Taller de Proyecto I

Informe Final

Arcade Stick implementado en EDU-CIAA

Carrera: Ingeniería en Computación

Facultad de Ingeniería, Universidad Nacional de La Plata

CALDERÓN Sergio Leandro

BLANCO Valentín Nicolás

BONIFACIO Lucas Gabriel

13 de febrero de 2023

Índice

1 - Introducción

Los videojuegos son un desarrollo tecnológico acorde a nuestros tiempos; se basan en principios de competencia y están directamente relacionados con el uso de estrategia, agilidad y concentración. Los videojuegos pueden ser utilizados como herramientas educativas, en la medida en que su uso no difiere mucho de lo que se hace o podría hacerse con las nuevas tecnologías de la información.

En la mayoría de videojuegos de PC, la conexión de un controlador (joystick) por lo general es opcional, a diferencia de las videoconsolas. Sin embargo, su utilización mejora la experiencia e inmersión del jugador. Nuestra infancia, así como la de la mayoría de los jóvenes adultos de hoy en día, abarcó un periodo de jugar arcades, por lo que resulta interesante estudiar cómo era la realización de uno en menor escala.

De acuerdo a lo mencionado anteriormente, se plantea un Arcade Stick (joystick tipo arcade). Si bien existen muchos modelos en el mercado, éstos suelen ser fabricados solo por grandes compañías como Sega (con el Astro City), Nintendo (junto a Hori lanzando el Fighting Stick), entre otras. La motivación de este proyecto es la realización de un sistema que resulte económico, accesible y funcional, en comparación a los ya existentes de costes generalmente elevados y baja disponibilidad.

Fig. 1.1: modelo Arcade Stick vendido por Sega.

El proyecto a presentar requiere una comunicación con el sistema donde se empleará, en este caso una PC, en principio bidireccional vía USB. A partir del pulsado de botones y movimiento de una palanca, se podrá jugar videojuegos que se controlen mediante flechas direccionales (o WASD) y hasta 6 teclas adicionales.

2 – Objetivos

2.1 – Objetivos primarios

El objetivo primario de este proyecto es el diseño e implementación de un *arcade stick* funcional para computadoras. El stick tendrá una carcasa plástica y estará compuesto por 6 pulsadores NA, organizados en 2 filas, y una palanca o joystick que permita movimientos en 8 direcciones (resolución 45°). El sistema tendrá como finalidad ser un prototipo de baja latencia.

El proyecto se encuentra dividido en los siguientes 4 módulos:

- Diseño y ubicación de los componentes del arcade stick.
- Calibración de la conversión de las señales analógicas.
- Implementación del código de lectura de pulsadores y joystick.
- Implementación de la comunicación con la PC.

A continuación, se presenta el diagrama en bloques del sistema:

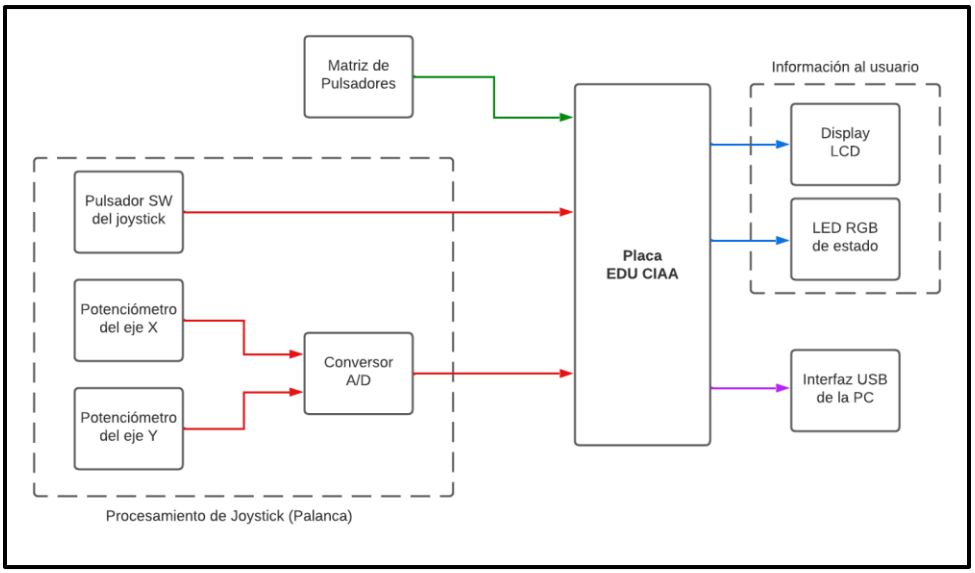


Fig. 2.1. Diagrama en bloques del sistema propuesto.

2.1.1 – Diseño y ubicación de componentes

Se conocen las dimensiones de la placa EDU-CIAA-NXP basada en LPC4337, de 137 x 86 mm [2], así como también las dimensiones aproximadas del *arcade stick* versión Astro City que se toma como base: 410 x 300 mm, grosor 170 mm [3].

Se ubicarán los 6 pulsadores, el joystick, el LED de funcionamiento y la pantalla LCD en las posiciones adecuadas siguiendo el diseño original, de modo que se aproveche el espacio disponible de manera óptima y resulte cómodo para el usuario.

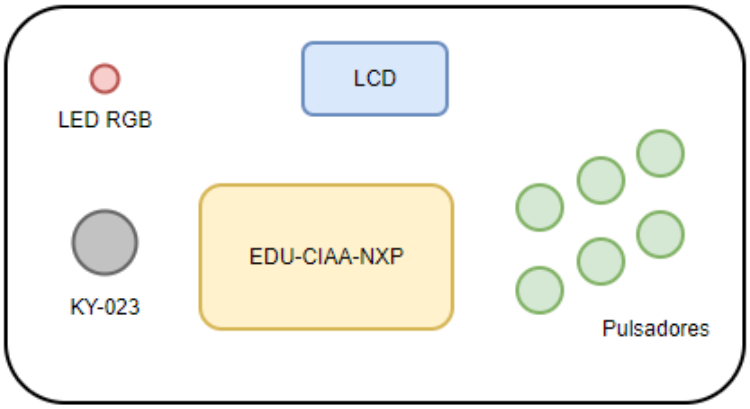


Fig. 2.2. Bosquejo de una posible ubicación de componentes.

2.1.2 – Calibración de conversión

La calibración buscará evitar la detección de un valor erróneo de dirección cuando el joystick se encuentre en reposo. Para ello se dará un margen o rango de valores a determinar, estableciendo un desplazamiento mínimo necesario del joystick para que se tome el movimiento como válido.

Además, se evaluarán las mediciones cuando el desplazamiento es máximo, para aplicar una corrección a los resultados de conversión en caso de ser necesario.

En la figura a continuación, se muestra el rango de valores discretos que toman las señales de posición de los ejes X e Y para una codificación 10 bits (de 0 a 1023).

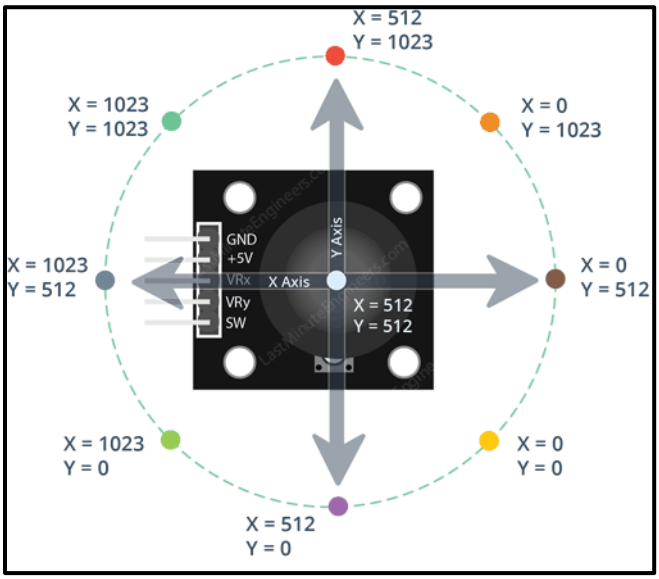


Fig. 2.3. Sistema de coordenadas de un joystick modelo KY-023. [4]

2.1.3 – Lectura de controles

En esta versión, se dispondrán de 6 pulsadores ubicados de forma matricial. Se permite que el usuario presione más de un botón a la vez, por ende, se implementará un algoritmo de lectura capaz de identificar y retornar todos los botones que se encuentren presionados, con un tiempo de procesamiento acorde al resto de tareas.

Por otra parte, para la lectura del joystick se iniciarán las conversiones y se leerán los resultados de manera periódica, debiendo traducir el par de coordenadas a alguna dirección admitida por el sistema, con previa calibración según 2.1.2. El joystick también posee un pulsador SW [5], que se considerará en la lectura de pulsadores descrita.

2.1.4 – Comunicación con la PC

Al conectar el arcade stick a una computadora, la misma proveerá de energía al sistema con una tensión de alimentación vía USB. Luego, el stick se comunicará siguiendo un protocolo a determinar para enviar uno o más códigos que representen la combinación de pulsadores presionados y dirección actual del joystick.

Se buscará en primera instancia que el microcontrolador actúe como un *game controller* nativo, similar al comportamiento de placas basadas en ATmega32u4 [6]. En su defecto, se enviarán caracteres que representen a alguna combinación vía UART, para que luego en un programa en la PC simule la presión de la tecla en el teclado.

El correcto funcionamiento de esta comunicación será informado a través de mensajes en la pantalla LCD y un color determinado en el LED RGB de estado.

2.2 – Objetivos secundarios

Se prevee agregar como futuras mejoras la posibilidad de añadir conectividad Bluetooth al joystick, permitiendo la jugabilidad mediante conexión cableada o inalámbrica. En este último caso, la alimentación sería sustituida por baterías.

También la existencia de un menú de configuración accesible mediante un botón específico del joystick para que el usuario pueda alterar el funcionamiento por defecto, con ayuda visual del LCD que muestre mensajes sobre el cambio en desarrollo.

Por otra parte, la visualización de la dirección actual de la palanca del joystick por la pantalla LCD de la placa, de modo que se pueda detectar el origen de una falla.

3 – Requerimientos

3.1 – Funcionales de información al usuario

I. Encender LED rojo cuando el sistema es iniciado.
II. Parpadear LED azul cuando la conexión con PC está en proceso.
III. Mantener encendido LED verde mientras la conexión con PC esté activa.
IV. Mostrar “Conectando…” en LCD mientras la conexión a PC esté en proceso.
V. Mostrar “Listo para jugar” en LCD cuando la conexión a PC esté activa.
VI. Garantizar que un único color del LED RGB esté encendido a la vez.
VII. Garantizar que la impresión de mensajes en LCD sea no-bloqueante.
VIII. Establecer una prioridad menor a las tareas de actualización de LED y LCD.

3.2 – Funcionales de controles

I. Establecer una frecuencia de lectura de los controles de 50 Hz mínimo.
II. Definir un rango de valores de “reposo” para los ejes del joystick.
III. Enviar dirección “UP” a la PC mientras el joystick está hacia adelante.
IV. Enviar dirección “DOWN” a la PC mientras el joystick está hacia atrás.
V. Enviar dirección “LEFT” a la PC mientras el joystick está hacia la izquierda.
VI. Enviar dirección “RIGHT” a la PC mientras el joystick está hacia la derecha.
VII. Enviar las 2 direcciones correctas a PC al desplazar el joystick en diagonal.
VIII. Enviar la dirección del joystick en cada lectura, aunque no haya cambiado.
IX. Detectar todos los pulsadores presionados en un único barrido.
X. Producir acción asociada en la PC al presionar 1 vez un botón del stick.
XI. Producir acciones combinadas en PC al presionar varios botones del stick.
XII. No duplicar una acción asociada a un botón mientras esté presionado.
XIII. Considerar una solución al efecto de rebote de los pulsadores.
XIV. Latencia óptima entre el pulsado de un botón y la recepción en PC.

3.3 – No funcionales

I. Utilización de 1x placa EDU-CIAA-NXP (controlador NXP LPC 4337).
II. Compatibilidad con el sistema operativo Windows, como mínimo.
III. Continuación del funcionamiento del sistema ante un error eventual.
IV. Ubicación del joystick a la izquierda y matriz de pulsadores a la derecha.
V. Finalización del proyecto en tiempo y forma para su muestra en diciembre 2022.

3.4 – Hardware a utilizar

I. Joystick modelo KY-023 (XY Axis).
II. 6x pulsadores de 6x6 mm, normalmente abiertos.
III. Display LCD de 16x2.
IV. LED RGB de 5 mm, de 4 pines, ánodo común.

4 – Diseño del hardware

En base al diagrama en bloques de la figura 2.1, en este apartado se explican los componentes que conforman cada uno y la interconexión entre los mismos.

4.1 – Bloque Joystick

Está compuesto por el sensor KY-023 y el conversor analógico-digital que se dispone en la EDU-CIAA. El sensor está compuesto internamente por 2 potenciómetros de 10 kΩ, utilizados para conocer la posición del joystick en cada eje, y un pulsador SW en el centro [7]. El rango de temperatura permitido para su correcto funcionamiento es de 0°C a 70°C. En la siguiente figura se muestran los terminales que posee:

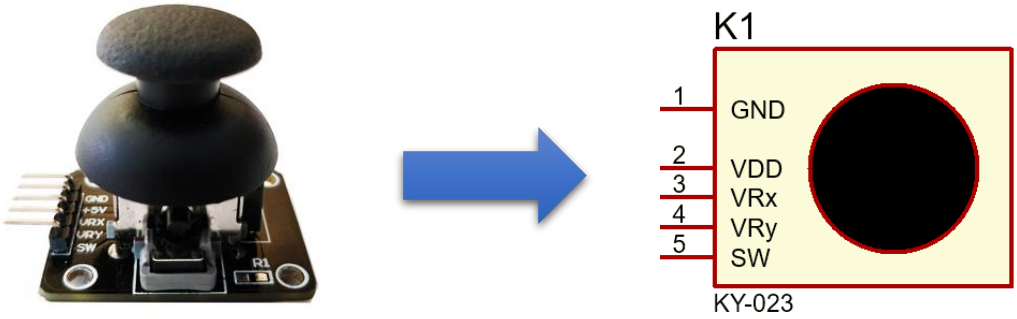


Fig. 4.1. Modelado externo del componente KY-023.

Los terminales GND y VDD corresponden a tierra y alimentación. El rango de alimentación que soporta es de 3.3V a 5V, en este proyecto se utilizará 3.3V.

Los terminales VRx y VRy son salidas analógicas, específicamente de los potenciómetros internos de cada eje, tomando valores desde 0V hasta la tensión de alimentación (en adelante, VDD). En estado de reposo, la tensión para ambos ejes es VDD / 2, mientras que los valores mínimos y máximos se obtienen al desplazar el joystick hacia los extremos. La corriente resulta: $I_{KY023} = 2 * \frac{V_{DD}}{R} = 2 * \frac{3.3V}{10\ k\Omega} = 0.66\ mA$

VRx y VRy estarán conectados a CH3 y CH2, entradas analógicas del ADC. Éste mismo, de 10 bits, devolverá valores entre 0 y 1023 para cada eje. Por otra parte, el terminal SW es una salida digital, conectada al pulsador ubicado en el centro.

4.2 – Bloque Pulsadores

Está compuesto por 6 pulsadores cuadrados de 6x6 mm. Se decidió que cada pulsador estará conectado a un pin de propósito general de la EDU-CIAA, en lugar de utilizar una distribución de “teclado matricial” de 2 filas y 3 columnas (5 pines).

Luego, cada pulsador, en el otro extremo, estará conectado a tierra, como se observa en la hoja n°03 del circuito esquemático (ver anexo).

4.3 – Bloque Información

El estado general del sistema se informa a través de un display LCD 16x2, provisto por la cátedra, y un LED RGB 5mm de 4 pines tipo ánodo común.

El display consta de los terminales listados a continuación [8]:

- VSS, VDD y VEE: tierra, alimentación +5V y regulación de contraste. Para este último se utiliza un divisor resistivo de 100K y 1K para una tensión de 4.95V.
- RS: entrada digital para indicar instrucción (nivel bajo) o dato (nivel alto).
- R/W: entrada digital para indicar lectura (nivel alto) o escritura (nivel bajo).
- E: entrada digital de la señal “Enable”, activada por flanco de bajada.
- D0-D7: líneas digitales de datos. En la configuración para su manejo con 4 líneas de datos y 2 de control, únicamente se utilizan los terminales D4 a D7.

En este proyecto, se trabajará con el LCD únicamente para mostrar mensajes al usuario, por lo que el terminal R/W está conectado a tierra, y se utiliza la configuración de 4 líneas de datos para disminuir la cantidad de conexiones con la placa EDU-CIAA.

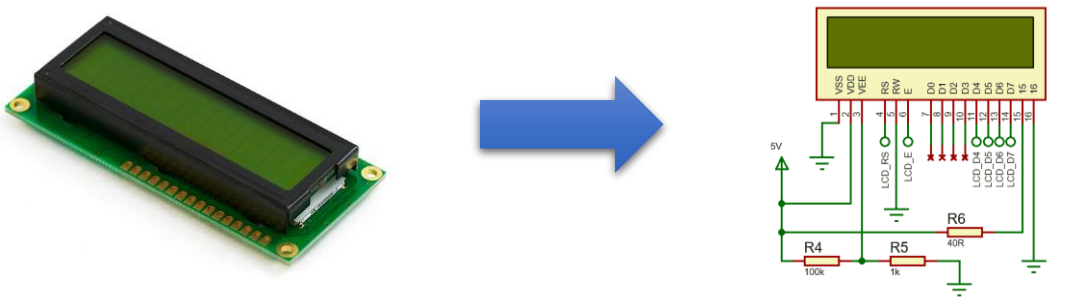


Fig. 4.2. Modelado externo del componente LCD 16x2.

Por otra parte, el LED RGB dispone de un terminal para encender cada color primario que lo compone de manera independiente, y un terminal de ánodo común. Se decidió usar 5V como tensión de alimentación, con resistencias intermedias de 220 Ω en todos los casos para limitar la corriente a un valor cercano a 10 mA en cada diodo.

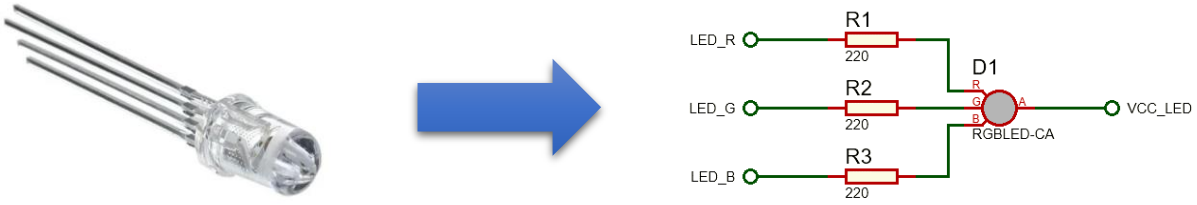


Fig. 4.3. Modelado interno del componente LED RGB.

Se recalcularon las intensidades máximas de corriente para cada diodo, considerando que la tensión V_{OL} de un pin es cero, menor al máximo (0.4V) [9].

$$I_{LED} = \frac{V_{CC} - V_{LED} - V_{OL}}{R} = \begin{cases} I_{rojo} = \dfrac{5V - 2.0V}{220\ \Omega} = 13.63\ mA \\ I_{verde} = \dfrac{5V - 2.81V}{220\ \Omega} = 9.95\ mA \\ I_{azul} = \dfrac{5V - 2.87V}{220\ \Omega} = 9.68\ mA \end{cases}$$

Al momento de la adquisición del LED, no se contó con la especificación del color asociado a cada uno de los 3 pines de menor longitud. Por tal motivo, esto se debió determinar experimentalmente y se obtuvo el resultado mostrado en la Figura 4.4.

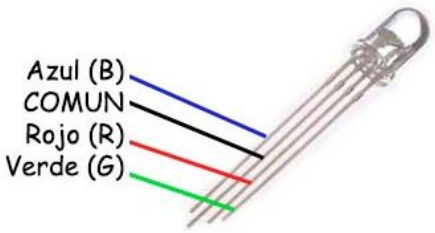


Fig. 4.4. Funcionalidad asociada a cada pin del LED adquirido.

4.4 – Bloque USB

Para la comunicación con la PC, se utiliza exclusivamente el puerto denominado USB-OTG, el cual sirve, a su vez, de alimentación a la EDU-CIAA.

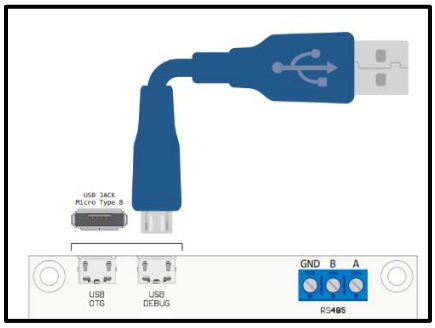


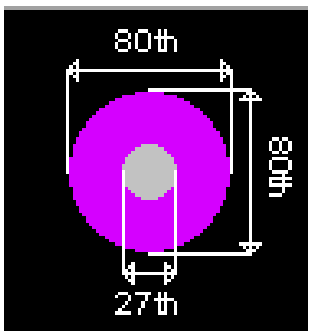
Fig. 4.5 Fichas USB disponibles para su uso en EDU-CIAA.

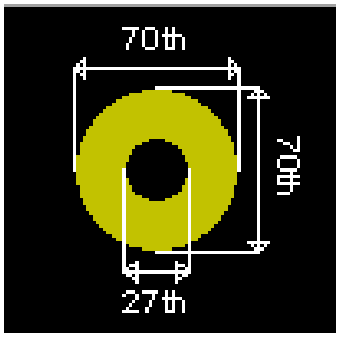
4.5 - Diseño del PCB

4.5.1 – Consideraciones generales

En base a las dimensiones de la EDU-CIAA, se ubicaron las tiras de pines J1 y J2 de acuerdo a los conectores P1 y P2, respectivamente. El LCD es el componente de mayor tamaño, por lo que su posicionamiento fue decidido en primer lugar.

En todos los componentes se reemplazaron los pads predeterminados por otros de diámetro externo 80th (2.032 mm) y diámetro interno 27th (0.6858 mm). Este cambio se realizó para respetar el diámetro de la broca a utilizar y poder centrar la misma correctamente. Las pistas utilizadas son de ancho T30 (0.76 mm). Las vías poseen un diámetro externo 70th (1.778 mm) e interno 27th (0.6858 mm).





Figs. 4.6 y 4.7. Configuración utilizada para pads (izquierda) y vías (derecha).

También se consideraron los requerimientos mínimos de separación entre pistas (Trace-Trace 0.7mm) y entre pad y pista (Pad-Trace 30th = 0.76mm).

4.5.2 – Tiras de pines (Pin Headers)

Para J1 y J2, se usó el componente base 10073456-049LF de Proteus. Como se puede observar en el circuito esquemático (anexo 9.3), no se utilizan todos los pines de los conectores P1 y P2 de la EDU-CIAA, por ende, se eliminaron todos aquellos pines sin utilidad para el presente proyecto, a excepción de los pines 1, 2, 39 y 40, para garantizar una sujeción estable del poncho diseñado. A continuación, se muestran los nuevos componentes posteriores a estas modificaciones.

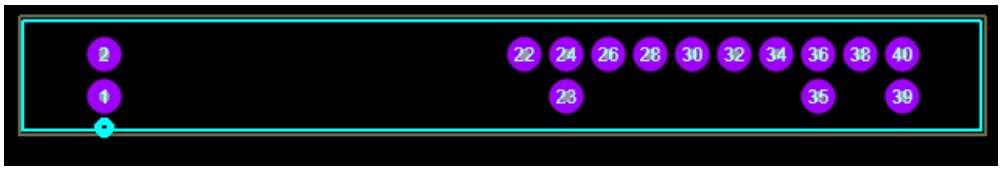


Fig. 4.8. Tira de pines J1, que se conecta a P1 presente en EDU-CIAA.

Fig. 4.9. Tira de pines J2, que se conecta a P2 presente en EDU-CIAA.

4.5.3 – Diseño de componentes

Como la biblioteca de Proteus no cuenta con varios de los diseños de *footprint* de componentes utilizados en el PCB, éstos fueron diseñados e implementados.

Uno de los footprint desarrollados es el correspondiente al LED RGB, que cuenta con 4 pines (R, VCC, G y B), cuya distancia real entre ellos es de 1mm, la cual es menor a la requerida para el PCB. Considerando esto último, se utilizó una distancia entre pads de 2.5mm, implicando que posteriormente en la colocación del LED RGB, dichos pines deberán ser “doblados”. A continuación, se presenta el diseño final de este componente.

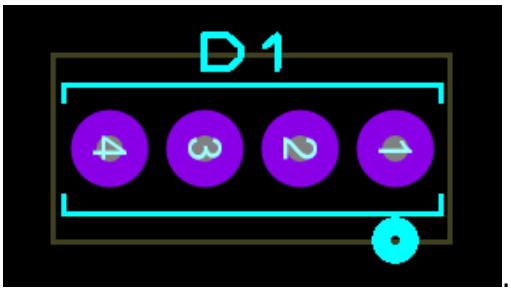
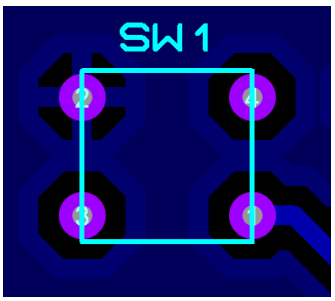
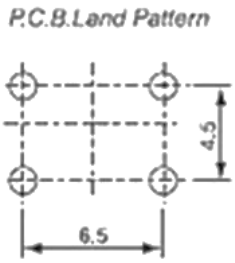


Fig. 4.10. Footprint del componente LED RGB

También se realizó un diseño propio del footprint de los botones y del KY-023 (joystick). Los pulsadores cuentan con 4 “patas” posicionadas de tal manera que forman un cuadrado, uno de los pines (número 1) debe estar conectado a la entrada asociada de la tira de pines y el correspondiente al extremo opuesto (número 2) a GND [12].





Figs. 4.11 y 4.12. Footprint del pulsador utilizado en el proyecto.

Por otra parte, el KY-023 cuenta con 5 pines, que corresponden a GND, VCC, VRx, VRy, SW, ubicados de manera contigua y separados 0.1” (2.54 mm) entre sí.

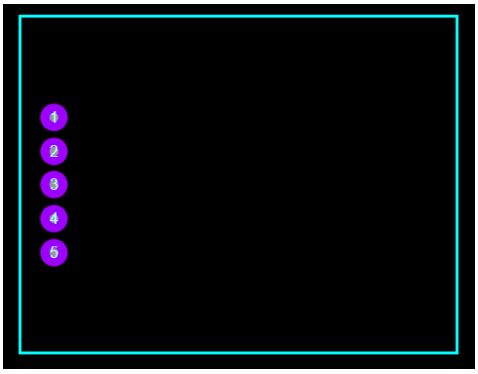


Fig. 4.13. Footprint simplificado del joystick KY-023.

5 – Diseño de software

El firmware del Arcade Stick está basado en la biblioteca sAPI, que proporciona una capa de abstracción respecto del hardware, como se observa a continuación:

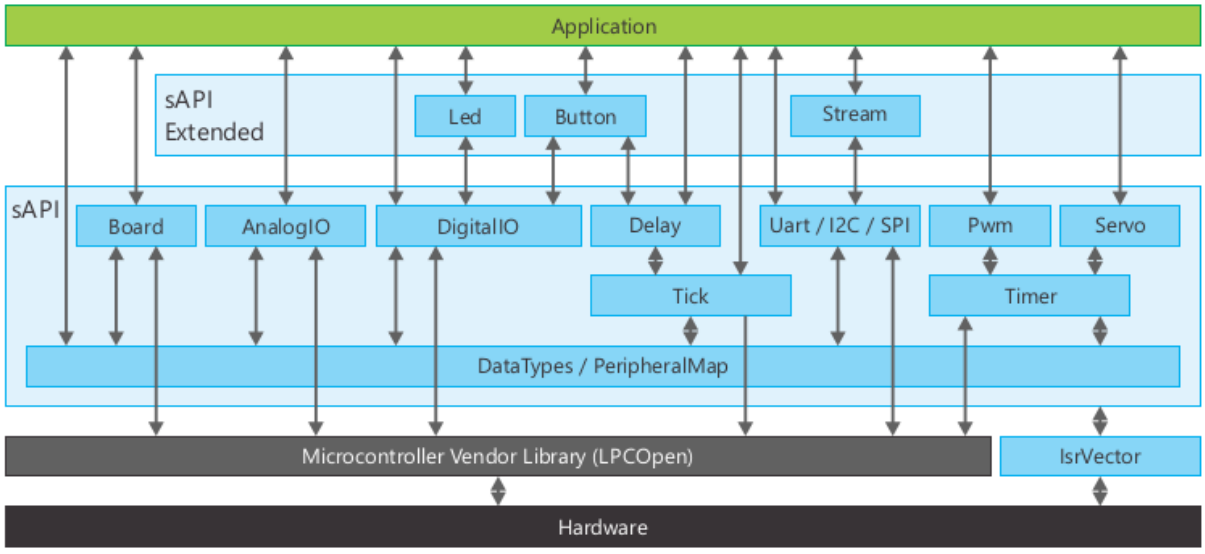


Fig. 5.1. Capas del firmware proporcionado por sAPI.

Para cada periférico descrito en el apartado 4, existe un módulo controlador:

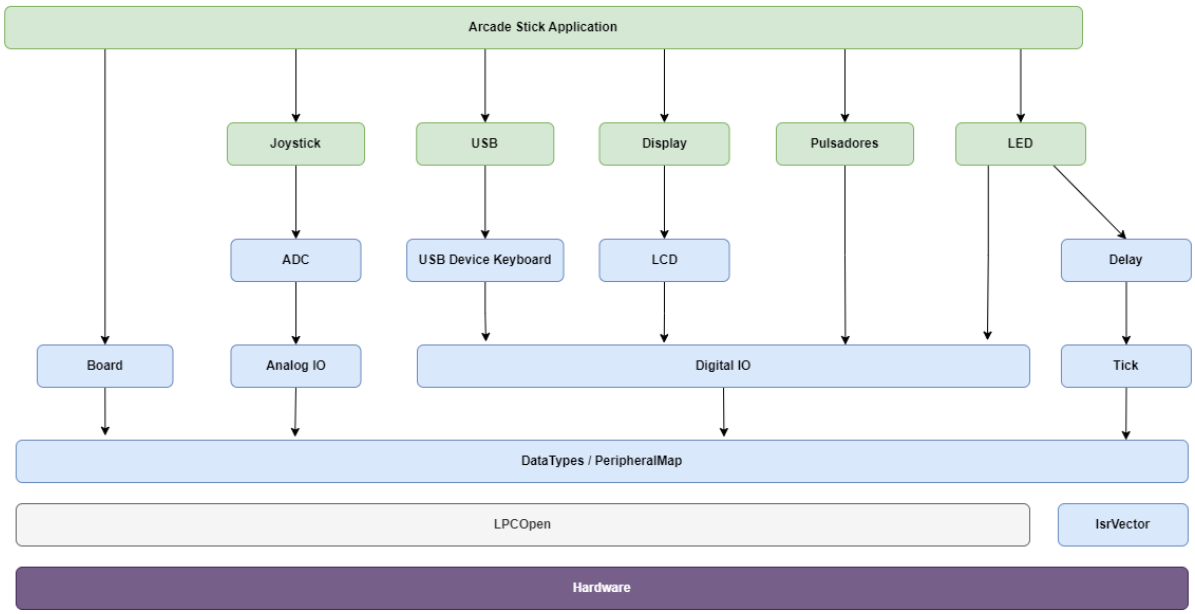


Fig. 5.2. Capas del firmware de Arcade Stick (en verde) basado en sAPI (en azul).

Se diseñó una arquitectura *super-loop*, basada en el funcionamiento de una máquina de estado finito temporizada, explicada en la sección 5.6. El período principal es 20 ms, dado el requerimiento de una frecuencia de lectura de controles de 50 Hz mínimo. En las secciones siguientes se detalla la implementación de cada módulo.

5.1 – Comunicación USB

En el desarrollo del software se utilizaron dos drivers para que el sistema Arcade Stick actúe como *device* respecto a la PC. Ambos se explican a continuación.

5.1.1 – Versión 1: Driver Teclado

En el primer avance, se logró la comunicación con la PC a través de la biblioteca sAPI *"usbd_keyboard.h"*, basada a su vez en *"usbd_rom_api.h"*, de LPCOpen. También se utiliza *"sapi_usb_device.h"*, que contiene una función de inicialización general para mouse, teclado y otros dispositivos USB.

De los 2 archivos sAPI indicados, se utilizaron las siguientes funciones:

- usbDeviceInit: establece la configuración inicial del periférico especificado.
- usbDeviceKeyboardCheckKeysCallbackSet: recibe un puntero a la función que indica cuáles teclas deben "presionarse" y enviar a PC según las condiciones.
- usbDeviceKeyboardPress: se marca la tecla indicada por parámetro como presionada. Existe un enumerativo con las teclas mapeadas en ASCII.
- usbDeviceKeyboardTasks: si el transmisor se encuentra disponible, invoca a la función de *callback* para enviar las teclas indicadas a la PC. Por tal motivo, dicha función debe invocar *usbDeviceKeyboardPress* para marcar las teclas.

Las teclas enviadas son "W", "A", "S", "D" y "Enter" a partir del estado del joystick, como se explica en la sección 5.2. El descriptor tipo Input definido para teclados define la siguiente estructura de reportes, donde "M" es un modificador (Ctrl, Alt, Shift, etc.)

Tabla 5.1. Estructura de reportes para el driver Keyboard.

	b7	b6	b5	b4	b3	b2	b1	b0
Byte 0	M	M	M	M	M	M	M	M
Byte 1	*Reservado*							
Byte 2	Keycode 1							
Byte 3	Keycode 2							
Byte 4	Keycode 3							
Byte 5	Keycode 4							
Byte 6	Keycode 5							
Byte 7	Keycode 6							

5.1.2 – Versión 2: Driver Gamepad

Debido a que la implementación del Arcade Stick como un teclado USB requiere definir una tecla para cada acción, y dicha tecla no puede ser modificada durante ejecución, se propuso cambiar el driver a uno tipo Gamepad.

Esta modificación requirió modificar el descriptor de hardware utilizado para el protocolo USB. A continuación, se muestra la nueva estructura definida por el descriptor.

Tabla 5.2. Estructura de reportes para el driver Gamepad.

	b7	b6	b5	b4	b3	b2	b1	b0
Byte 0	Valor del Eje X (entre -127 y 127)							
Byte 1	Valor del Eje Y (entre -127 y 127)							
Byte 2	*Reservado (normalmente para Eje Z)*							
Byte 3	Rotación del Eje X *(no utilizado)*							
Byte 4	Rotación del Eje Y *(no utilizado)*							
Byte 5	*Reservado (normalmente para rotación del Eje Z)*							
Byte 6	Valor del Hat (Switch del Joystick)							
Byte 7	*Res.*	*Res.*	S6	S5	S4	S3	S2	S1

Como se puede observar en la Tabla 5.2, el rango de valores aceptados para los ejes X e Y es [-127, 127], por lo que el número obtenido en la lectura del Joystick (entre 0 y 1023) es dividido por 4 y desplazado de modo que el valor de reposo sea el 0.

En el caso del Hat, únicamente se escribe el número 0 o 1, según el Switch se encuentre en reposo o presionado, respectivamente. En el último byte del reporte, se tiene un bit dedicado a cada pulsador, pudiendo utilizar hasta 8 en simultáneo.

A diferencia del teclado, no existe ninguna biblioteca o implementación previa del driver Gamepad usando la EDU-CIAA, por lo que se debió crear todo el firmware desde cero. Se desarrollaron entonces las siguientes funciones públicas.

- usbDeviceGamepadInit: inicializa el driver en base al descriptor definido.
- usbDeviceGamepadCheckCallbackSet: establece función periódica a invocar.
- usbDeviceGamepadPress: recibe estado de pulsadores y escribe en reporte.
- usbDeviceGamepadHat: recibe estado actual del Hat y escribe en reporte.
- usbDeviceGamepadMove: recibe eje y su valor para escribirlo en reporte.
- usbDeviceGamepadTasks: envía el reporte a PC si el transmisor está libre.

Resulta importante aclarar que los archivos contenedores de dichas funciones y los descriptores asociados se encuentran bajo un directorio ***driver***, separado del código principal del sistema, localizado en la carpeta app. Esta diferenciación se realizó porque el código de driver es extenso y realiza invocaciones a funciones de LPCOpen.

Para una mejor legibilidad y uso, se desarrolló un módulo “gamepad_api” de alto nivel en la carpeta app, que provee las siguientes funciones públicas.

- **USB_Init**: realiza la configuración del driver mediante el llamado a las respectivas *Init* y *CheckCallbackSet*. En esta API se encuentra implementada la función de *callback*, que actualiza los bits del reporte según el estado de los controles.
- **USB_Attempt**: invoca a la función *usbDeviceGamepadTasks* para intentar un envío de datos hacia la PC, devolviendo TRUE en caso de resultar exitoso.
- **USB_Update**: realiza la lectura de los controles (joystick y pulsadores) y realiza el envío de datos mediante la función que USB_Attempt.

5.2 – Lectura del Joystick

Mediante la biblioteca sAPI, se utiliza la función *adcInit* para habilitar el ADC. En esta API se lo configura para trabajar con frecuencia de muestreo $f_{ADC} = 200\,kHz$ y resolución $n = 10$ bits.

Se utiliza la función *adcRead*, que habilita el canal indicado, inicia la conversión y se bloquea hasta que finalice para retornar el valor. El tiempo de cada conversión se puede calcular como $n * T_{ADC} = \frac{n}{f_{ADC}} = \frac{10}{200\,kHz} = 50\,\mu s$, resultando $100\,\mu s$ en total.

Dado que sAPI opera con un único conversor de la EDU-CIAA, primero se realiza la lectura del eje X y posteriormente la del eje Y.

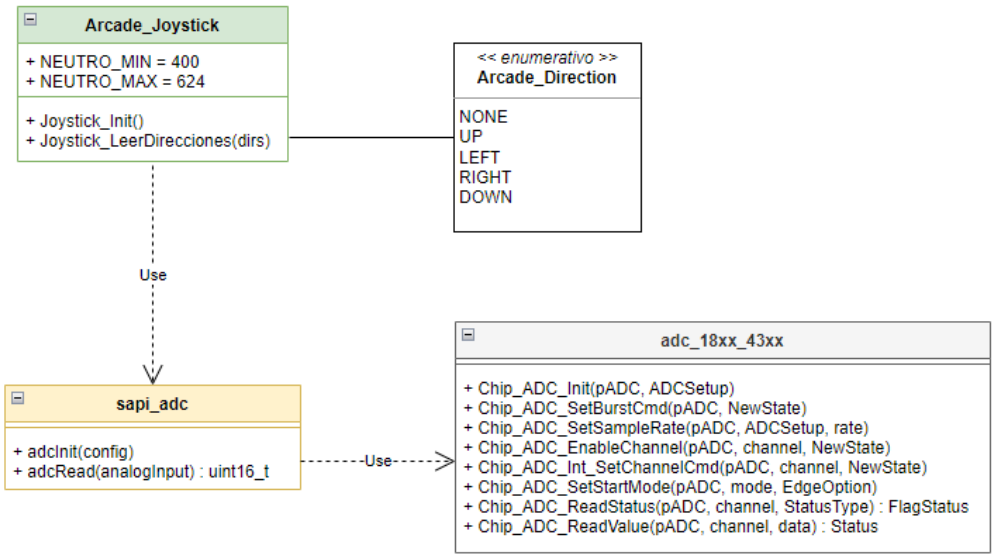


Fig. 5.3. Dependencias sAPI y LPCOpen para módulo Joystick.

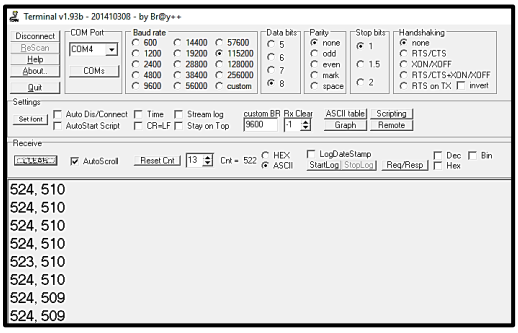
A partir de los valores leídos del conversor (entre 0 y 1023), se asigna por software una dirección principal de cada eje (horizontal y vertical) según cada valor:

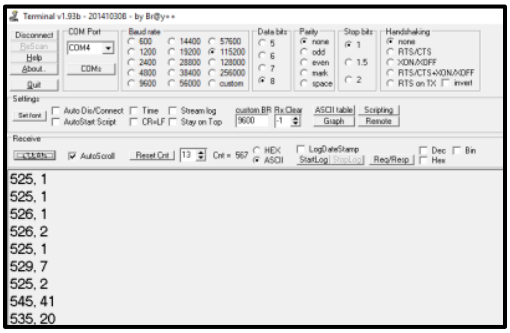
- Para el rango del eje X entre 0 y 400: dirección LEFT, mapeado por tecla “A”.
- Para el rango del eje X entre 624 y 1023: dirección RIGHT, mapeado por “D”.
- Para el rango del eje Y entre 0 y 400: dirección UP, mapeado por tecla “W”.
- Para el rango del eje Y entre 624 y 1023: dirección DOWN, mapeado por “S”.
- Para el rango de un eje entre 400 y 624: dirección NONE, sin mapear.

Como el procesamiento de cada eje es independiente, luego se envían las teclas correspondientes a cada dirección, pudiendo ser hasta 2 en el mismo envío.

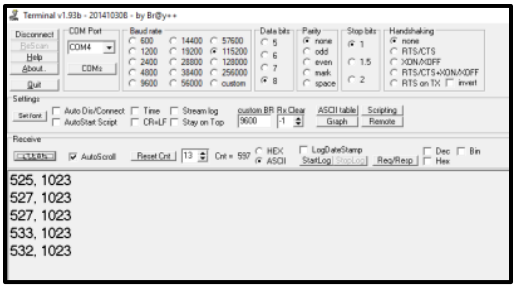
5.3 – Depuración del Joystick

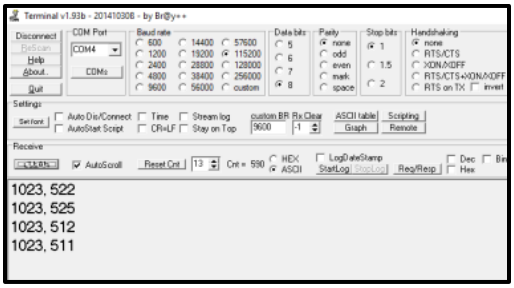
Para verificar que el funcionamiento del sistema de coordenadas, se utilizó la UART de EDU-CIAA para enviar el par de valores de los ejes. Los resultados para cada dirección se muestran a continuación. En la zona de reposo se obtuvo el par (524, 509) en lugar de (512, 512), implicando un error del 2.4%, mucho menor al 20% considerado.



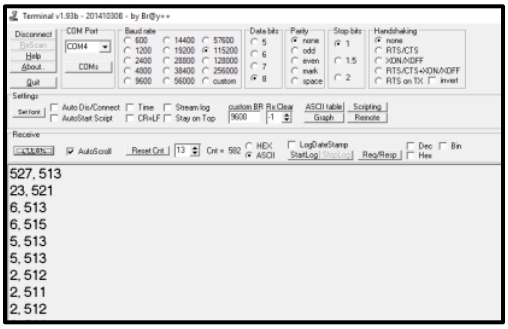


Figs. 5.4 y 5.5. Valores (x,y) para el joystick en reposo o *deadzone* // hacia arriba.





Figs. 5.6 y 5.7. Valores (x,y) para el joystick desplazado hacia abajo // hacia la derecha.



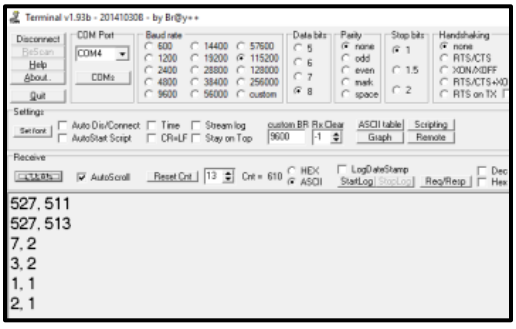


Fig. 5.8 y 5.9. Valores (x,y) para el joystick desplazado hacia la izquierda // hacia arriba e izquierda.

5.4 – Simulación de Joystick

En un primer programa, se accionan LEDs que, según su color, indican en qué posición está siendo accionado el joystick. Cabe destacar que se utilizan los LEDs que ya proporciona la placa EDU-CIAA. En estado de reposo, todos los LEDs se encuentran apagados, como se puede observar en la imagen a continuación:

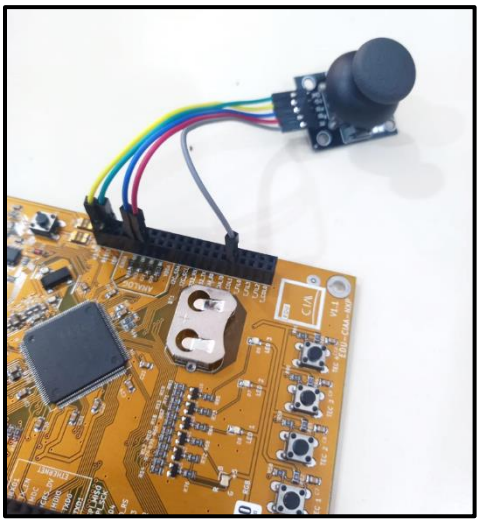
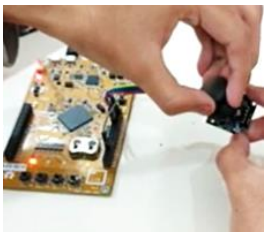


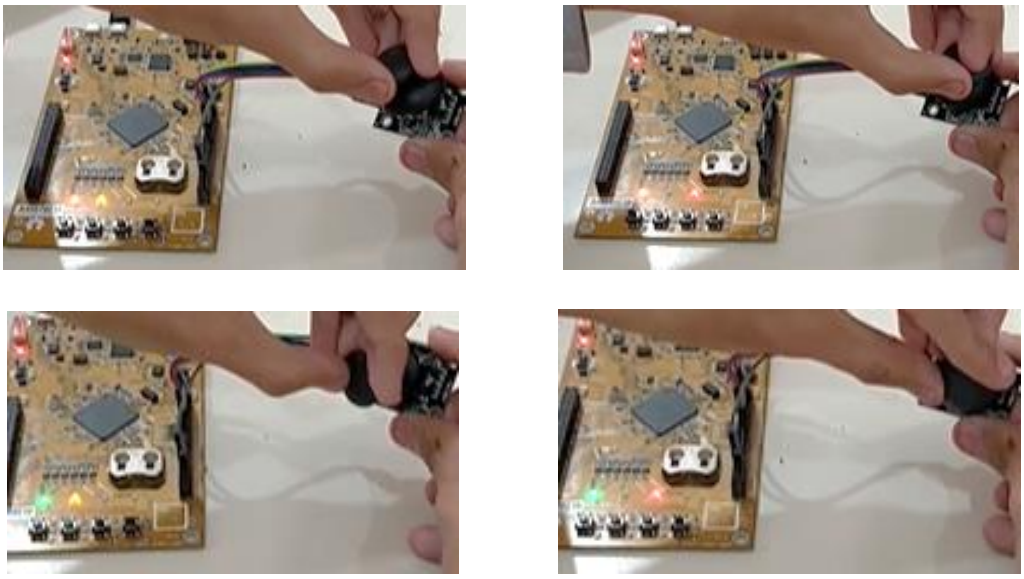
Fig. 5.10. Estado de LEDs mientras el joystick está en su *deadzone*.

Luego, se desplazó el joystick hacia cada una de las direcciones principales, como se puede observar en el siguiente video: https://drive.google.com/file/d/1VUfrMsMoO9ti-2vdj6m1foljctk7Kjif/view?usp=sharing




Figs. 5.11 a 5.14. Estado de LEDs para dirección UP, DOWN, LEFT y RIGHT.

Se prosiguió con el desplazamiento del joystick en direcciones diagonales. Link: https://drive.google.com/file/d/1b2GRXS9GWm1Hsp58RO01RIIvj6qIyMJW/view?usp=sharing



Figs. 5.15 a 5.18. Estado de LEDs para UP-LEFT, UP-RIGHT, DOWN-LEFT y DOWN-RIGHT.

Luego, se probó el funcionamiento en conjunto con la comunicación USB, enviando las direcciones indicadas en la sección 5.2 en cada ciclo de lectura. También se asignó que al accionar el pulsador SW del joystick se envíe la tecla “Enter”. Se adjunta enlace al video: https://drive.google.com/file/d/1Cax3STiZzMDgGUvNg0gO2CHOMenl-58S/view?usp=sharing

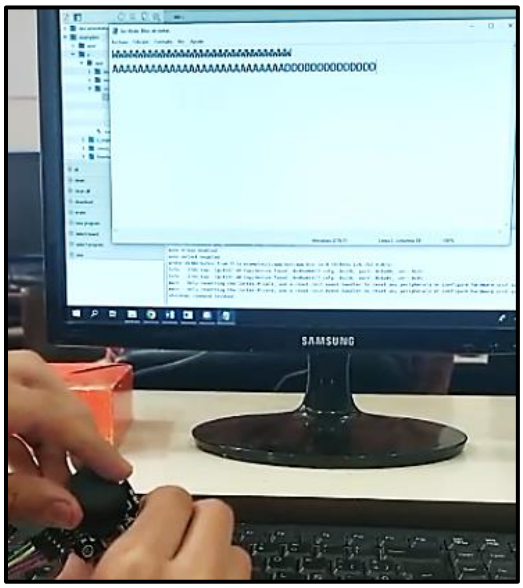


Fig. 5.19. Prueba del envío de teclas “WASD” en el bloc de notas, vía USB OTG.

5.5 – Simulación del Driver Gamepad

En el presente segundo avance, se comprobó el funcionamiento del nuevo driver mediante la utilidad *joy.cpl* del Panel de Control de Windows, accesible también desde las propiedades del dispositivo. En la pestaña “Test” se pueden verificar los pulsadores.

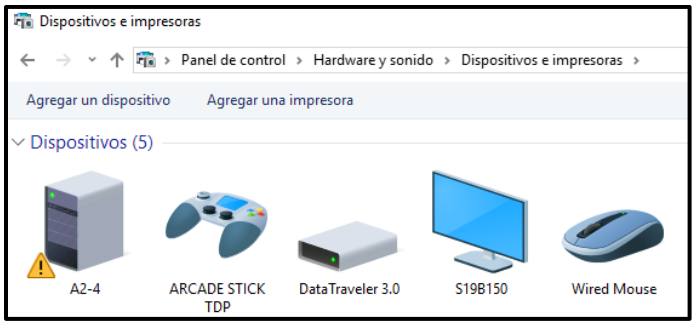


Fig. 5.20. Reconocimiento del Arcade Stick como dispositivo de juego.

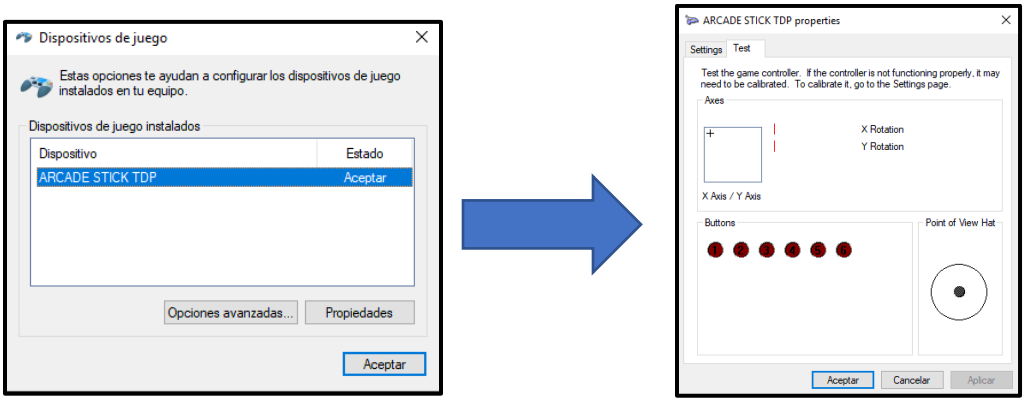


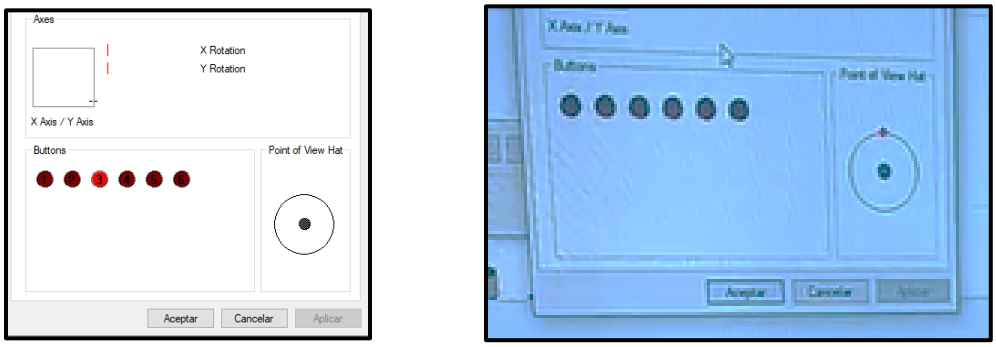
Fig. 5.21. Ventana de propiedades del Arcade Stick.

Para alternar entre la versión de simulación y de fabricación, existe un archivo “mapeoGpio.h” que define macros para cada pin utilizado según la versión. Por ejemplo, para los pulsadores se utilizaron los botones TEC1..4, presentes en la EDU-CIAA.

En primer lugar, se presionó cada pulsador de manera individual para verificar su correcta identificación en la PC. Luego se presionaron varios en simultáneo de acuerdo al requerimiento 3.2.XI, pudiéndose enviar el máximo de 6 botones activados a la vez.

Se adjunta el enlace al vídeo: https://drive.google.com/file/d/14702FVslKVK-bGdbcLft3siLKgOq1Ye-/view?usp=share_link

La siguiente prueba fue la lectura del Hat. Al presionar el switch del joystick, se muestra una indicación en la perspectiva (Point of View) del mismo. Se adjunta enlace al vídeo de dicha prueba: https://drive.google.com/file/d/142gwTWzbA0lPZr7wkZM9n-5QKv27fD6E/view?usp=share_link



Figs. 5.22 y 5.23. Recepción del pulsador 3 activado (izquierda) y SW presionado (derecha).

Por último, se utilizó la pestaña “Settings” para ingresar al menú de calibración, donde se puede obtener un mayor detalle de la coordenada actual del joystick. Se desplazó primero en las 4 direcciones principales y luego en diagonales, como se observa en el video: https://drive.google.com/file/d/142VQYYNUjrlbgYvbU_jSeJQ322L-gTxu/view?usp=share_link

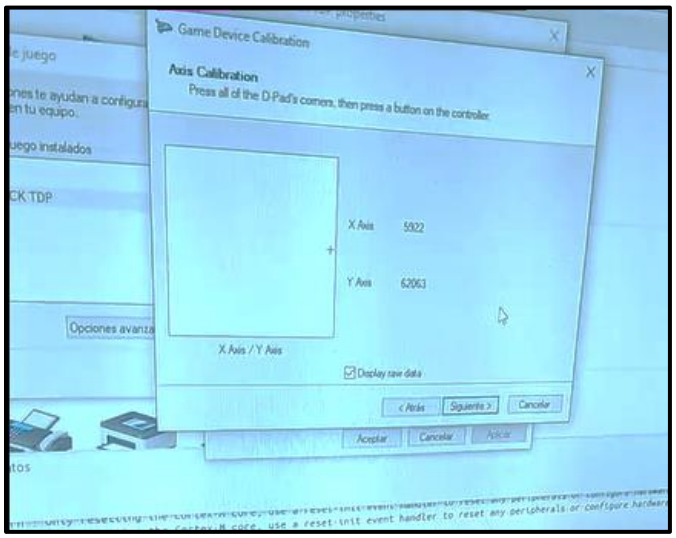


Fig. 5.24. Menú de calibración de ejes X-Y luego de desplazar el joystick hacia la derecha.

5.6 – Programa principal

5.6.1 – Inicialización

Al conectar el sistema Arcade Stick a la alimentación, la primera etapa consiste en la inicialización de los componentes, como se describe a continuación.

- **EDU-CIAA:** se realiza un llamado a la función *boardConfig*, provista por sAPI.
- **LED RGB:** se invoca a *LED_Init*, que establece los pines asociados a los colores rojo, verde y azul como salidas para su posterior manipulación.
- **Joystick XY:** se invoca la función *Joystick_Init*, que habilita el conversor analógico digital e indica el pin conectado al pulsador de Switch como entrada.
- **Pulsadores:** se inicializan mediante el llamado a *Buttons_Init*, la cual establece los 6 pines conectados a pulsadores como entradas.
- **Display LCD:** es el último componente en inicializarse, y se invoca a *Display_Init* con un retardo de 900 ms debido a su latencia. La función mencionada establece la configuración del modelo utilizado (2 filas de 16 caracteres, cada uno de tamaño 5x8 pixeles), oculta el cursor y crea caracteres customizados para el sistema.

5.6.2 – Descripción de MEF

En la siguiente etapa se da inicio a una máquina de estado finito de Moore, cuya actualización se realiza cada 20 ms (frecuencia de 50 Hz). Posee 4 estados definidos:

- **Conectando:** inicial y de ejecución única, se enciende el LED en color azul y se imprime el mensaje "*Conectando…*" en el display LCD. Luego, se realiza la inicialización e instalación del driver Gamepad mediante la función *USB_Init*, la cual retorna un valor booleano indicando si la operación fue exitosa.

- **Chequeo:** consiste en realizar intentos de envíos de datos a la PC mediante la función *USB_Attempt*. Mientras no se confirme una recepción exitosa, se persistirá en este estado hasta un máximo de 10 segundos (500 ciclos). Además, cada 500 ms (25 ciclos), se enciende o apaga el LED mediante *LED_Alternar.*

- **Listo:** luego de la primera recepción exitosa, se cambia el color del LED de azul a verde y se imprime el mensaje *"Listo para jugar"* en el display LCD. En cada actualización, se realiza la lectura de los controles y se envían los datos a la PC, mediante *USB_Update*. Además, se muestra el estado del Joystick en el LCD.

- **Fallo:** alcanzable desde cualquiera de los 3 estados mencionados anteriormente, consiste en cambiar el color del LED a rojo y mostrar un mensaje de error en la pantalla del Arcade Stick. Se puede deber a un reconocimiento fallido del dispositivo por parte de la PC (en estado Conectando) o la ausencia de confirmación luego del envío de datos (en estados Chequeo y Listo). La causa típica es la conexión del sistema mediante el puerto Debug en lugar de USB OTG, por tal motivo el mensaje mostrado es *"USB incorrecto"*.

En la Figura 5.25 se muestra el diagrama de estado explicado, donde la variable X en las transiciones indica si el éxito de la operación involucrada (0 = true, 1 = false).

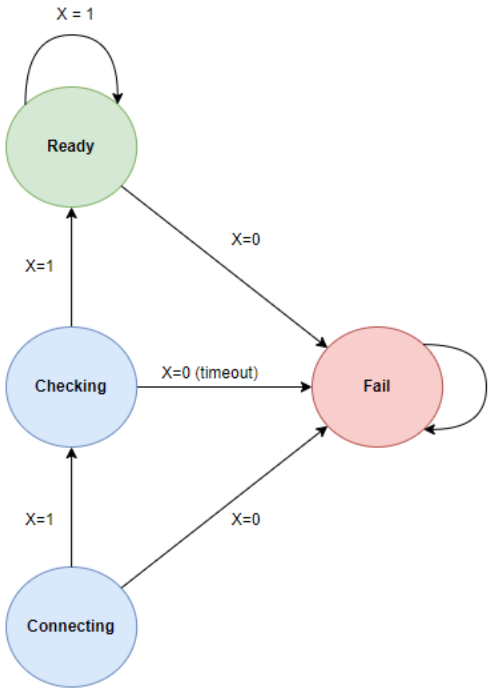


Fig. 5.25. Diagrama de estados del sistema principal Arcade Stick.

5.6.3 – Temporización de MEF

Como se mencionó en la sección anterior, la máquina de estados implementada es del tipo temporizada. El estado inicial (Conectando) se asigna mediante la función *MEF_Start*. Luego, en el bucle sin fin del programa principal, se realiza la actualización mediante la función *MEF_Update*.

Se utilizó la función *delay*, de tipo bloqueante, para el retardo de 20 ms. El motivo de esta decisión es la inexistencia de acciones que requieran una base de tiempo precisa que justifiquen el uso de interrupciones de reloj. En este proyecto, se idealiza una frecuencia de 50 Hz para la lectura de los controles, pero solo con fines prácticos para que el usuario perciba una respuesta rápida del sistema, por lo que puede ser menor.

Por otra parte, la impresión de mensajes en la pantalla del LCD no tiene un impacto significativo en la duración de la actualización, ya que el mensaje permanece constante durante un estado, pudiéndose escribir solo en el primer llamado del mismo.

En el pseudocódigo 5.1 se muestra el funcionamiento interno de una función de estado, tomando como ejemplo el estado “Listo”.

```
Asignar variable estática “impreso” como falso
Si no está impreso
        Encender LED verde
        Escribir mensaje “Listo para jugar” en LCD
        Marcar “impreso” como verdadero
Leer controles y enviar a PC
Si el envío fue exitoso
        Recuperar direcciones actuales del Joystick
        Mostrar direcciones en LCD
Sino
        Pasar a estado de Fallo
```

Pseudocódigo 5.1. Función de estado *MEF_Ready()* del estado “Listo”.

5.7 – Manejo de LCD

La inicialización del display y el envío de caracteres al mismo se realiza por medio del módulo *"display_lcd_hd44780_gpios.h"*, de sAPI. Su utilización permite tener una abstracción respecto a la escritura directa sobre los pines. Se agregó además un módulo propio **"display_api"**, que internamente invoca las siguientes funciones de sAPI:

- lcdInit: inicializa el display especificando el ancho de línea (caracteres por línea), la cantidad de líneas, el ancho de carácter y la altura de carácter.
- lcdCursorSet: establece un modo definido del cursor del LCD (ON, OFF, BLINK).
- lcdGoToXY: mueve el cursor a la celda (x,y) indicada por parámetros.
- lcdSendChar: escribe el carácter en línea de datos y activa señal Enable.
- lcdSendString: escribe los caracteres del mensaje pasado por parámetros.
- lcdCreateChar: agrega un carácter personalizado de 5x8 con un ID particular.
- lcdData: escribe el carácter personalizado indicando el ID por parámetro.
- lcdClear: limpia la pantalla del LCD.
- lcdClearAndHome: limpia la pantalla y reinicia la posición del cursor hacia (0,0).

5.7.1 – Caracteres customizados

Además de los mensajes informativos del estado en que se encuentra el sistema, en el caso de "Listo" también se muestran las direcciones accionadas del joystick. Para ello, se crearon los caracteres correspondientes a las 4 direcciones elementales, cuyas representaciones gráficas se presentan en la Figura 5.26.

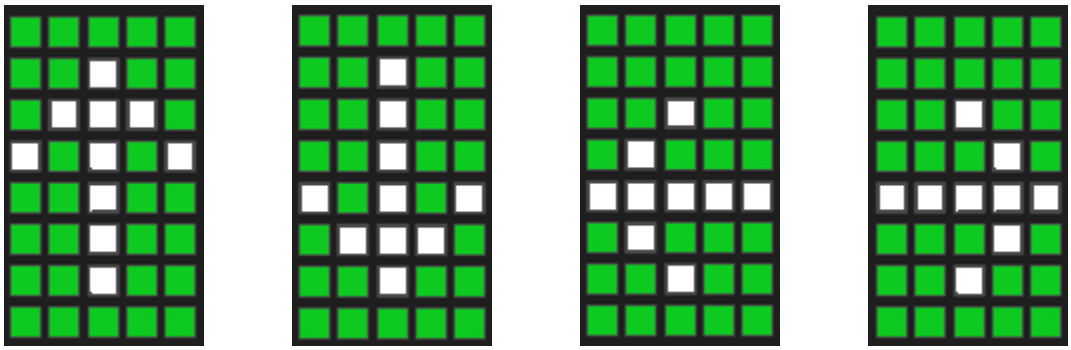
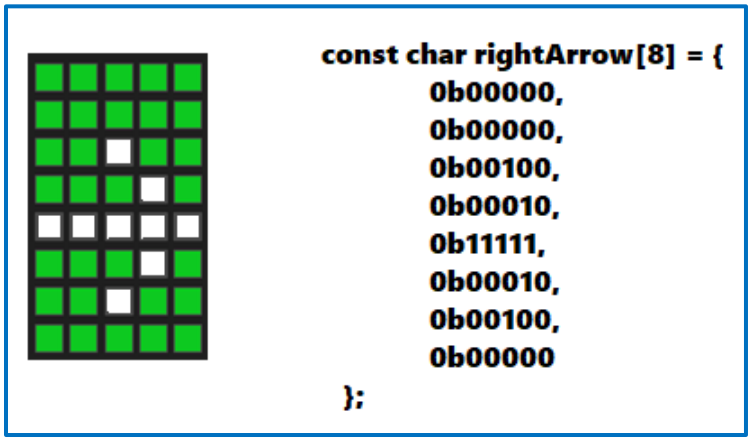


Fig. 5.26. Caracteres UP, DOWN, LEFT y RIGHT agregados al LCD.

Un carácter de 5x8 se representa en código mediante un vector de 8 bytes, donde cada los 5 bits menos significativos de cada byte representan las 5 celdas de una fila. Un bit en 1 indica que el pixel correspondiente debe encenderse, 0 en caso contrario. La Figura 5.27 ejemplifica el caso de la flecha de la dirección RIGHT.



```
const char rightArrow[8] = {
        0b00000,
        0b00000,
        0b00100,
        0b00010,
        0b11111,
        0b00010,
        0b00100,
        0b00000
};
```

Fig. 5.27. Representación gráfica y en código del carácter RIGHT.

5.7.2 – Distribución de caracteres

Se decidió que los tres mensajes de tipo indicativos del estado actual, es decir, "*Conectando…", "Listo para jugar"* y "*USB incorrecto",* se muestren en la primera línea del display LCD. Para ello, el módulo creado (display_api), provee la función *Display_WriteMessage*, que limpia la pantalla, reinicia el cursor al origen de coordenadas y escribe los caracteres del String pasado por parámetros.

Por otra parte, la segunda línea se encuentra reservada para las direcciones activas, que varían de acuerdo a la posición del joystick en cada lectura realizada. El estado de la MEF permanecerá fijo mientras no se interrumpa la comunicación con la PC, entonces la pantalla no será limpiada. Se provee una función *Display_WriteDirs*, que recibe 4 parámetros booleanos indicando el estado de cada dirección.

L	i	s	t	o		p	a	r	a		j	u	g	a	r
		←			↑			↓			→				

Fig. 5.28. Ubicación de los caracteres del estado "Listo" en el display.

Como se puede observar en la Figura 5.28, el orden de las direcciones mostradas, de izquierda a derecha, es LEFT, UP, DOWN, RIGHT. La posición de cada una es completamente fija, ubicándose la primera flecha en la posición (2,1) y las siguientes con dos celdas de separación entre ellas hasta la posición (11,1).

Si una dirección no está activa, se imprime un espacio en blanco (‘ ‘) en su lugar para borrar el posible carácter preexistente en pantalla.

5.8 – Utilización de GPIO

La lectura y escritura directa de los pines I/O de la placa EDU-CIAA se realiza para el barrido de estados de los pulsadores (S1-S6) y el manejo del LED RGB. Sin embargo, para ambos componentes se especificó un módulo que se encarga de estas tareas, de modo de abstraer al programa principal respecto a la distribución elegida de los pines.

La biblioteca sAPI utilizada para este propósito es *“sapi_gpio.h”*.

5.8.1 – Lectura de pulsadores

El módulo dedicado a los 6 botones del Arcade Stick se denomina ***buttons***. Se provee una función de inicialización para establecer cada pin asociado como entrada, y una función de barrido *Buttons_Read*, que contempla el efecto de rebote presente en los pulsadores y devuelve los estados confirmados por referencia.

Internamente, la lectura se realiza mediante la función *gpioRead* de sAPI, la cual retorna el valor 0 si el pulsador se encuentra presionado, 1 en caso contrario. Este valor es comparado con el último valor almacenado (lectura anterior a la actual) y con el valor confirmado de dicho pulsador (enviado a la PC). Se decidió que la confirmación del estado se realice si se consigue leer al menos 2 veces seguidas el mismo valor, es decir, si se cumple alguna de las siguientes condiciones:

- El valor de lectura actual coincide con el valor confirmado.
- El valor de lectura actual coincide con el valor de lectura anterior.

5.8.2 – Control del LED RGB

El módulo que dispone control directo sobre el LED se denomina de manera homónima. Provee una función de inicialización y varias funciones de control descritas más adelante. Se puede establecer si el LED es ánodo común o cátodo común a partir de un macro ANODO_COMUN, que se puede comentar si corresponde. Luego, se definen las macros ENCENDIDO y APAGADO; en el caso de ánodo común toman los valores 0 y 1, respectivamente; y los opuestos para cátodo común.

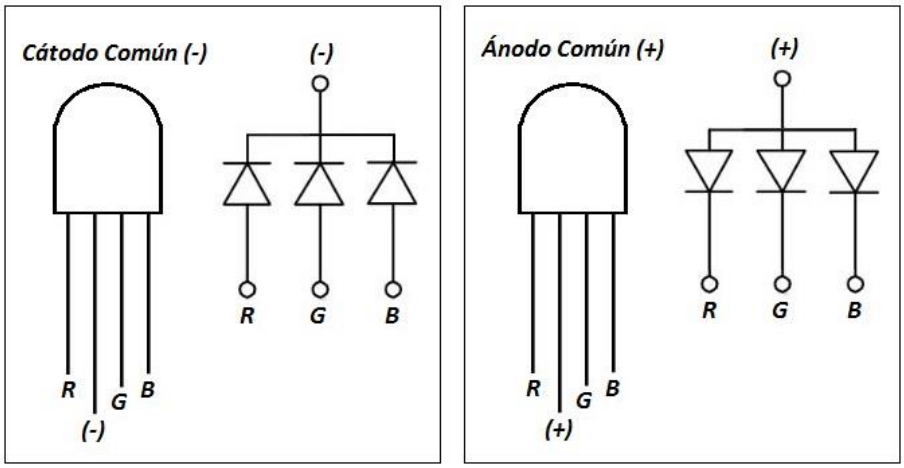


Fig. 5.29. Diferencia entre un LED RGB tipo cátodo común y ánodo común.

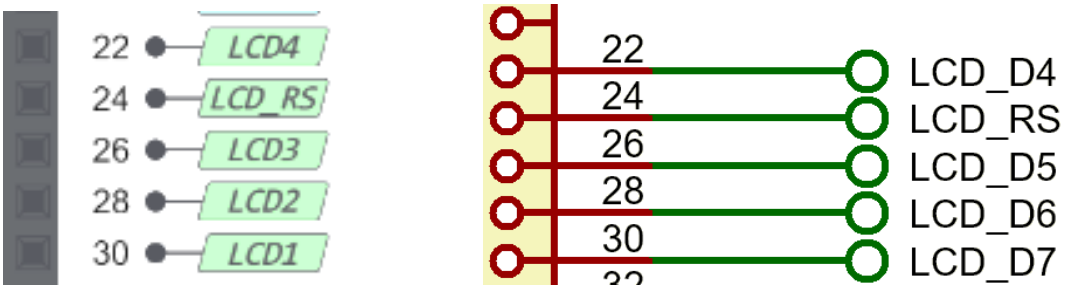
La escritura sobre un pin se realiza mediante la función *gpioWrite* de sAPI. Las funciones para controlar los colores son las siguientes:

- **LED_ApagarTodos:** escribe el valor APAGADO en los 3 pines asociados a colores de modo que no circule corriente por los mismos.
- **LED_Encender[Color]:** apaga los colores y luego escribe el valor ENCENDIDO en el pin asociado al color indicado (rojo, verde o azul únicamente).
- **LED_Alternar:** a partir de variables auxiliares, apaga o enciende el último color que se haya encendido. Recibe por parámetro un *período*, para realizar la alternancia luego de dicha cantidad de llamados a la función.

5.9 – Cambios en la biblioteca sAPI

Como se explicó en la sección 5.1, el driver de tipo keyboard se encuentra implementado y disponible para su uso en el firmware v3 de la EDU-CIAA. Sin embargo, para el agregado del driver tipo gamepad fue necesario modificar la función de inicialización (usbDeviceInit) de la biblioteca *"sapi_usb_device"*. Las demás funciones de dicho archivo tratan las interrupciones USB, que es indiferente del driver utilizado. De esta manera, ahora si se especifica la macro *HID_GAMEPAD*, se invoca la función *usbDeviceGamepadInit* implementada de manera propia en la carpeta driver.

Por otra parte, para una simplificación del PCB, se alteró el orden de los 4 pines de datos utilizados por el display. Esto implicó una actualización de las macros definidas en el archivo *"sapi_lcd.h"*. A continuación, se presenta la diferencia entre la organización provista por sAPI (a la izquierda) y la versión final fabricada (derecha).



Figs. 5.30 y 5.31. Pines asociados a las líneas de datos del LCD.

6 – Ensayos y mediciones

Una vez soldados los componentes a la placa diseñada, se ensambló la misma a la EDU-CIAA y se conectó esta última mediante USB a una computadora. Se realizaron las pruebas descritas a continuación, haciendo énfasis en cada componente.

I. **Prueba de funcionamiento del LED**: en primer lugar, se verificó que el LED RGB recibiera la alimentación correspondiente mediante la observación de un encendido total (intensidad máxima) de los colores. Se presentó una dificultad debido a un mal contacto en las soldaduras, detectada luego de que el color encendido no se correspondía con el programado. Por otra parte, se percibió que, a su vez, se había adquirido una versión distinta del componente (cátodo común en lugar de ánodo común) por lo que se aisló al led de VCC y se conectó mediante un puente el mismo a plano de tierra (GND).

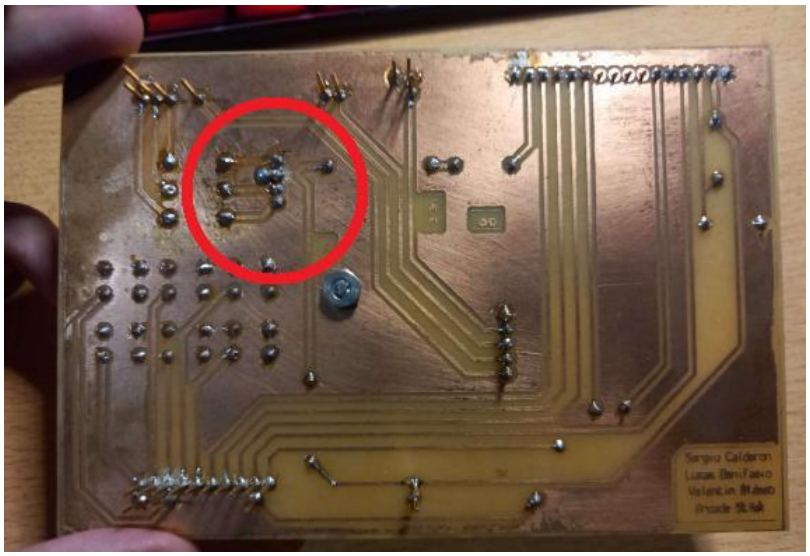


Fig. 6.1. Puente a tierra realizado para el pin de cátodo común.

II. **Prueba de funcionamiento de LCD**: para esta prueba se conectó la CIAA de dos formas, una es a través del USB DEBUG en el que se podía observar el mensaje de “USB incorrecto” en el LCD, y posteriormente mediante el USB OTG, en el que inicialmente aparece la leyenda “Conectando” y luego “Listo para jugar”.

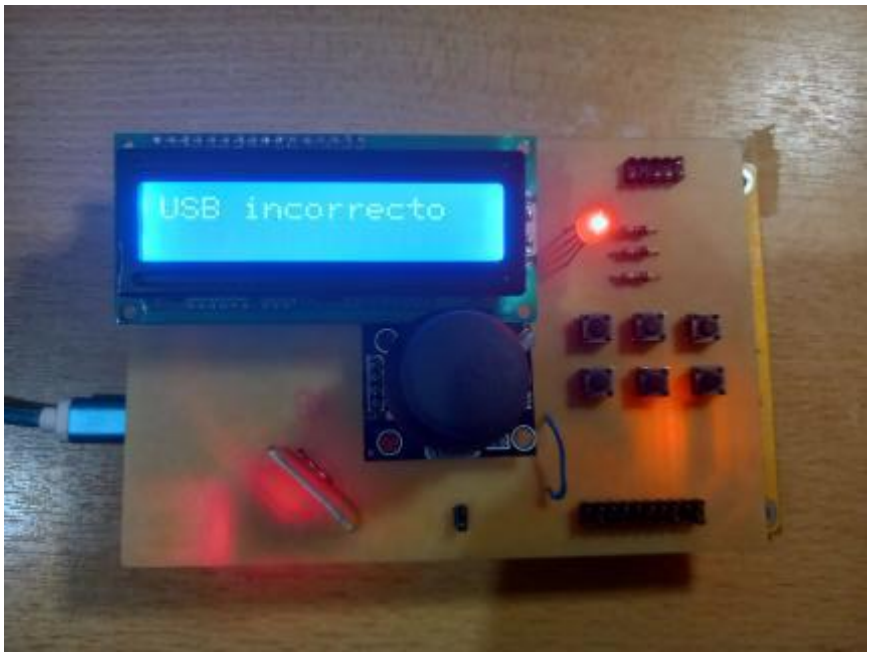


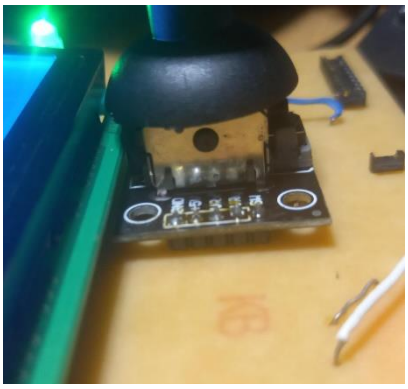
Fig. 6.2. Display encendido con el mensaje “USB incorrecto”.

Fig. 6.3. Display encendido con el mensaje “Listo para jugar”.

Se adjunta vídeo del primer caso, donde se observa la etapa de chequeo de 10 segundos: https://drive.google.com/file/d/14kBz2ThWj_ZsTj3EGm7pNMoEzWCqkzoC

III. **Prueba de funcionamiento de los botones**: la verificación de los pulsadores actualmente activos en el Arcade Stick se realizó mediante las propiedades del dispositivo provista en el Panel de Control. En el primer intento se encontró que en la PC se recibían los botones 3 y 5 como presionados a pesar de que todo el sistema estuviera en reposo. La causa, nuevamente, fueron cortocircuitos por soldadura incorrecta. Se corrigió el problema, y se comprobó que la respuesta en la computadora se percibía en un tiempo prácticamente instantáneo. Se adjunta un video: https://drive.google.com/file/d/1LIrcKb0tk0SjmlcZEFW4kHlQVSk5s6WD

IV. **Prueba de funcionamiento del Joystick**: las pruebas asociadas a este componente se realizaron detalladamente, previo a la fabricación de la placa. Los resultados fueron mostrados en la sección 5.4, y no se percibieron alteraciones en el sistema final. Durante el soldado y colocación del joystick se debió utilizar una tira de pines auxiliar para respetar la orientación (Figura 6.4) y se observó, además, que el mismo no se mantenía firme, lo cual generaba una dificultad al momento de su utilización, por lo que se optó por fijarlo a la placa mediante un tornillo, como se muestra en la Figura 6.5 a la derecha.



Figs. 6.4 y 6.5. Tira de 5 pines y tornillo incorporados a la placa.

Adicionalmente, se verificó la correcta impresión de las direcciones accionadas en el display según la posición del joystick, como puede observarse en el siguiente video adjunto: https://drive.google.com/file/d/1pRd5UcPGho8c0o2Zmc-ArLa2dec99N4i

6.1 – Funcionamiento en Windows

Posteriormente a las pruebas esenciales del sistema, se configuró el Arcade Stick para su uso principal en un emulador de videojuegos. En este caso, se utilizó el programa *Kega Fusion*, que permite ejecutar ROMs de Sega Mega Drive, entre otros. Al ingresar al menú y elegir el mando, se encuentra disponible el Arcade Stick, sin haber realizado acciones previas más que su conexión a la PC e instalación automática (Plug & Play).

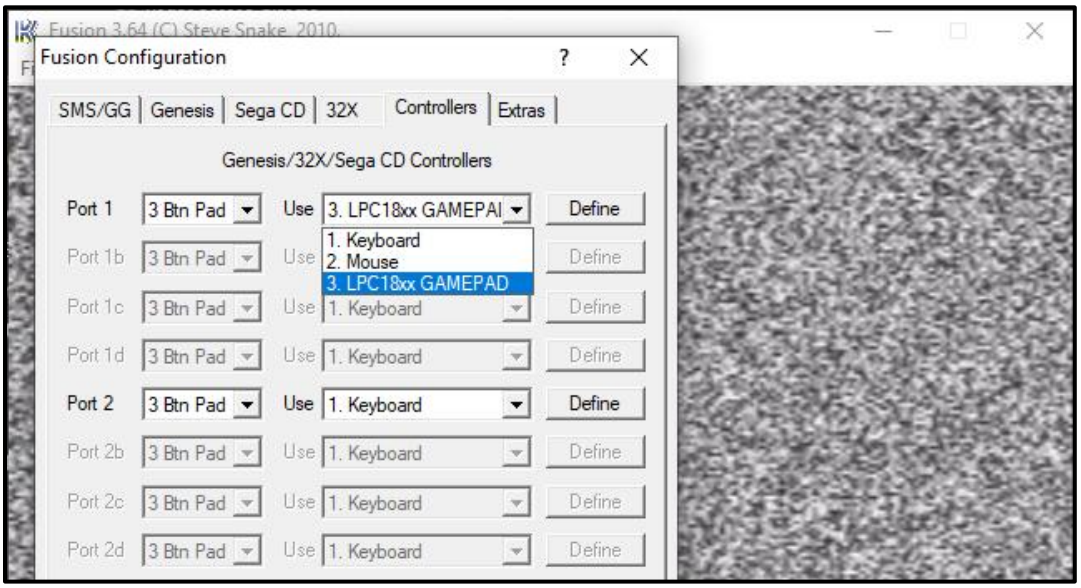


Fig. 6.6. Menú de configuración de Kega Fusion.

Al presionar el botón "*Define"*, el programa procede a solicitar la acción asociada a cada botón de un mando de consola original. Por ejemplo, la tecla UP se representa mediante el accionado del joystick del Arcade Stick hacia la pantalla del display. Video completo: https://drive.google.com/file/d/1SMGFla5DGp5MpzgbVHcXzP7hMUYGlZtW

Luego se observó que el tiempo de respuesta y precisión de los controles accionados sea correcto durante el desarrollo de un juego. A fines prácticos de grabación del video, se usó el joystick únicamente (https://drive.google.com/file/d/1vV6VOaGqK-GVD5_SO7MdMBs9VhZSk6ZG), pero todo el conjunto de controles fue testeado con diferentes videojuegos (joystick y pulsadores).

6.2 – Funcionamiento en Linux

Complementariamente, se comprobó el funcionamiento del Arcade Stick en el sistema operativo Linux Mint. Para ello, fue necesario instalar un conjunto de herramientas y calibración denominado ***joystick***, mediante el comando:

sudo apt-get install joystick

Para contar con una interfaz gráfica similar al Panel de Control de Windows se instaló el programa ***jstest-gtk*** [11]. Una vez conectado el sistema construido, al abrir el programa se muestra una ventana de preferencias, y desde allí se accede a la verificación de los 6 pulsadores y las 4 direcciones del joystick, como se observa en las Figuras 6.7 y 6.8 a continuación. Se adjunta un video adicional de las pruebas: https://drive.google.com/file/d/1SMghLMz3QuFSLfFyiKhMIIXa9An88nay

Figs. 6.7 y 6.8. Testeo de controles del Arcade Stick en Linux Mint.

7 – Conclusiones

Al comenzar el planteo del proyecto, la idea fue realizar un Arcade Stick que se asemeje lo más posible a uno real tanto funcional como estéticamente. Pero a lo largo del desarrollo del mismo se priorizó el correcto funcionamiento de los principales apartados necesarios para su ejecución, por lo que entonces el diseño mecánico completo no fue realizado. Con respecto a los demás objetivos se puede decir que la calibración de la conversión de la señal analógica se realizó de manera precisa y correcta al igual que el desarrollo de activación de los botones. También se puede mencionar que se implementó el driver más adecuado para la comunicación con la PC, ya que en primera instancia se habría optado por usarlo como teclado al joystick, asignándole teclas al mismo que correspondían a las habituales en los videojuegos, junto a un menú (a futuro) para modificar dichas teclas. Sin embargo, luego se optó por la realización de que el MCU fuese usado como configuración gamepad.

De forma similar, respecto a los requerimientos se pueden destacar las siguientes apreciaciones:

Funcionalidades de información al usuario: dentro de dichos requerimientos se realizaron todos a excepción de garantizar que la impresión de mensajes en LCD sea no-bloqueante y establecer una prioridad menor a las tareas de actualización de LED y LCD. Estas no se realizaron ya que las mismas no fueron necesarias debido a que el color del LED no varía durante el transcurso de un estado, permaneciendo encendido en verde o rojo según el resultado de la conexión. Con respecto a LCD, el mismo no se realizó bajo motivos similares: existe un único mensaje posible en cada estado.

Funcionales de controles: en dicha categoría de funcionalidades se pudo cumplir completa y correctamente todas las tareas, modificando los requerimientos de lectura donde, en lugar de enviar “UP”, “LEFT”, “DOWN”, “RIGHT” a la computadora de manera directa, se arma una estructura de reporte que contiene la intensidad accionada en cada eje (un número en dirección X y otro en Y), y los bits de pulsadores activos.

No funcionales: El único requisito no cumplido en este apartado es la fecha de muestra ya que el proyecto fue presentado y mostrado en febrero 2023.

Hardware a utilizar: Con respecto al hardware, se obtuvieron todos los componentes de manera correcta exceptuando el LED RGB, ya que accidentalmente se adquirió uno de tipo cátodo común en lugar de ánodo común, pero de todas maneras se logró su funcionamiento realizando una modificación manual al circuito.

A modo de resumen, se presenta una tabla que indica los requerimientos cumplidos por tipo (fila) y número (columna). Si se completó bajo las condiciones esperadas se muestra en verde; si se realizó de una forma distinta se muestra en amarillo; y en naranja aquellos que no se realizaron pero poseen justificación.

Tabla 6.1. Grado de cumplimiento de los requerimientos del proyecto.

	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII	XIV
3.1									-	-	-	-	-	-
3.2														
3.3						-	-	-	-	-	-	-	-	-
3.4					-	-	-	-	-	-	-	-	-	-

A continuación, se detalla una tabla representando las actividades desarrolladas por cada integrante, junto a las horas totales invertidas individuales:

Alumno	Tareas	Horas totales
Sergio	Especificación de requisitos Planificación diseño de componentes Desarrollo del código de comunicación con PC Diseño del esquemático Diseño del PCB Confección del poncho / Ensamble Testeo de comunicación con PC Armado de informes de avance Armado de informe final	65

Valentín	Especificación de requisitos Adquisición de componentes Algoritmos de lectura de controles Desarrollo del código de comunicación con PC Diseño del PCB Confección del poncho / Ensamble Testeo de controles Testeo de comunicación con PC Armado de informes de avance Armado de informe final	60
Lucas	Planificación diseño de componentes Adquisición de componentes Algoritmos de lectura de controles Desarrollo del código de comunicación con PC Diseño del esquemático Confección del poncho / Ensamble Testeo de controles Armado de informes de avance Armado de informe final	60

8 - Bibliografía

[1] Pernia, Eric (2019). *Especificaciones EDU-CIAA-NXP v1.1*. Disponible en: https://github.com/epernia/firmware_v3/blob/master/documentation/CIAA_Boards/NXP_LPC4337/EDU-CIAA-NXP/EDU-CIAA-NXP%20v1.1%20Board%20-%202019-01-03%20v5r0.pdf

[2] CIAA (2015). *Dimensiones de la placa EDU-CIAA-NXP.* Disponible en: https://github.com/ciaa/Hardware/blob/master/PCB/EDU-NXP/Doc/PCB.pdf

[3] Play Asia (2020). *Astro City Mini Arcade Stick*. Disponible en: https://www.play-asia.com/astro-city-mini-arcade-stick/13/70dptj

[4] Last Minute Engineers (s.f). *In Depth: How 2-Axis Joystick Works & Interface with Arduino + Processing.* Consultado el 14 de septiembre de 2022. Disponible en: https://lastminuteengineers.com/joystick-interfacing-arduino-processing/

[5] Joy-IT (2017). *KY-023 Joystick Module (X-Y Axis) Datasheet*. Disponible en: https://datasheetspdf.com/pdf-file/1402034/Joy-IT/KY-023/1

[6] Arduino Education (2021). *DIY Game Controllers.* Disponible en: https://www.arduino.cc/education/diy-game-controllers

[7] Component101 (2018). *Joystick Module: Pinout, Features, Arduino Circuit & Datasheet.* Disponible en: https://components101.com/modules/joystick-module

[8] Hitachi (s.f.). *LM016L Datasheet.* Disponible en: https://datasheetspdf.com/pdf-file/1462370/Hitachi/LM016L/1

[9] NXP (2020). *LPC435x/3x/2x/1x Datasheet.* Capítulo 10, p. 91.

[10] Semana (2013). *Los videojuegos estimulan la creatividad en adolescentes*. Disponible en: https://www.semana.com/vida-moderna/articulo/los-videojuegos-estimulan-la-creatividad-en-adolescentes/363868-3/

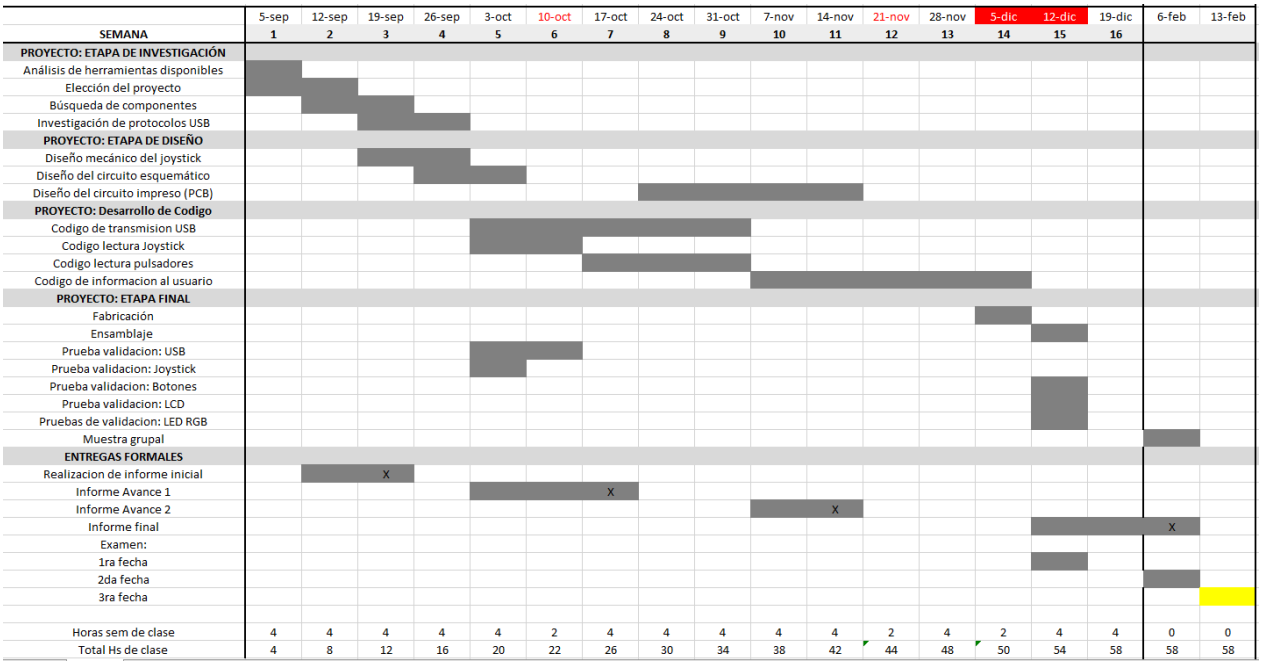
[11] Ruhnke, Ingo (2009). *Ubuntu Manpage: jstest-gtk.* Disponible en: https://manpages.ubuntu.com/manpages/xenial/man1/jstest-gtk.1.html

[12] Nubbeo (2022). *Pulsador Tact Switch 6mm x 6mm x 5.1mm.* Disponible en: https://www.nubbeo.com.ar/productos/pack-5-boton-pulsador-tecla-tact-switch-6mm-x-6mm-nubbeo/

9 – Anexos

9.1 – Cronograma

A continuación, se muestra el cronograma estimado del desarrollo del proyecto.



	5-sep	12-sep	19-sep	26-sep	3-oct	10-oct	17-oct	24-oct	31-oct	7-nov	14-nov	21-nov	28-nov	5-dic	12-dic	19-dic	6-feb	13-feb
SEMANA	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**		
PROYECTO: ETAPA DE INVESTIGACIÓN																		
Análisis de herramientas disponibles																		
Elección del proyecto																		
Búsqueda de componentes																		
Investigación de protocolos USB																		
PROYECTO: ETAPA DE DISEÑO																		
Diseño mecánico del joystick																		
Diseño del circuito esquemático																		
Diseño del circuito impreso (PCB)																		
PROYECTO: Desarrollo de Codigo																		
Codigo de transmision USB																		
Codigo lectura Joystick																		
Codigo lectura pulsadores																		
Codigo de informacion al usuario																		
PROYECTO: ETAPA FINAL																		
Fabricación																		
Ensamblaje																		
Prueba validacion: USB																		
Prueba validacion: Joystick																		
Prueba validacion: Botones																		
Prueba validacion: LCD																		
Pruebas de validacion: LED RGB																		
Muestra grupal																		
ENTREGAS FORMALES																		
Realizacion de informe inicial			X															
Informe Avance 1							X											
Informe Avance 2											X							
Informe final																	X	
Examen:																		
1ra fecha																		
2da fecha																		
3ra fecha																		
Horas sem de clase	4	4	4	4	4	2	4	4	4	4	4	2	4	2	4	4	0	0
Total Hs de clase	4	8	12	16	20	22	26	30	34	38	42	44	48	50	54	58	58	58

Fig. 9.1. Cronograma de realización del proyecto.

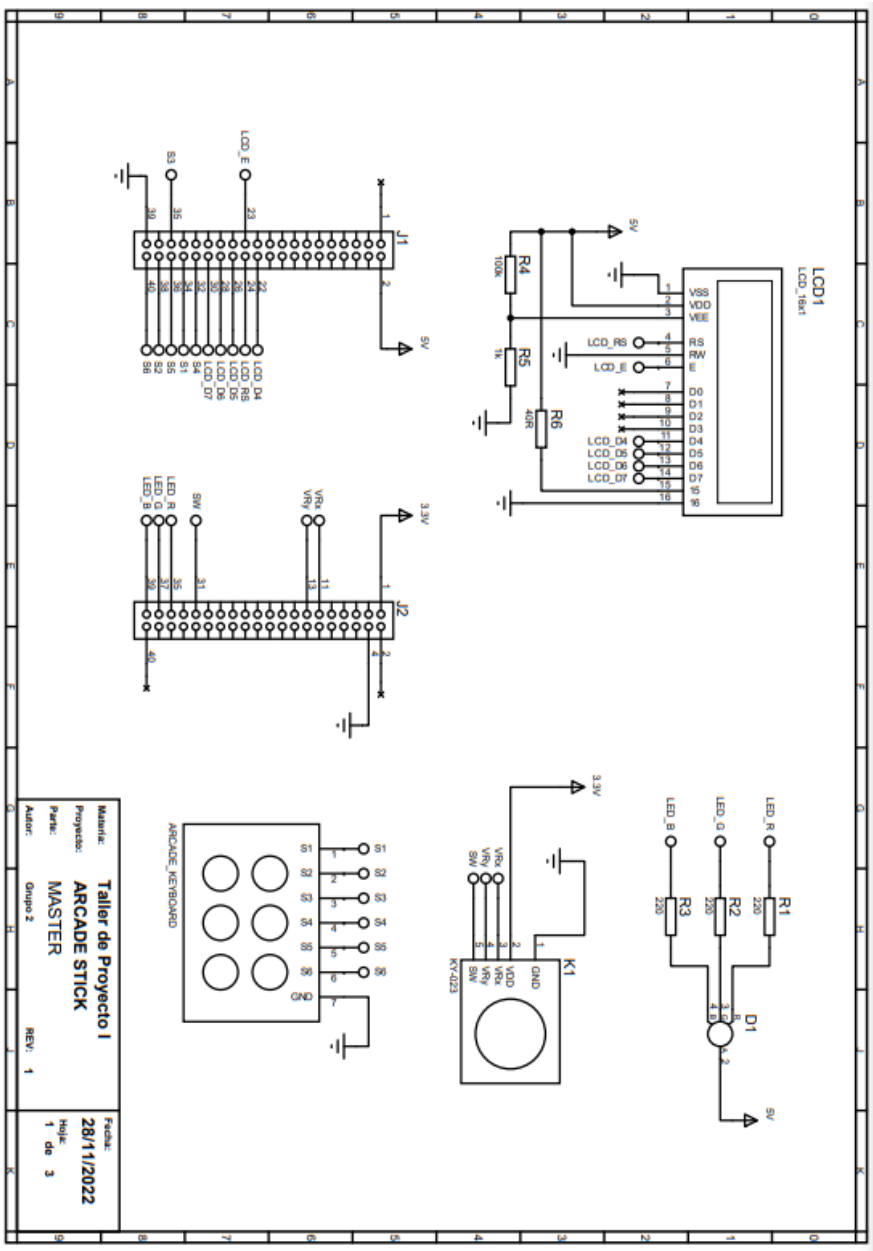
9.2 - División de tareas

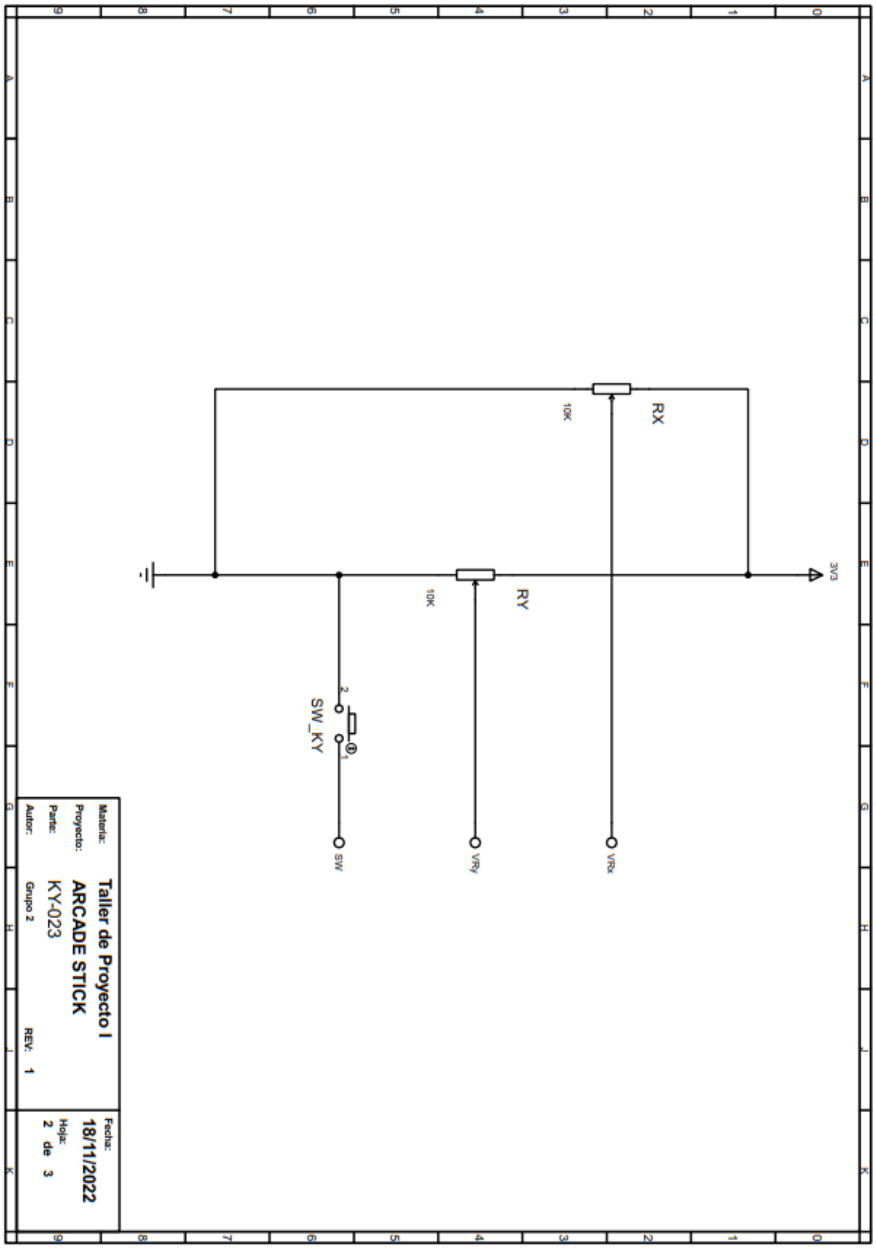
Se especifica la delegación de las actividades del proyecto a continuación:

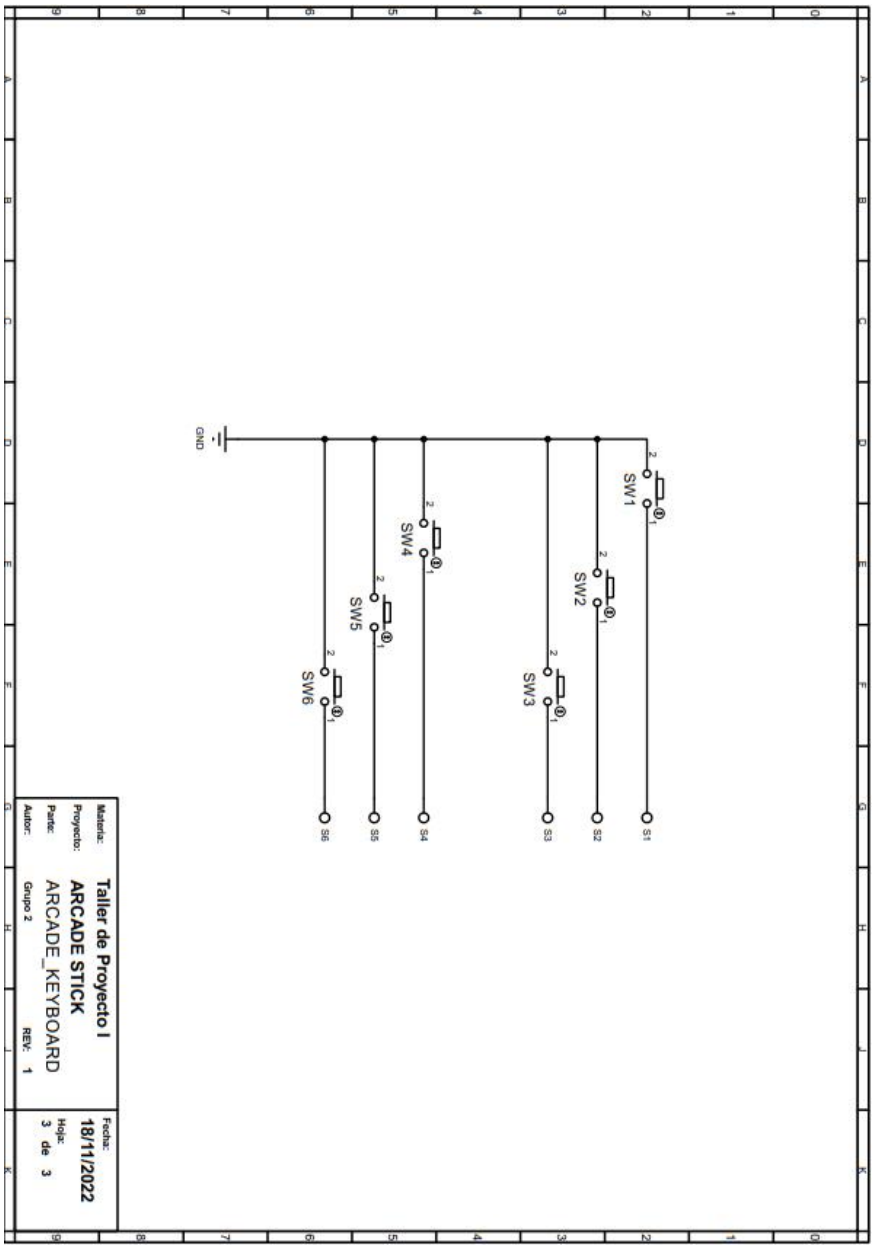
Tabla 9.1. División de tareas acordado por el grupo.

Tarea	Responsable	Colaborador
Especificación de requisitos	Calderón, Sergio	Blanco, Valentín
Adquisición de componentes	Blanco, Valentín	Bonifacio, Lucas
Planificación diseño de componentes	Calderón, Sergio	Bonifacio, Lucas
Algoritmos de lectura de controles	Bonifacio, Lucas	Blanco, Valentín
Desarrollo del código de comunicación con PC	Calderón, Sergio	Blanco, Valentín Bonifacio, Lucas
Diseño del esquemático	Bonifacio, Lucas	Calderón, Sergio
Diseño del PCB	Blanco, Valentín	Calderón, Sergio
Confección del poncho / Ensamble	Calderón, Sergio	Blanco, Valentín Bonifacio, Lucas
Testeo de controles	Bonifacio, Lucas	Blanco, Valentín
Testeo de comunicación con PC	Calderón, Sergio	Blanco, Valentín
Armado de informes de avance	Blanco, Valentín Bonifacio, Lucas	Calderón, Sergio
Armado de informe final	Blanco, Valentín Calderón, Sergio Bonifacio, Lucas	-

9.3 – Circuito Esquemático




10K
RX
3V3
10K
RY
2
SW_KY
SW
VRy
VRx
Materia: Taller de Proyecto I
Proyecto: ARCADE STICK
Parte: KY-023
Autor: Grupo 2
REV: 1
Fecha: 18/11/2022
Hoja: 2 de 3


GND
SW1
SW2
SW3
SW4
SW5
SW6
S1
S2
S3
S4
S5
S6
Materia: Taller de Proyecto I
Proyecto: ARCADE STICK
Parte: ARCADE_KEYBOARD
Autor: Grupo 2
REV: 1
Fecha: 18/11/2022
Hoja: 3 de 3

9.4 – Lista de materiales

A continuación, se presenta la lista de resistencias, diodos, sensores y otros componentes a adquirir para la realización de este proyecto. Los potenciómetros de 10kΩ y el pulsador SW_KY vienen incluidos en el sensor KY-023, por lo que su costo es cero.

Taller de Proyecto I

Bill Of Materials for ARCADE STICK

Design Title	ARCADE STICK
Author	Grupo 2
Document Number	1
Revision	1
Design Created	lunes, 28 de noviembre de 2022
Design Last Modified	lunes, 28 de noviembre de 2022
Total Parts In Design	20

6 Resistores

Quantity	References	Value	Stock Code	Unit Cost
3	R1-R3	220	M220R	$30,00
1	R4	100k	M220R	$30,00
1	R5	1k	M220R	$30,00
1	R6	40R	M220R	
Sub-totals:				$150,00

6 Pulsadores

Quantity	References	Value	Stock Code	Unit Cost
6	SW1-SW6	6x6x8mm		$50,00
Sub-totals:				$300,00

1 Sensores

Quantity	References	Value	Stock Code	Unit Cost
1	K1	KY-023		$549,00
Sub-totals:				$549,00

1 Diodos

Quantity	References	Value	Stock Code	Unit Cost
1	D1	5mm Ánodo Común		$50,00
Sub-totals:				$50,00

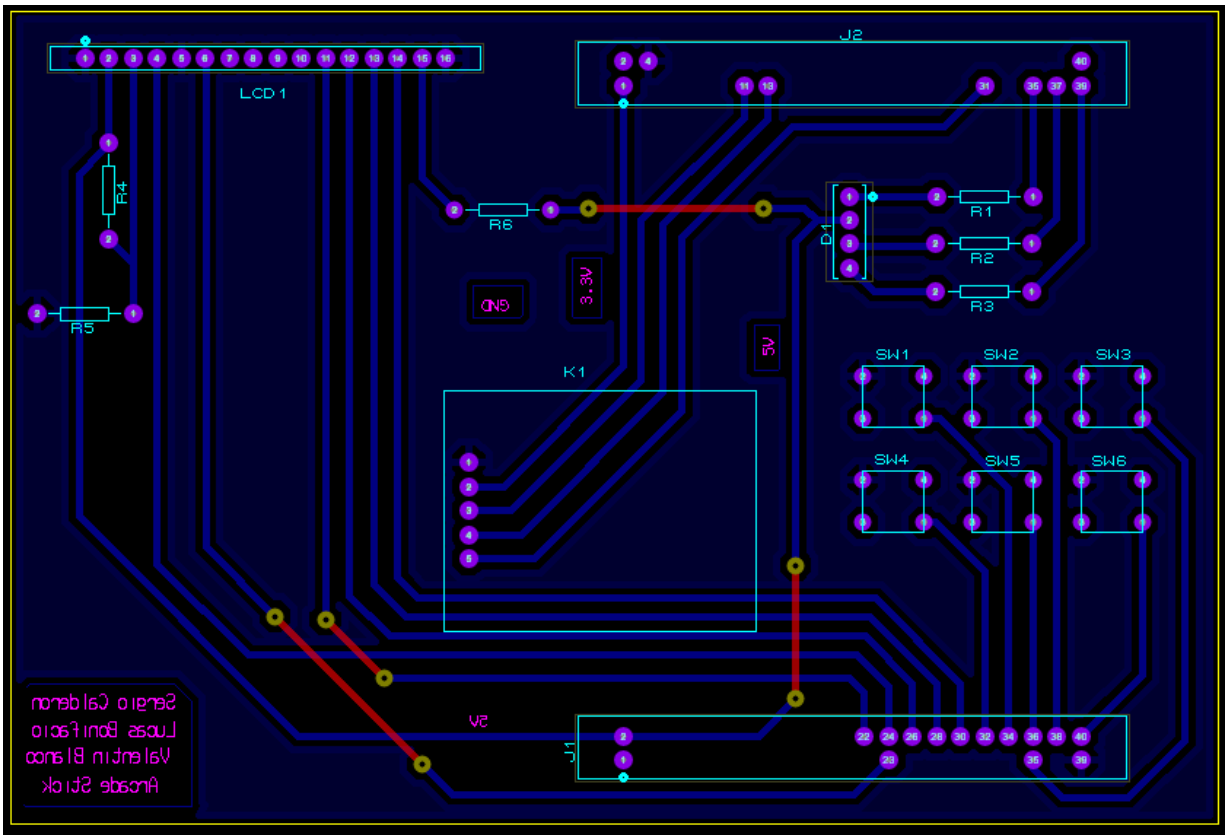
6 Misceláneos

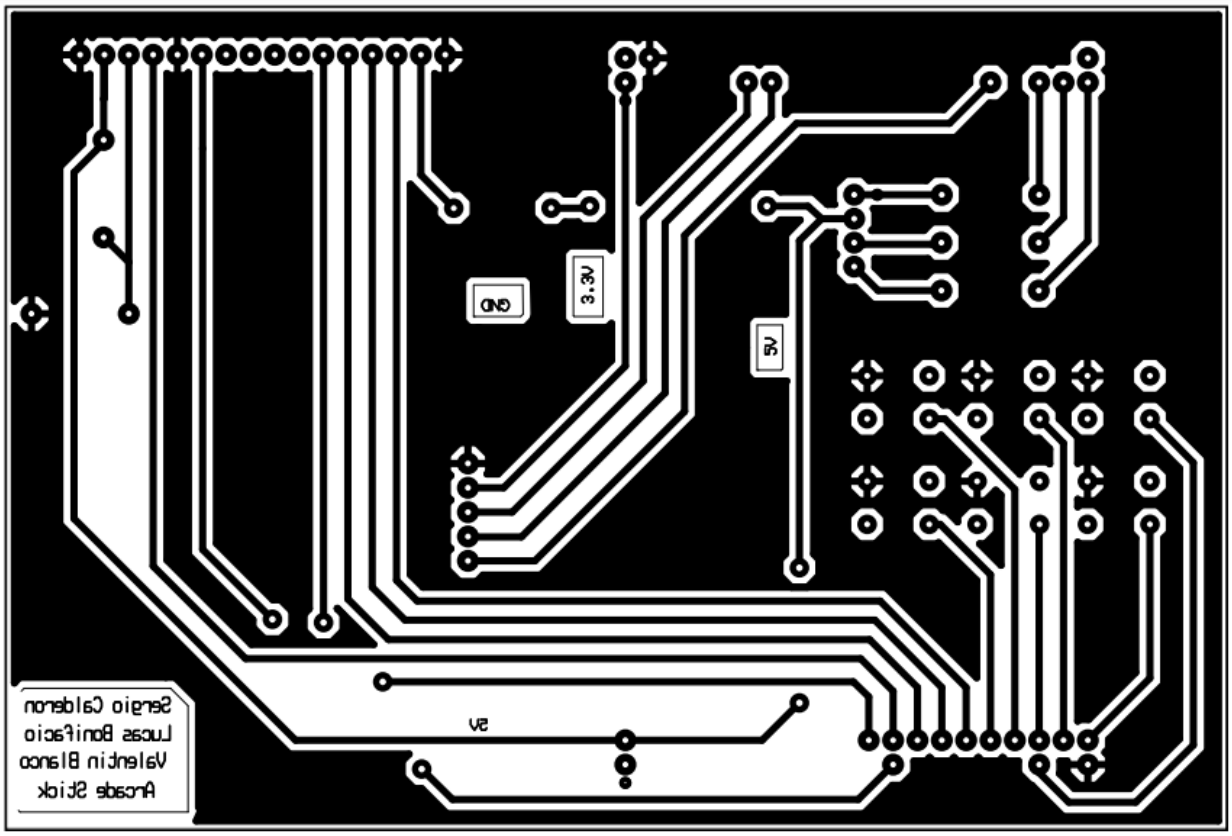
Quantity	References	Value	Stock Code	Unit Cost
1	J1	Macho 2x20	FCI 10073456-049LF	$130,00
1	J2	Macho 2x20		$130,00
1	LCD1	LCD_16x1		$980,00
2	RX,RY	10K	Digikey 3005P-103-ND	
1	SW_KY			$0,00
Sub-totals:				$1.240,00

Totals: $2.289,00

9.5 – Circuito Impreso (PCB)

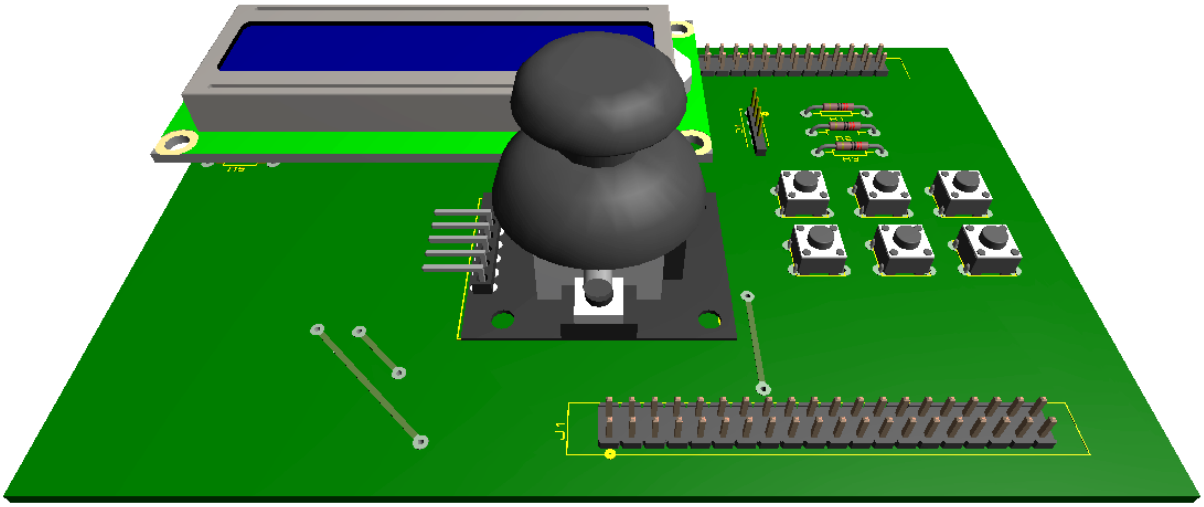
- Nota 1: tanto el proyecto de Proteus como el circuito impreso en formato PDF y escala 100% se encuentran a disposición en el siguiente enlace: https://drive.google.com/drive/folders/1wJFsdkhXU7rSod3emb0tdoCq3Gz_00d2?usp=sharing

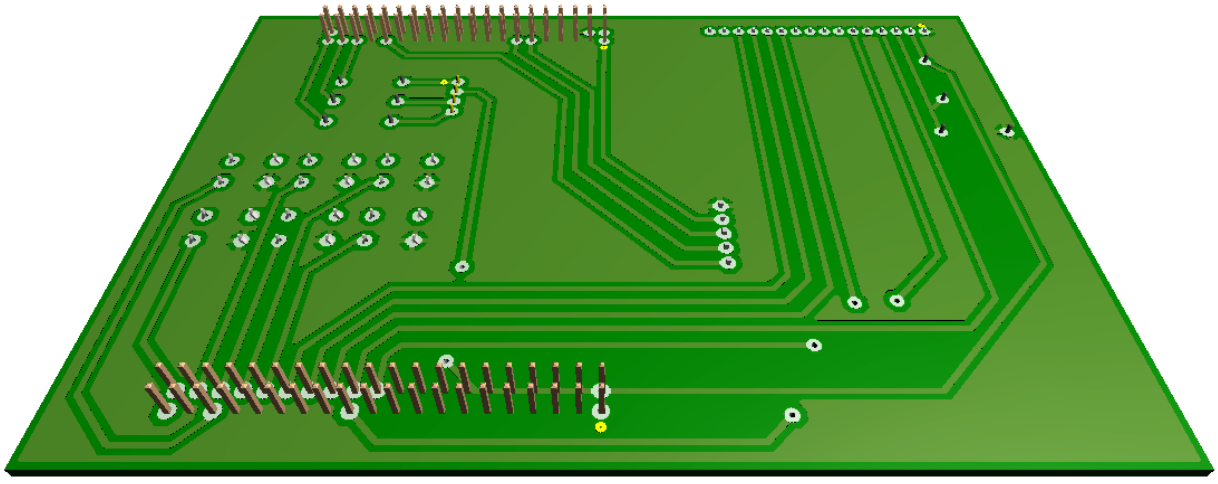



GND
3.3V
5V
5V
Sergio Calderon
Lucas Bonifacio
Valentin Blanco
Arcade Stick

9.6 – Vista 3D

- Nota 2: los modelos 3D (archivos STEP) del LCD, joystick, pulsadores y tiras de pines se adjuntan en la misma carpeta del enlace anterior.
- Nota 3: no se consiguió modificar el modelo 3D de la tira de pines para eliminar aquellos no utilizados, de modo que coincida con el PCB.





9.7 – Código fuente

En esta sección se presenta el código principal del sistema, es decir, los archivos de la carpeta app. El código completo se encuentra en un repositorio de GitHub, que se puede acceder mediante el siguiente link: https://github.com/sergiocarp10/tdp1_grupo2.

app.h

```
/*============================================================================
 * Autor:          Blanco Valentín, Bonifacio Lucas y Calderón Sergio
 * Fecha:          6 de febrero de 2023
 *==========================================================================*/

#ifndef _APP_H_
   #define _APP_H_

   /*==================[inclusiones]=============================*/

   // Biblioteca sAPI
   #include "sapi.h"

   // Perifericos
   #include "components.h"

   // Máquina de estados
   #include "MEF.h"

   // API simplificada para comunicación USB
   #include "gamepad_api.h"

   // Mapeo de los pines a utilizar para cada funcionalidad
   #include "mapeoGpio.h"

#endif
```

app.c

```
/*============================================================================
 * Autor:          Blanco Valentín, Bonifacio Lucas y Calderón Sergio
 * Fecha:          6 de febrero de 2023
 *============================================================================*/

// Cabecera del archivo
#include "app.h"

// Prototipos de funciones privadas
static void initComponents();


// -------------- Implementación de funciones públicas -------------- //

// FUNCION PRINCIPAL, PUNTO DE ENTRADA AL PROGRAMA LUEGO DE ENCENDIDO O RESET.
int main(void)
{
   /* ------------- INICIALIZACIONES ------------- */

   // Inicializar periféricos
   initComponents();

   // Iniciar máquina de estados (Conectando...)
   MEF_Start();

   // ---------- REPETIR POR SIEMPRE -------------------------

   while (1){
      // Actualizar estado con frecuencia 50 Hz
      MEF_Update();
      delay(20);
   }

   return 0;
}


// ---------------------- Funciones privadas ----------------------

static void initComponents(){

   // Inicializar EDU-CIAA
   boardConfig();
```

```
    // Inicializar LED RGB
    LED_Init();

    // Inicializar joystick (ADC)
    Joystick_Init();

    // Inicializar pulsadores
    Buttons_Init();

    // Inicializar display
    delay(900);
    Display_Init();
}
```

MEF.h

```
#ifndef _MEF_H_

    #define _MEF_H_

    /*==================[inclusiones]=============================================*/

    #include "components.h"
    #include "gamepad_api.h"

    /*============== [ declaración de funciones públicas] ================== */

    // Establece el estado inicial de la máquina de estados
    void MEF_Start();

    // Ejecuta el estado actual y establece el próximo
    void MEF_Update();

    /* ========================================================================= */

#endif
```

MEF.c

```
#include "MEF.h"

// Declaración de estados
typedef enum {
    CONNECTING, FAIL, CHECKING, READY
} MEF_Status;

// Variables privadas
static MEF_Status status;

// ------------------ Prototipos de funciones privadas ------------------ //

// Funciones propias de cada estado
static void MEF_Connecting();
static void MEF_Fail();
static void MEF_Checking();
static void MEF_Ready();

// Punteros a funciones de estado
static void (*MEF_Functions[])(void) = {
    MEF_Connecting, MEF_Fail, MEF_Checking, MEF_Ready
};

// ------------------- Implementación de funciones públicas ---------------- //

void MEF_Start(){
    // Establecer estado inicial
    status = CONNECTING;
}

void MEF_Update(){
    // Invocar función correspondiente al estado actual
    (*MEF_Functions[status])();
}

// ---------------- Implementación de funciones de estado ------------------- //

static void MEF_Connecting(){

    // Encender LED azul
    LED_EncenderAzul();

    // Mostrar mensaje "Conectando..." en el display
```

```
    Display_WriteMessage("Conectando...");

    // Establecer conexión con PC
    bool_t success = USB_Init();

    // Si el dispositivo fue reconocido, pasar a fase de chequeo
    if (success) status = CHECKING;
    else status = FAIL;
}


static void MEF_Fail(){
    static bool_t printed = false;

    if (!printed){
        // Mantener encendido el LED rojo
        LED_EncenderRojo();

        // Mostrar mensaje "USB incorrecto" por display
        Display_WriteMessage("USB incorrecto");

        // Marcar como impreso para evitar reescribir en display
        printed = true;
    }

}

static void MEF_Checking(){
    static uint16_t attempts = 0;

    // Alternar led azul cada 25 x 20 ms = 500 ms
    LED_Alternar(25);

    // Realizar intento de envío de datos
    bool_t success = USB_Attempt();

    // Si fue exitoso, pasar a estado Ready
    // en caso contrario, luego de 500 intentos (10s), pasar a Fail
    if (success) status = READY;
    else if (++attempts == 500) status = FAIL;
}


static void MEF_Ready(){
    static bool_t printed = false;
```

```
    if (!printed){
        // Mantener encendido el LED verde
        LED_EncenderVerde();

        // Mostrar mensaje "Listo para jugar" por display
        Display_WriteMessage("Listo para jugar");

        // Marcar como impreso para evitar reescribir en display
        printed = true;
    }

    // Realizar lectura de controles y enviarlo a PC
    bool_t success = USB_Update();

    // Si fue exitoso, mostrar direcciones actuales en display
    if (success){
        bool_t up, left, right, down;

        Joystick_GetDirs(&up, &left, &right, &down);
        Display_WriteDirs(up, left, right, down);
    } else {
        // Sino, pasar a estado de Fallo
        status = FAIL;
    }
}
```

components.h

```
#ifndef _COMPONENTS_H_
    #define _COMPONENTS_H_

   /*==================[inclusiones]=============================================*/

    #include "buttons.h"
    #include "display_api.h"
    #include "joystick.h"
    #include "led.h"

#endif
```

buttons.h

```
#ifndef _BUTTONS_H_
   #define _BUTTONS_H_

   /*==================[inclusiones]============================================*/

   #include "mapeoGpio.h"
   #include "sapi.h"

   /*================== [declaraciones de funciones públicas]==================*/

   // Establece los pines conectados a los pulsadores como entradas
   void Buttons_Init();

   // Realiza el barrido sobre los pulsadores considerando el efecto rebote
   // Devuelve por referencia los valores confirmados de cada botón
   void Buttons_Read(bool_t* values);

#endif
```

buttons.c

```
#include "buttons.h"

// Variables privadas
static uint8_t connectedPins[] = {PIN_S1, PIN_S2, PIN_S3, PIN_S4, PIN_S5,
PIN_S6};
static bool_t lastValues[] = {false, false, false, false, false, false};
static bool_t confirmedValues[] = {false, false, false, false, false, false};

// Prototipos de funciones privadas
static void Buttons_Check(bool_t readValue, uint8_t position);

// ------------------------ Funciones públicas ----------------------------

void Buttons_Init()
{
    // Variables locales
    uint8_t i;

    // Establecer cada pin conectado a un botón como entrada
    for (i = 0; i < 6; i++)
    {
```

```
        gpioInit(connectedPins[i], GPIO_INPUT);
    }
}

void Buttons_Read(bool_t *values)
{
    // Variables locales
    bool_t readValue;
    uint8_t i;

    // Para cada pulsador del gamepad
    for (i = 0; i < 6; i++)
    {
        // Leer estado actual
        readValue = !gpioRead(connectedPins[i]);

        // Realizar chequeo anti-rebote
        Buttons_Check(readValue, i);

        values[i] = confirmedValues[i];
    }
}

// ------------------------- Funciones privadas -------------------------------

// Anti-rebote: requiere leer varias veces el mismo valor para confirmarlo
static void Buttons_Check(bool_t readValue, uint8_t position)
{

    // Si el nuevo valor leído es distinto al confirmado
    if (readValue != confirmedValues[position])
    {
        // Si el nuevo valor leído es igual al último leído
        if (readValue == lastValues[position])
        {
            // Confirmar valor
            confirmedValues[position] = readValue;
        } else {
            // Sino, actualizar ultimo valor leido pero no confirmar
            lastValues[position] = readValue;
        }
    }
}
```

display_api.h

```
#ifndef _DISPLAY_API_H_
   #define _DISPLAY_API_H_

   /*==================[inclusiones]============================================*/

   #include "sapi.h"

   /*============== [declaraciones de funciones públicas]====================*/

   // Configura los parámetros del LCD y crea los caracteres de flechas
   void Display_Init();

   // Escribe el mensaje pasado por parámetro de forma bloqueante
   void Display_WriteMessage(char* msg);

   // Escribe las direcciones que se encuentren activas según parámetros
   void Display_WriteDirs(bool_t up, bool_t left, bool_t right, bool_t down);


/*================== [end of file]============================================*/

#endif
```

display_api.c

```
#include "display_api.h"

// Índice de caracteres personalizados
typedef enum {UP_ARROW, DOWN_ARROW, LEFT_ARROW, RIGHT_ARROW} Display_Arrows;

// Prototipos de funciones privadas
static void Display_AddCustomChars();


// ---------------------- Funciones públicas ------------------------- //

void Display_Init(){

   // Inicializar LCD de 16x2 con cada carácter de 5x8 pixeles
   lcdInit( 16, 2, 5, 8 );

   delay( LCD_STARTUP_WAIT_MS );
   // Apagar el cursor
```

```
    lcdCursorSet( LCD_CURSOR_OFF );

    // Agregar caracteres personalizados
    Display_AddCustomChars();
}

void Display_WriteMessage(char* msg){
     // limpiar pantalla y ubicar cursor al inicio
    lcdClearAndHome();

    // Escribir mensaje de manera bloqueante
    lcdSendString(msg);
}

void Display_WriteDirs(bool_t up, bool_t left, bool_t right, bool_t down){

    // Ubicar cursor en la segunda fila
    lcdGoToXY(2, 1);
    if (left) lcdData(LEFT_ARROW);
    else lcdSendString(" ");

    lcdGoToXY(5, 1);
    if (up) lcdData(UP_ARROW);
    else lcdSendString(" ");

    lcdGoToXY(8, 1);
    if (down) lcdData(DOWN_ARROW);
    else lcdSendString(" ");

    lcdGoToXY(11, 1);
    if (right) lcdData(RIGHT_ARROW);
    else lcdSendString(" ");
}

// --------------------- Funciones privadas ----------------------- //

static void Display_AddCustomChars(){

    const char upArrow[8] = {
        0b00000,
        0b00100,
        0b01110,
        0b10101,
        0b00100,
```

```
    0b00100,
    0b00100,
    0b00000
  };

  const char downArrow[8] = {
    0b00000,
    0b00100,
    0b00100,
    0b00100,
    0b10101,
    0b01110,
    0b00100,
    0b00000
  };

  const char leftArrow[8] = {
    0b00000,
    0b00000,
    0b00100,
    0b01000,
    0b11111,
    0b01000,
    0b00100,
    0b00000
  };

  const char rightArrow[8] = {
    0b00000,
    0b00000,
    0b00100,
    0b00010,
    0b11111,
    0b00010,
    0b00100,
    0b00000
  };

  lcdCreateChar( UP_ARROW, upArrow );
  lcdCreateChar( DOWN_ARROW, downArrow );
  lcdCreateChar( LEFT_ARROW, leftArrow );
  lcdCreateChar( RIGHT_ARROW, rightArrow );
}
```

gamepad_api.h

```
#ifndef _GAMEPAD_API_H_
   #define _GAMEPAD_API_H_

   /*==================[inclusiones]=======================================*/

   #include "sapi.h"
   #include "usbd_gamepad.h"

   #include "joystick.h"      // lectura de ejes X,Y
   #include "buttons.h"       // Lectura de pulsadores
   #include "mapeoGpio.h"


   /*=============== [declaraciones de funciones públicas]===================*/

   // Configura el driver del gamepad para su utilización
   bool_t USB_Init();

   // Realiza una lectura de los controles y los envía a la PC
   bool_t USB_Update();

   // Realiza un envío de datos a la PC, devolviendo true si resultó exitoso
   bool_t USB_Attempt();

#endif
```

gamepad_api.c

```
/* ----------------------------------------------------------

   Esta es una biblioteca simplificada para
   el manejo del driver USB desde el programa principal

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

// Includes
#include "gamepad_api.h"

// Variables compartidas
static volatile int8_t X_VALUE = 0;
static volatile int8_t Y_VALUE = 0;
```

```
// Variable auxiliar: bits de pulsadores a enviar
static uint8_t aux_bitsPulsadores = 0x00;

// Prototipos de funciones privadas
static int8_t convertRawToSigned(uint16_t);
static uint8_t USB_MarcarBoton(uint8_t numero);
static void USB_PresionarBotones();

void checkForPressedButtons(void* unused)
{
   // Leer pulsadores y marcar seleccionados
   bool_t pulsados[CANT_PULSADORES];
   Buttons_Read(pulsados);

   for (uint8_t i=0; i<CANT_PULSADORES; i++){
      if (pulsados[i]) USB_MarcarBoton(i);
   }

   // Actualizar pulsadores activos en PC
   USB_PresionarBotones();

   // Enviar posición de los ejes X e Y
   usbDeviceGamepadMove(X_VALUE, 0);
   usbDeviceGamepadMove(Y_VALUE, 1);

   // Enviar estado del Switch
   usbDeviceGamepadHat(Joystick_LeerSwitch());
}


// ----------------- Implementación de funciones públicas -------------------- //

bool_t USB_Init(){

   // Configurar driver y establecer conexión con PC
   bool_t driverSuccess = usbDeviceConfig(USB_HID_GAMEPAD);

   if (driverSuccess){
      // Asignar función de actualización de pulsadores
      usbDeviceGamepadCheckCallbackSet(checkForPressedButtons);
      return true;
   } else return false;
}
```

```
bool_t USB_Update(){
   // Leer eje X: el 0 está a la izquierda
   uint16_t valorEjeX = Joystick_LeerX();

   // Leer eje Y: el 0 está hacia arriba
   uint16_t valorEjeY = Joystick_LeerY();

   // Reducir valores al rango del driver
   X_VALUE = convertRawToSigned(valorEjeX);
   Y_VALUE = convertRawToSigned(valorEjeY);

   // Actualizar reporte
   return usbDeviceGamepadTasks();
}

bool_t USB_Attempt(){
   // No se realiza lectura del joystick, es una prueba de envío
   return usbDeviceGamepadTasks();
}

// ------------------- Implementación de funciones privadas -----------------

static int8_t convertRawToSigned(uint16_t raw){
   uint8_t reduced = (uint8_t) (raw / 4);
   return (int8_t) (reduced - 128);
}

static uint8_t USB_MarcarBoton(uint8_t numero){
   if (numero >= CANT_PULSADORES) return 0;

   // Los bits correspondientes a cada pulsador están
   // ordenados del menos al más significativo.
   // Ejemplo: 31 representa a los primeros 5 pulsadores activos
   aux_bitsPulsadores |= (1 << numero);
   return 1;
}

static void USB_PresionarBotones(){
   // Escribir el byte del reporte correspondiente a los pulsadores
   usbDeviceGamepadPress(aux_bitsPulsadores);

   // Una vez enviado, se reinicia para la próxima iteración
   aux_bitsPulsadores = 0x00;
}
```

joystick.h

```
#ifndef _JOYSTICK_H_
   #define _JOYSTICK_H_

   /*================== [inclusiones]=======================================*/

   #include "sapi_adc.h"      // adcInit() y adcRead()
   #include "sapi_gpio.h"     // gpioInit()

   #include "mapeoGpio.h"     // define pines SW, VRx, VRy

   /*================== [macros]============================================*/

   #define NEUTRO_MIN 400
   #define NEUTRO_MAX 624

   /*================== [declaraciones de funciones públicas]===============*/

   // Habilita el ADC y establece pin conectado al Switch
   void Joystick_Init();

   // Lee el valor del eje X y produce resultado digital entre 0 y 1023
   uint16_t Joystick_LeerX();

   // Lee el valor del eje Y y produce resultado digitañ entre 0 y 1023
   uint16_t Joystick_LeerY();

   // Lee el estado actual del Switch, devolviendo true si está presionado
   bool_t Joystick_LeerSwitch();

   // Recupera las direcciones activadas en la última lectura de ejes
   void Joystick_GetDirs(bool_t* dirUp, bool_t* dirLeft, bool_t* dirRight,
bool_t* dirDown);

   /*=============== [end of file]========================================= */

#endif /* _JOYSTICK_H_ */
```

joystick.c

```
// Este módulo es el único que tiene acceso directo a las funciones del ADC
#include "joystick.h"

// Variables privadas
static bool_t up, left, right, down;


void Joystick_Init(){
   // Inicializar y habilitar ADC
   adcConfig(ADC_ENABLE);

   // Asignar pin conectado al Switch como entrada digital
   gpioInit(PIN_SW, GPIO_INPUT);
}

uint16_t Joystick_LeerX(){
   uint16_t raw = adcRead(PIN_VRX);

   left = (raw < NEUTRO_MIN);
   right = (raw > NEUTRO_MAX);
   return raw;
}


uint16_t Joystick_LeerY(){
   uint16_t raw = adcRead(PIN_VRY);

   up = (raw < NEUTRO_MIN);
   down = (raw > NEUTRO_MAX);
   return raw;
}

bool_t Joystick_LeerSwitch(){
   return !gpioRead(PIN_SW);
}

void Joystick_GetDirs(bool_t* dirUp, bool_t* dirLeft, bool_t* dirRight, bool_t*
dirDown){
   *dirUp = up;
   *dirLeft = left;
   *dirRight = right;
   *dirDown = down;
}
```

led.h

```
#ifndef _LED_H_
   #define _LED_H_

   /*================ [inclusiones]=============================================*/

   #include "mapeoGpio.h"        // pines indicados para cada led
   #include "sapi.h"             // para gpioInit() y gpioWrite()

   /*==================[macros]=================================================*/

   // #define ANODO_COMUN
   #define CATODO_COMUN

   #ifdef ANODO_COMUN
      // El terminal común es VCC
      #define ENCENDIDO OFF
      #define APAGADO ON
   #else
      // El terminal común es GND
      #define ENCENDIDO ON
      #define APAGADO OFF
   #endif

   /*============== [declaraciones de funciones públicas]====================*/

   // Configura los pines utilizados como salida
   void LED_Init();

   // Apaga los 3 diodos del LED
   void LED_ApagarTodos();

   // Funciones para encender un color del LED
   void LED_EncenderRojo();
   void LED_EncenderVerde();
   void LED_EncenderAzul();

   // Apaga o enciende el último LED modificado
   void LED_Alternar(uint8_t periodo);


#endif
```

led.c

```
// Cabecera del archivo
#include "led.h"

// Variables privadas
static uint8_t ledEncendido = 0;
static bool_t apagado = true;

// Prototipos de funciones privadas
static void LED_Apagar(uint8_t);
static void LED_Encender(uint8_t);

// -------------------------- Funciones públicas ----------------------

void LED_Init(){
   gpioInit(PIN_LED_R, GPIO_OUTPUT);
   gpioInit(PIN_LED_G, GPIO_OUTPUT);
   gpioInit(PIN_LED_B, GPIO_OUTPUT);
}

void LED_Alternar(uint8_t periodo){
   static uint8_t count = 0;

   if (++count == periodo){
      if (ledEncendido > 0){
         if (apagado) LED_Encender(ledEncendido);
         else LED_Apagar(ledEncendido);
      }

      count = 0;
   }
}

void LED_ApagarTodos(){
   LED_Apagar(PIN_LED_R);
   LED_Apagar(PIN_LED_G);
   LED_Apagar(PIN_LED_B);
}

void LED_EncenderRojo(){
   LED_ApagarTodos();
   LED_Encender(PIN_LED_R);
}
```

```
void LED_EncenderVerde(){
   LED_ApagarTodos();
   LED_Encender(PIN_LED_G);
}

void LED_EncenderAzul(){
   LED_ApagarTodos();
   LED_Encender(PIN_LED_B);
}

// ------------------------- Funciones privadas ------------------------

static void LED_Apagar(uint8_t id){
   gpioWrite(id, APAGADO);
   apagado = true;
}

static void LED_Encender(uint8_t id){
   gpioWrite(id, ENCENDIDO);
   ledEncendido = id;
   apagado = false;
}
```

mapeoGpio.h

```
// En este header se indica cual es el pin de EDU CIAA utilizado para
// cada periférico del Arcade Stick
#ifndef _MAPEOGPIO_H_
   #define _MAPEOGPIO_H_

   /*=============== [inclusiones]=============================================*/

   // Cabecera donde se encuentran definidos los pines de la EDU-CIAA
   #include "sapi_peripheral_map.h"

   /*============== [macros]===================================================*/

   // #define DEPURACION

   // --------------------------- joystick ------------------------

   #define PIN_VRX   CH2
   #define PIN_VRY   CH1
   #define PIN_SW    T_COL1
```

```
   // ---------------------------- pulsadores ----------------------

   #ifdef DEPURACION
      #define PIN_S1    TEC1
      #define PIN_S2    TEC2
      #define PIN_S3    TEC3
      #define PIN_S4    TEC4
   #else
      #define PIN_S1    GPIO3
      #define PIN_S2    GPIO7
      #define PIN_S3    GPIO6
      #define PIN_S4    GPIO1
   #endif

   #define PIN_S5    GPIO5
   #define PIN_S6    GPIO8

   // ---------------------------- display -------------------------

   #define PIN_LCD_E    LCDEN
   #define PIN_LCD_RS   LCDRS
   #define PIN_LCD_D4   LCD1
   #define PIN_LCD_D5   LCD2
   #define PIN_LCD_D6   LCD3
   #define PIN_LCD_D7   LCD4

   // ---------------------------- leds ----------------------------

   #ifdef DEPURACION
      #define PIN_LED_R    LEDR
      #define PIN_LED_G    LEDG
      #define PIN_LED_B    LEDB
   #else
      #define PIN_LED_R    T_FIL2
      #define PIN_LED_G    T_COL0
      #define PIN_LED_B    T_FIL3
   #endif


#endif
```