Universidad Popular Autónoma del Estado de Puebla

MEC213: Sistemas Embebidos

Grupo 1 (08:30-10:00 Hrs)

Dr. Olmedo Urrutia Edson

Proyecto Final

Por:

Iván Ortiz De Lara

Emmanuel Issac García Sanabria

Edgar Enrique Castillo Quiñonez

Oscar Muñoz Cante

Domingo 02 de Diciembre del 2025

Objetivo

Desarrollar un Gamepad utilizando el microcontrolador PIC18F4550, el gamepad contara con un joystick analogico, D-Pad(cursor) y 6 botones (B1,B2,B3, B4,...).

Introducción

En la actualidad, los sistemas embebidos se encuentran en todas partes a nuestro alrededor: computadoras, microondas, celulares, licuadoras, entre otros. Este tipo de sistemas integran lo mejor de dos mundos, el hardware y el software, y, como si se tratara de una misión programada, tienen la capacidad de ejecutar repetidamente una misma tarea con alta precisión y autonomía.

Hablando de misiones, el objetivo de este proyecto fue desarrollar un gamepad alámbrico equipado con un joystick, un cursor y 6 botones. Este sistema embebido emplea el microcontrolador PIC18F4550, seleccionado gracias a la experiencia adquirida en clase mediante su uso y programación, lo que permitió aprovechar sus capacidades de comunicación USB, procesamiento y control de periféricos.

Este control alámbrico es capaz de conectarse a cualquier computadora y, como si se tratara de un modelo comercial, puede utilizarse como dispositivo de entrada para interactuar y brindar una experiencia dinámica en videojuegos, cumpliendo así su propósito principal: ofrecer una herramienta funcional orientada al entretenimiento.

Materiales

Los materiales utilizados para esta práctica fueron los siguientes:

- PIC18F4550
- Oscilador de cristal de 20MHz
- 2 Capacitores de 22pF
- 1 Capacitor de 220nF
- 11 Resistencias de 2k Ohms
- Resistencia de 220 Ohms
- Modulo Joystick 2 ejes
- Protoboard
- Placa fenólica
- Cable calibre AWG 24
- Puerto USB tipo A hembra
- Led verde
- 9 Push button
- Pines dupont macho
- Impresora 3D
- Cable USB tipo A macho-macho

Modificación de librería

Durante la primera simulación en Proteus, se detectó que la librería HID base no describía al control como un gamepad estándar. Para corregirlo, se modificó el descriptor HID, ajustando el Usage ID a 0x05, que en las USB HID Usage Tables corresponde a la categoría Gamepad. Esto permitió que el sistema operativo reconociera el control correctamente como un dispositivo HID de videojuego sin requerir drivers adicionales.

También se reorganizó el formato del reporte HID, dividiéndolo en 4 bytes: uno para la cruceta (4 bits), dos para los ejes analógicos del joystick (X e Y, 8-bits cada uno provenientes del ADC) y uno para los 6 botones

principales empaquetados como máscara de bits. Adicionalmente, se actualizó el nombre lógico del dispositivo para facilitar su identificación al conectarlo por USB.

La librería modificada se compiló junto con el firmware usando el compilador CCS C Compiler, quedando integrada al programa principal y validando su funcionamiento durante la simulación en Proteus.

Programación

El código del sistema embebido implementa la lectura de controles físicos (botones y joystick) y su transmisión a una computadora a través de comunicación USB, usando el microcontrolador PIC18F4550. Inicialmente se incluye el encabezado del dispositivo y se definen los fusibles, donde se configura el oscilador principal con PLL para obtener un reloj estable de 48 MHz, necesario para manejar correctamente la interfaz USB en modo Full Speed. También se habilita la regulación interna de 3.3 V para el módulo USB y se desactiva el watchdog timer y opciones de bajo voltaje que podrían reiniciar el sistema accidentalmente.

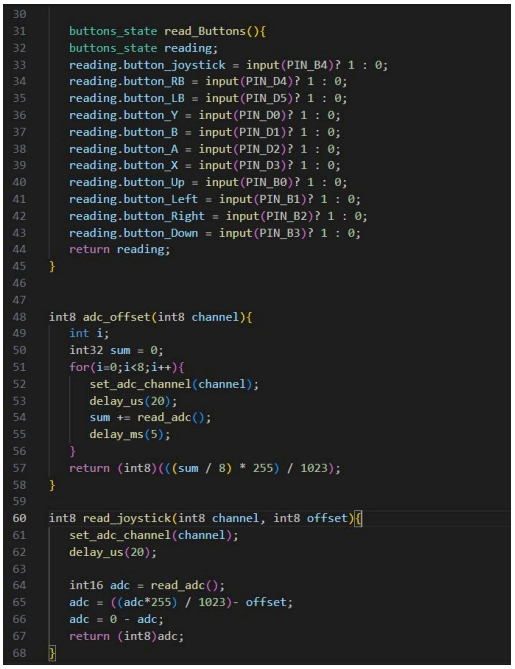
Posteriormente se incorporan las librerías de soporte USB: pic18_usb.h, usb.c y pic18_usb.c, que contienen las rutinas para la pila USB y la comunicación con el host, además de joy_control.h, que define los descriptores y formato del reporte HID del joystick/gamepad. Se definen constantes para identificar los canales ADC correspondientes a los ejes X y Y del joystick.

Para manejar los botones, se crea una estructura personalizada buttons_state, que agrupa todos los estados digitales de los botones del control, usando variables de 8 bits con valores binarios (1 = presionado, 0 = liberado). Esto permite almacenar de forma organizada el estado del joystick, bumpers (LB/RB), botones frontales (A, B, X, Y) y los 4 botones de la cruceta (D-Pad).

```
C:\Users • Contains emphasized items

#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN, XT
#use delay(clock=48000000, crystal=20000000,USB_FULL)

#include <pic18_usb.h>
#include <joy_control.h>
#include <usb.c>

#define JOY_AXIS_X 0
#define JOY_AXIS_Y 1

#define LED_RED PIN_B5
#define LED_GREEN PIN_B6
#define LED_BLUE PIN_B7

typedef struct buttons_state {
   int8 button_joystick;
   int8 button_RB;
   int8 button_LB;
   int8 button_Y;
   int8 button_B;
   int8 button_A;
   int8 button_X;
   int8 button_Up;
   int8 button_Left;
   int8 button_Right;
   int8 button_Down;
} buttons_state;
```

Figura 1.1 Código, definición de estructura, lectura de botones.

La función read_Buttons() se encarga de leer cada pin digital del microcontrolador, evaluando si hay un nivel lógico alto o bajo. Cada lectura se asigna al campo correspondiente dentro de una instancia de la estructura buttons_state y finalmente se devuelve dicho paquete de estados. Esta función aísla completamente la lógica de lectura de hardware del flujo principal del programa.



```
buttons_state read_Buttons(){
buttons_state reading;
reading.button_joystick = input(PIN_B4)? 1 : 0;
reading.button_RB = input(PIN_D4)? 1 : 0;
reading.button_LB = input(PIN_D5)? 1 : 0;
reading.button_Y = input(PIN_D0)? 1 : 0;
reading.button_B = input(PIN_D1)? 1 : 0;
reading.button_A = input(PIN_D2)? 1 : 0;
reading.button_X = input(PIN_D3)? 1 : 0;
reading.button_Up = input(PIN_B0)? 1 : 0;
reading.button_Left = input(PIN_B1)? 1 : 0;
reading.button_Right = input(PIN_B2)? 1 : 0;
reading.button_Down = input(PIN_B3)? 1 : 0;
return reading;
}


int8 adc_offset(int8 channel){
   int i;
   int32 sum = 0;
   for(i=0;i<8;i++){
      set_adc_channel(channel);
      delay_us(20);
      sum += read_adc();
      delay_ms(5);
   }
   return (int8)(((sum / 8) * 255) / 1023);
}

int8 read_joystick(int8 channel, int8 offset){
   set_adc_channel(channel);
   delay_us(20);

   int16 adc = read_adc();
   adc = ((adc*255) / 1023)- offset;
   adc = 0 - adc;
   return (int8)adc;
}
```

Figura 1.2 Lectura de botones.

Debido a que los joysticks analógicos presentan pequeñas variaciones incluso en estado de reposo, se implementa la función adc_offset(channel), la cual toma 8 lecturas consecutivas del canal ADC seleccionado, promedia los valores y luego convierte el resultado a una escala de 0–255. Este valor será utilizado como punto de calibración para compensar la desviación del sensor cuando el joystick está en neutral.

Una vez obtenidos los offsets, la función read_joystick(channel, offset) lee el valor actual del ADC en ese eje, lo escala a 8 bits (0 – 255), resta el offset previamente calculado para que el punto neutral sea 0, e invierte el signo multiplicando por –1 para ajustar la dirección según la convención del control. El resultado final se devuelve en formato int8, permitiendo valores positivos y negativos que representan el desplazamiento del joystick.

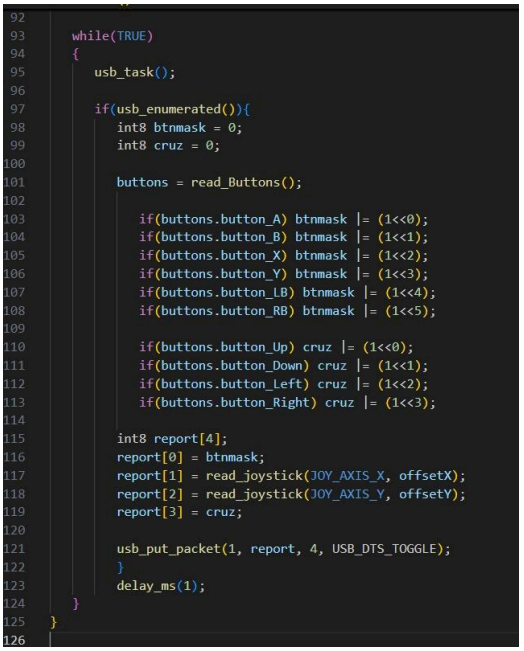
Dentro de main(), se inicializa el módulo USB mediante usb_init_cs() y luego se configura el ADC para recibir señales analógicas únicamente en AN0 y AN1 (ejes del joystick). También se configuran las direcciones de los puertos con TRIS, donde PORTB y PORTD se usan como entradas digitales para los botones.

```
void main()
{
   usb_init_cs();


   int8 offsetX = 128;
   int8 offsetY = 128;

   buttons_state buttons;

   setup_adc_ports(AN0_TO_AN1_ANALOG);
   setup_adc(ADC_CLOCK_INTERNAL);

   set_tris_a(0b00000011);
   set_tris_b(0b00011111);
   output_b(0b00000000);
   set_tris_d(0b00111111);

   offsetX = adc_offset(0);
   offsetY = adc_offset(1);

   output_high(LED_RED);
```

Figura 1.3 Función main principal.

Después de la calibración inicial del joystick, el LED rojo se enciende para indicar el arranque exitoso del sistema. El programa entra a un ciclo infinito donde usb_task() mantiene activa la pila USB. Una vez que el dispositivo ha sido enumerado correctamente por la computadora, se procede a construir dos máscaras de bits: btnmask, donde se empaquetan en un solo byte los 6 botones principales, y cruz, donde se empaquetan los 4 botones del D-Pad.

Finalmente, se construye un arreglo report[4] que representa el paquete HID enviado al host. Este reporte se compone de: report[0] la máscara de botones principales, report[1] el eje X, report[2] eje Y, y report[3] la máscara del D-Pad. Este paquete es enviado a través del endpoint USB 1 usando usb_put_packet(), a una velocidad controlada de 1 ms para asegurar estabilidad en el envío y evitar saturar el bus USB.



```
92
93      while(TRUE)
94      {
95         usb_task();
96
97         if(usb_enumerated()){
98            int8 btnmask = 0;
99            int8 cruz = 0;
100
101           buttons = read_Buttons();
102
103              if(buttons.button_A) btnmask |= (1<<0);
104              if(buttons.button_B) btnmask |= (1<<1);
105              if(buttons.button_X) btnmask |= (1<<2);
106              if(buttons.button_Y) btnmask |= (1<<3);
107              if(buttons.button_LB) btnmask |= (1<<4);
108              if(buttons.button_RB) btnmask |= (1<<5);
109
110              if(buttons.button_Up) cruz |= (1<<0);
111              if(buttons.button_Down) cruz |= (1<<1);
112              if(buttons.button_Left) cruz |= (1<<2);
113              if(buttons.button_Right) cruz |= (1<<3);
114
115           int8 report[4];
116           report[0] = btnmask;
117           report[1] = read_joystick(JOY_AXIS_X, offsetX);
118           report[2] = read_joystick(JOY_AXIS_Y, offsetY);
119           report[3] = cruz;
120
121           usb_put_packet(1, report, 4, USB_DTS_TOGGLE);
122           }
123           delay_ms(1);
124     }
125  }
126
```

Figura 1.4 Ciclo del main.

Con esta arquitectura, el sistema garantiza que el joystick en reposo no envíe valores residuales, que los botones sean transmitidos de forma compacta y eficiente, y que el control sea reconocido como un dispositivo HID estándar por cualquier computadora compatible con comunicación USB Full Speed.

Simulación de Proteus

Para la simulación inicial del gamepad se utilizó el software Proteus, el cual ya había sido empleado previamente en clase, facilitando así su implementación. En él se diseñó el esquemático integrando la PIC 18F4550, botones, capacitores y osciladores, permitiendo observar el comportamiento del circuito en un entorno virtual antes de la manufactura.

El código firmware se compiló usando el compilador CCS C Compiler, generando el archivo binario que se cargó a la PIC dentro de Proteus para validar que los datos del joystick y botones fueran leídos y transmitidos correctamente por USB bajo el protocolo HID.

Gracias a esta simulación se pudo verificar el funcionamiento lógico del hardware y software de manera conjunta, asegurando compatibilidad entre el circuito y el código antes de avanzar al diseño PCB y ensamble físico.

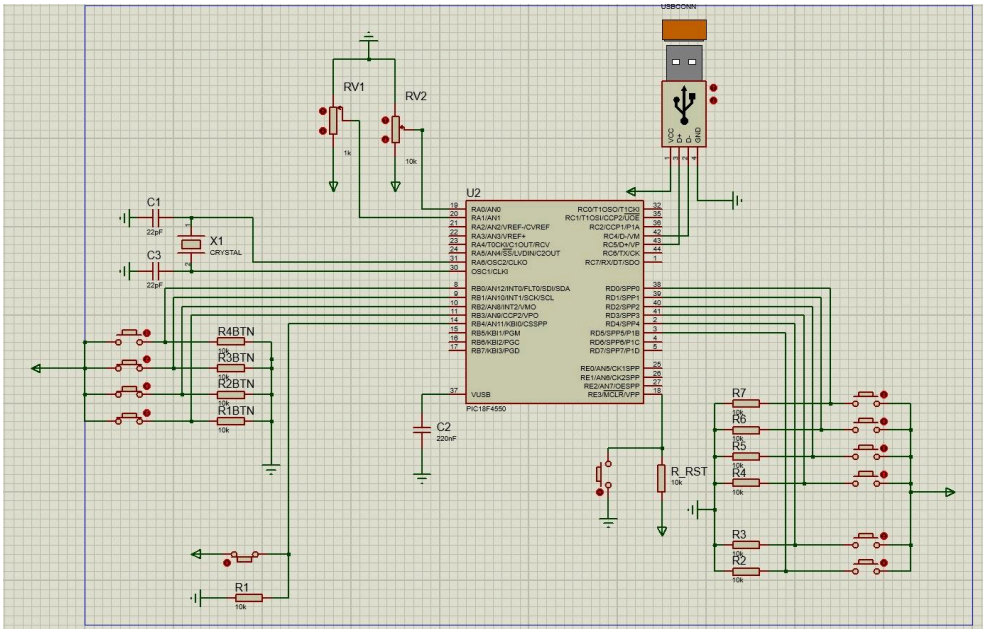


Figura 2.1 Simulación en Proteus

Ensamblado en Protoboard

Por último, se ensamblo el circuito en la protoboard con las mismas conexiones de la simulación realizada en Proteus, así mismo conectamos el PicKit 3 donde posteriormente con un cable usb se conecta el circuito en la computadora y esta la reconocía como un gamepad, el último paso fue utilizar una aplicación que te permite remapear las señales del control a Xinput para que los juegos puedan reconocer el control.

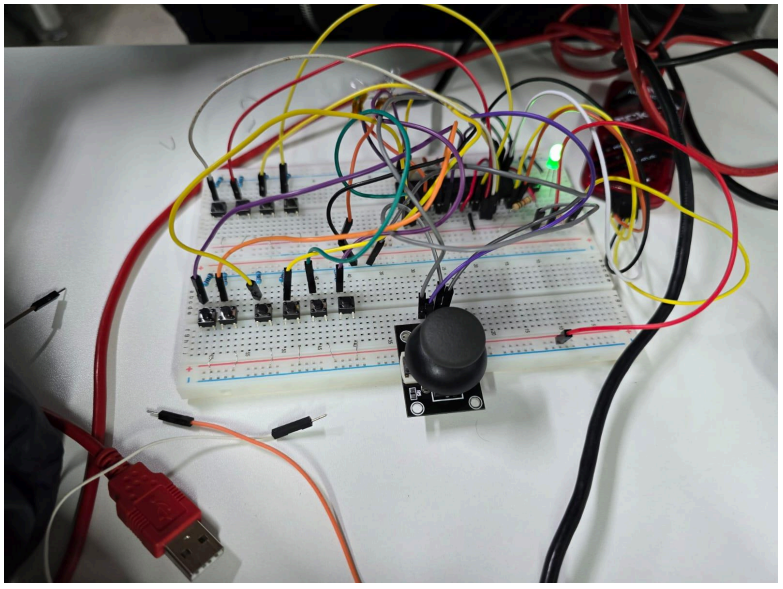


Figura 3.1 Circuito armado

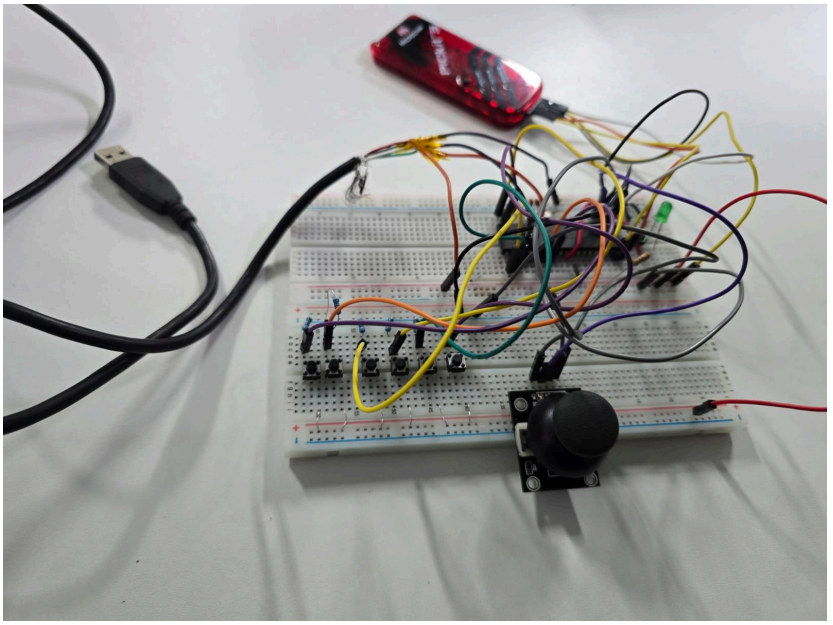


Figura 3.2 Circuito conectado a pickit3

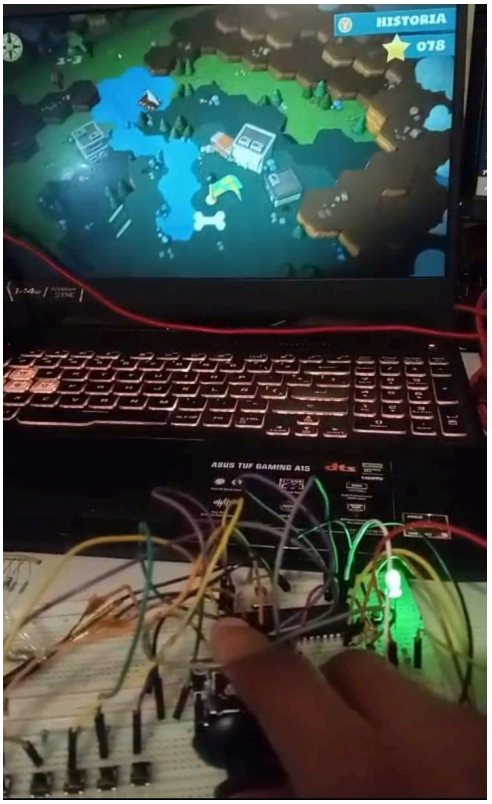


Figura 3.3 Protoboard funcional

Diseño de PCB

A partir del esquemático funcional simulado previamente en el software Proteus, se realizó la migración del diseño hacia el editor de PCB del software KiCad EDA, ya que este entorno no permite simulación eléctrica en tiempo real con los componentes físicos empleados posteriormente en el control.

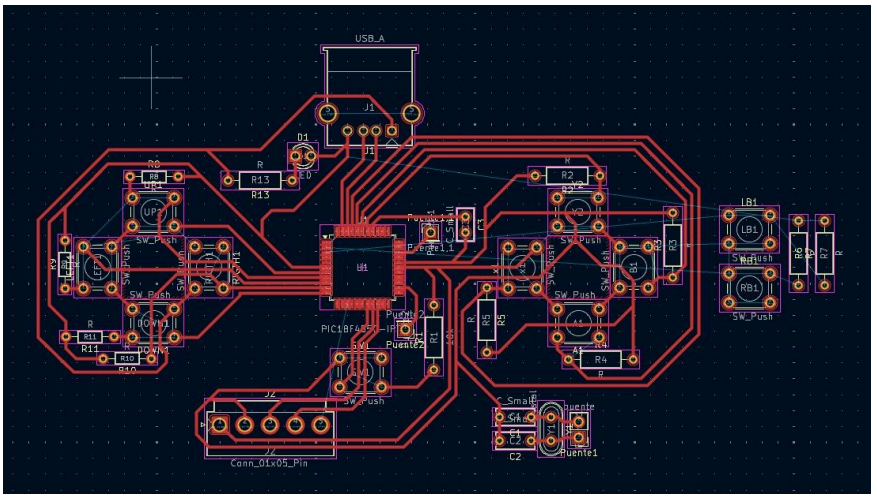


Figura 4.1 Diseño de circuito electrónico

Dentro del editor de placas, se definieron inicialmente las áreas de restricción de componentes y zonas libres de ruteo, asegurando espacio adecuado para el montaje. Posteriormente, se importaron los componentes y sus huellas correspondientes, organizándolos y alineándolos siguiendo una referencia ergonómica inspirada en un layout comercial similar al de un control de la marca PlayStation, tomando especial atención en las terminales más pequeñas para definir el ancho mínimo de pistas y evitar fallos por interferencias o dificultad de soldadura.

Una vez establecido el ruteo, se usó el visor 3D del sistema para validar la distribución final de componentes. Tras la revisión visual, se generaron los archivos de fabricación (drill y rutas de corte), los cuales fueron exportados hacia el software de maquinado FlatCAM para realizar el desbaste en CNC. El proceso de fabricación fue ejecutado en una fresadora, montando el material y definiendo el punto de origen y área de corte, realizando el maquinado en múltiples pasadas e intercambiando puntas según la operación requerida.

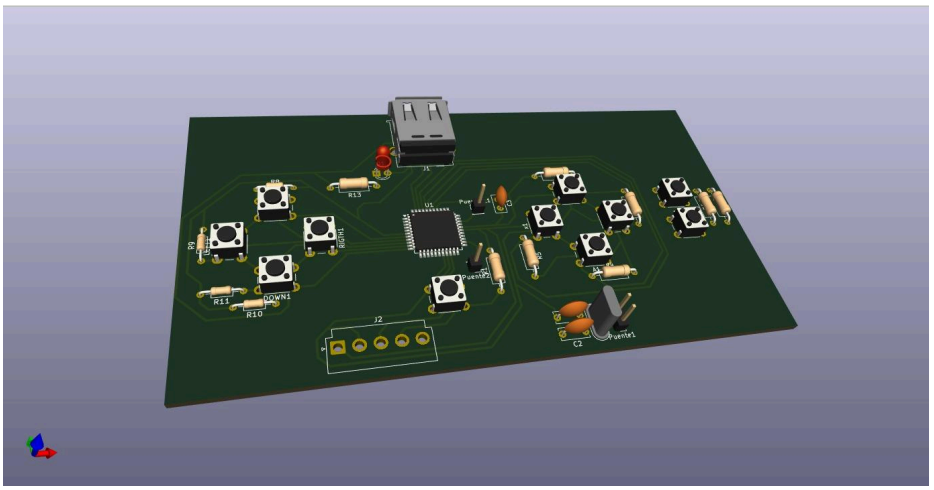


Figura 4.2 Visor en 3D del circuito final con todos los componentes soldados

Finalmente, la placa fue ensamblada mediante soldadura manual, utilizando cautín con punta delgada y pistola de calor para el montaje superficial de la PIC 18F4550, minimizando riesgo de cortos por exceso de material de aporte. Y corrigiendo con un exacto en caso de que surgiera algún corto en el circuito.

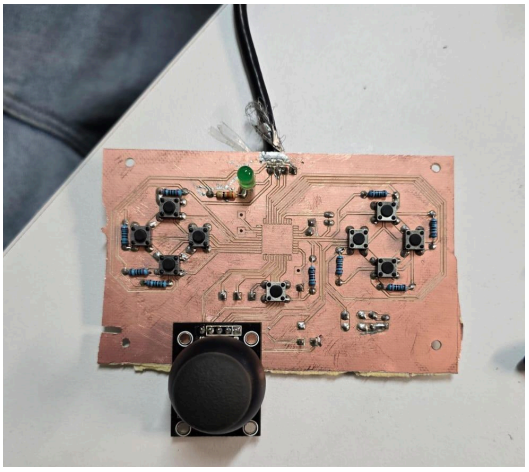


Figura 4.3 PCB en físico

Figura 4.4 PCB Funcional

Pruebas de prototipo

Después de verificar que las conexiones de los componentes electrónicos funcionaban correctamente en la PCB física y que el dispositivo era reconocido y operaba con normalidad en los videojuegos de la computadora, se integraron los botones correspondientes a los gatillos. Posteriormente, se realizaron pruebas adicionales para asegurar el funcionamiento completo del sistema, permitiendo así avanzar a la etapa de diseño de la carcasa. Esta fase tuvo como objetivo mejorar la ergonomía y el acabado del control, garantizando una experiencia más cómoda y precisa al utilizar el gamepad como dispositivo de interacción en videojuegos.

Figura 5.1 Diseño de carcasa del gamepad

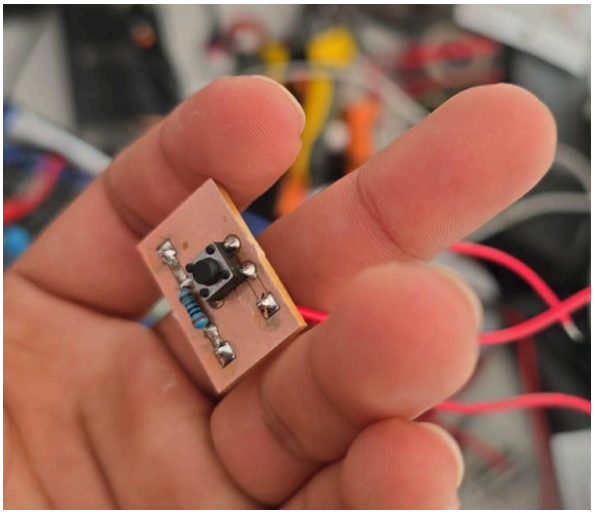


Figura 5.2 Tarjeta PCB de gatillos

Figura 5.3 Simulación de ensamblaje final

Figura 5.4 Renderización del control completo

Primer prototipo

El primer prototipo del gamepad se materializó mediante manufactura aditiva, empleando una impresora 3D y filamento de PLA, utilizando específicamente material de origen renovable derivado del maíz. Como resultado, se obtuvo un control funcional con una carcasa personalizada, diseñada para diferenciarse de los modelos comerciales existentes en el sector del entretenimiento digital.

No obstante, durante la evaluación del prototipo físico se identificaron distintos aspectos de mejora, entre los cuales destacan: dimensiones generales excesivas que dificultaban su uso prolongado, ergonomía inadecuada que generaba incomodidad al sostener y operar el control simultáneamente, y gatillos con ausencia de recorrido mecánico, lo que impedía percibir retroalimentación táctil tipo “clic”, afectando la experiencia de interacción esperada en videojuegos.

Estos hallazgos permitieron establecer criterios de optimización para iteraciones posteriores, priorizando la ergonomía, la sensación táctil, y la adecuación dimensional del dispositivo.

Figura 6.1 Carcasa del control

Figura 6.2 Carcasa completa del prototipo 1

Resultados

Gracias a los ajustes aplicados en el diseño, la optimización de las dimensiones, y la integración de un LED adicional como indicador de estado, se logró alcanzar el resultado esperado: un control tipo gamepad completamente funcional, con mejoras significativas en ergonomía, retroalimentación visual y calidad de ensamble.

El prototipo final ofrece un diseño sólido y preciso, y se encuentra preparado para ser utilizado en videojuegos mediante conexión USB, bajo el estándar de comunicación HID, garantizando compatibilidad nativa con cualquier computadora.

Figura 7.1 Simulación del nuevo diseño de carcasa

Figura 7.2 Nuevo prototipo gamepad

Figura 7.3 Ensamblado final de gamepad

Figura 7.4 Resultado final exitoso

Conclusiones

El desarrollo del gamepad permitió integrar de manera práctica los fundamentos de los sistemas embebidos, combinando hardware, firmware y comunicación USB bajo el estándar HID. A lo largo del proyecto se emplearon herramientas como el microcontrolador PIC18F4550, el compilador CCS C Compiler y el software de diseño electrónico KiCad EDA, validando inicialmente el circuito en el software de simulación Proteus.

La práctica demostró que un dispositivo embebido puede ejecutar tareas específicas de forma cíclica y confiable, como la lectura de entradas digitales y analógicas para interactuar con un sistema externo. Tras múltiples iteraciones y optimizaciones dimensionales, así como la manufactura de la carcasa mediante impresión 3D con PLA, se obtuvo un control funcional, ergonómico y compatible con cualquier computadora que soporte dispositivos HID nativos.

Con esta experiencia se concluye que los sistemas embebidos son pilares del desarrollo tecnológico actual, y que la correcta integración entre firmware, electrónica y diseño físico es esencial para materializar dispositivos funcionales destinados a interfaces de usuario, como el gamepad desarrollado, cumpliendo exitosamente su propósito en aplicaciones de entretenimiento digital.