Sistema basado en microprocesadores para la reproducción de una máquina Arcade

Autor: Kilian Peiró Conde  
Director: Enric Xavier Martin Rull

Especialidad: Ingeniería de computadores  
Departamento: ESAII

Abril 2018
Hasta no hace mucho, hacía falta ir a un salón recreativo para poder jugar a algún videojuego. Gracias a la evolución de la informática, hoy en día podemos jugar desde nuestro smartphone, ordenador personal o videoconsola en nuestra propia casa. Gráficos, sonido, inteligencia artificial... muchas han sido las mejoras que se han ido aplicando a los videojuegos, no obstante, hay gente que añora los de antaño. Los emuladores nos permiten emular un largo número de videoconsolas y jugar a sus juegos, cumpliendo los deseos de esta gente. Este proyecto aprovecha los computadores de placa reducida y los microprocesadores para fabricar una máquina arcade que emulará diversas videoconsolas y un controlador para la misma.

Abstract

A few years ago, we needed to go to the amusement arcades to play video games. Thanks to developments on technology, today we can play games on our smartphone, personal computer or video game console. Graphics, music, artificial intelligence... video games have been hugely developed since then, however, some people still long for the old ones. Emulators are used to emulate a large number of video consoles and play their games, fulfilling the wishes of those people. This project uses single-board computers and microcontrollers to build an arcade machine that will emulate some video consoles and a controller.

Resumen

Hasta no hace mucho, hacía falta ir a un salón recreativo para poder jugar a algún videojuego. Gracias a la evolución de la informática, hoy en día podemos jugar desde nuestro smartphone, ordenador personal o videoconsola en nuestra propia casa. Gráficos, sonido, inteligencia artificial... muchas han sido las mejoras que se han ido aplicando a los videojuegos, no obstante, hay gente que añora los de antaño. Los emuladores nos permiten emular un largo número de videoconsolas y jugar a sus juegos, cumpliendo los deseos de esta gente. Este proyecto aprovecha los computadores de placa reducida y los microprocesadores para fabricar una máquina arcade que emulará diversas videoconsolas y un controlador para la misma.

Resum

Fins no fa molt, feia falta anar a un saló recreatiu per a poder jugar a un videojoc. Gràcies a l’evolució de la informàtica, avui dia podem jugar des de el nostre smartphone, ordinador personal o videoconsola a casa nostra. Gràfics, so, intel·ligència artificial... moltes han estat les millorques que s’han anat aplicant als videojocs, no obstant, encara hi ha gent que sent nostàlgia per els d’abans. Els emuladors ens permeten emular un llarg número de video consoles i jugar als seus jocs, tornant realitat els desitjos d’aquesta gent. Aquest projecte aprofita les computadores de placa reduïda i els microprocessadora per fabricar una màquina arcade que emularà diferents video consoles i un controlador per la mateixa.
Agradecimientos:

A mi familia, por quererme y aguantarme pese a todo.

A los amigos de la planta 1 del Edifici Omega, por todos los buenos momentos.
Índice

1. Introducción 3
   1.1. Contexto 3
   1.2. Estado del arte 3
       1.2.1. Ordenador de sobremesa 3
       1.2.2. Ordenador de sobremesa en caja de máquina recreativa 3
       1.2.3. Ordenador de placa reducida en caja de máquina recreativa 3
       1.2.4. Software de emulación 4

2. Objetivos 5

3. Esquema del proyecto 5

4. Control de mando: PIC24F16KA101 6
   4.1. Software 6
   4.2. Firmware 6

5. Control de emulador: Raspberry Pi 12
   5.1. Software 12
   5.2. Firmware 12
       5.2.1. uinput 12
       5.2.2. xboxdrv y rc.local 13

6. Hardware: Esquemático de Eagle y montaje 15
   6.1. Primera versión 16
   6.2. Segunda versión 17
   6.3. Corrección de errores 17

7. Gestión del Proyecto 17
   7.1. Alcance 17
       7.1.1. Elección de hardware 17
       7.1.2. Entrada de datos y comunicación 18
       7.1.3. Manual de uso 18
   7.2. Metodología y rigor 18
       7.2.1. Método de trabajo 18
       7.2.2. Métodos de evaluación 18
   7.3. Situación legal 19
       7.3.1. Legalidad de emuladores 19
       7.3.2. Legalidad de ROMs de videoguegos 19
       7.3.3. Abandonware, freeware 20
   7.4. Planificación temporal 20
       7.4.1. Especificación del trabajo 20
           7.4.1.1. Gestión del proyecto 20
           7.4.1.2. Realización del proyecto 21
               7.4.1.2.1. Compra y testeo de componentes 21
               7.4.1.2.2. Testeo de componentes y decisiones 21
               7.4.1.2.3. Cableado y programación del firmware 21
               7.4.1.2.4. Retoques y housing 22
7.4.2. Diagrama de Gantt ........................................... 22
7.5. Gestión económica ........................................... 24
  7.5.1. Recursos Humanos ....................................... 24
  7.5.2. Recursos Software ....................................... 24
  7.5.3. Recursos Hardware ....................................... 24
  7.5.4. Consumibles ............................................. 25
  7.5.5. Costes indirectos ....................................... 25
  7.5.6. Presupuesto final ....................................... 26
  7.5.7. Imprevistos y contingencias ............................ 26
7.6. Sostenibilidad y compromiso social .......................... 27
  7.6.1. Dimensión Ambiental .................................... 27
  7.6.2. Dimensión Económica .................................... 27
  7.6.3. Dimensión Social ........................................ 28

8. Resultados ......................................................... 29
9. Conclusiones ....................................................... 32
  9.1. Posibles ampliaciones ..................................... 32

Referencias .......................................................... 34

Anexo ................................................................. 36
  A. newmainXC16.c ................................................. 36
  B. uinput.c ....................................................... 41
  C. rc.local ....................................................... 45
1. Introducción

1.1. Contexto

Un emulador[1] es un hardware o software que permite a un dispositivo de computación comportarse como otro dispositivo de computación. Hay emuladores de muchos tipos, por ejemplo de impresoras, de sistemas operativos... En este proyecto nos centramos en los emuladores de videoconsolas.

Un emulador de videoconsola es un programa informático de ordenador, o algún otro dispositivo de computación, que es capaz de emular una videoconsola, bien sea de sobremesa o portátil, de modo que el ordenador pueda ser utilizado para jugar a videojuegos que fueron creados para esta consola o desarrollar juegos para la misma. Pueden ser utilizados para traducir juegos a otros idiomas, modificar juegos existentes o probar juegos de demostración. Los emuladores de consola pueden además ser utilizados entre consolas, haciendo que una consola de videojuegos moderna pueda emular a una más antigua.

Mi proyecto trata de la fabricación de una máquina arcade low cost, para ello, se usarán microprocesadores, dado su bajo coste de compra y de consumo eléctrico. La máquina arcade emulará videojuegos de diversas plataformas.

1.2. Estado del arte

Actualmente hay varias maneras de emular juegos arcade, ya que al fin y al cabo simplemente hace falta un ordenador medianamente potente, dependiendo de la consola a emular.

1.2.1. Ordenador de sobremesa


1.2.2. Ordenador de sobremesa en caja de máquina recreativa

Otro enfoque es tener un ordenador de sobremesa dentro de un mueble, con forma de caja de máquina recreativa y usarlo únicamente para emular videojuegos. Al no tener periféricos como teclado o ratón para navegar por el sistema operativo, estas configuraciones suelen usar múltiples emuladores en algo conocido como un gestor de emuladores, que permite ser navegado usando los botones y el joystick previamente configurados.

1.2.3. Ordenador de placa reducida en caja de máquina recreativa

Tras la llegada de ordenadores de placa reducida como Raspberry Pi u Odroid, muchos usuarios crearon software para poder emular videojuegos en estos ordenadores, presentando una máquina arcade más compacta y portable. Además, estos
ordenadores tienen bajo consumo, son baratos, y presentan unos pines laterales, conocidos como GPIO, a los que se les puede dar múltiples usos, como por ejemplo, configurar un joystick y botones de máquina recreativa. No obstante, se puede jugar con mandos de otras consolas vía USB. También se utilizan gestores de emuladores para navegar por la consola.

Figura 1: Ordenadores de placa reducida corriendo videojuegos arcade.

El enfoque de mi proyecto se centra en este tipo de ordenadores, por lo que utilizaremos este sistema.

1.2.4. Software de emulación

Los software actuales y más conocidos para gestión de emuladores son RetroPie[4], Lakka[5], PiPlay[6], recalbox[7] y emulationstation[8].

Estos programas ofrecen un cómodo sistema de navegación para elegir a qué consola quieres jugar, y tienen uno o más emuladores para emularla. La mayoría suele utilizar RetroArch[9] o emulationstation como puente entre el programa y el emulador. Una de las diferencias más notables entre estos gestores (algunos aún en constante desarrollo) es el tipo de input que aceptan, esto es algo muy importante ya que hay gente a la que le gusta jugar con los mandos originales de las respectivas videoconsolas.

Todos estos gestores pueden ser utilizados por una Raspberry Pi.

Figura 2: Pantalla de emulationstation.
2. **Objetivos**

El objetivo principal del proyecto es la fabricación de una máquina arcade. Vamos a usar una Raspberry Pi para correr el software de emulación y un PIC para detectar la pulsación de botones y el movimiento del joystick. Las máquinas arcade de antaño eran de un gran tamaño, se quiere dar a entender que, con la evolución de la tecnología, ahora podemos tener todo lo que ofrecían en un housing de tamaño reducido.

Los requisitos mínimos de esta máquina serán:

- Emulación de varios videojuegos de diversas videoconsolas.
- Ausencia de lag o retraso en los videojuegos.
- Comunicación entre la Raspberry Pi y el PIC, sin problemas de input lag, rebotes o pérdida de input.
- Fabricación de un mando utilizando el PIC, botones y un joystick.
- Permitir varios jugadores.
- Housing compacto y portable, que con una pantalla y dos tomas de corriente funcione en cualquier lugar.

3. **Esquema del proyecto**

En la Figura 3 se muestra un esquema del proyecto, para tener una idea general sobre la disposición de los diferentes componentes.

![Esquema general del proyecto](image)

Figura 3: Esquema general del proyecto.
4. Control de mando: PIC24F16KA101

Los PIC[10] son una familia de microcontroladores tipo RISC fabricados por Microchip Technology Inc. y derivados del PIC1650, originalmente desarrollado por la división de microelectrónica de General Instrument. Se utilizará el PIC24F16KA101 para la detección de pulsación de botones, detección de movimiento de joystick y para la comunicación con la Raspberry Pi. Se elige este PIC porque tiene los pines suficientes para poder llevar a cabo estas tareas.

4.1. Software

Para poder introducir código en el PIC se usa un dispositivo llamado ICD3 [11], que se conecta entre un PIC y un ordenador utilizando algunos pines del PIC y el puerto USB del ordenador. El IDE que hemos utilizado es el oficial de Microchip, llamado MPLAB [12], ofrece herramientas para escribir código, debugar, compilar, y facilitar algunas configuraciones internas y librerías del PIC.

4.2. Firmware

Las dos funciones del PIC son claras, debe guardar el input de botones y, cuando la Raspberry Pi pida los datos por I2C, saltará una interrupción para ofrecérselos.

Una de las primeras ideas fue utilizar una librería de I2C para el PIC, pero tras muchos intentos de hacer que funcionase, leer a gente con problemas similares no solucionados en los foros y ver que los ejemplos que proporcionaba la librería eran todos del PIC haciendo de master, se optó por no usarlas. Hubo cierto momento durante el proyecto en el que vi que sólo funcionaban algunos de los botones, esto era debido a que el PIC usa algunos pines para otras funcionalidades y hay que desactivarlas, algunas desde la configuración interna usando #pragma config, y otras desde el main.
Primero se desactivan algunas funcionalidades del PIC, para poder utilizar todos los pines:

// FBS
#pragma config BWRP = OFF // Boot segment may be written
#pragma config BSS = OFF // No boot program Flash segment
#pragma config GWRP = OFF // General segment may be written
#pragma config GCP = OFF // General Segment Code Protection bit
#pragma config FNOSC = FRCDIV // 8 MHz FRC oscillator with divide-by-N
#pragma config IESO = OFF // Internal External Switchover
#pragma config POSCMOD = NONE // Primary oscillator disabled
#pragma config OSCIOFNC = ON // CLKO output disabled;
#pragma config POSCFREQ = HS // Primary Oscillator Frequency
#pragma config SOSCSEL = SOSCHP // Secondary oscillator high-power
#pragma config FCKSM = CSDCMD // Clock Switching and Monitor
#pragma config WDTPS = PS32768 // Watchdog Timer Postscale Select bits
#pragma config FWPSA = PR128 // WDT Prescaler
#pragma config WINDIS = OFF // Standard WDT selected
#pragma config FWDTEN = OFF // WDT disabled
#pragma config BOREN = BOR0 // Brown-out Reset disabled;
#pragma config PWRTEN = OFF // Power-up Timer Enable bit
#pragma config I2C1SEL = PRI // Alternate I2C1 Pin Mapping bit
#pragma config MCLRE = ON // MCLR pin enabled
#pragma config ICS = PGx1 // PGC1/PGD1 used for programming and debugging
#pragma config DSWDTPS = DSWDTPSF // Deep Sleep Watchdog Timer Postscale
#pragma config DSWDOSC = LPRC // DSMDT uses LPRC
#pragma config RTCDSC = SOSC // RTCC uses SOSC
#pragma config DSBOREN = OFF // Deep Sleep BOR disabled
#pragma config DSMDTEN = OFF // Deep Sleep Watchdog Timer Enable bit
Después se define la frecuencia y la dirección de slave del I2C, también se declaran variables y estados necesarios para el I2C:

```c
#define Fosc (8000000) // 8MHz crystal
#define Fcy (Fosc*4/2) // w.PLL (Instruction Per Second)
#define Fsck 400000 // 400 KHz I2C
#define I2C1_BRG ((Fcy/2/Fsck)-1)
#define SLAVE_ADD 15 // slave address

typedef enum {
    STATE_WAIT_FOR_ADDR,
    STATE_WAIT_FOR_WRITE_DATA,
    STATE_SEND_READ_DATA,
    STATE_SEND_READ_LAST
} STATE;

volatile STATE e_mystate = STATE_WAIT_FOR_ADDR;

#define BUFFSIZE 15
volatile char read_buffer[BUFFSIZE]; // master write slave read
volatile char write_buffer[BUFFSIZE]; // master read slave write
volatile uint16_t buffer_index;
```

A continuación mostramos la interrupción del I2C. Generalmente el PIC se encuentra en estado STATE_WAIT_FOR_ADDR, cuando le llega una petición de escritura, cambia al estado STATE_SEND_READ_DATA, envía el estado de los 14 botones y pasa al estado STATE_SEND_READ_LAST, y de vuelta al estado STATE_WAIT_FOR_ADDR:
void __attribute__((__interrupt__, no_auto_psv )) _SI2C1Interrupt(void) {
    uint8_t u8_c;
    IFS1bits.SI2C1IF = 0;

    switch (e_mystate) {
    case STATE_WAIT_FOR_ADDR:
        u8_c = I2C1RCV; // clear RBF bit for address
        buffer_index = 0;

        if (I2C1STATbits.R_W) {
            // check the R/W bit and see if read or write transaction
            I2C1TRN = write_buffer[buffer_index]; // send first data byte
            buffer_index++;
            I2C1CONbits.SCLREL = 1;
            // release clock line so MASTER can drive it
            e_mystate = STATE_SEND_READ_DATA; // read transaction
        } else e_mystate = STATE_WAIT_FOR_WRITE_DATA;
        break;

    case STATE_WAIT_FOR_WRITE_DATA:
        // character arrived, place in buffer
        read_buffer[buffer_index] = I2C1RCV; // read the byte

        if (read_buffer[buffer_index] == 0) {
            e_mystate = STATE_WAIT_FOR_ADDR; // wait for next transaction
        }
        break;

    case STATE_SEND_READ_DATA:
        I2C1TRN = write_buffer[buffer_index];
        buffer_index++;
        I2C1CONbits.SCLREL = 1;
        // release clock line so MASTER can drive it
        if (write_buffer[buffer_index] == -1) e_mystate = STATE_SEND_READ_LAST;
        // last character, release the clock line again
        break;

    case STATE_SEND_READ_LAST:
        // this is interrupt for last character finished shifting out
        e_mystate = STATE_WAIT_FOR_ADDR;
        break;

    default:
        e_mystate = STATE_WAIT_FOR_ADDR;
    }
}
En el programa principal se declaran los pines a usar como input, se limpian los canales de lectura y escritura del I2C, `write_buffer` y `read_buffer`, y se deshabilitan algunos módulos para que estos pines se puedan utilizar:

```c
unsigned int i;
for(i = 0; i < BUFFSIZE; ++i) {
    read_buffer[i] = 0;
    write_buffer[i] = 0;
}
read_buffer[BUFFSIZE - 1] = -1;
write_buffer[BUFFSIZE - 1] = -1;
```

//disable modules to make buttons work
AD1PCFG = 0b1101110001111111;
PMD1 = 0b0011100001101001;
PMD2 = 0b0000000000000001;
PMD3 = 0b0000011010000000;
PMD4 = 0b0000000000000110;
IEC0 = 0x0000;
IEC1 = 0x0000;
IEC3 = 0x0000;
IEC4 = 0x0000;
CTMUCONbits.CTMUEN = 0;
OSCCONbits.SOSCEN = 0;
REFOCONbits.ROEN = 0;
RCFGCALbits.RTCEN = 0;
RCFGCALbits.RTCOE = 0;
SPI1STATbits.SPIEN = 0;
SPI1CON1bits.DISSCK = 1;
SPI1CON1bits.DISSDO = 1;
OC1CONbits.OCM2 = 0;
OC1CONbits.OCM1 = 0;
OC1CONbits.OCM0 = 0;
HLVDCONbits.HLVDEN = 0;
CTMUCONbits.CTMUEN = 0;
```
También se habilitan los pines y el I2C:

```c
TRISA = 0xFFFF;
TRISB = 0xFFFF;

I2C1CONbits.I2CEN   = 1;
I2C1CONbits.I2CSIDL = 0;
I2C1CONbits.IPMIEN  = 0;
I2C1CONbits.A10M    = 0;
I2C1CONbits.DISSLW  = 1;
I2C1CONbits.SMEN    = 0;
I2C1CONbits.GCEN    = 0;
I2C1CONbits.STREN   = 1;

I2C1BRG = I2C1_BRG;
I2C1ADD = SLAVE_ADD;
```

Y en el bucle principal se leen los botones y el joystick:

```c
while(1) {
  /********* Main Buttons *********
  write_buffer[0] = PORTAbits.RA1;
  write_buffer[1] = PORTBbits.RB0;
  write_buffer[2] = PORTBbits.RB1;
  write_buffer[3] = PORTBbits.RB2;
  write_buffer[4] = PORTAbits.RA2;
  write_buffer[5] = PORTAbits.RA3;
  write_buffer[6] = PORTBbits.RB4;
  write_buffer[7] = PORTAbits.RA4;
  /********* Joystick **********/
  write_buffer[8] = PORTBbits.RB15;
  write_buffer[9] = PORTBbits.RB14;
  write_buffer[10] = PORTBbits.RB13;
  /******** Start+Select **********/
  write_buffer[12] = PORTAbits.RA6;
  write_buffer[13] = PORTBbits.RB7;
}
```
5. Control de emulador: Raspberry Pi

Raspberry Pi[13] es un computador de placa reducida, del tamaño de una tarjeta de crédito, creado por The Raspberry Pi Foundation en 2012 para promover la docencia de ciencias de computación en escuelas y países subdesarrollados. Además de ser pequeña, tiene un precio muy asequible (de unos 30 euros) y consume entre 1.5 W y 6.7 W.

En la placa hay puerto HDMI, USB, Ethernet... Además ofrece 40 pines, la mayoría programables, para poder añadir periféricos o comunicación con otros microprocesadores.

Las dos opciones para el computador de placa reducida eran Raspberry Pi o Odroid, me he decantado por Raspberry Pi porque ya había trabajado anteriormente con ella, además, hay mucha más documentación y soporte para la Raspberry Pi. Dentro de las múltiples versiones del computador, he decidido utilizar la Raspberry PI 3 ya que es la más nueva y cuenta con un mejor procesador.

5.1. Software

El software instalado en la Raspberry Pi es el sistema operativo que ofrece RetroPie, su imagen está basada en Raspbian Jessie, actualmente en estado oldstable.

RetroPie nos permite convertir nuestra Raspberry Pi en una máquina arcade, ofrece diferentes configuraciones y el usuario tiene acceso root a la consola, esto es obligatorio para el proyecto. También tiene diferentes opciones de energía, y abundantes métodos de entrada de datos.

5.2. Firmware

La Raspberry Pi se encarga de la emulación de videojuegos, de la comunicación con el PIC24F16KA101 y de la entrada de datos hacia el software de emulación, tanto RetroPie como retroarch. La parte de la emulación está más que cubierta usando el software de RetroPie, la parte de comunicación y entrada de datos se explica a continuación.

5.2.1. uinput

Para la entrada de datos, Linux tiene una carpeta de dispositivos en /dev llamada /input. En esta carpeta se añaden ficheros automáticamente al insertar dispositivos, con el nombre eventX, siendo X un número empezando por 0.
La primera idea fue documentarse e intentar crear un dispositivo de entrada usando la librería de Linux input.h, pero se descartó porque ofrece muchas mas cosas de las
que se necesitan y la implementación es difícil para lo que se requiere.
La segunda idea, y la que se está utilizando actualmente, es usar una librería llama-
da user input, o uinput.h. Esta librería nos permite crear un dispositivo a nuestra
elección usando pocos parámetros, y crea un fichero en /dev llamado uinput y tam-
bién crea el correspondiente eventX en /dev/input.

Se asignan los 14 botones (10 normales y 4 de joystick) a 14 teclas de un teclado,
por tanto, la Raspberry Pi detecta el dispositivo uinput y su respectivo eventX como
un teclado de 14 teclas. Las teclas asignadas son W A S D para el joystick y 0 1
2 3 4 5 6 7 8 9 para los botones.

La primera prueba realizada fue con un botón en la misma Raspberry Pi, al
apretar el botón aparecía el número 4 en la pantalla. También se probó en RetroPie,
en el apartado de configuración de entrada de datos, que detectaba un teclado al
pulsar el botón.
El primer fallo vino al probar un videojuego y ver que retroarch no detectaba el
botón como método de entrada, por lo visto, el emulador únicamente acepta dis-
positivos de tipo eventX en la carpeta /dev/input en unos formatos específicos,
incompatibles con el uinput implementado. La solución a este problema se explica
en el apartado 5.2.2.

Una vez lograda la comunicación con el emulador, hacía falta implementar la co-
municación I2C con el PIC para obtener los botones pulsados. Para esto he usado la
librería bcm2835[14], que cuenta con funciones para el protocolo I2C. Al saber la di-
rección I2C del esclavo y el tamaño de los datos a leer, la implementación fue sencilla.

El código del programa uinput2.c se encuentra en el Anexo B.

5.2.2. xboxdrv y rc.local

Como crear un dispositivo usando la librería input.h era demasiado engorroso y
tampoco me aseguraba que retroarch lo aceptase, busqué maneras alternativas para
que se reconociese mi eventX. Vi que los desarrolladores de RetroPie usan un driver
llamado xboxdrv, que transforma cualquier tipo de entrada al tipo de entrada gene-
rado por un mando de XBOX360. Por conocimientos previos a este proyecto, se que
el mando de la XBOX360 es uno de los más usados en comunidades de desarrollo indie por distintas facilidades y librerías que existen para él.

El driver se descarga desde el mismo RetroPie, y al ejecutar un comando de Bash
con los datos necesarios, mi entrada eventX pasa a ser joystickX, haciendo que la
consola, RetroPie y retroarch detecten todos los botones y el joystick perfectamente.
Este proceso tiene que ser totalmente transparente a ojos del usuario, por tanto,
se necesita que el programa uinput2 y el comando xboxdrv se ejecuten en un script
cada vez que se encienda la máquina.

Existe un fichero en Linux llamado rc.local, es un script que se ejecuta automáti-
camente al encender la máquina y es editable, es decir, se permite añadir comandos
to este script, justo lo que se necesita.
La parte importante del script de muestra a continuación:

`sudo /home/pi/ArcadeMachine-TFG/./uinput2 &
sleep 2`

`sudo /opt/retropie/supplementary/xboxdrv/bin/xboxdrv \
--evdev /dev/input/event2 \
--silent \
--detach-kernel-driver \
--force-feedback \
--deadzone-trigger 15\% \
--deadzone 4000 \
--mimic-xpad \
--dpad-as-button \
--evdev-keymap KEY_W=du,KEY_A=d1,KEY_S=dd,KEY_D=dr,KEY_1=start, KEY_2=back,KEY_3=a,KEY_4=b,KEY_5=x,KEY_6=y, KEY_7=1b,KEY_8=rb,KEY_9=tl,KEY_0=tr \
&`

Primero se ejecuta el programa `uinput`, seguidamente se esperan dos segundos para asegurar que el programa ha creado el dispositivo de entrada de datos y los demás parámetros necesarios. Después se ejecuta el comando `xboxdrv`, indicando que `eventX` debe leer y asignando los 14 botones de nuestro teclado a 14 botones del mando de XBOX360. Hay que tener en cuenta que el orden de aparición de los ficheros `eventX` puede cambiar entre una sesión y otra, actualmente y sin conectar ningún dispositivo adicional, el fichero generado por el programa `uinput2` es `event2`.  


6. Hardware: Esquemático de Eagle y montaje

En el entorno de pruebas se usaron cables de propósito general para unir el PIC a los botones y a la Raspberry Pi mediante protoboard. Posteriormente se soldó el PIC a una placa de topos, permitiendo un diseño más robusto y utilizando WireWrap hasta que, una vez conseguimos hacer que el proyecto funcionase, se soldaron todos los botones y el joystick a la placa de topos.

Para tener claro como se harían las soldaduras y no equivocarme, me ayudé de un esquemático de Eagle, que se muestra en la Figura 6.

Figura 6: Esquemático de Eagle.

Para la Raspberry Pi y para el ICD3 se siguen usando cables normales, esto nos permite poder cambiar de Raspberry Pi y actualizar el firmware del PIC cuando queramos sin necesidad de eliminar soldaduras.

Las siguientes imágenes muestran las dos etapas del proyecto, la primera con cables normales y la segunda con soldaduras.
6.1. Primera versión

Figura 7: PIC24F16KA101 en protoboard y Raspberry Pi con cables normales.

Figura 8: Versión para pruebas con sólo un botón.
6.2. Segunda versión

Figura 9: Soldaduras en una placa de topos, imágenes frontal y trasera.

6.3. Corrección de errores

Durante la elaboración del proyecto se rompieron algunos pines y vi que otros estaban muy saturados, por lo que soldé pines adicionales para solventar estos errores.

7. Gestión del Proyecto

A continuación se muestra la gestión del proyecto, realizada en Septiembre en la asignatura de GEP. Al haber extendido el TFG se han hecho algunas modificaciones.

7.1. Alcance

Teniendo en cuenta los objetivos del apartado 2, se necesita acotar el alcance del proyecto, los pasos a seguir se muestran a continuación.

7.1.1. Elección de hardware

Primero voy a probar diferentes emuladores en diferentes versiones de Raspberry Pi, se busca que no haya *lag* en los juegos, que haya un gran catálogo de juegos y que el gestor de emulador escogido permita el *input* necesario. Una vez elegidos, usaré siempre ese emulador y esa Raspberry Pi.
7.1.2. Entrada de datos y comunicación

Crear un programa o firmware que registre el input de un botón, primero directamente al microprocesador (Raspberry Pi) y después usando un intermediario (PIC o arduino). Probar diferentes protocolos de comunicación, ya que voy a necesitar el que tenga menos input lag, menor tiempo desde que aprietas un botón hasta que ves el resultado en pantalla. Una vez elegido el protocolo, ampliaré el código de un botón para que funcione con varios botones y joystick.

7.1.3. Manual de uso

Tengo que dejar claros los límites de mi creación, para que el usuario que lo vaya a utilizar no compe el producto con la idea de jugar a algo que va a tener mala usabilidad, es decir, no crear falsas expectativas. Si es necesario el uso de alguna librería o servicio que no venga por defecto en el emulador, debe constar en el documento.

7.2. Metodología y rigor

7.2.1. Método de trabajo

Al ser un proyecto algo grande para una sola persona, es necesario algún método de planificación que nos permita ver las tareas hechas y por hacer, y tener una idea general del estado del proyecto.

Para el desarrollo del proyecto se utilizará la herramienta Trello[15], que permite al usuario organizar tarjetas en un tablero. En concreto utilizaremos una tabla Kanban[16], disponiendo las tarjetas en los diferentes estados:

- **Por hacer**: Tareas por realizar.
- **En proceso**: Tareas realizándose.
- **En espera**: Tareas a la espera de algo ajeno a nosotros, como que llegue una compra online o la respuesta a un correo electrónico.
- **Hecho**: Tareas realizadas.

El proyecto tiene la ventaja de tener unas etapas muy marcadas, por lo que en principio no va a hacer falta organizar las tareas por importancia, sino por etapas.

Con el fin de tener el código accesible desde cualquier ordenador con internet y guardar un historial del mismo, utilizaré un sistema de control de versiones llamado GitHub[17].

7.2.2. Métodos de evaluación

Es necesaria una manera de comprobar si lo que se ha ido haciendo hasta ahora funciona.
Para evaluar el *lag* de los emuladores usaré un visor de *frames* por segundo, el emulador y la Raspberry Pi con más *frames* por segundo serán los elegidos.

Para evaluar el código y comportamiento de los botones, se realizará el proceso en diferentes etapas, comprobando que la entrada de datos funciona en todas ellas. También se hará uso del osciloscopio cuando sea necesario.

### 7.3. Situación legal

Los emuladores, argumentan sus usuarios, permiten jugar a videojuegos que jugaron en su juventud y que ya no están disponibles de otra forma que no sea a través de la emulación, pero, como bien sabemos, no estamos pagando ni por la videoconsola ni por el videojuego que estamos emulando, por tanto, debemos saber en qué marco legal nos encontramos. Nos centramos en las leyes del Ministerio de cultura de España [18], que no profundiza mucho en este tema, pero sí que hay varias leyes sobre la propiedad intelectual a las que nos deberemos ceñir.

#### 7.3.1. Legalidad de emuladores

Pese a que muchas empresas están en contra, es legal implementar código que emule una consola en un ordenador, no obstante, para desarrollar el código no se puede usar el original, por tanto, mucha gente usa técnicas de ingeniería inversa para que no sea ilegal.

Ocurre lo mismo con la ROM de *firmware/BIOS* que necesitan algunas plataformas para poder funcionar. La ROM del fabricante está protegida por *copyright*, por lo que el código tiene que ser conseguido a través de ingeniería inversa.

#### 7.3.2. Legalidad de ROMs de videojuegos

Los videojuegos están dentro de la categoría de programa de software, eso significa que tienen derechos de autor (igual que un libro, un cuadro, una película o una canción).

Por eso, la ley fija un plazo de tiempo durante el cual sólo los autores o las empresas de videojuegos que tengan esos derechos pueden venderte el juego, aunque sea en forma de ROM, en concreto, hasta que pasen 70 años de la muerte del autor (del último en caso de ser varios autores).

En el caso de que el *copyright* haya expirado, o si los desarrolladores ofrecen el videojuego libremente, es legal tener una ROM y jugarla.

También es legal tener una copia de seguridad hecha por nosotros mismos de algún videojuego, y jugarla en un emulador. Es legal tener una ROM de un videojuego que en su forma original tuviese algún *bug* o fallo que impida disfrutarlo, siempre y cuando lleguemos a creer que la ROM sin el fallo la hayamos arreglado nosotros.

Si hemos comprado la ROM al propietario, también es legal jugarlo en cualquier plataforma, hay colecciones de ROMs en plataformas de distribución digital como Steam [19] en las que los mismos distribuidores guían a los consumidores sobre cómo usar la ROM en otras plataformas vía emulación.
En todos los demás casos, es ilegal poseer una ROM de un videojuego, y la multa por hacerlo se agrava cuanto más actual sea el juego y cuanto más al público lo ofrezcamos.

### 7.3.3. Abandonware, freeware

Con la llegada de los emuladores, mucha gente se ha entretenido programando y diseñando videojuegos para videoconsolas antiguas u obsoletas. La mayoría de gente lo hace por hobby y suele ofrecer la ROM en páginas especializadas [3] [20] [21]. En otros casos, programadores han pedido permiso a los desarrolladores originales para hacer un port a alguna videoconsola obsoleta o poco conocida, ofreciendo el código o la ROM gratuitamente, y se lo han concedido.

### 7.4. Planificación temporal

El tiempo aproximado para la realización de este proyecto es de 4 meses, comprendidos entre Septiembre de 2017 y Enero de 2018. Este documento presenta la planificación del proyecto, pero, a medida que se vaya realizando el mismo se pueden producir cambios en la planificación, para conseguir terminar el proyecto en el tiempo previsto. Separamos la planificación temporal en dos principales etapas, de los 18 créditos del TFG, 3 pertenecen a GEP y 15 a la realización del proyecto.

Debido a la extensión del proyecto, se han alterado algunos datos de la planificación, por ejemplo, la duración del proyecto ha sido de 7 meses, comprendidos entre Septiembre de 2017 y Abril de 2018. No obstante, la mayoría de planificación no se ha visto alterada, ya que el número de horas invertidas ha sido el mismo, únicamente más esparcidas.

### 7.4.1. Especificación del trabajo

En este apartado describimos al detalle los diferentes apartados del proyecto.

#### 7.4.1.1. Gestión del proyecto

El trabajo realizado en la asignatura de GEP, correspondiente a 3 créditos, por tanto 90 horas. GEP consiste en la preparación y entrega de 6 documentos, separados por horas en la Tabla 1.

<table>
<thead>
<tr>
<th>Documento</th>
<th>Horas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alcance del proyecto y contextualización</td>
<td>20</td>
</tr>
<tr>
<td>Planificación temporal</td>
<td>20</td>
</tr>
<tr>
<td>Gestión económica y sostenibilidad</td>
<td>20</td>
</tr>
<tr>
<td>Presentación preliminar</td>
<td>10</td>
</tr>
<tr>
<td>Condiciones de especialidad</td>
<td>10</td>
</tr>
<tr>
<td>Documento final, presentación oral</td>
<td>10</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>90</strong></td>
</tr>
</tbody>
</table>

Tabla 1: .
7.4.1.2. Realización del proyecto
Los 15 créditos, 450 horas, correspondientes a la elaboración del proyecto. Sumados a las 90 horas de GEP, nos da un total de entre **540 horas**.
Calculo que cada mes haré 20 horas de memoria, esto puede variar dependiendo del trabajo hecho durante la semana.

7.4.1.2.1. Compra y testeo de componentes
Trabajo realizado durante los meses de Septiembre y Octubre, hay que pensar en los componentes necesarios para llevar a cabo el proyecto, además, habrá que hacer comparaciones en diferentes tiendas, y habrá que comprar los componentes para poder llevar a cabo el proyecto. Las etapas se muestran en la Tabla 2.

<table>
<thead>
<tr>
<th>Etapa</th>
<th>Dependencias</th>
<th>Horas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pensar en los componentes</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td>Buscar los componentes</td>
<td>Pensar en los componentes</td>
<td>30</td>
</tr>
<tr>
<td>en diferentes tiendas</td>
<td>necesarios</td>
<td></td>
</tr>
<tr>
<td>Compra de componentes</td>
<td>Buscar los componentes</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>en diferentes tiendas</td>
<td></td>
</tr>
<tr>
<td>Testeo de componentes</td>
<td>Compra de componentes</td>
<td>20</td>
</tr>
<tr>
<td>Memoria</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>

Tabla 2: Compra y testeo de componentes

7.4.1.2.2. Testeo de componentes y decisiones
Trabajo realizado durante el mes de Noviembre, se prueban las diferentes Raspberry Pi y emuladores para elegir los mejores, se toma una decisión final, se diseña el cableado y se realiza la memoria. Las etapas se muestran en la Tabla 3.

<table>
<thead>
<tr>
<th>Etapa</th>
<th>Dependencias</th>
<th>Horas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pruebas con diferentes</td>
<td>Testeo de componentes</td>
<td>40</td>
</tr>
<tr>
<td>Raspberry y emuladores</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decisión final</td>
<td>Pruebas con diferentes</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>SO y emuladores</td>
<td></td>
</tr>
<tr>
<td>Diseño del cableado</td>
<td>Decisión Final</td>
<td>30</td>
</tr>
<tr>
<td>Memoria</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>

Tabla 3: Testeo de componentes y decisiones

7.4.1.2.3. Cableado y programación del firmware
Cableado y programación del firmware: Etapa realizada durante los meses de Diciembre, Enero y Febrero, aquí realizamos el cableado diseñado previamente y empezamos a programar, finalmente se prueba el trabajo realizado. Las etapas se muestran en la Tabla 4.
<table>
<thead>
<tr>
<th>Etapa</th>
<th>Dependencias</th>
<th>Horas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cableado</td>
<td>Diseño del cableado</td>
<td>30</td>
</tr>
<tr>
<td>Programación del firmware</td>
<td>Decisión Final</td>
<td>50</td>
</tr>
<tr>
<td>Testeo del firmware</td>
<td>Programación del firmware, Cableado</td>
<td>20</td>
</tr>
<tr>
<td>Memoria</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>120</td>
</tr>
</tbody>
</table>

Tabla 4: Cableado y programación del firmware

7.4.1.2.4. Retoques y housing

Etapa realizada durante los meses de Febrero y Marzo, en la que se hace un testeo final, haciendo todos los cambios necesarios. Si hubiesen muchos cambios, subetapas de esta etapa se moverían al mes de Abril, concretamente la revisión de la memoria y la realización de la presentación. Las etapas se muestran en la Tabla 5.

<table>
<thead>
<tr>
<th>Etapa</th>
<th>Dependencias</th>
<th>Horas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testeo Final</td>
<td>Testeo del firmware</td>
<td>40</td>
</tr>
<tr>
<td>Diseño del <em>housing</em></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Fabricación y testeo del <em>housing</em></td>
<td>Diseño del <em>housing</em></td>
<td>10</td>
</tr>
<tr>
<td>Revisión de la memoria</td>
<td>Memoria (todas las etapas)</td>
<td>50</td>
</tr>
<tr>
<td>Preparación de la presentación</td>
<td>Revisión Memoria</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>140</td>
</tr>
</tbody>
</table>

Tabla 5: Retoques

7.4.2. Diagrama de Gantt

A continuación se adjunta el diagrama de Gantt, que muestra las diferentes etapas, subetapas y las semanas en los que está pensado hacerlas. Gracias al Gantt tenemos una mejor visibilidad de la planificación del proyecto. También nos permite decidir si invertir más tiempo o no en alguna etapa, viendo la semana en la que estamos y las etapas hechas hasta la fecha.
7.5. Gestión económica

En este apartado se muestra una tabla para cada tipo de recurso junto con sus gastos, también se revisan los imprevistos y el presupuesto total.

7.5.1. Recursos Humanos

En la Tabla 6 se tratan los costes del tiempo que han tardado las personas en llevar el proyecto a cabo.

<table>
<thead>
<tr>
<th>Dedicación(horas)</th>
<th>Precio(€/hora)</th>
<th>Coste Total(€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Director de proyecto</td>
<td>30</td>
<td>32.29</td>
</tr>
<tr>
<td>Responsable de lab.</td>
<td>6</td>
<td>21.48</td>
</tr>
<tr>
<td>Becario</td>
<td>450</td>
<td>14.84</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tabla 6: Costes de recursos humanos.

7.5.2. Recursos Software

En la Tabla 7 se tratan los costes del software necesario para llevar el proyecto a cabo. Como se puede observar, la mayoría del software es libre, o podría serlo, por lo que no tiene coste.
Las siglas PDC indican Precio de Compra

<table>
<thead>
<tr>
<th>Programa</th>
<th>Vida(años)</th>
<th>Uso(horas)</th>
<th>Amor.</th>
<th>PDC(€)</th>
<th>Precio(€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu</td>
<td>3</td>
<td>200</td>
<td>0.03</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>RetroPie</td>
<td>3</td>
<td>50</td>
<td>0.01</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Eagle</td>
<td>3</td>
<td>20</td>
<td>0.00</td>
<td>400</td>
<td>1.33</td>
</tr>
<tr>
<td>SketchUp</td>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>MPLAB</td>
<td>3</td>
<td>80</td>
<td>0.01</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>GitHub</td>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Trello</td>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Google Drive</td>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Android</td>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Overleaf</td>
<td>3</td>
<td>50</td>
<td>0.01</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>1.33 €</strong></td>
</tr>
</tbody>
</table>

Tabla 7: Costes de recursos software.

7.5.3. Recursos Hardware

En la Tabla 8 se tratan todas esas herramientas y máquinas que tienen una vida útil más grande que el tiempo de uso que le hemos dado, por tanto, indicamos también la amortización del producto y el precio en base a la amortización.
<table>
<thead>
<tr>
<th>Herramientas</th>
<th>Vida(años)</th>
<th>Uso(h)</th>
<th>Amor.</th>
<th>Tienda</th>
<th>PDC(€)</th>
<th>Precio(€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>kit Soldador</td>
<td>4</td>
<td>60</td>
<td>0.01</td>
<td>Amazon</td>
<td>21.59</td>
<td>0.16</td>
</tr>
<tr>
<td>Alicates</td>
<td>4</td>
<td>3</td>
<td>0.00</td>
<td>Aliexpress</td>
<td>1.38</td>
<td>0.00</td>
</tr>
<tr>
<td>Herr. WireWrap</td>
<td>4</td>
<td>10</td>
<td>0.00</td>
<td>Aliexpress</td>
<td>10.25</td>
<td>0.01</td>
</tr>
<tr>
<td>Multímetro</td>
<td>4</td>
<td>10</td>
<td>0.00</td>
<td>Amazon</td>
<td>7.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Bolígrafo</td>
<td>4</td>
<td>2</td>
<td>0.00</td>
<td>Aliexpress</td>
<td>0.46</td>
<td>0.00</td>
</tr>
<tr>
<td>Teclado y ratón</td>
<td>4</td>
<td>300</td>
<td>0.04</td>
<td>Amazon</td>
<td>24.99</td>
<td>1.00</td>
</tr>
<tr>
<td>Protoboard</td>
<td>4</td>
<td>200</td>
<td>0.04</td>
<td>Amazon</td>
<td>3.15</td>
<td>0.13</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>56.67€</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Máquinas</th>
<th>Vida(años)</th>
<th>Uso(h)</th>
<th>Amor.</th>
<th>Tienda</th>
<th>PDC(€)</th>
<th>Precio(€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smartphone</td>
<td>1</td>
<td>10</td>
<td>0.00</td>
<td>Google</td>
<td>340.00</td>
<td>1.69</td>
</tr>
<tr>
<td>Ordenador</td>
<td>3</td>
<td>300</td>
<td>0.05</td>
<td>Amazon</td>
<td>499.00</td>
<td>24.95</td>
</tr>
<tr>
<td>Pantalla</td>
<td>3</td>
<td>300</td>
<td>0.05</td>
<td>Amazon</td>
<td>499.00</td>
<td>24.95</td>
</tr>
<tr>
<td>Osciloscopio</td>
<td>3</td>
<td>60</td>
<td>0.01</td>
<td>Amazon</td>
<td>136.99</td>
<td>1.36</td>
</tr>
<tr>
<td>Impresora 3D</td>
<td>3</td>
<td>8</td>
<td>0.00</td>
<td>Aliexpress</td>
<td>155.17</td>
<td>0.21</td>
</tr>
<tr>
<td>Raspberry Pi 0</td>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>Amazon</td>
<td>25.50</td>
<td>0.04</td>
</tr>
<tr>
<td>Raspberry Pi</td>
<td>3</td>
<td>10</td>
<td>0.00</td>
<td>Amazon</td>
<td>30.00</td>
<td>0.05</td>
</tr>
<tr>
<td>Raspberry Pi 3</td>
<td>3</td>
<td>300</td>
<td>0.05</td>
<td>Amazon</td>
<td>34.90</td>
<td>1.75</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>82.79€</td>
<td></td>
</tr>
</tbody>
</table>

Tabla 8: Lista de máquinas y herramientas usadas.

7.5.4. Consumibles

En la Tabla 9 se tratan los costes de consumibles, es decir, todos los componentes que se van a utilizar únicamente para el proyecto.

<table>
<thead>
<tr>
<th>Consumible</th>
<th>Tienda</th>
<th>Precio(€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC24F16KA101</td>
<td>Amazon</td>
<td>7.25</td>
</tr>
<tr>
<td>Kit Electrónica</td>
<td>Aliexpress</td>
<td>8.96</td>
</tr>
<tr>
<td>Botones y Joystick</td>
<td>Aliexpress</td>
<td>11.08</td>
</tr>
<tr>
<td>WireWrap</td>
<td>Aliexpress</td>
<td>7.42</td>
</tr>
<tr>
<td>Placa de topos</td>
<td>Amazon</td>
<td>6.99</td>
</tr>
<tr>
<td>Cables</td>
<td>Amazon</td>
<td>7.09</td>
</tr>
<tr>
<td>Alimentación</td>
<td>Amazon</td>
<td>20.00</td>
</tr>
<tr>
<td>Tarjeta SD</td>
<td>Amazon</td>
<td>14.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>82.79€</td>
</tr>
</tbody>
</table>

Tabla 9: Lista de consumibles usados.

7.5.5. Costes indirectos

En la Tabla 10 se tratan los costes indirectos, como podría ser el alquiler del piso, el alquiler de un aula para reunirme con el director, el gasto de electricidad de un ordenador, de la Raspberry Pi, y otros gastos, como tener que comer en la universidad debido a haberse reunido con el director.
### Tabla 10: Costes indirectos.

<table>
<thead>
<tr>
<th>Tipo</th>
<th>Precio(€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aula FIB</td>
<td>100</td>
</tr>
<tr>
<td>T-jove 1 zona</td>
<td>35</td>
</tr>
<tr>
<td>Alquiler</td>
<td>300</td>
</tr>
<tr>
<td>Electricidad</td>
<td>60</td>
</tr>
<tr>
<td>Otros gastos</td>
<td>80</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2,265.00 €</strong></td>
</tr>
</tbody>
</table>

### 7.5.6. Presupuesto final

Finalmente se muestra la suma de todos los costes agrupados en la Tabla 11.

<table>
<thead>
<tr>
<th>Tipo</th>
<th>Precio(€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursos Humanos</td>
<td>7,775.58</td>
</tr>
<tr>
<td>Recursos Software</td>
<td>1.33</td>
</tr>
<tr>
<td>Recursos Hardware</td>
<td>56.67</td>
</tr>
<tr>
<td>Consumibles</td>
<td>82.79</td>
</tr>
<tr>
<td>Costes indirectos</td>
<td>2,265.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>10,181.37 €</strong></td>
</tr>
</tbody>
</table>

Tabla 11: Coste total del proyecto.

Vemos que el coste total ronda los 10,000 euros, y que los campos más costosos son los de recursos humanos y los costes indirectos, además, estos campos ya se han ajustado bastante, por lo que no los vamos a poder abaratar.

### 7.5.7. Imprevistos y contingencias

Los posibles errores son mal diseño de cableado y firmware mal programado. Ambos errores conducen al mismo problema: Quemar algún pin de la Raspberry Pi 3 B o del PIC24F16KA101.

Quemar un pin de la Raspberry Pi es menos problemático, ya que tiene muchos y se pueden usar cambiando un poco el código. Tenemos los pines del PIC muy justos para lo que queremos diseñar, sobran exactamente dos pines, por lo que si cometemos más de dos fallos podríamos tener problemas. En el caso de quemar el chip entero, claramente, es más problemático quemar la Raspberry Pi que el PIC, ya que la primera es más costosa.

Para solventar estos problemas, disponemos del osciloscopio y del conocimiento sobre electrónica obtenido en las asignaturas de la especialidad.

Se estima un porcentaje de contingencia del 5%, ya que el presupuesto está muy detallado y muchos de los recursos usados no tienen coste.
7.6. Sostenibilidad y compromiso social

En la Tabla 12 se muestra la matriz de sostenibilidad del proyecto, en los siguientes apartados se profundizan los aspectos ambientales, económicos y sociales del mismo.

<table>
<thead>
<tr>
<th></th>
<th>PPP</th>
<th>Vida útil</th>
<th>Riesgos</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ambiental</strong></td>
<td>Consumo del diseño</td>
<td>Huella ecológica</td>
<td>Ambientales</td>
</tr>
<tr>
<td></td>
<td>10:10</td>
<td>20:20</td>
<td>0:0</td>
</tr>
<tr>
<td><strong>Económico</strong></td>
<td>Factura</td>
<td>Plan de viabilidad</td>
<td>Económicos</td>
</tr>
<tr>
<td></td>
<td>10:10</td>
<td>20:20</td>
<td>0:0</td>
</tr>
<tr>
<td><strong>Social</strong></td>
<td>Impacto personal</td>
<td>Impacto social</td>
<td>Sociales</td>
</tr>
<tr>
<td></td>
<td>10:10</td>
<td>10:20</td>
<td>-5:0</td>
</tr>
<tr>
<td><strong>Rango sostenibilidad</strong></td>
<td>30:30</td>
<td>50:60</td>
<td>-5:0</td>
</tr>
<tr>
<td></td>
<td>75:90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tabla 12: Matriz de sostenibilidad.

Los aspectos ambientales y económicos son altos debido a que utilizamos microprocesadores, de bajo coste energético y económico. El rango de sostenibilidad tiene una puntuación de 75 sobre 90.

7.6.1. Dimensión Ambiental

En este proyecto se trabaja con el hardware más económico y de los que menos consumen. Se utilizan una Raspberry Pi y un PIC, ambos con un consumo inferior a 5 W. Además, los componentes que utilizamos en el proyecto son los que el usuario se lleva finalmente a casa.

La alternativa a mi proyecto es usar una máquina arcade, que consume mucho más, u otro emulador con un hardware similar, que tendría unos costes energéticos similares, por tanto, se disminuye la huella ecológica usando mi producto.

Hay algunos componentes que van a sobrar, como por ejemplo el cable de WireWrap, que se utilizará para otros proyectos. También hay otros componentes que se reutilizarán, como el caso de la pantalla, el teclado y el ratón.

Si queremos desmantelar por completo mi proyecto, la Raspberry Pi y el PIC se pueden reutilizar totalmente.

7.6.2. Dimensión Económica

En la gestión del proyecto se ha realizado una tabla detallada con todos los costes, también se tienen en cuenta imprevistos y planes de contingencia.

El proyecto tiene competidores, ya existen prototipos parecidos al mío. El proyecto se podría hacer en menos tiempo comprando algunos componentes directamente con el firmware instalado, pero parte de la magia y la gracia del proyecto está en hacerlo desde cero. Además, probablemente comprar estos componentes acabaría
siendo más caro que el enfoque actual.

Se podrían abaratar costes si quisiéramos realizar el proyecto a gran escala, pidiendo placas con el firmware programado a China, pero el prototipo propuesto en este proyecto ya tiene un coste lo suficientemente bajo como para no ser un problema.

7.6.3. Dimensión Social

España se encuentra en un leve crecimiento económico, saliendo de la crisis de 2008. No obstante, la economía del país es claramente mejorable, al igual que la calidad de vida. En el aspecto político, estamos en una situación difícil, ya que Cataluña quiere independizarse de España, y el estado ha actuado con políticas de suspensión de derechos (Artículo 155) y con brutalidad policial en la comunidad autónoma de Cataluña.

En España, la industria del videojuego es considerada como una industria cultural, por tanto, este proyecto tiene el claro objetivo de mejorar la calidad de vida de la gente, permitiendo a jóvenes y a mayores disfrutar de juegos antiguos y crear lazos y amistad. En Cataluña hay mucha cultura del videojuego retro o arcade, por tanto, este proyecto ayuda a las comunidades que quieran fabricar una máquina arcade compacta de bajo coste.

![Figura 10: Retrobarcelona, evento dedicado a la informática y al videojuego clásico en Barcelona.](image)

A nivel global, este proyecto también ayuda, ya que, aunque en otros países los videojuegos no sean considerados cultura o arte, también hay un inmenso público que dedica su tiempo a entretererse con este maravilloso ocio. Este proyecto también puede ayudar a gente que esté empezando a aprender electrónica y programación a bajo nivel, ya que el código utilizado no es muy complejo, utilizando la excusa de crear una máquina arcade la gente aprendería a soldar, programar y Linux.
Hay organizaciones en contra de los videojuegos, estas organizaciones verían este proyecto como algo malo o nocivo para la sociedad.

8. Resultados

Una vez acabado el proyecto miramos si hemos cumplido los objetivos propuestos al principio del proyecto.

La Raspberry Pi emula numerosos juegos de diversas consolas gracias a RetroPie y retroarch, la mayoría de juegos tienen ausencia de lag, aunque es posible encontrar lag si se juega a algún videojuego de consolas actuales o potentes. Como en principio la idea era emular videojuegos arcade antiguos, estos dos puntos se han cumplido correctamente. Todos los juegos que he probado funcionan a 30FPS o más.

Se ha fabricado un mando o controlador con joystick y 10 botones utilizando un PIC, que comunica la entrada de datos a la Raspberry Pi usando el protocolo I2C, sin problemas de input lag, rebotes o pérdida de input. El tiempo de respuesta desde que se aprieta un botón hasta que obtenemos el resultado es de unos 50 mili segundos, muy parecido al de un teclado convencional.

El proyecto actual está hecho para un solo jugador, pero el diseño permite en principio varios jugadores, se explica en el apartado 9.1.

A continuación mostramos varias imágenes de los resultados.

Figura 11: El programa detecta nuestro mando como un mando de XBOX 360.
Figura 12: Versión final, a falta de conseguir un *housing*.
Figura 13: Al apretar un botón, hacemos que el personaje salte.
También mostramos un esquema del software que se ejecuta en la Raspberry Pi y el PIC desde que se encienden hasta que el usuario puede jugar.

Figura 14: Esquema del software

9. Conclusiones

Este proyecto me ha ayudado a consolidar muchos aspectos de la carrera, además he podido trabajar en un proyecto interesante pensado por mí, haciendo que las dificultades fuesen mucho más amenas. Tengo pensado seguir mejorando un poco la máquina arcade y utilizarla para jugar de vez en cuando. También me ha gustado mucho trabajar con la Raspberry Pi, probablemente la utilice para futuros proyectos, ya que ofrece numerosas posibilidades.

9.1. Posibles ampliaciones

Hay dos ampliaciones para el proyecto que tengo claras y que van algo ligadas, la primera es cómo permitir varios jugadores y la segunda es cómo adecuar el housing para prototipos de varios jugadores.

9 El código actual es para un único jugador, pero si quisiéramos ampliarlo a varios,
el proceso a seguir sería el siguiente:

- Comprar PICs adicionales, ya que cada mando/jugador necesita un PIC, también 10 botones y un joystick por jugador.

- Asignar una dirección de esclavo diferente a cada mango en el código del PIC, es el único cambio a hacer en este código.

- En el código de la Raspberry Pi, se puede resolver el problema de varias maneras, la que parece más sencilla sería la siguiente: hace falta añadir tantos buffers y dispositivos uinput como jugadores haya. En el momento de hacer la petición por I2C, la Raspberry Pi debería iterar por cada jugador antes de escribir el input en el dispositivo correspondiente.

- En el script rc.local haría falta crear un joystick por cada eventX que se haya generado, utilizando el comando xboxdrv.

A la hora de adecuar el housing, no hay más remedio que tener tres prototipos diferentes, para uno, dos o cuatro jugadores.
Referencias


Anexo

A. newmainXC16.c

/*
 * File: newmainXC16.c
 * Author: Kilian
 */

// FBS
#pragma config BWRP = OFF // Boot segment may be written
#pragma config BSS = OFF // No boot program Flash segment
// FGS
#pragma config GWRP = OFF // General segment may be written
#pragma config GCP = OFF // General Segment Code Protection bit
// FOSCSEL
#pragma config FNOSC = FRCDIV // 8 MHz FRC oscillator with divide-by-N (FRCDIV)
#pragma config IESO = OFF // Internal External Switchover disabled
// FOSC
#pragma config POSCMOD = NONE // Primary oscillator disabled
#pragma config OSCIOFNC = ON // CLKO output disabled
#pragma config POSCFREQ = HS // Primary Oscillator Frequency Configuration
#pragma config SOSCSEL = SOSCHP // Secondary oscillator high-power
#pragma config FCKSM = CSDCMD // Clock Switching and Monitor
// FWDT
#pragma config WDTPS = PS32768 // Watchdog Timer Postscale Select bits
#pragma config FWPSA = PR128 // WDT Prescaler ratio
#pragma config WINDIS = OFF // Standard WDT selected
#pragma config FWDTEN = OFF // WDT disabled
// FPOR
#pragma config BOREN = BOR0 // Brown-out Reset disabled;
#pragma config PWRTEN = OFF // Power-up Timer Enable bit
#pragma config I2C1SEL = PRI // Alternate I2C1 Pin Mapping bit
#pragma config BORV = V18 // Brown-out Reset Pin Mapping bit
#pragma config MCLRE = ON // MCLR pin enabled
// FICD
#pragma config ICS = PGx1 // PGC1/PGD1 are used for programming and debugging
// FDS
#pragma config DSWDTPS = DSWDTPSF // Deep Sleep Watchdog Timer Postscale bit
#pragma config DSWDTOSC = LPRC // DSWDT uses LPRC
#pragma config RTCOSC = SOSC // RTCC uses SOSC
#pragma config DSBOREN = OFF // Deep Sleep BOR disabled
#pragma config DSWDTEN = OFF // Deep Sleep Watchdog Timer Enable bit
```c
#define USE_AND_OR

#include "xc.h"
#include <p24F16KA101.h>
#include <i2c.h>

volatile char write_byte;
#define Fosc (80000000) // 8MHz crystal
#define Fcy (Fosc*4/2) // w.PLL (Instruction Per Second)
#define Fsck 4000000 // 400 KHz I2C
#define I2C1_BRG ((Fcy/2/Fsck)-1)
#define SLAVE_ADD 15 // slave address

typedef enum {
   STATE_WAIT_FOR_ADDR,
   STATE_WAIT_FOR_WRITE_DATA,
   STATE_SEND_READ_DATA,
   STATE_SEND_READ_LAST
} STATE;
volatile STATE e_mystate = STATE_WAIT_FOR_ADDR;

#define BUFFSIZE 15
volatile char read_buffer[BUFFSIZE]; // master write slave read
volatile char write_buffer[BUFFSIZE]; // master read slave write
volatile uint16_t buffer_index;

void __attribute__((__interrupt__, no_auto_psv )) _SI2C1Interrupt(void) {
   uint8_t u8_c;
   IFS1bits.SI2C1IF = 0;

   switch (e_mystate) {
   case STATE_WAIT_FOR_ADDR:
      u8_c = I2C1RCV; // clear RBF bit for address
      buffer_index = 0;

      if (I2C1STATbits.R_W) {
         // check the R/W bit and see if read or write transaction
         I2C1TRN = write_buffer[buffer_index]; // send first data byte
         buffer_index++;
         I2C1CONbits.SCLREL = 1;
         // release clock line so MASTER can drive it
         e_mystate = STATE_SEND_READ_DATA; // read transaction
      }
      else e_mystate = STATE_WAIT_FOR_WRITE_DATA;
      break;
```

case STATE_WAIT_FOR_WRITE_DATA:
    // character arrived, place in buffer
    read_buffer[buffer_index] = I2C1RCV;  // read the byte
    if (read_buffer[buffer_index] == 0) {
        e_mystate = STATE_WAIT_FOR_ADDR;
        // wait for next transaction
    }
    break;

case STATE_SEND_READ_DATA:
    I2C1TRN = write_buffer[buffer_index];
    buffer_index++;
    I2CICONbits.SCLREL = 1;
    // release clock line so MASTER can drive it
    if (write_buffer[buffer_index] == -1) e_mystate = STATE_SEND_READ_LAST;
    // last character, release the clock line again
    break;

case STATE_SEND_READ_LAST:
    // this is interrupt for last character finished shifting out
    e_mystate = STATE_WAIT_FOR_ADDR;
    break;

default:
    e_mystate = STATE_WAIT_FOR_ADDR;
}

int main (void) {
    unsigned int i;
    for(i = 0; i < BUFSIZE; ++i) {
        read_buffer[i] = 0;
        write_buffer[i] = 0;
    }
    read_buffer[BUFSIZE - 1] = -1;
    write_buffer[BUFSIZE - 1] = -1;
//disable modules to make buttons work
AD1PCFG = 0b1101110000111111;
PMD1 = 0b0011110001101001;
PMD2 = 0b0000000000000001;
PMD3 = 0b0000011010000000;
PMD4 = 0b0000000000000110;
IEC0 = 0x0000;
IEC1 = 0x0000;
IEC3 = 0x0000;
IEC4 = 0x0000;
CTMUCONbits.CTMUEN = 0;
OSCCONbits.SOSCEN = 0;
REFOCONbits.ROEN = 0;
RCFGCALbits.RTCEN = 0;
RCFGCALbits.RTCOE = 0;
SPI1STATbits.SPIEN = 0;
SPI1CON1bits.DISSCK = 1;
SPI1CON1bits.DISSDO = 1;
OC1CONbits.OCM2 = 0;
OC1CONbits.OCM1 = 0;
OC1CONbits.OCM0 = 0;
HLVDCONbits.HLVDEN = 0;
CTMUCONbits.CTMUEN = 0;

TRISA = 0xFFFF;
TRISB = 0xFFFF;

I2C1CONbits.I2CEN = 1;
I2C1CONbits.I2CSIDL = 0;
I2C1CONbits.IPMIEN = 0;
I2C1CONbits.A1OM = 0;
I2C1CONbits.DISSLW = 1;
I2C1CONbits.SMEN = 0;
I2C1CONbits.GCEN = 0;
I2C1CONbits.STREN = 1;

I2C1BRG = I2C1_BRG;
I2C1ADD = SLAVE_ADD;

IFS1bits.SI2C1IF = 0;
IPC4bits.SI2C1P2 = 1;
IPC4bits.SI2C1P1 = 1;
IPC4bits.SI2C1P0 = 1;
IEC1bits.SI2C1IE = 1;
while(1) {
    /******** Main Buttons ********/
    write_buffer[0] = PORTAbits.RA1;
    write_buffer[1] = PORTBbits.RB0;
    write_buffer[2] = PORTBbits.RB1;
    write_buffer[3] = PORTBbits.RB2;
    write_buffer[4] = PORTAbits.RA2;
    write_buffer[5] = PORTAbits.RA3;
    write_buffer[6] = PORTBbits.RB4;
    write_buffer[7] = PORTAbits.RA4;
    /********* Joystick ************/
    write_buffer[8] = PORTBbits.RB15;
    write_buffer[9] = PORTBbits.RB14;
    write_buffer[10] = PORTBbits.RB13;
    /******** Start+Select ********/
    write_buffer[12] = PORTAbits.RA6;
    write_buffer[13] = PORTBbits.RB7;
}
return 0;
}
B. uinput.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <linux/input.h>
#include <linux/uinput.h>
#include <bcm2835.h>

#define die(str, args...) do { 
   perror(str); 
   exit(EXIT_FAILURE); 
} while(0)

// I2C SETTINGS
#define MODE_READ 0
#define MODE_WRITE 1
#define MAX_LEN 14

typedef enum {
   NO_ACTION,
   I2C_BEGIN,
   I2C_END
} i2c_init;

uint8_t init = I2C_BEGIN;
uint16_t clk_div = BCM2835_I2C_CLOCK_DIVIDER_626;
uint8_t slave_address = 0x0F;
uint32_t len = MAX_LEN;
uint8_t mode = MODE_READ;

char buf[MAX_LEN];
int i = 0;
uint8_t data;
// END OF I2C SETTINGS

41
/ get i2C data on buf[]
void get_i2c() {
    for(i = 0; i < MAX_LEN; ++i) buf[i] = 0;

    if(init == I2C_BEGIN) {
        if(!bcm2835_i2c_begin()) {
            printf("begin failed, root\n");
        }
    }

    bcm2835_i2c_setSlaveAddress(slave_address);
    bcm2835_i2c_setClockDivider(clk_div);

    for(i = 0; i < MAX_LEN; ++i) {
        data = bcm2835_i2c_read(buf, len);
    }

    if(init == I2C_END) bcm2835_i2c_end();
}

void emit(int fd, int type, int code, int val) {
    // send a key stroke to OS

    struct input_event ie;
    ie.type = type;
    ie.code = code;
    ie.value = val;
    // timestamp values below are ignored
    ie.time.tv_sec = 0;
    ie.time.tv_usec = 0;

    if(write(fd, &ie, sizeof(ie)) == -1) die("error: write");
}

int main(void) {

    if(!bcm2835_init()){
        printf("init failed, root\n");
        return 1;
    }

    // SETTING UINPUT
    struct uinput_user_dev usetup;
    int fd = open("/dev/uinput", 0_RDONLY | O_NONBLOCK);
    if(fd < 0) die("error: open");
// add keystroke and repeat events
if(ioctl(fd, UI_SET_EVBIT, EV_KEY) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_EVBIT, EV_REP) == -1) die("error: ioctl");
// add joystick and button keys
if(ioctl(fd, UI_SET_KEYBIT, KEY_W) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_A) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_S) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_D) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_0) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_1) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_2) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_3) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_4) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_5) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_6) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_7) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_8) == -1) die("error: ioctl");
if(ioctl(fd, UI_SET_KEYBIT, KEY_9) == -1) die("error: ioctl");

// set device information
memset(&usetup, 0, sizeof(usetup));
usetup.id.bustype = BUS_USB;
usetup.id.vendor = 0x0001; /* sample vendor */
usetup.id.product = 0x0002; /* sample product */
usetup.id.version = 0x0003; /* sample version */
strcpy(usetup.name, "Arcade Machine Analog and Buttons");

if(write(fd, &usetup, sizeof(usetup)) == -1) die("error: write");
if(ioctl(fd, UI_DEV_CREATE) == -1) die("error: ioctl");
sleep(2);
while(1) {
    // get PIC data into buf[]
    get_i2c();
    // send all key status
    emit(fd, EV_KEY, KEY_0, buf[0]);
    emit(fd, EV_KEY, KEY_1, buf[1]);
    emit(fd, EV_KEY, KEY_2, buf[2]);
    emit(fd, EV_KEY, KEY_3, buf[3]);
    emit(fd, EV_KEY, KEY_4, buf[4]);
    emit(fd, EV_KEY, KEY_5, buf[5]);
    emit(fd, EV_KEY, KEY_6, buf[6]);
    emit(fd, EV_KEY, KEY_7, buf[7]);
    emit(fd, EV_KEY, KEY_8, buf[8]);
    emit(fd, EV_KEY, KEY_9, buf[9]);
    emit(fd, EV_KEY, KEY_W, buf[10]);
    emit(fd, EV_KEY, KEY_A, buf[11]);
    emit(fd, EV_KEY, KEY_S, buf[12]);
    emit(fd, EV_KEY, KEY_D, buf[13]);
    // report all keys at once
    emit(fd, EV_SYN, SYN_REPORT, 0);
    // wait some time until next poll
    delay(45);
}

sleep(1);

if(ioctl(fd, UI_DEV_DESTROY) == -1) die("error: ioctl");
close(fd);
    bcm2835_close();

return 0;
C. rc.local

#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
#
# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

sudo /home/pi/ArcadeMachine-TFG/.uinput2 &
sleep 2

sudo /opt/retropie/supplementary/xboxdrv/bin/xboxdrv \
  --evdev /dev/input/event2 \  
  --silent \  
  --detach-kernel-driver \  
  --force-feedback \  
  --deadzone-trigger 15\% \  
  --deadzone 4000 \  
  --mimic-xpad \  
  --dpad-as-button \  
  --evdev-keymap KEY_W=du,KEY_A=dl,KEY_S=dd,KEY_D=dr,KEY_1=start, \  
     KEY_2=back,KEY_3=a,KEY_4=b,KEY_5=x,KEY_6=y, \  
     KEY_7=1b,KEY_8=rb,KEY_9=t1,KEY_0=tr \  
  &

exit 0