Alternate Setting Number
    1,                      // Number of endpoints in this
intf
    HID_INTF,               // Class code
    BOOT_INTF_SUBCLASS,     // Subclass code
    HID_PROTOCOL_MOUSE,     // Protocol code
    0,                      // Interface string index

    /* HID Class-Specific Descriptor */
    0x09,//sizeof(USB_HID_DSC)+3,    // Size of this
descriptor in bytes RRoj hack
    DSC_HID,                // HID descriptor type
    DESC_CONFIG_WORD(0x0111),                 // HID Spec
Release Number in BCD format (1.11)
    0x00,                   // Country Code (0x00 for Not
supported)
    HID_NUM_OF_DSC,         // Number of class descriptors,
see usbcfg.h
    DSC_RPT,                // Report descriptor type
    DESC_CONFIG_WORD(50),   //sizeof(hid_rpt01),      //
Size of the report descriptor

    /* Endpoint Descriptor */
    0x07,/*sizeof(USB_EP_DSC)*/
    USB_DESCRIPTOR_ENDPOINT,    //Endpoint Descriptor
    HID_EP | _EP_IN,            //EndpointAddress
    _INTERRUPT,                       //Attributes
    DESC_CONFIG_WORD(3),                  //size
    0x01                        //Interval
};
```

```
//Language code string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD
string[1];}sd000={
sizeof(sd000),USB_DESCRIPTOR_STRING,{0x0409
}};

//Manufacturer string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD
string[37];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{'U','n','i','v','e','r','s','i','d','a ','d','
','P','o','l','i','t','e','c','n','i','c','a ','
','d','e',' ','C','a','r','t','a','g','e','n','a','.',
}};

//Product string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD
string[26];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{'R','a','t','o','n',
' ','p','a','r','a','
','D','i','s','c','a','p','a','c','i','t','a','d','o','s','
.'
}};

//Class specific descriptor - HID mouse
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
    {0x05, 0x01, /* Usage Page (Generic Desktop)
*/
    0x09, 0x02, /* Usage (Mouse)
*/
    0xA1, 0x01, /* Collection (Application)
*/
    0x09, 0x01, /*  Usage (Pointer)
*/
    0xA1, 0x00, /*  Collection (Physical)
*/
    0x05, 0x09, /*      Usage Page (Buttons)
*/
    0x19, 0x01, /*      Usage Minimum (01)
*/
    0x29, 0x03, /*      Usage Maximum (03)
*/
    0x15, 0x00, /*      Logical Minimum (0)
*/
    0x25, 0x01, /*      Logical Maximum (0)
*/
    0x95, 0x03, /*      Report Count (3)
*/
```

```
    0x75, 0x01, /*      Report Size (1)
*/
    0x81, 0x02, /*      Input (Data, Variable, Absolute)
*/
    0x95, 0x01, /*      Report Count (1)
*/
    0x75, 0x05, /*      Report Size (5)
*/
    0x81, 0x01, /*      Input (Constant)    ;5 bit padding
*/
    0x05, 0x01, /*      Usage Page (Generic Desktop)
*/
    0x09, 0x30, /*      Usage (X)
*/
    0x09, 0x31, /*      Usage (Y)
*/
    0x15, 0x81, /*      Logical Minimum (-127)
*/
    0x25, 0x7F, /*      Logical Maximum (127)
*/
    0x75, 0x08, /*      Report Size (8)
*/
    0x95, 0x02, /*      Report Count (2)
*/
    0x81, 0x06, /*      Input (Data, Variable, Relative)
*/
    0xC0, 0xC0}
};/* End Collection,End Collection            */

//Array of configuration descriptors
ROM BYTE *ROM USB_CD_Ptr[]=
{
    (ROM BYTE *ROM)&configDescriptor1
};

//Array of string descriptors
ROM BYTE *ROM USB_SD_Ptr[]=
{
    (ROM BYTE *ROM)&sd000,
    (ROM BYTE *ROM)&sd001,
    (ROM BYTE *ROM)&sd002
};

/** EOF usb_descriptors.c
*************************************************/

#endif
```

HardwareProfile.h

```
#ifndef HARDWARE_PROFILE_H
#define HARDWARE_PROFILE_H

#define self_power          1
#define USB_BUS_SENSE       1
#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER
#define CLOCK_FREQ 48000000

//INICIALIZACIÓN DE LOS PUERTOS
#define Inicializar_Hardware() TRISC &= 0xFC; TRISB &=
0X80; ANSEL &= 0X00; ANSELH &= 0X00;

//ETIQUETAS DE PUERTOS
#define joystickArriba        PORTBbits.RB7//Asignar
patilla rb7 para joystick arriba
#define joystickAbajo         PORTCbits.RC6 //Asignar
patilla rc6 para joystick abajo
#define joystickDerecha       PORTCbits.RC3 //Asignar
patilla rc3 para joystick derecha
#define joystickIzquierda     PORTCbits.RC7 //Asignar
patilla rc7 para joystick izquierda
#define botonDerecho          PORTCbits.RC4 //Asignar
patilla rc4 para boton derecho
#define botonIzquierdo        PORTCbits.RC5 //Asignar
patilla rc5 para boton izquierdo
#define salidaLed             PORTCbits.RC0 //Asignar
patilla rc0 para salida de led
#define salidaPiezo           PORTCbits.RC1 //Asignar
patilla rc1 para salida de piezoelectrico
```

mouse.c

```
#ifndef USBMOUSE_C
#define USBMOUSE_C

/** INCLUDES
*************************************************/
#include "./USB/usb.h"
#include "HardwareProfile.h"
#include "./USB/usb_function_hid.h"
#include "delays.h"

/** CONFIGURATION
*************************************************/

        #pragma config CPUDIV = NOCLKDIV
        #pragma config USBDIV = OFF
        #pragma config FOSC   = HS
```

```
        #pragma config PLLEN  = ON
        #pragma config FCMEN  = OFF
        #pragma config IESO   = OFF
        #pragma config PWRTEN = OFF
        #pragma config BOREN  = OFF
        #pragma config BORV   = 30
        #pragma config WDTEN  = OFF
        #pragma config WDTPS  = 32768
        #pragma config MCLRE  = OFF
        #pragma config HFOFST = OFF
        #pragma config STVREN = ON
        #pragma config LVP    = OFF
        #pragma config XINST  = OFF
        #pragma config BBSIZ  = OFF
        #pragma config CP0    = OFF
        #pragma config CP1    = OFF
        #pragma config CPB    = OFF
        #pragma config WRT0   = OFF
        #pragma config WRT1   = OFF
        #pragma config WRTB   = OFF
        #pragma config WRTC   = OFF
        #pragma config EBTR0  = OFF
        #pragma config EBTR1  = OFF
        #pragma config EBTRB  = OFF

/** VARIABLES
*****************************************************/
#pragma udata

char buffer[3];
char old_buffer[3];
char send_buffer[3];
USB_HANDLE lastTransmission;
void sonidoPiezo();
void Emulate_Mouse(void);
static void InitializeSystem(void);
void ProcessIO(void);
void UserInit(void);
void YourHighPriorityISRCode();
void YourLowPriorityISRCode();



/** VECTOR REMAPPING
*********************************************/

//#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER
//#define
PROGRAMMABLE_WITH_USB_LEGACY_CUSTOM_CLASS_BOOTLOADER

#if defined(__18CXX)
```

```
    #if defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)
        #define REMAPPED_RESET_VECTOR_ADDRESS
    0x1000
        #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
    0x1008
        #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
    0x1018
    #elif
defined(PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)
        #define REMAPPED_RESET_VECTOR_ADDRESS
    0x800
        #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
    0x808
        #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
    0x818
    #else
        #define REMAPPED_RESET_VECTOR_ADDRESS
    0x00
        #define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
    0x08
        #define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
    0x18
    #endif

    #if
defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(PROG
RAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)
    extern void _startup (void);        // See c018i.c in
your C18 compiler dir
    #pragma code REMAPPED_RESET_VECTOR =
REMAPPED_RESET_VECTOR_ADDRESS
    void _reset (void)
    {
        _asm goto _startup _endasm
    }
    #endif
    #pragma code REMAPPED_HIGH_INTERRUPT_VECTOR =
REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
    void Remapped_High_ISR (void)
    {
        _asm goto YourHighPriorityISRCode _endasm
    }
    #pragma code REMAPPED_LOW_INTERRUPT_VECTOR =
REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
    void Remapped_Low_ISR (void)
    {
        _asm goto YourLowPriorityISRCode _endasm
    }
```

```
    #if
defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(PROG
RAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER)

    #pragma code HIGH_INTERRUPT_VECTOR = 0x08
    void High_ISR (void)
    {
        _asm goto REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
_endasm
    }
    #pragma code LOW_INTERRUPT_VECTOR = 0x18
    void Low_ISR (void)
    {
        _asm goto REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
_endasm
    }
    #endif    //end of "#if
defined(PROGRAMMABLE_WITH_USB_HID_BOOTLOADER)||defined(PROG
RAMMABLE_WITH_USB_LEGACY_CUSTOM_CLASS_BOOTLOADER)"

    #pragma code

    //These are your actual interrupt handling routines.
    #pragma interrupt YourHighPriorityISRCode
    void YourHighPriorityISRCode()
    {

       #if defined(USB_INTERRUPT)
           USBDeviceTasks();
       #endif

    }    //This return will be a "retfie fast", since this
is in a #pragma interrupt section
    #pragma interruptlow YourLowPriorityISRCode
    void YourLowPriorityISRCode()
    {


    }    //This return will be a "retfie", since this is
in a #pragma interruptlow section
#endif



/** DECLARATIONS
*****************************************************/
#pragma code

void main(void)
{
```

```
    InitializeSystem();

    #if defined(USB_INTERRUPT)
        USBDeviceAttach();
    #endif

    while(1)
    {
        #if defined(USB_POLLING)

        USBDeviceTasks();

        #endif
        ProcessIO();
    }//end while
}//end main


static void InitializeSystem(void)
{

        ADCON1 |= 0x0F;                 // Default all pins
to digital

    #if defined(USE_USB_BUS_SENSE_IO)
    tris_usb_bus_sense = INPUT_PIN; // See
HardwareProfile.h
    #endif

    #if defined(USE_SELF_POWER_SENSE_IO)
    tris_self_power = INPUT_PIN;   // See HardwareProfile.h
    #endif

    UserInit();

    USBDeviceInit();     //usb_device.c.  Initializes USB
module SFRs and firmware
                         //variables to known states.
}//end InitializeSystem


void UserInit(void)
{
     //Inicializa los puertos. Ver HardwareProfile.h.
    Inicializar_Hardware();
    //Initialize all of the mouse data to 0,0,0 (no
movement)
    buffer[0]=buffer[1]=buffer[2]=0;
    old_buffer[0]=old_buffer[1]=old_buffer[2]=0;
    send_buffer[0]=send_buffer[1]=send_buffer[2]=0;
```

```
    lastTransmission = 0;
}//end UserInit


void ProcessIO(void)
{

    // User Application USB tasks
    if((USBDeviceState <
CONFIGURED_STATE)||(USBSuspendControl==1)) return;

    //Call the function that emulates the mouse
    Emulate_Mouse();

}//end ProcessIO

void sonidoPiezo()//SUBPROGRAMA QUE ENVÍA AL PIEZOELECTRICO
UNA SEÑAL CUADRADA

{
                salidaPiezo=1;//ENVÍA UN NIVEL ALTO AL
PIEZOELECTRICO
                Delay1KTCYx(10);//ESPERA 0.86 MILISEFUNDOS
                salidaPiezo=0;//ENVÍA UN NIVEL BAJO AL
PIEZOELECTRICO
                Delay1KTCYx(10);//ESPERA 0.86 MILISEGUNDOS
}


void Emulate_Mouse(void)
{
      if(joystickArriba == 1)
     //ESTRUCTURA IF-ELSE QUE DISCRIMINA QUE ENTRADAS ESTÁN
ACTIVAS
          {
          buffer [2] = -1;
          }
     else if(joystickAbajo ==1)
          {
          buffer [2] = 1;
          }
     else
          {
          buffer [2] = 0;
          }
     if(joystickDerecha == 1)
          {
          buffer [1] = 1;
          }
     else if(joystickIzquierda ==1)
```

```
            {
            buffer [1] = -1;
            }
      else
            {
            buffer [1] = 0;
            }

      if(botonDerecho == 1)
            {
         buffer [0] = 1;
            }
      else if(botonIzquierdo ==1)
            {
            buffer [0] = 2;
            }
      else
            {
            buffer [0] = 0;
            }
//RUTINA ANTI REBOTES
      if(buffer[0]==old_buffer[0])
            {
            send_buffer[0]=buffer[0];
            }
      else
            {
            old_buffer[0]=buffer[0];
            }
      if(buffer[1]==old_buffer[1])
            {
            send_buffer[1]=buffer[1];
            }
      else
            {
            old_buffer[1]=buffer[1];
            }

      if(buffer[2]==old_buffer[2])
            {
            send_buffer[2]=buffer[2];
            }
      else
            {
            old_buffer[2]=buffer[2];
            }
     if(HIDTxHandleBusy(lastTransmission) == 0)
     {
         //copy over the data to the HID buffer
         hid_report_in[0] = send_buffer[0];
         hid_report_in[1] = send_buffer[1];
```

```
        hid_report_in[2] = send_buffer[2];

        lastTransmission = HIDTxPacket(HID_EP,
(BYTE*)hid_report_in, 0x03);
    }
     if(send_buffer[0]||send_buffer[1]||send_buffer[2] !=
0)
          {
          salidaLed =1;
          sonidoPiezo();
          }
     else
          {
          salidaLed=0;
          }
}//end Emulate_Mouse


void USBCBSuspend(void)
{

}

void USBCBWakeFromSuspend(void)
{

}

void USBCB_SOF_Handler(void)
{

}


void USBCBErrorHandler(void)
{

}



void USBCBCheckOtherReq(void)
{
    USBCheckHIDRequest();
}//end


void USBCBStdSetDscHandler(void)
{

}//end
```

```
void USBCBInitEP(void)
{
    //enable the HID endpoint

USBEnableEndpoint(HID_EP,USB_IN_ENABLED|USB_HANDSHAKE_ENABL
ED|USB_DISALLOW_SETUP);
}

void USBCBSendResume(void)
{
    static WORD delay_count;

    USBResumeControl = 1;                // Start RESUME
signaling

    delay_count = 1800U;                // Set RESUME line
for 1-13 ms
    do
    {
        delay_count--;
    }while(delay_count);
    USBResumeControl = 0;
}


BOOL USER_USB_CALLBACK_EVENT_HANDLER(USB_EVENT event, void
*pdata, WORD size)
{
    switch(event)
    {
        case EVENT_CONFIGURED:
            USBCBInitEP();
            break;
        case EVENT_SET_DESCRIPTOR:
            USBCBStdSetDscHandler();
            break;
        case EVENT_EP0_REQUEST:
            USBCBCheckOtherReq();
            break;
        case EVENT_SOF:
            USBCB_SOF_Handler();
            break;
        case EVENT_SUSPEND:
            USBCBSuspend();
            break;
        case EVENT_RESUME:
            USBCBWakeFromSuspend();
            break;
        case EVENT_BUS_ERROR:
            USBCBErrorHandler();
```

```
            break;
        case EVENT_TRANSFER:
            Nop();
            break;
        default:
            break;
    }
    return TRUE;
}

/** EOF mouse.c
*********************************************/
#endif
```

8.4 Índice de figuras

Figura	Nombre del archivo	Fuente
1.1	linea_braille.jpg	http://www.baum.de/images/products/svario_torsten.jpg
1.2	ratones_discapacitados.jpg	Elaboración propia
1.3	esquema_proyecto.jpg	Elaboración propia
2.1	replica_prototipo_raton.jpg	http://commons.wikimedia.org/wiki/File:Science_museum_027.jpg
2.2	raton_mecanico.jpg	http://btgsf1.fsanook.com/weblog/entry/185/929009/2.jpg
2.3	raton_optico.jpg	http://news.mydrivers.com/Img/20100621/S10015022.jpg
2.4	raton_laser.jpg	Elaboración propia
2.5	trackball.jpg	http://i.stack.imgur.com/zYvAL.jpg
2.6	teclado_raton.jpg	Elaboración propia
2.7	software_libre.svg	http://commons.wikimedia.org/wiki/File:Mapa_conceptual_del_software_libre.svg
2.8	stallman.jpg	http://img1.liveinternet.ru/images/attach/c/2//72/91/72091223_Richard_Matthew_Stallman.jpg
2.9	gpl.png	http://www.lcosll.org/images/1989.png
2.10	agpl.png	http://upload.wikimedia.org/wikipedia/commons/0/06/AGPLv3_Logo.svg
2.11	bsd.png	http://livedoor.blogimg.jp/webprog/imgs/2/a/2ab166a2.jpg
2.12	mpl.png	http://de.wikipedia.org/wiki/Datei:Mozilla_Foundation_logo.svg
2.13	copyleft.png	http://images1.wikia.nocookie.net/__cb20100206032231/sporerp/images/f/fc/Copyleft.png
2.14	cc.png	http://www.somerights.org.uk/images/logo/cc-black.png
2.15	opensourcehardwarelogo	http://www.nooelec.com/store/media/catalog/p

	.png	roduct/cache/1/image/800x800/9df78eab33525d08d6e5fb8d27136e95/o/s/oshw-logo-800-px.png
2.16	spartan3.jpg	http://tenettech.com/content/images/thumbs/0000076_spartan_3_board_with_200k_gates_600.png
2.17	beepic.png	http://www.teknochips.com/images/Beepic.png
2.18	openeeg.jpg	http://openeeg.sourceforge.net/doc/images/banner.jpg
2.19	partes_openeeg.jpg	Elaboración propia
2.20	Arduino.jpg	http://infouelec.files.wordpress.com/2013/03/arduino_logo.png
2.21	ArduinoDuemilanove.jpg	http://shop.kids-of-all-ages.de/media/catalog/product/cache/2/image/800x600/17f82f742ffe127f42dca9de82fb58b1/a/r/arduinoduemilanove.jpg
2.22	placas_arduino.jpg	Elaboración propia
2.23	pinguino.jpg	http://www.pinguino.cc/
2.24	pinguinos.jpg	Elaboración propia
2.25	Raspberry-Pi-logo.jpg	http://www.bigambition.co.uk/BigAmbition/Images/News/pi%20thumb.png
2.26	Raspberrypi.jpg	http://cdn.imthi.com/c70689/wp-content/uploads/2012/12/Raspberry-Pi.jpg
2.27	Raspberry_Pi_cam.jpg	http://www.elektor.fr/Uploads/2013/5/FR-RS146-Raspberry-Pi-camera-LoRes.1.jpg
3.1	puertos.jpg	Elaboración propia
3.2	arquitectura_red_usb.jpg	Elaboración propia
3.3	arbol_descriptores.png	Elaboración propia
3.4	comunicacion_usb1.jpg	Elaboración propia
3.5	comunicacion_usb2.jpg	Elaboración propia
3.6	lowpincountkit.png	Elaboración propia
3.7	lowpincountboard.png	Datasheet Low Pin Count USB Development Kit
3.8	mplab.jpg	Elaboración propia
3.9	mplab_añadiendo_archivos_enumeracion.jpg	Elaboración propia
3.10	Captura_dispositivo_enumeracion.jpg	Elaboración propia
3.11	flujograma_ejemplo_raton_circulos.jpg	Datasheet Low Pin Count USB Development Kit
3.12	Captura_dispositivo_mouse.jpg	Elaboración propia
3.13	teclado_diagrama_estados.jpg	Datasheet Low Pin Count USB Development Kit
3.14	usando_ejemplo_teclado.jpg	Elaboración propia

3.15	hidbootloader.jpg	Elaboración propia
3.16	arboles_proyecto.jpg	Elaboración propia
3.17	esquema_proyecto_mplab.png	Elaboración propia
3.18	archivos.jpg	Elaboración propia
3.19	flujograma proyecto 1	Elaboración propia
3.20	buffer_raton.jpg	Elaboración propia
3.21	flujograma_SonidoPiezo	Elaboración propia
3.22	onda_piezo.png	Elaboración propia
3.23	onda_piezo2.png	Elaboración propia
3.24	osciloscopio p1.png	Elaboración propia
3.25	osciloscopio p1_2.png	Elaboración propia
3.26	osciloscopio p1_3.png	Elaboración propia
3.27	flujograma proyecto 2.png	Elaboración propia
3.28	flujograma proyecto 3.png	Elaboración propia
3.29	rebote.jpg	Elaboración propia
4.1	actuadores.png	Elaboración propia
4.2	eagle.jpg	http://www.stech.cz/res/dwe-files/401159715.jpg
4.3	eagle_pic18f154k50.jpg	Elaboración propia
4.4	eagle_pic18f154k50_layout.jpg	Elaboración propia
4.5	eagle_pic18f154k50_conexion_pines.jpg	Elaboración propia
4.6	eagle_esquematico_1.jpg	Elaboración propia
4.7	esquema_elaboración_placa2.png	Elaboración propia
4.8	placa perforada.jpg	Elaboración propia
4.9	placa perforada abajo.jpg	Elaboración propia
4.10	pcb.png	Elaboración propia
4.11	fotolito.jpg	Elaboración propia
4.12	exposicion.jpg	Elaboración propia
4.13	revelado.jpg	Elaboración propia
4.14	atacado.jpg	Elaboración propia
4.15	taladrado.jpg	Elaboración propia
4.16	placa_terminada_arriba.jpg	Elaboración propia
4.17	placaterminada_abajo.jpg	Elaboración propia
4.18	gerber_milling.png	Elaboración propia
4.19	gerber_top.png	Elaboración propia
4.20	gerber_board_outline.png	Elaboración propia

4.21	circuitcam.jpg	Elaboración propia
4.22	fresado.jpg	Elaboración propia
4.23	pcb_fresado.jpg	Elaboración propia
4.24	prototipo_banco.jpg	Elaboración propia
4.25	actuadores_banco.jpg	Elaboración propia
4.26	cableado_actuadores.jpg	Elaboración propia