Análisis de rendimiento de aplicaciones en Arduino

Adrián Rivera Sollai - NIU: 1526394

2 de julio de 2024

Resumen– En este proyecto se presenta un caso de uso de estudio de una aplicación, en concreto el videojuego Pong, para determinar si ésta se puede ejecutar en un sistema embebido formado por la placa Arduino Mega 2560 y una pantalla TFT ST7735. Para ello, primero se realiza una introducción y un estudio del estado del arte de los sistemas embebidos, seguidamente se plantean los objetivos de llevar a cabo una descripción del Hardware o Arquitectura, una definición de los requisitos de la aplicación, un análisis del Software y un análisis del rendimiento de ésta. Finalmente, se mostrará la integración del sistema, los resultados obtenidos y la conclusión.

Parabras Clave– Arduino, Atmega 2560, SPI, ST7735, rendimiento, Pong, Hardware, Software, aplicación

Abstract– This project presents a use case of a study of an application, specifically a video game, to determine if it can be executed on an embedded system formed by the Arduino Mega 2560 board and a TFT ST7735 Display. To do this, first an introduction and a study of the state of the art of embedded systems are carried out, then the objectives of carrying out a Hardware or Architecture's description, an application requirements' definition, a Software's analysis and a performance's analysis are set out. Finally, the system's integration, the results obtained and the conclusion will be shown.

Keywords– Arduino, Atmega 2560, SPI, ST7735, performance, Pong, Hardware, Software, application

1 Introducción

A principio de la década de 1940, surgió el interés por desarrollar métodos veloces para llevar a cabo cálculos, lo que llevó a investigar el desarrollo de los ordenadores digitales[1, p.53]. Los primeros ordenadores digitales estaban formados por varios tubos de vacío, los cuales se usaban para guardar información binaria. Un ejemplo de este tipo de arquitectura es el Atanasoff-Berry computer creado en la Universidad de Iowa en 1942[1, p.4].

A finales de la década de 1950, los tubos de vacíos serían sustituidos por los transistores, los cuales eran más pequeños y tenían un consumo reducido de energía[1, p.4]. A principios de la década de 1960, llegarían los circuitos integrados, permitiendo fabricar los transistores y otros componentes a la vez en una base semiconductora, lo que permite reducir el tamaño respeto utilizar piezas individuales[1, p.6]. No obstante, se diseñaban según el producto al cual iban dirigido, limitando sus aplicaciones. Finalmente, en 1971 Intel crearía un chip de propósito general que se acabaría llamando microprocesador[1, p.7].

Dentro de este contexto, ya se habían llevado a cabo proyectos que consistían en embeber ordenadores[2, p.2]: El primer ordenador diseñado para llevar a cabo operaciones en tiempo real, el Whirlwind (1951), tenía un diseño que cumplía con los requisitos de un sistema embebido; el primer microprocesador, el Intel 4004, fue creado para un sistema embebido (una calculadora)[2, p.2].

La industria de los sistemas embebidos tendría sus inicios a partir de los años 70 cuando, debido a la reducción de costes en los ordenadores gracias a los microprocesadores, varios diseñadores aprovecharían estos dispositivos para realizar sus productos, ya que hasta el momento los miniordenadores, pese a su gran variedad de aplicaciones, los costes limitaban los ámbitos donde se usaban[3, p.4]. Esta reducción de coste, en parte es debida a los avances que

• E-mail de contacto: 1526394@uab.cat
• Mención realizada: Ingeniería de Computadores
• Trabajo tutorizado por: Sergio Villar Mesurado (CAOS)
• Curso 2023/24

supuso los transistores y los circuitos integrados(tamaño reducido y ahorro de energía) y también que permite el uso de menos componentes (un ejemplo claro es el caso del Intel 4004, que permitió sustituir el sistema de 12 circuitos integrados a solo 4 para llevar a cabo la calculadora[1, p.7]), elementos que también favorecen su producción en masa.

Debido a este modelo de negocio, existe un interés en aprender a desarrollar proyectos y/o prototipos de manera rápida y sencilla para su posterior comercialización. En este trabajo se pretende mostrar un posible método de cómo se debería de trabajar con este tipo de tecnologías, ya que se han dado casos en la industria que se realizan sistemas con prestaciones pobres, como es el caso de la Crisis del Videojuego de 1983[4], siendo una de las principales causas el desarrollo de consolas y videojuegos en masa de mala calidad.

2 Estado del arte

Tammy Noergaard recoge varias definiciones de lo que es un sistema embebido en su libro[5, p.5]: "Un sistema embebido es un sistema informático aplicado, a diferencia de otros sistemas sistemas informáticos como Ordenadores Personales o supercomputadores", "un sistema embebido está diseñado para llevar a cabo una función específica". Se pueden encontrar sistemas embebidos en diferentes ámbitos comerciales y dentro de estos en diferentes sistemas[5, p.6], por ejemplo: en la industria del automóvil se pueden encontrar en los sistemas de freno o en el control del motor; a nivel de usuario como los GPS o juegos; incluso en el ámbito de la medicina como bombas de infusión[5, p.6].

En la Figura 1 se presenta un esquema general de lo que se compone un sistema embebido. Se puede observar que el núcleo del sistema es el microcontrolador, o en otros casos el microprocesador (el primero diferenciándose del segundo al incluir elementos de memoria y entrada/salida en el mismo chip). Estos dispositivos presentan la ventaja respecto otros tipos de arquitecturas de ser muy eficientes(más rápidos) y de facilitar el diseño de familias de productos[2, p.6], además de abaratar costes ya que sustituyen la circuitería específica para tareas de control que incluían antiguos sistemas embebidos[6, p.1].

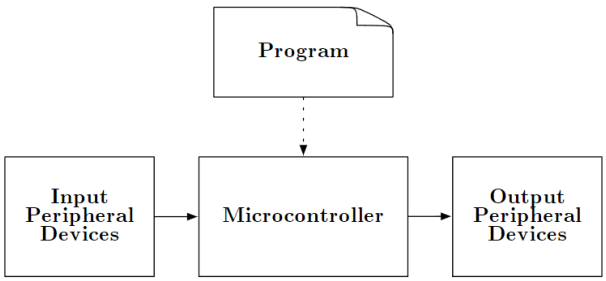

Fig. 1: Esquema de sistema embebido[6, p.4]

Según la arquitectura de la CPU, el microprocesador se puede clasificar como una arquitectura von Neumann (Figura 2) o como una arquitectura Harvard (Figura 3)[2, p.56]. La primera se caracteriza por incluir en la misma memoria tanto el programa como los datos, mientras que la segunda tiene una memoria para datos y otra para el programa, lo que le permite un mayor rendimiento en el procesamiento de señales digitales[2, p.56]. Otro aspecto importante son las instrucciones que puede ejecutar la CPU: las primeras arquitecturas disponían de CISC, pero a medida que se requerían rendimientos mayores se desarrolló la arquitectura RISC. Que el sistema tenga un elevado rendimiento es importante debido a que la mayoría de estos requieren de ejecutarse en tiempo real (de hecho es uno de sus principales atractivos), ya que de no ser el caso puede producir consecuencias negativas, como que el sistema entero falle.

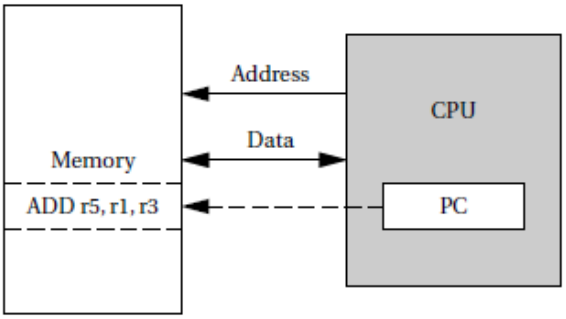

Fig. 2: Arquitectura von Neumann[2, p.56]

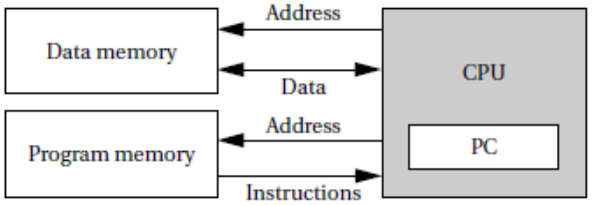

Fig. 3: Arquitectura Harvard[2, p.56]

Otro elemento importante de los sistemas embebidos son los periféricos de entrada y salida, son los encargados de realizar la comunicación con el exterior (operaciones de entrada y salida). Para llevar a cabo la conexión con estos dispositivos, uno de los métodos básicos que dispone la CPU es el Busy-wait I/O[2, p.95], el cual consiste en esperar a que se hagan las operaciones pertinentes. Otro método más eficiente es el uso de interrupciones[2, p.96], en el cual es el propio periférico el que avisa mediante una interrupción de estar listo, dejando libre a la CPU. En la Figura 4 se presenta la estructura de estos dispositivos.

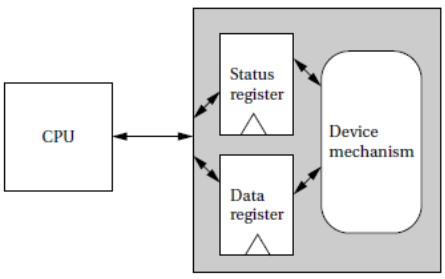

Fig. 4: Estructura de un dispositivo de Entrada/Salida[2, p.92]

El último elemento importante de los sistemas embebidos es el programa. Comúnmente, se siguen tres tipos de estructuras[2, p.210]: una basada en máquinas de estados, otra en flujos de datos y buffers circulares y otra en colas. La primera resulta útil para sistemas reactivos, mientras que las otras dos resultan útiles para el procesamiento de señales digitales[2, p.210]. También, con el fin de facilitar el análisis de código, se realiza un modelo del programa basado en un gráfico de control de flujo[2, p.215], que es un esquema del desarrollo que seguirá el programa. En la Figura 5 se muestra un ejemplo de este esquema, siendo la parte superior el equivalente en C.

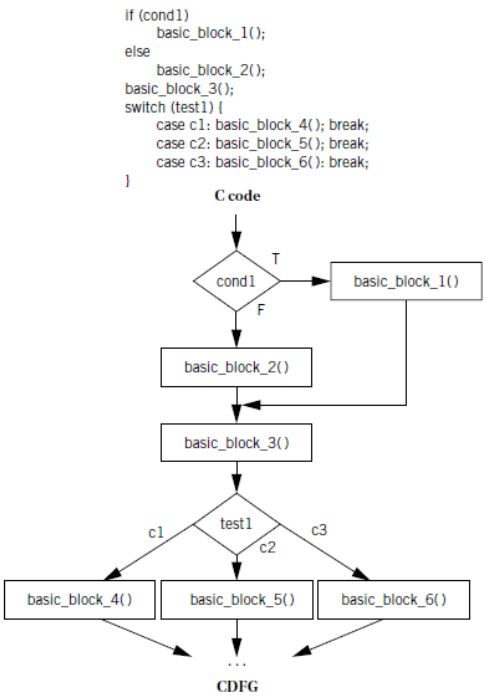

Fig. 5: Ejemplo de gráfico de control de flujo[2, p.219]

A la hora de diseñar un sistema embebido, la metodología a seguir es importante para garantizar que el sistema funcione y que no haya errores. Algunos de los ejemplos de flujos de diseños que se siguen para desarrollar un sistema son[2, p.439]: Modelo Cascada, Modelo espiral, Refinamiento sucesivo y Diseño de Hardware/Software. Todas las metodologías mencionadas, aunque diferentes, incluyen una fase de definición de requisitos y especificaciones, una fase de diseño de la arquitectura (ya sea software, hardware o ambas), una fase de integración y una fase de testeo.

Dada la popularidad de estos sistemas, surgen empresas que comercializan dispositivos que ya incluyen todos los elementos básicos (microcontrolador, memoria, conversores...) dentro de una misma placa para que sean accesibles para todo el mundo y sea sencillo llevar a cabo prototipos de estos sistemas. Massimo Banzi, cofundador de Arduino, y Michael Shiloh mencionan en su libro respecto al desarrollo de prototipos de sistemas electrónicos que ”lo que quieres es ser capaz de confirmar que algo funciona de manera muy rápida de manera que te puedas motivar a ti mismo para ir al siguiente paso”[7, p.4]. A partir de esta idea es de donde salen empresas como la propia Arduino, o Raspberry Pi, siendo la primera la que se utilizará en este proyecto.

Arduino es de hardware y código libre, lo que quiere decir que cualquiera puede usarlo y modificarlos[8]. Gracias a esto, se dispone de una comunidad amplia que realiza proyectos y sube su código a la red, de manera que es fácil encontrar librearías ya implementadas que facilitan la programación de diferentes periféricos. También cuenta con un entorno propio de desarrollo (IDE) que se puede conseguir de manera gratuita en la página oficial[8] y, debido a que al ser de hardware abierto muchas empresas desarrollan su propia placa Arduino, se pueden conseguir a un precio económico.

3 Objetivos

El objetivo del proyecto es mostrar un caso de uso de un análisis que permita confirmar si una determinada aplicación Software podrá ser ejecutada según se espera en un determinado Hardware. En este caso, la aplicación que se probará es un videojuego (se utilizará un código del Pong encontrado por Internet[9]) y la arquitectura es un Arduino Mega 2560 que está conectado con una pantalla TFT ST7735. Para poder confirmar que la aplicación es viable, se realizarán los siguientes pasos:

1. Descripción arquitectura Hardware del sistema
2. Definición requisitos de la aplicación
3. Analizar el Software del sistema
4. Analizar el rendimiento de la aplicación en el sistema

Para ello, se dispone de diferentes herramientas, como el propio IDE de Arduino y profilers que calculan el tiempo de ejecución de una sección de código. La comunicación SPI entre Arduino y pantalla es un punto clave de este sistema, por lo que habrá que fijar la velocidad de transferencia de los datos para que el videojuego funcione de manera correcta y fluida.

4 Metodología

Como se ha mencionado en el estado del arte, el diseño de los sistemas embebidos incluyen las siguientes fases: definición de requisitos y especificaciones, diseño de la arquitectura, integración del sistema y testeo. En el caso de este trabajo, la fase de diseño se sustituye por una fase de descripción del Hardware y análisis del Software, y se alterará el orden de las fases.

Para poder definir los requisitos del sistema, primero es necesario llevar a cabo un estudio de las especificaciones del Hardware. La mejor manera para llevar a cabo esto es ir directamente al Datasheet del fabricante: respecto el controlador de la placa, la información que interesa conocer es la frecuencia de trabajo del controlador, las propiedades de la ALU, la memoria, el repertorio de instrucciones y los ciclos que consumen; respecto el controlador de la pantalla, interesa saber cómo se lleva a cabo el proceso de escritura (protocolo, bits, comandos), la tasa de refresco de la pantalla y la velocidad de transmisión de datos. Una vez se conoce el Hardware, los requisitos se establecen según las especificaciones del Hardware.

Seguidamente, se realizará el análisis Software, que implica analizar el código ensamblador de la aplicación. Para esta fase, se utilizará el propio IDE que proporciona Arduino en su web[8] para programar el Firmware de la placa. Este IDE es open-source, pudiendo encontrar su código fuente en el GitHub de Arduino[10], y se basa en el lenguaje Processing/Wiring[11][12], el cual utiliza una versión simplificada de C/C++. Gracias al ser open source, el IDE dispone de librerías creadas por la comunidad para controlar diferentes periféricos. Para el caso de la pantalla TFT ST7735, existen las librerías Adafruit_ST7735 y Adafruit_GFX con las que se trabajará para llevar a cabo la comunicación de la pantalla, y por lo tanto serán analizadas en esta fase.

Finalmente, se procederá a explicar la integración de los componentes del sistema probando diferentes parámetros de la velocidad de transferencia SPI mediante profilers. Destacar que la fase de testeo no se llevará a cabo debido a que carece de sentido utilizando una aplicación que ya ha sido probada por otra persona.

5 Planificación y desarrollo

En este apartado se presenta la planificación y el desarrollo del trabajo:

5.1. Planificación

En el Anexo A.1 se encuentra la tabla con la planificación inicial del proyecto.

5.2. Desarrollo

En el Anexo A.2 se encuentra la tabla con el desarrollo real del proyecto.

5.3. Evolución

Para la realización de este proyecto, se planteó un seguido de metas a llevar a cabo durante las diferentes semanas disponibles. El objetivo era dedicar unas 15 horas o más repartidas a lo largo de la semana, idealmente 3 horas o más durante 5 días, ofreciendo de esta forma un margen de maniobra ante posibles problemas que impidan llevar a cabo este horario (si un día no se ha podido trabajar en el proyecto se puede recuperar otro día). No obstante, la dedicación de horas ha sido inferior a las planificadas. Esto es debido en parte a que la mayoría de metas se han podido realizar de manera satisfactoria antes de lo previsto, pero también por situaciones ajenas al trabajo de ámbito personal.

El principal cambio que se puede apreciar es la ampliación de la fase de análisis. Esto es debido a que esta fase ha resultado ser más compleja y al ser la parte importante del trabajo, ha llevado más tiempo del esperado. Aprovechando que el desarrollo propio del videojuego era algo secundario, se ha preferido dedicarle la mayor parte del tiempo a entender bien y a llevar a cabo el análisis de las aplicaciones en el Arduino, de manera que si al final había tiempo, se realizaría el videojuego y analizarlo no sería muy complicado. No obstante, al final no ha dado tiempo ha realizar el videojuego propio ya que las últimas tres semanas del trabajo se han dedicado enteramente a reestructurar y terminar el informe final del proyecto.

6 Descripción Hardware

En este apartado se presenta el sistema implementado y los componentes que participan:

6.1. Sistema

El sistema que se implementará para este proyecto consta de los siguientes componentes:

- Arduino Mega 2560 R3 (Marca ELEGOO)
- Pantalla 1.8 Pulgadas SPI TFT ST7735
- Joystick con pulsador
- Botón
- Cables DuPont calibre de 28 AWG (Corriente máxima 1.4A) y de Aluminio revestido de cobre (CCA)

El sistema será montado sobre una placa protoboard y los componentes quedarán expuestos en todo momento para así tener fácil acceso a poder cambiar o añadir conexiones. En la Figura 6 se muestra el conexionado de los componentes y en el anexo A.3 se muestran los pines de la placa Arduino.

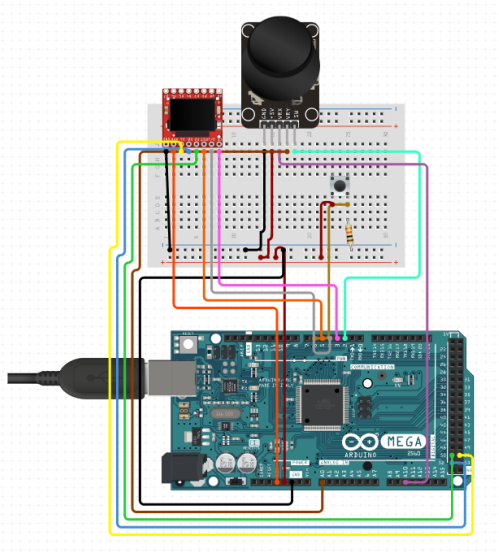

Fig. 6: Esquema de conexiones

En la tabla 6 del anexo A.4 se muestra el Bill Of Materials del sistema que se montará para este trabajo. Se observa que el coste de realizar equipo similar es de 49.96€, pero que para realizar varios sistemas iguales el precio se reduce. Como ejemplo, si se quisieran realizar 100 productos idénticos, el coste sería de 3348.56€, siendo el coste por unidad de 33.48€.

6.2. Componentes

6.2.1. Microcontrolador ATmega-2560

El Arduino Mega 2560 dispone del microcontrolador ATmega-2560. Este microcontrolador de la familia AVR de Atmel tiene una frecuencia de operación de 16 MHz y un rendimiento de hasta 1 MIPS (Millón de Instrucciones Por Segundo) por MHz. Incluye una Memoria Flash de 256 KB para guardar el programa, 8 KB de los cuales están reservados para el Bootloader, y una SRAM de 8 KB para datos. En la Figura 7 se puede observar como se divide la memoria, siendo la primera parte el programa y a partir de la posición 0x1FFFF el Bootloader.

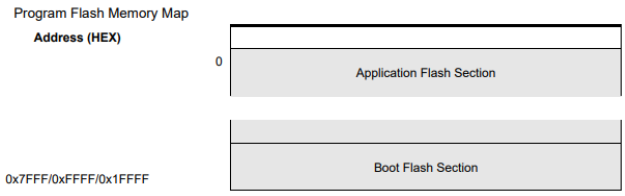

Fig. 7: Esquema de la memoria Flash[13]

El microcontrolador cuenta con una ALU de 8 bits y un repertorio de 135 instrucciones RISC, incluyendo operaciones aritméticas, lógicas, de desplazamiento de bits, comparaciones, entre otras. Éstas pueden ser de 16 o de 32 bits, y la duración en ciclos varia. Por ejemplo, la instrucción CALL es de 32 bits y consume 5 ciclos de reloj , mientras que instrucciones como ADD, SUB y AND son de 16 bits y tardan 1 ciclo. En la Figura 8 se puede observar la arquitectura de la CPU, la cual corresponde a una arquitectura Harvard.

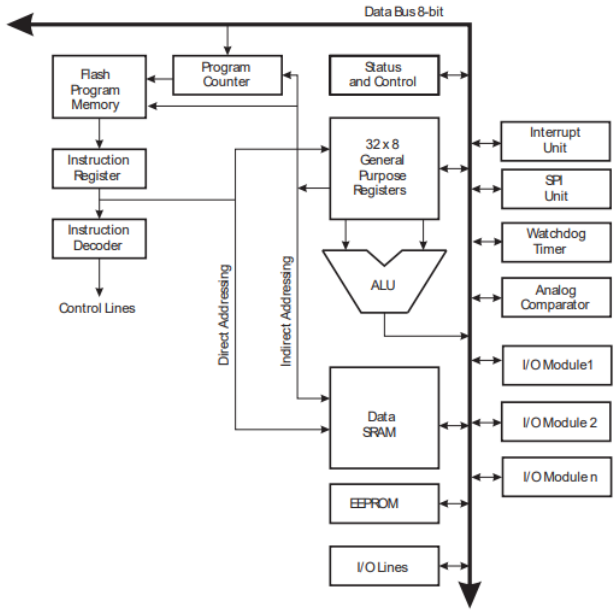

Fig. 8: Arquitectura CPU ATmega-2560[13]

6.2.2. Pantalla y Controlador ST7735

La pantalla TFT ST7735 de AZ-delivery dispone de 8 pines: alimentación, tierra, reset, Chip Select, D/C, SDA, SCK y LED. Los tres primeros se conectan a los pines equivalentes de la placa (5V, GND, Reset), Chip Select y D/C a pines digitales, SDA y SCK se conectan a los pines MOSI y SCK dedicados a SPI de la propia placa, y el LED a la alimentación de 3.3V.

Dispone del controlador ST7735, el cual funciona a través del protocolo SPI. El dispositivo tiene una resolución de 128x160 píxeles y una profundidad de color de 6 bits, es decir, puede representar hasta 262144 colores. Esto determina cómo se debe de llevar a cabo la comunicación con el controlador, ya que cada color (Rojo, Verve y Azul) viene dado por 6 bits de información. En el caso de este proyecto, debido a cómo están implementadas las librerías de Arduino para este controlador, se utilizará una configuración de RGB 5-6-5 bits respectivamente.

Según el datasheet del controlador[14], la escritura en memoria se llevará a cabo de la manera que se representa en la Figura 9: la señal RESX sirve para realizar RESET, la señal IM2 sirve para indicar al microcontrolador que se utilizará SPI (dispone de un método de comunicación paralelo pero esta pantalla no lo admite, por lo que en este caso es como si no existiera), la señal CSX es el Chip Select y se pone a 0 para indicar que va a empezar la comunicación, la señal D/CX indica si lo que se envía por SDA son datos(1) o un comando(0) y SCL es la señal del reloj. El color del píxel viene dado por 16 bits de información y con una tabla de valores se traduce a un código de 18 bits, que se guarda en la RAM.

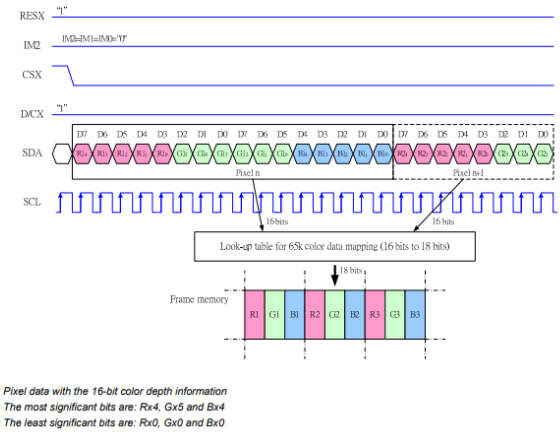

Fig. 9: Proceso de escritura en ST7735 para SPI de 4 lineas y 16 bit/pixel (RGB 5-6-5 bits)[14]

Para hacer funcionar la pantalla, el controlador dispone de varios comandos, pero los que interesan en este caso son CASET, RASET y RAMWR. El primero permite seleccionar las columnas en las cuales se escribirán, el segundo permite seleccionar las filas, y el último es el comando para escribir en la RAM. En la Figura 10 se muestra como funcionan los comandos, transmitiendo de 8 en 8 bits.

La información de la pantalla se guarda en una RAM (Frame Memory) de 132x162x18 bits, la cual tiene una tasa de refresco (Frame Rate) de 60Hz. Para el caso de la pantalla de este proyecto, se utilizará una configuración de 128x160x16 bits.

Instruction	D/C	D7	D6	D5	D4	D3	D2	D1	D0	Function
CASET	0	0	0	1	0	1	0	1	0	Column address set
	1	XS15	XS14	XS13	XS12	XS11	XS10	XS9	XS8	X address start: 0≦XS≦X
	1	XS7	XS6	XS5	XS4	XS3	XS2	XS1	XS0	
	1	XE15	XE14	XE13	XE12	XE11	XE10	XE9	XE8	X address end: S≦XE≦X
	1	XE7	XE6	XE5	XE4	XE3	XE2	XE1	XE0	
RASET	0	0	0	1	0	1	0	1	1	Row address set
	1	YS15	YS14	YS13	YS12	YS11	YS10	YS9	YS8	Y address start: 0≦YS≦Y
	1	YS7	YS6	YS5	YS4	YS3	YS2	YS1	YS0	
	1	YE15	YE14	YE13	YE12	YE11	YE10	YE9	YE8	Y address end:S≦YE≦Y
	1	YE7	YE6	YE5	YE4	YE3	YE2	YE1	YE0	
RAMWR	0	0	0	0	1	0	1	1	0	0	Memory write
	1	D7	D6	D5	D4	D3	D2	D1	D0	Write data

Fig. 10: Comandos ST7735[14]

6.2.3. Joystic y botón

El joystick dispone de 5 pines: alimentación, tierra, eje X, eje Y y Switch. Los dos primeros se conectan a 5V y GND respectivamente, eje X y eje Y a dos pines analógicos y el Switch a un pin digital. El botón extra se conecta por un lado a tierra y por otro lado a un pin digital. En el caso de esta placa, en los botones no hace falta añadir resistencias ya que ya dispone de resistencias pullup internas. Para leer los valores de estos elementos, se llama a la función correspondiente para que se guarde el estado del pin en ese instante, por lo que no hay esperas ni interrupciones.

7 Descripción Requisitos Aplicación

Gracias a las especificaciones del Hardware, se puede establecer como principal requisito que el programa funcione dentro del límite de la tasa de refresco de 60 Hz de la pantalla, es decir, cada iteración del Pong debe suceder en 16 ms. Respecto a las especificaciones, la frecuencia de trabajo será de 16 MHz, el programa debe ocupar menos que los 256 KB de la memoria flash y las variables no deben superar los 8 KB de la SRAM.

Otro requisito importante es la velocidad de transferencia, pero ésta se debe de calcular para garantizar la tasa de refresco. Para ello, se utiliza la siguiente fórmula:

$$Tasa\_refresco = \frac{Bits\_transmitir}{Vel\_SPI} + Tinstr \qquad (1)$$

Se quiere que una iteración del juego suceda en la tasa de refresco de la pantalla, ésta es 60 Hz o 0.01666 ms, por lo que 0.01666=bits/Vel_SPI+Tinstrucciones, de manera que se aisla Vel_SPI para calcularla.

8 Análisis Software

Para poder analizar el código de la aplicación, hay que obtener el código en ensamblador del programa que se carga en la memoria flash del Arduino. Esto se puede conseguir mediante una opción incluida en el propio IDE de Arduino que guarda en ficheros aparte este código[15].

Son 5 los ficheros que se obtienen con la opción de exportar binarios: dos ficheros .HEX, ambos incluyendo el código del programa en hexadecimal, pero uno incluyendo el bootloader y otro no (ver Figura 7), el binario del programa, un fichero .ELF, que proporciona una versión más detallada que los .HEX, y un fichero .EEP.

Para poder interpretar el fichero . HEX y .ELF, se utiliza el comando del terminal avr-objdump, que se incluye al descargar el IDE, para traducirlo a lenguaje ensamblador y así poder entenderlo. El comando incluye una opción para que también se refleje el código original (-S) que facilita la lectura.

```
0000183e <_ZN15Adafruit_SPITFT11sendCommandEhPhh.constprop.8>:
    183e:  0f 93           push r16
    1840:  1f 93           push r17
    1842:  cf 93           push r28
    1844:  df 93           push r29
    1846:  ec 01           movw r28, r24
    1848:  8b 01           movw r16, r22
    184a:  0e 94 9c 07     call 0xf38 ; 0xf38
<_ZN15Adafruit_SPITFT21SPI_BEGIN_TRANSACTIONEv>
    184e:  8f a9           ldd  r24, Y+55  ; 0x37
    1850:  87 fd           sbrc r24, 7
    1852:  06 c0           rjmp .+12       ; 0x1860
<_ZN15Adafruit_SPITFT11sendCommandEhPhh.constprop.8+0x22>
    1854:  eb 8d           ldd  r30, Y+27  ; 0x1b
    1856:  fc 8d           ldd  r31, Y+28  ; 0x1c
    1858:  80 81           ld   r24, Z
    185a:  9a a9           ldd  r25, Y+50  ; 0x32
    185c:  89 23           and  r24, r25
    185e:  80 83           st   Z, r24
    1860:  ed 8d           ldd  r30, Y+29  ; 0x1d
    1862:  fe 8d           ldd  r31, Y+30  ; 0x1e
    1864:  80 81           ld   r24, Z
    1866:  9c a9           ldd  r25, Y+52  ; 0x34
    1868:  89 23           and  r24, r25
    186a:  80 83           st   Z, r24
    186c:  66 e3           ldi  r22, 0x36  ; 54
    186e:  ce 01           movw r24, r28
    1870:  0e 94 fd 06     call 0xdfa ; 0xdfa
<_ZN15Adafruit_SPITFT8spiWriteEh>
    1874:  ed 8d           ldd  r30, Y+29  ; 0x1d
    1876:  fe 8d           ldd  r31, Y+30  ; 0x1e
    1878:  80 81           ld   r24, Z
    187a:  9b a9           ldd  r25, Y+51  ; 0x33
```

Fig. 11: Ejemplo Código ensamblador

En la Figura 11 se muestra un ejemplo del código ensamblador. Se puede ver que este código pertenece a la función sendCommand de la clase Adafruit_SPITFT (clase superior de Adafruit_ST7735) y que ésta llama a las funciones SPI_BEGIN_TRANSACTION y spiWrite de la misma clase, siendo las instrucciones call de 184a y 1870 respectivamente, por lo que también se sabe que esas funciones se encuentran en las direcciones f38 y dfa respectivamente.

Para analizar la aplicación, se obtuvieron los ficheros binarios y se tradujo el .HEX. El problema de este primer intento es que, a parte del código del juego (cuyo lenguaje ensamblador es desconocido), también hay código de otras funciones, por lo que encontrarlo es una tarea que llevaría horas. Por esta razón, se decide realizar un código Saxpy, ya que este se traduce en un seguido de operaciones de multiplicación y suma, por lo que resultaría más fácil encontrarlo y familiarizarse con cómo trabaja Arduino.

En el momento de realizar el código Saxpy, se encontraron varios problemas: al introducirlo con todo el código del menú, los ficheros generados contenían mucho código, por lo que analizarlos resultaba un poco liosos. Otro problema que se encontró fue que a veces el programa del Saxpy directamente no se compilaba: el IDE al compilar el código te muestra la memoria que ocupa el programa y las variables globales, y al analizar estos valores se observó que cuando solo se añadía el código del Saxpy, el espacio de las variables globales si correspondía al esperado (se utilizan arrays de floats, y se observó como al incrementarlos el espacio aumentaba de 4 en 4 bytes), pero al añadir el código del Pong el Saxpy no se compilaba.

La solución al segundo problema fue rápida, porque resultó ser un error en implementar el switch case en el código. Respeto el primer problema, se decidió simplificar el código creando un nuevo programa de Arduino en que solo se llamase a la función del Saxpy. De esta manera, se consiguió comparar fácilmente el código ensamblador de cuando no hay Saxpy del de cuando si que está para ubicarlo. También se descubrió que con el fichero .ELF previamente mencionado, al aplicarle la traducción con el comando avr-objdump, se obtenía un código ensamblador dividido por bloques mucho más fácil de entender.

La traducción del .ELF incluye etiquetas de las funciones que se van llamando, por lo que lo primero que se busca es la función main, que incluye el código de la aplicación. Una vez ubicado, se pasa a analizar las diferentes instrucciones para encontrar las de la aplicación, que estarán ubicadas dentro de un bucle (ya que es como funciona Arduino, se llama todo el rato a la función loop). Con la opción -S, al imprimir el código en C del main (el usuario no tiene acceso a esta función), ayuda a limitar la zona de código a analizar y a entender qué hace el código. En la Figura 12 se muestra cómo se imprime el código C equivalente de la función main y como la función loop es donde se encuentra el código de la aplicación se sabe que las direcciones indicadas en los rjmp son donde empieza la aplicación.

```
    3b8e:  70 e0            ldi   r23, 0x00  ; 0
    3b90:  60 e0            ldi   r22, 0x00  ; 0
    3b92:  82 e5            ldi   r24, 0x52  ; 82
    3b94:  93 e0            ldi   r25, 0x03  ; 3
    3b96:  0e 94 f4 11      call  0x23e8     ; 0x23e8
<_ZN12Adafruit_GFX10fillScreenEj>
    3b9a:  8c e0            ldi   r24, 0x0C  ; 12
    3b9c:  0e 94 a7 04      call  0x94e ; 0x94e <digitalRead>
    3ba0:  01 97            sbiw  r24, 0x01  ; 1
    3ba2:  d9 f3            breq  .-10       ; 0x3b9a <main+
0xb58>

    setup();

    for (;;) {
          loop();
          if (serialEventRun) serialEventRun();
    3ba4:  80 e0            ldi   r24, 0x00  ; 0
    3ba6:  90 e0            ldi   r25, 0x00  ; 0
    3ba8:  89 2b            or    r24, r25
    3baa:  09 f4            brne  .+2        ; 0x3bae <main+
0xb6c>
    3bac:  08 cb            rjmp  .-2544     ; 0x31be <main+
0x17c>
    3bae:  0e 94 00 00      call  0     ; 0x0 <__vectors>
    3bb2:  02 cb            rjmp  .-2556     ; 0x31b8 <main+
0x176>
```

Fig. 12: Ejemplo uso opción -S

Una vez ubicado el código del Saxpy, se procedió a calcular el tiempo que tardaba. Para hacer esto, se extrajo las instrucciones en ensamblador correspondientes en un .txt aparte y se programó un programa en python el cual coge las instrucciones una a una y con un diccionario que contiene la información de los ciclos que consume cada instrucción (información disponible en el Datasheet[13]). Sabiendo que el controlador funciona a una frecuencia de 16 MHZ, simplemente con una división de el número de ciclos totales entre la frecuencia se obtiene el tiempo en que se ejecuta el código. Finalmente, para comprobar el resultado, con la librería Profiler se puede obtener el tiempo que tarda en ejecutarse la parte de código que se desee al ejecutarse en la placa, de manera que se pueden comparar los resultados.

Con el conocimiento adquirido, se vuelve a proceder con el análisis del Pong. En este segundo intento se pudo ubicar fácilmente el código del juego y todas las funciones a las que llama: el fichero .ELF proporciona etiquetas del tipo ⟨Clase_Función⟩, siendo importante saber la estructura de herencias de clases para no equivocarse de clase. En este punto solo se tiene la mitad de la información que interesa para el análisis, la relacionada con el controlador ATmega 2526, por lo que ahora se pasa a analizar las funciones encargadas de la comunicación con SPI. A parte del código individual de cada una, se observó que todas seguían el mismo patrón para enviar datos: primero se inicia la comunicación activando la señal del Chip Select, seguido se llama a la función setAddrWindows que se encarga de seleccionar las columnas y filas de la Frame memory que se escribirán (incluye el control de la señal D/C y ejecución de los comandos CASET, RASET y RAMWR), tercero se procede a escribir el color de cada pixel involucrado y finalmente termina la comunicación desactivando la señal Chip Select. Destacar que se dispone de varias funciones para transmitir datos dependiendo del tamaño de datos, por ejemplo, para CASET el comando se envía por SPI_WRITE(8 bits) y para la posición SPI_WRITE32 (32 bits), para un solo Píxel SPI_WRITE16(bits) y para una región de píxeles se realiza un bucle de X píxeles que llama a la función básica transfer, que envía la información de 8 en 8 bits. Para los casos de más de 8 bits, se llama bits/8 veces a la función transfer. En la Figura 13 se presenta el gráfico de control de flujo de la comunicación, y se puede observar que el protocolo SPI sigue una estructura de flujo de datos y que el procesador se espera a que termine de enviar los datos (Busy-wait I/O).

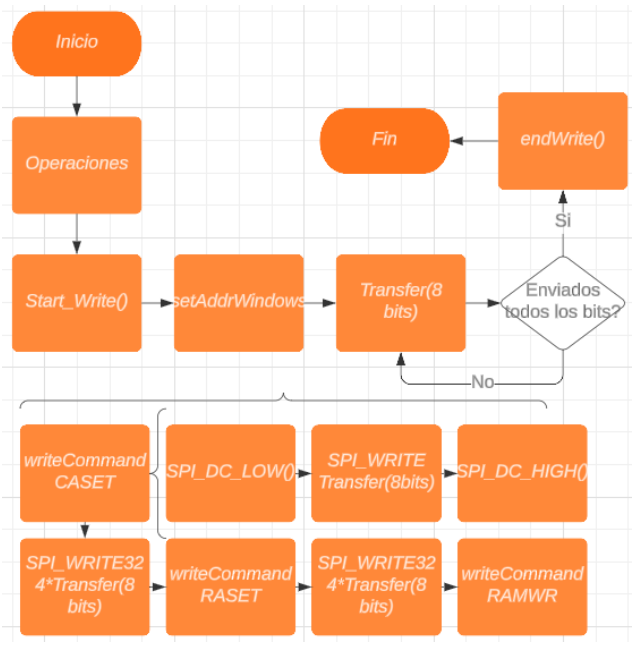

Fig. 13: Gráfico de control de flujo de la comunicación SPI

Finalmente, al tener tanto las instrucciones que se ejecutan y los bits a enviar en total, se puede calcular el tiempo de cómputo y el tiempo de la comunicación SPI, por lo que al utilizar la ecuación 1 se obtiene la velocidad de transferencia.

Debido a que la librería SPI de Arduino funciona de manera que calcula la frecuencia inferior más cercana de dividir la frecuencia máxima (16 MHz) entre potencias de 2, si se obtiene una velocidad de 3 MHz, el Arduino pondría una velocidad de 2 MHz, por lo que hay que configurarlo a 4 MHz, que es la frecuencia superior más cercana.

9 Integración y Resultados

Para llevar a cabo la integración, se ha implementado un menú sencillo que te indica una lista de las aplicaciones que dispone el programa de Arduino y una opción Exit para apagar la pantalla. En la Figura 14 se muestra el menú.

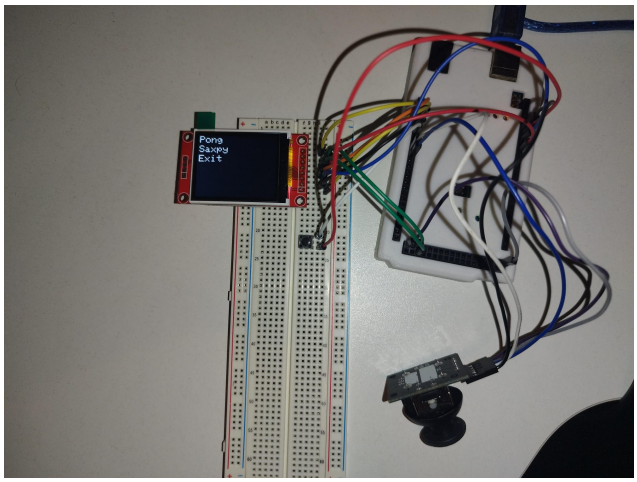

Fig. 14: Menú

Para navegar por este, se utiliza el joystick para marcar la aplicación deseada y pulsando el mismo se pasa a la ejecución de dicha aplicación. En este caso, se han implementado el Saxpy y el Pong. El primero como es un código de prueba simplemente ejecuta varias operaciones de multiplicación y suma a unos vectores previamente inicializados e imprime el primer y último resultado. El segundo se controla con el joystick y cuando acabe una partida puedes empezar otra pulsando el joystick o volver al menú pulsando el botón. En la Figura 15 se muestra el Pong en ejecución.

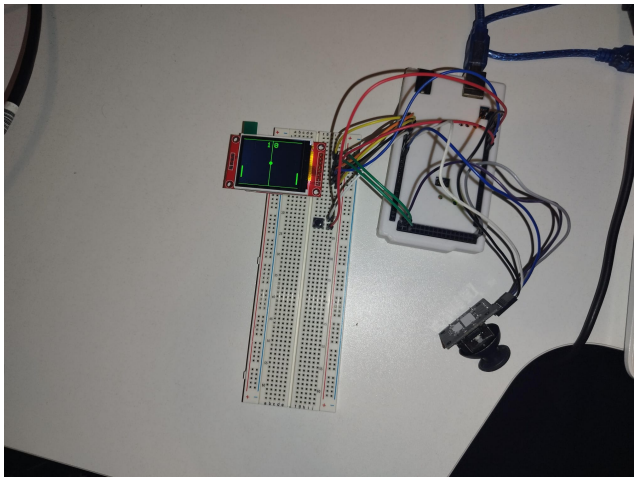

Fig. 15: Pong

En la Figura 16 se muestra el gráfico de control del flujo del programa realizado. Primero se llama a la función setup que inicializa la pantalla, los pines digitales y el menú, seguidamente se entra en la ejecución de la función loop, que es el bucle del programa, que consta de primero mirar si se ha realizado algún desplazamiento vertical en el joystick para desplazarse por el menú y segundo de mirar si el botón del joystick se ha pulsado para acceder a ejecutar la parte del código correspondiente. Mencionar que en cada iteración se llama a la función encargada de imprimir el menú para ir actualizando el estado.

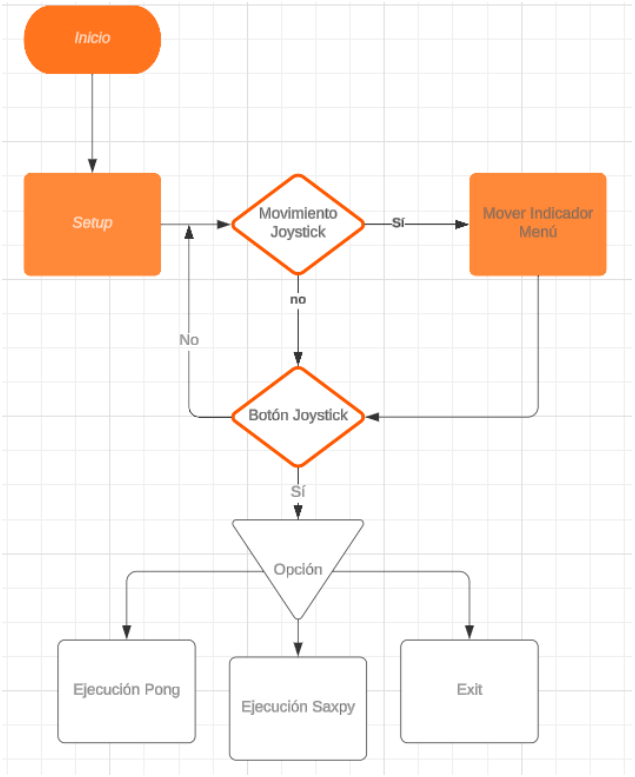

Fig. 16: Diagrama del Programa Principal

Una vez construido el sistema, se procede a realizar un análisis de rendimiento de las aplicaciones. En la Tabla 1 se presentan los resultados obtenidos para el código Saxpy, tanto para vectores de 200 y 400 de tamaño. Respecto los tiempos, se observa que el calculado difiere un poco del real. Esto es debido a que se ha considerado una aproximación secuencial para calcular los ciclos de las instrucciones máquina, cosa que realmente no pasa, ya que hay jumps, condiciones y bucles que alteran el orden de las instrucciones. No obstante, la opción implementada se acerca a la realidad. Mencionar que se cumple que al triplicar el tamaño de los vectores también se triplica el tiempo de ejecución, lo que quiere decir que son directamente proporcionales.

Respecto el espacio utilizado, el programa ocupa en ambos casos unos 5016 bytes en la memoria Flash del Arduino y las variables 2713 bytes y 7513 bytes en la memoria SRAM. Cabe destacar que, sabiendo que los floats ocupan 4 bytes, se cumple que al añadir 600 floats a cada vector(que son 3) respecto la versión de 200 el espacio corresponde al esperado en la versión de 600: 2713+400x4x3=7513.

Saxpy	Tiempo Calculado	Tiempo Real	Espacio Programa	Espacio Variables
200 iteraciones	5 ms	4 ms	5014 bytes	2713 bytes
600 iteraciones	16 ms	12 ms	5016 bytes	7513 bytes

TABLA 1: RESULTADOS SAXPY

En la Tabla 2 se presentan los resultados obtenidos al analizar el Pong. Aplicando la aproximación secuencial, se obtiene que el código tarda 0.01 segundos en ejecutarse, y dado que hay que enviar 22232 bits a través de SPI, se calcula una velocidad de transferencia de 3.8 MHz, por lo que se configura a 4 MHz. Aplicando esa velocidad, al utilizar el Profiler para obtener el tiempo que tarda en ejecutarse una iteración del juego da que tarda 0.017 segundos. Respecto al espacio, el juego ocupa 18448 bytes de la memoria Flash y las variables 586 bytes de la SRAM.

Tiempo Instrucciones	Bits a Transmitir	Velocidad de Transferencia	Tiempo Iteración
0.010078 s	22232 bits	3.8 Hz	17ms

TABLA 2: RESULTADOS PONG

Al ejecutar el juego en el Arduino, aplicando la velocidad de transferencia calculada, dado que el tiempo es similar al objetivo, la velocidad de éste no se ve afectada. No obstante, la pantalla presenta flash de manera intermitente. Se decide probar diferentes velocidades de transferencia:

Velocidad Transferencia	Tiempo Real	Tins+Tspi	Visualización
16 Mhz	0.012 s	0.012+0	No ralentizado No flashea
12 Mhz	0.012 s	0.012+0	No ralentizado No flashea
8 Mhz	0.013 s	0.012+0	No ralentizado No flashea
4 Mhz	0.017 s	0.012+0.005	No ralentizado Flash intermitente
2 Mhz	0.024 s	0.012+0.012	Ralentizado Flashea
1 Mhz	0.04 s	0.012+0.028	Más Ralentizado Flashea

TABLA 3: RESULTADOS PARA DIFERENTES VELOCIDADES

En la Tabla 3 y en la Figura 17 se presentan los resultados obtenidos para diferentes velocidades de transferencia. Se observa que para frecuencias superiores a 8 Mhz el programa ya no presenta ninguna mejora, por lo que se puede concluir que la velocidad del protocolo SPI tiende a 0, por lo que todo el tiempo presente es el tiempo de cómputo de las instrucciones, el cual se aproxima al teórico obtenido previamente de 0.01 segundos. Para el caso de 4 Mhz, el tiempo real se aproxima al deseado de 60 Hz, y efectivamente en este caso el juego no sufre ninguna ralentización, igual se frecuencias superiores, pero si que sufre algún flash de manera intermitente.

A medida que se va reduciendo la frecuencia, el juego cada vez se ve más ralentizado y el flash se manifiesta de manera constante. Destacar que se observa un factor inversamente proporcional en la ralentización, cuando la frecuencia se reduce a la mitad, el tiempo de SPI se incrementa aproximadamente el doble.

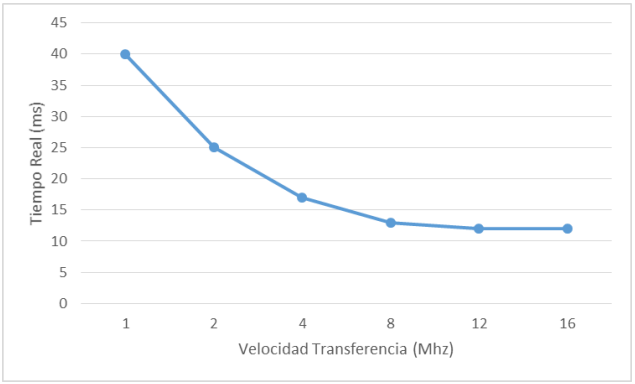

Fig. 17: Gráfico Velocidad-Tiempo

Finalmente, obteniendo un tiempo de instrucciones real de 0.012, a diferencia del de la aproximación secuencial de 0.01 segundos, se vuelve a probar de calcular la velocidad de transmisión, obtetiendo 4.8 Mhz, que entra en el marco óptimo de ejecución.

10 CONCLUSIONES

En conclusión, el videojuego Pong sí que funciona sin errores en el Arduino Mega 2560 y se visualiza correctamente en la pantalla TFT ST7735.

La descripción de la arquitectura Hardware ha servido para determinar a que frecuencia trabaja la placa, 16 MHz, y que la pantalla tiene una tasa de refresco de 60 Hz, que ha permitido fijar el principal requisito de que la aplicación debe ejecutarse a 16 ms por iteración.

El análisis Software ha proporcionado el tiempo de cómputo, 0.01 segundos, y la cantidad de bits a transmitir a la pantalla, 22232 bits, lo que ha permitido obtener que el videojuego debe correr a una velocidad de transferencia de 4 MHz.

En el análisis del rendimiento se ha llevado a cabo la integración del sistema aplicando los valores previamente encontrados, y se ha obtenido que la aplicación funciona de manera correcta, pero presenta aspectos mejorables en la visualización. Se han llevado a cabo pruebas con diferentes velocidades de transferencia, y se ha visto que a partir de la velocidad de 8 MHz el tiempo de SPI llega al mínimo y la aplicación se ejecuta y visualiza de manera correcta.

11 TRABAJO A FUTURO

En este proyecto se ha probado una aplicación ya implementada para Arduino, por lo que se sabe que esta va a funcionar. Como trabajo a futuro, sería interesante implementar una aplicación nueva e implementar un nuevo motor gráfico que no use las librerías ya existentes para así tener un control total de cómo funciona la aplicación y de cómo se realiza la comunicación por pantalla, de manera que se puede realizar un análisis más detallado.

Otro aspecto importante que se podría tratar es la fase de testeo. Se podrían realizar pruebas de tener la aplicación corriendo durante varias horas para ver si el sistema sufre algún fallo, o probar situaciones excepcionales para ver si el sistema reacciona de manera inesperada.

12 AGRADECIMIENTOS

Este trabajo no habría sido posible sin la ayuda de mi tutor Sergio Villar, que ha estado en todo momento guiándome y ayudándome a llevar a cabo este proyecto.

También agradecer a mi familia y compañeros por haberme apoyado y acompañado a lo largo de estos 6 meses de haber estado realizando este trabajo.

REFERENCIAS

[1] Gerard O'Regan, "A Brief History of Computing", 3 edición, 2021

[2] Wayne Wolf, "Computers as Components - Principles of Embedded Computing System Design", 2 edición, Elsevier, 2008

[3] Jack Ganssle, "The Art of Designing Embedded Systems", 2 edición, 2008

[4] Mirko Ernkvist, "Down Many Times, but Still Playing the Game: Creative Destruction and Industry Crashes in the Early Video Game Industry 1971-1986", 2008

[5] Tammy Noergaard, "Embedded Systems Architecture - A Comprehensive Guide for Engineers and Programmers", 2 edición, Elsevier, 2013

[6] David Russell, "Introduction to Embedded Systems : Using ANSI C and the Arduino Development Environment", 2010

[7] Massimo Banzi and Michael Shiloh, "Getting started with Arduino", 4 edición, 2022

[8] Web oficial Arduino, disponible: https://www.arduino.cc

[9] Github del videojuego Pong, David Rutherford, disponible: https://github.com/TheKitty/EsploraApps/blob/master/EsploraPong.ino

[10] GitHub Arduino IDE, disponible: https://github.com/arduino/Arduino

[11] Web oficial Processing, disponible: https://processing.org

[12] Web oficial Wiring, disponible: https://wiring.org.co

[13] Datasheet microcontrolador ATmega-2560, Microchip, disponible: https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega640-1280-1281-2560-2561-Datasheet-DS40002211A.pdf

[14] Stylesheet controlador ST7735, Sitronix, disponible: https://www.displayfuture.com/Display/datasheet/controller/ST7735.pdf

[15] Obtener el código en ensamblador de un sketch de Arduino, Construyendo a Chispas, 21 de enero de 2021, disponible: https://construyendoachispas.blog/2021/01/21/obtener-el-codigo-en-ensamblador-de-un-sketch-de-arduino/

ANEXOS

A.1. Anexo 1

Semana	Tarea	Tiempo	Explicación
1	Reunión con el Tutor	1 h	Reunión para acabar de especificar el tema del TFG
2	Búsqueda Autores relevantes	15 h 3h/5d	Buscar autores relevantes sobre Sistemas Embebidos para estudiar el estado del arte
3	Lectura de obras	15 h 3h/5d	Lectura de las obras importantes de los autores encontrados
4		15 h 3h/5d	
5	Librerías Arduino y menú	15 h 3h/5d	Estudiar las librerías a utilizar de Arduino e implementar un menú de selección
6	Buscar Aplicación	15 h 3h/5d	Buscar un videojuego) ya existente para cargar en el Arduino
7		15 h 3h/5d	
8	Análisis Aplicación	15 h 3h/5d	Llevar a cabo el análisis de la aplicación
9		15 h 3h/5d	
10		15 h 3h/5d	
11		15 h 3h/5d	
12		15 h 3h/5d	
13	Desarrollar videojuego (opcional)	15 h 3h/5d	Como idea extra, realizar un videojuego propio
14		15 h 3h/5d	
15		15 h 3h/5d	
16	Análisis videojuego (opcional)	15 h 3h/5d	Realizar el análisis del videojuego propio
17		15 h 3h/5d	
18		15 h 3h/5d	

TABLA 4: PLANIFICACIÓN

A.2. Anexo 2

Semana	Tarea	Tiempo	Explicación
1	Reunión con el Tutor	1h	Reunión para acabar de especificar el tema del TFG
2	Búsqueda Autores relevantes	9 h 3h/3d	Buscar autores relevantes sobre Sistemas Embebidos para estudiar el estado del arte
3	Lectura de obras	9 h 3h/3d	Lectura de las obras importantes de los autores encontrados
4		9 h 3h/3d	
5	Librerías Arduino y menú	9 h 3h/3d	Estudiar las librerías a utilizar de Arduino e implementar un menú de selección
6	Buscar Aplicación	3 h	Buscar un videojuego ya existente para cargar en el Arduino
7	Análisis Aplicación	20 h 4h/5d	Llevar a cabo la descripción Hardware, definición requisitos, análisis Software e integración
8		12 h 4h/3d	
9		8 h 2h/4d	
10		4 h 2h/2d	
11		4 h 2h/2d	
12		5 h 2.5h/2d	
13		9 h 3h/3d	
14		12 h 4h/3d	
15		12 h 3h/4d	
16	Documentación	3 h 1h/3d	Realización del Informe del trabajo
17		6 h 2h/3d	
18		6 h 2h/3d	

TABLA 5: DESARROLLO

A.3. Anexo 3

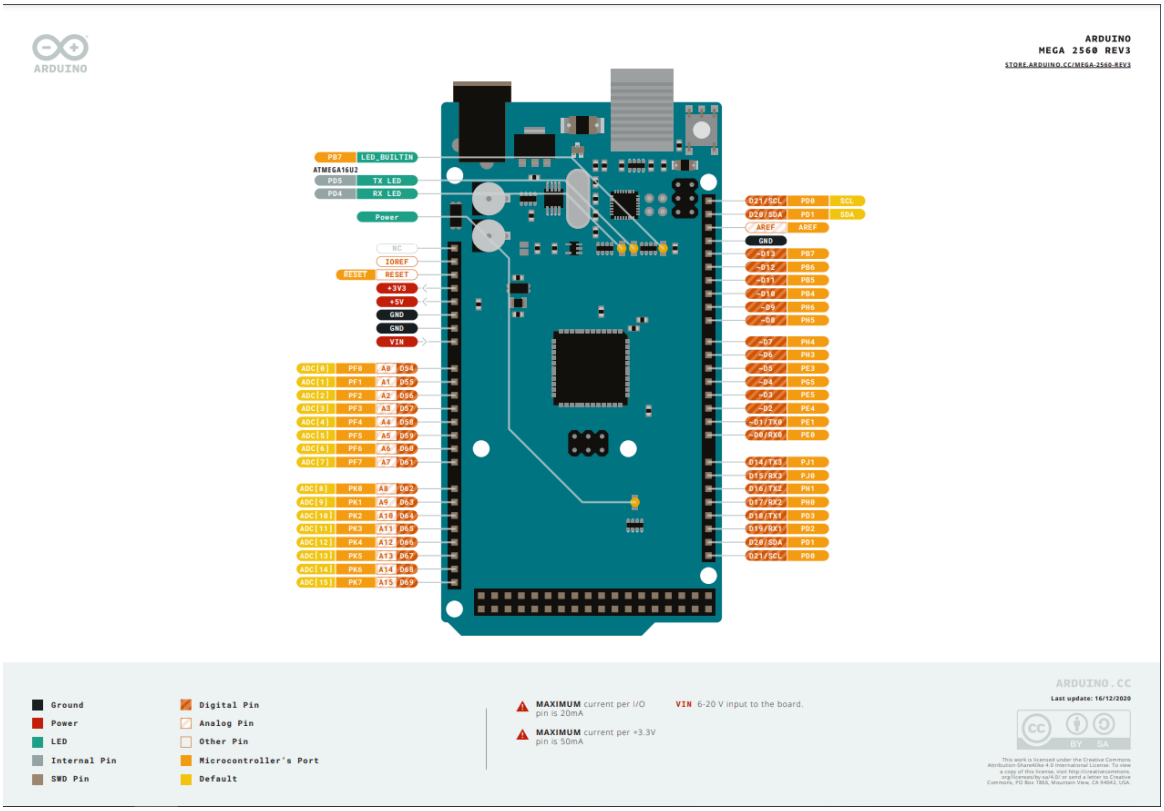

Fig. 18: Pinout Arduino 1

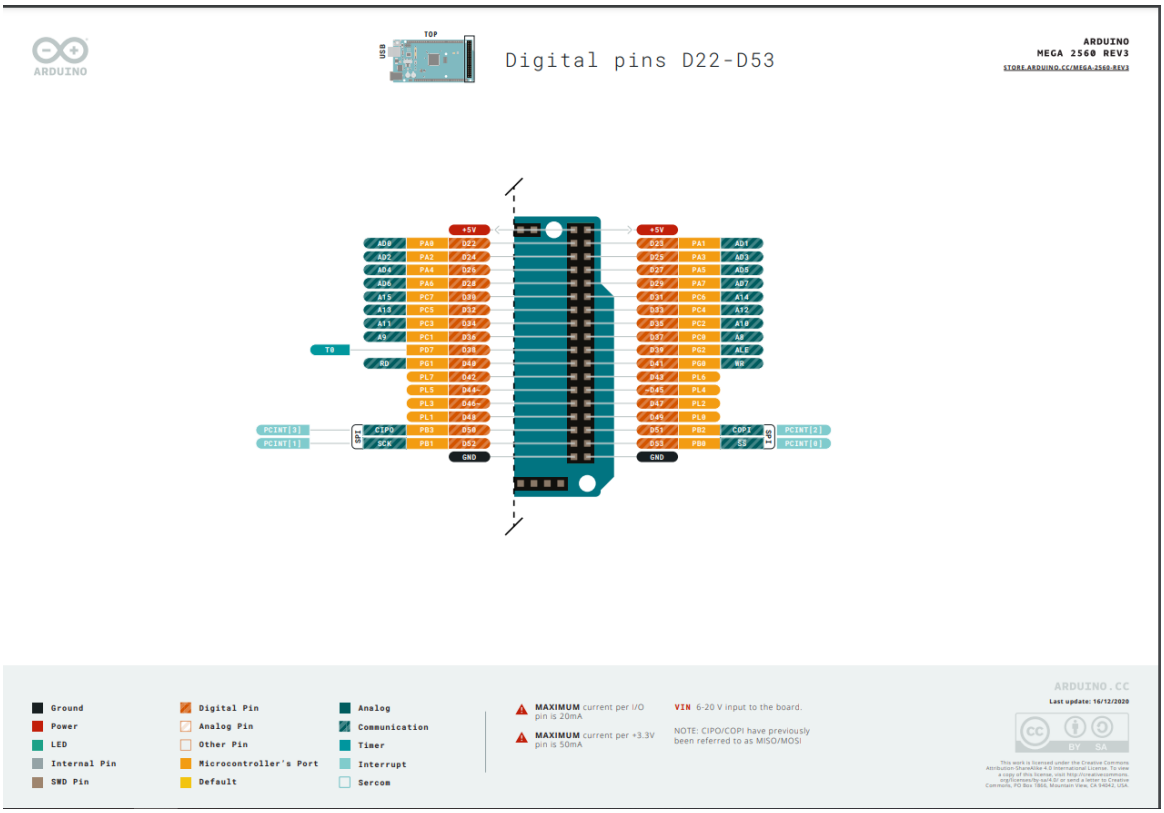

Fig. 19: Pinout Arduino 2

A.4. Anexo 4

TABLA 6: BILL OF MATERIALS

Sistema encastado basado en arduino				
Nombre del Componente	**Vendedor**	**Descripción**	**Cant.**	**Precio**
Elegoo Mega 2560 R3	Elegoo	Placa compatible con arduino IDE Microcontrolador ATmega-2560 256 KB de memoria flash 8 KB de SRAM 4 KB de EEPROM	1	25.99€
Pantalla 1.8 Pulgadas SPI TFT ST7735	AZ-Delivery	Pantalla TFT Resolución 128x160 Interfaz SPI Incluye lector de tarjetas 5/3.3 V	1	8.99€ 17.99€/pack de 3 25.99€/pack de 5
Joystick	AZ-Delivery	Palanca Multidireccional La palanca también sirve de botón	1	4.99€ 7.99€/pack de 3 9.49€/pack de 5
Cables DuPont	Elegoo	Cables DuPont Macho-Macho Calibre 28 AWG	13	9.99€/pack 120 (40 M-M, 40 M-H, 40 H-H)