



Laboratorio de desarrollo Hardware

Memoria de PCB

Ramón Chávez García
Curso 2016-2017

ÍNDICE

| | |
|--|-----------|
| OBJETIVOS | 2 |
| DISEÑO DE LA PCB | 3 |
| INTRODUCCIÓN: PLATAFORMAS, ENTORNOS DE DESARROLLO Y COMPONENTES | 3 |
| ➤ Eagle | 3 |
| ➤ Introducción e instalación | 3 |
| ➤ Creación de proyecto y librerías en Eagle | 4 |
| ➤ Diseño del esquemático y layout..... | 7 |
| ➤ Generación de archivos GERBERS y fabricación de la placa | 9 |
| ➤ KiCad | 10 |
| ➤ Introducción e instalación | 10 |
| ➤ Creación de proyecto y librerías en KiCad | 11 |
| ➤ Diseño del esquemático y layout..... | 13 |
| ➤ Componentes | 15 |
| DESARROLLO PRÁCTICO: ENSAMBLADO Y TESTADO DE LA PLACA | 15 |
| DESARROLLO PRÁCTICO: SOFTWARE DE VERIFICACIÓN Y USO COMO MANDO | 19 |
| ➤ Software de verificación | 19 |
| ➤ Configuración de la placa como mando de juego | 20 |
| ➤ UnoJoy..... | 20 |
| ➤ Emulador GBA..... | 27 |
| TRABAJO FINAL Y TEST EURO CIRCUITS | 29 |
| CONCLUSIONES | 33 |

Objetivos

Durante estas sesiones hemos llevado a cabo el desarrollo de un sistema para alguna de las plataformas vistas durante la memoria de prácticas. Concretamente, el trabajo realizado estaba pensado para construir una placa de expansión para **Arduino** o **Papilio**, o bien construir un híbrido que unificase ambas placas de expansión bajo una misma **PCB** (Printed Circuit Board: placa de circuito impreso).

Una vez pensado el diseño, es decir, la disposición de los componentes actuadores y la plataforma (o las plataformas) sobre la que se desarrollaría dicho diseño, el siguiente paso constaba de realizar este diseño con una herramienta software. En esta asignatura se ha dado como opción el desarrollo bajo **Eagle** o **KiCad**.

Tras ello, el siguiente paso era llevar dicho diseño software a una realidad tangible, lo cual logramos con la ayuda del profesor y con una máquina fresadora como lo es la **LPKF ProtoMat S62**.

Fabricada la expansión de la plataforma elegida bajo el programa de diseño, el siguiente paso fue el ensamblado de componentes. Utilizamos productos específicos para alcanzar el objetivo deseado, de forma que el soldado de dichos componentes resultara más cómodo por las propiedades de estas herramientas.

Como último apunte en estas sesiones, y tras realizar los testeos pertinentes, el objetivo final ha sido poder utilizar la placa de expansión desarrollada como un mando. Esto es, que un ordenador reconociese la placa de expansión con esta configuración. La demostración del mismo se ha llevado a cabo con la ayuda de un emulador, ejecutándose la PCB como un dispositivo de control de videojuegos (mando de consola).



Diseño de la PCB

INTRODUCCIÓN: PLATAFORMAS, ENTORNOS DE DESARROLLO Y COMPONENTES

➤ Eagle

➤ *Introducción e instalación*

Es uno de los paquetes softwares más populares para la automatización de diseño electrónico y de PCBs. Alcanza casi 30 años de vida en el mercado y ha estado involucrado en el diseño de PCB estándar para aficionados a la electrónica, estudiantes y firmas de ingeniería que se iniciaron con este software. Las razones son sencillas, es bueno para la mayoría de diseños simples y oferta una versión gratuita. La alternativa open source que puede ser comparable es [KiCad](#), del que hablaremos en el siguiente punto.

[Eagle](#), para lo bueno y para lo malo, es un estándar. Compañías como [Sparkfun](#) o [Adafruit](#) lo usan de manera asidua y han creado librerías de componentes de alta calidad. Durante las clases de laboratorio tuvimos unas clases introductorias a este software.

Fue desarrollado por [CadSoft Computer GmbH](#), que desde 2009 pertenecía a [Farnell](#). Este verano, a finales de junio, [Autodesk](#) se hacía con los servicios de este programa y, ahora mismo, ofrece de forma libre la versión 8.0.0 de Eagle. Actualmente, la página web para la descarga de Eagle ha cambiado drásticamente y ya no ofrece la versión 7.7.0.

Nosotros hemos tratado con la versión 7.7.0. Particularmente, la instalación la he llevado a cabo en [Ubuntu](#). La instalación de esta versión en Ubuntu conllevaba una serie de particularidades. Una vez teníamos descargado el software, debíamos cambiar la configuración del archivo descargado para que fuera ejecutable mediante el comando ***“chmod +x archivoACambiarAEjecutable”*** desde un terminal abierto en la carpeta donde se encontraba. Una vez realizado esto, bastaba con ejecutar dicho archivo escribiendo en la terminal ***“./archivoAEjecutar”***.

La instalación la dejamos por defecto y elegimos la licencia Express (había una licencia educativa que incluía otros aspectos). El programa en sí se encuentra dentro de la carpeta ***“eagle-7.7.0/bin”***. Bajo ***“eagle-7.7.0/”*** hay una serie de carpetas, además de ***“bin”***, entre las que destacamos: ***“lib”***, carpeta destinada a las librerías de componentes; ***“projects”***, para proyectos; ***“doc”***, para documentación.

Dado que íbamos a utilizar el programa en más de una ocasión, decidí incorporar el programa al lanzador de Ubuntu. Para ello, abrimos **gedit** y creamos un archivo con el siguiente contenido:

```
[Desktop Entry]
Name=Nombre que le queremos dar al programa
Comment=Comentario al poner el cursor sobre el icono del programa
Exec=/home/usuario/carpetaPrograma/bin/programa
Icon=/home/usuario/Images/iconoPrograma
Terminal=false
Type=Application
```

Guardamos este archivo con este formato: ***“nombreArchivo.desktop”*** en la ruta ***“/usr/share/applications/”***. Acto seguido, buscamos el programa desde el **dash** de Ubuntu y lo ejecutamos. Una vez en ejecución hacemos click derecho sobre su icono en el lanzador y le damos a mantener en el lanzador. Recurrí a este método porque ejecutándolo desde la terminal y anclándolo al lanzador no se mantenía una vez cerraba el programa.

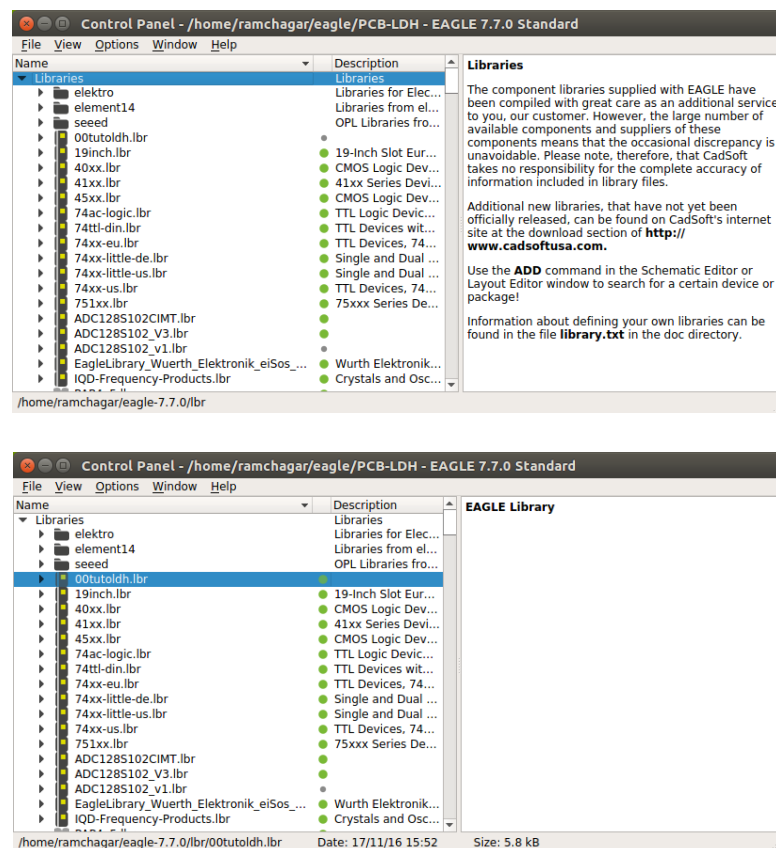
➤ ***Creación de proyecto y librerías en Eagle***

La creación de un proyecto en Eagle es algo sencillo. Primeramente, expandimos el desplegable ***“Projects”*** y seleccionamos la carpeta donde guardaremos nuestro proyecto. Acto seguido, pulsamos botón derecho sobre ella y seleccionamos ***“New Project”***, escogemos un nombre para este y ya tenemos el proyecto vacío.

La creación de un proyecto requiere de unos componentes, que en ocasiones puede que no contemos con ellos. Este tipo de herramientas incluyen la opción de diseñar nuestras propias partes bajo lo que se conoce como **librería**, de importar las librerías propiamente dichas, o de utilizar las que vienen por defecto en Eagle. En mi caso hemos hecho uso de las tres opciones.


Utilicé la librería que hicimos en clase para los botones; busqué e importé de Internet la de la ferrita, la de la board de Arduino y la del Joystick; modifiqué la del ADC (cambiando el símbolo únicamente) y la de las wings de Papilio (retiré pines que no usábamos); utilicé las que incluía Eagle para las resistencias, alimentaciones, condensadores y LEDs.

Para crear una nueva, pulsamos sobre ***“File > New > Library”*** y abrimos el editor de librerías. Guardamos nuestra librería. Justo después, es necesario activarla dirigiéndonos a la pantalla inicial una vez ejecutamos el programa, en ***“Libraries”*** y pulsamos sobre el círculo gris que aparece a la derecha de nuestra recién creada librería. Dicho círculo se convierte en verde, quedando la librería activada:



De vuelta a nuestra librería diseñamos las tres partes de las que consta: **símbolo**, **huella** y **dispositivo**.

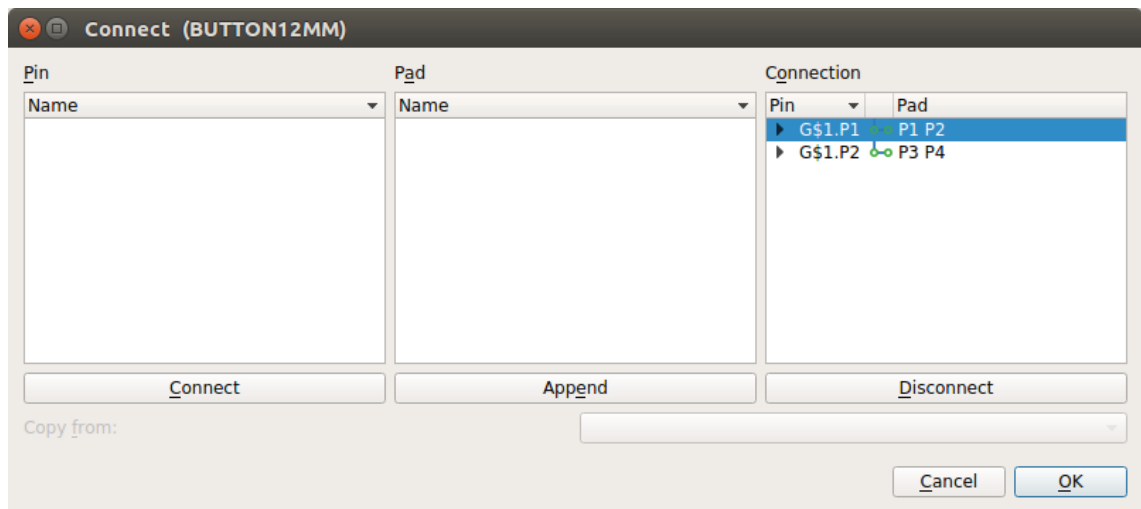
El primero de ellos es la representación del dispositivo en el esquemático. La huella o package es la representación que tiene el componente con las medidas exactas que posee en la realidad. El dispositivo es la combinación de ambos, y relaciona los **pin**es del primero con los **pad**s (huellas) del segundo. Eagle permite utilizar huellas y símbolos para conformar nuevos dispositivos.

La creación de un nuevo dispositivo pasa por la elaboración de la huella y símbolos del mismo. Una vez abierta la librería creada, pulsamos sobre los botones que permiten crear estas partes (). El primero de ellos que fabricamos es la huella (se corresponde con el icono del medio), así que pulsamos sobre el icono correspondiente a package y, fijándonos en el **datasheet** o las especificaciones de los componentes, construimos la huella centrándola en la cruceta, para que luego al moverla en el **layout** nos sea sencillo.

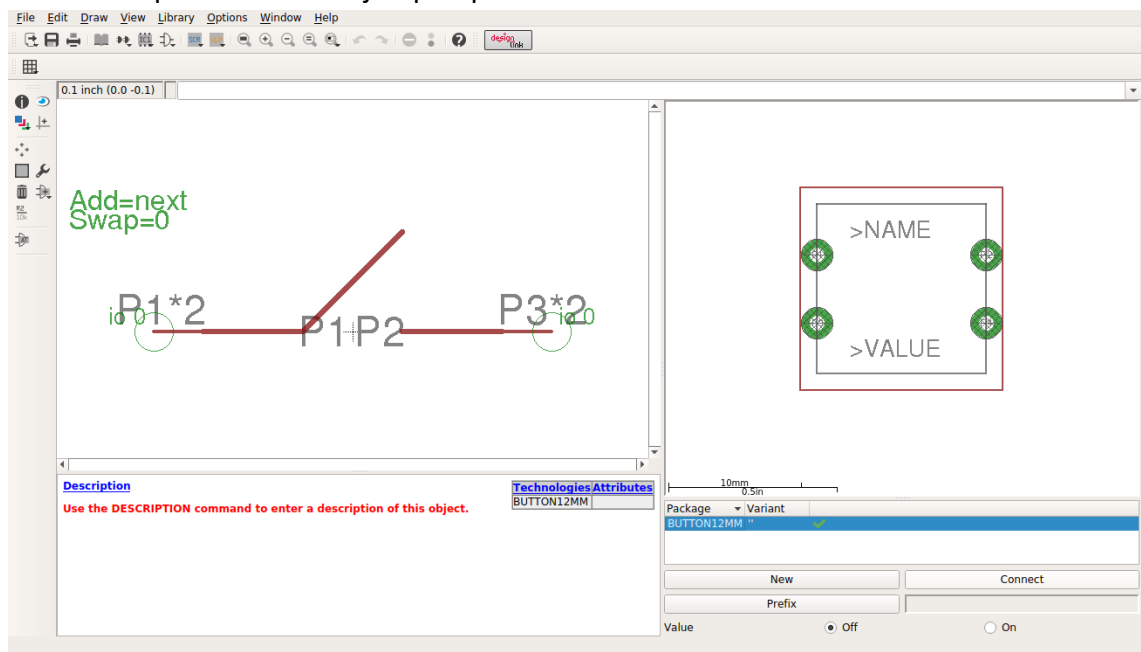
Es importante seleccionar el tipo de pad del componente, así como sus dimensiones. Colocados los pads, estos se deben renombrar y debemos hacer uso de las etiquetas '>NAME' y '>VALUE'. Una buena práctica es añadir una capa de envoltura, para evitar que los componentes se interpongan (choquen) entre ellos.

Para dibujar el símbolo, clicamos sobre el último icono y dibujamos el contorno del símbolo. A continuación, colocamos los pines y los renombramos. De forma similar al procedimiento anterior, hacemos uso de las etiquetas '>NAME' y '>VALUE' y centramos en la cruceta.

Tenemos las dos partes que conforman al dispositivo, así que nos ponemos a crear este. Pulsamos sobre el primer icono de los mostrados tres párrafos atrás y le damos un nombre a nuestro dispositivo. Si hacemos click en 'New' podemos seleccionar la huella que hemos creado anteriormente. El botón para añadir símbolos permite incluir el que nosotros hemos creado. Una vez hecho esto, solo falta conectar ambas partes con el botón 'Connect':

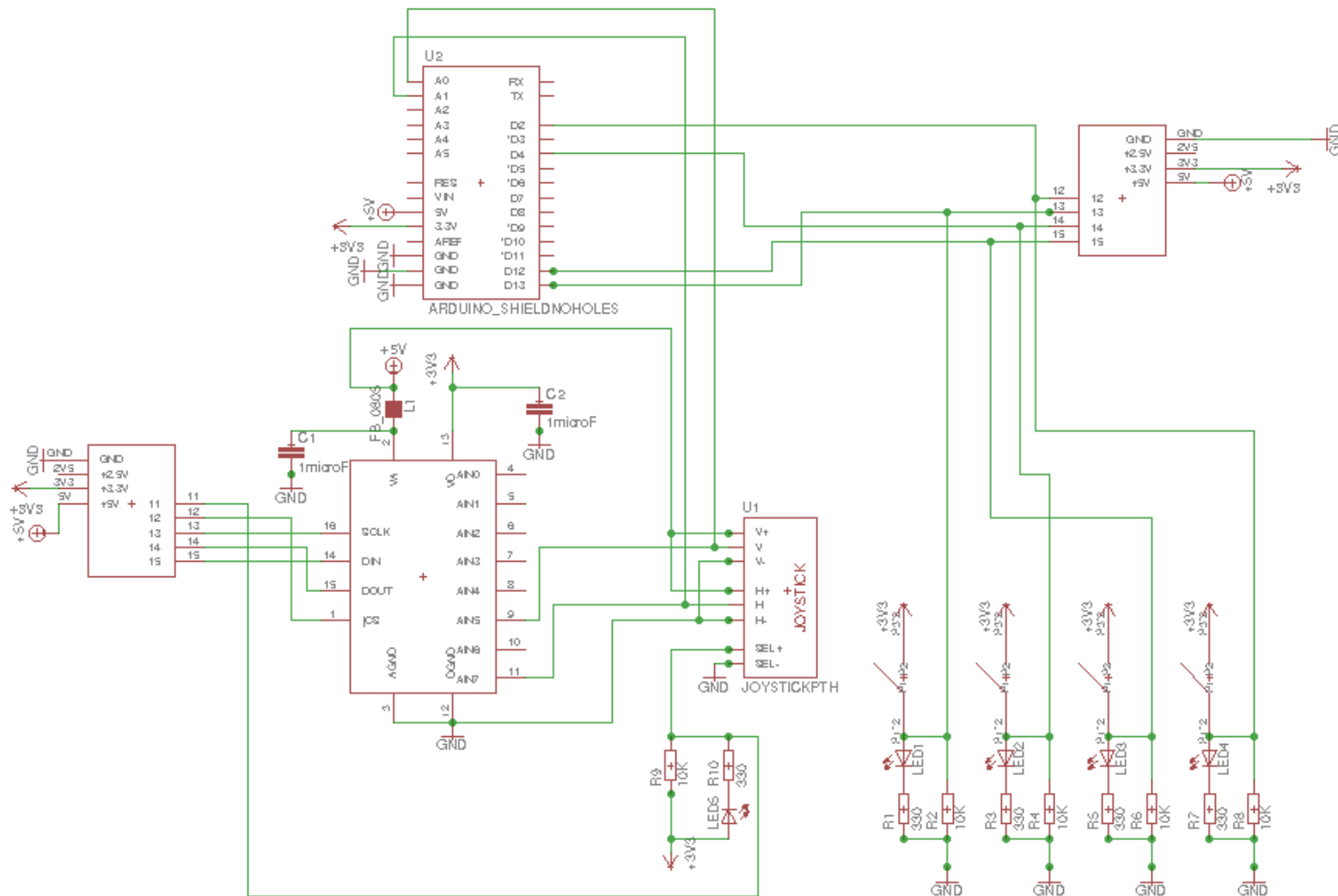


La librería quedó así en el ejemplo que desarrollamos en clase:



➤ **Diseño del esquemático y layout**

Bajo el proyecto que hemos creado previamente en el apartado anterior, seguimos los pasos *'File > New > Schematic'* y lo guardamos. Acto seguido, añadimos todos los componentes que vayamos a utilizar (podemos utilizar la búsqueda desde el menú de añadir componentes). Conectamos todos estos mediante la herramienta *'Wire'* y comprobamos si existen errores con la herramienta ERC (Electrical Rule Check). El resultado es este:



Como se puede apreciar, hubo un fallo de diseño y tiré una pista desde el botón del joystick a Arduino. Este error fue solventado por medio de un cable al pin 3 de Arduino. Pasado el chequeo, pulsamos sobre *'File > Switch to board'* para cambiar a la vista del layout.

El layout representa el espacio real de trabajo, es decir, como vamos a distribuir los componentes en la realidad a la hora de fabricar nuestra placa. En primer lugar, se define el área de trabajo con la herramienta *'Info'*. Movemos los componentes dentro de dicho área. Es bueno utilizar la herramienta *'Ratsnest'* para ver las conexiones que tenemos que realizar trazando pistas posteriormente. Una vez tenemos los componentes colocados, procedemos a

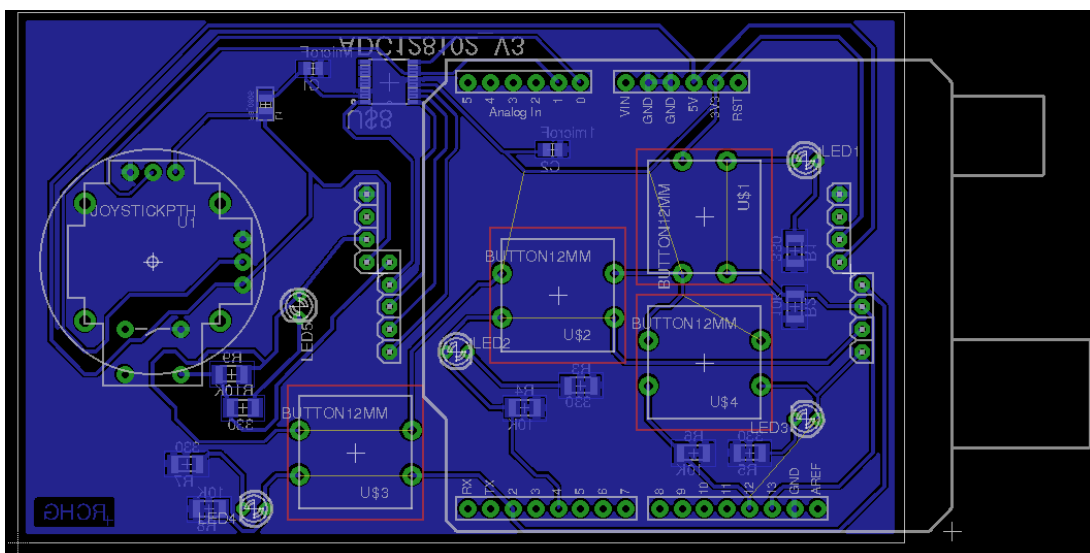
trazar dichas pistas. Esto se puede hacer de forma manual o automática, aunque la herramienta automática no garantiza hacer el 100% de las rutas, con lo que se recomienda encarecidamente el trazo manual.

Durante el rutado de la PCB es posible que, dada la interconexión de dispositivos, sea necesario recolocarlos. Esto se debe repetir hasta encontrar una disposición óptima y adecuada. Es importante el tamaño de la pista, así como la distancia mínima que debe haber entre varias rutas. Conocidos estos aspectos, trazamos todas las rutas a excepción de GND.

La creación de la GND la haremos mediante un plano de tierra. Esto consiste en cubrir con el polo negativo la mayor cantidad de cobre posible. Seleccionamos la capa “bottom”, ya que nuestra placa es de una sola cara de soldadura, y pulsamos sobre la herramienta ‘Polygon’. Podemos fijar un “Isolate” de igual manera que lo hicimos para las pistas (para dejar como espacio de separación mínimo).

El uso que le damos a la herramienta del polígono, es envolver toda la placa con ella, esto es, crear un rectángulo que cubra toda el área de trabajo. Después, pulsamos sobre la herramienta que nos permite renombrar esta área como ‘GND’. Justo en ese momento, cuando pulsamos en ‘OK’, conectamos a la señal de ‘GND’ en el siguiente cuadro de diálogo. Aceptamos y vemos el resultado. En caso de que no se muestre nada, podemos refrescar con la herramienta ‘Ratsnest’.

Hay que tener en cuenta que hay componentes que poseen terminales conectados de manera interna, por lo que no es necesario trazar la ruta aunque nos la pida el layout (el caso de los botones creados manualmente). Estos pueden dar errores DRC (Design Rule Check) cuando pasemos esta herramienta al acabar la PCB, pero debemos obviarlos (no así el resto de errores, que deben ser tratados y resueltos adecuadamente):



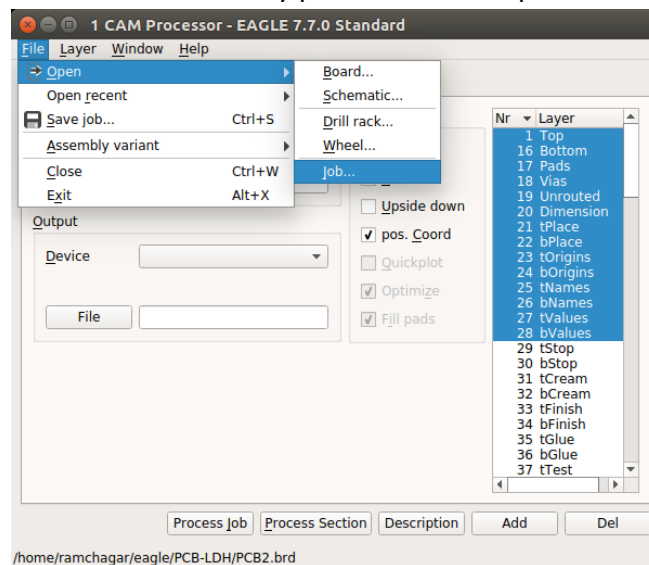
Finalizado el diseño del layout, pasamos a generar los archivos que nos permiten fabricarlo.

➤ Generación de archivos GERBERS y fabricación de la placa

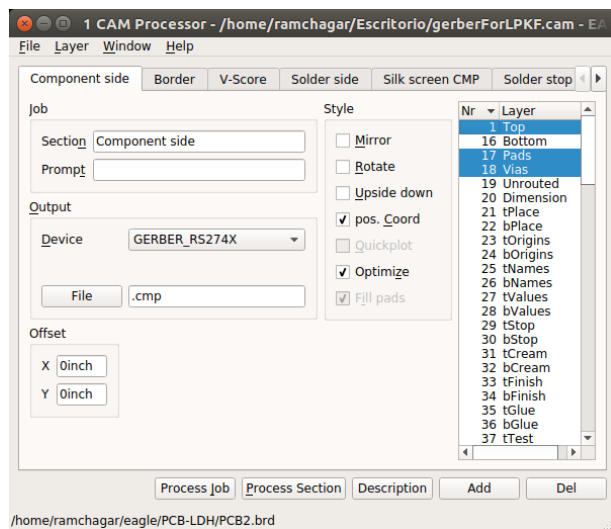
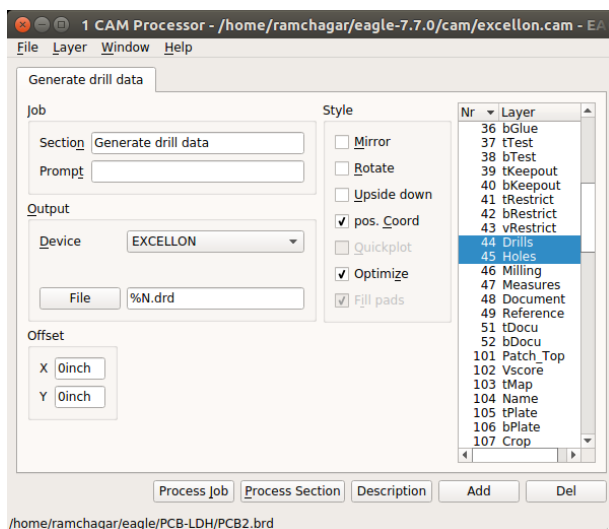
Llega el momento de hacer nuestra placa una realidad. Para ello transformamos nuestra board en ficheros **GERBERS**. Estos son ficheros de extensión conocida en el mundo de la electrónica y que contienen información de las distintas capas que conforman nuestro layout (tamaño de los taladros a realizar, de las pistas a trazar,...).

Contamos con una máquina en el laboratorio que se encarga de transformar estos ficheros en un objeto físico: ProtoMat S62. Esta máquina es la fresadora de la que debemos tomar los ficheros necesarios para generar los GERBERS con los que fabrique nuestra placa. Estos se encuentran [aquí](#).

La forma de generar dichos GERBERS es sencilla. Pulsamos en Eagle, desde el layout, sobre 'File > CAM Processor' y procedemos tal que así:



Cargamos nuestros dos ficheros descargados previamente:



En ambos casos presionamos sobre *'Process Job'*. No es necesario modificar parámetros ni guardar cambios tras cerrar esta ventana. Los archivos generados se encuentran en la misma ruta donde tenemos nuestro layout.

Los archivos que le enviamos al profesor son los *".cmp"* (componentes), *".bor"* (dimensiones), *".dri"* (taladros usados), *".drd"* (posicionamiento de los agujeros), *".sol"* (capa de soldadura). Estos son los que necesitó para fabricar la placa con la máquina mencionada anteriormente. Dicha máquina tipo plotter elabora un "dibujo" eliminando el cobre necesario para la creación de las pistas y pads, así como de los agujeros correspondientes.



➤ KiCad

➤ *Introducción e instalación*

KiCad es un entorno de desarrollo software empleado para en el diseño de circuitos electrónicos. Se caracteriza por ser abierto, libre, muy flexible y adaptable. Su herramienta principal para el diseño inicial de dichos circuitos es el *'Eeschema'*, donde se colocan los componentes creados y editados, así como los que vienen por defecto al instalar esta plataforma. KiCad permite el diseño de circuitos impresos modernos de forma sencilla e intuitiva.

Además, con su herramienta *'Pcbnew'*, podemos diseñar circuitos con múltiples capas y ser visualizados en 3D. Este diseño en 3D también puede ser creado por el usuario de KiCad.

Otras herramientas interesantes con las que cuenta este programa son *'CvPcb'* y *'Gerbview'*. La primera de ellas se encarga de asignar una huella a cada componente posicionado en nuestro esquemático, para modificar su posición final en la board. En cuanto a la segunda, su utilidad reside en poder visualizar los archivos GERBERS para comprobar que se han generado correcta y

adecuadamente. Estos ficheros serán los que enviaríamos a fábrica una vez se encuentre validada nuestra PCB.

La creación de este software se inició en 1992 por Jean-Pierre Charras, aunque actualmente está bajo desarrollo por el Equipo de Desarrolladores Kicad. Es soportado por la Universidad de Grenoble (universidad donde trabajó el autor original) y GIPSA-lab, SoftPLC, el CERN, la Fundación Raspberry Pi, Arduino LLC, GleSYS y Digi-Key Electronics.

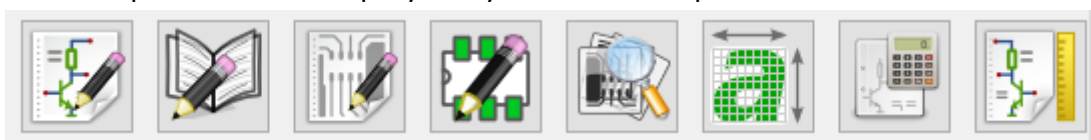


La instalación es estándar. Lo único que tenemos que hacer es descargarlo y seguir los pasos que muestras las ventanas. En mi caso lo he instalado en Windows, principalmente por comodidad a la hora de editar este documento, pero está disponible para una gran cantidad de sistemas.

➤ **Creación de proyecto y librerías en KiCad**

Este programa ha sido traducido al español “recientemente” en su totalidad, lo cual facilita el trabajo que desarrollamos en él. Así pues, la creación del proyecto procede de la siguiente manera: abierto el programa, pulsamos sobre “*Archivo > Nuevo Proyecto > Nuevo Proyecto*”. Acto seguido, seleccionamos la ubicación donde queda guardado nuestro proyecto.

Vemos que se nos crea el proyecto y esta serie de opciones:



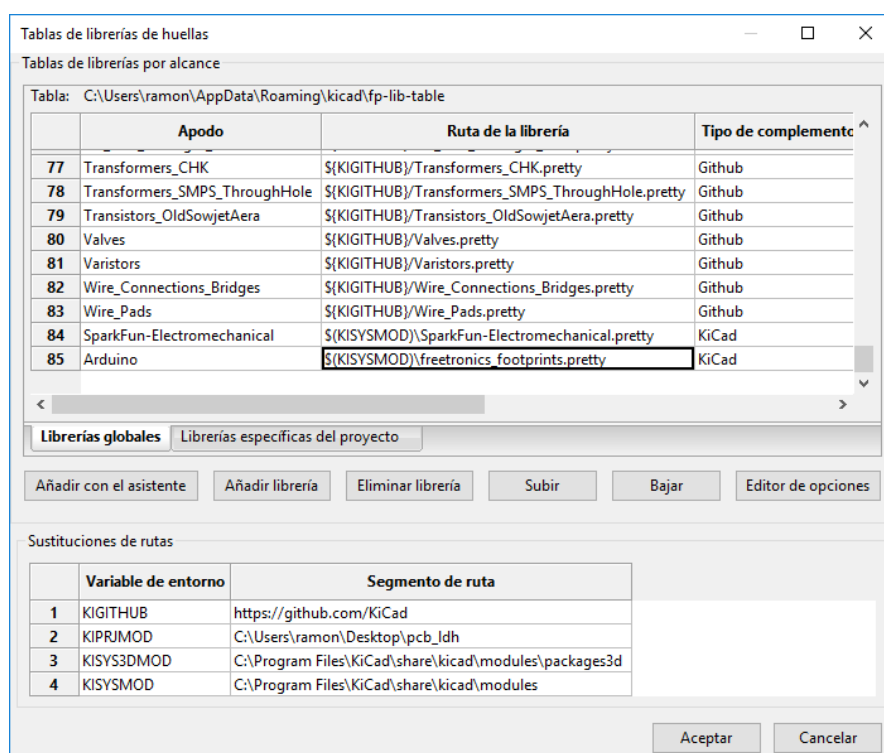
Al pulsar sobre la primera comenzando por la izquierda, abrimos el que será nuestro esquemático. Ahora bien, no tenemos todos los componentes que vamos a usar en este proyecto, que será un diseño para un shield de Arduino.

Es necesario importar las librerías. En este caso, no ha sido necesario crear ninguna, por lo que el proceso de creación de librerías no queda contemplado en esta memoria. Sin embargo, el método de importación de las mismas es un tanto peculiar, así que procedo a detallarlo.

Una vez ubicamos las librerías que queremos importar (en nuestro caso faltaban las del joystick, la de la board de Arduino y la de los botones), procedemos a “instalarlas” en KiCad. Las librerías en KiCad tienen dos partes: una de ellas es un “.lib” que debemos colocar bajo la ruta “C: \ Program Files \ KiCad \ share \ kicad \ library” y hace la función de símbolo; la segunda es un “.kicad_mod”, está bajo una carpeta del tipo “NombreDeLaLibreria.pretty” que se encuentra en “C: \ Program Files \ KiCad \ share \ kicad \ modules” y hace la función de huella.

KiCad también ofrece un modelado 3D, aunque no hemos entrado en detalle sobre él. Volviendo al tema anterior, es recomendable poner nuestros archivos en dichas rutas para lo que es la importación de las mismas. La librería del símbolo se carga desde nuestro esquemático. Pulsamos sobre “*Preferencias > Librerías de componentes*”. Seleccionamos la ruta y le damos a “*Añadir*”. Buscamos la librería y la seleccionamos. Pulsamos aceptar y ya tendremos el símbolo.

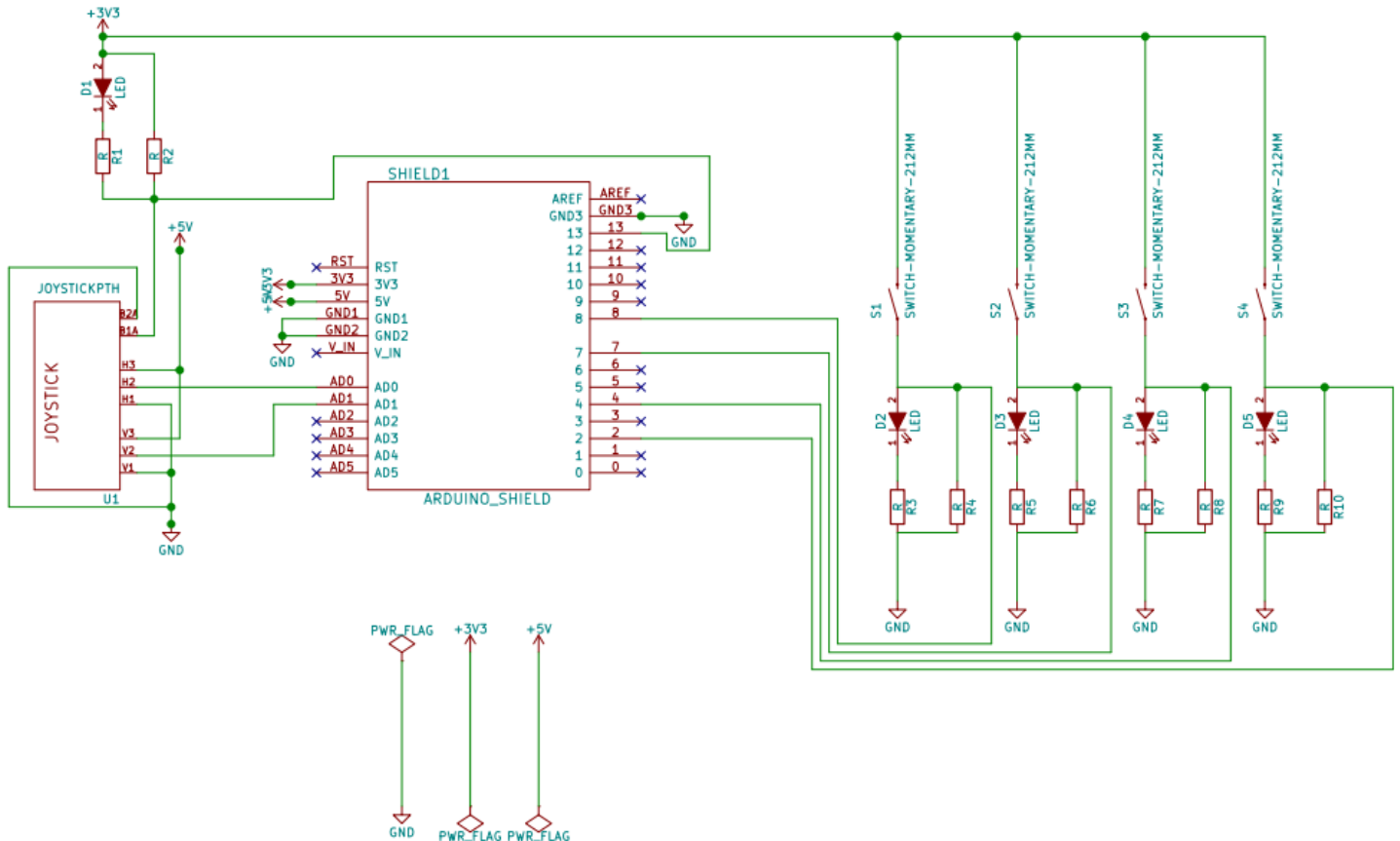
Para la huella, la forma de proceder es similar. Abrimos el editor de huellas, que es el cuarto icono desde la izquierda de la imagen que puse en la página anterior. De nuevo en “*Preferencias > Administrador de librerías de huellas*”. La carga de las huellas es mejor hacerla de forma manual; así pues, pulsamos sobre “*Añadir librería*” y ponemos la ruta de la carpeta a la que hacíamos referencia dos párrafos atrás:



Como se puede observar, se ha utilizado una macro de ruta relativa hasta dicha carpeta. Pulsando sobre “Aceptar” tenemos nuestra librería agregada. Si nos fijamos en la última columna, en nuestro caso hemos importado dos librerías (aparece KiCad en la columna tipo de complemento).

➤ Diseño del esquemático y layout

El diseño del esquemático corresponde al shield de Arduino y es el siguiente:



Se observan algunas diferencias apreciables con respecto al esquemático de Eagle. Las más curiosas son las cruces en los pines no conectados y las “PWR_FLAG”. La razón de incluir los primeros es para indicar al ERC que no hay errores al dejar esos pines sin conectar; las segundas se incluyen para indicarle a la misma herramienta que son señales de alimentación.

La herramienta ERC es la del medio.



La de la izquierda permite la anotación o nombrado de los componentes y la de la derecha la generación de la red de estos. Estas dos últimas herramientas constituyen un paso intermedio junto a ‘CvPcb’ ().



Este paso requiere primero de un nombrado de componentes, luego el uso de ‘CvPcb’ para asignar las huellas a cada símbolo (muy diferente de Eagle, que directamente un símbolo tiene una huella en cuanto lo ponemos sobre el esquemático) y finalmente la generación de la red de componentes.

Tras realizar estos pasos, abrimos 'Pcbnew' (tercer icono por la izquierda de las opciones que presentamos tres páginas atrás) y lo primero que hacemos es leer la "netlist", que tiene el mismo símbolo que en el esquemático. La cargamos y tenemos los componentes sobre el área de trabajo.

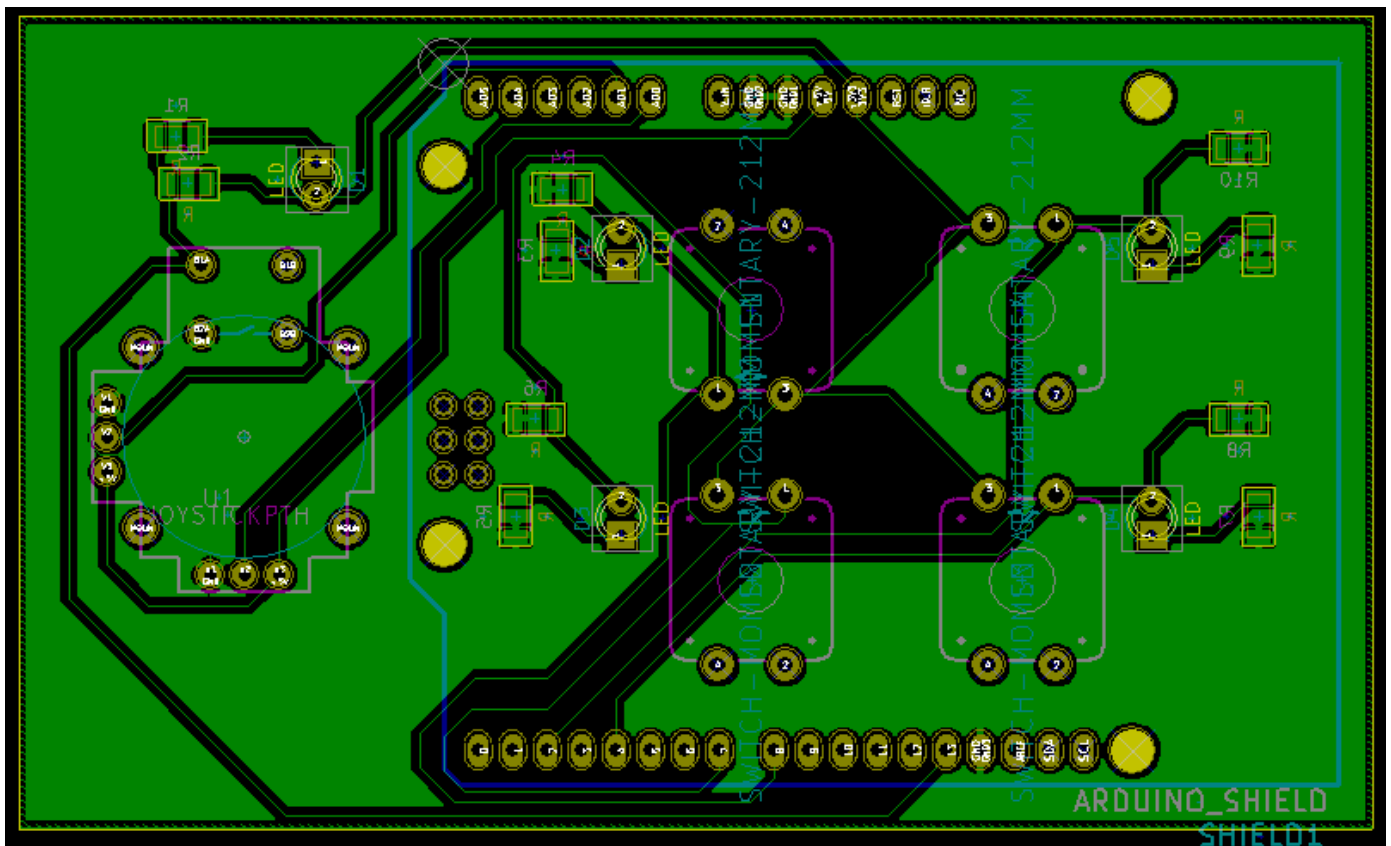
Lo siguiente es definir las dimensiones de nuestra placa. Para ello recurrimos a la capa 'EdgeCuts' y pulsamos sobre su margen izquierdo, quedando seleccionada. A continuación, pulsamos sobre la siguiente herramienta:



Dibujamos cuatro líneas que no tienen por qué formar un cuadrado, pues las vamos a dimensionar de forma absoluta. Nos colocamos sobre cualquiera de ellas y pulsamos 'e', dotándola de las medidas que queremos (en nuestro caso 10cmx6cm).

El siguiente paso ya sabemos cuál es, disponemos los componentes sobre las dimensiones de nuestra placa y hacemos el trazado manual. Para agregar el plano de tierra hacemos click sobre "Añadir zonas de relleno", dos herramientas por encima de la anterior, y sobre la capa de cobre en la cara bottom ('B.Cu').

Clicamos en una de las esquinas de la placa y seleccionamos GND en la ventana que se nos abre. Rodeamos toda la placa y llegamos al punto de partida formando el rectángulo de GND. Como resultado, tenemos nuestra placa:



Es bueno realizar el DRC (mismo símbolo que el ERC del esquemático) para evitar dejarnos alguna pista sin conectar o algún otro tipo de problema (que una pista pise un pad o similar).

➤ Componentes

Al desarrollar la placa compatible con Arduino y con Papilio, los componentes electrónicos que hemos necesitado para el diseño físico son los siguientes:

- Joystick (Thumb Joystick Sparkfun).
- 4 Momentary pushbutton switch 12mm square
- ADC128S102
- 5 resistencias 10K Ω de encapsulado SMD 1206
- 5 resistencias 330 Ω de encapsulado SMD 1206
- 2 condensadores 1 μ F de encapsulado SMD 0805
- 1 ferrite bead de encapsulado SMD 0805
- Tiras de pines estándar
- LEDs Through-Hole de 3mm

DESARROLLO PRÁCTICO: ENSAMBLADO Y TESTADO DE LA PLACA

Una vez fabricada la placa, lo primero que debemos realizar es un test. Puede sonar confuso si lo escuchas por primera vez, pero este test es muy necesario para evitar arrastrar errores luego. Esta comprobación se realiza para asegurarnos de que la fabricación de la placa no ha tenido percance alguno (al incluir el nombre nos ha pisado una pista, nos ha levantado otra o ha cortado la placa siguiendo unas dimensiones que no eran las esperadas).

Para ello hacemos uso del polímetro. Esta herramienta nos va a ser muy útil a lo largo de todo el proceso de ensamblado. Con poner ambos terminales, podemos conocer si hay conexión entre ellos (emite un pitido) o no. Esto consigue que durante la soldadura de componentes conozcamos si cometemos un corto y repararlo si así ocurre.

Antes de comenzar a soldar, configuramos la estación de soldadura. Normalmente con 350° es suficiente para fundir la aleación de estaño, aunque en determinados momentos funcionaba mejor a 400° (por tanto, siempre hemos realizado una **soldadura blanda** por debajo de 450°). El estaño del que disponemos no es el usual (que tiene una composición aproximada de 63% Sn y 37% Pb), por lo que ha dificultado un poco el desarrollo de las sesiones (actualmente se utiliza estaño sin plomo, con lo que la aleación suele estar en torno a 96,5% Sn y 3,5% Ag).

Sin embargo, con ayuda del **Flux (fundente)** hemos solventado los problemas que se nos han presentado. Este material se aplicaba antes de realizar la soldadura

en sí, limpiando de impurezas la superficie y ayudando a nuestra soldadura a distribuirse por el pad de manera adecuada y evitando que tocara el plano de tierra u otra pista.

Al soldador hubo que pre-estañar tras una previa limpieza de la punta con la esponja, para que atrapara la aleación de forma correcta. Con todo esto ya podíamos empezar a soldar.

El primer paso es soldar los componentes **SMD**, ya que la placa, al estar desnuda, es totalmente estable y son los componentes más pequeños y que no van a suponer un desequilibrio de la misma. Para realizar esta soldadura, es necesario pre-estañar los pads (al menos uno de ellos); seguidamente, colocamos el componente con las pinzas y con ayuda del soldador conseguimos soldarlo. Ahora completamos el otro extremo acercando aleación y el soldador, y retirando ambos en el mismo orden que han sido colocados tras dos o tres segundos.

A excepción del ADC, elemento que se dejó para la soldadura final, todos los componentes SMD se sueldan al comienzo. Lo siguiente a soldar suelen ser las tiras de pines, que ya son componentes **Through-Hole** y permiten darle cierta estabilidad a la placa antes de unir a ella el resto de componentes. La forma de proceder es algo distinta a los SMD.

Los componentes Through-Hole se sueldan poniendo el soldador en un lado del pad y el estaño justo en el opuesto. Si lo hacemos bien, el soldador calentará todo el pad, y al acercar el estaño a este, se fundirá, recubriendo todo el pad y uniéndose al pin del conector.

De la misma manera, procedemos a soldar botones, LEDs y finalmente el joystick, ya que es el componente que más abulta y es el más problemático en cuanto a estabilidad de la placa se refiere. Los LEDs tienen la particularidad de que debemos cortar sus patillas con los alicates antes de soldar. Es importante fijarse bien en la polaridad de los mismos y hay que intentar que no se caigan en el proceso, ya que después quizás no podamos identificar la polaridad correctamente.

La soldadura del ADC se llevó a cabo al final, y resultó fallida. Aunque se consiguió soldar el ADC con ayuda del microscopio y los soldadores más precisos de otro laboratorio, por algún motivo las señales del joystick no llegaban adecuadamente a los pines. Finalmente, se decidió por no soldar el ADC.

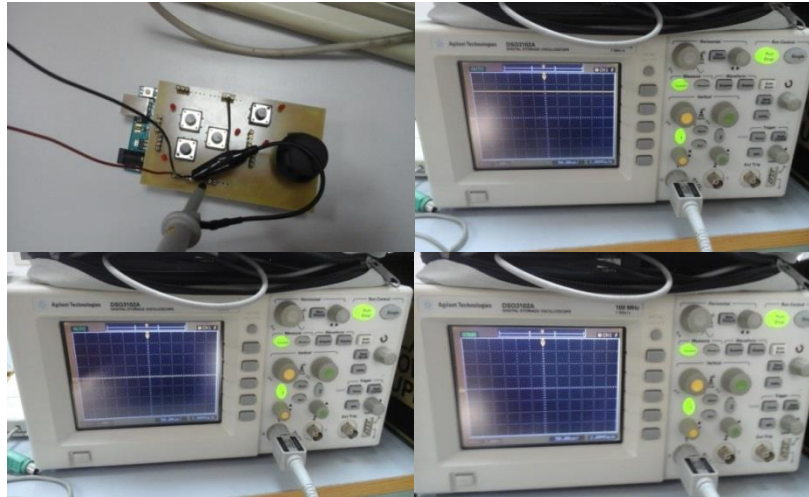


En ocasiones, el Flux no permitía la solución total de los problemas. Alguna vez hemos hecho uso de un cúter, una malla desoldadora o del propio soldador; eso sí, con sumo cuidado para no dañar las pistas (en caso de dañar alguna pista, ha sido necesario reconducir dicha pista con ayuda del cúter y aleación, “creándola” de nuevo, o bien ha sido necesario recurrir a un cable).

Como hemos comentado anteriormente, el polímetro ha sido un gran aliado en estas sesiones, y gracias a él hemos podido comprobar si todo estaba correcto o no en todo momento.

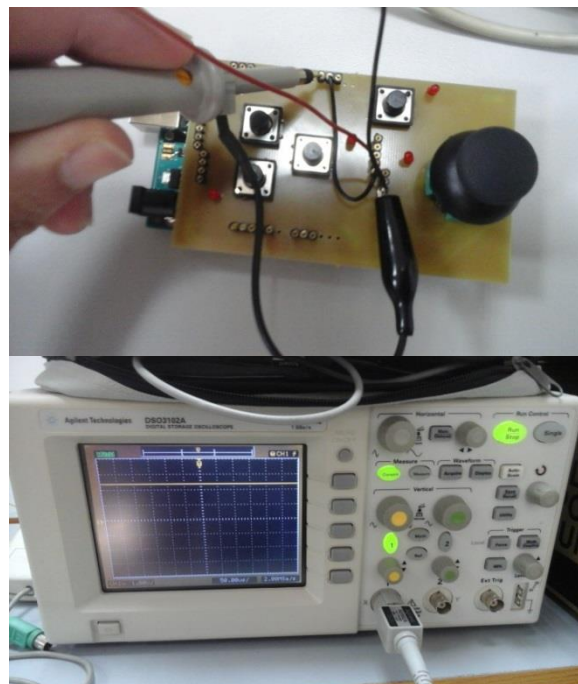


Tras la comprobación de todas las pistas, el siguiente paso era comprobar si electrónicamente, nuestro diseño funcionaba. Para ello nos recurrimos al uso de una **fente de alimentación** y un **osciloscopio**. Conectando la fuente a los pines de alimentación de nuestra placa, debíamos comprobar que obteníamos, de manera efectiva señal en los pines correspondientes al pulsar los botones y mover el joystick:



Gracias al osciloscopio, pudimos ver que se producían fallos en la lectura de ambos ejes del joystick. Tras investigar si era fallo del propio joystick o no (algo que es casi improbable que suceda), el fallo suponía la conexión del ADC. Desoldando el mismo, desgraciadamente, perdió tres patillas. Además, se levantaron pistas del mismo, por lo que la placa pasó de ser compatible para ambos diseños a ser válida solamente para Arduino.

El diseño en Papilio, en lo que respecta a botones funciona correctamente, por lo que, aunque no se pueda destinar su uso a mando, puede dársele otro uso de control, por ejemplo:



DESARROLLO PRÁCTICO: SOFTWARE DE VERIFICACIÓN Y USO COMO MANDO

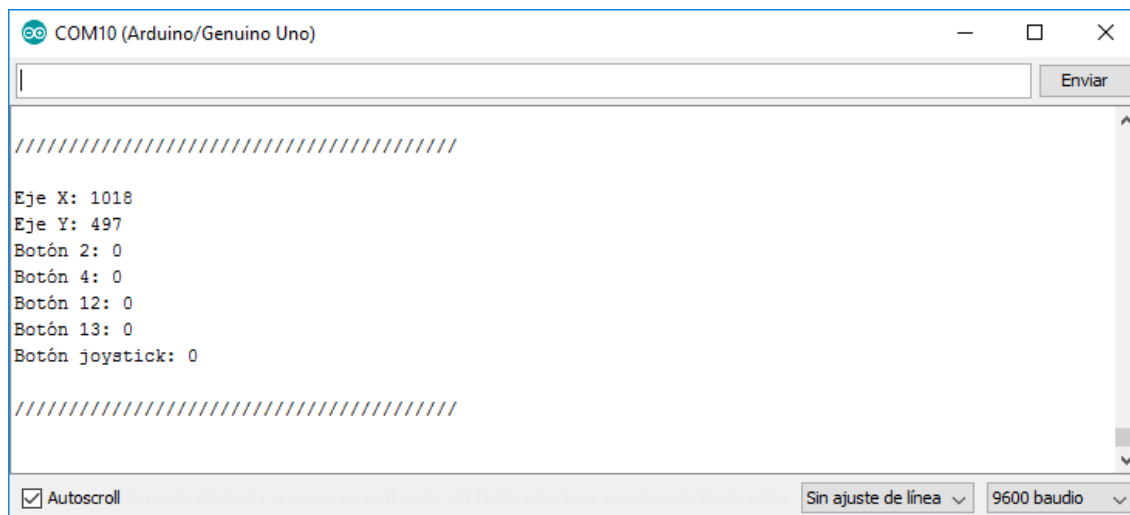
➤ Software de verificación

Aunque en el apartado anterior hicimos uso del osciloscopio para descubrir la falla del ADC, recurrimos también a Arduino para mostrar que recibimos los valores correctamente. Para ello desarrollamos el siguiente sketch:

```
void setup() {
  //Inicialización de pines digitales (botones) y analógicos (joystick)
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(4, INPUT);
  pinMode(12, INPUT);
  pinMode(13, INPUT);
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  //Inicialización del monitor serie
  Serial.begin(9600);
}

void loop() {
  Serial.print("Eje X: ");
  Serial.println(analogRead(A1));
  Serial.print("Eje Y: ");
  Serial.println(analogRead(A0));
  Serial.print("Bot");
  Serial.print((char)243);
  Serial.print("n 2: ");
  Serial.println(digitalRead(2));
  Serial.print("Bot");
  Serial.print((char)243);
  Serial.print("n 4: ");
  Serial.println(digitalRead(4));
  Serial.print("Bot");
  Serial.print((char)243);
  Serial.print("n 12: ");
  Serial.println(digitalRead(12));
  Serial.print("Bot");
  Serial.print((char)243);
  Serial.print("n 13: ");
  Serial.println(digitalRead(13));
  Serial.print("Bot");
  Serial.print((char)243);
  Serial.print("n joystick: ");
  Serial.println(digitalRead(3));
  Serial.println("");
  Serial.println("////////////////////////////////////////");
  Serial.println("");
  delay(3000);
}
```


El resultado de este pequeño programa es el siguiente:



En esta captura de la ejecución del mismo, observamos que no se está pulsando ningún botón y que el joystick está desplazado en su eje X hacia la derecha.

➤ Configuración de la placa como mando de juego

➤ *UnoJoy*

Para lograr este objetivo final, hemos configurado nuestra placa de expansión como un mando de videojuegos mediante **UnoJoy**, un proyecto que permite precisamente esto.

Lo primero que tuvimos que hacer es instalar **Flip** desde la [página de Atmel](#). Este software es necesario para cargar el archivo hexadecimal cuando programamos el Arduino Uno en modo DFU (Device Firmware Update). Este modo permite cambiar el **firmware** de Arduino, de forma que el PC pueda reconocer Arduino como un joystick en este caso.

Tras instalar Flip, debemos descargar el proyecto UnoJoy desde [aquí](#). Colocamos nuestra placa sobre el Arduino y cargamos desde el IDE de Arduino este sketch (contiene la configuración para mi placa en particular. Habría que cambiar los botones según la placa de expansión, pero el código es muy intuitivo):

```
#include "UnoJoy.h"

void setup() {
  setupPins(); //Subrutina que configura los pines de Arduino
  setupUnoJoy(); //Inicializa las funciones de UnoJoy
}

// Actualización de los datos enviados al computador
void loop() {
  dataForController_t controllerData = getControllerData();
  setControllerData(controllerData);
}

// Configuramos los pines digitales como entrada y con pull-up activo
```

```
void setupPins(void) {
    for (int i = 2; i <= 13; i++) { //Solo utilizamos 5 pines.
        pinMode(i, INPUT);
        digitalWrite(i, HIGH);
    }
}

//Configuración local de como los datos de control son almacenados
dataForController_t getControllerData(void) {
    //Limpia el buffer de memoria donde son almacenados los datos de control
    dataForController_t controllerData = getBlankDataForController();
    //Asocia los pines digitales con un botón de UnoJoy
    controllerData.triangleOn = !digitalRead(13);
    controllerData.circleOn = !digitalRead(13);
    controllerData.squareOn = !digitalRead(4);
    controllerData.crossOn = !digitalRead(12);
    controllerData.selectOn = digitalRead(3);
    controllerData.startOn = !digitalRead(2);

    //Configuración del joystick analógico
    /*Como la lectura de un pin analógico devuelve un valor 10
    bits(0~1023), utilizamos un mapa para pasar a 8 bits (0~255).
    Aprovechamos para invertir la lectura del eje Y*/
    controllerData.leftStickX = map((analogRead(A1)), 0, 1023, 0, 255);
    controllerData.leftStickY = map((analogRead(A0)), 1023, 0, 255, 0);
    return controllerData;
}
```

Este sketch incluye una librería que viene en el proyecto de UnoJoy descargado anteriormente. Es este “.h”:

```
/* UnoJoy.h
   Alan Chatham - 2012
   RMIT Exertion Games Lab
```

This library gives you a standard way to create Arduino code that talk to the UnoJoy firmware in order to make native USB game controllers.

Functions:

```
setupUnoJoy()
getBlankDataForController()
setControllerData(dataForController_t dataToSet)
```

NOTE: You cannot use pins 0 or 1 if you use this code - they are used by the serial communication. Also, the setupUnoJoy() function starts the serial port at 38400, so if you're using the serial port to debug and it's not working, this may be your problem.

=== How to use this library ===

If you want, you can move this file into your Arduino/Libraries folder, then use it like a normal library. However, since you'll need to refer to the details of the dataForController_t struct in this file, I would suggest you use it by adding it to your Arduino sketch manually (in Arduino, go to Sketch->Add file...)

To use this library to make a controller, you'll need to do 3 things:

```
Call setupUnoJoy(); in the setup() block
Create and populate a dataForController_t type variable and fill it
with your data. The getBlankDataForController() function is good for
that.
Call setControllerData(yourData); where yourData is the variable from
above, somewhere in your loop(), once you're ready to push your
controller data to the system.
```

If you forget to call sendControllerData in your loop, your controller won't ever do anything

You can then debug the controller with the included Processing sketch, UnoJoyProcessingVisualizer

To turn it into an actual USB video game controller, you'll reflash the

Arduino's communication's chip using the instructions found in the 'Firmware' folder, then unplug and re-plug in the Arduino.

Details about the `dataForController_t` type are below, but in order to create and use it, you'll declare it like: `dataForController_t sexyControllerData`; and then control button presses and analog stick movement with statements like:

```
sexyControllerData.triangleOn = 1; //Marks the triangle button as pressed
sexyControllerData.squareOn = 0; //Marks the square button as unpressed
sexyControllerData.leftStickX = 90; //Analog stick values can range from
    0 - 255 */
```

```
#ifndef UNOJOY_H
#define UNOJOY_H
#include <stdint.h>
#include <util/atomic.h>
#include <Arduino.h>

// This struct is the core of the library.
// You'll create an instance of this and manipulate it,
// then use the setControllerData function to send that data out.
// Don't change this - the order of the fields is important for
// the communication between the Arduino and it's communications chip.
typedef struct dataForController_t
{
    uint8_t triangleOn : 1; // Each of these member variables
    uint8_t circleOn : 1;   // control if a button is off or on
    uint8_t squareOn : 1;   // For the buttons,
    uint8_t crossOn : 1;    // 0 is off
    uint8_t l1On : 1;       // 1 is on
    uint8_t l2On : 1;
    uint8_t l3On : 1;       // The : 1 here just tells the compiler
    uint8_t r1On : 1;       // to only have 1 bit for each variable.
    // This saves a lot of space for our type!
    uint8_t r2On : 1;
    uint8_t r3On : 1;
    uint8_t selectOn : 1;
    uint8_t startOn : 1;
    uint8_t homeOn : 1;
    uint8_t dpadLeftOn : 1;
    uint8_t dpadUpOn : 1;
    uint8_t dpadRightOn : 1;

    uint8_t dpadDownOn : 1;
    uint8_t padding : 7;    // We end with 7 bytes of padding to make sure we
                           // get our data aligned in bytes

    uint8_t leftStickX : 8; // Each of the analog stick values can range from 0
                           // to 255
    uint8_t leftStickY : 8; // 0 is fully left or up
    uint8_t rightStickX : 8; // 255 is fully right or down
    uint8_t rightStickY : 8; // 128 is centered.
    // Important - analogRead(pin) returns a 10 bit value, so if you're getting
    // strange results from analogRead, you may need to do (analogRead(pin) >> 2) to
    // get good data
} dataForController_t;

// Call setupUnoJoy in the setup block of your program.
// It sets up the hardware UnoJoy needs to work properly
void setupUnoJoy(void);

// You can also call the set
void setupUnoJoy(int);

// This sets the controller to reflect the button and
// joystick positions you input (as a dataForController_t).
// The controller will just send a zeroed (joysticks centered)
// signal until you tell it otherwise with this function.
void setControllerData(dataForController_t);
```

```
// This function gives you a quick way to get a fresh
// dataForController_t with:
//   No buttons pressed
//   Joysticks centered
// Very useful for starting each loop with a blank controller, for instance.
// It returns a dataForController_t, so you want to call it like:
//   myControllerData = getBlankDataForController();
dataForController_t getBlankDataForController(void);

//----- End of the interface code you should be using -----//
//----- Below here is the actual implementation of

// This dataForController_t is used to store
// the controller data that you want to send
// out to the controller. You shouldn't mess
// with this directly - call setControllerData instead
dataForController_t controllerDataBuffer;

// This updates the data that the controller is sending out.
// The system actually works as following:
// The UnoJoy firmware on the ATmega8u2 regularly polls the
// Arduino chip for individual bytes of a dataForController_t.
//
void setControllerData(dataForController_t controllerData) {
    // Probably unnecessary, but this guarantees that the data
    // gets copied to our buffer all at once.
    ATOMIC_BLOCK(ATOMIC_FORCEON) {
        controllerDataBuffer = controllerData;
    }
}

// serialCheckInterval governs how many ms between
// checks to the serial port for data.
// It shouldn't go above 20 or so, otherwise you might
// get unreliable data transmission to the UnoJoy firmware,
// since after it sends a request, it waits 25 ms for a response.
// If you really need to make it bigger than that, you'll have to
// adjust that timeout in the UnoJoy ATmega8u2 firmware code as well.
volatile int serialCheckInterval = 1;
// This is an internal counter variable to count ms between
// serial check times
int serialCheckCounter = 0;

// This is the setup function - it sets up the serial communication
// and the timer interrupt for actually sending the data back and forth.
void setupUnoJoy(void) {
    // First, let's zero out our controller data buffer (center the sticks)
    controllerDataBuffer = getBlankDataForController();

    // Start the serial port at the specific, low-error rate UnoJoy uses.
    // If you want to change the rate, you'll have to change it in the
    // firmware for the ATmega8u2 as well. 250,000 is actually the best rate,
    // but it's not supported on Macs, breaking the processing debugger.
    Serial.begin(38400);

    // Now set up the Timer 0 compare register A
    // so that Timer0 (used for millis() and such)
    // also fires an interrupt when it's equal to
    // 128, not just on overflow.
    // This will fire our timer interrupt almost
    // every 1 ms (1024 us to be exact).
    OCR0A = 128;
    TIMSK0 |= (1 << OCIE0A);
}

// If you really need to change the serial polling
// interval, use this function to initialize UnoJoy.
// interval is the polling frequency, in ms.
```

```

void setupUnoJoy(int interval) {
    serialCheckInterval = interval;
    setupUnoJoy();
}

// This interrupt gets called approximately once per ms.
// It counts how many ms between serial port polls,
// and if it's been long enough, polls the serial
// port to see if the UnoJoy firmware requested data.
// If it did, it transmits the appropriate data back.
ISR(TIMERO_COMP_vect) {
    serialCheckCounter++;
    if (serialCheckCounter >= serialCheckInterval) {
        serialCheckCounter = 0;
        // If there is incoming data stored in the Arduino serial buffer
        while (Serial.available() > 0) {
            //pinMode(13, OUTPUT);
            //digitalWrite(13, HIGH);
            // Get incoming byte from the ATmega8u2
            byte inByte = Serial.read();
            // That number tells us which byte of the dataForController_t struct
            // to send out.
            Serial.write(((uint8_t*)&controllerDataBuffer)[inByte]);
            //digitalWrite(13, LOW);
        }
    }
}

// Returns a zeroed out (joysticks centered)
// dataForController_t variable
dataForController_t getBlankDataForController(void) {
    // Create a dataForController_t
    dataForController_t controllerData;
    // Make the buttons zero
    controllerData.triangleOn = 0;
    controllerData.circleOn = 0;
    controllerData.squareOn = 0;
    controllerData.crossOn = 0;
    controllerData.l1On = 0;
    controllerData.l2On = 0;
    controllerData.l3On = 0;
    controllerData.r1On = 0;
    controllerData.r2On = 0;
    controllerData.r3On = 0;
    controllerData.dpadLeftOn = 0;
    controllerData.dpadUpOn = 0;
    controllerData.dpadRightOn = 0;
    controllerData.dpadDownOn = 0;
    controllerData.selectOn = 0;
    controllerData.startOn = 0;
    controllerData.homeOn = 0;
    //Set the sticks to 128 - centered
    controllerData.leftStickX = 128;
    controllerData.leftStickY = 128;
    controllerData.rightStickX = 128;
    controllerData.rightStickY = 128;
    // And return the data!
    return controllerData;
}

#endif

```

Tras cargar el sketch, vamos a la carpeta del proyecto de UnoJoy descargado y ejecutamos esto: `“UnoJoyWin > UnoJoyProcessingVisualizer > UnoJoyProcessingVisualizer.exe”`. El programa se muestra tal que así:

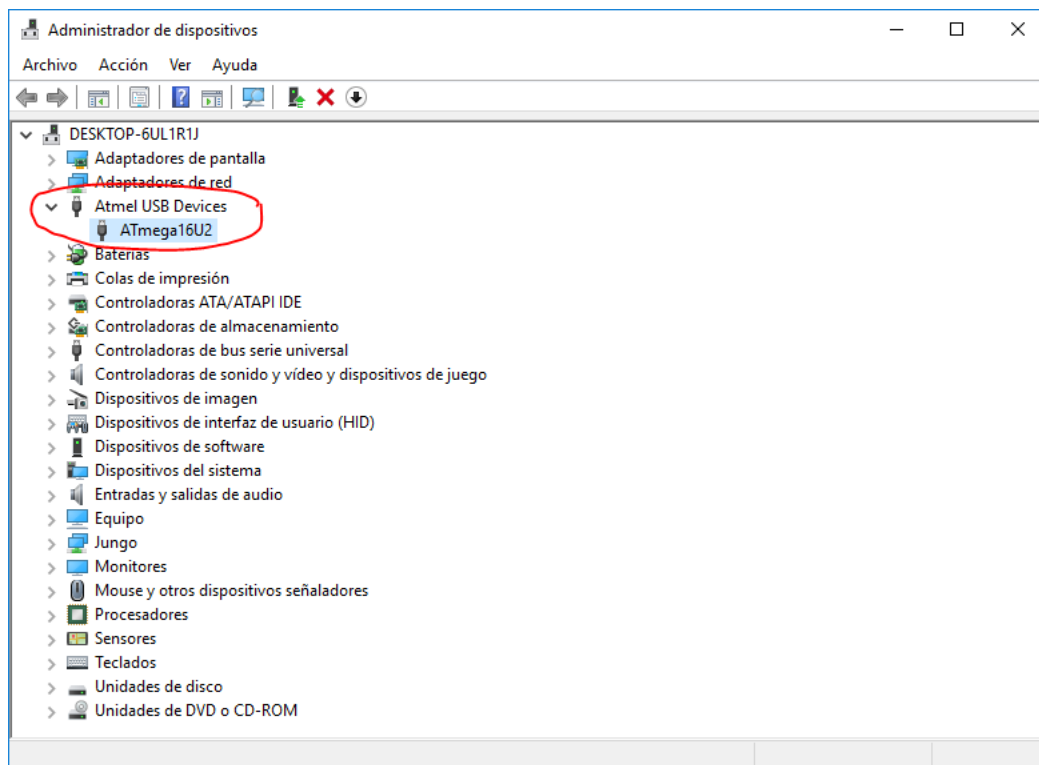


Comprobamos que al mover el joystick, se mueve el joystick izquierdo del mando de la imagen. De la misma forma, los botones de la imagen reaccionan cuando pulsamos los nuestros. Por temas de diseño con respecto al software donde emularemos un juego, los botones con forma de triángulo y círculo corresponden al mismo botón, pues serán la L y la R de una **GBASP** (Game Boy Advance SP), que suelen tener la misma función en muchos juegos.

El siguiente paso, tras comprobar que la configuración que le queremos dar al mando es la correcta, es ponerlo en modo DFU. Para ello, quitamos nuestro shield de Arduino y realizamos el siguiente corto durante un segundo; luego lo quitamos y ya estará en modo DFU:



Acto seguido, en el administrador de dispositivos, Arduino aparece de esta forma:



Tras ello instalamos los drivers de UnoJoy ejecutando el *“.bat”* *“UnoJoyWin > UnoJoyDriverInstaller.bat”*. Cuando se instalen, ejecutamos *“UnoJoyWin > TurnIntoAJoystick.bat”*. Y aparece la siguiente ventana:

```
ATMEL FLIP Command Line Interpreter
Abracadabra!
Attempting to re-flash for an Arduino Uno R1/R2
C:\Users\ramon\Desktop\UnoJoyWin\ATmega8u2Code\HexFiles>batchisp -device at90usb82 -hardware usb -operation erase f memory flash blankcheck loadbuffer "UnoJoy.hex" program verify start reset 1024
Running batchisp 1.2.5 on Wed Jan 25 17:13:57 2017

AT90USB82 - USB - USB/DFU

Device selection..... PASS
Hardware selection..... PASS
Opening port..... PASS
AtlibUsbDfu: 3EB 2FF7 no device present.
P111: Could not open USB device.
ISP done.
Trying to re-flash for an Arduino Uno R3...
C:\Users\ramon\Desktop\UnoJoyWin\ATmega8u2Code\HexFiles>batchisp -device atmega16u2 -hardware usb -operation erase f memory flash blankcheck loadbuffer "UnoJoy.hex" program verify start reset 1024
Running batchisp 1.2.5 on Wed Jan 25 17:13:57 2017

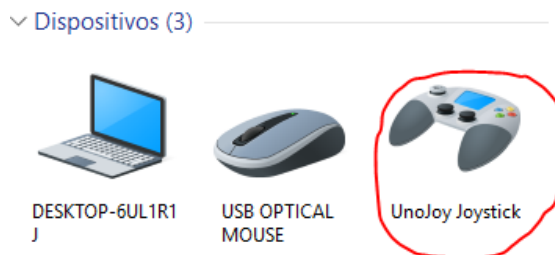
ATMEGA16U2 - USB - USB/DFU

Device selection..... PASS
Hardware selection..... PASS
Opening port..... PASS
Reading Bootloader version..... PASS 1.2.0
Erasing..... PASS
Selecting FLASH..... PASS
Blank checking..... PASS 0x00000 0x02fff
Parsing HEX file..... PASS UnoJoy.hex
Programming memory..... PASS 0x00000 0x00ad9
Verifying memory..... PASS 0x00000 0x00ad9
Starting Application..... PASS RESET 1024

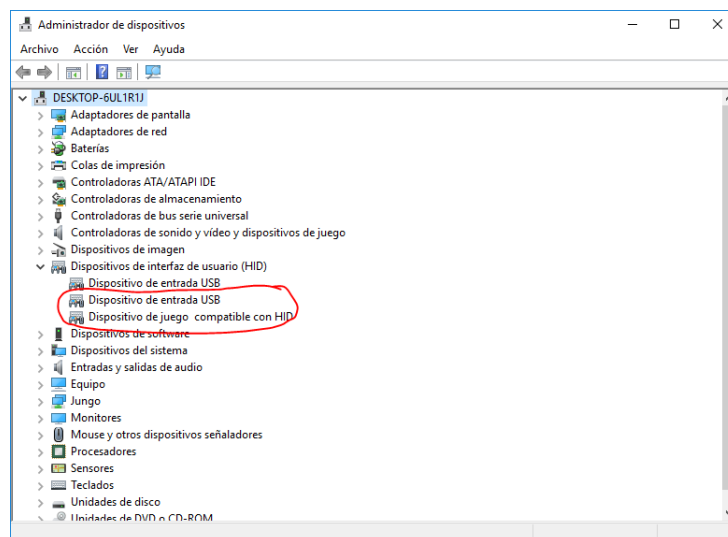
Summary: Total 11 Passed 11 Failed 0
Now, you need to unplug the Arduino and plug it back in,
but it will show up as a joystick! Press any key to exit....
```

Si nos fijamos en los comandos, lo que este *“.bat”* hace es identificar el dispositivo y borrar su memoria para cargar el firmware *“UnoJoy.hex”*. Una vez hecho esto, presionamos cualquier tecla para salir de esta ventana. Desconectamos Arduino, lo volvemos a conectar y nos vamos a *“Inicio > Panel de control > Dispositivos e impresoras”*.

Allí aparece el nuevo dispositivo:



El administrador de dispositivos también ha cambiado:

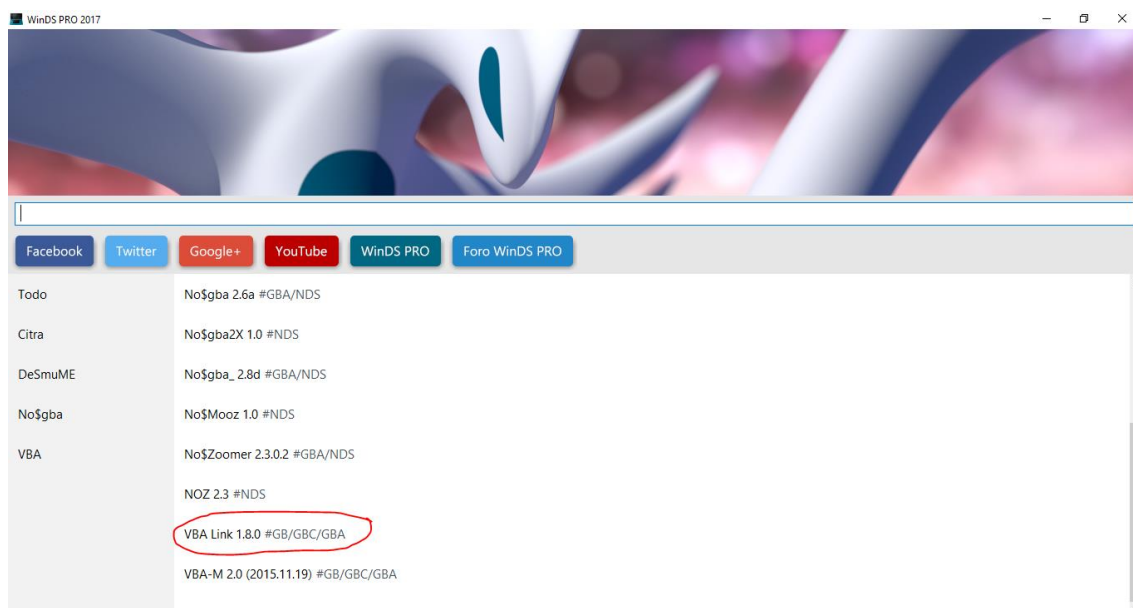


Así que, tenemos la placa configurada como mando. Solo falta probarla en un emulador. Revertir el proceso es tan sencillo como volver a repetir los pasos desde que lo ponemos en modo DFU (incluido) y cambiar el segundo “.bat” por “UnoJoyWin > TurnIntoAnArduino.bat”. Si lo hacemos correctamente, cargaría el “Arduino-usbserial-uno.hex” en lugar del anterior.

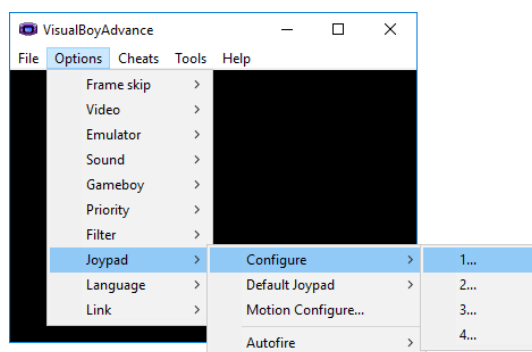
➤ Emulador GBA

Tenemos casi todo configurado para echarnos una partida. Solo nos falta un juego. En mi caso he decidido recurrir a una consola que disfruté tiempo atrás y en cuyo diseño me he basado para darle forma a la placa. El emulador de la GBA puede simular juegos de la Nintendo Game Boy Advance SP. La configuración es muy sencilla.

Inicialmente, descargamos el software del emulador desde [aquí](#). Lo instalamos con normalidad y al acabar dicha instalación, ejecutamos el programa y hacemos doble click sobre **VBA Link**:

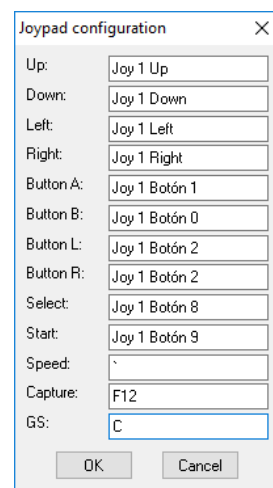


Los demás emuladores suelen funcionar bien, pero este, en particular, tiene muy buen funcionamiento. Se nos abre la ventana del emulador y pulsamos como se muestra en esta imagen:



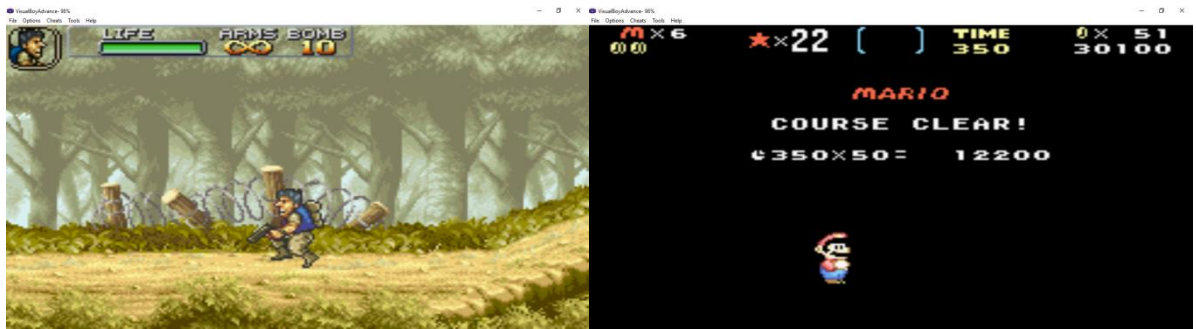
Esto da lugar a una nueva ventana, la última de configuración:

Para configurarlo, basta con hacer click con el mouse sobre el cuadro al lado de “Up:”. Acto seguido, cogemos nuestra placa insertada en Arduino y pulsamos el joystick arriba. El resto de la configuración es similar. Disponemos de cinco pulsadores, aunque destinamos seis botones a estos como se explicó anteriormente. El resto de controles son para acelerar el emulador o tomar captura de pantalla del mismo.



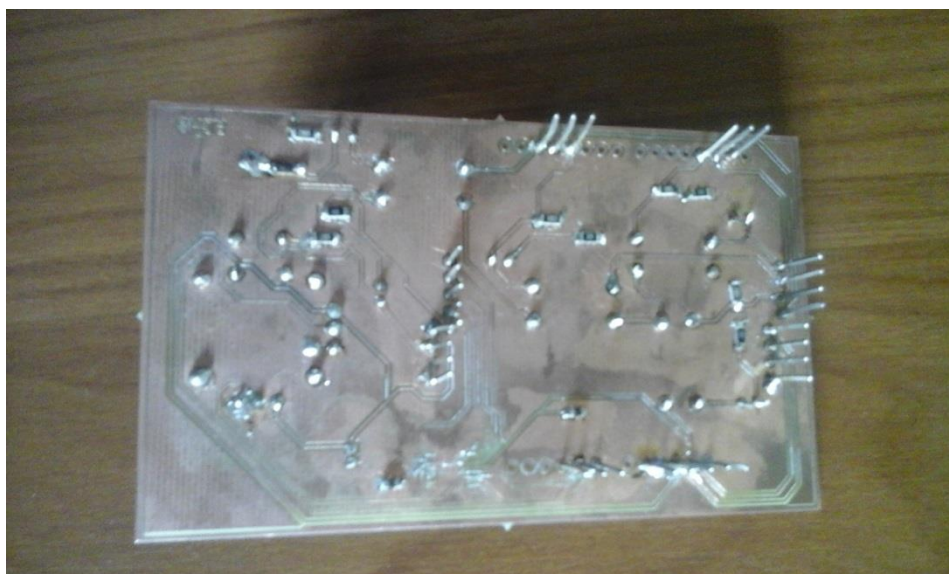
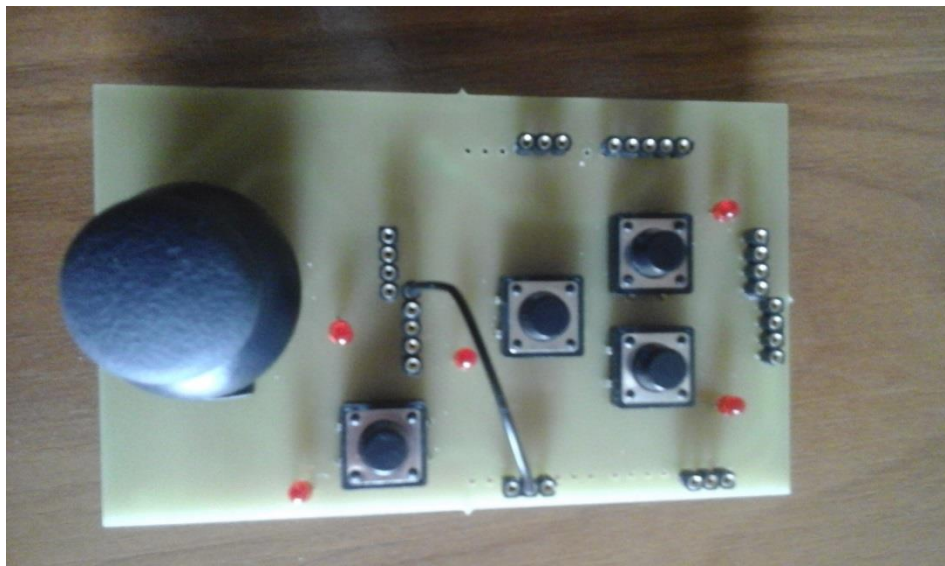
El último paso antes de echarse una partida es descargar una ROM de internet de esta consola. Hay multitud de páginas y no es difícil (basta con poner el título del juego en cuestión y acompañarlo de “ROM” al final en el buscador de Google). Una vez descargada la ROM, desde el menú del emulador pulsamos

“File > Open...” y navegamos hasta la carpeta donde se encuentra el ROM descargado. Tras cargarlo podemos echarnos una partida con nuestro mando:



Trabajo final y test Eurocircuits

La placa ha quedado finalmente con este aspecto tras el diseño por Eagle y el ensamblado final:



Se incluye, en las páginas siguientes el test realizado por Eurocircuits para esta placa. Eurocircuits ofrece un servicio de evaluación para una placa como la nuestra. Tras crearnos una cuenta en su [página web](#) e iniciar sesión con la misma, pulsamos, en el margen izquierdo sobre “Calculate and Order”.

El siguiente paso es darle a “Analyse your data” y cargar un “.rar” que contenga nuestro esquema y board.



Tras acabar la inspección, Eurocircuits nos facilita un archivo pdf que resume todas estas características:

PCB Visualizer® v1.4-25-170125

PCB Configurator PCB Checker

DRC - DFM information

DRC information DFM information

| Layer | Required | Measured |
|---------------------------------------|----------|----------|
| Bottom copper | 0.200 mm | 0.203 mm |
| Outer layer annular ring (OAR) | | |
| Bottom copper | 0.200 mm | 0.210 mm |
| Smallest final hole | | |
| Non Plated Through Hole (NPTH) | 0.60 mm | 0.80 mm |

PCB Visualizer® v1.4-25-170125

PCB Configurator PCB Checker

DRC - DFM information

DRC information DFM information

| Layer | Values |
|---|------------|
| Plating | |
| Bottom copper | 0.91 |
| Solderpaste surface | |
| Bottom solderpaste | 83.17 mm² |
| Not-connected soldermask-free pads - Potential fiducials | |
| Bottom copper | 6 |
| Copper free of soldermask | |
| Bottom copper | 73.06% |
| Copper surface | |
| Top copper | 0.0000 dm² |
| Bottom copper | 0.4383 dm² |



QUOTATION

We thank you for your price request and are pleased to present you the following quotation today

19 December 2016

Administrative details

Your references

| | | | |
|------------|------------|--------------------|---|
| Offer nr. | B0893389 | | |
| Service | NAKEDproto | Purchase reference | - |
| Board name | PCB2-LDH | Project reference | - |
| | | Article number | - |

Invoicing & delivery details

Invoice to:

ramonchg495@gmail.com
Ramón Chávez
Spain
+34 (00000) 00000000000000000000
ramonchg495@gmail.com

Delivered to:

Ramón Chávez
Ramón Chávez
Spain
+34 (00000)
00000000000000000000
ramonchg495@gmail.com

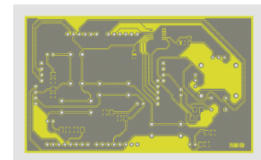
PCB Visualizer

PCB images

Top view :

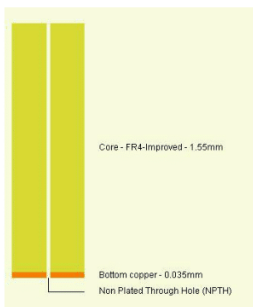


Bottom view:



Buildup & Mechanical plan

Board buildup :



Technology & options

Board definition

| | | | |
|----------------------------|----------|-----------------|---------|
| Number of layers | 1 | Delivery format | No |
| PCB width (X) | 100.0 mm | PCB height (Y) | 60.0 mm |
| eC-registration compatible | No | | |

Board definition

| | | | |
|--------------------|----------------------|-------------------|----|
| Top soldermask | No | Bottom soldermask | No |
| Top legend | No | Bottom legend | No |
| Surface finish | Any lead free finish | | |
| Bare board testing | No | | |

Board technology

| | | | |
|--|----------|-------------------------|-----------|
| Pattern class | 4 | Drill class | Drill A |
| Outer layer trackwidth (OL-TW) | 0.250 mm | Hole density | <1000/dm2 |
| Outer layer isolation distance (OL-TT-TP-PP) | 0.200 mm | Holes <= may be reduced | 0.45 mm |
| Outer layer annular ring (OAR) | 0.200 mm | | |

Material definition

| | | | |
|-------------------------|---------|----------------------------|------------------|
| Board thickness | 1.55 mm | Board buildup | Standard buildup |
| Base material | FR4IMP | Material Tg | 145-150°C |
| Outer layer copper foil | 35µm | Inner layer copper foil | 0 |
| Extra PTH runs | 0 | Extra press cycles | 0 |
| Reversed buildup | No | Inner layer core thickness | Standard |

Advanced options

| | | | |
|-------------------------|----|--------------------------------|----|
| Copper up to board edge | No | Plated holes on the board edge | No |
| Specific tolerances | No | Specific marking | No |
| Press-fit holes | No | Depth routing | No |
| Round-edge plating | No | Chamfered mechanical holes | No |

Pricing

Printed circuits

| Basket nr. | Delivery term | Quantity | Unit price | Transport price | Transport mode | Total price | VAT | Gross |
|------------|----------------|----------|------------|-----------------|----------------|-------------|---------|-----------|
| B0893389 | 7 Working days | 1 | 27.00 EUR | 3.68 EUR | Express | 30.68 EUR | 21.00 % | 37.12 EUR |

Payment terms & conditions

The payment term is 30 days from invoice date.

This quotation is valid for 30 days. All our deliveries are according to our general terms and conditions of delivery. These are available on the website , and agreed upon between us during the registration procedure. All above mentioned prices are an indication on the basis of the information at our disposal on the moment of quotation. These prices may be reviewed at the moment of order on the basis of the final documentation and conditions. The final quantity to be delivered can vary up to 5% of the ordered quantity. Delivery terms start counting upon receipt of the complete documentation and firm order.

EUROCIRCUITS N.V.
Antwerpsesteenweg 66
2800 MECHELEN
Belgium

www.eurocircuits.com

Phone: +3215281630
Fax: +32 15 28 16 31
E-mail: euro@eurocircuits.com

Conclusiones

Hemos conocido el desarrollo completo de un proceso de fabricación de PCB. El proceso ha sido bueno, ya que se han presentado problemas menores, como el hecho de que tirásemos alguna pista mal en el diseño o se nos olvidara hacerlo, o que algún componente se resistiese a la hora de realizar la soldadura.

Estos problemas nos han dado experiencia para saber adaptarnos a ellos y resolverlos trazando la pista o recurriendo a cables externos y a algo de maña. La realización de una PCB es algo complejo: hay que tener en cuenta que nuestro diseño no tenía la cantidad de elementos que tiene una PCB de mayor calibre, que además suelen ser multicapas y tienen mucha más densidad de componentes que la nuestra.

A pesar de ello, se ha resuelto bien la práctica. Me ha gustado mucho aprender todo lo que conlleva elaborar un sistema prácticamente desde cero. Me gustaría haber conseguido que la placa se comportase como dual al 100%, sin embargo, el ADC no me ha dado opción a ello y lo he tenido que retirar. De cualquier forma, he conseguido el propósito principal, que era hacer que el sistema funcionase perfectamente, consiguiendo utilizarlo como mando de control.