****

*Prácticas 2-3*

Proyecto Hardware. Curso 2021-22

Juan Plo Andrés 795105 · Ignacio Ortega Lalmolda 610720

EINA - UNIZAR

7 de enero de 2022

# ÍNDICE DE CONTENIDOS

[1 ÍNDICE DE CONTENIDOS 1](#_Toc85714100)

[2 RESUMEN EJECUTIVO 2](#_Toc85714101)

[2.1 Resumen 2](#_Toc85714102)

[3 INTRODUCCIÓN 3](#_Toc85714103)

[3.1 Mapa de memoria del LPC2105 4](#_Toc85714104)

[4 OBJETIVOS 5](#_Toc85714105)

[4.1 Objetivo del proyecto 5](#_Toc85714106)

[5 METODOLOGÍA 6](#_Toc85714107)

[5.1 Pasos realizados 6](#_Toc85714108)

[5.2 Esquema del proyecto 6](#_Toc85714109)

[5.2.1 Ficheros del proyecto 6](#_Toc85714110)

[5.2.2 Funciones que componen el proyecto 7](#_Toc85714111)

[5.2.3 Código desarrollado en C 8](#_Toc85714112)

[5.2.4 Código desarrollado en ARM y optimizaciones realizadas 10](#_Toc85714113)

[6 RESULTADOS 19](#_Toc85714114)

[6.1 Resultados con la opción de compilación -O0 19](#_Toc85714115)

[6.1.1 Resultados obtenidos 19](#_Toc85714116)

[6.1.2 Análisis de rendimiento 19](#_Toc85714117)

[6.2 Resultados con la opción de compilación -O1 20](#_Toc85714118)

[6.2.1 Resultados obtenidos 20](#_Toc85714119)

[6.2.2 Análisis de rendimiento 20](#_Toc85714120)

[6.3 Resultados con la opción de compilación -O2 20](#_Toc85714121)

[6.3.1 Resultados obtenidos 20](#_Toc85714122)

[6.3.2 Análisis de rendimiento 21](#_Toc85714123)

[6.4 Resultados con la opción de compilación -O3 21](#_Toc85714124)

[6.4.1 Resultados obtenidos 21](#_Toc85714125)

[6.4.2 Análisis de rendimiento 21](#_Toc85714126)

[7 CONCLUSIONES 22](#_Toc85714127)

# RESUMEN EJECUTIVO

## Resumen

El trabajo desarrollado ha consistido en la realización de código para un sistema empotrado para el procesador LPC2105, en concreto, se la aplicación que se ha desarrollado ha sido la del juego del Sudoku con entrada por parte del usuario de las celdas y los valores a introducir en ellas para el desarrollo del juego.

Nuestro código se divide en una serie de módulos básicos entre los cuales se organizan de manera jerárquica. Nuestra arquitectura ofrece una capa que interaccionan con el hardware, una capa de manejadores que se encargan de la interacción con los periféricos, otra con la aplicación (el sudoku) y un planificador que va encolando y descolando los eventos que se van generando en el sistema.

# INTRODUCCIÓN

El trabajo realizado ha consistido en el desarrollo de un sistema empotrado real para el procesador LPC2105 interaccionando con los diferentes dispositivos de entrada/salida del procesador. El entorno de trabajo empleado ha sido el emulador Keil μVision el cual también simula el procesador con el que hemos trabajado.

~~El trabajo realizado ha consistido en la implementación y optimización del rendimiento una serie de funciones para ayudar al cálculo de los candidatos de los huecos vacíos en un tablero de sudoku para, posteriormente, analizar los resultados obtenidos. A partir de un código original en C, se ha implementado el código equivalente en ensamblador para el procesador LPC2105, y se ha comprobado el rendimiento de las distintas soluciones. El entorno de trabajo empleado ha sido el emulador Keil μVision el cual también simula el procesador con el que hemos trabajado.~~

~~Las reglas del sudoku que se emplean en la implementación son sencillas, el objetivo es el de rellenar las celdas vacías en un tablero (9x9) de tal modo que un número del 1-9 sólo pueda aparecer una vez en su fila, su columna y en su región (3x3). Nuestro programa calculará los candidatos posibles de cada celda del tablero(9x9) de acuerdo a las reglas descritas anteriormente para ayudar a los jugadores a tomar la decisión de que valor puede ir en cada celda.~~

# OBJETIVOS

## Objetivo del proyecto

~~El objetivo principal del proyecto es el de poder comparar los distintos niveles de compilación en C y compararlos con nuestra versión ARM para detectar cual es la función más costosa en tiempo y comprobar si el compilador es más rápido y está mejor optimizado que nuestro código ARM según los diferentes niveles de compilación:~~

* ~~-O0: Sin ninguna optimización~~
* ~~-O1: Pequeñas optimizaciones tratando de reducir el tamaño del código generado, pero no que no reduce apenas el tiempo de ejecución. Opción de compilación intermedia entre -O0 y -O2.~~
* ~~-O2: Más optimizaciones que -O1~~
* ~~-O3: Todas las optimizaciones posibles. Aumenta el tamaño del código.~~

~~Se comparará según los niveles de compilación en tiempo y memoria para determinar si merece la pena el esfuerzo de escribir el código en ARM o, si bien, el trabajo realizado por el compilador es suficiente.~~

~~También se comprobará que todas las versiones del código tienen una funcionalidad equivalente y que todas las versiones obtienen el mismo resultado tras su ejecución variando únicamente en tiempo y en espacio.~~

# METODOLOGÍA

## Pasos realizados

Lo primero que se ha desarrollado para el proyecto ha sido la cola de eventos. El programa principal la llama y esta se queda en un bucle infinito encolando y desencolando eventos. A continuación, se ha realizado la gestión de los temporizadores el timer11reloj del sistema y el timer0, empleado para el encolar eventos periódicos. La siguiente parte realizada fue la configuración del GPIO para el manejo de la entrada salida con el usuario y la gestión de las interrupciones con la gestión de las pulsaciones del usuario. Se ha realizado la gestión de energía del procesador, poniéndolo en modo IDLE si el planificador no tiene nuevos eventos o en powerdown si tras 15 segundos no hay cambios en la entrada. Finalmente se ha integrado la aplicación del sudoku con la entrada salida del procesador.

Bla bla bla……

## Esquema del proyecto

### Ficheros del proyecto

En la siguiente tabla se resumen los ficheros que componen el proyecto:

|  |  |  |
| --- | --- | --- |
| FICHEROS EN EL PROYECTO | | |
| Fichero | Líneas | Descripción |
| startup.s | 311 | Fichero de inicio |
| SWI.s | 105 | Fichero para las software interrupt |
| softI.h | 15 | Fichero de caberceras con los identificadores de la software interrupt |
| main.c | 6 | Fichero con el programa principal que realiza una llamada a planificador\_main() |
| cola.h | 60 | Fichero de cabeceras de cola.c |
| cola.c | 85 | Funciones y tipos de datos de la cola |
| evento.h | 56 | Fichero de cabeceras de evento.c y tipos de evento |
| evento.c | 14 | Funciones para los eventos |
| planificador.h | 13 | Fichero de cabeceras de planificador.c |
| planificador.c | 156 | Función del planificador que inicializa el sistema y va gestionando los eventos |
| gestor\_IO.h | 66 | Fichero de cabeceras de gestor\_IO.c |
| gestor\_IO.c | 92 | Funciones de los manejadores del sistema de entrada/salida por el GPIO |
| gestor\_ pulsacion.h | 29 | Fichero de cabeceras de gestor\_pulsacion.c |
| gestor\_ pulsacion.c | 125 | Funciones de los manejadores del sistema de las pulsaciones en EINT1 y EINT2 |
| gestor\_ energia.h | 23 | Fichero de cabeceras de gestor\_ energia.c |
| gestor\_ energia.c | 28 | Funciones de los manejadores del sistema para gestionar los modos IDLE y power down |
| gestor\_ alarmas.h | 64 | Fichero de cabeceras de gestor\_ alarmas.c |
| gestor\_ alarmas.c | 160 | Funciones de los manejadores del sistema para gestionar las alarmas |
| sudoku\_2021.h | 30 | Fichero de cabeceras de sudoku\_2021.c |
| sudoku\_2021.c | 94 | Funciones que interactúan directamente con el sudoku |
| celda.h | 123 | Contiene las operaciones que se aplican a una celda |
| tableros.h | 94 | Contiene una serie de tableros en hexadecimal con la situación inicial de cada uno para cada versión. También incluye un tablero solución para verificar la corrección de cada versión ejecutada. |
| gpio.h | 38 | Fichero de cabeceras de gpio.c |
| gpio.c | 97 | Funciones dependientes del hardware que interactúan con el GPIO |
| buttons.h | 30 | Fichero de cabeceras de buttons.c |
| buttons.c | 70 | Funciones dependientes del hardware que interactúan con los botones EINT1 y EINT2 |
| idle\_pwdwn.h | 21 | Fichero de cabeceras de idle\_pwdwn.c |
| idle\_pwdwn.c | 22 | Funciones dependientes del hardware que ponen al procesador en modo idle o powerdown |
| timers.h | 37 | Fichero de cabeceras de timers.c |
| timers.c | 95 | Funciones dependientes del hardware que configuran el timer 0 y el timer 1 |
| relojes.h | 39 | Fichero de cabeceras de relojes.c |
| relojes.c | 73 | Funciones dependientes del hardware que configuran el RTC y el Watch Dog |
| serial\_port.h | 86 | Fichero de cabeceras de serial\_port.c |
| serial\_port.c | 482 | Funciones dependientes del hardware que configuran la UART |

## Arquitectura del sistema

La arquitectura del sistema que se ha desarrollado sigue el siguiente esquema, en donde cada módulo se comunica o bien con otro módulo, o bien con el planificador a través del encolado y desencolado de eventos.

### Esquema de los módulos que componen el sistema

En la imagen se muestra como se ha organizado la comunicación entre módulos de la aplicación:

DEPENDIENTE DEL HARDWARE MANEJADORES MICRO KERNEL APLICACIÓN

|  |
| --- |
|  |
| Figura 1: Esquema de los módulos |

En las siguientes tablas se incluyen las funciones que componen cada fichero, así como una pequeña descripción de su comportamiento y parámetros para cada capa del sistema.

### Aplicación

|  |  |  |
| --- | --- | --- |
| SUDOKU | | |
| Fichero | Funciones | Descripción |
| sudoku.c  sudoku.h | candidatos\_actualizar\_c | Versión del código en C. Recibe como parámetro una celda. Calcula todas las listas de candidatos (tras borrar o cambiar un valor) y llama a *“candidatos\_propagar\_c”.* Devuelve el número de celdas vacías. |
| candidatos\_propagar\_c | Versión en C. Recibe como parámetro una celda, su fila y su columna. Propaga el valor de una celda para actualizar la lista de candidatos de su fila, columna y región. Recibe la celda a propagar como parámetro. |
| tiempo\_actualizar | Devuelve tiempo contado que tarda la funcion candidatos actualizar |
| reset\_tiempo\_actualizar | Pone a 0 el tiempo contado que tarda la funcion candidatos actualizar |
| celda.h | celda\_eliminar\_candidato | Recibe como parámetro el puntero a una celda y un valor. Elimina el candidato que coincida con valor. |
| celda\_poner\_valor | Recibe como parámetro el puntero a una celda y un valor. Pone como valor el valor recibido. |
| celda\_leer\_valor | Recibe como parámetro una celda. Devuelve el valor almacenado en ella. |
| esVacia | Recibe como parámetro una celda. Devuelve 0 si su valor es igual a 0. |
| eliminaCandidatos | Recibe como parámetro el puntero a una celda. Pone a 0 todos sus candidatos. |
| esPista | Devuelve 1 si la celda seleccionada es pista. 0 en caso contrario |
| esError | Devuelve 1 si la celda seleccionada es error. 0 en caso contrario |
| celda\_borrarError | Pone a 0 el bit de error de la celda \*celdaptr |
| celda\_ponerError | Pone a 1 el bit de error de la celda \*celdaptr |
| celda\_noEsCandidato | Devuelve 1 si no es candidato valor en la celda. 0 en caso contrario |
| tableros.h | borrar\_tablero | Borra el tablero para dejarlo solo con las pistas |
| check\_resuelto | Compruba si el tablero ha sido resuelto mirnado que ninguna celda sea error y que todas tengan valor |

### Micro kernel

|  |  |  |
| --- | --- | --- |
| MICRO KERNEL | | |
| Fichero | Funciones | Descripción |
| SWI.s | SWI\_Handler |  |
| \_\_disable\_isr | Desabilita las interrupciones isr |
| \_\_enable\_isr | Habilir las interrupciones isr |
| \_\_disable\_isr\_fiq | Desabilita las interrupciones isr y las fiq |
| \_\_enable\_isr\_fiq | Habilita las interrupciones isr y las fiq |
| cola.c | cola\_crear\_vacia | Crea la cola circular vacía, es decir, sin eventos. |
| cola\_guardar\_evento | Si numEventos < maxEventos devuelve la cola circular resultante de añadir a c el Evento e. Si no, devuelve la cola resultante de añadir a c el Evento e, eliminando el Evento más antiguo de la cola, activando el flag V (Overflow). |
| cola\_desencola\_mas\_antiguo | Si numEventos > 0, devuelve el Evento más antiguo de la cola (el primero) y devuelve la cola circular resultante de eliminar de la cola dicho Evento. En caso contrario, devuelve un Evento inválido. |
| cola\_es\_vacia | Devuelve 1 (verdad) sí y sólo si la cola c no tiene ningún Evento. Devuelve 0 (falso) en caso contrario. |
| cola\_ultimo | Si numEventos > 0, devuelve el Evento más nuevo encolado en la cola (el último). En caso contrario, devuelve un evento inválido. |
| vaciar\_cola | Vacía la cola y crea una nueva vacía |
| evento.c | es\_valido | Devuelve 1 si el evento tiene id 255 InvalidID. Devuelve 0 en caso contrario. |
| el\_invalido | Devuelve un evento inválido |
| planificador.c | planificador\_main | Planificador infinito que va desencolando eventos e inicializa todo el sistema. Si la cola se llena WatchDog reiniciará la partida. |
| main.c | main | Programa principal que llama a planificador\_main |

### Manejadores

|  |  |  |
| --- | --- | --- |
| GESTORES | | |
| Fichero | Funciones | Descripción |
| gestor\_IO.c  gestor\_IO.h | gIO\_inicializar | Inicializa el gestor\_IO iniciclizando la GPIO y pone una alarma periódica que Check\_Entrada para mostrar en el GPIO el valor de la celda seleccionada (valor y candidatos) |
| gIO\_reset | Vuelve a poner una alarma periodica para el latido tras resetear una partida |
| gIO\_encender\_overflow | Escribe el bit de overfow del GPIO para saber si ha desbordado la cola de eventos |
| gIO\_leer\_overflow | Lee el estado del bit de overfow del GPIO para saber si ha desbordado la cola de eventos |
| gIO\_apagar\_validacion | Apaga el led de validación (bit 13 GPIO) |
| gIO\_encender\_validacion | Enciende el led de validación (bit 13 GPIO) |
| gIO\_encender\_latido | Enciende el bit 31 del GPIO |
| gIO\_apagar\_latido | Apaga el bit 31 del GPIO |
| gIO\_alternar\_latido | Alterna el latido del GPIO encendiendo el bit 31 si estaba apagado y apagandolo si estaba encendido |
| gIO\_alternar\_validacion | Alterna la validacion del GPIO encendiendo el bit 13 si estaba apagado y apagandolo si estaba encendido |
| gestor\_pulsacion.c  gestor\_pulsacion.h | gp\_inicializar | Inicializa las interrupciones en EINT1 |
| gp\_leer\_pulsacion\_1 | Devuelve 1 si el estado es PULSADO. Devuelve 0 en caso contrario |
| leer\_estado\_1 | Devuelve el estado de EINT1 |
| leer\_entrada\_1 | Devuelve 1 si el botón sigue pulsado, 0 en caso contrario |
| gp\_actualizar\_estado\_EINT1 | Comprueba si se ha levantado la pulsacion y actualiza el estado. Si se ha levantado la pulsación vuelve a habilitar las interrupciones en el VIC |
| gestor\_energia.c  gestor\_energia.h | ge\_inicializar | Inicia la alarma para poner al procesador a PWDOWN |
| ge\_modo\_IDE | Pone al procesador en modo IDE |
| ge\_modo\_pwdwn | Pone al procesador en modo powerdown |

### Dependiente del hardware

|  |  |  |
| --- | --- | --- |
| DEPENDIENTE DEL HARWARE | | |
| Fichero | Funciones | Descripción |
| gpio.c  gpio.h | GPIO\_iniciar | Inicializa el GPIO |
| GPIO\_leer (uint8\_t bit\_inicial, uint8\_t num\_bits) | bit\_inicial indica el primer bit a leer. num\_bits indica cuántos bits queremos leer. La función devuelve un entero con el valor de los bits indicados. |
| GPIO\_escribir (uint8\_t bit\_inicial, uint8\_t num\_bits, uint32\_t valor) | similar al anterior, pero en lugar de leer escribe en los bits indicados el valor (si valor no puede representarse en los bits indicados se escribirá los num\_bits menos significativos a partir del inicial) |
| GPIO\_marcar\_entrada (uint8\_t bit\_inicial, uint8\_t num\_bits) | Los bits indicados se utilizarán como pines de entrada desde bit\_inicial hasta bit\_inicial + num\_bits. |
| GPIO\_marcar\_salida (uint8\_t bit\_inicial, uint8\_t num\_bits) | Los bits indicados se utilizarán como pines de salida desde bit\_inicial hasta bit\_inicial + num\_bits. |
| buttons.c  buttons.h | eint1\_ISR | ISR para las EINT1 |
| button\_nueva\_pulsacion\_1 | Devuelve 1 si ha habido una nueva pulsación |
| button\_clear\_nueva\_pulsacion\_1 | Resetea a 0 las nuevas pulsaciones |
| button\_enable\_interrupts\_1 | Habilitas interrupciones en VIC |
| eint1\_init | Habilita las interrupciones EINT1 en el pin P0.14 |
| idle\_pwdwn.c  idle\_pwdwn.h | idle | Pone al procesador en modo IDLE para ahorrar energía |
| pwdwn | Pone al procesador en modo powerDown y sólo se despertará por las interrupciones externas marcadas EINT1 |
| timers.c  timers.h | temporizador\_iniciar | Inicializa el timer1 para que pueda ser usado |
| temporizador\_empezar | Inicia la cuenta en el timer1 |
| reset\_timer1 | Resetea el timer 1 |
| temporizador\_leer | Devuelve el tiempo que lleva contado timer1 |
| temporizador\_parar | Para el temporizador y devuelve el tiempo transcurrido |
| temporizador\_peridico | Programa el timer0 para que encole un evento periódicamente en ms |
| relojes.c  relojes.h | RTC\_init | Inicializa el RTC |
| RTC\_reset | Resetea el RTC |
| RTC\_leer\_minutos | Devuelve los minutos que lleva contados el RTC |
| RTC\_leer\_segundos | Devuelve los segundos que lleva contados el RTC |
| WD\_init | Inicializa el WatchDog y se prepara para alimentarlo con una alarma periodica cada 5 seg |
| Alimenta el WatchDog | WD\_feed |

|  |  |  |
| --- | --- | --- |
| serial\_port.c  serial\_port.h | UART\_actualizar | Actualiza el sudoku mostrado en la UART |
| UART\_ha\_terminado | Devuelve 1 si se ha terminado de enviar el string a la UART. 0 en caso contrario |
| UART\_enviar\_candidatos | Escribe por la UART los candidatos de la celda correspondiente a la fila fil y de la columna col.  Si la celda seleccionada era una pista devuelve el mensaje "La celda seleccionada es una pista" |
| UART\_introducir\_jugada | Procesa la jugada que el usuario quiere introducir en la celda correspondiente a la fila fil y la columna col. Si la celda seleccionada era una pista devuelve el mensaje "La celda en la que quieres introducir valor es una pista"  Si la juagada es valida, envia el mensaje "¿Confirmar jugada?" la realiza, pone una alarma a 3 seg para que el usuario valide la juagada y enciende con otra alarma el led de latido. |
| UART\_acaba\_jugada | Apaga el led de latido de validación con una alarma con su tiempo a 0 |
| UART\_cancelar\_jugada | Si el usuario no valida la jugada, se cancela y muestra el mensaje "Jugada cancelada" |
| UART\_init\_serial | Inicializa las interrupciones de la UART0 |
| UART\_enviar\_string | Manda el string a la UART. Copia el string en un buffer send\_buffer y manda el primer caracter del buffer a la UART |
| UART\_borrar\_tablero | Borra el tablero y recalcula los candidatos |
| en\_jugada | Devuelve 0 si estamos en medio de una jugada, 1 en caso contrario |
| UART\_reset | Prepara la UART para un reinicio de partida |

## Descripción ……………

### Planificador, cola y eventos

Para comenzar a bordar el diseño del planificador se desarrolla el elemento más importante, la cola de eventos, definida en cola.h y cola.c. esta será la responsable de ir encolando y desencolando los eventos que nos vayan llegando, sin embargo, antes del diseño de la cola pensamos en como representar los eventos que guardará.

1. Eventos

Para comenzar a bordar el diseño del planificador se desarrolla el elemento más importante, la cola de eventos, definida en cola.h y cola.c. esta será la responsable de ir encolando y desencolando los eventos que nos vayan llegando, sin embargo, antes del diseño de la cola pensamos en como representar los eventos que guardará.

Definimos en evento.h un struct que se compone de lo siguiente:

|  |
| --- |
| /\*  \* Cada Evento es caracterizado por su ID (único), además cada  \* evento tiene una serie de datos asociados y una estampilla temporal que  \* indica cuando el evento fue generado.  \*/  typedef struct Evento Evento;  struct Evento {  uint8\_t ID\_evento; //entero de 8 bits para el identificador del mismo  uint32\_t auxData; //32 bits para información necesaria para tratar con el evento  uint32\_t timestamp; //32 bits que guarda el tiempo en ms que fue creado el evento  }; |

Para cada evento, tenemos una serie de identificadores los cuales definimos mediante una enumeración, poco a poco a medida que los vamos necesitando mientras desarrollamos el proyecto:

|  |
| --- |
| #define INVALID\_ID 255 //ID de los eventos inválidos  /\*  \* Tipo de evento invalido  \*/  static Evento eventoInvalido = {INVALID\_ID, 0, 0};  /\*  \* Tipo para definir el id de cada tipo de evento  \*/  enum {  Temp\_perio = 0,  Set\_Alarma = 1,  Check\_Pulsacion\_EINT1 = 2,  Pulsacion\_EINT1 = 3,  Power\_Down = 4,  Apagar\_Validacion = 5,  Terminar = 6,  Latido = 7,  No\_Confir\_Jugada = 8,  Latido\_Validacion = 9,  Candidatos = 10,  Jugada = 11,  Feed = 12,  Start = 13,  Check\_Terminado\_PWDOWN = 14  }; |

Además, añadimos una función es\_valido(Evento\* e) para saber que eventos son validos, debido a que hay funciones que han de devolver un dato de tipo evento de manera forzosa. Se define un evento inválido con identificador 255 el cual es devuelto en dichas ocasiones.

1. Cola de eventos

Para el diseño de la cola creamos un struct que se compone de lo siguiente:

* Array de "Eventos" de tamaño 32
* Un entero que guarda el índice del evento mas antiguo
* Un entero que guarda el índice del siguiente evento encolado
* Un entero que cuente el numero de eventos guardados en el array

|  |
| --- |
| #define MAX\_EVENTOS 32  typedef struct Cola Cola;  struct Cola {  Evento eventos[MAX\_EVENTOS];  uint8\_t indPrimEv; // Índice del evento más antiguo de la cola  uint8\_t indProxEv; // Índice del siguiente evento a encolar  uint8\_t numEventos;  }; |

Creamos las operaciones básicas para gestionar una cola:

* cola\_crear\_vacia() para crear una cola vacía
* cola\_guardar\_evento(Evento e) para añadir un evento a la cola y en caso de pasarnos activar el led de overflow del GPIO
* cola\_desencola\_mas\_antigua() para sacar de la cola el evento mas antiguo y devolverlo
* cola\_es\_vacia() para comprobar si la cola es vacía, devuelve 1 en caso afirmativo y 0 en caso contrario
* cola\_ultimo() devuelve el evento mas nuevo encolado

1. Planificador

Se trata del programa que va a estar ejecutándose de manera infinita. Su comportamiento es el siguiente:

1. Inicializa todos los componentes del sistema creando la cola vacía, marca las entradas y salidas del GIPO para la interacción con el Sudoku, inicializa el gestor de alarmas, pulsación, energía y pasa a un bucle infinito.
2. Una vez comenzado el bucle infinito, primero se lee el bit de overflow del GPIO para saber si la cola de eventos ha dado error (se ha desbordado) en cuyo caso para la ejecución.
3. A continuación, va mirando si hay eventos en la cola y desencola el más antiguo (cola FIFO) y para cada tipo de evento si es válido realiza las operaciones que tenga que hacer.
4. Si la cola hubiese sido vacía, se pone al procesador en modo IDLE.

### Temporizadores y Gestor de alarmas

1. Temporizadores

Para el desarrollo del sistema se utilizará el timer0 para la gestión de las alarmas y el timer1 para medir el tiempo del sistema.

Creamos las siguientes operaciones para interactuar con los temporizadores en donde llevaremos la cuenta de las interrupciones del timer1 con un entero volátil:

1. Temporizador iniciar(): configura el timer1 para que interrumpa el numero mínimo de veces osea, cada 2^32 - 1 tics de reloj, además de configurar el reloj de los relojes para que trabajen a la máxima frecuencia posible. También habilitamos sus interrupciones y le asignamos su RSI, timer\_ISR() que aumenta la cuenta de interrupciones del timer1
2. Temporizador\_empezar(): pone en marcha el timer1
3. Temporizador\_leer(): devuelve en us el tiempo del sistema
4. Temporizador\_parar(): para el timer1 y devuelve en us el tiempo del sistema
5. Temporizador\_periodico(int periodo): configura el timer0 para generar una interrupcion cada <periodo> ms y se habilita sus interrupciones además de asignarle su RSI
6. timer0\_ISR(): encola un evento de comprobar alarmas en la cola de eventos
7. Gestor de alarmas

El gestor de alarmas es el encargado de iniciar los temporizadores y gestionar las alarmas que se configuran, para ello tiene un vector de ocho alarmas en donde cada uno de sus índices guardan una Alarma definida en gestor\_alarmas.h del siguiente modo:

|  |
| --- |
| /\*  \* Tipo de datos de una alarma  \*/  typedef struct Alarma Alarma;  struct Alarma {  uint8\_t esValida; //indica 1 si es valida 0 no es valida  uint8\_t IDevento;  uint8\_t esPeriodica;  int periodo; //ms a disparar la alarma  int timestamp; //timestamp actual en ms  int timeToLeave; //tiempo en el que se ha de sacar la alarma en ms  }; |

Las funciones incluidas en gestor\_alarmas.c son las siguientes:

1. ga\_inicializar(void): marca todas las alarmas a inválidas e inicia los temporizadores. El temporizador periódico se marca a 50ms, debido a que todas las alarmas que se encolan son múltiplo de 50.
2. ga\_comprobar\_alarmas(void): cada vez que llega un evento de temporizador periódico que desencola el planificador se comprueban todas las alarmas con el instante de tiempo actual para ver si se han de encolar en el planificador. Si ese es el caso, si una alarma no es periódica se desactiva. Si es periódica se actualiza su timeToLeave.
3. actualizar\_alarma(Alarma al): operación interna. Comprueba si una nueva alarma que nos llega está ya en la lista de alarmas. Si está y el tiempo indicado en la nueva alarma es cero, la alarma se desactiva, si no, actualiza el tiempo timeToLeave de la alarma. Si existe la alarma devuelve 1, si no devuelve 0.
4. crear\_alarma(Evento e): operación interna. Crea una alarma a partir de un evento Set\_alarma.
5. ga\_guardar\_evento(Evento e): Crea una alarma a partir de un evento. Si no se ha de actualizar la alarma, la introcice en el vector de alarmas

### GPIO

Para la interacción con el GPIO creamos gpio.h con distintas operaciones:

1. GPIO\_iniciar(): que configura los pines del GPIO para que este de acuerdo al sistema:

bit 31: output | bit 30: output | bit 27-24: input | bit 23-20: input | bit 19-16: input | bit 15-14: input | bit 13: output | bit 12-4 : output | bit 3-0: output

1. GPIO\_leer(uint8\_t bit\_inicial, uint8\_t num\_bits): que devuelve un entero de 32 bits con los bits leidos desde <bit\_inicial> hasta <bit\_inicial> + <num\_bits>
2. GPIO\_escribir(uint8\_t bit\_inicial, uint8\_t num\_bits, uint32\_t valor): que escribe <valor> desde <bit\_inicial> hasta <bit\_inicial> + <num\_bits> y en el caso de que <valor> tuviese mas bits que <num\_bits> escribira los de menos peso.
3. GPIO\_marcar\_entrada(uint8\_t bit\_inicial, uint8\_t num\_bits): configura el GPIO para que los bits desde <bit\_inicial> hasta <bit\_inicial> + <num\_bits> sean leds de entrada.
4. GPIO\_marcar\_salida(uint8\_t bit\_inicial, uint8\_t num\_bits): configura el GPIO para que los bits desde <bit\_inicial> hasta <bit\_inicial> + <num\_bits> sean leds de salida.

### Gestor de pulsación

El gestor de pulsación definido en gestor\_pulsacion.c es el encargado de indicarnos cuando se ha levantado la pulsación del botón por parte del usuario. En la RSI del botón, la primera vez que nos interrumpen se deshabilitan las interrupciones de botón en el VIC para que sólo nos interrumpan una vez. Se crea una pequeña máquina de estados que según la entrada pone a pulsado o no pulsado el estado del botón. Si ha sido despulsado, vuelve a habilitar las interrupciones en el VIC del siguiente modo.

|  |
| --- |
| /\*  \* Comprueba si se ha levantado la pulsacion y actualiza el estado.  \* si se ha levantado la pulsación vuelve a habilitar las interrupciones  \* en el VIC  \*/  void gp\_actualizar\_estado\_EINT1(void) {  unsigned int estado, entrada;  int retardo;  Evento eAlarma;    if (button\_nueva\_pulsacion\_1() == 1) { //ha habido una nueva pulsacion se ajusta el estado  estado\_pulsacion\_EINT1 = PULSADO;  }  estado = leer\_estado\_1();  entrada = leer\_entrada\_1();  switch(estado) {  case PULSADO:  if (entrada == 0) { // se ha despulsado el boton  estado\_pulsacion\_EINT1 = NO\_PULSADO;  button\_enable\_interrupts\_1();  }  break;  case NO\_PULSADO:  if (entrada == 1) { // se ha pulsado  estado\_pulsacion\_EINT1 = PULSADO;  }  break;  }  } |

En donde button\_enable\_interrupts\_1 simplemente habilita en el VIC Y leer\_entrada lee la entrada actual del botón del siguiente modo:

|  |
| --- |
| unsigned int leer\_entrada\_1(void) {  EXTINT = EXTINT | 2; // clear interrupt flag de EINT0, EINT1 y EINT2  if ((EXTINT & 0x2) == 2) {  return 1; //El botón sigue pulsado  }  return 0; //El botón no está pulsado  } |

Podemos también dibujar el siguiente autómata (figura 2) que define el comportamiento de gp\_actualizar \_entrada():

|  |
| --- |
|  |
| Figura 2: autómata para la pulsación de los botones |

### ~~Modos IDLE, PowerDown y gestor de energía~~

1. ~~Modos IDLE y PowerDown~~
2. ~~Gestor de energía~~

### Integración de la I/O en el juego

Como ya se ha mostrado antes se configuran los leds del GPIO para este cometido:

1. GPIO13: led validar jugada
2. GPIO14: EINT1 (escribir dato en celda seleccionada)
3. GPIO30: overflow
4. GPIO31: latido modo idle

### UART

1. Inicialización de la UART

La implementación de la interacción con el usuario se realiza mediante la UART1.

Para ello primero se inicializa la UART1 de la siguiente manera para que nos pueda interrumpir:

|  |
| --- |
| void UART\_init\_serial(void) { // Initialize Serial Interface  PINSEL0 = PINSEL0 | 0x5; // Enable RxD1 and TxD1  U0LCR = 0x83; // 8 bits, no Parity, 1 Stop bit  U0DLL = 97; // Div Latch LSB 9600 Baud Rate @ 15MHz VPB Clock  U0LCR = 0x03; // Line Control Register DLAB = 0    //habilitar interrupciones de uart1  VICIntEnable = VICIntEnable | 0x00000040;  VICVectAddr5 = (unsigned long)serial\_ISR;  VICVectCntl5 = 0x20 | 6;  U0IER = U0IER | 0x3; // Interrupt Enable Register, habilitar las RBR  // (Receiver send\_buffer Register) y las THR  // (Transmit Holding Register)  } |

1. Enviar un string a la UART

Una vez inicializada la UART nos interesa tener una función para poder enviar de golpe un string a la UART y no tener que enviar carácter por carácter:

|  |
| --- |
| void UART\_enviar\_string(char \*string) {  memset(&send\_buffer[0], 0, sizeof(send\_buffer)); //Wipe previous buffer  size\_send\_buffer = 0;  index\_send\_buffer = 0;  terminado = 0;    //Copiar el string a mi 'send\_buffer' y controlar que no nos salgamos del limite  while (string[size\_send\_buffer] != '\0' && size\_send\_buffer < MAX\_SEND\_BUFFER) {  send\_buffer[size\_send\_buffer] = string[size\_send\_buffer];  size\_send\_buffer++;  }  U0THR = send\_buffer[index\_send\_buffer]; //manda el primer caracter a la UART0  index\_send\_buffer++;  } |

En donde se le pasa a la función un char\* que se copia a un buffer (send\_buffer). Una vez copiado dicho buffer se le manda a la UART el primer carácter, entonces, una vez que nos empiece a interrumpir se le mandará el siguiente carácter de ese buffer dentro de la RSI hasta que termine.

1. RSI de la UART

Dentro de la RSI lo primero que se mira es si la RSI ha sido invocada por un mensaje para mostrar por pantalla o bien si ha sido invocada por la entrada de un usuario aplicando una máscara al U0IIR para quedarnos con los bits 3:1 que nos dicen que tipo de interrupción ha sido.

Si se ha producido una interrupción por un imput de un usuario en la UART

* Se actualiza la alarma de powerdown (segumos jugando) y se resetea la cuenta.
* Actualizamos una máquina de estados (que explicaremos más adelante) que comprueba si el carácter introducido es correcto.
* Leemos el carácter recibido en la UART

Sino si ha sido porque queríamos escribir algo en la UART

* Llamamos a continuar\_msj que envía el siguiente carácter del buffer a la UART

En la siguiente tabla podemos observar la ISR de la UART:

|  |
| --- |
| void serial\_ISR (void) \_\_irq {  int mask;  int retardo;  disable\_isr\_fiq();  VICVectAddr = 0;  mask = 0xE; // Nos quedamos con los bits 3:1 nos dicen que tipo de interr ha sido  mask &= U0IIR;    // Recibo datos de UART0 (3:1 = 010) (man pg 88)  if (mask == 0x4) {  if (UART\_ha\_terminado() && jugada == 0){  //Actualizamos la alarma para resetear el countdown para irnos a powerdown  retardo = TIME\_PWDN & 0x007FFFFF; // Asegurarnos que retardo es de 23bits  set\_Alarma(Power\_Down, retardo, 1);  if (index\_rec\_buffer < MAX\_REC\_BUFFER) {  rec\_buffer[index\_rec\_buffer] = U0RBR; //nos da el char introducido en UART  actualizar\_estado(rec\_buffer[index\_rec\_buffer]);  }  //Comprobación de cada tipo de comando para generear el evento correspondiente  if (rec\_buffer[index\_rec\_buffer] == '!') { // Fin del mensaje  if (strcmp(rec\_buffer, "#RST!") == 0) {  // Terminar partida  Evento terminar;  empezado = 0;  terminar.ID\_evento = Terminar;  cola\_guardar\_evento(terminar);  }  (...)  }  // Escribo datos en UART0 (3:1 = 010) (pg 88)  else if (mask == 0x2) {  continuar\_msj();  }  enable\_isr\_fiq();  } |

1. Máquina de estados

Como hemos mencionado anteriormente cada vez que el usuario escribe un carácter en la UART comprobamos que el carácter es válido, es decir, que pertenezca a un comando correcto el cual debe ser:

* #NEW! Para comenzar una partida
* #RST! Para terminarla
* #fcC! Para mostrar los candidatos de una fila ‘f’ y de una columna ‘c’
* #fcvs! Para introducir una jugada en la fila ‘f’ y columna ‘c’ con valor ‘v’ y checksum ‘s’

Si por algún casual el usuario introduce un comando erróneo, la máquina de estados lo detectará y enviará a la UART el string con el mensaje “Instrucción no reconocida” y volverá al estado inicial “ESPERA\_CMD”.

|  |
| --- |
|  |
| Figura 3: esquema de la máquina de estados |

### ~~Código desarrollado en ARM y optimizaciones realizadas~~

~~Respecto a la versión en C una de las optimizaciones con las que cuentan todas las funciones es el “inlining” de todas las funciones de celda.h, es decir, que el cuerpo de las funciones de celda.h está embebido directamente en el código ensamblador, evitando llamadas a subrutinas evitándose así las instrucciones que suponen. Otra de las optimizaciones que se realizan es el uso de lo flags de condición de las instrucciones, para evitar saltos “branch” e instrucciones cmp antes de los saltos como se muestra en la siguiente tabla:~~

|  |  |
| --- | --- |
| ~~OPTIMIZACIÓN CON FLAGS~~ | |
| ~~Instrucciones originales~~ | ~~Instrucciones optimizadas~~ |
| ~~;esVacia~~  ~~;---------------------------~~  ~~ldrh r5,[r0]~~  ~~and r4,r5,#0x000F~~  ~~;---------------------------~~  ~~cmp r4,#0~~  ~~bne noEliminar~~  ~~;elminaCandidatos si es vacia~~  ~~;---------------------------~~  ~~ldrh r5,[r0]~~  ~~and r5,r5,#0x007F~~  ~~strh r5,[r0]~~  ~~noEliminar (…)~~ | ~~;esVacia~~  ~~;---------------------------~~  ~~ldrh r5,[r0]~~  ~~ands r4,r5,#0x000F~~  ~~;---------------------------~~    ~~;elminaCandidatos si es vacia~~  ~~;---------------------------~~  ~~ldrheq r5,[r0]~~  ~~andeq r5,r5,#0x007F~~  ~~strheq r5,[r0]~~ |

~~Además, los bucles realizados en vez de ser, por ejemplo, desde 0 hasta NUM\_FILAS se realizan desde NUM\_FILAS hasta 0, ahorrando un número de instrucciones igual a NUM\_FILAS por bucle como se muestra en la tabla a continuación:~~

|  |  |
| --- | --- |
| ~~OPTIMIZACIÓN DE BUCLES~~ | |
| ~~Bucle original~~ | ~~Bucle optimizado~~ |
| ~~mov r7,#NUM\_FILAS~~  ~~;for(j=0;j>NUM\_FILAS;j++)~~  ~~for1 (…)~~  ~~add r5,r5,#1~~  ~~cmp r5,r7~~  ~~bne for1~~ | ~~mov r7,#NUM\_FILAS~~  ~~;for(j=NUM\_FILAS;j>0;j--)~~  ~~for1 (…)~~  ~~subs r7,r7,#1~~  ~~bne for1~~ |

~~A continuación, se presenta el código nuevo desarrollado en ARM que se ha añadido al proyecto.~~

1. ~~Candidatos actualizar ARM~~

|  |  |
| --- | --- |
| ~~CANDIDATOS ACTUALIZAR ARM C~~ | |
| ~~Parámetros recibidos~~ | |
| ~~Registro~~ | ~~Descripción~~ |
| ~~r0~~ | ~~@ini\_cuadricula~~ |
| ~~Código en ARM~~ | |
| ~~candidatos\_actualizar\_arm\_c~~  ~~PUSH {r4-r10,lr} ;regs a usar y lr~~  ~~mov r9,#0 ;r9 = celdas\_vacias~~  ~~mov r7,#NUM\_FILAS ;r7 = i (AMBAS, numero de iteraciones del~~  ~~primer for y para direccionar la fila de la~~  ~~celda)~~  ~~subs r7,r7,#1 ;como el valor de NUM\_FILAS es 9,~~  ~~le resto 1 para poder direccionar bien las~~  ~~celdas~~  ~~mov r6,r0 ;como la dirección de la celda necesita estar~~  ~~en r0 para pasarlo como parámetro a esVacia y~~  ~~eliminaCandidatos~~    ~~b11 mov r8,#NUM\_FILAS ;r8 = j (AMBAS, numero de iteraciones del~~  ~~segundo for y para direccionar la columna de~~  ~~la celda)~~  ~~subs r8,r8,#1 ;como el valor de NUM\_FILAS es 9, le resto 1~~  ~~para poder direccionar bien las celdas~~    ~~b21 mov r0,r6 ;recupero el valor de la primera celda para~~  ~~calcular la que toca~~  ~~add r0,r0,r8,LSL#1~~  ~~add r0,r0,r7,LSL#5 ;calculo la dirección de la celda~~    ~~;esVacia~~  ~~;---------------------------~~  ~~ldrh r5,[r0] ;r5 = 31-16(DONT CARE) 15-0(INFORMACION DE LA~~  ~~CELDA)~~  ~~ands r4,r5,#0x000F ;r4 = valor de la celda~~  ~~;---------------------------~~  ~~;elminaCandidatos si es vacia~~  ~~;---------------------------~~  ~~andeq r5,r5,#0x007F~~  ~~strheq r5,[r0]~~  ~~;---------------------------~~  ~~subs r8,r8,#1 ;una iteración del segundo for~~  ~~bpl b21 ;acaba cuando j < 0 para calcular la columna 0~~    ~~subs r7,r7,#1 ;una iteración del primer for~~  ~~bpl b11 ;acaba cuando i < 0 para calcular la fila 0~~    ~~;SEGUNDO PAR DE BUCLES ANIDADOS~~  ~~;---------------------------~~  ~~mov r7,#NUM\_FILAS ;r7 = i (AMBAS, numero de iteraciones del~~  ~~primer for y para direccionar la fila de la~~  ~~celda)~~  ~~subs r7,r7,#1 ;como el valor de NUM\_FILAS es 9, le resto 1~~  ~~para poder direccionar bien las celdas~~  ~~mov r6,r0 ;como la dirección de la celda necesita estar~~  ~~en r0 para pasarlo como parámetro a esVacia y~~  ~~eliminaCandidatos~~  ~~b12 mov r8,#NUM\_FILAS ;r8 = j (AMBAS, numero de iteraciones del~~  ~~segundo for y para direccionar la columna de~~  ~~la celda)~~  ~~subs r8,r8,#1 ;como el valor de NUM\_FILAS es 9, le resto 1~~  ~~para poder direccionar bien las celdas~~  ~~b22 mov r0,r6 ;recupero el valor de la primera celda para~~  ~~calcular la que toca~~  ~~add r0,r0,r8,LSL#1~~  ~~add r0,r0,r7,LSL#5 ;calculo la dirección de la celda~~    ~~;esVacia~~  ~~;---------------------------~~  ~~ldrh r5,[r0] ;r5 = 31-16(DONT CARE) 15-0(INFORMACION DE LA~~  ~~CELDA)~~  ~~ands r4,r5,#0x000F ;r4 = valor de la celda~~  ~~;---------------------------~~  ~~addeq r9,r9,#1 ;si estaba vacía sumo 1 a celdas vacías~~  ~~movne r0,r6~~  ~~movne r1,r7~~  ~~movne r2,r8~~    ~~blne candidatos\_propagar\_c ;si la celda estaba vacía, propago sus~~  ~~candidatos(r0 = dirección celda, r1 = i, r8 =~~  ~~j)~~    ~~subs r8,r8,#1 ;una iteración del segundo for~~  ~~bpl b22 ;acaba cuando j < 0 para calcular la columna 0~~    ~~subs r7,r7,#1 ;una iteración del primer for~~  ~~bpl b12 ;acaba cuando i < 0 para calcular la fila 0~~  ~~mov r0,r9 ;se pasan las celdas vacías a r0 para~~  ~~devolverlas~~  ~~POP {r4-r10,pc} ;restaurar los registros usados y vuelta del pc~~  ~~a sudoku\_2021.c~~ | |

1. ~~Candidatos propagar ARM~~

|  |  |
| --- | --- |
| ~~CANDIDATOS PROPAGAR ARM~~ | |
| ~~Parámetros recibidos~~ | |
| ~~Registro~~ | ~~Descripción~~ |
| ~~r0~~ | ~~@ini\_cuadricula~~ |
| ~~r1~~ | ~~fila~~ |
| ~~r2~~ | ~~columna~~ |
| ~~Código en ARM~~ | |
| ~~candidatos\_propagar\_arm~~    ~~PUSH {r4-r12,lr} ; registros a usar~~    ~~;obtener la cuadricula[fila][columna]~~  ~~add r6,r0,r1,LSL#5 ;r6 = @ini\_fila~~  ~~add r5,r6,r2,LSL#1 ;r5 = cuadricula[fila][columna]~~    ~~ldrh r12,[r5] ;r12 = cuadricula[fila][columna] =~~  ~~mem[@ini\_fila + col\*2] (r6+r5 =~~  ~~@cuadricula[fila][columna])~~  ~~;valor = celda\_leer\_valor~~  ~~and r12,#0x00F ;r12 = valor~~  ~~;---------------------------~~  ~~mov r7,#NUM\_FILAS ;r7 = j = 9~~  ~~for1 ;for(j=NUM\_FILAS;j>0;j--)~~  ~~;celda\_eliminar\_candidato~~  ~~mov r11,#0x40 ;0000 0000 0100 0000~~  ~~mov r11,r11,LSL r12 ;muevo el 1 tantas posiciones como valor y así~~  ~~obtengo la máscara para eliminar el candidato~~  ~~ldrh r4,[r6] ;r4 = mem[r6]~~  ~~orr r4,r4,r11 ;r4 = or bit a bit para poner a 1 el bit de r4~~  ~~que está a 1 en r11 y no tocar el resto~~  ~~strh r4,[r6] ;mem[r6] = r4~~  ~~;---------------------------~~  ~~add r6,r6,#2 ;avanza a la columna siguiente~~  ~~subs r7,r7,#1 ;r7 = j-- (condición de fin de bucle j = 0)~~  ~~bne for1~~  ~~;---------------------------~~  ~~mov r7,#NUM\_FILAS ;r7 = i = 9~~  ~~sub r5,r5,r1,LSL#5 ;r5 = @ini\_col~~  ~~for2 ;for(i=NUM\_FILAS;i>0;i--)~~  ~~;celda\_eliminar\_candidato~~  ~~mov r11,#0x40 ;0000 0000 0100 0000~~  ~~mov r11,r11,LSL r12 ;muevo el 1 tantas posiciones como valor y así~~  ~~obtengo la máscara para eliminar el candidato~~  ~~ldrh r4,[r5] ;r4 = mem[r5]~~  ~~orr r4,r4,r11 ;r4 = or bit a bit para poner a 1 el bit de r4~~  ~~que está a 1 en r11 y no tocar el resto~~  ~~strh r4,[r5] ;mem[r5] = r4~~  ~~;---------------------------~~  ~~add r5,r5,#32 ;r5 = cuadricula[fila + 1][columna]~~  ~~subs r7,r7,#1 ;r7 = i-- (condición de fin de bucle i = 0)~~  ~~bne for2~~  ~~;---------------------------~~    ~~;Encontrar el inicio de la región~~  ~~;init\_i = init\_region[fila];~~  ~~;init\_j = init\_region[columna];~~  ~~;end\_i = init\_i + 3;~~  ~~;end\_j = init\_j + 3;~~    ~~LDR r6,=init\_region~~  ~~add r7,r6,r1~~  ~~add r8,r6,r2~~  ~~ldrb r7,[r7] ;r7 = init\_i (fila)~~  ~~ldrb r8,[r8] ;r8 = init\_j (col)~~  ~~add r9,r7,#3 ;r9 = end\_i~~  ~~add r10,r8,#3 ;r10 = end\_j~~  ~~for3 ;for(i=init\_i; i<end\_i; i++) ;avanza fila a fila~~  ~~mov r6,r8 ;se restaura en r6 el valor de la columna para~~  ~~repetir for4~~  ~~;Calcula @cuadricula[ini\_i][ini\_j]~~  ~~add r5,r0,r7,LSL#5~~  ~~add r5,r5,r8,LSL#1~~  ~~for4 ;for(j=init\_j; j<end\_j; j++) ;avanza columna a columna~~  ~~;celda\_eliminar\_candidato~~  ~~mov r11,#0x40 ;0000 0000 0100 0000~~  ~~mov r11,r11,LSL r12 ;muevo el 1 tantas posiciones como valor y así~~  ~~obtengo la máscara para eliminar el candidato~~  ~~ldrh r4,[r5] ;r4 = mem[r5] = valor cuadricula~~  ~~orr r4,r4,r11 ;r4 = or bit a bit para poner a 1 el bit de r4~~  ~~que está a 1 en r11 y no tocar el resto~~  ~~strh r4,[r5] ;mem[r5] = r4~~  ~~;---------------------------~~  ~~add r5,r5,#2 ;avanza r6 en la región~~  ~~add r6,r6,#1 ;avanza en la columna~~  ~~cmp r6,r10 ;se compara r6 (init\_j que va avanzando por~~  ~~las columnas) con r10 end\_j~~  ~~bne for4~~  ~~;---------------------------~~  ~~add r7,r7,#1 ;avanzar la fila en para for3~~  ~~cmp r7,r9 ;se compara r7 (init\_i que va avanzando por~~  ~~las filas) con end\_i~~  ~~bne for3~~  ~~;---------------------------~~  ~~POP {r4-r12,pc} ;restaurar los registros usados y vuelta del~~  ~~pc a sudoku\_2021.c~~ | |

1. ~~Candidatos actualizar ARM~~

|  |  |
| --- | --- |
| ~~CANDIDATOS ACTUALIZAR ARM~~ | |
| ~~Parámetros recibidos~~ | |
| ~~Registro~~ | ~~Descripción~~ |
| ~~r0~~ | ~~@ini\_cuadricula~~ |
| ~~Código en ARM~~ | |
| ~~candidatos\_actualizar\_arm\_c~~  ~~PUSH {r4-r12,lr} ;regs a usar y lr~~  ~~mov r9,#0 ;r9 = celdas\_vacias~~  ~~mov r7,#NUM\_FILAS ;r7 = i (AMBAS, numero de iteraciones del~~  ~~primer for y para direccionar la fila de la~~  ~~celda)~~  ~~subs r7,r7,#1 ;como el valor de NUM\_FILAS es 9,~~  ~~le resto 1 para poder direccionar bien las~~  ~~celdas~~  ~~mov r6,r0 ;como la direccion de la celda necesita estar~~  ~~en r0 para pasarlo como parametro a esVacia y~~  ~~eliminaCandidatos~~    ~~arm\_arm\_b11 mov r8,#NUM\_FILAS ;r8 = j (AMBAS, numero de iteraciones del~~  ~~segundo for y para direccionar la columna de~~  ~~la celda)~~  ~~subs r8,r8,#1 ;como el valor de NUM\_FILAS es 9, le resto 1~~  ~~para poder direccionar bien las celdas~~    ~~arm\_arm\_b21 mov r0,r6 ;recupero el valor de la primera celda para~~  ~~calcular la que toca~~  ~~add r0,r0,r8,LSL#1~~  ~~add r0,r0,r7,LSL#5 ;calculo la direccion de la celda~~    ~~;esVacia~~  ~~ldrh r5,[r0] ;r5 = 31-16(DONT CARE) 15-0(INFORMACION DE LA~~  ~~CELDA)~~  ~~ands r4,r5,#0x000F ;r4 = valor de la celda~~  ~~;---------------------------~~  ~~;elminaCandidatos si es vacia~~  ~~andeq r5,r5,#0x007F~~  ~~strheq r5,[r0]~~  ~~;---------------------------~~  ~~subs r8,r8,#1 ;una iteracion del segundo for~~  ~~bpl arm\_arm\_b21 ;acaba cuando j < 0 para calcular la columna 0~~    ~~subs r7,r7,#1 ;una iteracion del primer for~~  ~~bpl arm\_arm\_b11 ;acaba cuando i < 0 para calcular la fila 0~~    ~~;SEGUNDO PAR DE BUCLES ANIDADOS~~  ~~mov r7,#NUM\_FILAS ;r7 = i (AMBAS, numero de iteraciones del~~  ~~primer for y para direccionar la fila de la~~  ~~celda)~~  ~~subs r7,r7,#1 ;como el valor de NUM\_FILAS es 9, le resto 1~~  ~~para poder direccionar bien las celdas~~  ~~mov r6,r0 ;como la direccion de la celda necesita estar~~  ~~en r0 para pasarlo como parametro a esVacia y~~  ~~eliminaCandidatos~~  ~~arm\_arm\_b12 mov r8,#NUM\_FILAS ;r8 = j (AMBAS, numero de iteraciones del~~  ~~segundo for y para direccionar la columna de~~  ~~la celda)~~  ~~subs r8,r8,#1 ;como el valor de NUM\_FILAS es 9, le resto 1~~  ~~para poder direccionar bien las celdas~~  ~~arm\_arm\_b22 mov r0,r6 ;recupero el valor de la primera celda para~~  ~~calcular la que toca~~  ~~add r0,r0,r8,LSL#1~~  ~~add r0,r0,r7,LSL#5 ;calculo la direccion de la celda~~    ~~;esVacia~~  ~~ldrh r5,[r0] ;r5 = 31-16(DONT CARE) 15-0(INFORMACION DE LA~~  ~~CELDA)~~  ~~ands r4,r5,#0x000F ;r4 = valor de la celda~~  ~~;---------------------------~~  ~~addeq r9,r9,#1 ;si estaba vacia sumo 1 a celdas vacias~~  ~~movne r0,r6~~  ~~beq noPropagar~~  ~~mov r1,r7~~  ~~mov r2,r8~~    ~~push {r6-r9}~~    ~~;candidatos\_propagar\_arm~~  ~~;=====================================================================================~~    ~~;obtener la cuadricula[fila][columna]~~  ~~add r6,r0,r1,LSL#5 ;r6 = @ini\_fila~~  ~~add r5,r6,r2,LSL#1 ;r5 = cuadricula[fila][columna]~~    ~~ldrh r12,[r5] ;r12 = cuadricula[fila][columna] =~~  ~~mem[@ini\_fila + col\*2] (r6+r5 =~~  ~~@cuadricula[fila][columna])~~    ~~;valor = celda\_leer\_valor~~  ~~and r12,#0x00F ;r12 = valor~~  ~~;---------------------------~~    ~~mov r7,#NUM\_FILAS ;r7 = j = 9~~  ~~arm\_arm\_for1 ;for(j=NUM\_FILAS;j>0;j--)~~  ~~;celda\_eliminar\_candidato~~  ~~mov r11,#0x40 ;0000 0000 0100 0000~~  ~~mov r11,r11,LSL r12 ;muevo el 1 tantas posiciones como valor y así~~  ~~obtengo la máscara para eliminar el candidato~~  ~~ldrh r4,[r6] ;r4 = mem[r6]~~  ~~orr r4,r4,r11 ;r4 = or bit a bit para poner a 1 el bit de r4~~  ~~que está a 1 en r11 y no tocar el resto~~  ~~strh r4,[r6] ;mem[r6] = r4~~  ~~;---------------------------~~  ~~add r6,r6,#2 ;avanza a la columna siguiente~~  ~~subs r7,r7,#1 ;r7 = j-- (concición de fin de bucle j = 0)~~  ~~bne arm\_arm\_for1~~  ~~;---------------------------~~  ~~mov r7,#NUM\_FILAS ;r7 = i = 9~~  ~~sub r5,r5,r1,LSL#5 ;r5 = @ini\_col~~  ~~arm\_arm\_for2 ;for(i=NUM\_FILAS;i>0;i--)~~  ~~;celda\_eliminar\_candidato~~  ~~mov r11,#0x40 ;0000 0000 0100 0000~~  ~~mov r11,r11,LSL r12 ;muevo el 1 tantas posiciones como valor y así~~  ~~obtengo la máscara para eliminar el candidato~~  ~~ldrh r4,[r5] ;r4 = mem[r5]~~  ~~orr r4,r4,r11 ;r4 = or bit a bit para poner a 1 el bit de r4~~  ~~que está a 1 en r11 y no tocar el resto~~  ~~strh r4,[r5] ;mem[r5] = r4~~  ~~;---------------------------~~  ~~add r5,r5,#32 ;r5 = cuadricula[fila + 1][columna]~~  ~~subs r7,r7,#1 ;r7 = i-- (concición de fin de bucle i = 0)~~  ~~bne arm\_arm\_for2~~  ~~;---------------------------~~    ~~;Encontrar el inicio de la región~~  ~~;init\_i = init\_region[fila];~~  ~~;init\_j = init\_region[columna];~~  ~~;end\_i = init\_i + 3;~~  ~~;end\_j = init\_j + 3;~~    ~~LDR r6,=init\_region~~  ~~add r7,r6,r1~~  ~~add r8,r6,r2~~  ~~ldrb r7,[r7] ;r7 = init\_i (fila)~~  ~~ldrb r8,[r8] ;r8 = init\_j (col)~~  ~~add r9,r7,#3 ;r9 = end\_i~~  ~~add r10,r8,#3 ;r10 = end\_j~~  ~~arm\_arm\_for3 ;for(i=init\_i; i<end\_i; i++) ;avanza fila a fila~~  ~~mov r6,r8 ;se restaura en r6 el valor de la columna para~~  ~~repetir for4~~  ~~;Calcula @cuadricula[init\_i][init\_j]~~  ~~add r5,r0,r7,LSL#5~~  ~~add r5,r5,r8,LSL#1~~    ~~arm\_arm\_for4 ;for(j=init\_j; j<end\_j; j++) ;avanza columna a columna~~  ~~;celda\_eliminar\_candidato~~  ~~mov r11,#0x40 ;0000 0000 0100 0000~~  ~~mov r11,r11,LSL r12 ;muevo el 1 tantas posiciones como valor y así~~  ~~obtengo la máscara para eliminar el candidato~~  ~~ldrh r4,[r5] ;r4 = mem[r5] = valor cuadricula~~  ~~orr r4,r4,r11 ;r4 = or bit a bit para poner a 1 el bit de r4~~  ~~que está a 1 en r11 y no tocar el resto~~  ~~strh r4,[r5] ;mem[r5] = r4~~  ~~;---------------------------~~  ~~add r5,r5,#2 ;avanza r6 en la región~~  ~~add r6,r6,#1 ;avanza en la columna~~  ~~cmp r6,r10 ;se compara r6 (init\_j que va avanzando por~~  ~~las columnas) con r10 end\_j~~  ~~bne arm\_arm\_for4~~  ~~;---------------------------~~  ~~add r7,r7,#1 ;avanzar la fila en para for3~~  ~~cmp r7,r9 ;se compara r7 (init\_i que va avanzando por~~  ~~las filas) con end\_i~~  ~~bne arm\_arm\_for3~~  ~~;---------------------------~~  ~~pop{r6-r9}~~  ~~;==========================================================================~~    ~~noPropagar subs r8,r8,#1 ;una iteracion del segundo for~~  ~~bpl arm\_arm\_b22 ;acaba cuando j < 0 para calcular la columna 0~~    ~~subs r7,r7,#1 ;una iteracion del primer for~~  ~~bpl arm\_arm\_b12 ;acaba cuando i < 0 para calcular la fila 0~~  ~~mov r0,r9 ;se pasan las celdas vacías a r0 para~~  ~~devolverlas~~  ~~POP {r4-r12,pc} ;restaurar los registros usados y vuelta del pc~~  ~~a sudoku\_2021.c~~ | |

# ~~RESULTADOS~~

A continuación, se muestran y analizan los resultados obtenidos para los distintos niveles de compilación. Notar que la versión *“candidatos\_actualizar\_arm”* va a arrojar en todos los casos los mismos resultados debido a que está escrita directamente en ensamblador por lo que el nivel de compilación no le afecta.

## Resultados con la opción de compilación -O0

Opción de compilación por defecto. En esta opción no se aplica ninguna optimización, el código en ensamblador hace una traducción casi literal del código en C. En los resultados obtenidos la función “candidatos\_actualizar\_arm” no va a variar según ninguna opción de compilación debido a que toda ella está escrita en ensamblador.

### Resultados obtenidos

|  |  |  |
| --- | --- | --- |
| RESULTADOS OBTENIDOS -O0 | | |
| Función | Coste en tiempo | Coste en memoria |
| candidatos\_actualizar\_c | 2,61775 ms | 200 B |
| candidatos\_actualizar\_arm | 1,28966 ms | 336 B |
| candidatos\_actualizar\_c\_arm | 1,60025 ms | 200 B |
| candidatos\_actualizar\_arm\_c | 2,40842 ms | 156 B |

### Análisis de rendimiento

En este caso tenemos el mayor coste en tiempo de todos los niveles de compilación. Al tratarse de una traducción casi literal del código en C, no está activo el *inlining*, conservando todas las llamadas a las funciones de C del código original, como todas las llamadas a las funciones de “*celda.h*”, penalizando en tiempo y en número de instrucciones ejecutadas. Así se observa como en la versión *"candidatos\_actualizar\_arm",* que ha sido escrita sin esas llamadas a función o el uso de *flags* de condición en instrucciones (para reducir el número de instrucciones), se obtiene un tiempo de ejecución de 1,2 ms frente a 2,6 ms de la versión en C, ósea, un *SpeedUp* del **50,73% más rápida nuestra versión en ensamblador frente a la versión en C.**

Asimismo, entre las versiones *"candidatos\_actualizar\_c\_arm"* y *"candidatos\_actualizar\_arm\_c"* que sólo se diferencian en que una tiene en C la parte de *"candidaros\_propagar"* y la parte de *“candidatos\_actualizar”* en ensamblador y viceversa, se observa como entre las dos funciones, *"candidaros\_propagar"* es la que tiene mayor peso (en instrucciones y tiempo) por lo que es una función critica que si se optimiza se ganará sustancialmente en tiempo. En este caso el *speedUp* entre las versiones es de un **33,5% más rápida la versión de "candidaros\_propagar" en ensamblador**, en gran medida gracias al *inlining* que hace la versión en ensamblador que se ahorra todas las llamadas a funciones de *“celda.h”.*

## Resultados con la opción de compilación -O1

En este caso realiza un conjunto mínimo de optimizaciones en el código generado respecto al nivel-O0. Esta opción de compilación a diferencia de las demás (-O2 y -O3) trata de reducir el tamaño del código generado, pero no reduce apenas el tiempo de ejecución. Se trata de una opción de compilación intermedia entre -O0 y -O2.

### Resultados obtenidos

|  |  |  |
| --- | --- | --- |
| RESULTADOS OBTENIDOS -O1 | | |
| Función | Coste en tiempo | Coste en memoria |
| candidatos\_actualizar\_c | 2,25767 ms | 208 B |
| candidatos\_actualizar\_arm | 1,28967 ms | 336 B |
| candidatos\_actualizar\_c\_arm | 1,53766 ms | 212 B |
| candidatos\_actualizar\_arm\_c | 2,11192 ms | 156 B |

### Análisis de rendimiento

En los resultados obtenidos se observa una pequeña reducción en tiempo con respecto a -O0, manteniendo el tamaño del código más o menos igual respecto al nivel -O0. En este caso la reducción en tiempo del código al código en C de "candidatos\_actualizar\_c" con respecto a la versión en ensamblador de "candidatos\_actualizar\_arm" ofrece un speedUp de un **42,88% más rápida la versión en ensamblador.**

## Resultados con la opción de compilación -O2

Realiza todas las opciones de compilación posibles sin que supongan un gran impacto en el tamaño. Como resultado el tiempo de compilación es mayor y la velocidad de ejecución es también mayor si lo comparamos con la opción -O0 Incluye todos los flags de optimización de -O1 y algunos más.

### Resultados obtenidos

|  |  |  |
| --- | --- | --- |
| RESULTADOS OBTENIDOS -O2 | | |
| Función | Coste en tiempo | Coste en memoria (idem -O1) |
| candidatos\_actualizar\_c | 2,25517 ms | 208 B |
| candidatos\_actualizar\_arm | 1,28966 ms | 336 B |
| candidatos\_actualizar\_c\_arm | 1,46775 ms | 212 B |
| candidatos\_actualizar\_arm\_c | 2,17942 ms | 156 B |

### Análisis de rendimiento

Sobre el nivel de optimización O2, no difiere mucho del O1, pero como si intenta hacer que mejore el rendimiento del codigo se nota una mejoria en los tiempos de ejecucion y en el tamaño del codigo en Bytes. Así el SpeedUp con esta opción de compilación, de **“candidatos\_actualizar\_arm” sobre “candidatos\_actualizar\_c” es un 42,82% más rápida** que **no representa una gran mejoría respecto al nivel O1** (42,88%).

## Resultados con la opción de compilación -O3

Esta opción da la mayor velocidad posible al código en C compilado. Realiza todas las optimizaciones posibles, pero, a cambio, el tamaño del código aumentará, así como el tiempo de compilación.

### Resultados obtenidos

|  |  |  |
| --- | --- | --- |
| RESULTADOS OBTENIDOS -O3 | | |
| Función | Coste en tiempo | Coste en memoria |
| candidatos\_actualizar\_c | 0,9315 ms | 428 B |
| candidatos\_actualizar\_arm | 1,28950 ms | 336 B |
| candidatos\_actualizar\_c\_arm | 1,32242 ms | 160 B |
| candidatos\_actualizar\_arm\_c | 1,29558 ms | 156 B |

### Análisis de rendimiento

Sobre el nivel de optimización O3, aquí hay un gran cambio en el rendimiento visible en los tiempos de ejecución los cuales se reducen drásticamente, superando por aproximadamente 2 ms nuestra solución, pero a cambio usa mucho mas código.

En este caso la versión en C **“candidatos\_actualizar\_c” tiene un SpeedUp del 27,76% por lo que es rápida con respecto a “candidatos\_actualizar\_arm”,** nuestra versión en ensamblador. Este **es el único caso en que el compilador ofrece una mayor velocidad** de ejecución con respecto a la versión en ensamblador. Sin embargo, a pesar de que en este caso el compilador en tiempo mejora la versión en ensamblador, el coste en memoria se dispara, **siendo la única versión en la que la función en c “candidatos\_actualizar\_c” ocupa más que “candidatos\_actualizar\_arm”, en concreto , un 127,38% más.**

# CONCLUSIONES

~~Para terminar, nuestro trabajo ha servido para demostrar y entender que, si queremos programar y que sea óptimo en tiempo, la mejor opción es confiar en el compilador. En este caso no merece la pena el esfuerzo en tiempo (ni es económicamente rentable), por parte del programador de ponerle a programar este programa en ensamblador si tiene a su disposición un compilador de C.~~

~~Para los casos en los que el código en ensamblador resultaba más óptimo en tiempo (-O0, -O1 y -O2), la diferencia es de 1ms, lo que para este caso no merece la pena. Si pensamos en como se jugará el sudoku el tablero estará casi siempre esperando la entrada de un usuario, que decida que valor poner en la celda, por lo que ese 1ms que ganamos en actualizar los candidatos tras recibir la entrada del usuario resulta despreciable y no creemos que merezca la pena el esfuerzo de programar en ensamblador para ganar tan poco.~~