Universidad Zaragoza

Proyecto Hardware
Trabajo de la asignatura

Implementación de un juego sudoku

Guillermo Robles González - NIP: 604409

Supervisado por:
Javier Resano Ezcaray (coordinador)
María Villarroya Gaudó
Enrique Torres Moreno
Jesús Alastruey Benedé
Darío Suárez Gracia

16 de diciembre de 2015

Pagina intencionalmente en blanco.

Índice

1. Resumen

2. Introducción

Apoyandonos sobre el trabajo desarrollado en la practica anterior, en la cual se probaron distintos algoritmos para comprobar la correctitud de sudokus, el desarrollo de esta practica consistió principalmente en la eliminación del problema de los rebotes la creación de una librería grafica particular (sudoku_graphics) encargado de actuar de Proxy entre el método Main y la librería Lcd, permitiendonos generalizar el método Main, de tal forma que en la situación en la cual se desee llevar este proyecto a otro hardware solo se tendría que cambiar el modulo Proxy (sudoku_graphics), y no será necesario modificar la función Main. Esta propiedad de generalización ha sido uno de los objetivos primarios de este proyecto, intentando siempre que sea posible el mover cualquier función generalizable a su lugar adecuado (Como ejemplo practico de esto, en el marcaje de errores se ha elegido el marcar en negativo las casillas error, para ello se creó una función que inicialmente estaba en el Main, pero primero fué movida a sudoku_graphics, y finalmente fué completamente generalizada y movida a Lcd, de esta forma si en algún momento se ha de desarrollar otro proyecto con este hardware, la librería ya esta preparada para soportar más operaciones)

3. Objetivos

Los objetivos se organizan en 2 conjuntos, los objetivos propuestos por el profesorado, y un conjunto de objetivos autopropuestos por el alumno.

Objetivos del profesorado:

- Evitado de rebotes en los botones.
- Implementación de un juego de sudoku.
-
- Conocimiento de un sistema hardware particular.

Objetivos del alumno:

- Generalización de funciones y funcionalidades.
- Organizado del código adecuada, uso de archivos .h, separación del sistema en los módulos adecuados.
- Uso de un sistema de documentado en código estandar (Doxygen)
- Uso de un sistema de control de versiones (Git)

4. Partes del proyecto

4.1. Trabajo previo

El trabajo a realizar requiere un conjunto de funcionalidades previas, que pese a no estar explicitamente incluidas en el producto final, nos permiten desarrollar adecuadamente las secciones siguientes. Estas funcionalidades son:

- Conocimiento del funcionamiento de los timers, y uso de los mismos tanto para medidas de tiempo como para programar tareas.
- Implementado de una sencilla pila de debug, que nos permite almacenar estampillas temporales e informacion deseada, y nos simplifica por ejemplo medir cual es el tiempo que duran los rebotes.
- Manejo de excepciones, tanto para informar al programador como para eventualmente resolveralas y recuperarnos de las mismas.

4.1.1. Timers

El control de sucesos temporales es vital, tanto para ejecutar funciones periódicamente (por ejemplo, capturas de datos) o con un cierto retardo (por ejemplo, esperas).

En el proyecto, el uso de los timers en los rebotes es usado tanto para, y basado en el proyecto dado (Un sencillo programa que parpadea alternativamente 2 leds) se pudo generalizar una simple librería de tiempos que cumplía las caracteristicas necesitadas. El timer usado es el 2, lo cual deja también el 3 inutilizable

La unidad de medida inicial elegida fueron los µsegundos, que posteriormente fué cambiado a milisegundos, dado que en la medida de tiempos de juego (que pueden llegar a las horas) ocurrían problemas de overflow.

4.1.2. Pila de Debug

Una pila (lo llamamos pila pero la estructura implementada sería más cercana a una cola circular) de Debug es una zona especial de la memoria, en la cual introducimos información que podemos consultar, que en nuestra situación es usada para reconocer y ordenar sucesos temporales, como excepciones o interrupciones, que guarda una estampilla temporal, un suceso e información extra acerca del mismo.

En nuestro caso la pila se ha escogido de tamaño 20 sucesos, situada en la zona de las pilas, por debajo de la pila de usuario, de tal forma que podamos localizarla con facilidad.

Esta pila utiliza el Timer programado previamente para la generación de las estampillas temporales, por lo que para su correcto funcionamiento el módulo Timer2 ha de ser inicializado por el usuario de esta pila.

4.1.3. Excepciones

Las excepciones son sucesos inesperados, generalmente asociados a situaciones de error, que nos permiten reconocer situaciones inesperadas, y advertir al programador, al usuario o incluso intentar recuperarnos de la misma, tanto para continuar el programa como para finalizarlo suavemente.

El procesador ya incluye un sencillo sistema de gestión de Excepciones, que simplemente atasca o reinicia la placa. manejar este sistema simplemente se han de cambiar los punteros `pISR_UNDEF`, `pISR_SWI`, `pISR_PABORT`, `pISR_DABORT` a una función, que será llamada en la situación de que salte alguna de las excepciones. Las excepciones que podemos tratar en el procesador usado son:

Puntero	Excepción que lo llama	Descripción
`pISR_RESET`	Reset	Reset por hardware de la placa
`pISR_UNDEF`	Undefined Instruction	Opcode no reconocido
`pISR_SWI`	Software Interrupt(SWI)	Lanzado mediante la instrucción swi
`pISR_PABORT`	Prefetch Abort	Error al realizar el fetch de la instrucción
`pISR_DABORT`	Data Abort	Error al leer los argumentos de una instrucción

Para reconocer la excepcion en la cual nos encontramos se puede mirar el modo de procesador, dado que el lanzado de ciertas excepciones fuerza el procesador a modos particulares, por lo que mirando el modo actual se puede detectar la excepción lanzada, tanto para intentar recuperar (por ejemplo, en el caso de SWI) como para poder reconocer y guardar la excepción ocurrida.

Una situación en la cual no se puede reconocer la excepción por el modo es entre `pISR_PABORT` y `pISR_DABORT`, dado que ambas pasan al modo `Abort`; o entre `pISR_RESET` y `pISR_SWI` dado que ambas pasan al modo `Supervisor`; en estos casos la única posibilidad es utilizar varias funciones gestoras, de tal forma que por la propia función en la que estemos nos indique que excepción estamos manejando.

Dado el alcance de este proyecto, no se ha decidido complicar la librería de gestión de excepciones con estas consideraciones, pero en una situación en la cual este módulo quiera generalizarse, habría que tener en cuenta tanto la gestión correcta de algunas excepciones (por ejemplo `pISR_RESET`) como la recuperación de aquellas que lo sean (como `pISR_SWI`).

4.2. Rebotes en los botones

El problema de los rebotes ha sido uno de los primeros problemas a los que tenemos que enfrentarnos en un proyecto de estas caracteristicas, y uno de los más complejos.

Este problema aparece cuando nos enfrentamos a hardware (botones) reales. Mientras un botón teórico da una señal similar a la siguiente: Un botón real sigue un patrón más cercano a este:

Estos rebotes son un problema grave, dado que mientras el usuario cree que ha pulsado el botón una única vez (asumiendo como una vez una pulsación y un soltado) el sistema ha recibido múltiples pulsaciones, y por tanto reacciona a todos ellos (en nuestro caso, el sistema incrementa su contador interno múltiples veces).

Existen múltiples soluciones a este problema, tanto por hardware como por software.

Dentro de los solucionadores por hardware aparecen 2 grandes grupos, los basados en SR latches y los basados en circuitos RC, ambos suficientemente efectivos para la mayoría de situaciones, y ampliamente usados en circuitos reales.

Desafortunadamente, el hardware escogido no tiene estos sistemas, lo cual nos fuerza a utilizar solucionados por software. Al igual que para el solucionado por hardware, para el solucionado por software existen múltiples sistemas, entre los que encontramos:

- Sencillo sistema de polling, mediante el cual se compruebe el estado del botón con una cierta frecuencia (por ejemplo 250 ms) y si el botón esta pulsado, se realiza la acción asociada.
- Contar el número de "pulsaciones" (pulsaciones reales y rebotes), y solo realizar la acción cada x pulsaciones.
- Iniciar un contador en el momento en el que llegue una "pulsación", e ignorar cualquier pulsación en los siguientes x milisegundos.

Para el proyecto en cuestión se ha elegido el último algoritmo, que, siendo relativamente simple de implementar, nos ofrece bastante protección frente a los rebotes.

En nuestro caso, se ha decidido realizar una modificación al algoritmo simple, añadiendo la posibilidad de autorepetición, es decir, que el botón actuará mientras sea pulsado (a una frecuencia deseada).

Habiendo elegido el algoritmo, es necesario decidir como se implementará. Para ello, se genera la siguiente máquina de estados:

En nuestro caso, el botón solo reacciona a 2 señales, la interrupción de pulsado (indicada en el diagrama con PULSA) y la interrupción de reloj interna (ofrecida por el Timer4, con una frecuencia de 10ms, indicada como CLK(10ms)).

No se ha encontrado ningún problema destacable en el implementado, a excepción quizás de la determinación de los valores TRP y TRD (ciclos de 10ms de espera en pulsado y en soltado, respectivamente). Para la determinación de los mismos se ha usado la pila de Debug ya comentada, que nos permite conocer el número de interrupciones que lanza los botones en cuestión en que intervalo de tiempo. Cabe destacar que se han elegido unos tiempos algo mas grandes de lo estrictamente necesario, sin embargo, se ha tomado esta decisión para permitir que el algoritmo pueda comportarse adecuadamente en las distintas placas en las cuales se ha probado, aunque se pierda algo de eficacia en situaciones en las cuales el usuario pulse los botones a gran velocidad (situaciones que, de todas formas, no tienen razón para aparecer en el sistema desarrollado).

4.3. Interaccion y Ejecución del sistema

La interacción se realiza principalmente a través de los 2 botones de la placa. Por comodidad, el juego se ha separado en un conjunto de menús, cada uno con una función asociada en sudoku_graphics, estos menús son `title_screen`, `final_screen`, `aperture`, `instructions`, `sudoku`. Cada menú tiene asociado un estado, a excepción del menú `sudoku`, que contiene 3 estados (`esperando_fila`, `esperando_columna`, `esperando_valor`). El menú `aperture` contiene a su vez 119 subestados, que consisten en las distintas líneas de los créditos, pero no se consideraba que mereciera la pena el añadir otros 119 estados al espacio de estados actual. Este subestado es mantenido por la variable `iterador_aperture`.

La organización de este espacio de estados se muestra en el siguiente diagrama de estados:

4.4. Gráficos

Para la gestión de los gráficos se ha intentado desplazar todo el sistema al módulo sudoku_graphics, que se encarga de gestionar toda la comunicación con la pantalla, ofreciendo una interfaz de alto nivel (con funciones como sudoku_graphics_fill_from_data(cuadricula),

que rellena la pantalla con los numeros de la cuadrícula pasada). Este modelo de abstracción nos permitiría reutilizar el codigo de la función Main en caso de desear llevar un sistema similar.

En cualquiera de los casos en los que ha sido posible, se ha utilizado el sistema de macros del preprocesador de C para crear código adaptable, de tal forma que si se deseara cambiar la resolución de funcionamiento, o cambiar la fuente de escritura, se pueda hacer sin más que cambiar las pocas macros que describen tamaños relativos y posiciones. Esto tiene el efecto de que la cuadricula en la pantalla no es tan grande como pudiera, dado que para poder adaptarse correctamente, las medidas han de cumplir ciertas propiedades (este problema se describe con más detalle en el propio fichero fuente `sudoku_graphics.h`)

La actualización de los graficos es uno de los temas que mas complicaciones ha traido, mas exactamente la decisión del momento de actualización de la pantalla.

En el proyecto inicialmente mandado, solo existían 2 posibles fuentes de actualización de la pantalla (el momento en el cual el usuario introducía un número y el momento en el que avanzaba el reloj), por lo que se podía montar un sistema de actualización a demanda, en el cual simplemente se actualizan ciertas zonas de la pantalla en ciertos momentos (Por ejemplo, la zona en la que mostramos el reloj en el momento en el que este cambie).

Sin embargo, algunas de las extensiones que se han decidido realizar (principalmente, la adición de un cursor, que permite eliminar completamente el 8 segmentos del proyecto; y también la adición de algunas animaciones simples) provocan que aparezcan más fuentes de posible actualización, por lo que se tomó la decisión de cambiar a un sistema de actualización constante, que refresca la pantalla siempre que sea posible, eliminando todo el contenido y recalculándolo de nuevo. Esto tiene algunos efectos negativos, entre ello, si cambia el segundo durante una actualización de la pantalla, un segundo en la pantalla puede durar ligeramente más que un segundo real, pero como se mantiene perfectamente la cuenta interna, se considera un problema asumible. Otro de los problemas que aparece es una ligera sensación de parpadeo, pero dado que solo aparece en uno de los sistemas de prueba, tampoco se considera alarmante.

Para la organización del sistema se a utilizado un modelo-vista-controlador, en la cual el controlador es proporcionado por la función Main, que se encarga de actualizar la vista, consistente en la pantalla, siguiendo la información del modelo, que se reparte entre el modulo botón y la función Main (esto es contrario a la filosofía de encapsulamiento, y provoca que, por ejemplo, el modulo Boton no sea todo lo general que pudiera, esto se comenta con más profundidad en la sección 7.1). Los beneficios de usar un modelo como este son inmediatos, dado que nos permite manejar con comodidad múltiples fuentes de información (botones, sudoku, tiempo...) y múltiples presentaciones de la información (pantalla, cuadricula...), y mantenerlos sincronizados sin problemas.

5. Metodología y Tecnologías

Para la organización de tareas a alto nivel, se ha utilizado un modelo orientado a objetivos, de tal forma que cada objetivo propuesto se dividía en un conjunto de subobjetivos, repitiendo el proceso de manera recursiva hasta llegar a tareas que se consideraban asumibles por sí solas, las cuales eran asignadas una prioridad, y a continuación eran añadidas al conjunto de tareas

para hacer.

Dentro de cada día o sesión de trabajo, se seleccionaban las tareas de más prioridad del conjunto de tareas, y utilizando un sistema de planificación a corto plazo basado en la conocida tabla kanban, se conseguía una visualización clara de las tareas actuales, además de permitir organizar fácilmente las tareas por prioridad, evitando que el desarrollo divague.

Respecto a la organización y priorización de funcionalidades, se a optado por jugabilidad y extensibilidad antes que optimización. En el proyecto, cada fichero .h representa uno de los módulos, a excepción de la carpeta resources/, que almacena los BitMap usados y algunas de las cadenas largas, para separar los datos del código.

Para la realización del presente documento se ha seleccionado el sistema de maquetación y control tipográfico TEX, con ayuda del conjunto de macros LATEX. Pese a ser algo más complejo que otros sistemas de maquetado del mercado (como suites ofimáticas o Scribus), ofrece un nivel de calidad bastante mayor, además de tener características especialmente orientadas al desarrollo de documentos de índole técnica o científica, como el presente.

Para el control de versiones se ha elegido el sistema Git, utilizando el servicio GitHub, que ofrece la posibilidad de gestionar repositorios online, ademas de un conjunto de herramientas extra que simplifican el desarrollar un proyecto de cierta magnitud (Repositorio). Dado que los ordenadores del laboratorio en el cual se ha desarrollado la practica no disponia del software necesario, ha sido imposible seguir ninguno de los flujos de trabajo recomendados. Pese a ello, el sistema de control de versiones ayudó a resolver varios problemas durante el desarrollo (principalmente, https://en.wikipedia.org/wiki/Software_regression)

Como entorno de desarrollo se ha utilizado Eclipse, dado que era el ofrecido por el profesorado, con la ToolChain GNU, que nos permitía la compilación para sistemas particulares (En nuestro caso el procesador ARMv7 de la placa de pruebas)

Como sistema/estandar de documentación en codigo se ha elegido Doxygen, y aunque por falta de tiempo y fuerza de trabajo no ha sido posible documentar adecuadamente todos los módulos usados.

6. Resultados y características

El sistema, aparte de las características mandadas por el profesorado, tiene un conjunto de características que, aunque no explícitamente mandadas, se considera que mejoran la experiencia del usuario. Entre dichas caracteristicas aparecen:

- **Módulo Lcd:** El módulo ofrecido por el profesorado era bastante pobre, por ello se ha extendido con algunas funciones que nos permiten generalizarlo más. Algunas de las mejoras propuestas han sido funciones que ofrecen más capacidades a una funcionalidad existente, como el dibujado de líneas punteadas. Otra de las funcionalidades experimentadas (aunque no realmente usadas en la practica) es la conversion de la función `Lcd_Dma_Trans()` en una función no bloqueante, mediante la eliminación de la espera frente al flag `ucZdma0Done`. Esto provoca otro conjunto de posibles problemas (principalemente de sincronización, dado que hemos de evitar el escribir en el buffer virtual mientras se realiza el DMA, bajo riesgo de provocar problemas como el tearing, que complican desproporcionadamente la gestión del renderizado) que se considera que escapan al nivel de esta practica.

- **Pantalla de menú principal:** Además de ofrecer una interfaz de bienvenida más

amigable al usuario, nos permite seleccionar la visualización de instrucciones o de jugar directamente, esto evita al usuario el tener que releer las instrucciones cada vez que desee jugar. Como característica de Debug, preparada exclusivamente de cara a la presentación, se ofrece la posibilidad de escoger una cuadricula casi resuelta, de tal forma que se pueda observar la reacción del sistema frente a la resolución del sudoku, sin necesidad de resolverlo del todo, o utilizar atajos.

- **Cursor e información en pantalla:** Ambos elementos juntos permiten abandonar completamente el 7 segmentos, dando al usuario toda la información necesaria de otro modo (la casilla seleccionada mediante el cursor, y el valor a escribir mediante la pantalla). Actualmente el 7 segmentos esta únicamente activo por la información de Debug que ofrece. Ademas de un cursor para seleccionar la casilla, se usa un cursor algo más pequeño, que se superpone al mostrado de los errores, y permite ver rápidamente si el número que se va a introducir esta dentro de los candidatos.

- **Múltiples cuadriculas:** Se ha añadido todo el sistema necesario para la inserción de múltiples cuadrículas. En el sistema mostrado se ofrece como ejemplo una cuadricula a resolver y una cuadricula de Debug. Uno de los problemas comprobados es el gran tamaño que ocupa una cuadrícula (Actualmente, una ocupa 2304 bits, o 288 bytes, que aunque inicialmente parece poco, si se desea introducir 1000 posibles tablas, comenzamos a movernos sobre los 281 Kib). Para gestionar esto, se plantea la posibilidad de utilizar un sencillo algoritmo de compresión, que nos permitiría reducir el espacio ocupado en un factor de 7[1]. Esto introduciría un pequeño coste en el sistema de decodificación de la cuadrícula compresa, pero se considera un coste asumible.

  La otra posibilidad para ofrecer la posibilidad de usar distintas tablas en cada partida es la generación procedural de las mismas, sin embargo, esta posibilidad introduce otro conjunto de problemas (generación de numeros aleatorios, creación de cuadriculas, eliminación de números de cuadriculas para crear distintas dificultades...) que se considera que escapan al alcance de esta practica.

- **Adición de pantalla final y creditos:** Otra característica que, aunque no mejora realmente el propio juego, ofrece una mejor experiencia de uso, dado que los creditos solo se muestran en el caso de resolver correctamente el sudoku, provocando una reacción de búsqueda del premio. No estrictamente relacionado, los créditos muestran la gestión de alguna animación simple, más específicamente la de texto moviéndose. Para la gestión de animaciones se utiliza el reloj maestro Timer2, y mediante un sencillo sistema de conteo, se decide cuando se debe cambiar de cuadro, provocando la sensación visual de movimiento.

7. Conclusiones

La finalización del trabajo fué satisfactoria, cumpliendo la mayoría de los objetivos propuestos.

[1]Exactamente el sistema consistiría en recortar exclusivamente los primeros 4 bits de cada cuadro en la cuadricula de tal forma que un numero ocupe solo 4 bits, y eliminar el padding de 7 columnas extra introducidas en la practica 1, por lo que una tabla completa ocuparía solo 41 bytes, y nuestras 1000 tablas previas se moverían ahora sobre los 40 Kib, lo cual es una mejora importante

7.1. Margen de mejora

- El módulo Boton no es suficientemente general, dado que esta integrado tanto el control del 7 segmentos como algunas variables de control (next, update...). Esto se podría arreglar haciendo que en el constructor el botón reciba 2 punteros a función, que serán las que llamará cuando sea pulsado alguno de los botones; además de un conjunto de parámetros (Si el botón tiene autorepetición, tiempo de repetición...). No se ha realizado dado que se considera que complicaría el trabajo innecesariamente.

- Muchas de las funciones creadas para tratar textos, tanto fijos como en movimiento, podrían ser refactorizadas a su propio módulo TextUtils, persiguiendo el ya mencionado objetivo de una correcta organización del código. No se ha realizado por falta de fuerza de trabajo.

- Por falta de tiempo, no ha sido posible añadir música ni efectos de sonido. No ha sido posible por la dificultad del mismo.

- Las animaciones de los creditos son bastante crudas, y tienen posibilidad de mejorar. Una opción sería añadir suavizado, provocando que los créditos se muevan más lentamente.

8. Bibliografía

- Manuales de consulta ofrecidos por el profesorado
- Proyectos de la placa ofrecidos por el profesorado
- `http://www.eng.utah.edu/~cs5780/debouncing.pdf`
- `http://infocenter.arm.com/help/index.jsp` (Especialmente el manual de referencia de ARM7)
- `http://www.sudoku-solutions.com/`