MEMORIA PRÁCTICAS 2 Y 3

PROYECTO HARDWARE

Daniel Rueda Macías y Adrián Susinos Moreno
559207 y 650220

15-1-2016

RESUMEN

En estas dos partes hemos dotado de numerosas funcionalidades a nuestro código acompañándonos por la placa S3CEV40. Lo primero que hicimos fue familiarizarnos con la placa y configurar el timer para que funcionara correctamente a la máxima precisión. Este timer sería usado posteriormente para numerosos casos del proyecto.

Tras comprobar el correcto funcionamiento del timer, continuamos con el siguiente paso el cual trataba de establecer un comportamiento para diferentes excepciones (PABORT, DABORT y UNDEF). Una vez establecida la rutina, generamos una excepción de estos tipos para ver que realizaba el comportamiento deseado.

Además, hemos desarrollado una pila de depuración, la cual nos permite apilar eventos, con tres parámetros: un id de evento, datos aclaratorios y el tiempo en el que se ha generado el evento (tiempo leído del timer). Cabe destacar que esta pila es una pila de tipo circular la cual solo permite apilar n eventos.

Una vez realizadas estas tareas “complementarias”, pasamos a desarrollar una máquina de estados que nos ayudará a implementar el código para gestionar las pulsaciones de los botones de la placa y eliminar todos los rebotes que se producían en ella.

Cuando los botones funcionaban correctamente y sin rebotes, iniciamos la medición de los tiempos de ejecución de las distintas versiones del código del sudoku realizado en la práctica uno, pero esta vez con el timer de la placa configurado en esta práctica para así poder conseguir una mayor precisión y ver que se se asemejaba a los tiempos calculados en la primera práctica.

Finalizado lo anterior, añadimos al código la implementación de la máquina de estados que representa la interacción del usuario con la placa. Esto añadió distintas funcionalidades al código que nos permitían jugar con él: añadir y eliminar valores, incrementar el led al mantener pulsado el botón…

Con este último paso, teníamos nuestro sistema en funcionamiento el cual nos permitía hacer un sudoku ayudándonos de los pulsadores y el 8led de la placa, pero el juego carecía de una interfaz gráfica por lo que jugar se hacía un poco tedioso. Implementar dicha interfaz es la tarea que realizaríamos a continuación.

La interfaz gráfica trata de ver en todo momento el tablero del sudoku en la pantalla e ir actualizándolo conforme el usuario interactúa con la placa. Teniendo como punto de partida una pantalla inicial que explica las instrucciones del juego, además de una pantalla final que indica el resultado del juego y el final de este.

En el tablero del sudoku se distinguen qué números ha introducido el usuario, cuales son pista, que números introducidos son erróneos y los posibles candidatos de las celdas vacías. Además en la pantalla se puede observar el tiempo total de partida y el tiempo de cálculo acumulado.

Con el sudoku completamente funcionable y su correspondiente representación gráfica, la última tarea fue introducirlo en la memoria de la placa para poder ejecutarlo directamente desde allí y así tener un sistema empotrado autónomo.

INTRODUCCIÓN

En esta práctica partimos de la parte fundamental del sistema ya realizado, es decir, todo el código desarrollado en la práctica uno. Ahora, tenemos la placa S3CEV40 la cual no hemos utilizado anteriormente, con su respectiva documentación la cual nos hará falta comprender para su correcto funcionamiento.

Debemos integrar nuestro código con la placa, para tener un sistema autónomo que nos permita jugar a un sudoku determinado apoyándonos en los componentes de dicho sistema y utilizándolos para interactuar con él.

OBJETIVOS

El objetivo de la práctica es poseer un sudoku jugable en un sistema autónomo. Para completar este sudoku, poseemos distintas funcionalidades además de información que nos va a ir mostrando el sistema. Entre esta información vamos a poder ver distintos tiempos (tiempo de cálculo y tiempo de juego), también vamos a poder comprobar en tiempo real los candidatos de una celda vacía, así como los errores que hayamos podido cometer al introducir un número. Todas estas funcionalidades tienen el objetivo de ayudarnos a la hora de completar el tablero. Una vez finalizado el sudoku, ya sea porque hemos acabado el tablero o porque hemos decidido salir del juego, el sistema además nos va a dar información de la partida, así como los errores que hemos cometido o si el tablero esta completo o incompleto.

METODOLOGÍA

Dado que esta memoria está compuesta por dos prácticas, vamos a separar la metodología en dos partes correspondientes a cada práctica para que todo quede mejor estructurado.

Jerarquía de ficheros

En nuestro proyecto se pueden encontrar los siguientes ficheros y carpetas:

- Carpeta common: En esta carpeta se encuentran ficheros de inicialización del proyecto.
- Carpeta sudoku: En esta carpeta se encuentran los ficheros correspondientes al código de la práctica 1.
- El resto de ficheros son los correspondientes a todo lo desarrollado en las práctica 2 y 3, entre estos ficheros se encuentran el fichero elementosComunes.h en el que se define una enumeración para que así esta se pueda utilizar en todos los ficheros del proyecto y que exista una abstracción a la hora de leer el código por una persona que no haya sido participe en la elaboración de este.

Práctica 2

Configuración del timer

Esta fue la primera parte de la práctica. Lo primero que hicimos fue mirar el manual para ver qué registros estaban involucrados en el timer (timer 2). Las primeras modificaciones las hicimos en la función timer2_inicializar() la cual es la que se encarga de poner en marcha el timer. Cada vez que modificábamos un registro, sólo modificábamos los bits que nos interesaban, dejando los demás como estaban. Esto lo conseguíamos realizando máscaras con operaciones lógicas (AND y OR) bit a bit según la necesidad. También establecimos una rutina de servicio para este timer que se llamó rutina_Timer2().
Los registros que tuvimos en cuenta para la configuración fueron los siguientes:

- rINTMSK: Registro que enmascara las líneas IRQ, desenmascarando la del timer 2.
- rTCFG0: Registro que se encarga de ajustar el pre-escalado de los timer de la placa, en nuestro caso, establecimos los bits correspondientes al pre-escalado del timer 2 a 0. Esto junto a la frecuencia sirve para establecer la precisión del timer.
- rTCGF1: Registro que corresponde al valor del divisor del reloj del timer 2, en nuestro caso pusimos el valor 0 que corresponde al valor ½. Junto con este registro y el anterior, aplicando esta fórmula: *F = señal_interna_reloj/ ((valor de pre-escalado + 1)(valor del divisor))*. Conseguimos la precisión máxima para el timer.
- rTCNTB2: Registro que indica el valor inicial de la cuenta descendente del timer. En nuestro caso lo definimos a 65535.
- rTCMPB2: Registro que indica el valor al que tiene que llegar rTCNTB2 para completar un ciclo.

- rTCON: Registro de control para los timer. En nuestro caso antes de inicializar un timer, establecemos la actualización de los valores de forma manual (bit 14 en timer2) e inmediatamente establecemos la actualización a modo automático (bit 16 en timer2) e iniciamos el timer al mismo tiempo (bit 13 en timer2).

La rutina de interrupción que programamos para este timer se encarga de incrementar la variable que cuenta el número de interrupciones (*timer2_num_int++*) y bajar el bit del timer2 en el registro rI_ISPC que se encarga de almacenar las interrupciones pendientes.

Otra función que definimos en el timer2 es timer2_leer(), ésta se encarga de devolver un entero que corresponde al tiempo en μs desde que se ha inicializado el timer. Este tiempo lo obtenemos aplicando la siguiente fórmula:
*((rTCNTB2-rTCMPB2)*timer2_num_int+(rTCNTB2-rTCNTO2))/32*. Básicamente esta fórmula multiplica el tiempo de ciclo por los ciclos que se han producido y le suma el tiempo del ciclo actual. Todo esto lo dividimos por la frecuencia del reloj.

Por último, tenemos una función timer2_empezar() que se encarga de reiniciar el timer y pone la variable para contar el número de interrupciones a 0.

Una vez configurado, comprobamos su correcto funcionamiento utilizando la función delay().

Gestión de excepciones

Para gestionar el tratamiento de excepciones que se nos pedía en el guión (Undef, Pabort, Dabort), creamos una función que establecía una rutina de interrupción para estos tres tipos de excepción. Esta rutina (ISR_queHacer()) copiamos el registro de estado del procesador (CPSR) a una variable y realizamos una máscara para quedarnos con los últimos 5 bits y así obtener el tipo de excepción. Además mostramos una E en el 8led.

Una vez configurado este tratamiento de excepción, forzamos que salten estas mediante un código que obtuvimos del material de moodle. Con esto pudimos comprobar el correcto funcionamiento de la función.

Pila de depuración

Para conseguir crear una pila en la cual podamos gestionar eventos, lo primero que hicimos fue definirla en una zona de memoria ubicada justo después de las demás pilas de modos del procesador, la definición de la pila se hizo de esta manera:
*.equ DebugStack, _ISR_STARTADDRESS-0xf00+256*6 /* c7ff600*/.*

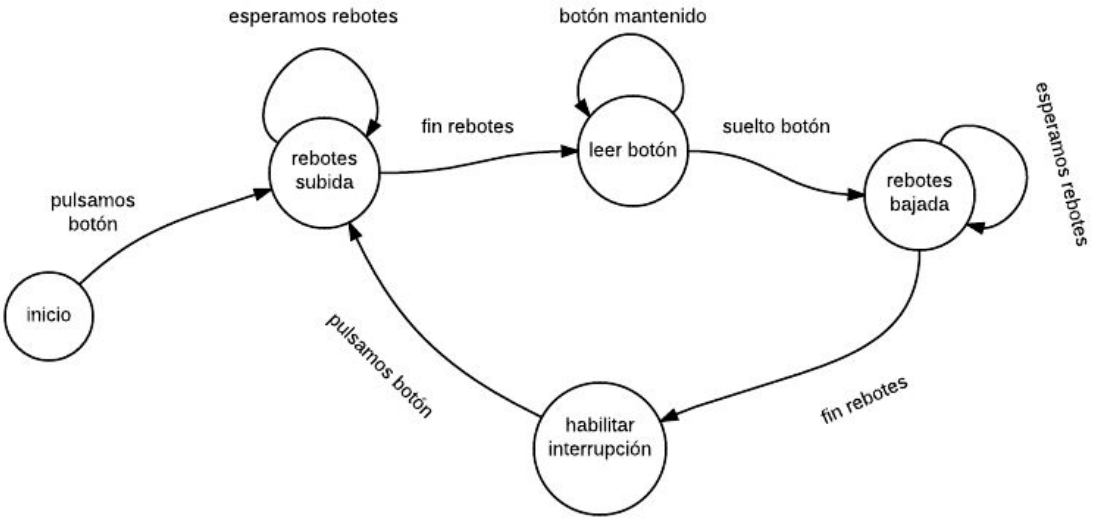
Una vez que tenemos la zona de memoria reservada, implementamos una función llamada push_debug() la cual le pasamos por parámetro un identificador del evento y un dato auxiliar. Esta función se encarga de almacenar en la pila estos dos datos, además del tiempo en el que se ha llamado a esta función, para esto utilizamos un timer de la placa.

Esta pila es una pila de tipo circular en la que solo se pueden apilar n elementos y tras esto, se vuelve a apilar desde el principio. Para conseguir esto, en la función comprobamos si se ha alcanzado el final de la pila para saber donde hay que apilar.

Eliminación de los rebotes en los pulsadores

Para eliminar los rebotes en los pulsadores lo primero que tuvimos que hacer es medir el tiempo que tarda en estabilizarse la señal de pulsación tanto al pulsar como al soltar un botón. Para comprobar esto, utilizamos la pila de depuración creada anteriormente y obtuvimos un valor de unos 20 ms tanto para los rebotes de subida como los de bajada.

Una vez con esto, diseñamos nuestra máquina de estados:



Para implementar la máquina de estados hemos declarado una enumeración para respetar la modularidad del código con los siguientes estados (menos el inicial): rebotes_subida, leer_boton, rebotes_bajada, habilitar_int.

Nuestra máquina de estados parte de un estado inicial en el que no hay ningún botón pulsado. Al pulsar un botón se salta a la función de interrupción Eint4567_ISR(). Esta función, deshabilita las interrupciones de botón e iniciar el timer 0 para gestionar los rebotes y los botones.

Al iniciar el timer0 los configuramos de forma que produzca una interrupción cada 3,5 μs es decir, que para los 20 ms que necesitamos es necesario que se hayan producido 5714 interrupciones. Además, pondremos como siguiente estado rebotes subida.

En **rebotes_subida**, para leer el botón pulsado, vamos a esperar los 20 ms que tardan en desaparecer los rebotes, esto lo conseguiremos con una variable (rebotes) que tiene el valor 5714, por lo tanto, cuando el número de interrupciones es igual a 5714, quiere decir

que hemos llegado a los 20 ms y los rebotes de subida han acabado. Ahora por lo tanto, establecemos como siguiente estado leer_boton y ponemos el número de interrupciones a 0 para contabilizar en el siguiente estado el tiempo.

En **leer_boton**, vamos a comprobar que botón hemos pulsado, para esto, tal como dice el enunciado, tenemos que monitorizar cada 10 ms si el botón sigue pulsado, para ello, tenemos otra variable análoga (monitorizarPulsado) a la de rebotes pero con un valor de 2257. Cada vez que han pasado esos 10 ms registramos si hay algún botón pulsado (valor_actual), como la primera vez que monitorizamos seguro que hay un botón pulsado vamos a guardarnos de que botón se trata en otra varible (valor_a_pintar) que trataremos en otro estado. Así, lo que nos hará saber si tenemos que cambiar de estado (porque el botón puede estar mantenido o no) es la variable valor_actual. Si al monitorizar no hay ningún botón pulsado, pasaremos al estado rebotes_bajada.

En caso contrario, vamos a mirar que el botón que estamos pulsando sea el izquierdo. Si es así, necesitamos llevar otra cuenta (cuenta50) que nos monitorizará cuando han pasado 50 ms con el botón pulsado. Si han pasado esos 50 ms, iniciaremos otra cuenta que monitorizará cada 30 ms, por lo tanto cada dicho tiempo deberemos incrementar el valor del led pero teniendo en cuenta las siguientes restricciones:

- Si aún no hemos seleccionado la fila (fila=0), los valores que se deberán mostrar en el 8led serán de 1 a A. Para ello realizamos un módulo 11 guardando el resultado en una variable llamada valorLed que utilizaremos para saber que valor pintar. Cómo el dominio de este módulo es [0,10], necesitamos salvar el caso de que el módulo sea 0, solucionandolo poniéndolo a 1.
- Si hemos elegido la fila nos encontramos en el caso de que tenemos que elegir la columna, por lo tanto los valores comprendidos van a ser de 1 a 9, módulo 10 salvando el caso de que dicho módulo sea 0.
- Si hemos elegido fila y columna, los valores comprendidos van a ser de 0 a 9.

Por último, se muestra el valor correspondiente en el 8led, es decir, el valor que tiene la variable valorLed y ponemos el número de interrupciones a cero, para volver a monitorizar a los 10 ms.

En el estado **rebotes_bajada,** es el estado que representa que hemos soltado el botón. Al igual que en rebotes subida, debemos esperar 20 ms, a que desaparezcan los rebotes, para ello usaremos la misma variable (rebotes).

Debemos comprobar la variable mantenido, que nos indicará si habíamos mantenido pulsado el botón, para en este caso, no aumentar el valor del 8led (en el caso de que sea el botón izquierdo). Si el que hemos pulsado es el botón derecho, deberemos confirmar el número seleccionado además de iniciar el programa en el caso de que no se haya iniciado. Si el programa ya ha sido iniciado, la primera vez que pulsemos el botón derecho, seleccionaremos la fila, seguido de la columna y el valor. Al finalizar el estado, pasaremos al siguiente: habilitar_int.

Por último en el estado **habilitar_int**, nos encargaremos de volver a habilitar las interrupciones, de forma que enmascaramos la interrupción del timer 0, lo paramos

poniendo un 0 en el bit 0 del registro rTCON, bajamos el flag de la interrupción de botón y del timer 0 y volvemos a activar las interrupciones de botón.

Práctica 3

En esta práctica tuvimos que trabajar con la pantalla lcd para mostrar el sudoku y el comportamiento del juego que habíamos desarrollado en la práctica 2, además de añadir alguna nueva funcionalidad que detallaremos más adelante. Tambíen. una vez creada la interfaz gráfica, introdujimos el programa a la placa para hacer de ella un sistema autónomo.

Para el uso de la pantalla utilizamos el fichero *lcd* y *bmp* los cuales contenían funciones que se encargaban de dibujar en la pantalla. Tras familiarizarnos con estas funciones y ver cómo funcionaban, empezamos a dibujar nuestro tablero en la pantalla.

Dentro del fichero lcd, tenemos funciones que se encargan de dibujar diferentes datos por pantalla, las cuales describiremos a continuación:

- Lcd_inicio(): Función que se encarga de dibujar la pantalla de inicio del sudoku, en ella aparecen instrucciones del juego. Para pintar utilizamos la función Lcd_DspAscII8x16(), y para refrescar la pantalla Lcd_Dma_Trans().

- pintar_Sudoku(): Función que pinta el tablero del sudoku, así como cada uno de los índices. Para dibujar el tablero utilizamos la función Lcd_Draw_Box() que pinta un cuadrilátero de las dimensiones indicadas a la función. Para pintar las líneas verticales utilizamos la función Lcd_Draw_VLine() y para las horizontales Lcd_Draw_HLine(). Además distinguimos las líneas de cuadrícula que serán más gruesas que las líneas finas.

- pintarNumeros(): Esta función se encarga de pintar los diferentes números del tablero, además de llamar a la función pintarCandidatos() que se encargará de dibujar los diferentes puntos que indican los posibles candidatos. La función pintar números trabaja de la siguiente forma:
  Vamos pasando por cada celda del sudoku y comprobamos si:

  - ❖ Celda vacía: Obtenemos bits de posibles candidatos, de cada candidato obtenemos la coordenada del punto a pintar y llamamos a la función pintarCandidatos() para cada uno.

  - ❖ Celda llena:
    - ➢ Pista: Obtenemos el valor y lo pintamos en el lcd.

    - ➢ Número introducido por el usuario: Obtenemos el valor y lo pintamos en el lcd con mayor grosor que la pista. Además comprobamos el bit

de error, para en caso de que sea erróneo, añadir más grosor en los bordes de la celda.

- pintarCandidatos(): Esta función se encarga de pintar el punto del candidato correspondiente en la cuadrícula.
- finPartida(): Esta función se encarga de llamar a la función recalcular para comprobar si hay errores en el tablero, el tablero está completo o incompleto e informar de cada caso por pantalla.

Por último vamos a explicar el funcionamiento global del programa, el cual se efectúa en la clase Main() y se comporta de la siguiente manera:
Al iniciar el Main, lo primero que hacemos es inicializar el estado de la máquina de estados al estado de inicio. También inicializamos el sistema (sys_init()), el lcd dibujando la pantalla de inicio (Lcd_inicio()), habilitamos los pulsadores (Eint4567_init()), iniciamos el 8led dibujando una F (D8Led_init()) e inicializamos el timer2 que lo utilizaremos para medir los tiempos de cálculo y de partida.

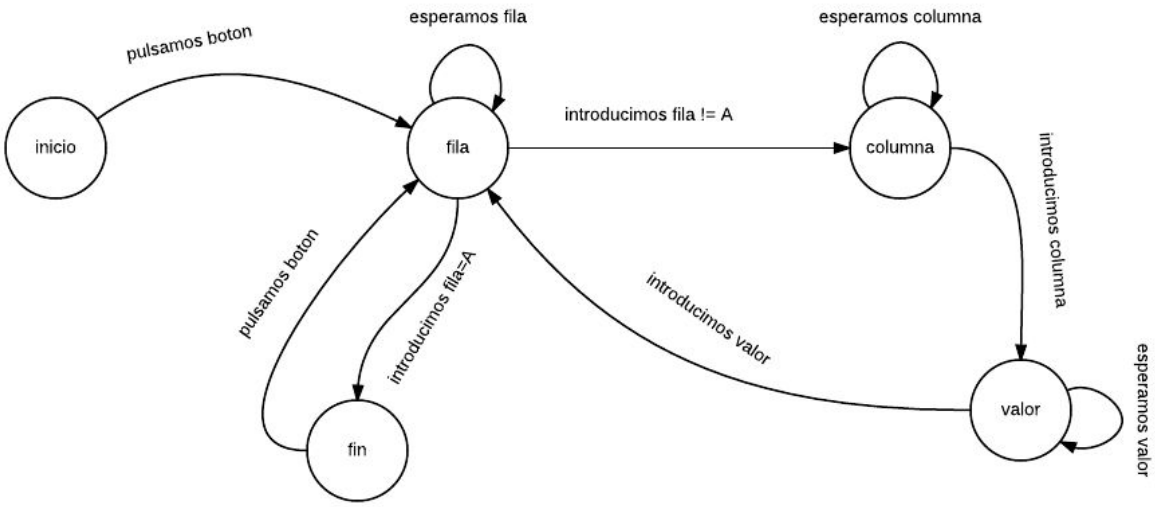
Tras esto el programa queda a la espera de que el usuario pulse un botón, esto lo realizamos con una variable (empiezo), la cual no cambiará hasta la interacción del usuario. Una vez que el botón haya sido pulsado, se realizará una recalculación del tablero (sudoku9x9()) y pintaremos el tablero (pintarSudoku()) y los números (pintarNumeros()).

La primera vez que iniciamos el juego, no se ha realizado ningún cálculo y por lo tanto no tenemos un tiempo de cálculo por lo que nos quedaremos a la espera de que se seleccione una fila. Mientras esperamos a que se elija la fila, vamos contando el tiempo de partida y actualizándolo por pantalla. Para esto utilizamos el timer 2 que está generando interrupciones cada 2 ms, así que cada 489 interrupciones, contaremos un segundo.
Una vez que se ha seleccionado la fila, debemos comprobar si el valor elegido es una A o cualquier valor distinto de este. Si es una A, entonces la partida finaliza mostrando una F en el 8led y llamando a la función finPartida() y restaurando la cuadrícula de inicio.

Si hemos seleccionado un valor distinto de A, mostraremos una C en el 8led y esperaremos a que el usuario introduzca el valor de la columna contabilizando el tiempo de partida al igual que al esperar la introducción de la fila.

Una vez introducida la columna, esperamos a que el usuario introduzca el valor para poner en la celda. Cuando introduzca el valor, por un lado empezaremos a leer el tiempo a través del timer 2 lo cual utilizaremos para mostrar el tiempo de cálculo y por otro lado llamaremos a la función poner_valor() que se encarga de añadir un número a la cuadrícula o de eliminarlo en caso de que el valor sea 0. Acto seguido restablecemos los valores de fila, columna y valor para que vuelva a iniciar el proceso de selección de fila.

A continuación adjuntamos la máquina de estados de la interacción con el usuario.



Introducción del programa a la placa

Para que el juego se ejecute correctamente de forma autónoma en la placa, modificamos el fichero 44binit.asm. Se efectuaron los siguientes cambios:

1. En la línea 257 del fichero se ha cambiado la línea por la siguiente instrucción: *ldr r0,=(SMRDATA-0xc000000)*. Esto se encarga de establecer la dirección de inicio del programa en la memoria de la placa.
2. A partir de la línea 279 se ha escrito el siguiente código:

```
LDR r0,=0x0
LDR r1,=Image_RO_Base
LDR r3,=Image_ZI_Limit

LoopRw:

cmp     r1, r3
ldrcc   r2, [r0], #4
strcc   r2, [r1], #4
bcc     LoopRw



mov r3, #0
LDR r0, =Image_ZI_Base
LDR r1, =Image_ZI_Limit
```

```
LoopZI:
cmp r0, r1
strcc r3, [r0], #4
bcc LoopZI

MRS    r0, CPSR
BIC    r0, r0, #NOINT /* enable interrupt */
MSR    CPSR_cxsf, r0
```

Todo este código se encarga de cargar el programa en la placa, y resetear la memoria de datos para que no haya problemas.

Otros aspectos a destacar

Tiempo dedicado

Hemos llevado un control de las horas de trabajo realizadas tanto en el laboratorio como fuera de él, teniendo como resultado un total de:

- 68 horas aprox. para la realización de la práctica 2.
- 31 horas aprox. para la realización de la práctica 3.
- 9 horas para la realización de la memoria.

Número de líneas de código y tamaño de los ficheros

- 8led.c: 56 líneas y 2KB.
- Bmp.c: 195 líneas y 9KB.
- Button.c: 52 líneas y 2KB.
- debug.c: 36 líneas y 2KB.
- lcd.c: 650 líneas y 21KB.
- led.c: 80 líneas y 2KB.
- main.c: 132 líneas y 6KB.
- timer.c: 175 líneas y 7KB.
- timer2.c: 88 líneas y 3KB.
- tratamientoExcepcion.c: 22 líneas y 1KB.

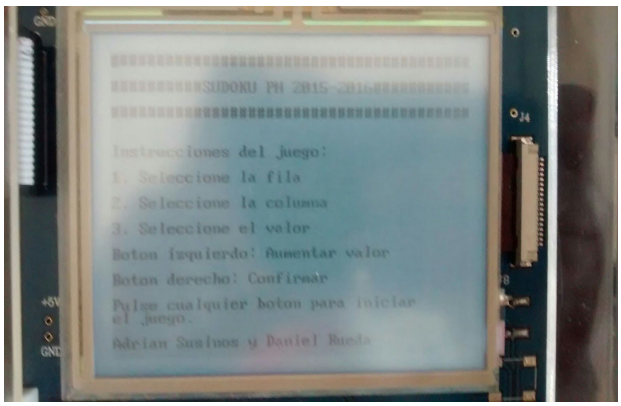
RESULTADOS

En esta práctica hemos vuelto a calcular el tiempo de ejecución de la función recalcular con todas las combinaciones de las distintas implementaciones pero esta vez midiendolos con el timer de la placa, lo que nos aporta una mayor precisión de cálculo. Los tiempos calculados son los siguientes:

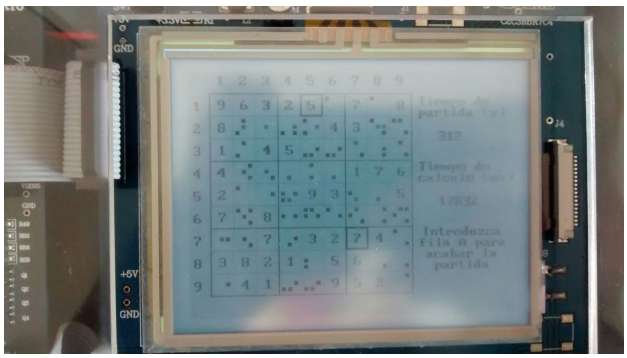
- C-C: 29.381 ms.
- ARM-ARM: 7.485 ms.
- C-ARM: 7.707 ms.
- ARM-C: 29.183 ms.
- C-THUMB: 9.198 ms.
- ARM-THUMB: 8.987 ms.

Podemos comprobar que los tiempos se asemejan a los calculados en la práctica 1 siendo la combinación más eficiente en cuanto a tiempo de ejecución la de ARM-ARM.

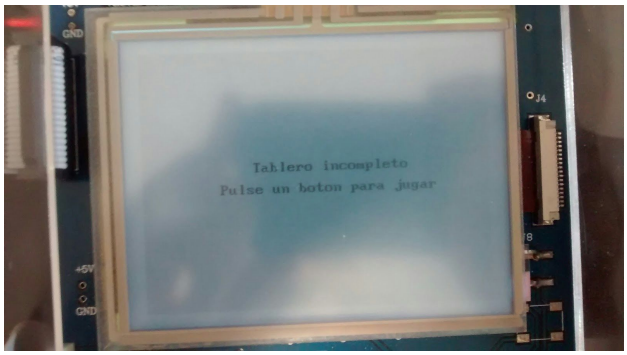
A continuación vamos a adjuntar distintas imágenes que muestran nuestro sistema en funcionamiento:



Pantalla inicial del juego



Pantalla de juego



Pantalla final juego

Por último se adjunta un enlace de YouTube, donde se puede ver el juego en funcionamiento: https://www.youtube.com/watch?v=hy_BBJ5fqBg

CONCLUSIONES

Una vez que hemos realizado todo el proyecto, podemos destacar que ha sido un proyecto en el que se ha requerido mucho esfuerzo y tiempo. Dado que nunca habíamos trabajado con hardware sin tener un intermediario como un sistema operativo, además de ser nueva la placa para nosotros. Hemos tenido que leer mucha documentación y realizar muchas pruebas para comprender cómo funcionaba pero creemos que al final hemos conseguido realizar un programa usable y robusto. Hemos visto que la placa tenía muchas otras funcionalidades que con más tiempo podían haber sido implementadas, como por ejemplo la pantalla táctil, el sonido o el teclado.

Entre los problemas que nos hemos encontrado podemos decir que el primero de ellos fue la configuración del timer, ya que en vez de realizar máscaras para los registros, sobreescribíamos estos produciendo con ello efectos laterales que alteraban la funcionalidad. También porque en la función de inicialización, se modificaban registros sin realizar ninguna máscara, por lo cual tuvimos que cambiarlo.

El otro punto en el cual encontramos numerosas dificultades fue a la hora de eliminar los rebotes de los pulsadores ya que cada placa del laboratorio tenía un tiempo de estabilización distinto. Por otro lado tuvimos dificultad a la hora de implementar la máquina de estados en C, ya que nunca habíamos realizado algo similar.

Pasando ya a realizar la interfaz gráfica, tuvimos problemas para familiarizarnos con las dimensiones de la pantalla y a la hora de transformar caracteres a enteros y viceversa, ya que nos fue necesario importar de internet una función que pasara de enteros a caracter (itoa). Sin salir de este problema, algo parecido sucedió a la hora de pintar un número por pantalla, ya que aprovechándonos de la flexibilidad de C, declarábamos los caracteres con el ASCII correspondiente pero a la hora de pintar introducía basura, al final nos dimos cuenta de que esto sucedía porque no había un ‘\0’ que finalizara la cadena, por lo cual nos fue necesario una lookup table.

Llegando casi al final del proyecto, nos costó un poco comprender el código del 44binit.asm para poder cambiarlo y meterlo a la placa.

Por último pero no menos importante, una parte de los problemas han sido por placas en mal estado o insuficiencia de estas según el número de gente que había en el laboratorio en algunas sesiones cercanas a la entrega, especialmente en la práctica 2.

Aún con todo esto creemos que nos hemos organizado de manera correcta y hemos conseguido terminar todo a tiempo esforzándonos en ello.

Creemos que esta asignatura nos ha aportado bastantes conocimientos de hardware, haciendo así nuestra formación más completa.