UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO:

CONTROL MEDIANTE IDE ECLIPSE, DE UNA PLACA DE ENTRADAS/SALIDAS MULTIPLES CONECTADA AL MICROCONTROLADOR STM32L-DISCOVERY

AUTOR: NORBERTO J. RIVILLO MATÍA

TUTOR: DR. JUAN VÁZQUEZ MARTÍNEZ

Septiembre 2012

AGRADECIMIENTOS:

Quiero gradecer este proyecto en primer lugar a mi familia, por haberme dado la posibilidad de estudiar y haberme apoyado cada vez que lo necesitaba, y en cada decisión que he tomado, aunque no siempre estuviesen de acuerdo.

A Paloma, por aguantarme después de tanto tiempo y saber sacarme siempre una sonrisa.

Y, por supuesto, a Juan, mi tutor en este proyecto, por que gracias a su asignatura de cuarto curso me empecé a interesar por los microprocesadores, y por el apoyo y los consejos que me ha dado en todo momento.

ÍNDICE:

LISTA DE FIGURAS:

1-INTRODUCCIÓN Y OBJETIVOS:

El presente proyecto tiene como objetivo básico la mejora del sistema utilizado actualmente para realizar las prácticas de laboratorio en varias asignaturas de Mircroprocesadores, impartidas por el Departamento de Tecnología Electrónica.

Dichas prácticas utilizan placas de desarrollo comerciales, conectadas a otra de desarrollo propio de la UC3M, denominada placa de Entrada/Salida o E/S, que proporciona diversos periféricos adaptados a las necesidades de enseñanza propias de cada asignatura y microprocesador, en concreto los modelos basados en arquitecturas tipo 8x51 de 8 bits y ARM7 de 32 bits.

Durante el pasado curso académico 2011-2012 se han actualizado los contenidos docentes teórico-prácticos, introduciendo en varias asignaturas el nuevo microcontrolador ARM Cortex-M3, de 32 bits al igual que el ARM7.

Las nuevas prácticas de laboratorio utilizan la placa comercial denominada STM32L-Discovery y una placa auxiliar propia de nuevo diseño, pero con prestaciones limitadas. Por ello, se plantea como una ampliación conveniente la reutilización de la placa de E/S original con el nuevo modelo de microprocesador, lo que requiere una serie de estudios y actuaciones:

-Evaluación de la posibilidad de migrar a un nuevo entorno de desarrollo o IDE, basado en software libre, en sustitución del actual IDE de Atollic.

-Diseño de la interconexión física entre la placa de E/S y el STM32L-Discovery.

-Diseño y prueba de las funciones escritas en lenguaje C, que permitan controlar los periféricos de la placa de E/S desde el Cortex-M3.

Por tanto, el objetivo inicial se desglosa en los escritos en los tres puntos previos.

La placa STM332L-Discovery puede ser utilizada desde diversos entornos de desarrollo, entre los que destacan son los proporcionados por Keil, IAR y ATOLLIC, siendo este último el adoptado para uso docente.

El IDE proporcionado por Atollic está basado en la herramienta libre Eclipse y sus prestaciones son suficientes para las necesidades de un alumno, aunque con ciertas restricciones. Por ello, se ha planteado adoptar un nuevo IDE que carezca de limitaciones temporales, asi como de tamaño de los ficheros ejecutables que presenta el IDE TrueSTUDIO de Atollic. Se ha planteado Eclipse por ser el IDE de software libre más utilizado, por ser la base el actual IDE de Atollic, y también por ser familiar para los estudiantes que han estudiado Java previamente.

2-RECURSOS DISPONIBLES DE HW Y SW:

2.1-IDE ATOLLIC TRUESTUDIO

2.1.1-CARACTERÍSTICAS:

El entorno de desarrollo TrueSTUDIO de Atollic es una aplicación que permite al usuario desarrollar programas en C y C++ para microprocesadores ARM7, ARM9 y ARM Cortex, cumpliendo tanto las funciones de compilación como de depuración sobre una placa de desarrollo.

Atollic TrueSTUDIO está basado en el entorno de desarrollo de software libre Eclipse, y aprovecha el toolchain de GNU, además de compiladores también del proyecto GNU como GCC. Además, incluye un gran número de librerías adaptadas para diversas placas de desarrollo y permite crear plantillas de proyecto para cada una de ellas.

La interfaz del programa es como la que vemos en la figura siguiente:

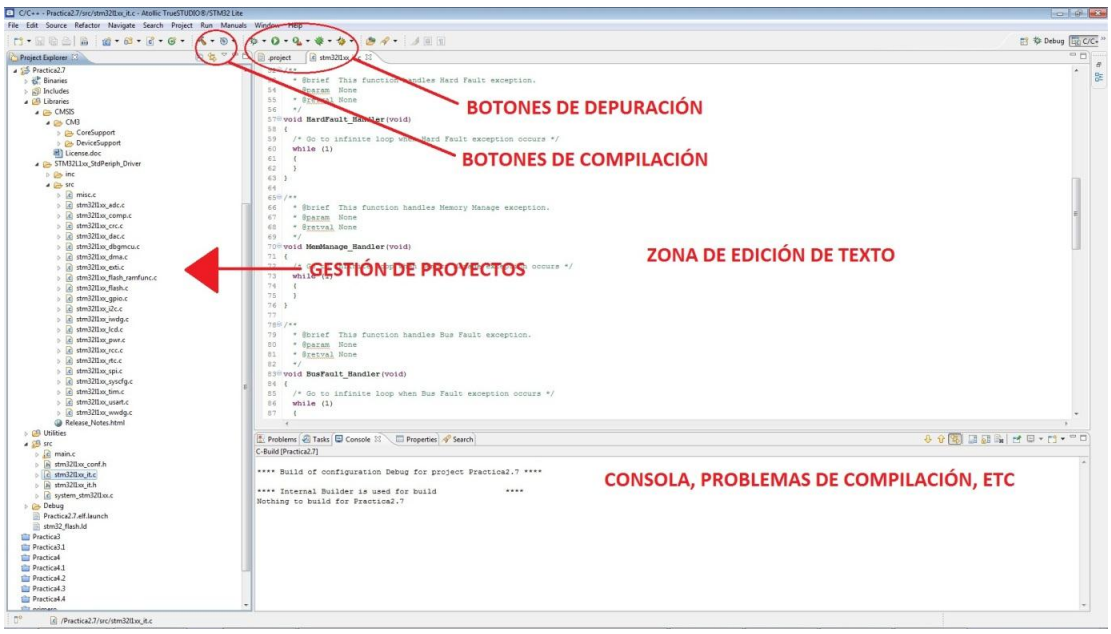


Figura 1

Como se puede ver, la interfaz es muy limpia, cuenta con botones de compilación y depuración, una zona de gestión de los contenidos del proyecto, una zona de edición de código y otra donde obtenemos información de problemas, búsquedas o las salidas de la consola.

Los elementos de la pantalla son totalmente personalizables y se pueden distribuir según los gustos o necesidades del usuario.

La interfaz en el modo de depuración también es limpia y configurable, y permite controlar todo lo que el usuario desee en cada momento: variables, breakpoints, etc; como se puede observar en la figura 2.2:

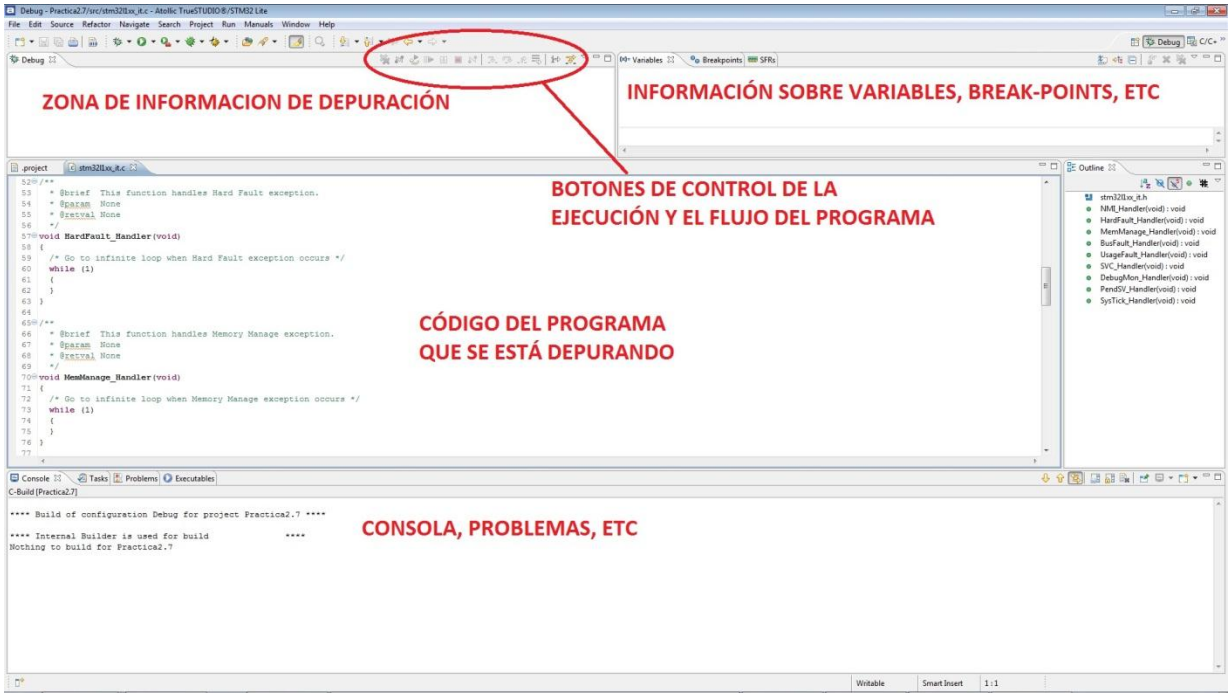


Figura 2

2.1.2-VENTAJAS E INCONVENIENTES

Atollic TrueSTUDIO tiene un gran número de puntos a favor:

-Al tratarse de un software profesional, el funcionamiento es muy estable, sin apenas problemas ni cuelgues.

-La interfaz es muy intuitiva, prácticamente calcada a la del Eclipse original.

-Tiene una buena gestión de proyectos, y permite la creación automatizada de plantillas.

-Es tremendamente versátil: permite desarrollar y depurar proyectos para un gran número de microcontroladores.

Sin embargo, existe un problema, y es que la versión utilizada en la Universidad es la edición gratuita del producto, por lo que tiene algunas limitaciones:

-No permite crear proyectos en C++.

-Número de Break-points limitado.

-Incluye anuncios que saltan cada determinado tiempo.

-Limita el tamaño máximo del archivo de código.

-Editor de textos menos detallado.

-Solicita registro en la página web.

2.2-PLACA DE E/S:

2.2.1-CARACTERÍSTICAS:

La placa que se desea controlar con el STM32L-discovery ha sido diseñada en el departamento de Tecnología Electrónica de la UC3M, con la función de facilitar la realización de prácticas en asignaturas relacionadas con microcontroladores, proporcionando un sistema físico perfectamente operativo con el que el alumno debe interactuar.

La placa de E/S está dotada de los siguientes periféricos:

-DISPLAY LCD:

Se trata de un display de la marca HITACHI, en concreto el modelo LM016L. Consta de dos filas en las que escribir, con hasta 16 caracteres cada una, cada uno de los cuales están compuestos por una matriz de 5x7 puntos. Además, el LCD cuenta con una pequeña memoria interna que le permite almacenar hasta 248 caracteres, 240 de los cuales se encuentran pre-cargados, y se corresponden con aquellos más comunes dentro de la tabla ASCII.

Figura 3

El display cuenta con su propio controlador, un HD44780, con el que se debe comunicar el STM32L-Discovery para controlar el display LCD. La interfaz de comunicación entre los controladores consiste en un bus de datos de 8 bits (que sirve para enviar comandos al LCD y para leer la señal de “Busy”) y tres señales de control: Register Select, Read/Write y Enable.

Para conocer los comandos que se deben usar para controlar el display LCD se debe consultar el manual del mismo, un extracto del cual se puede encontrar en los anexos del proyecto.

-RELÉ:

Este dispositivo electromecánico se cierra cuando su tensión de entrada supera cierto valor, esto sucederá cuando se active el pin de la placa de desarrollo conectado a su entrada.

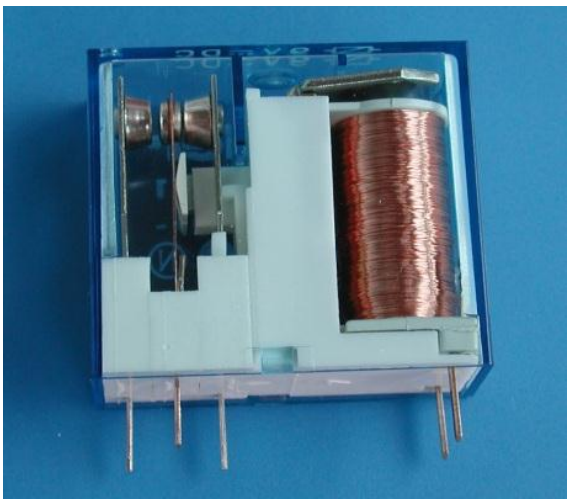


Figura 4

-MOTOR DE CORRIENTE CONTINUA CON SENSOR ÓPTICO:

Otro de los periféricos de la placa de E/S es un motor convencional de corriente continua, cuya velocidad se puede controlar mediante una señal con modulación PWM.

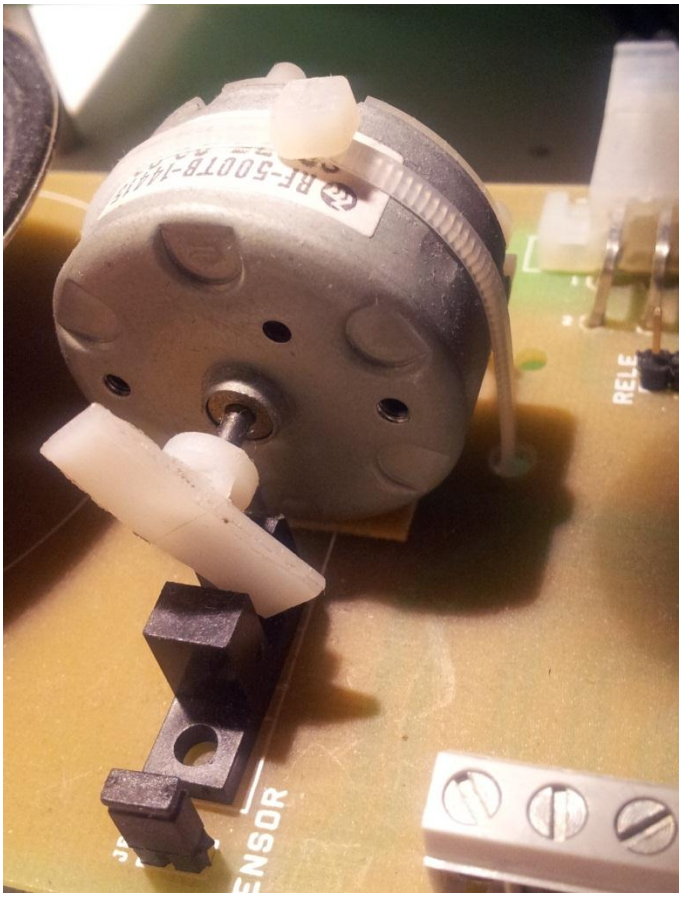


Figura 5

Si se administra una tensión fija al motor, éste girará a su velocidad nominal, y se detendrá con un ligero retardo debido a la inercia al desaparecer esta tensión. Esto significa que a priori no se puede controlar la velocidad de este motor, pero uno de los métodos más sencillos que existen para obtener esta funcionalidad es utilizar una señal con modulación PWM en el control del motor:

La modulación PWM consiste en generar una señal formada por pulsos de diferentes anchuras de forma que, en cada ciclo o periodo de la señal, el tiempo que la misma ha estado a nivel alto es equivalente al valor del ciclo de trabajo de la función expresado en tanto por ciento.

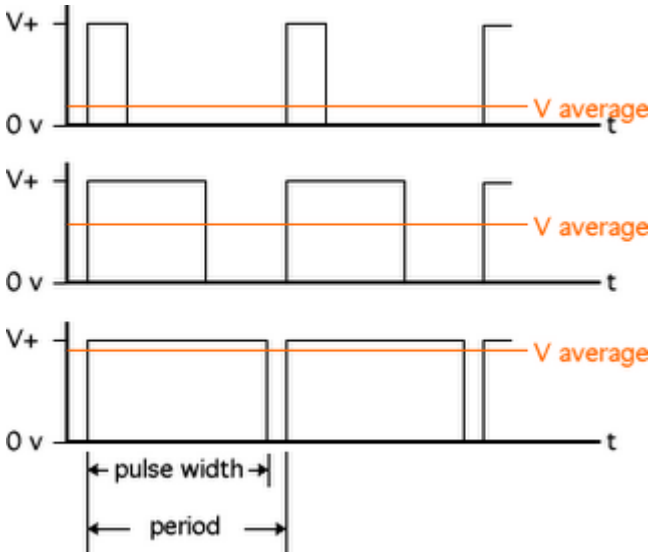


Figura 6

Si se configura la modulación PWM adecuadamente, se puede regular la velocidad del motor variando el ciclo de trabajo, lo que modifica el tiempo que la señal se encuentra a nivel alto, controlando la velocidad neta que alcanza el motor.

-MATRIZ DE LED´S:

Las matrices de LEDs son un sistema alternativo a los LCD para proporcionar información al usuario, en este caso, la placa de E/S cuenta con una matriz de 7x5 LEDs.

Para manejar la matriz solo se han habilitado 4 líneas de comunicación, lo cual significa que, al no tener una señal por cada fila y columna, el control de la matriz resulta ligeramente más complicado de lo que se podría esperar, siendo necesario servirse de dos registros de desplazamiento de 8 bits para almacenar los valores correspondientes a las columnas y filas que se desea activar.

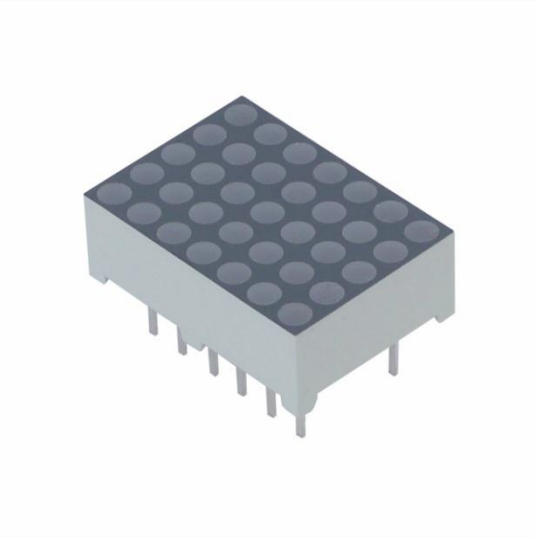


Figura 7

Los registros funcionan como una cola FIFO (First In, First Out), y cada uno de ellos posee una señal de reloj, con la cual se consigue que los bits almacenados avancen una posición dentro del registro de 8 bits, entrando un dato nuevo en el último registro.

Existe también una seña de habilitación, la cual da salida en forma de puerto serie de 8 líneas a los 8 bits almacenados en cada registro.

Por ultimo existe una señal denominada Dato, por la cual se enviará información de forma secuencial a los registros.

De este modo, para poder encender un LED de la matriz, se debe poner a nivel alto los registros correspondientes a su fila y su columna, y dejar el resto a cero. Para ello se debe colocar en la señal de Dato el 1 o el 0 que se desee grabar, dar un pulso a la señal de reloj del registro correspondiente, y repetir esta secuencia 16 veces para cada uno de los bits de los dos registros de desplazamiento.

Como se puede observar, es imposible activar simultáneamente dos LEDs de distintas filas y columnas, por lo que en caso de que sea necesario se debe recurrir a encenderlos de manera alternativa con una frecuencia de

refresco suficiente como para que el ojo humano los perciba a ambos activados (y lo mismo se puede aplicar a un numero superior de LEDs).

-ALTAVOZ:

La placa de E/S también cuenta con un altavoz, que se puede utilizar para emitir señales audibles o reproducir una serie de tonos a modo de melodía.

Figura 8

El altavoz solo cuenta con una señal de entrada, que corresponde a la señal que se envía para que éste reproduzca un sonido; para generar diferentes tonos, solo es necesario modificar la frecuencia y el periodo de esta señal.

-SENSOR DE INFRARROJOS:

La placa de E/S cuenta también con un sensor de infrarrojos, que permite recibir señales desde el exterior por medio de una transmisión serie mediante este sistema.

El sensor cuenta con una lente, un receptor infrarrojo y la electrónica que le permite reconocer las señales moduladas en el espectro infrarrojo.

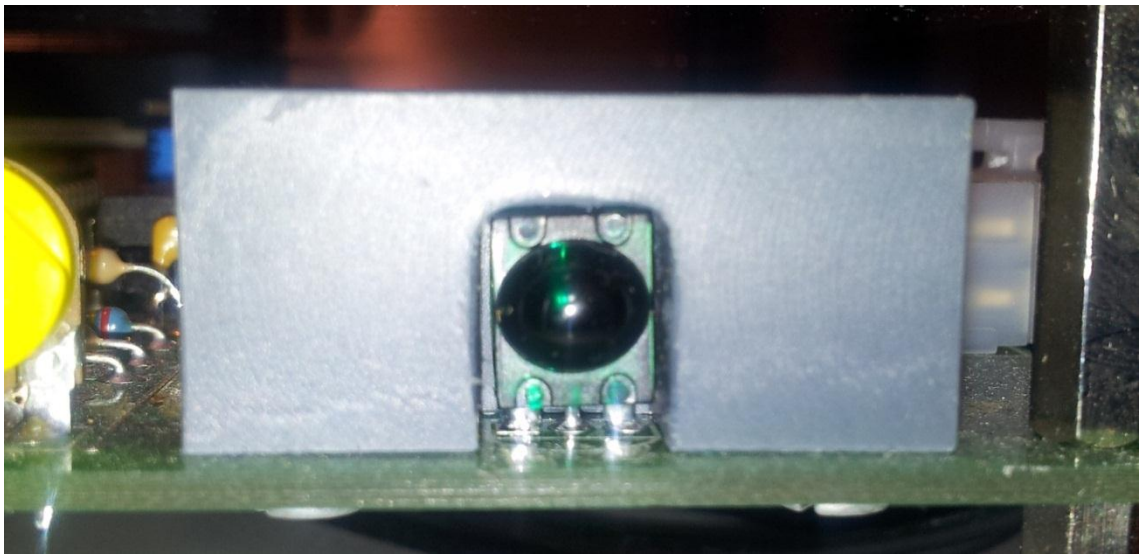


Figura 9

Este periférico resulta especialmente interesante porque se puede llegar a reconocer una amplia gama de señales distintas, que permitirían provocar diferentes reacciones por parte del microcontrolador una vez descifradas. Sin embargo esto supone un problema, y es que cada mando infrarrojo del mercado posee una codificación diferente para cada uno de las señales emitidas al pulsar un botón, por lo que no se puede desarrollar una biblioteca estándar de correspondencia de estas señales. La solución que se propone aplicar en este caso es desarrollar un método para leer una señal y asignarla a una determinada función en cada caso. Sin embargo, esa aplicación no ha sido abordada en este proyecto.

-DISPOSITIVOS I2C: MEMORIA EEPROM:

El protocolo I2C es uno de los sistemas de comunicación serie síncrona más sencillos de implementar, y en el caso de la placa de E/S se ha diseñado su uso para acceder a una memoria EEPROM de 16 kbits integrada en la misma.

El protocolo I2C es una interfaz de dos hilos, uno de datos (bidireccional) y otro que se encarga de transmitir la señal de reloj o habilitación. La comunicación tiene lugar mediante niveles TTL, y permite modo semi-duplex, es decir, comunicación bidireccional pero no simultánea.

Esto permite al usuario de la placa de E/S almacenar datos en la memoria EEPROM y leerlos cuando lo desee. De esta forma, se pueden almacenar mensajes predeterminados para presentarlos por el display LCD en ciertos momentos de la ejecución de un programa.

Para comunicarse con un dispositivo esclavo, el maestro debe enviar mensajes prefijados de START y STOP, que marcan el principio y el final de una trama de datos o comandos, y recibe a su vez una serie de flags informativos del funcionamiento del esclavo.

El sistema de comunicaciones se explicará mas detalladamente en el apartado de las librerías implementadas para controlar este periférico en concreto.

-**TECLADO MATRICIAL:**

Los teclados son uno de los sistemas de interfaz de usuario más utilizados. En este caso, la placa de E/S cuenta con un teclado matricial pasivo con dieciséis teclas, distribuidas en cuatro filas y cuatro columnas.

Figura 10

El teclado se controla mediante ocho señales, cuatro de entrada y cuatro de salida, y está montado sobre el siguiente esquema de conexiones:

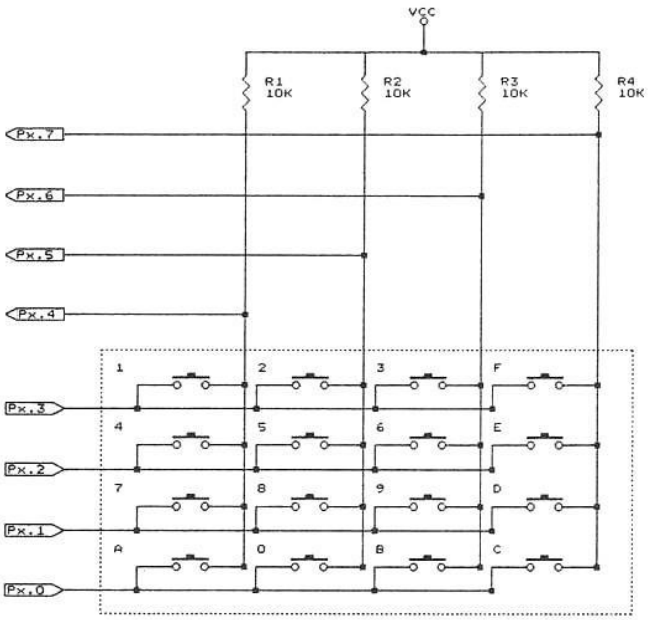


Figura 11

Como se puede observar, en las columnas se han colocado resistencias de pull-up, lo que provoca que si la señal correspondiente a una fila se pone a 0, se podrá leer también un 0 en la columna correspondiente a la tecla que se pulse en esa fila.

Estas conexiones se aprovecharán para reconocer cual es la tecla que se ha pulsado en el teclado en un determinado momento.

En la siguiente figura se presenta el conjunto de la placa de E/S, con todos sus periféricos:

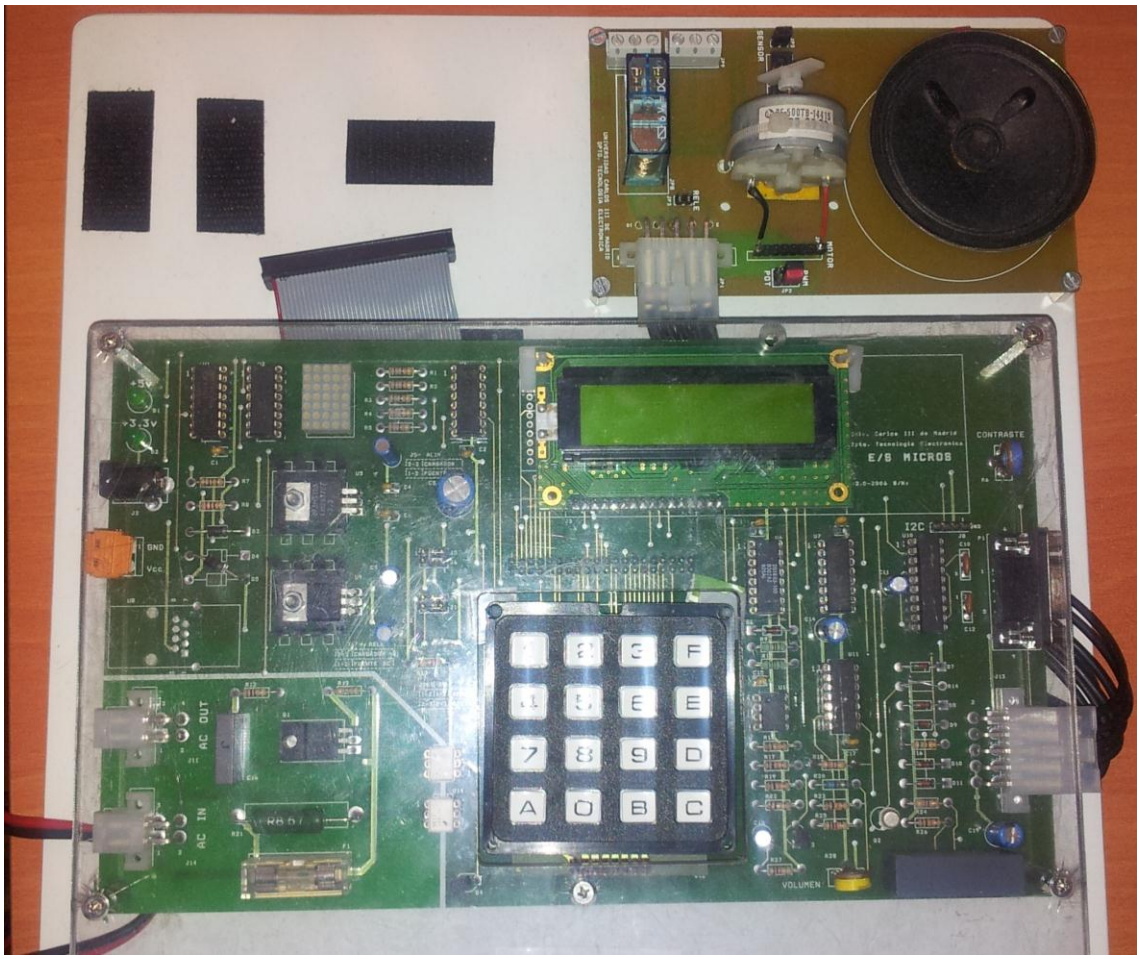


Figura 12

2.3-EL MICROONTROLADOR ARM7:

2.3.1-CARACTERÍSTICAS:

El ARM7 es un microcontrolador de 32 bits diseñado en 1994, a partir de la arquitectura ARM v4T, de tipo Von-Neumann y está especialmente orientado a dispositivos portátiles, debido a sus buenas prestaciones y su bajo consumo.

Se trata de uno de los diseños más populares de ARM, siendo utilizado en numerosos dispositivos electrónicos tan conocidos como la Nintendo DS, el iPod, o la Dreamcast de SEGA… Gracias a esta popularidad, existe una gran comunidad de desarrolladores de aplicaciones para este microcontrolador, y por lo tanto un enorme soporte no solo por parte de los fabricantes, sino también de los usuarios.

El set de instrucciones que se utiliza para desarrollar las aplicaciones del microcontrolador está basado en el sistema “Thumb”, que utiliza instrucciones de 16 bits para ahorrar espacio en memoria, aunque no proporciona un funcionamiento tan eficaz como las instrucciones nativas de ARM, que lógicamente también están soportadas, pero son menos utilizadas debido que ocupan más espacio en memoria.

El ARM7 cuenta con sistemas de interrupciones (FIQ e IRQ), y dos niveles de prioridad configurables para controlarlas, aunque la versión que se utiliza en el laboratorio cuenta además con un sistema de interrupciones vectorizadas, mucho más eficiente que los dos citados anteriormente.

La placa de desarrollo utilizada en el laboratorio monta la versión ARM7TMDI-S, que está especialmente diseñada para trabajar con el set de instrucciones Thumb y para emulación y depuración sobre la propia placa.

Esta versión del ARM7 cuenta con los siguientes periféricos:

-GPIO: Interfaz de entradas y salidas de propósito general.

-UART: Interfaz de comunicación serie asíncrona (dos periféricos).

-I2C: Interfaz de comunicación serie síncrona semi-duplex.

-SPI: Interfaz de comunicación serie síncrona full-duplex.

-SSP: Puerto de interfaz de comunicación serie síncrona.

-TIMER0/1: dos timers de propósito general.

-PWM: Generación de señales con modulación PWM.

-WDT: Watchdog Timer, se encarga de resetear el microcontrolador en caso de que este se mantenga en un estado de error durante un cierto tiempo.

-RTC: Reloj de tiempo real (aplicaciones de hora, fecha, etc).

-CAN: interfaz de comunicación serie para control en tiempo real con una alta seguridad.

-ADC: Conversor analógico-digital.

En la figura que aparece a continuación podemos ver el diagrama de bloques de la placa de desarrollo utilizada en el laboratorio:

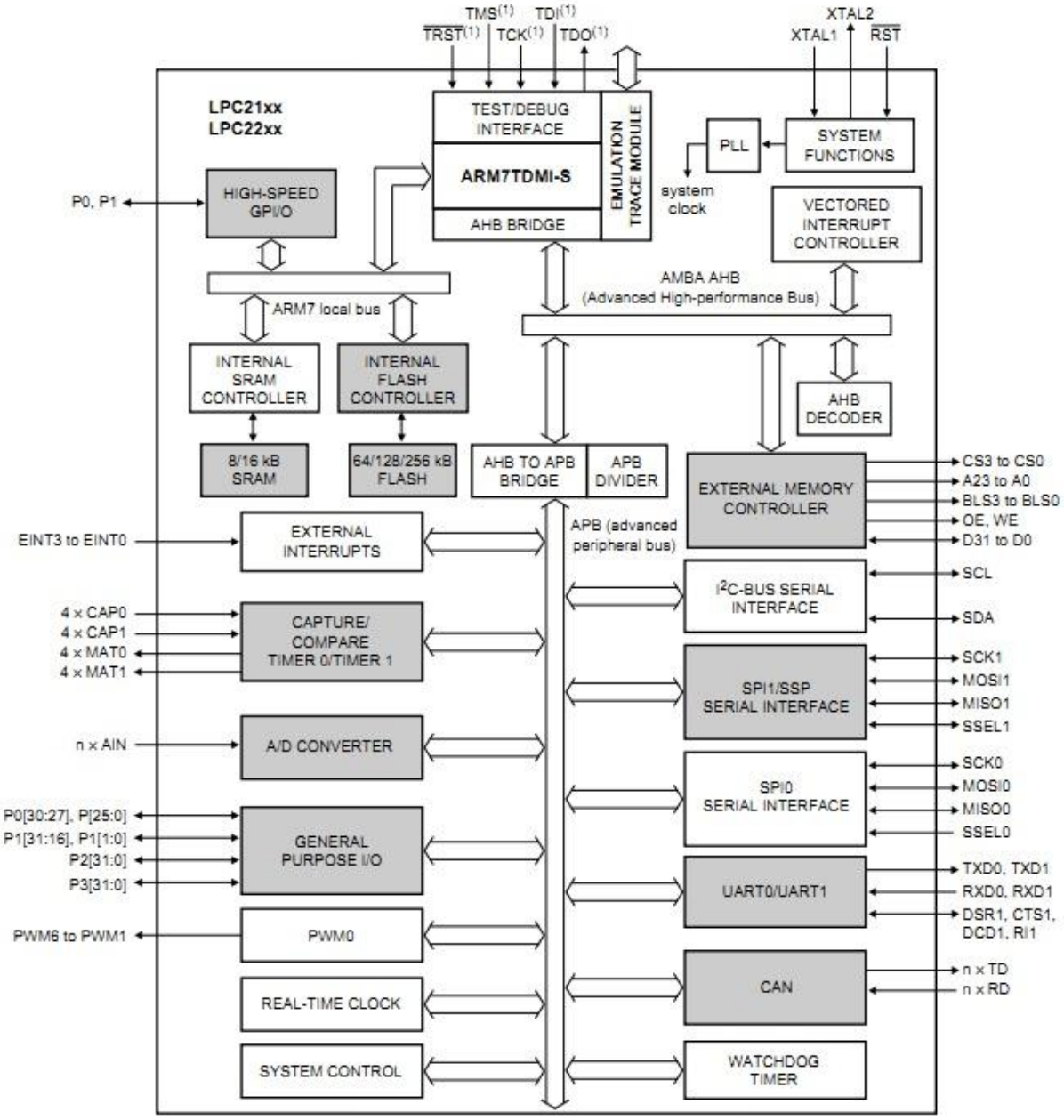


Figura 13

2.3.2-LIBRERÍAS EXISTENTES:

Existen una serie de librerías escritas para la placa de desarrollo basada en el ARM7 que se ha utilizado hasta ahora para trabajar con la placa de E/S.

Estas librerías contienen funciones que facilitan el uso de cada uno de los periféricos en un entorno de programación C, utilizando comandos de nivel medio que modifican los registros del microcontrolador para configurar las funcionalidades deseadas.

Existen librerías para los principales periféricos de la placa de E/S:

-Sensor de infrarrojos: Permite recibir y reconocer comunicaciones moduladas a través del espectro de los infrarrojos.

-Altavoz: Permite emitir sonidos a través del altavoz con la frecuencia y el periodo deseados.

-I2C/EEPROM: Permite leer y escribir caracteres y cadenas de caracteres en la memoria EEPROM a través del protocolo I2C.

-Display LCD: Permite escribir en el display LCD cadenas de caracteres en la posición que se desee dentro del mismo.

-Matriz de LEDs: Permite iluminar una línea de LEDs de la longitud deseada en la posición deseada de la matriz.

-PWM: Permite generar una onda modulada mediante PWM para controlar la velocidad del motor de corriente continua.

-Teclado: Permite reconocer la tecla que se ha pulsado en el teclado.

-Timer: Permite temporizaciones y generación de señales cuadradas para estimular el altavoz.

-UART: Permite enviar un dato a través de la UART (Universal Asynchronous Receiver-Transmitter).

2.4-EL MICROCONTROLADOR ARM Cortex-M3:

2.4.1-DESCRIPCIÓN:

El Cortex-M3, que es el corazón del STM32L, es un microcontrolador de propósito general de 32 bits, presentado en 2005. Su diseño esta basado en la versión v7-M de los microprocesadores ARM, que cuenta con una arquitectura Harvard y una reducción del consumo notable en comparación con versiones anteriores.

A continuación se puede ver el diagrama de bloques del Cortex-M3:

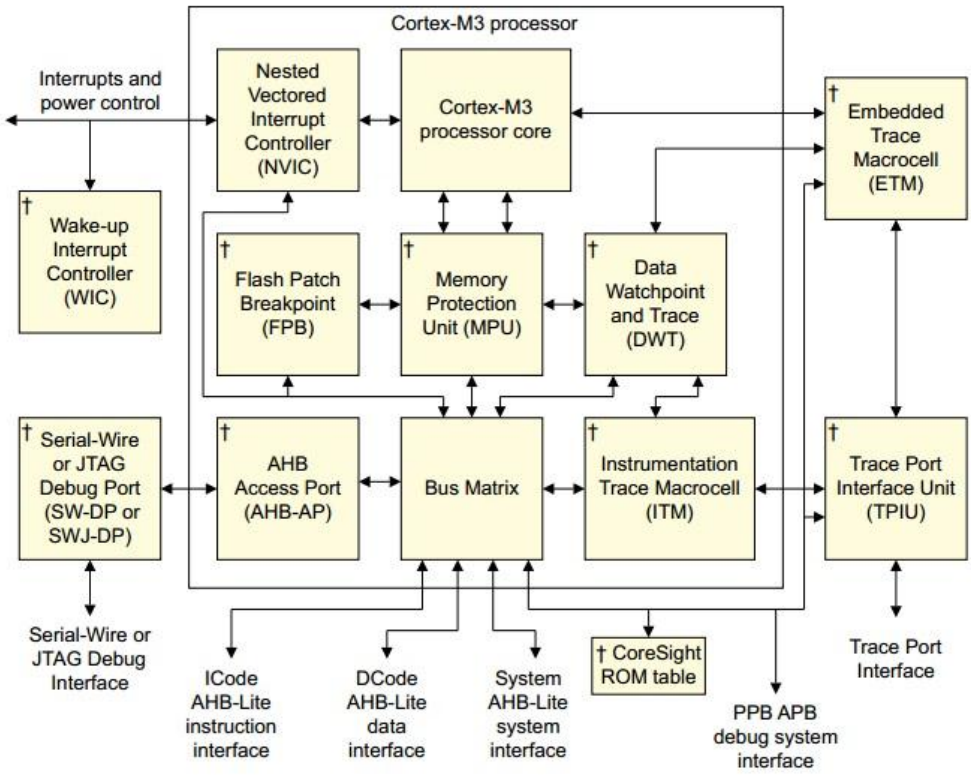


Figura 14

A continuación se detallan algunas de las principales características de este micorcontrolador:

-Utiliza el set de instrucciones conocido como Thumb-2, que es una evolución del set utilizado por el ARM7. Este nuevo juego de instrucciones reúne las ventajas de los dos juegos de instrucciones soportados por el ARM7: por un lado, al estar compuesto por instrucciones de 16 y 32 bits, ocupa poco espacio en memoria y por otro lado, es altamente eficiente y flexible.

-Integra control del Sleep Mode y diversos modos de consumo.

-Es capaz de monitorizar hasta 240 interrupciones físicas externas o internas (IRQ), el mismo numero de interrupciones de Wake-Up (despertar) y una interrupción no enmascarable (NMI), con entre 8 y 256 niveles de prioridad configurables, gracias al nuevo sistema de control de interrupciones NVIC (Nested Vectored Interrupt Controller). Además, el NVIC también permite escribir las rutinas de interrupción completamente en C, sin necesidad de utilizar el lenguaje ensamblador.

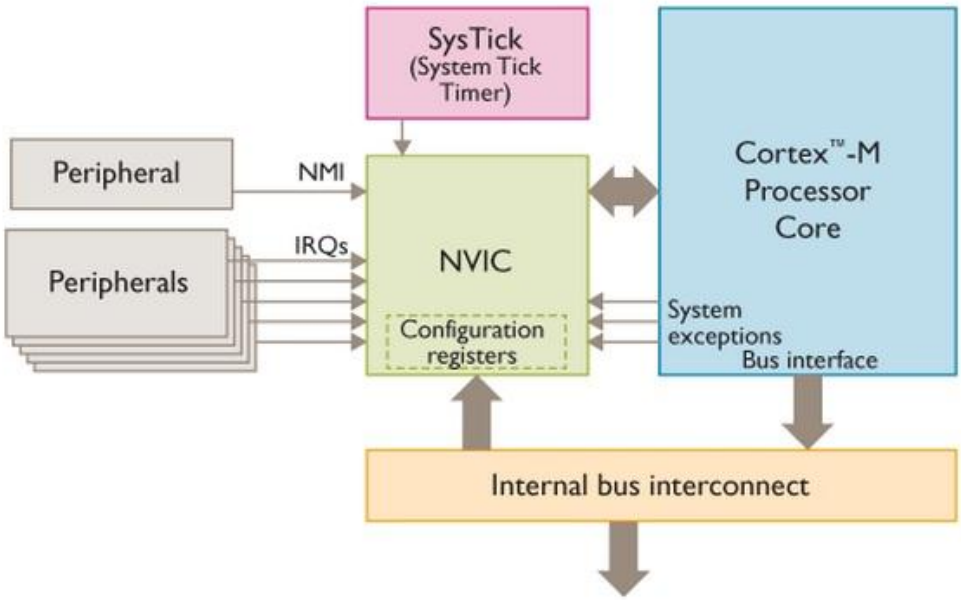


Figura 15

La existencia de diversos niveles de prioridad en el NVIC permite que si mientras se esta ejecutando una interrupción salta el flag de otra con una prioridad mayor, se detiene la interrupción menos prioritaria y se ejecuta la que

acaba de saltar, reanudando la primera una vez ésta ha terminado. A continuación podemos ver un esquema explicativo de esta aplicación:

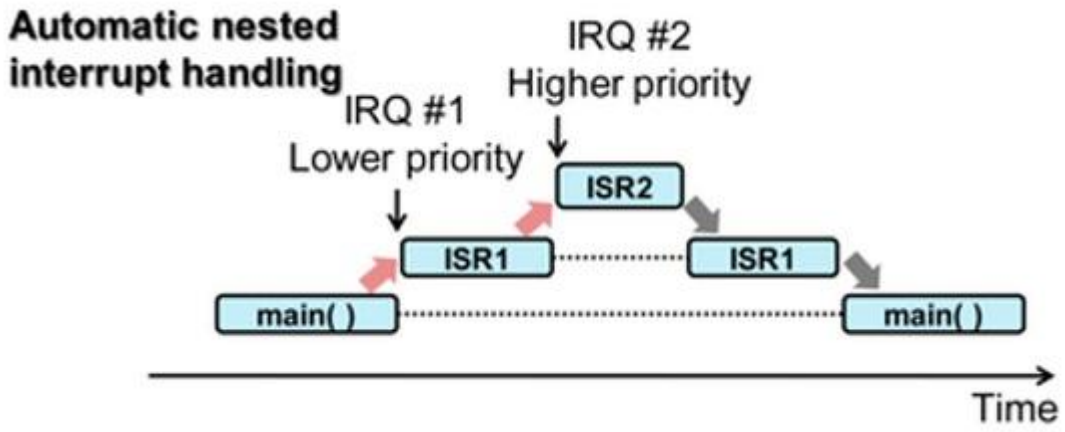


Figura 16

-Se trata de una arquitectura altamente retro-compatible, es relativamente sencillo portar aplicaciones desde el ARM7 al Cortex-M3, gracias al set de instrucciones Thumb2 y a la gran cantidad de documentación sobre este tema publicada por ARM, para conservar los usuarios que utilizaron arquitecturas anteriores y ahora necesitan actualizarse.

-Cuenta con instrucciones integradas para el manejo de bits de memoria, y con un sistema de Bit-Banding.

-Integra una Unidad de Protección de Memoria (MPU).

-Utiliza CMSIS (Cortex Microcontroller Software Interface Standard), una capa de software desarrollada especialmente para los procesadores Cortex-M. Estas librerías implementan un software de intercomunicación sencillo y robusto entre el microcontrolador y los periféricos, haciendo más cómoda la programación de aplicaciones y más sencilla la reutilización del software desarrollado.

2.4.2- DIFERENCIAS Y VENTAJAS RESPECTO AL ARM7:

-Arquitectura Harvard frente a Von Neuman: al tener dos buses independientes permite un funcionamiento más completo, aunque es más complejo de controlar.

-El ARM7 necesita una serie de comandos en lenguaje ensamblador para la inicialización del microcontrolador, mientras que el Cortex-M3 permite inicialización y depuración mediante código C/C++, además del ensamblador. Esto hace que sea más sencillo desarrollar y depurar el código al tratarse de un lenguaje de alto nivel.

-A la hora de controlar interrupciones, el NVIC incorporado en el Cortex-M3 hace que no sea necesario recurrir a un controlador de interrupciones externo, cosa que sí sucede en el ARM7. Esto genera la siguiente situación: la latencia de respuesta ante interrupciones del Cortex-M3 es fija, y mayor que la latencia mínima del ARM7, pero la latencia media del segundo, al depender de controladores de interrupción externos es mucho mayor que la del primero. En resumen: en el mejor caso, el ARM7 es capaz de reaccionar con más rapidez que el Cortex-M3 ante una interrupción, pero en la práctica en la mayoría de las situaciones se obtiene una mejor respuesta con el segundo.

-Ambas arquitecturas soportan el set de instrucciones nativo de ARM, pero el Cortex-M3 soporta Thumb2, mientras que el AMR7 solo soporta Thumb.

El problema del set Thumb es que, aunque ocupa menos espacio en memoria que el set nativo de ARM, el rendimiento es notablemente inferior que con éste. Sin embargo, Thumb2 aúna las ventajas de ambos juegos de instrucciones, consiguiendo un alto rendimiento en un espacio de memoria reducido.

-como se puede observar en la siguiente tabla, el rendimiento del COrtex-M3 es notablemente superior al del ARM7:

	ARM7	Cortex-M3
Max Frequency	85 MHz	100 MHz
DMIPS/MHz	0.94	1.25
Max DMIPS	80	125
DMIP/mW	3.36	3.75
Area (mm^2)	0.62	0.37

Figura 17

-Aunque ambas arquitecturas están diseñadas con la intención de tener unos consumos moderados, el Cortex-M3 posee mecanismos más avanzados de ahorro de energía, optimización de procesos y configuración de “sleep mode” y “wake-up”, por lo que se puede afirmar que tiene un consumo de energía menor en situaciones similares.

-El ARM7 es un microcontrolador bastante más extendido en la actualidad que el Cortex-M3, aunque la mayoría de los fabricantes están dando más relevancia a la fabricación del segundo con el paso de los años.

Conclusiones:

El ARM7 tiene una mayor comunidad de usuarios detrás, numerosa documentación disponible y puede resultar más barato a la hora de ser utilizado como repuesto en ciertos dispositivos, comparado con portarlos al nuevo Cortex-M3.

Por otro lado, el Cortex-M3 es mas sencillo de programar y adaptar gracias a la programación 100% en C/C++, posee un controlador de interrupciones mejorado, mejor rendimiento en casi todos los casos y un mejor control del consumo.

2.5-LA PLACA DE DESARROLLO STM32L-DISCOVERY:

La STM32l-discovery es una placa de desarrollo basada en el microcontrolador STM32L152RBT6, una de las versiones de muy bajo consumo (ultra-low power) del microcontrolador Cortex-M3 desarrolladas por el fabricante de componentes electrónicos ST, construido usando una tecnología de 130 nm. Se trata de un dispositivo de bajo coste, ya que esta diseñado para favorecer el aprendizaje y el testeo de las diferentes características de este microcontrolador.

La placa de desarrollo cuenta con los siguientes accesorios:

-Interfaz STLINK/V2 empotrada, que permite flashear aplicaciones en la memoria de la placa y depurarlas on-board.

-Un pequeño display LCD de 24 segmentos.

-Dos LEDs de usuario, uno verde y otro azul que se pueden utilizar para testear el funcionamiento del GPIO y como “chivatos” cuando el programa que se está depurando pasa por algún punto de interés o realiza alguna acción especialmente representativa.

-Dos LEDs informativos, uno rojo que indica cuando está conectada la alimentación de la placa vía USB y otro que alterna entre verde y rojo cuando está activa la comunicación con un PC a través del mismo conector.

-Dos pulsadores, uno denominado RESET, que reinicia la aplicación que está corriendo la placa, y otro denominado USER BUTTON, que permite recibir una señal a través del GPIO, lo que puede leerse de modo tradicional a través del pin asociado al pulsador o configurarse para que provoque una interrupción externa del sistema.

-Un sensor táctil lineal y cuatro sensores táctiles de posición, que tienen las mismas aplicaciones que el pulsador USER BUTTON gracias a una serie de librerías incluidas con el IDE de la placa de desarrollo, y que también podemos descargar de la web del fabricante.

-Un dispositivo medidor de corriente.

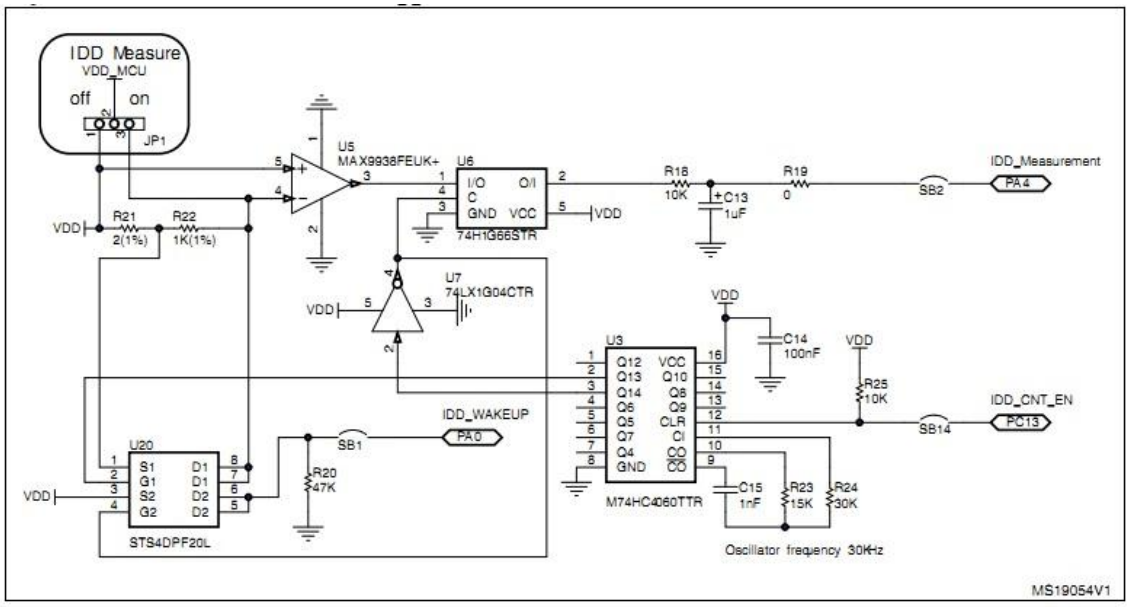


Figura 18

-Dos entradas de alimentación, una a través del puerto USB y otra para alimentación eterna de la placa a 3.3V o 5V.

-La placa cuenta también con una serie de unidades de memoria:

-Memoria Flash: 128 kB

-Memoria RAM: 16 kB

-Memoria EEPROM: 4 kB

A continuación se puede ver una imagen de la placa de desarrollo STM32L-Discovery y el diagrama de bloques del hardware que incluye:

Figura 19

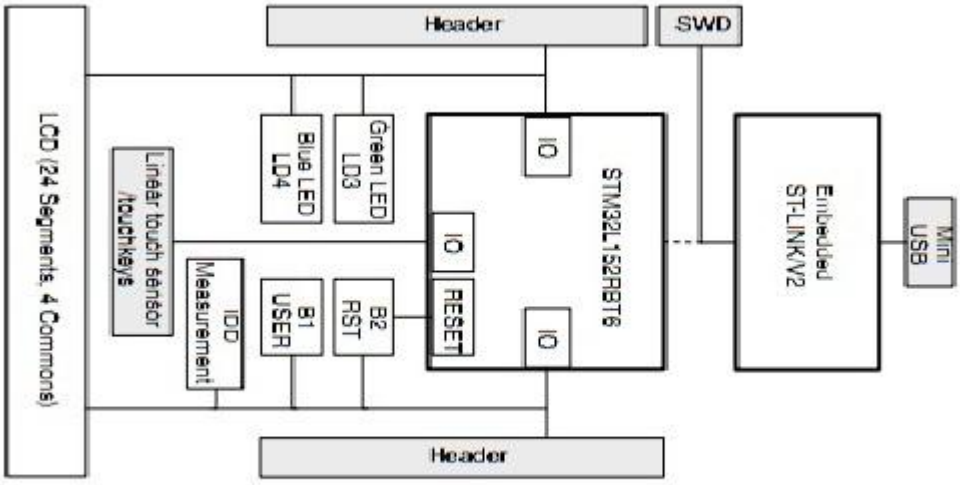


Figura 20

La placa cuenta también con un sistema DMA (Direct Acces Memory), que permite utilizar hasta 12 canales para acceder a la memoria de forma independientemente al resto de operaciones realizadas por el microprocesador, lo que permite un gran ahorro en tiempos de ejecución de programas que utilicen la memoria de la placa de desarrollo para leer o cargar datos.

A continuación se muestra el diagrama de bloques del STM32L-Discovery, con todos los periféricos que incluye esta placa de desarrollo:

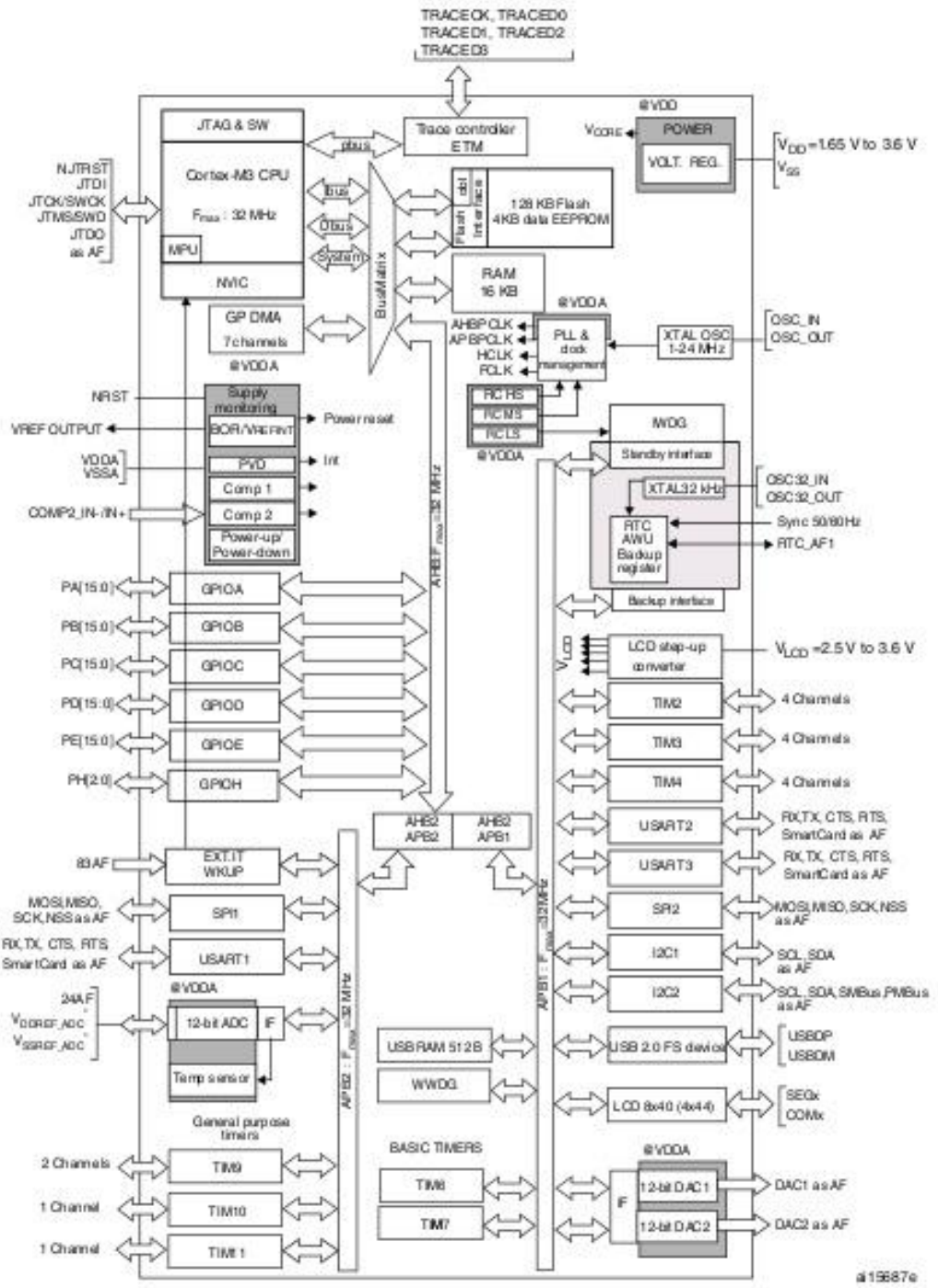


Figura 21

Como se puede observar en la Figura 2.19, el STM32L-Discovery cuenta con los siguientes periféricos:

-6 Timers de propósito general, basados en contadores de 16 bits con auto-recarga y capacidad de precarga de 16 bits. Cuentan con entre 1 y 4 canales independientes que pueden actuar como input capture, output compare, generación de PWM o generación de pulsos.

Los timers 2, 3, y 4 tienen además la capacidad de activar el modo DMA para trabajar simultáneamente con registros de memoria mientras el microcontrolador ejecuta otras tareas.

-2 Timers básicos, utilizados preferentemente junto con los conversores digital-analógicos, aunque también pueden utilizarse como contadores de tiempos normales.

-51 puertos de E/S de propósito general (GPIOs). Estos pines pueden ser configurados como salidas (en modo push-pull u open-drain), entradas o “Alternate Functions”, lo cual les proporciona una variedad de funcionalidades tremendamente útil en una placa de desarrollo.

Además, otra de las funcionalidades más interesantes es la posibilidad de configurarlos como receptores de interrupciones externas (EXTI), ya que, como se ha explicado anteriormente, el controlador de interrupciones del Cortex-M3 es uno de los principales atractivos de esta nueva arquitectura de ARM.

-Dos interfaces de comunicación I2C, que pueden operar en modo esclavo, maestro y multimaestro, y en modos de comunicación estándar y rápido. Además, la funcionalidad del DMA también puede ser activada a través de este bus.

-Dos interfaces de comunicación SPI. Estos buses pueden funcionar en modo maestro y esclavo a una velocidad de hasta 16 Mbits/s, y llevar a cabo una comunicación semi-duplex o simplex, según la configuración que se les aplique. Este bus también es capaz de activar el DMA.

-Tres interfaces de comunicación USART, con velocidades de transmisión de hasta 4Mbits/s y capacidad de controlar el DMA.

-Un puerto de comunicaciones USB, principalmente utilizado para alimentar la placa, cargar programas en la memoria y depurarlos, pero que también puede ser utilizado como bus de comunicaciones en una aplicación.

-Un conversor analógico-digital de 12 bits con 20 canales, que puede configurarse para efectuar una única lectura en un momento dado, o para escanear el valor de la entrada de manera continua. La frecuencia de muestreo debe ser controlada por uno de los timers de propósito general, y se puede aprovechar el DMA para agilizar el funcionamiento del conversor.

-Dos conversores digital-analógico de 12 bits con 2 canales, que permiten generación de ondas triangulares, simulación de ruido o acceso a registros a través de DMA para conocer la señal que deben convertir.

-Una interfaz preconfigurada de control del display LCD.

-Dos comparadores, que pueden tomar como tensión de referencia un valor interno o uno externo, obtenido desde la salida de un DAC o desde un pin perteneciente al GPIO.

Estos periféricos se encuentran distribuidos a través de dos buses APB (Advanced Peripheral Bus), que facilitan el funcionamiento independiente de cada uno de ellos.

Para poder utilizar la placa de desarrollo es necesario registrarse en la web de ST, y descargar e instalar uno de los IDEs disponibles para la misma.
En este caso se ha utilizado Atollic TrueSTUDIO, sobre el que ya se ha hablado en el capitulo correspondiente.

3-ENTORNO DE DESARROLLO

3.1-INTRODUCCIÓN

A la hora de trabajar con una placa de desarrollo, y sobretodo en el ámbito docente, una de las necesidades fundamentales es contar con un buen entorno de desarrollo.

El entorno de desarrollo es el conjunto de aplicaciones de software que permiten al usuario interactuar con su placa de desarrollo: cargar programas en memoria, depurarlos y obtener información de lo que está sucediendo dentro del hardware con el que están trabajando. El entorno de desarrollo debe tener un equilibrio entre número de funcionalidades y usabilidad: un entorno de desarrollo con un elevado número de botones de acción y ventanas informativas puede ofrecer mucha funcionalidad, pero llegaría a saturar al usuario, impidiéndole concentrarse en la información importante, mientras que una interfaz demasiado sencilla puede ofrecer demasiada poca información. Cuanto más equilibrado sea el entorno de desarrollo, más información tendrá el alumno que está realizando las practicas, y más fácil le será comprender el funcionamiento de un microcontrolador.

Por eso se plantea el desarrollo de un nuevo entorno de desarrollo. No porque Atollic TrueSTUDIO sea un mal entorno de desarrollo, sino porque su versión Lite tiene unas pequeños defectos que, solucionados, harían de él un gran entorno de desarrollo.

3.2-IDE LIBRE ECLIPSE

3.2.1-¿POR QUÉ ECLIPSE?

Aunque Atollic True Studio funciona perfectamente y apenas cuenta con limitaciones, el hecho de contar con un IDE propio y que carezca del límite de "break-points" impuesto en Atollic True Studio es suficiente para motivar el desarrollo de esta parte del proyecto, además de evitar depender de un entorno de desarrollo propietario y que el fabricante puede modificar o dejar de dar soporte cuando lo desee.

Atollic TrueSTUDIO es una versión mejorada y adaptada a la programación de placas de desarrollo del IDE de software libre Eclipse. Dado que se considera un entorno de desarrollo muy adecuado para el ámbito docente, gracias a la claridad de su interfaz y al gran número de posibilidades que ofrece, es lógico pensar que el hecho de contar con un IDE con las mismas cualidades pero sin ninguna de las limitaciones es la mejor solución. Por lo tanto, se decide utilizar Eclipse como base del nuevo entorno de desarrollo.

Además, Eclipse es el IDE de software libre más utilizado por la comunidad internacional, por lo que cuenta con mucho soporte en la web y está en constante evolución.

3.2.2-MÓDULOS DE SOFTWARE REQUERIDOS

Una vez que se sabe sobre qué base se desea trabajar, se realiza una búsqueda en la web de configuraciones que permitan desarrollar un IDE basado en Eclipse para controladores de la familia Cortex-M3.

Después de investigar por diversos foros y webs dedicados a la los microcontroladores, se llega a la conclusión de que la forma más sencilla y fácil de usar es aprovechar la aplicación GDB Server de Atollic True Studio para depurar mediante Eclipse, crear unas configuraciones de compilación y depuración adecuadas y conseguir un conjunto de librerías parecidas a las de Atollic o desarrolladas a partir de éstas, para que el entorno y la forma de trabajar sean relativamente similares a los de Atollic y se pueda aprovechar la experiencia adquirida con el entorno de desarrollo.

Una de las webs en las que se ha podido encontrar más información es http://www.chibios.org, que proporciona diversos tutoriales y explicaciones de cómo trabajar con microcontroladores, e incluso sus propias librerías, de las que se hablará más adelante.

El principal problema que se plantea a la hora de intentar llevar a cabo la implementación del IDE es la escasa información existente acerca del STM32L-Discovery, comparada con otros miembros de la familia, como el STM32Fxx o el STM32VL. A tenor de esta situación, se presentan dos opciones: buscar una aplicación concreta para el STM32L-Discovery o adaptar una existente para otro miembro de la familia a esta placa.

Inicialmente se opta por la segunda opción debido a la aparente similitud entre los diferentes miembros de la familia Cortex M3 y la dificultad a la hora de encontrar material específico y gratuito para el STM32L-Discovery:

La configuración de Eclipse y la inclusión del GDB Server de Atollic en el mismo se llevan a cabo fácilmente siguiendo las guías que se pueden encontrar en la web de ChibiOS, como se explicará en el siguiente apartado.

Para poner en funcionamiento el entorno de desarrollo es necesario contar con el siguiente paquete de software:

-IDE: el entorno de desarrollo en si, se utilizará Eclipse (versión Indigo), en su edición para programadores de C/C++.

-Para que Eclipse funcione correctamente es necesario tener instalada la última versión de Java de Oracle.

-Una toolchain basada en GCC y adaptada para los microcontroladores de ARM: Sourcery CodeBench Lite for ARM EABI.

-Aplicación que permita ejecutar los comandos de Linux en Windows: Cygwin.

-Servidor de depuración y comunicaciones: GDB Server (extraído de Atollic TrueSTUDIO).

Todo el software necesario está incluido en la documentación del proyecto, y se recomienda usar las versiones indicadas, ya que pueden surgir incompatibilidades en caso de usar versiones actualizadas de alguna de las aplicaciones.

3.3-LIBRERÍAS BÁSICAS Y PLANTILLA DE PROYECTO

Una vez se ha decidido la forma en que se va a implementar el entorno de desarrollo, el siguiente paso es encontrar unas librerías que permitan programar la placa de desarrollo STM32L-Discovery.

3.3.1-OPCIÓN 1: LIBRERÍAS ChibiOS

Aunque existen en internet varias opciones de librerías de código abierto para microprocesadores ARM, la que parece más desarrollada y fiable es la que proporciona ChibiOS, debido al gran número de placas de desarrollo que soporta, incluida la STM32L-Discovery.

Este conjunto de librerías contiene un gran número de funciones para diversas placas, y sigue una estructura similar a la de las librerías de Atollic, por lo que parece adecuada para el proyecto que nos ocupa.

Sin embargo, al igual que el resto de librerías que se han encontrado en internet, parece que se han desarrollado inicialmente para el STM32Fxx (posiblemente la placa de desarrollo más popular basada en el Cortex-M3), y posteriormente portado a otras versiones del microcontrolador, como la que nos ocupa.

Esto conlleva que no todas las versiones de las librerías incluyen soporte para la STM32L-Discovery, y las que lo incluyen no llegan a funcionar del todo correctamente, ya que son versiones en continuo desarrollo, pero no totalemnte operativas.

Para solucionar esta situación es necesario añadir ciertas librerías externas de funciones básicas que no están incluidas, y realizar ciertas modificaciones en otros ficheros para que se adapten a la placa y sean compatibles entre sí, ya que hay funciones incompatibles, variables que

cambian de nombre entre unos ficheros y otros, restos de código sin funcionalidad aparente que provocan errores de compilación…

Se puede afirmar que estas librerías son un proyecto muy ambicioso y tremendamente completo pero que, al menos de momento, no son una solución de garantías para el STM32L-Discovery.

Después de realizar cierto número de pruebas con las librerías de ChibiOS, todas ellas insatisfactorias por diversos motivos, y de demostrarse que el hecho de preparar unas librerías totalmente operativas a partir de ellas supondría un gasto de tiempo y recursos de los que no se dispone, se decide buscar cambiar la línea de trabajo y buscar otra opción.

3.3.2-OPCION 2: LIBRERÍAS ORIGINALES DE ATOLLIC

Debido a la escasez antes comentada de soluciones específicas para la STM32L-Discovery, se plantea la opción de reciclar las librerías disponibles de Atollic.

Sin embargo, esta opción también se descarta debido a que dependen de demasiadas librerías de bajo nivel que habría que conseguir y a que, tras varios test, parecen tener algunas incompatibilidades si no son utilizadas bajo el IDE para el que fueron diseñadas.

3.3.3-OPCION 3: LIBRERÍAS DE ATOLLIC ADAPTADAS

Después de descartar las dos primeras opciones, se realiza una nueva búsqueda en la web, y se encuentra la web de un aficionado japonés que ofrece su propia versión modificada de las librerías de Atollic, la cual permite compilar con un IDE basado en Eclipse y trabajar con el STM32L-Discovery.

La dirección de la web es la siguiente:

http://www.page.sannet.ne.jp/kenjia/STM32_main.html

Aunque la web está en japonés, se puede utilizar el traductor de google para obtener una traducción rudimentaria del contenido. Se puede acceder a un artículo sobre el STM32L-Discovery, en el cual se muestra un método similar al explicado en la web de ChibiOS para configurar un IDE basado en Eclipse y se ofrece una versión de las librerías de Atollic, compatibles con este entorno de desarrollo y, según la web, totalmente funcionales. Leyendo las librerías a fondo, parece que realmente están basadas en las del STM32F10x, o al menos varios de los ficheros provienen de éstas, ya que hay diferencias de nomenclatura en varios puntos entre éstas y las originales de Atollic para el STM32L-Discovery.

Al probar estas librerías se encuentra un error de compilación debido a un pequeño bug, pero se puede solucionar cambiando unas pocas líneas de código, del archivo core_cm3.c, como se indica en uno de los documentos anexos incluidos, “HowTo_ToolChain_STM32_Ubuntu.pdf”, en las paginas 19 y 20.

De esta forma, se consigue una plantilla de trabajo para el STM32L-Discovery totalmente operativa, ya que el ejemplo incluido con las librerías que proporciona la web japonesa es la demo que trae pre-cargada la placa, por lo que incluye librerías del sensor táctil, etc.

3.3.4-ELECCION DE LAS LIBRERÍAS BÁSICAS

Una vez se han valorado las tres opciones existentes, se elije la tercera, por ser una librería totalmente funcional y, además, muy similar a las originales de Atollic, con las que ya se tiene experiencia programando.

Las otras dos opciones también podrían llegar a ser válidas, pero requieren demasiadas modificaciones para funcionar correctamente con este entorno de desarrollo.

3.4-CONFIGURACION DEL SISTEMA

A continuación se dan una serie de indicaciones para instalar el entorno de desarrollo en un ordenador con Windows.

El primer paso es instalar en el equipo todo el software indicado en el apartado 3.2.2 (excepto Eclipse, que no será necesario). Es recomendable instalar las aplicaciones en la raíz del disco C:/, para evitar problemas con las carpetas, ya que se va a trabajar emulando comandos Linux, y éste tiene problemas con los nombres de carpeta demasiado largos o con espacios.

Una vez las aplicaciones estén instaladas, se debe descomprimir Eclipse en una carpeta expresamente creada para ello (preferentemente también en la raíz del disco C:/) y se crea una carpeta más, que será usada como workspace. Esta capeta puede estar donde se desee, pero es preferible elegir una ruta sencilla.

Una vez hemos seguido las instrucciones anteriores, el segundo paso es configurar eclipse según las guías existentes en la siguiente web:

http://www.chibios.org/dokuwiki/doku.php?id=chibios:guides:stlink_eclipse&s[]=stm32l

Lo primero se debe comprender es que Eclipse no se instala como la mayoría de los programas en Windows, sino que funciona como un “portable”: se coloca la carpeta que se ha descomprimido en el directorio deseada y dentro de ella se encuentra el ejecutable que lanza Eclipse. Los pasos que se deben llevar a cabo para configurar el proyecto son los siguientes:

-Permitir que Eclipse busque actualizaciones automáticamente desde “Help”→ “Check for updates”

-Instalar la extensión "C/C++ Hardware Debugging" para Eclipse desde "Help"→ "Install new software".

-Utilizando un programa de descompresión de archivos, como 7zip, extraer la carpeta ST-Link_gdbserver del instalador de Atollic TrueStudio, y guardarla, por ejemplo, en C:/Utils.

-Para hacer que Eclipse utilice la aplicación anterior para trabajar con el STM32L-Discovery, ir hasta "Run"→"External Tools"→ "External Tools Configuration", crear un nuevo lanzador haciendo clic derecho en "Program" → "New". A continuación se configuran las siguientes pestañas como aparece en las imágenes y se guarda la configuración.

Main (los path cambiarán según donde se ubique el lanzador):

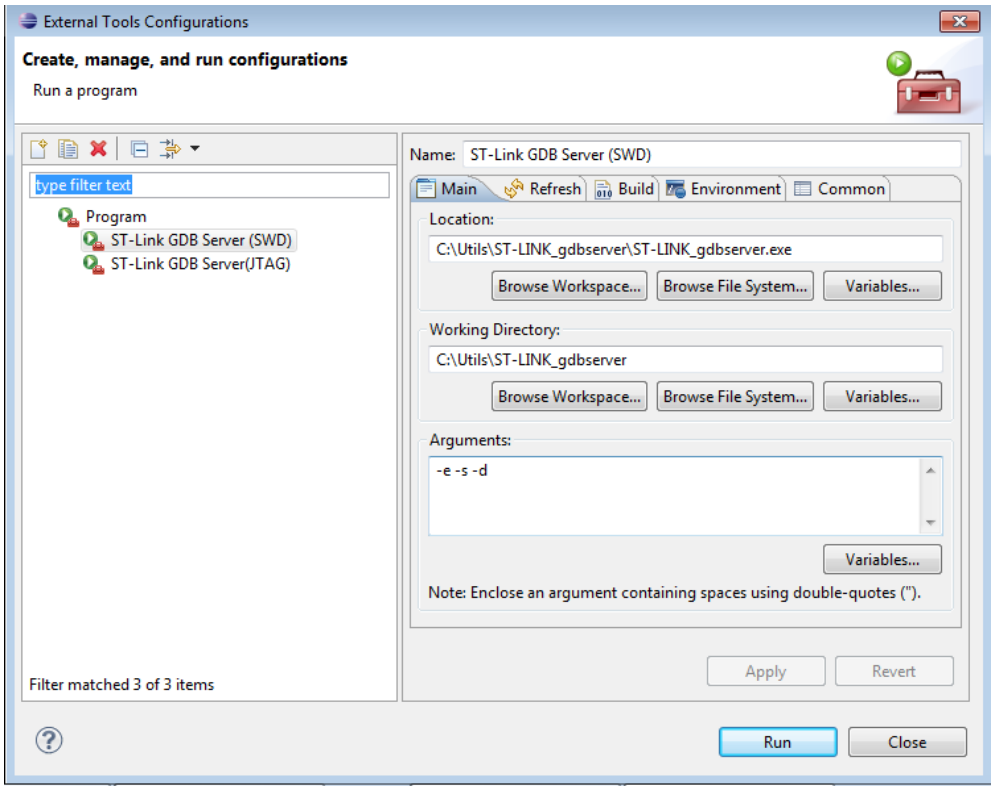


Figura 22

Common:

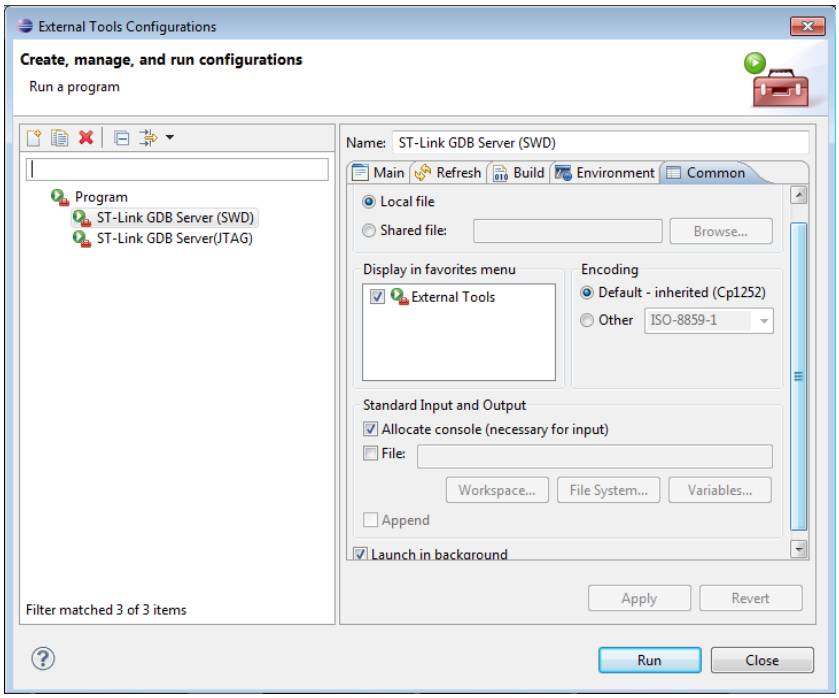


Figura 23

Build:

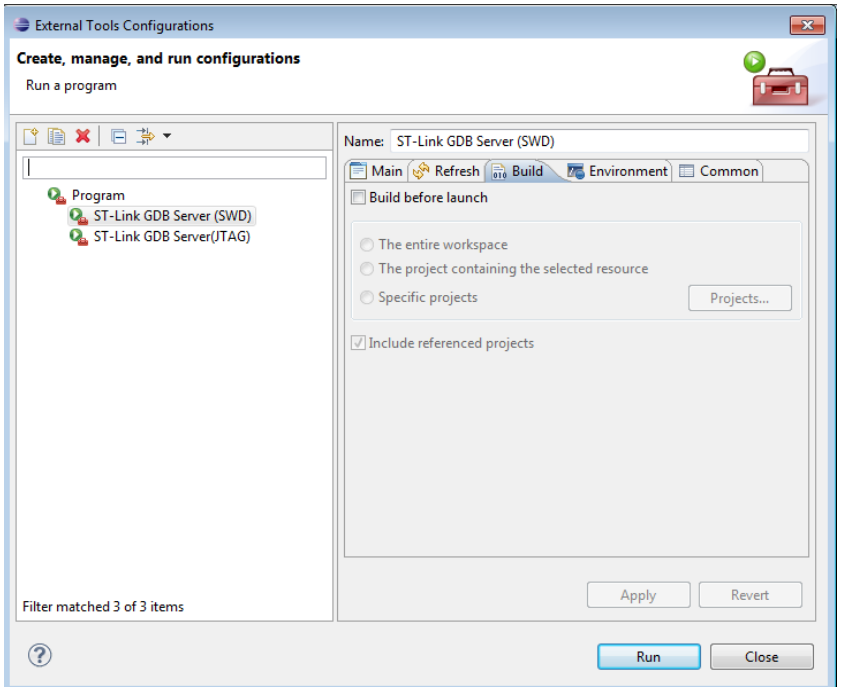


Figura 24

-Antes de seguir con la configuración de Eclipse, es necesario preparar la plantilla del proyecto para poder llevar a cabo su configuración de depuración. Para ello, se puede acceder al siguiente link, y descargar el proyecto de prueba que ST incluye en las placas, adaptado para Eclipse: https://skydrive.live.com/?lc=1041&ppud=4#cid=E726C1E354E854B2&id=E726C1E354E854B2%21122

A este proyecto habrá que realizarle una serie de modificaciones para tener una plantilla a partir de la cual crear nuevos proyectos, así que es más cómodo utilizar la carpeta “plantilla” que se incluye con la documentación.

-Una vez ya se tenga plantilla preparada, se debe importar el proyecto a Eclipse:

“File”→ “Import”→ “General”→ “Existing projects into workspace”→ “Next”→ “Browse”→ indica la ruta de la carpeta de la plantilla → seleccionar “Copy projects into workspace”→ “Finish”.

-Para evitar posibles problemas en la compilación: “Project”→ “Preferences”→ “Language mapping”→ asignar GNU C a C Header file y C Source file.

-A continuación se va a preparar la configuración de depuración: se despliega el menú de “Debug”→ “Debug configurations”→se crea una nueva configuración dentro de “GDB hardware debugging”, configurando las pestañas como se puede ver en las imágenes que aparecen a continuación:

Main:

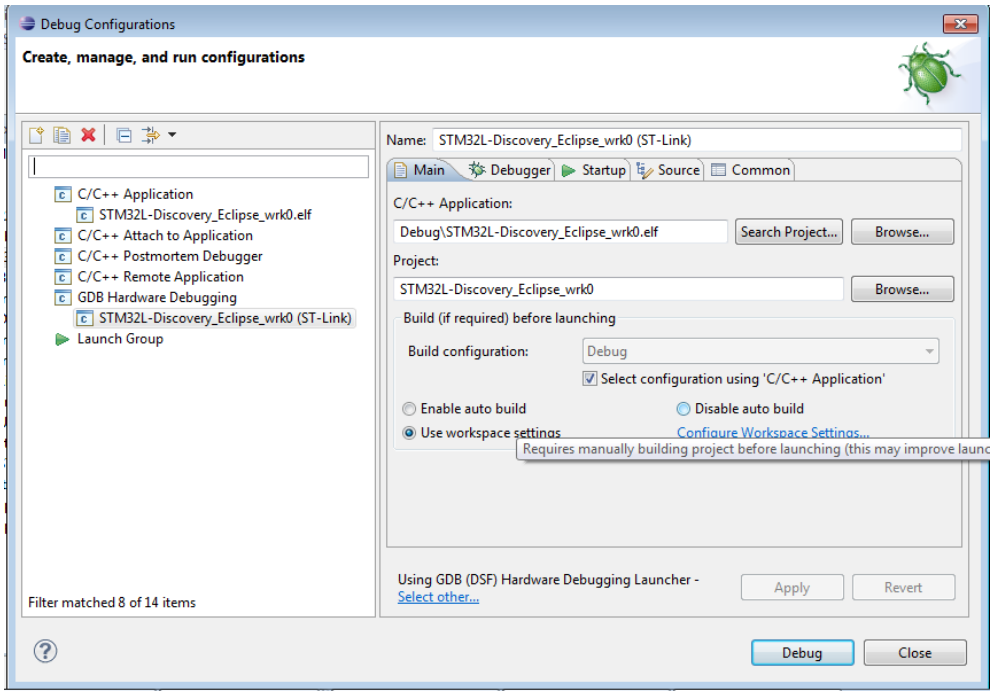


Figura 25

Debugger:

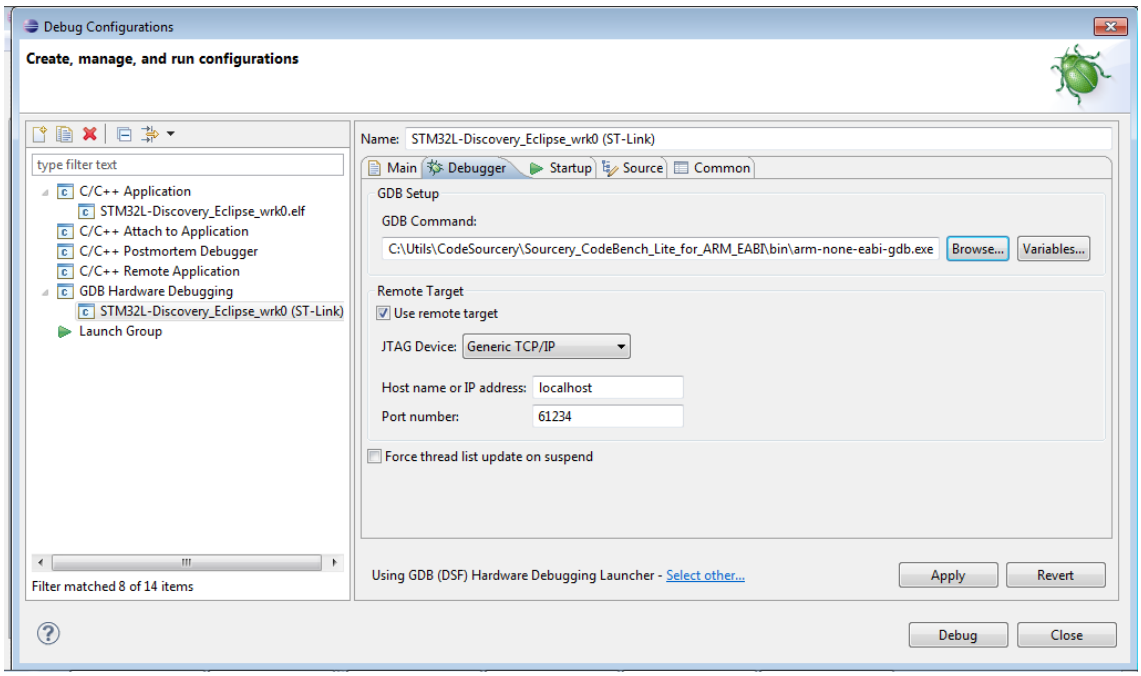


Figura 26

Startup:

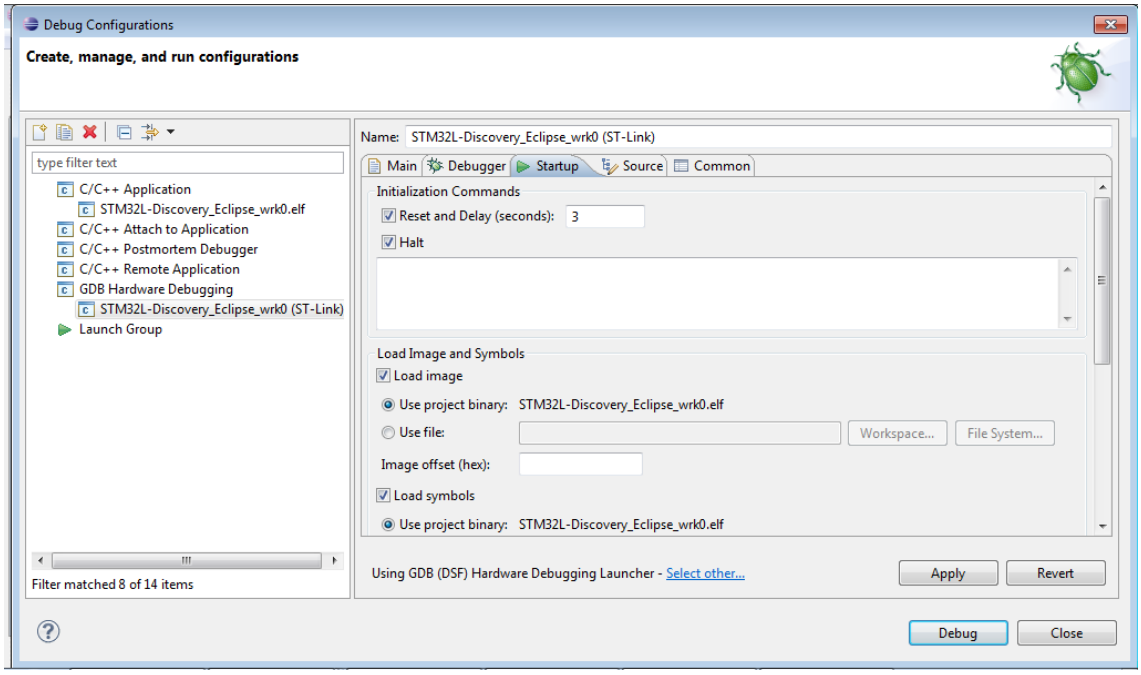


Figura 27

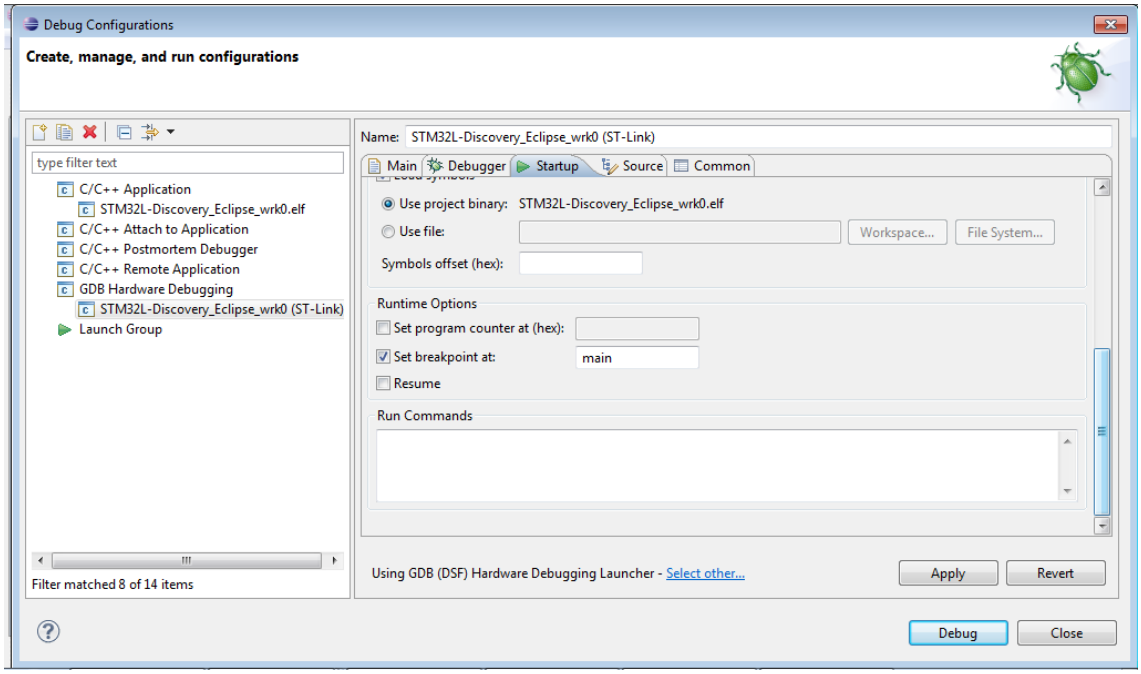


Figura 28

common:

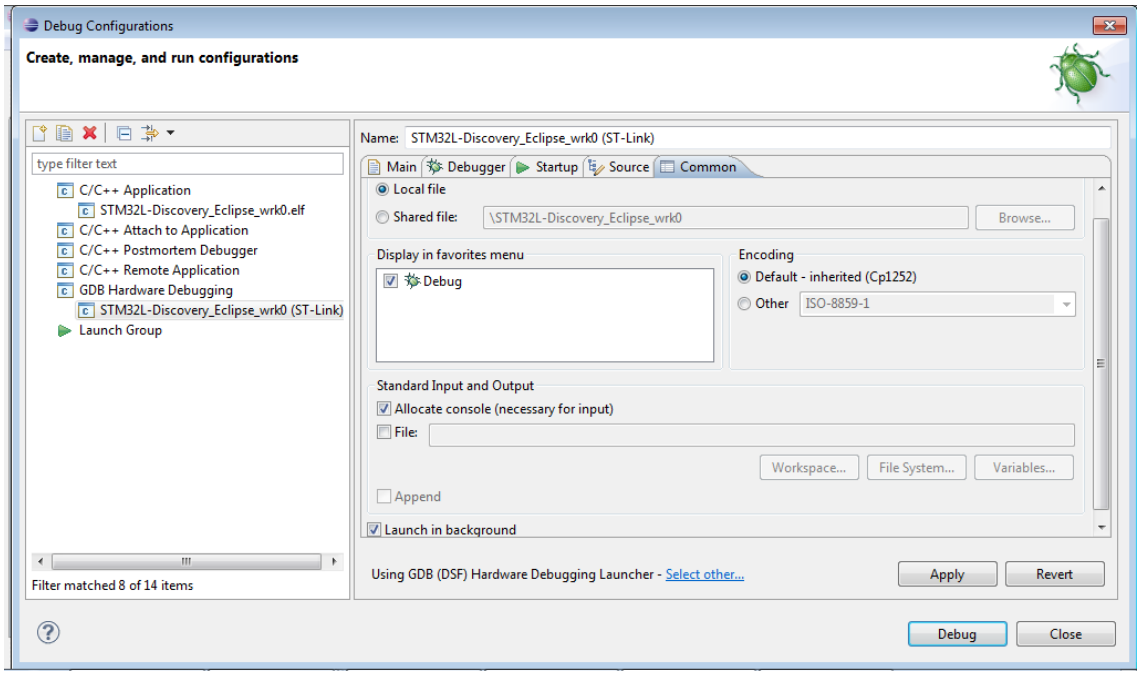


Figura 29

Una vez todo está configurado como en las imágenes, se hace clic en “Apply” para guardar la configuración.

Una vez seguidos estos pasos, se ha preparado la plantilla y configurado el proyecto, cuando se desee probar un programa en nuestra placa solo se deben seguir los siguientes pasos:

-Comprobar que el programa no tiene errores de compilación.

-Conectar la placa al ordenador mediante el cable USB.

-Desplegar el menú “External Tools” y lanzar el “ST-Link GDB Server (SWD)”.

-Desplegar el menú “Debug” y seleccionar “STM32L-Discovery_Eclipse_wrk0 (ST-Link)”.

Una vez ejecutados estos pasos, se procederá a comprobar el código y cargarlo en la placa. Una vez haya terminado de cargarse, bastará con pulsar el botón de “Play” para lanzar la aplicación, corriendo sobre la placa de desarrollo.

Debido a que no se está utilizando software diseñado por el fabricante, en muchas ocasiones, cuando se ha cargado un programa en la placa, se han modificado algunas líneas de código y se desea cargar otro programa después pulsar stop para detener el que está corriendo, el entorno de desarrollo no reconoce la placa. Para evitar esto, la recomendación que se hace es seguir siempre estos pasos cuando se desee cargar programas de forma consecutiva (situación muy común cuando se está testeando una aplicación y se le van realizando modificaciones sobre la marcha al observar los errores que van teniendo lugar durante la ejecución):

1-Se ha cargado un programa en placa y se está ejecutando o se ha ejecutado, si se desea cargar otro programa a continuación se debe pulsar el botón de “desconectar”, de forma que se desvincula la placa del entorno de desarrollo y ésta sigue ejecutando el programa.

2-Pulsar el botón de “debug”, al pulsar este botón, saltará un mensaje de error, pero la placa de desarrollo recibirá un intento de conexión y estará en condiciones de reconectar.

3-Desplegar el menú “External Tools” y lanzar el “ST-Link GDB Server (SWD)”.

4-Desplegar el menú “Debug” y seleccionar “STM32L-Discovery_Eclipse_wrk0 (ST-Link)”.

Una vez ejecutados estos pasos, el programa se cargará en la placa y se ejecutará.

También se ha observado que, al pulsar el botón “RESET” de la placa, se realiza una desconexión forzada del servidor de comunicaciones USB, por lo que si después se desea cargar un nuevo programa es recomendable desconectar y reconectar el USB para iniciar una conexión limpia.

4-CONEXIÓN ENTRE LA PLACA DE DESARROLLO Y LA DE E/S

4.1-HARDWARE Y CONEXIONADO EXISTENTES

4.1.1-LA PLACA DE E/S Y SUS PERIFERICOS

Anteriormente se han estudiado en detalle cada uno de los periféricos de la placa de E/S, en este apartado, se va a hacer un análisis de las señales que necesita cada periférico para funcionar correctamente según lo visto en el capitulo número 2.

-Display LCD: en este caso se necesitan 8 señales para el bus de datos (aunque en caso de escasez de líneas de comunicación se podría utilizar una interfaz de 4 bits de datos en vez de 8) y tres señales de control: Enable, Register Select y Read/Write.

-Relé: cuenta con una sola señal que lo activa y lo desactiva.

-Motor de Corriente Continua: para controlar el motor se utiliza una única señal, modulada mediante PWM.

Además, el sensor óptico que indica las interrupciones producidas por la hélice del motor activa otra señal cada vez que esto ocurre.

-Matriz de LEDs: son necesarias cuatro señales: los relojes de cada uno de los dos registros de desplazamiento, la señal de transmisión de datos y la señal de habilitación, que activa la salida serie de los 8 bits del registro de desplazamiento.

-Altavoz: se controla mediante una señal que tendrá el periodo y la frecuencia definidas por el programador.

-Sensor de Infrarrojos: al igual que el sensor óptico del motor, activa una señal cada vez que detecta un impulso en el rango infrarrojo.

-I2C: para llevar a cabo este tipo de comunicación serie síncrona se necesitan dos señales, el reloj y la señal de datos.

-Teclado: son necesarias 8 señales para poder controlarlo, 4 de entrada y 4 de salida, para poder controlar la matriz de 4x4.

4.1.2-CONEXIONES DE LA PLACA DE E/S

4.1.2.1-CONEXIONADO EXISTENTE

La conexión para comunicar la placa de E/S con el microcontrolador consta de un cable plano que termina en un conector de 40 pines, que están conectados a la placa de E/S tal y como se ve en la siguiente imagen:

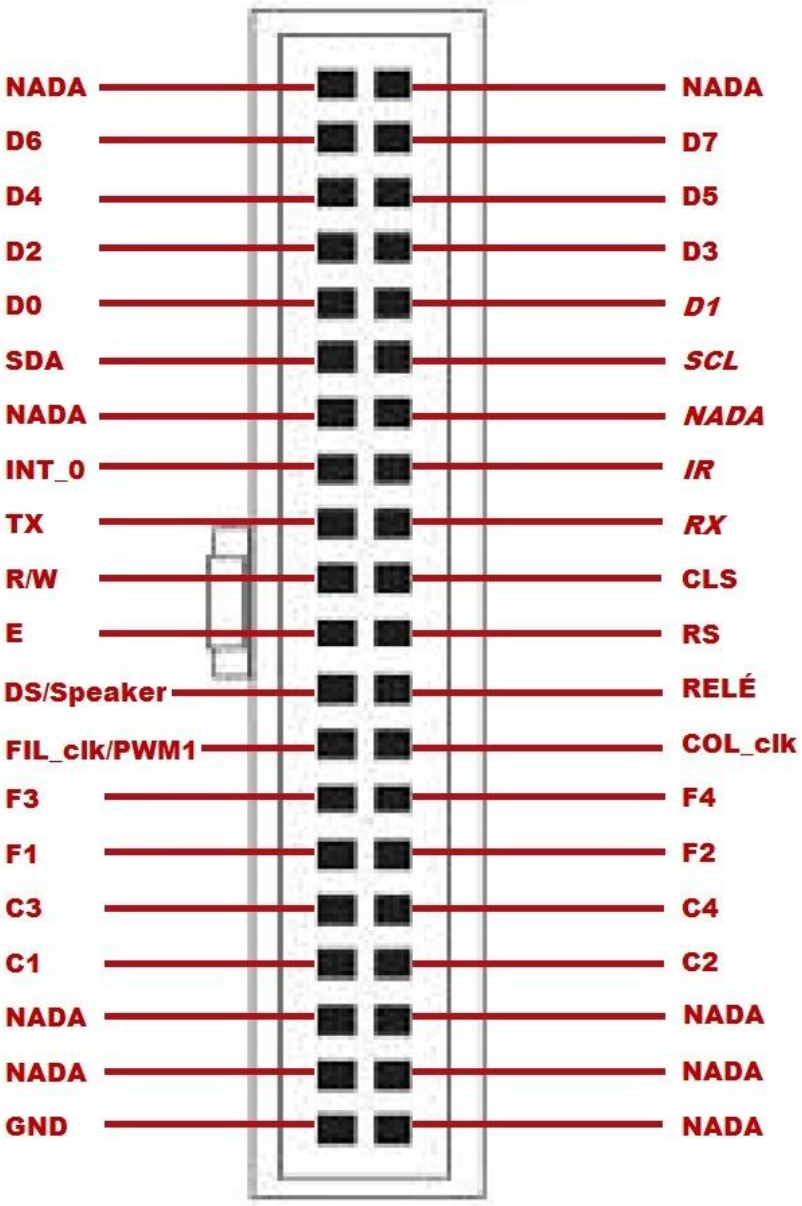


Figura 30

Teniendo en cuenta que esa es la única forma de conectar la placa de desarrollo a la de E/S, se debe tener en cuenta la configuración de este conector a la hora de diseñar la asignación de pines y la forma de realizar las conexiones.

A continuación se muestra la tabla existente para el ARM7 (figura1), que nos muestra la correlación entre los pines de la antigua placa de desarrollo y cada uno de los periféricos de E/S:

APLICACIÓN	Nº DE PATILLAS	ENTRADA			SALIDA		
		CONECTOR	ARM	VARIABLE	CONECTOR	ARM	VARIABLE
LCD (8 BITS)	11	4	P 0.23	BUSY	21	P 0.4	E
					22	P 0.5	RS
					19	P 0.6	R/W
					9	P 0.16	DATO 0
					10	P 0.17	DATO 1
					7	P 0.18	DATO 2
					8	P 0.19	DATO 3
					5	P 0.20	DATO 4
					6	P 0.21	DATO 5
					3	P 0.22	DATO 6
					4	P 0.23	DATO 7
TECLADO	8	29	P 1.20	FILA 1	33	P 1.16	COLUMNA 1
		30	P 1.21	FILA 2	34	P 1.17	COLUMNA 2
		27	P 1.22	FILA 3	31	P 1.18	COLUMNA 3
		28	P 1.23	FILA 4	32	P 1.19	COLUMNA 4
MOTOR	3	15	P 0.30	CUENTAVUELTAS	25	P 0.7	PWM 1
					26	P 0.8	PWM 2
CONTROL DE FASE	2	15	P 0.30	PASO POR 0	25	P 0.7	DISPARO
I2C	2			SDA	11	P 0.3	SDA
					12	P 0.2	SCL
ALTAVOZ	1				23	P 0.12	ONDA
MATRIZ DE LED'S	4				20	P 0.11	CLS
					23	P 0.12	DATO
					26	P 0.8	CLK 1
					25	P 0.7	CLK 2
RELE	1				24	P 0.13	RELE
INFRARROJOS	1	16	P 0.15	RECEPTOR			
PUERTO SERIE	2	18	P 0.1	Rx	17	P 0.0	Tx
XPORT	1				36	P 0.24	RESET

Figura 31

La tabla anterior servirá de modelo para diseñar la correspondencia de conexiones para la nueva placa.

4.1.2.2-INCOMPATIBILIDADES

Como se puede observar en la Figura 4.1, existen tres pines asignados a más de una señal (en realidad son cuatro porque la señal de COL_clk comparte hilo con la señal PWM2, pero ya que la segunda no se ha utilizado en este proyecto, no se tiene la incluye en este caso). Esto provoca que algunos de los periféricos de la placa de E/S no sean compatibles entre sí, en concreto sucede con los siguientes:

-Matriz de LEDs y altavoz: la señal que activa el altavoz comparte hilo con la señal de datos de la matriz.

-Matriz de LEDs y motor de CC: la señal que activa el motor comparte cableado con la señal de reloj del registro de desplazamiento que controla las filas de la matriz.

-Control de fase y relé, ya que comparten la señal de activación.

Debido a estas incompatibilidades, existe la posibilidad de que cuando se trabaje con un periférico se observen comportamientos anómalos en aquel con el que comparte una señal. Con un uso normal de la placa de E/S no debería observarse nada extraño porque las señales de la matriz de LEDs son demasiado cortas para provocar una reacción en el altavoz o el motor, y en el caso inverso las del altavoz o el motor no deberían afectar a la matriz porque son necesarias las cuatro señales de esta última para que se encienda algún LED.

En caso de que el usuario lo desee es posible quitar los jumpers del altavoz y el motor de corriente continua cuando no se vayan a utilizar estos periféricos. Sin embargo, solo se han experimentado comportamientos anómalos debido a este cableado compartido en los primeros test del proyecto, cuando todavía no se habían ajustado bien las frecuencias de las señales.

4.2-SELECCIÓN DEL NUEVO CONEXIONADO

4.2.1-INTERFACES DE E/S DEL STM32L-Discovery (GPIO)

La interfaz GPIO (General Purpose Input-Output) es una de las herramientas más importantes de la placa de desarrollo, es la principal fuente de comunicaciones con el exterior, por lo que todo el intercambio de información con cualquier periférico, empotrado en la placa o externo, se lleva a cabo a través de este canal.

Cada uno de los pines de cada puerto del GPIO cuenta con una serie de registros asignados de memoria, donde se guarda su configuración, su último valor medido, si esta asociado a alguna función, etc.

La configuración de un puerto GPIO es muy completa, y permite, entre otras cosas:

-Modo de entrada flotante, pull-up, pull-down o analógica.

-Modo de salida open-drain o push-pull y pull-up o pull-down.

-Selección de velocidades de transmisión individualizada.

-Cambio de modo en solo dos ciclos de reloj.

-Bloqueo de pines.

-Generación de interrupciones externas mediante la aplicación EXTI.

-Alternate Functions, variedad de funciones alternativas para cada uno de los puertos.

Todas las posibles configuraciones de entrada y salida están disponibles gracias al diseño de cada uno de los pines del GPIO, que se puede ver en la imagen a continuación:

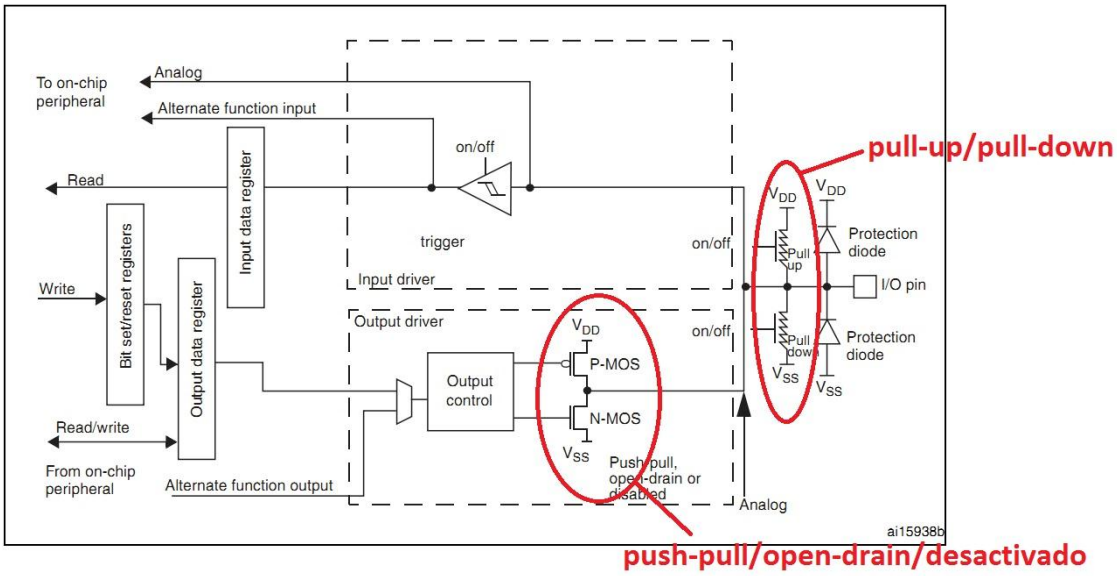


Figura 32

En la imagen se puede distinguir el pin del GPIO, a continuación un sistema de protección contra sobretensiones, dos resistencias variables para configurar el modo pull-up o pull down y un nodo del cual salen los hilos correspondientes a los modos de entrada y salida.

El hilo de entradas, en la mitad superior de la imagen se subdivide en los correspondientes a los modos de entrada normal, analógica y Alternate Function. Por otra parte, el hilo correspondiente al modo salida cuenta con un nodo en el que aparece la salida analógica del DAC, y otro que cuenta con dos transistores colocados en a modo de configuración C-MOS, que regulan si la salida está en modo push-pull, open-drain o desactivada, a continuación hay un pequeño controlador de salidas, cuya entrada multiplexa las líneas de Alternate Function y la salida normal.

Los pines accesibles con los que cuenta la placa de desarrollo tienen todos una configuración como la que se ha explicado en el párrafo anterior y se encuentran situados a ambos lados de la misma. Estos pines se utilizarán para comunicar el STM32L-Discovery con la placa de E/S. A continuación se pueden ver en una imagen:

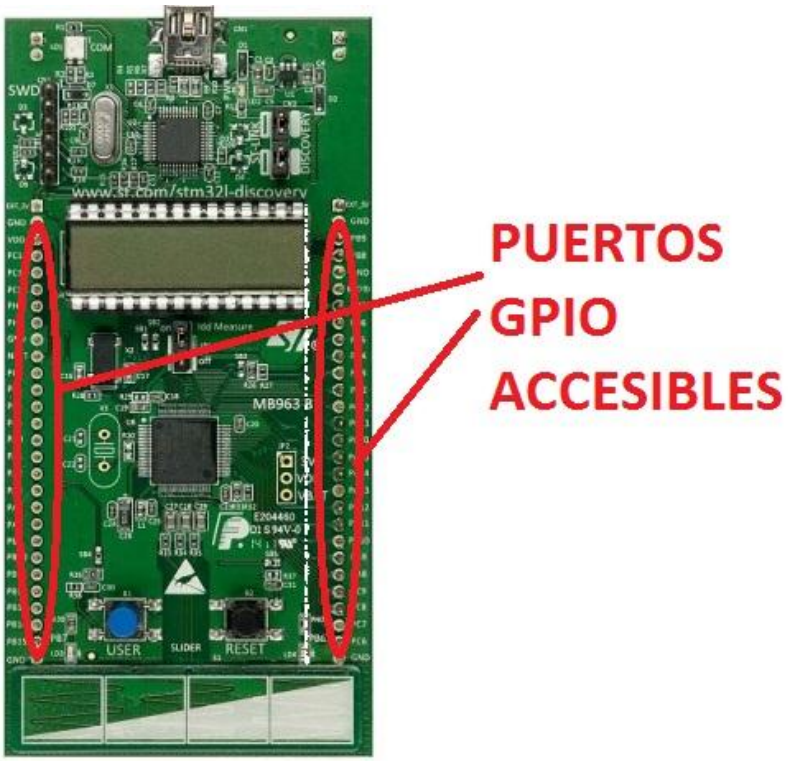


Figura 33

4.2.2-ALTERNATE FUNCTIONS

Las Alternate Functions son otra de las cualidades que hace del Cortex-M3 un microcontrolador tremendamente versátil.

Aunque cada pin del GPIO tiene una funcionalidad pre-asignada al arrancar la placa de desarrollo, mediante multiplexación se consigue que todos ellos tengan más funciones disponibles:

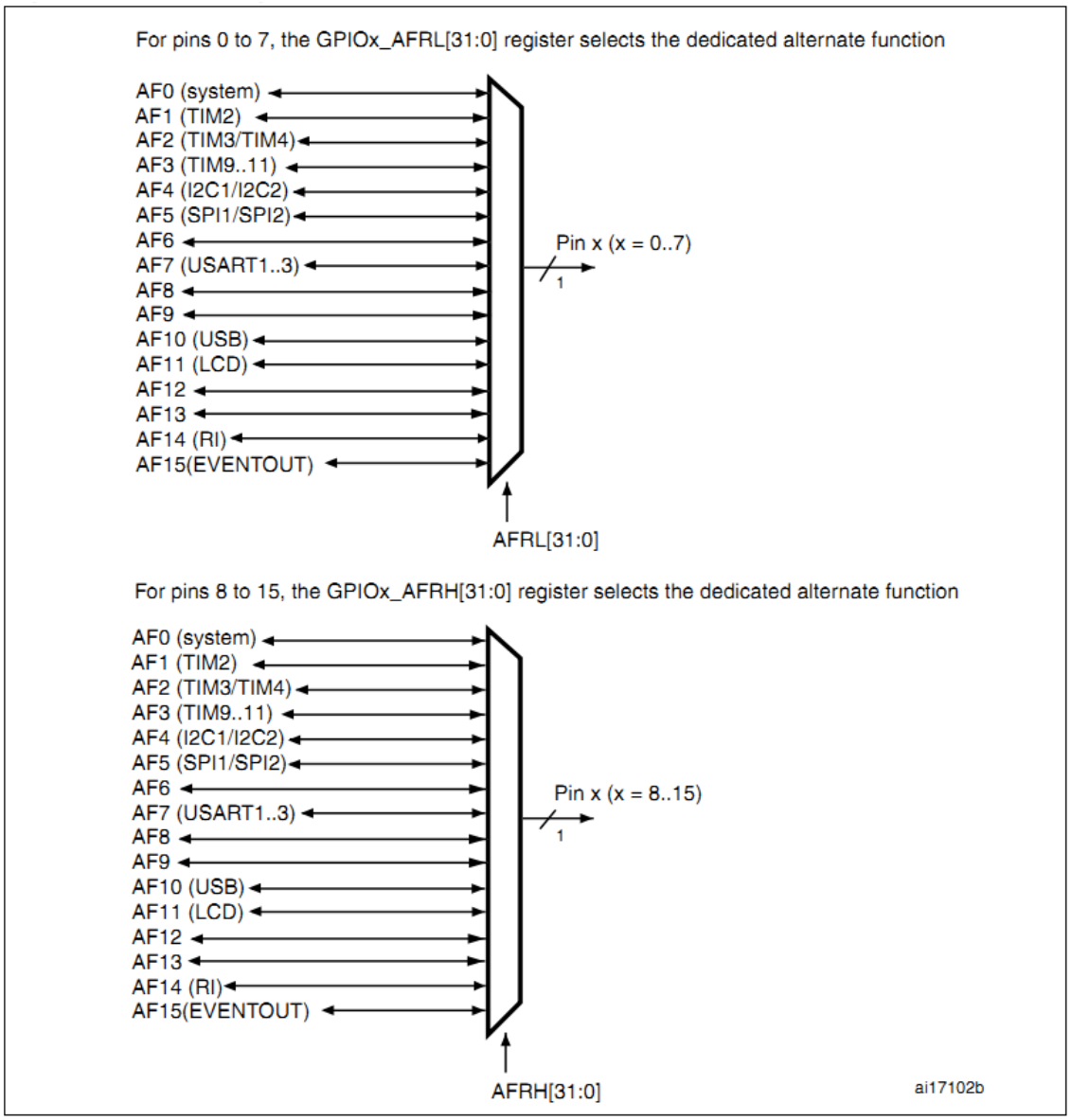


Figura 34

Esta configuración permite una mayor libertad a la hora de elegir los pines que controlarán cada periférico de la placa de E/S. Para llevar a cabo la asignación de los mismos, se debe tener en cuenta la siguiente tabla, que detalla las diferentes funcionalidades que puede adquirir cada uno de los pines del GPIO del STM32L-Discovery:

MCU pin			Board function										
Main function	Alternate functions	LQFP 64 pin num.	LCD glass	Linear Touch Sensor	Push button	I_{DD}	LED	SWD	OSC	Free I/O	Power supply	P1	P2
-	-	-									EXT_ 3V	1	
-	-										EXT_ 5V		1
BOOT0	-	60											6
NRST	-	7						NRST				10	
PA0	WKUP1/USART2_CTS/ ADC_IN0/TIM2_CH1_ETR /COMP1_INP	14			PA0	WAKE UP						15	
PA1	USART2_RTS/ADC_IN1/ TIM2_CH2/LCD_SEG0/ COMP1_INP	15	SEG0									16	
PA2	USART2_TX/ADC_IN2/ TIM2_CH3/TIM9_CH1/ LCD_SEG1/COMP1_INP	16	SEG1									17	
PA3	USART2_RX/ADC_IN3/ TIM2_CH4/TIM9_CH2/ LCD_SEG2/COMP1_INP	17	SEG2									18	
PA4	SPI1_NSS/USART2_CK/ ADC_IN4/DAC_OUT1/ COMP1_INP	20				Measure ment						19	
PA5	SPI1_SCK/ADC_IN5/ DAC_OUT2/ TIM2_CH1_ETR/COMP1_ INP	21								X		20	
PA6	SPI1_MISO/ADC_IN6/ TIM3_CH1/TIM1_BKIN/ LCD_SEG3/TIM10_CH1/ COMP1_INP	22		PA6									
PA7	SPI1_MOSI/ADC_IN7/ TIM3_CH2/TIM1_CH1N /LCD_SEG4/TIM11_CH1/ COMP1_INP	23		PA7									
PA8	USART1_CK/MCO/ LCD_COM0	41	COM0										23
PA9	USART1_TX/LCD_COM1	42	COM1										22
PA10	USART1_RX/LCD_COM2	43	COM2										21
PA11	USART1_CTS/USBDM/ SPI1_MISO	44								X			20
PA12	USART1_RTS/USBDP/ SPI1_MOSI	45								X			19
JTMS/ SWDIO	PA13	46						SWD IO					18

Figura 35

MCU pin			Board function										
Main function	Alternate functions	LQFP 64 pin num.	LCD glass	Linear Touch Sensor	Push button	I_{DD}	LED	SWD	OSC	Free I/O	Power supply	P1	P2
JTCK/ SWCLK	PA14	49						SW CLK					17
JTDI	TIM2_CH1_ETR/PA15/ SPI1_NSS/LCD_SEG17	50	SEG12										16
PB0	ADC_IN8/TIM3_CH3/ LCD_SEG5/COMP1_INP/ VREF_OUT	26		PB0									
PB1	ADC_IN9/TIM3_CH4/ LCD_SEG6/COMP1_INP/ VREF_OUT	27		PB1									
PB2/BOOT1	-	28										21	
JTDO	TIM2_CH2/PB3/TRACES WO/SPI1_SCK/COMP2_I NM/LCD_SEG7	55	SEG3					SWO					11
JNTRST	TIM3_CH1/PB4/SPI1_MIS O/COMP2_INP/LCD_SEG 8	56	SEG4										10
PB5	I2C1_SMBAI/TIM3_CH2/ SPI1_MOSI/COMP2_INP/ LCD_SEG9	57	SEG5										9
PB6	I2C1_SCL/TIM4_CH1/ USART1_TX/LCD_SEG8	58					Blue						8
PB7	I2C1_SDA/TIM4_CH2/ USART1_RX/PVD_IN	59					Green						7
PB8	TIM4_CH3/I2C1_SCL/ LCD_SEG16/TIM10_CH1	61	SEG13										4
PB9	TIM4_CH4/I2C1_SDA/ LCD_COM3/TIM11_CH1	62	COM3										3
PB10	I2C2_SCL/USART3_TX/ TIM2_CH3/LCD_SEG10	29	SEG6									22	
PB11	I2C2_SDA/USART3_RX/ TIM2_CH4/LCD_SEG11	30	SEG7									23	
PB12	SPI2_NSS/I2C2_SMBA/ USART3_CK/LCD_SEG1 2/ADC_IN18/COMP1_INP / TIM10_CH1	33	SEG8									24	
PB13	SPI2_SCK/USART3_CTS/ LCD_SEG13/ADC_IN19/ COMP1_INP/TIM9_CH1	34	SEG9									25	
PB14	SPI2_MISO/USART3_RT S/LCD_SEG14/ADC_IN20 / COMP1_INP/TIM9_CH2	35	SEG10									26	
PB15	SPI2_MOSI/TIM1_CH3N/ LCD_SEG15/ADC_IN21/ COMP1_INP/TIM11_CH1/ RTC_50_60Hz	36	SEG11									27	
PC0	ADC_IN10/LCD_SEG18/ COMP1_INP	8	SEG14									11	
PC1	ADC_IN11/LCD_SEG19/ COMP1_INP	9	SEG15									12	
PC2	ADC_IN12/LCD_SEG20/ COMP1_INP	10	SEG16									13	

Figura 36

MCU pin			Board function										
Main function	Alternate functions	LQFP 64 pin num.	LCD glass	Linear Touch Sensor	Push button	I_{DD}	LED	SWD	OSC	Free I/O	Power supply	P1	P2
PC3	ADC_IN13/LCD_SEG21/ COMP1_INP	11	SEG17									14	
PC4	ADC_IN14/LCD_SEG22/ COMP1_INP	24		PC4									
PC5	ADC_IN15/LCD_SEG23/ COMP1_INP	25		PC5									
PC6	TIM3_CH1/LCD_SEG24	37	SEG18										27
PC7	TIM3_CH2/LCD_SEG25	38	SEG19										26
PC8	TIM3_CH3/LCD_SEG26	39	SEG20										25
PC9	TIM3_CH4/LCD_SEG27	40	SEG21										24
PC10	USART3_TX/LCD_SEG28 /LCD_SEG40/LCD_COM4	51	SEG22										15
PC11	USART3_RX/LCD_SEG2 9/LCD_SEG41/ LCD_COM5	52	SEG23										14
PC12	USART3_CK/LCD_SEG3 0/LCD_SEG42/ LCD_COM6	53								X			13
PC13	RTC_AF1/WKUP2	2				CNT_ EN						4	
PC14	OSC32_IN	3							OSC32 _IN			5	
PC15	OSC32_OUT	4							OSC32 _OUT			6	
PD2	TIM3_ETR/LCD_SEG31/ LCD_SEG43/LCD_COM7	54								X			12
OSC_IN	PH0	5							OSC_ IN			7	
OSC_OUT	PH1	6							OSC_ OUT			8	
-	-	-									GND	2	2
-	-	-									GND	9	5
-	-	-									GND	28	28
-	-	-									VDD	3	

Figura 37

4.2.3-ASIGNACIÓN DE CONEXIONES

La asignación de los pines y su correspondencia será similar a la que se puede ver en la tabla correspondiente al ARM7 (Figura 4.2), solo es necesario consultar las funciones por defecto de cada uno de los pines del STM32L-Discovery (figuras 6, 7 y 8) para decidir qué pines podrán cumplir cada función, y asignarlos en consecuencia.

Los primeros que se deben asignar son aquellos pines que deben cumplir funciones específicas, como la comunicación I2C, temporización mediante timers…

-I2C: Se asigna el pin PB8 para realizar la función de SCL y el PB9 para el SDA, ambos pertenecientes al controlador I2C1 del STM32L-Discovery.

-Motor de CC: Se elige el pin PB6, porque puede actuar como salida de señal del canal 1 del timer TIM4, que se deberá configurar en modo PWM.

Para el sensor óptico se selecciona el pin PC10, que no tiene ninguna AF que se considere necesaria.

-Altavoz: se le asigna el pin PC9, que puede asignarse a la salida del canal 4 del TIM3, por el que se debe enviar la señal con la frecuencia deseada para conseguir el tono correspondiente.

-Puerto serie: los pines elegidos son el PA9 para la señal Rx y el PA10 para la Tx, ambos pertenecientes al controlador SCI1 de la placa de desarrollo.

-Relé: se asigna el pin PD2 para activarlo/desactivarlo.

-Matriz de LEDs: debe compartir del PB6 con el motor de CC debido al cableado existente en la placa de E/S, pero en este caso toma la función de

reloj del registro de desplazamiento de las columnas. El resto de pines asignados son: PC11 para el reloj del registro de desplazamiento de las filas, PC8 para la señal de CLS y PC9 para la señal de Dato, el cual se comparte con el altavoz.

-Sensor de infrarrojos: el pin seleccionado para ocuparse de las interrupciones producidas por este periféricos es el PB5.

-Teclado: se asignan los pines PC0, PC1, PC2 y PC3 a las filas, y los PA1, PA2, PA3 y PA5 a las columnas.

Por lo tanto, la nueva tabla de correlacion de los pines de la placa de desarrollo y los periféricos de la placa de E/S es la siguiente:

PERIFÉRICO	Nº PINES	ENTRADA MICRO		SALIDA MICRO		COMENTARIOS
		SEÑAL STM32l	SEÑAL PLACA E/S	SEÑAL STM32l	SEÑAL PLACA E/S	
MÓDULO LCD	11			PA8	E	
				PC6	RS	
				PC7	R/W	
				PA11	Dato 0	
				PA12	Dato 1	
				PB10	Dato 2	
				PB11	Dato 3	
				PB12	Dato 4	
				PB13	Dato 5	
				PB14	Dato 6	
		PB15	BUSY	PB15	Dato 7	
RELÉ	1			PD2	0/1 => On/Off	0/1 => Activado/Reposo
CONTROL DE FASE	2	PC10	INT paso por 0	PD2	Triac On	
MOTOR CC	3	PC10	INT giro motor	PC11	PWM 1	Motor en placa auxiliar
				PC12	PWM 2	(No hay motor)
MATRIZ DE LED'S	4			PC12	SH_CP Columnas	M. de LEDs: Columnas
				PC11	SH_CP Filas	M. de LEDs: Filas

				PC8	CLS (ST_CP)	M. de LEDs: Común
				PC9	Dato (DS)	M. de LEDs: Común
ALTAVOZ	1			PC9	Salida altavoz	Incluye filtro paso bajo
INFRARROJOS IR	1	PB5	INT de IR			Mando a distancia por IR
I 2C	2			PB8	SCL	EEPROM & Pot. Digital, I2C
				PB9	SDA	
PUERTO SERIE	2	PA9	Rx serie	PA10	Tx serie	Transmisión Serie RS 232
TECLADO	8	PC0	Fila 1	PA1	Columna 1	
		PC1	Fila 2	PA2	Columna 2	
		PC2	Fila 3	PA3	Columna 3	
		PC3	Fila 4	PA5	Columna 4	
NOTAS	Las señales con doble función no permiten el control simultáneo de los periféricos implicados:					
	PC9: M. LED's/Altavoz, PC11: M. LED's / Motor CC, PD2: Triac/Relé, PC10: INT de Triac/Motor CC					

Figura 38

4.3-HARDWARE DE INTERCONEXIÓN

Siguiendo la tabla del apartado anterior, se deben construir una serie placas para adaptar las conexiones de los pines del STM32L-Discovery al conector de 40 pines, según el croquis que aparece a continuación:

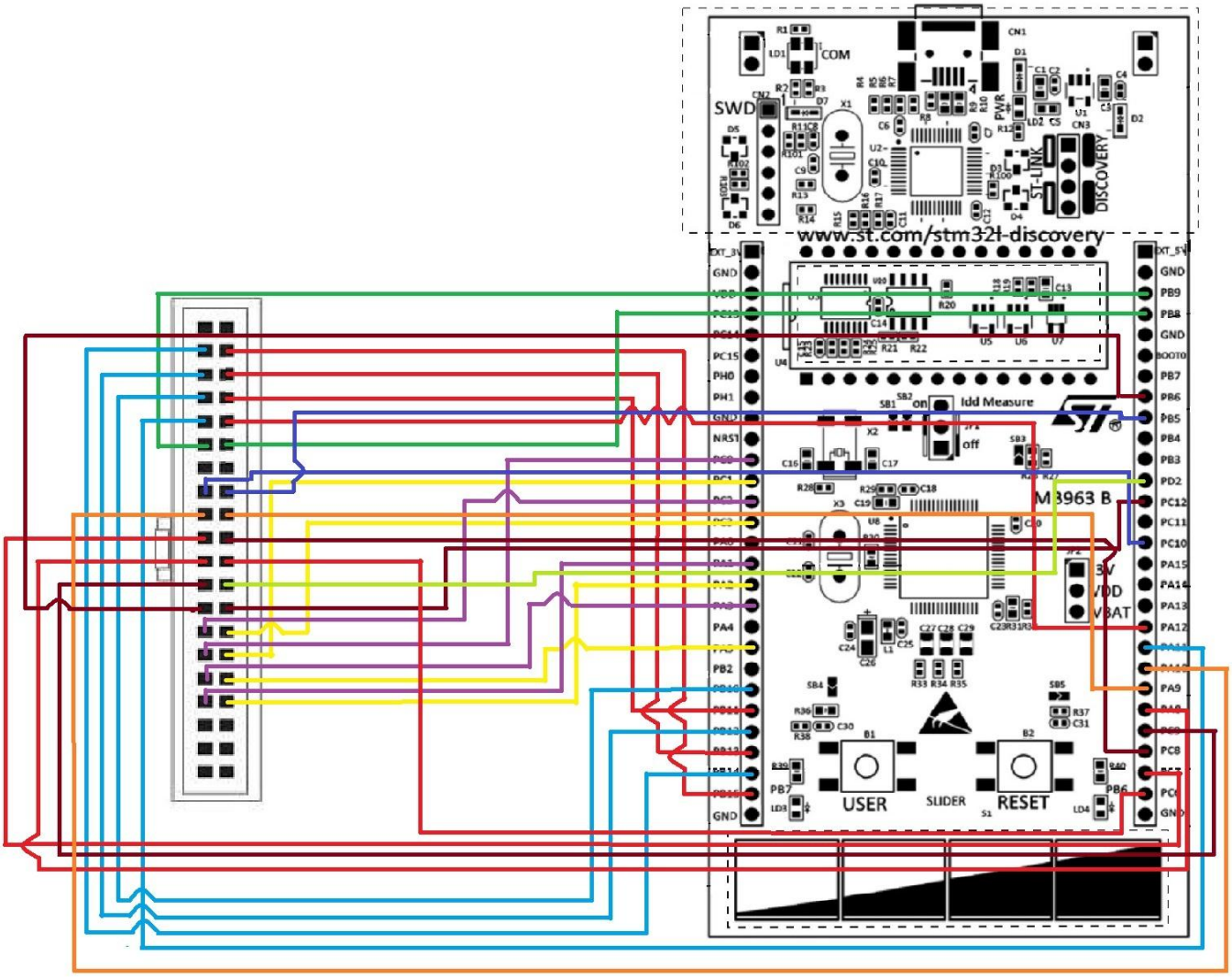


Figura 39

Para poder desarrollar el proyecto, se ha construido un prototipo de esta placa de interconexión, utilizando varios conectores, una placa perforada, cables y un soldador.

Aunque esta placa ha servido para desarrollar el proyecto, se recomienda que cuando se deseen construir las placas para el uso en el laboratorio se empleen circuitos impresos, ya que son mucho más resistentes que los materiales empleados.

En las imágenes que aparecen a continuación puede verse la placa construida:

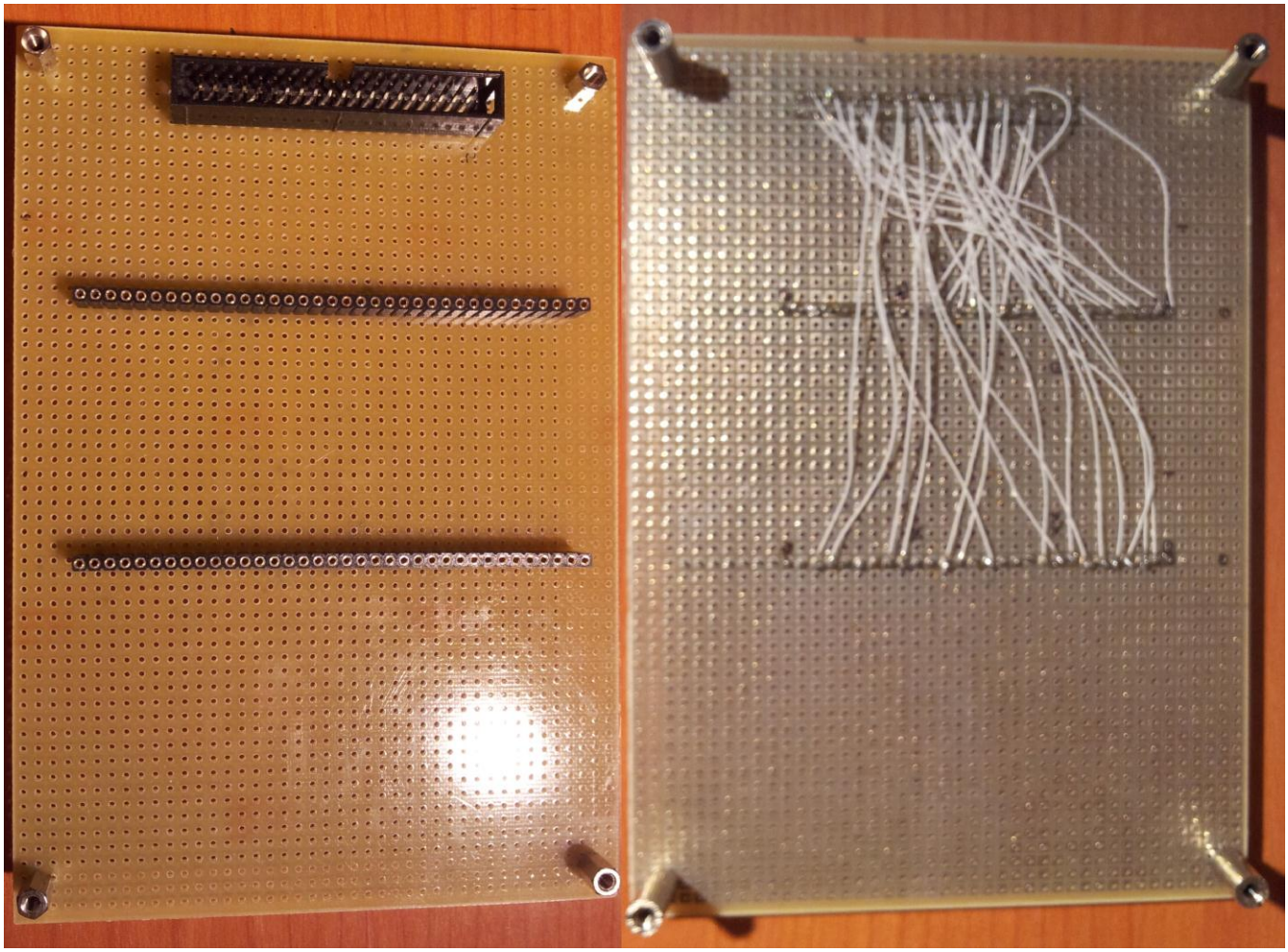


Figura 40

5-LIBRERÍAS PARA EL MANEJO DE LOS PERIFÉRICOS

5.1-INTRODUCCIÓN A LAS LIBRERÍAS

5.1.1-UTILIDAD DE UNA LIBRERÍA DE FUNCIONES

Una librería de funciones es una recopilación de las mismas que permite automatizar una serie de procesos, cada función es un pequeño programa que lleva a cabo una acción, como puede ser grabar algo en la memoria o presentar un dato por pantalla.

En este proyecto ya se está utilizando una librería básica de funciones, que permite al usuario configurar e interactuar con cada uno de los periféricos del STM32L-Discovery. Y el objetivo de esta parte del proyecto es desarrollar una librería similar (aunque a menor escala), que permita controlar los periféricos de la placa de E/S.

Ya se ha hablado en capítulos anteriores de cada uno de estos periféricos, y gracias a lo completa que es la librería básica destinada a controlar el STM32L-Discovery, es perfectamente posible controlarlos todos ellos mediante las funciones incluidas en ella. Sin embargo, existen varias razones que apoyan la creación de una librería de funciones propias para la placa de E/S:

-Permite que el código escrito sea más sencillo de comprender.

-La experiencia adquirida con las librerías del STM32L-Discovery indica que, cuanto más alto es el nivel de programación, más sencillo es para el programador desarrollar el software, aunque esto sea algunas veces en detrimento del rendimiento de las aplicaciones. Por ejemplo: es más sencillo realizar una llamada a una función de la siguiente forma:

“escribeCadena(‘hola’);” que configurar cada uno de los registros de comunicación del GPIO del microcontrolador, iniciar una comunicación con el LCD, indicarle que se le envía un comando determinado que corresponde a la letra ‘h’, y repetir la operación hasta que no queden más letras.

-Permite al alumno centrarse en la comprensión de lo que pretende explicarle el profesor, sin tener que revisar continuamente el programa en busca de pequeños errores de programación que impiden el correcto funcionamiento del dispositivo.

-Hace posible que el alumno utilice la placa de E/S desde el principio del curso.

En resumen, una librería de funciones hace la programación más cómoda para el usuario, y le permite centrarse en los problemas complejos al automatizar un cierto número de acciones de uso habitual.

5.2-LAS LIBRERÍAS

A continuación se describe la funcionalidad que se busca en cada uno de los periféricos y se detalla el funcionamiento y el objetivo de cada una de las funciones que se han implementado.

Estas librerías se incluyen en la plantilla que se proporciona con la documentación del proyecto, a partir de la cual se pueden escribir proyectos utilizando todos los periféricos de la placa de E/S.

5.2.1-DISPLAY LCD:

5.2.1.1-Objetivos:

El objetivo que se busca para este periférico es implementar una serie de funciones que automaticen las comunicaciones entre ambos controladores. Para ello, es necesario conocer el funcionamiento del display, y los comandos necesarios para controlarlo. Se desea poder escribir en el display tanto cadenas de texto como números, colocar los mensajes en cualquier lugar del display y poder borrarlo de forma sencilla.

También se debe inicializar automáticamente el display, y poder presentar cadenas de caracteres y números de la forma y en la posición del display que el usuario precise.

5.2.1.2-Funciones:

Las funciones que se implementan para satisfacer estas necesidades son las siguientes:

-enviar_LCD(int com, int select):

Esta función es la base de la comunicación entre ambos controladores, cuenta con dos entradas, un entero que contiene el mensaje que se desea enviar al display y otro que, en función de su valor (0 o 1), indicará si se está enviando un comando o un carácter a representar.

Su funcionamiento es el siguiente:

-Configura el GPIO de forma que los pines correspondientes al display estén en modo salida pull-up.

-Si se desea enviar un comando, pone a nivel bajo la señal RS, y en caso de que se desee enviar un carácter la pone a nivel alto.

-Pone a nivel bajo la señal de R/W para indicar que se va a enviar un mensaje.

-Inicializa el valor de los ocho pines de Dato a 0.

-Activa o desactiva cada uno de los pines de Dato en función del valor de los 8 bits menos significativos de la variable “com”, que lleva codificado el comando o carácter.

-Impone un pequeño retardo mediante la función “retardo_LED” de la librería de la matriz de LEDS para que se estabilicen los valores de las ocho señales de Dato.

-Una vez está preparada la señal que se va a enviar, se genera un pulso en la señal de Enable, para que el controlador del display lea el mensaje.

Esta función es la que hace posible que las tres funciones que aparecen a continuación envíen números enteros, caracteres y comandos al LCD.

-escribeComando_LCD(int com):

Esta función permite enviar comandos al LCD según la tabla que aparece en el manual del display.

Cuando se llama a esta función, ejecuta enviar_LCD forzando a 0 la variable “select” y escribiendo el comando indicado en la variable “com”.

-escribeLetra_LCD(int com)

Esta función permite enviar caracteres al LCD según la tabla que aparece en el manual del display, para que los presente por pantalla. La codificación de los caracteres coincide con la del código ASCII, por lo que no es necesario aplicar ninguna modificación a la variable de entrada.

Cuando se llama a esta función, ejecuta enviar_LCD forzando a 1 la variable “select” y escribiendo el carácter indicado en la variable “com”.

-escribeDigito_LCD(int com):

Esta función permite enviar números (de un solo digito) contenidos en una variable de tipo entero al LCD, para que los presente por pantalla. Para hacer coincidir la variable entera con su representación según el código ASCII, se le suma 48 a la variable “com”, ya que este es el valor del carácter que representa al numero 0.

Cuando se llama a esta función, ejecuta enviar_LCD forzando a 1 la variable “select” y escribiendo el carácter indicado en la variable “com”, una vez modificada.

-espera_LCD():

El objetivo de esta función es iniciar un lazo de espera que durará mientras el LCD mantenga a nivel alto la señal de “Busy”, que se activa realiza alguna tarea. Esta función permite que el STM32L-Discovery no envíe ninguna señal al controlador del display mientras éste esté trabajando, con lo cual se evitan problemas de comunicación.

Su funcionamiento es el siguiente:

-Configura el GPIO de forma que los pines correspondientes al Enable, RS y R/W estén en modo salida pull-up, y los correspondientes a las ocho

señales de Dato en modo entrada pull-up (la señal de Busy está conectada al pin asignado a Dato7, pero se ponen todos en el mismo modo por homogeneidad y para evitar posibles envíos de datos accidentales).

-Desactiva RS y activa R/W para indicar lectura.

-Activa la señal de Enable mientras lee Dato7, y la desactiva al finalizar la lectura.

-Estos dos últimos pasos se repiten indefinidamente hasta que el valor de Dato7 (Busy) sea igual a 0.

-iniciaLCD():

Esta función utiliza las anteriormente definidas para inicializar el display LCD, dejándolo listo para trabajar, no tiene parámetros de entrada ni de salida, ya que es una secuencia de comandos que arrancan el display y que siempre serán los mismos.

La secuencia es la siguiente:

-Se envía tres veces según los intervalos indicados en el manual una señal que tenga los datos 4 y 5 a nivel alto y los datos 6 y 7 a nivel bajo (los otros cuatro no son relevantes).

-Se envía el comando SET, configurando el controlador para manejar un display de dos líneas y con caracteres de 5x7 puntos.

-Se enciende el display con el comando de ON, configurando el cursor sin parpadeo.

-Se envía el comando ENTRY MODE SET, configurando el sentido de desplazamiento del cursor y su funcionamiento.

Entre estos comandos siempre se llama a la función espera_LCD, para no enviar uno antes de que el anterior haya terminado de ejecutarse.

-borra_LCD():

Esta función limpia la pantalla del display, dejándola en blanco. Para ello espera a que la señal de “Busy” este desactivada y envía el comando de DELETE DISPLAY.

-escribeCadena_LCD(char *cadena):

Esta función permite escribir cadenas de caracteres en la pantalla, a partir de la posición en la que se encuentre en ese momento el cursor. Para ello espera a que la señal de “Busy” esté desactivada y llama a la función escribeLetra_LCD tantas veces como caracteres tenga la cadena, avanzando por ella hasta llegar al final.

La función no devuelve ningún parámetro, pero si recibe un puntero al array de variables de tipo carácter que contiene la cadena que se desea mostrar por pantalla.

-escribeEntero_LCD(int entero):

Esta función tiene un objetivo similar a la anterior, pero en este caso permite escribir en la pantalla números a partir de una variable de tipo integer, lo que resultara muy útil a la hora de proporcionar información al usuario acerca de alguna variable del programa.

Al igual que la función anterior, no devuelve ningún parámetro, primero llama a esperar_LCD, y después representa los caracteres correspondientes. A

la hora de enviar los caracteres al display, en este caso es necesario realizar una pequeña operación para obtener los dígitos correspondientes:

-Se asume que el número más grande capaz de ser contenido en una variable entera (16 bits) no tendrá más de 5 dígitos.

-Se va calculando y almacenando cada uno de los dígitos del número calculando el cociente de la división entera entre varias potencias de 10.

-posicionCursor_LCD(int x, int y):

Esta función no devuelve ningún parámetro, y cuenta con dos variables de entrada, que le indican la fila y la columna en la que se desea colocar el cursor (“x” e “y”).

La función comprueba en que fila se desea colocar el cursor (variable “x”) y ubica el mismo en la primera posición de dicha fila. A continuación, avanza hacia la derecha tantas veces como le indique la variable “y”.

-escribeCadenaPos_LCD(char *cadena, int x, int y):

Esta función escribe una cadena de caracteres en la posición deseada del display, haciendo uso de las funciones posicionCursor_LCD y escribeCadena_LCD.

-escribeEnteroPos_LCD(int entero, int x, int y):

Esta función escribe un numero entero en la posición deseada del display, haciendo uso de las funciones posicionCursor_LCD y escribeEntero_LCD.

-escribeCadenaScrollD(char *cadena, int fila):

El objetivo de esta función es proporcionar al display la capacidad de hacer un scroll del texto, de forma que dé la impresión de que el mismo “se escapa” de la pantalla.

No devuelve ningún parámetro y recibe dos variables, la cadena a representar y la fila en la que se desea ver el scroll.

Su funcionamiento se basa en una bucle “for” que va incrementando la posición del cursor a partir de la cual se representa la cadena en el display hasta que llega al final de la fila

-escribeCadenaScrollI(char *cadena, int fila):

El objetivo de esta función es proporcionar al display la capacidad de hacer un scroll del texto, de forma que dé la impresión de que el mismo “entra” en la pantalla desde la parte derecha de la misma.

No devuelve ningún parámetro y recibe dos variables, la cadena a representar y la fila en la que se desea ver el scroll.

Su funcionamiento se basa en un bucle “for” que va disminuyendo el valor de la posición del cursor a partir de la cual se representa la cadena en el display hasta que llega al final de la fila.

5.2.2-PWM, MOTOR DE CC Y SENSOR ÓPTICO:

5.2.2.1-Objetivos:

En este caso lo que se necesita es generar una señal con modulación PWM cuyo ciclo de trabajo se pueda controlar, y generar una serie de interrupciones cada vez que la hélice del motor pase por el sensor óptico, además de buscar la forma de contar estas interrupciones para calcular las rpm del motor.

5.2.2.2-Funciones:

Para controlar este periférico se han desarrollado las siguientes funciones:

-iniciaPWM():

Esta función configura el timer y el pin necesario para enviar la señal PWM al motor de corriente continua:

-Configura el pin PB6 en modo output, y lo asocia al canal 1 del TIM4 mediante una “Alternate Function”.

-Configura el TIM4 en modo Output Compare, activando el modo PWM.
Al ser una función de configuración.

-ArrancarPWM(int ciclo):

Esta función inicia el TIM4 en modo PWM con un ciclo de trabajo igual al valor de la variable de entrada. Por lo tanto, la variable de entrada debe tener un valor entre 0 y 100.

La función llama a iniciaPWM y posteriormente configura un ciclo de trabajo correspondiente al parámetro que se le ha pasado, e inicia el timer.

-ParарPWM():

El objetivo de esta función es dar la posibilidad al programador de detener el motor en caso de que lo desee.

Para ello se desactiva el timer TIM4 y, para más seguridad se configura el pin PB6 en modo INPUT, para que no existan activaciones accidentales del motor.

-EXTI15_10_Config():

El nombre de esta función corresponde al nombre genérico de la función de configuración de las interrupciones externas (EXTI) de los canales 10 al 15.

Al ser una función de configuración, carece de variables de entrada o salida.

El objetivo que se persigue es configurar el pin PC10 como una interrupción externa generada por el sensor óptico que nos sirve para medir las rpm del motor de corriente continua.

Para ello se configura el GPIO poniendo el pin PC10 en modo input, no-pull, se configura la línea 10 del EXTI como interrupción externa por flanco de bajada (en principio da igual que sea flanco de subida o de bajada, pero se elige de bajada para que la interrupción se produzca cuando el motor ya ha completado la vuelta) y se habilitan las interrupciones de los canales 10 al 15, con prioridad máxima.

-iniciaSensorRPM():

Esta función sencillamente llama a la función EXTI15_10_Config, no es imprescindible, pero se ha implementado para dar una nomenclatura más sencilla e intuitiva a esta funcionalidad, aunque se conserva la función con su nombre original por coherencia con las librerías del STM32L-Discovery.

-Control_rpm():

Esta función tiene como objetivo el control de las revoluciones por minuto del motor de corriente continua.

Para ello inicializa a cero una variable que se va incrementando cada vez que se produce una interrupción externa debida al sensor óptico colocado junto al motor de corriente continua. A continuación provoca un retardo de 60 segundos mediante la función Delay y, una vez transcurrido ese tiempo, divide el valor de la variable interna por dos y lo devuelve como resultado, siendo éste el numero de revoluciones del motor durante un minuto (la división por dos se lleva a cabo porque el motor activa el sensor dos veces por vuelta, con cada uno de los extremos del aspa colocada en el eje de giro).

5.2.3-MATRIZ DE LEDS:

5.2.3.1-Objetivos:

Para controlar la matriz de LEDs se necesitan una serie de funciones que rellenen de forma adecuada los registros de desplazamiento en función de los LEDs que se necesiten iluminar.

5.2.3.2-Funciones:

-retardoLED():

El objetivo de esta función es generar un pequeño retardo que es necesario aplicar a la hora de grabar los registros que controlan la matriz de LEDs.

Es tan sencilla como un bucle que se repite un pequeño número de veces para que mientras tanto se estabilice el valor que se está enviando o recibiendo.

-iniciaMatrizLEDs():

Esta función configura el GPIO de forma que los cuatro pines correspondientes a la matriz de LEDs estén en modo OUTPUT, pull-up.

Una vez configurados los inicializa a cero, para que no haya ninguna comunicación accidental.

-borra_matrizLEDs():

Esta función sirve para apagar todos los LEDs de la matriz.

Para borrar la matriz, primero llama a la función iniciaMatrizLEDs, y luego va llenando los dos registros de desplazamientos, de forma que todos los bits sean iguales a 0, primero las columnas y luego las filas. Una vez hecho esto, se provoca un pequeño pulso en la señal de CLS, liberando los valores de los registros y indicando a todos los LEDs que se desactiven.

-clockcoltick():

Esta función provoca un pulso en la señal de reloj del registro de desplazamiento dedicado a las columnas, por medio de la función retardoLED.

-clockfiltick():

Esta función provoca un pulso en la señal de reloj del registro de desplazamiento dedicado a las filas, por medio de la función retardoLED.

-col_ON():

Esta función sirve para poner a 1 el siguiente bit del registro de desplazamiento dedicado a las columnas. Para ello pone a nivel alto la señal de Dato y provoca un pulso en la señal de reloj del registro de las columnas mediante la función retardoLED.

-fil_ON():

Esta función sirve para poner a 1 el siguiente bit del registro de desplazamiento dedicado a las filas. Para ello pone a nivel alto la señal de Dato y provoca un pulso en la señal de reloj del registro de las filas mediante la función retardoLED.

-col_OFF():

Esta función sirve para poner a 0 el siguiente bit del registro de desplazamiento dedicado a las columnas. Para ello pone a nivel bajo la señal de Dato y provoca un pulso en la señal de reloj del registro de las columnas mediante la función retardoLED.

-fil_OFF():

Esta función sirve para poner a 0 el siguiente bit del registro de desplazamiento dedicado a las filas. Para ello pone a nivel bajo la señal de Dato y provoca un pulso en la señal de reloj del registro de las filas mediante la función retardoLED.

-opt():

Esta función provoca un pulso en la señal de habilitación de los registros de desplazamiento, liberando los bits guardados y cargando la nueva configuración en la matriz de LEDs, por medio de la función retardoLED.

-unLED_ON(int x, int y):

El objetivo de esta función es encender un solo LED de la matriz, que viene indicado por las coordenadas “x” e “y”.

Para ello se tiene que tener en cuenta cuales son los registros asociados a cada fila y a cada columna: los bits Q7, Q6 y Q0 del registro de desplazamiento asociado a las columnas de la matriz no se corresponden con ninguna columna, por lo que se dejarán siempre activados; y lo mismo pasa con el bit Q0 del registro de desplazamiento asociado a las filas.

Una vez se ha comprendido el párrafo anterior, se van introduciendo los bits que se desean en cada uno de los registros de desplazamiento, empezando por el Q7 y acabando por el Q0, mediante las funciones col_ON,

col_OFF, fil_ON y fil_OFF. Finalmente se aplica un pequeño retardo con retardoLED y se carga la configuración en la matriz de LEDs llamando a la función opt.

-columna_ON(int c)

Esta función tiene como objetivo encender una columna entera de la matriz de LEDs (la indicada en la variable de entrada “c”).

Para ello se hace un switch, según el valor de la variable “c” se activa la columna correspondiente y se desactivan las demás mediante las funciones col_ON y col_OFF.

A continuación se activan todas las filas mediante la función fil_ON y se carga la configuración en la matriz de LEDs llamando a la función opt.

- fila_ON(int f):

Esta función tiene como objetivo encender una fila entera de la matriz de LEDs (la indicada en la variable de entrada “f”).

Para ello se activan todas las filas mediante la función fil_ON .

A continuación se hace un switch, según el valor de la variable “f” se activa la fila correspondiente y se desactivan las demás mediante las funciones fil_ON y fill_OFF.

Finalmente se carga la configuración en la matriz de LEDs llamando a la función opt.

-demoLED(int i):

demoLED es una función de test, desarrollada para la demostración y el chequeo de la placa, que va encendiendo una columna tras de otra, y posteriormente una fila tras de otra de la matriz de LEDs.

5.2.4-ALTAVOZ:

5.2.4.1-Objetivos:

Para controlar el altavoz se utiliza uno de los timers multifunción del STM32L-Discovery, que genera una señal que varíe entre 0 y 1 con la frecuencia que se le indique. También se utiliza una función de temporización para controlar la duración de cada sonido emitido por el altavoz.

El altavoz utiliza varias funciones de la librería de los Timer, por lo que algunos aspectos de ciertas funciones se explican en este apartado.

5.2.4.2-Funciones:

Las funciones utilizadas para controlar el altavoz son las siguientes:

-iniciaGPIOAltavoz():

Esta función configura el GPIO de forma que el pin correspondiente al altavoz esté en modo OUTPUT, no-pull.

-iniciaAltavoz():

Esta función llama a la función anterior para configurar el GPIO, y a la función que configura el TIM3 para enviar señales al altavoz.

De esta forma, una vez ejecutada la función, el altavoz queda preparado para funcionar.

-enviaNota(unsigned int duracion, unsigned int periodo):

El objetivo de esta función es generar la señal que estimule el altavoz de forma que genere un sonido con la frecuencia y duración deseadas, que se indican en las dos variables de entrada de la función.

Para ello inicia el TIM3 para generar una señal de la frecuencia indicada, y usa la función Delay, perteneciente a las librerías de los timers para provocar un retardo de la duración indicada, en ms.

-himnoAlegria():

Se trata de una función de demo que reproduce un fragmento del "Himno de la Alegría" de Beethoven.

-Tetris():

Se trata de una función de demo que reproduce un fragmento de la melodía del videojuego clásico "Tetris".

5.2.5-TIMER:

5.2.5.1-Objetivos:

Los timer son una de las principales herramientas a la hora de trabajar con microcontroladores. De hecho, en algunas librerías pertenecientes a otros periféricos se implementan funciones que configuran y utilizan algunos timers del STM32L-Discovery, como el que controla la modulación PWM del motor.

Con esta librería se pretenden desarrollar dos de las funciones más importantes y más comunes que desempeñan los timers: temporización y generación de ondas cuadradas, ambas según los parámetros que el programador indique.

5.2.5.2-Funciones:

En este apartado se explican las funciones implementas en la librería de los timer:

-void iniciaTim2Temp(void):

Esta función inicializa el TIM2 en modo output compare para que actúe como temporizador. Para ello debe seguir una serie de pasos:

-Inicializa las propiedades del timer: define una base de tiempos, un prescaler y configura la cuenta hacia arriba.

-Habilita las interrupciones de los canales que va a utilizar.

-Configura el gestor de interrupciones (NVIC), asignándole el TIM2 y activándolo.

-Configura el modo de output compare de los canales que desea utilizar e inicializa el GPIO de los pines que desea activar.

-void Arranca_Tim2Temp (unsigned int duracion):

Habilita el TIM2 y modifica la configuración del output compare para medir el tiempo deseado.

-void iniciaTim3Audio(void):

Esta función realiza todas las configuraciones necesarias para enviar una señal al altavoz:

-configura el pin correspondiente del GPIO en modo “Alternate Function”, pull-up, y lo asocia al TIM3.

-Habilita las interrupciones de ese timer.

-Inicializa las propiedades del timer: define una base de tiempos, un prescaler y configura la cuenta hacia arriba.

-Configura el modo de output compare del canal que desea utilizar.

-void Arranca_Tim3Audio(unsigned int periodo):

Habilita el TIM3 y modifica la configuración del output compare para que la señal generada tenga la frecuencia deseada.

-void Para_Tim3Audio (void):

Deshabilita el Tim3.

-void TIM3_IRQHandler(void):

Define las interrupciones del TIM3.

-void TIM2_IRQHandler(void):

Define las interrupciones del TIM2.

-void Delay(uint32_t nTime):

Provoca un retardo.

-void TimingDelay_Decrement(void):

Cuenta el tiempo para la función Delay.

5.2.6-SENSOR DE INFRARROJOS

5.2.6.1-Objetivos:

Se desea tener la capacidad de producir interrupciones mediante el sensor de infrarrojos.

5.2.6.2-Funciones:

-void EXTI9_5_Config(void):

Configura el pin del GPIO PB5, asociado al sensor infrarrojo en modo IN, no-pull, lo conecta a la línea 5 del gestor de interrupciones externas (EXTI), y habilita las interrupciones del mismo.

-void iniciaSensorIR(void):

Llama a la función anterior.

Puede parecer que una de las dos funciones sobra, pero es adecuado implementar la otra por coherencia con las librerías básicas que se están utilizando y también necesario contar con ésta para mantener la filosofía de sencillez y claridad de esta librería.

-void desactivaSensorIR(void):

Configura el pin PB5 del GPIO en modo OUTPUT para que el sensor infrarrojo deje de provocar interrupciones a través de él.

5.2.7-I2C

5.2.7.1-Objetivos:

El protocolo I2c es un sistema de comunicaciones extremadamente versátil, en este caso se va a utilizar para escribir y leer datos de una memoria EEPROM integrada en la placa de E/S.

5.2.7.2-Funciones:

Las funciones existentes en la librería básica para el control de comunicaciones I2C son bastante sencillas, y los ejemplos incluidos con la documentación original del STM32L-Discovery utilizan otras librerías, que contienen una serie de funciones que permiten la comunicación vía I2C utilizando DMA. Sin embargo, en este caso no es necesario utilizar DMA, y hacerlo supondría complicar demasiado unas funciones que lo que pretenden es ilustrar de forma sencilla como se puede llevar a cabo una comunicación serie.

Por lo tanto, se investiga en las librerías de otros modelos de placas de desarrollo de ST que montan un Cortex-M3 y se encuentra una librería de comunicación I2C que se ajusta a las necesidades del proyecto, perteneciente a un ejemplo desarrollado para el STM3210B-EVAL. De ella se extraen 5 funciones que se incluyen en esta librería para simplificar su funcionamiento.

-void I2C_EE_ByteWrite(uint8_t* pBuffer, uint8_t WriteAddr):

Escribe un byte de información en la dirección de memoria indicada.

-void I2C_EE_PageWrite(uint8_t* pBuffer, uint8_t WriteAddr, uint8_t NumByteToWrite):

Escribe un dato cuya longitud el numero de bytes indicado, en la direccion de memoria indicada.

-void I2C_EE_BufferWrite(uint8_t* pBuffer, uint8_t WriteAddr, uint16_t NumByteToWrite):

Escribe el numero de bytes del buffer indicados, en la direccion de memoria indcada.

-void I2C_EE_BufferRead(uint8_t* pBuffer, uint8_t ReadAddr, uint16_t NumByteToRead):

Lee de la direccion de memoria indicada el numero de bytes indicado y lo almacena en el buffer.

-void I2C_EE_WaitEepromStandbyState(void):

Espera que la EEPROM esté desocupada, para poder enviarle otra instrucción.

-void GPIO_I2C_Config(void):

Esta función inicializa el GPIO, configurando los pines asociados a la comunicación I2C como “Alternate Functions”, en open-drain y conectándolos al canal I2C1 del STM32L-Discovery.

-void Configura_I2C(void):

El objetivo de esta función es configurar e inicializar el modulo I2C del STM32L-Discovery, poniéndolo en modo I2C normal, indicándole el ciclo de

trabajo, la dirección de la memoria EEPROM que actúa como esclavo en el bus de comunicaciones que se establece, activando la señal de “acknowledge” y fijando la velocidad de transmisión.

-void Inicia_I2C(void):

Llama a las dos funciones anteriores para configurar e inicializar la comunicación I2C.

-void leer_I2CEEPROM(uint8_t* pBuffer, uint8_t dirLectura, uint16_t tamanodato):

Llama a la función I2C_EE_BufferRead para leer un dato del tamaño indicado de la posición de memoria indicada y guardarlo en la variable correspondiente. Después espera a que la memoria este libre de trabajo, para poder enviarle otra instrucción, mediante la función I2C_EE_WaitEepromStandbyState.

void escribir_I2CEEPROM(uint8_t* pBuffer, uint8_t dirEscritura, uint16_t tamanodato);

Llama a la función I2C_EE_BufferWrite para escribir un dato del tamaño indicado en la posición de memoria indicada.

5.2.8-TECLADO

5.2.8.1-Objetivos:

El control del teclado es uno de los más sencillos de toda la placa de E/S, ya que las únicas funcionalidades necesarias son reconocer cuándo se ha pulsado una tecla y cuál ha sido la tecla pulsada.

5.2.8.2-Funciones:

En la cabecera del archivo se define una matriz (array) de 4x4 que contiene los caracteres equivalentes a cada una de las 16 teclas. Además también se define un vector que contiene los pines correspondientes a cada una de las conexiones del STM32L-Discovery con el teclado, gracias a esto, es muy fácil utilizar bucles para llevar a cabo algunas de las operaciones necesarias de configuración y rastreo de pines que se llevan a cabo en esta librería, consiguiendo un código más limpio y claro.

Las funciones necesarias para controlar el teclado son las siguientes:

-void iniciaTeclado(void):

Esta función configura el GPIO de forma que los pines correspondientes a las filas estén en modo OUTPUT, pull-up, y los correspondientes a las columnas estén en modo IN, no-pull, e inicializa las salidas a 0.

-int dimeFila():

Con esta función se leen los cuatro pines de entrada y, en caso de que solo uno de ellos esté a nivel bajo, se devuelve el entero correspondiente a la fila que corresponde.

En caso de que no se cumpla la condición anterior, devuelve -1.

-signed char dimeTecla(void):

Esta función reconoce devuelve la tecla que se ha pulsado. Para ello sigue los siguientes pasos:

-Inicia un bucle “for” entre 0 y 3.

-Pone a nivel alto todas las salidas, para limpiar el estado anterior.

-Pone a nivel bajo una de las salidas.

-Llama a la función dimeFila().

-Si dimeFila() devuelve algo distinto de -1, guarda en valor y guarda también el valor del contador del bucle.

-Espera a que dimeFila() devuelva -1, lo que significa que se ha soltado la tecla.

-Los dos números son las coordenadas de la posición en la matriz del valor de la tecla pulsada, así que la función devuelve ese valor.

Si una vez ha ejecutado todo el bucle no se ha encontrado ninguna tecla pulsada, la función devuelve -1.

-signed char esperaTecla(void):

Esta función espera hasta que se pulsa una tecla y devuelve su valor.

Para conseguirlo, inicia un bucle “while” del que solo sale si la función dimeTecla() devuelve un valor diferente de -1, en ese caso lo guarda y lo devuelve.

6-CONCLUSIONES Y PROPUESTAS DE TRABAJOS FUTUROS

CONCLUSIONES:

Podemos afirmar que se ha cumplido el objetivo principal del proyecto, es decir la mejora de las herramientas usadas en las prácticas de laboratorio que se imparten en varias asignaturas de Microprocesadores, lo cual incluye de modo explícito el manejo de un IDE libre y la reutilización de la placa de E/S, de la cual existen 50 unidades en el Dpto. de Tecnología Electrónica.

Con este fin, se ha realizado un estudio para utilizar la plataforma libre Eclipse como entorno de desarrollo IDE sobre el STM32L-Discovery. Este análisis se ha basado en el manejo de la información publicada en diversos foros sobre microcontroladores existentes en la Web, donde se han encontrado varias soluciones, que han debido probarse para evaluar su adecuación a los requisitos de partida.

Como resultado de dicho análisis y evaluación, se ha conseguido configurar correctamente el IDE Eclipse para programar el microcontrolador de la placa de desarrollo Discovery, y depurar código sobre la misma, permitiendo así trabajar libres de las restricciones asociadas a las versiones de prueba de un IDE comercial.

Además, se ha diseñado un sistema de interconexión entre las placas STM32L-Discovery y E/S, que permite utilizar la funcionalidad de los periféricos existentes en ésta última placa, ahora bajo control del microcontrolador Cortex-M3 de la nueva placa de desarrollo Discovery.

Por último, se han escrito una serie de librerías de funciones, que se han añadido a una plantilla de proyecto para Eclipse, y que permiten al usuario controlar de manera sencilla e intuitiva cada uno de los periféricos de la placa de E/S desde el STM32L-Discovery.

Estas librerías se utilizan en un programa de demostración que, bajo control de su operador, permite ejecutar tests de verificación funcional sobre cada uno de los periféricos, y por tanto del sistema de desarrollo completo usado en prácticas.

Por lo tanto, se considera se ha conseguido un nuevo sistema de trabajo para las prácticas de laboratorio, basado la nueva generación de microcontroladores ARM, con una orientación de la programación basada en conceptos de alto nivel y realizado mediante software libre.

Trabajos futuros:

La principal vía de trabajo que queda abierta al término del proyecto es la realización del diseño de las placas de interconexión entre el STM32L-Discovery y la placa de E/S. Dado que estas nuevas placas formarán parte del material de laboratorio que se les entregará a los alumnos durante las prácticas, es recomendable que estén construidas de una forma más robusta que el prototipo utilizado para el desarrollo de este proyecto. Se recomienda por lo tanto que se realice el diseño de un circuito impreso sobre el que luego se monten los dos conectores para lo pines del STM32L-Discovery y el de 40 pines para la placa de E/S.

Por otro lado, también podría resultar interesante investigar algunas mejoras que hiciesen el IDE algo más cómodo de utilizar, como el desarrollo de scripts que eliminen la necesidad de pulsar el botón de external tools para reconectar con la placa de desarrollo, etc. Además, la instalación del software necesario para el entorno de desarrollo es compleja, por lo que sería conveniente desarrollar un método de instalación automática que permita facilitar este proceso.

7-PRESUPUESTO

El presente proyecto apenas ha generado costes de materiales, ya que ha consistido principalmente en la evaluación y desarrollo de software. Además, las placas de E/S ya han sido amortizadas por la universidad, por lo que no han supuesto un gasto adicional. Por lo tanto, no se han incluido en el presupuesto.

Los únicos gastos realizados en el proyecto son dos placas de desarrollo STM32L-Discovery, con un precio de mercado de 15.39 € (IVA incluido) y el precio de los materiales utilizados para construir la placa de interconexiones, que se estima menor de 3€, además del coste de ingeniería.

El proyecto ha requerido 600 horas de trabajo de ingeniería, a lo largo de 6 meses. Suponiendo un sueldo medio para un ingeniero electrónico de 25€/hora, los gastos de ingeniería ascenderían a 15000 €.

Por lo tanto, el presupuesto total del proyecto se estima en 15033.78 €.

ANEXOS:

En este proyecto se ha procurado reducir al máximo el número de anexos, ya que todas las hojas de catálogos de las placas de desarrollo, y los diferentes manuales para el uso de cada uno de ellos pueden encontrarse fácilmente en las páginas web de sus respectivos fabricantes, y en caso de ser incluidos incrementarían la longitud de esta memoria de manera innecesaria.

Sin embargo, sí que se considera adecuado incluir cinco anexos, distribuidos en tres grupos:

El primero es una guía de usuario para la instalación y uso del IDE basado en Eclipse, este documento está pensado para ser entregado al usuario que desee utilizar el IDE desarrollado en este proyecto, para ayudarle a instalar todo el software necesario, iniciar la plantilla y configurar el entorno de desarrollo. Esta guía es equivalente a lo explicado en el capítulo 3.4 de la memoria, pero se incluye porque se puede utilizar como documento independiente, imprimiendo las páginas correspondientes a este documento para que pueda ser distribuido.

Los anexos 2, 3 y 4 son una muestra del código desarrollado, para que se pueda apreciar la ausencia de referencias a registros o cualquier otro elemento de control de bajo nivel en la sintaxis del usuario.

Por último, el anexo 5 es una tabla extraída del manual del display LCD, en la cual se detallan los comandos para controlar la misma.

ANEXO 1: GUÍA DE INSTALACIÓN Y USO DEL IDE BASADO EN ECLIPSE

GUÍA DE INSTALACIÓN Y USO DE UN ENTORNO DE DESARROLLO BASADO EN SOFTWARE LIBRE (ECLIPSE) PARA UN STM32L-DISCOVERY

Versión: 1.0

Autor: Norberto J. Rivillo Matía

Software necesario:

-Eclipse (versión Indigo) for C/C++ Developers → http://www.eclipse.org/downloads/

-Última versión de Java de Oracle → http://www.java.com/

-ST-Link GDBServer, extraído del software Atollic TrueStudio (versión anterior a la 3.0, a partir de esta versión parece que da errores)

-Sourcery CodeBench Lite for ARM EABI
→ https://sourcery.mentor.com/sgpp/lite/arm/portal/release1802

-Cygwin (emulador de comandos Linux para Windows)
→ http://cygwin.com/install.html

Lo primero que debemos hacer es conseguir todos los elementos de la lista anterior, e instalarlos en el equipo (excepto Eclipse, que no será necesario). Es recomendable instalar las aplicaciones en la raíz del disco C:/, para evitar problemas con las carpetas, ya que vamos a trabajar emulando linux, y éste tiene problemas con los nombres de carpeta demasiado largos o con espacios. Una vez tengamos las aplicaciones instaladas, procedemos a descomprimir Eclipse en una carpeta expresamente creada para ello

(preferentemente también en la raíz del disco C:/) y creamos una última carpeta, que usaremos como workspace. Esta capeta puede estar donde queramos, pero cuanto mas sencilla sea la ruta, mejor.

Una vez hemos seguido las instrucciones anteriores, procedemos a configurar eclipse según las guías que encontramos esta página web:

http://www.chibios.org/dokuwiki/doku.php?id=chibios:guides:stlink_eclipse&s[]=stm32l

Lo primero que debemos tener en cuenta es que Eclipse no se instala como la mayoría de los programas en Windows, sino que funciona como un "portable": colocamos la carpeta que hemos descomprimido en el directorio que deseemos y dentro de ella encontramos el ejecutable que lanza Eclipse. Los pasos que debemos llevar a cabo para configurar el proyecto son los siguientes:

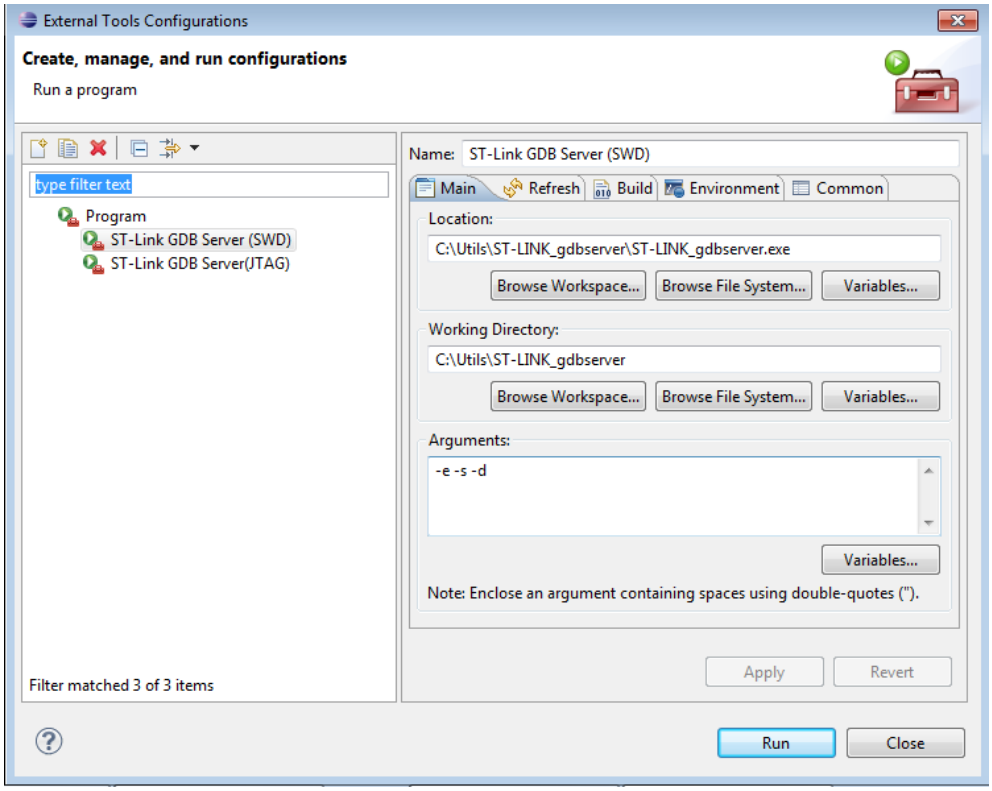
-Permitimos que Eclipse busque actualizaciones automáticamente desde "Help"→ "Check for updates"

-Instalamos la extensión "C/C++ Hardware Debugging" para Eclipse desde "Help"→ "Install new software".

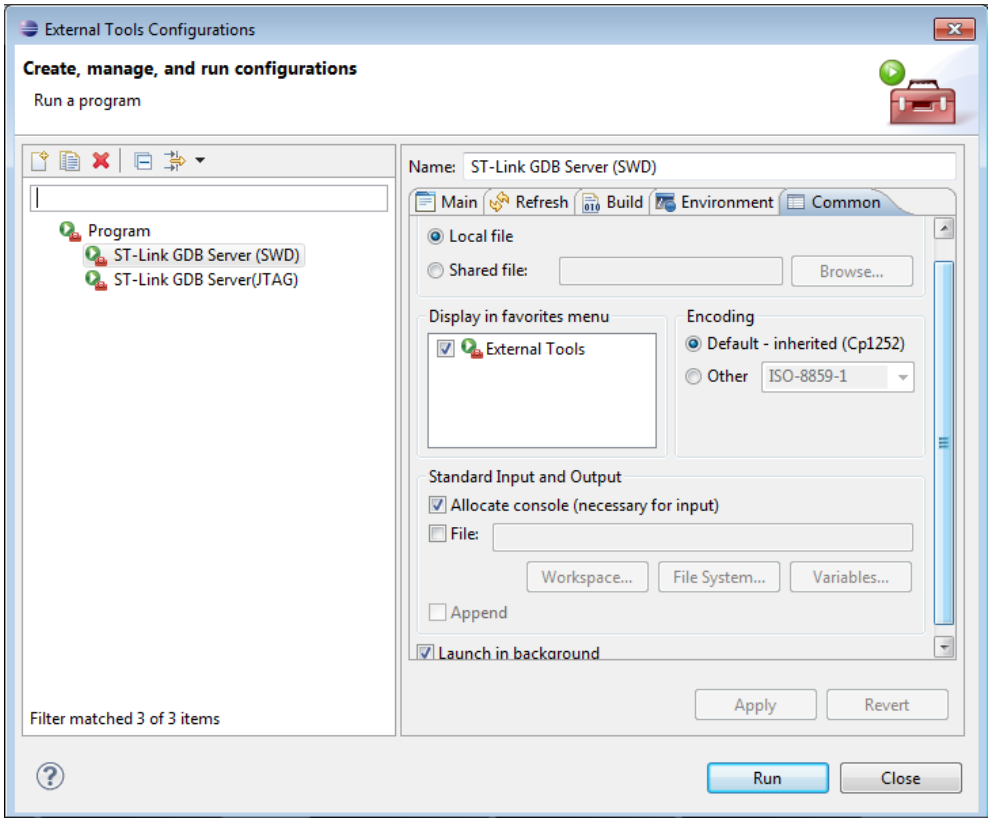
-Utilizando un programa de descompresión de archivos, como 7zip, extraemos la carpeta ST-Link_gdbserver del instalador de Atollic TrueStudio, y la guardamos, por ejemplo, en C:/Utils.

-Para hacer que Eclipse utilice la aplicación anterior al trabajar con nuestra placa, vamos a "Run"→"External Tools"→ "External Tools Configuration", creamos un nuevo lanzador haciendo clic derecho en "Program" → "New". A continuación configuramos las siguientes pestañas como aparece en las imágenes y guardamos la configuración.

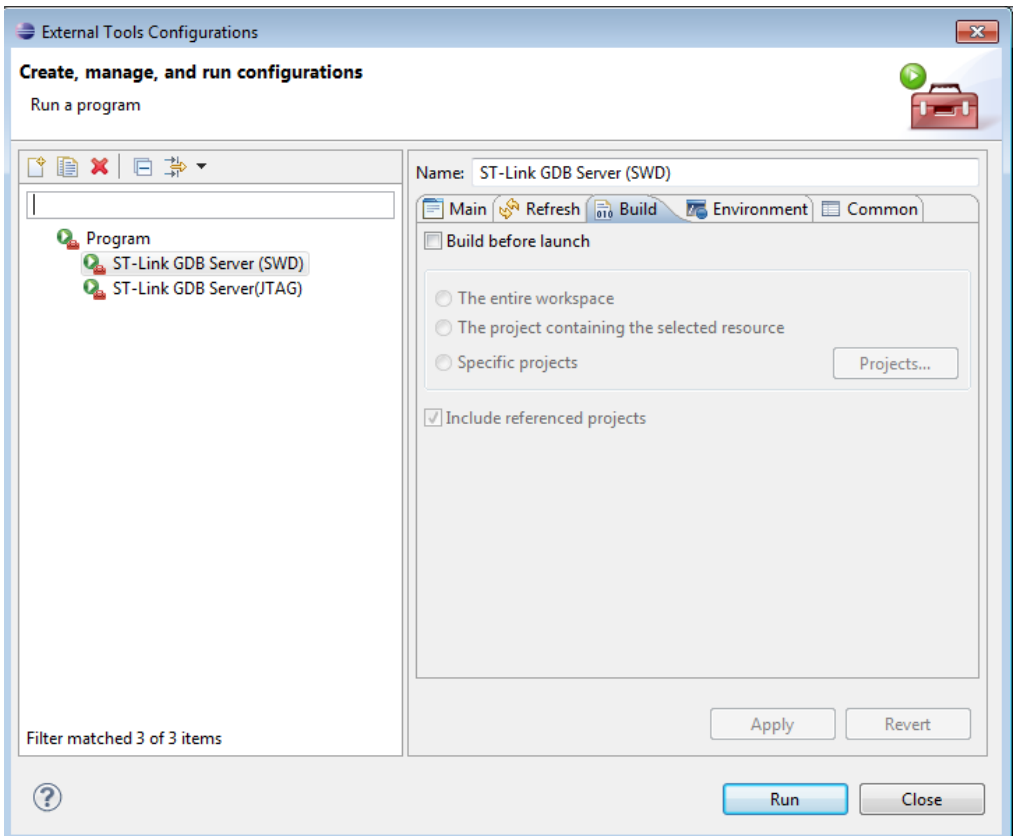
Main (los path cambiarán según donde se ubique el lanzador):



Common:



Build:



-Antes de seguir con la configuración de Eclipse, vamos a preparar la plantilla del proyecto para poder llevar a cabo su configuración de depuración. Para ello, podemos acceder al siguiente link, del que descargamos el proyecto de prueba que ST incluye en las placas, adaptado para Eclipse: https://skydrive.live.com/?lc=1041&ppud=4#cid=E726C1E354E854B2&id=E726C1E354E854B2%21122

A este proyecto habrá que hacerle una serie de modificaciones para tener una plantilla a partir de la cual crear nuevos proyectos, así que es más cómodo utilizar la carpeta “plantilla” que se incluye con la documentación.

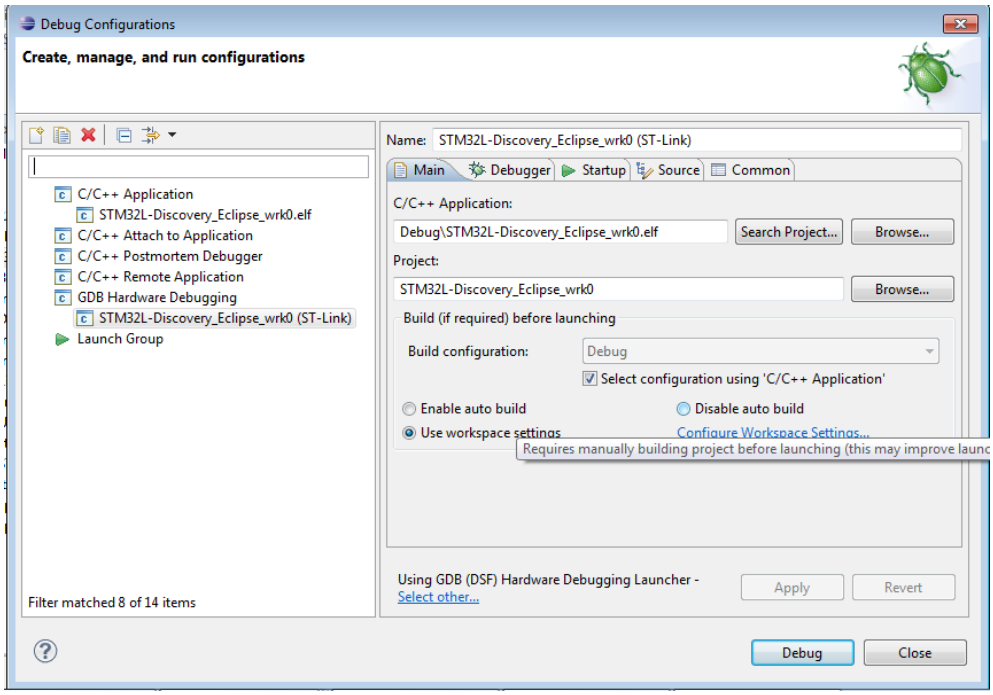
-Una vez ya tenemos una plantilla preparada, importaremos el proyecto a eclipse:

"File"→ "Import"→ "General"→ "Existing projects into workspace"→ "Next"→ "Browse"→ indico la ruta de la carpeta de la plantilla → seleccionar "Copy projects into workspace"→ "Finish".

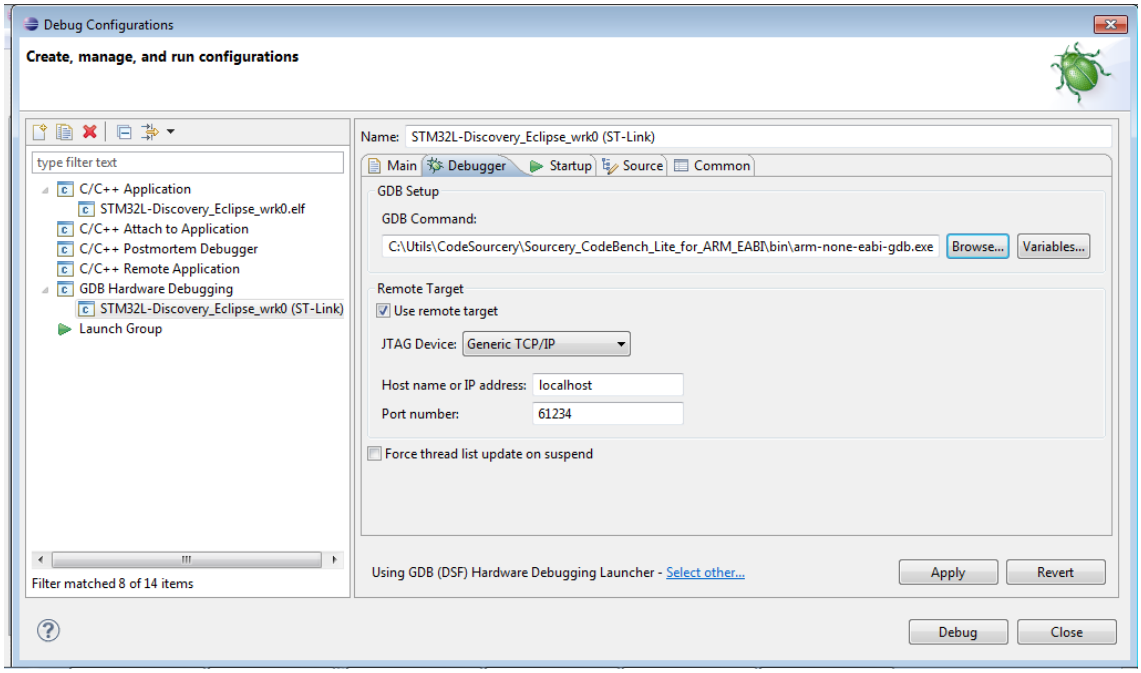
-Para evitar posibles problemas en la compilación: "Project"→ "Preferences"→ "Language mapping"→ asigno GNU C a C Header file y C Source file.

-Ahora procedemos a preparar la configuración de depuración: desplegamos el menú de "Debug"→ "Debug configurations"→creamos una nueva configuración dentro de "GDB hardware debugging", configurando las pestañas como se puede ver en las imágenes que aparecen a continuación:

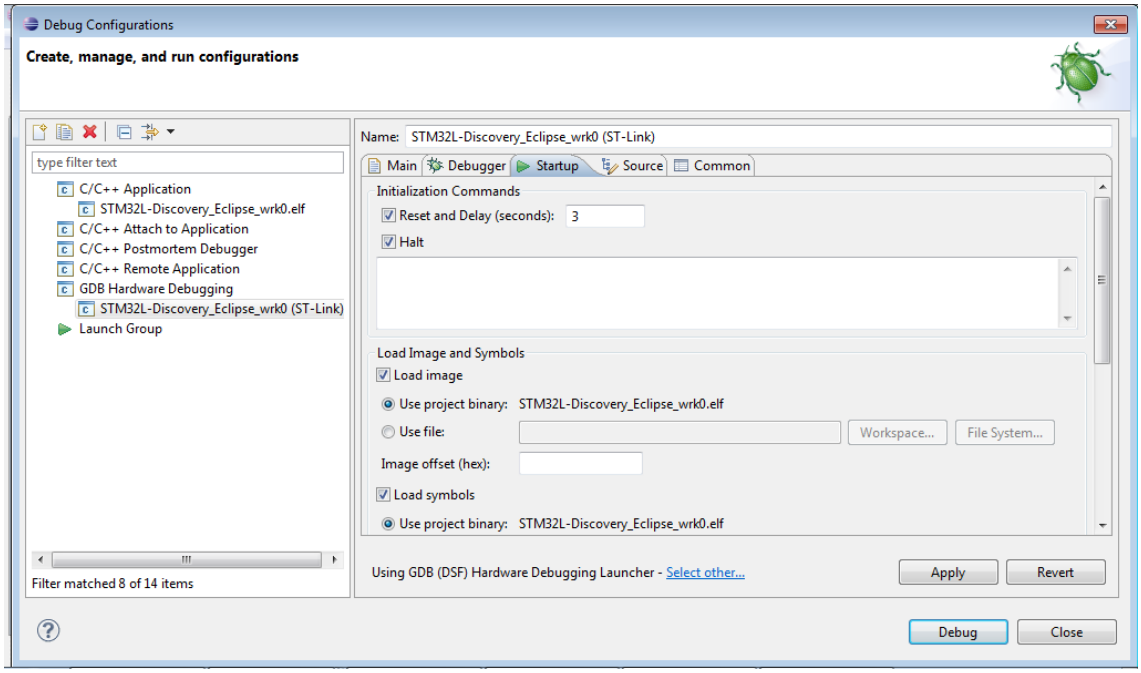
Main:

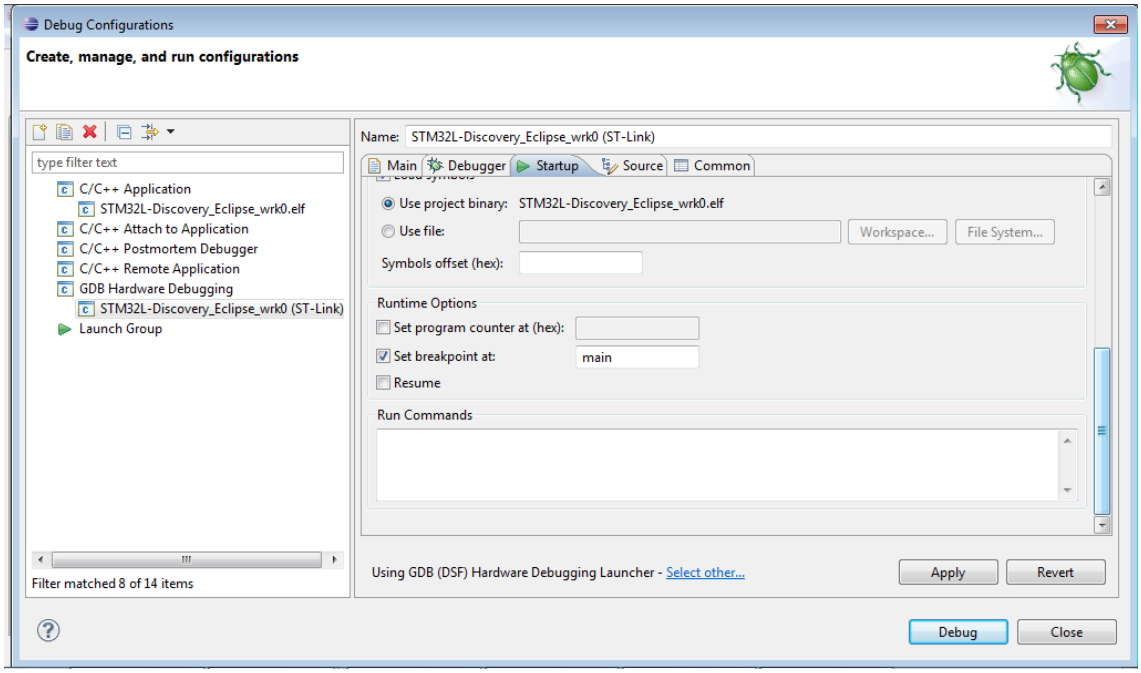


Debugger:

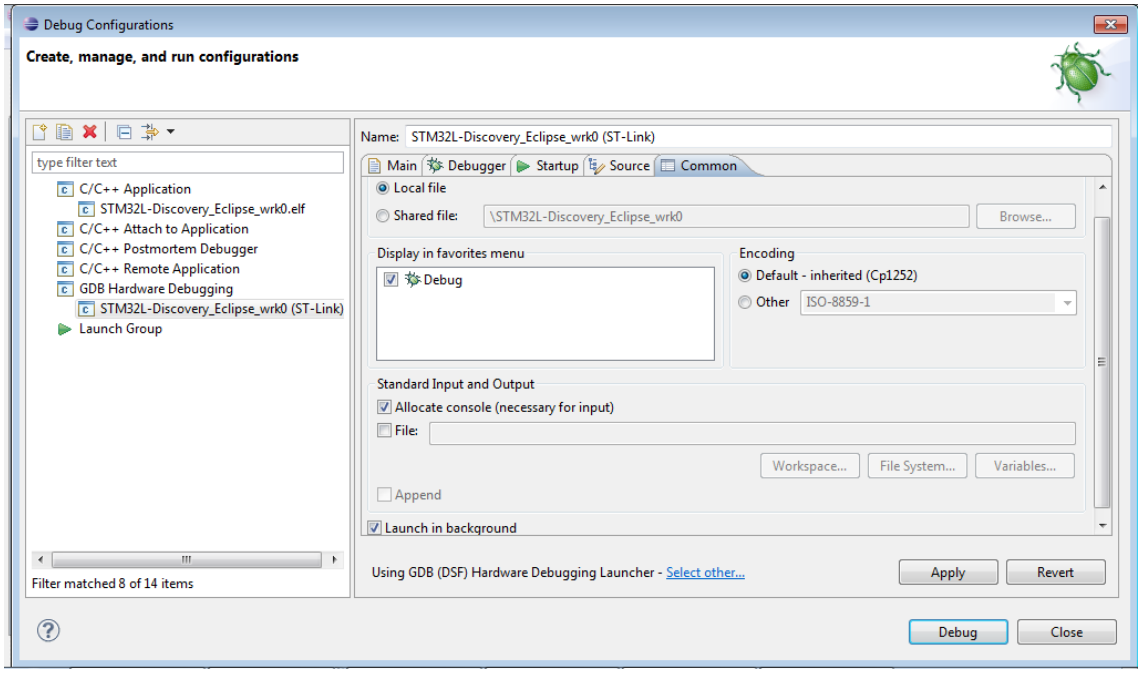


Startup:




Debug Configurations
Create, manage, and run configurations
type filter text
C/C++ Application
STM32L-Discovery_Eclipse_wrk0.elf
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
GDB Hardware Debugging
STM32L-Discovery_Eclipse_wrk0 (ST-Link)
Launch Group
Filter matched 8 of 14 items
Name: STM32L-Discovery_Eclipse_wrk0 (ST-Link)
Main
Debugger
Startup
Source
Common
Use project binary: STM32L-Discovery_Eclipse_wrk0.elf
Use file:
Workspace...
File System...
Symbols offset (hex):
Runtime Options
Set program counter at (hex):
Set breakpoint at:
main
Resume
Run Commands
Using GDB (DSF) Hardware Debugging Launcher - Select other...
Apply
Revert
Debug
Close

Common:


Debug Configurations
Create, manage, and run configurations
type filter text
C/C++ Application
STM32L-Discovery_Eclipse_wrk0.elf
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
GDB Hardware Debugging
STM32L-Discovery_Eclipse_wrk0 (ST-Link)
Launch Group
Filter matched 8 of 14 items
Name: STM32L-Discovery_Eclipse_wrk0 (ST-Link)
Main
Debugger
Startup
Source
Common
Local file
Shared file:
\STM32L-Discovery_Eclipse_wrk0
Browse...
Display in favorites menu
Debug
Encoding
Default - inherited (Cp1252)
Other
ISO-8859-1
Standard Input and Output
Allocate console (necessary for input)
File:
Workspace...
File System...
Variables...
Append
Launch in background
Using GDB (DSF) Hardware Debugging Launcher - Select other...
Apply
Revert
Debug
Close

Ahora que todo está configurado como en las imágenes, pulsamos “Apply para guardar la configuración”.

Ya hemos preparado la plantilla y hemos configurado el proyecto, cuando queramos probar un programa en nuestra placa seguiremos los siguientes pasos:

-Comprobar que el programa no tiene errores de compilación.

-Conectar la placa al ordenador mediante el cable USB.

-Desplegar el menú “External Tools” y lanzar el “ST-Link GDB Server (SWD)”.

-Desplegar el menú “Debug” y seleccionar “STM32L-Discovery_Eclipse_wrk0 (ST-Link)”.

Una vez hayamos seguido estos pasos, se procederá a comprobar el código y cargarlo en la placa. Una vez haya terminado de cargarse, bastará con pulsar el botón de “Play” para lanzar la aplicación, corriendo sobre nuestra placa.

ANEXO 2: EJEMPLOS DE CÓDIGO: LIBRERÍA DEL TECLADO

La primera librería que se presenta es una de las más sencillas de desarrollar y comprender, la que controla el funcionamiento del teclado matricial:

```
/*
 * Libreria_teclado.c
 *
 *  Created on: 07/06/2012
 *      Author: Norberto Rivillo
 */

#include "Libreria_teclado.h"
#include "Libreria_timer.h"
#include "stm32l1xx.h"
#include "stm32l1xx.h"
#include "stm32l1xx_conf.h"
#include <stddef.h>
#include "stm32l1xx.h"
#include "discover_board.h"
#include "Libreria_matrizLED.h"



//Definición de los pines correspondientes al teclado
//estas dos variables facilitan el uso de bucles for, como se verá en algunas funciones
uint16_t FilasTeclado[4]={GPIO_Pin_0 , GPIO_Pin_1 , GPIO_Pin_2,  GPIO_Pin_3};
uint16_t ColumnasTeclado[4]={GPIO_Pin_1 ,  GPIO_Pin_2,  GPIO_Pin_3,  GPIO_Pin_5 };

//esta matriz contiene cada uno de los caracteres correspondientes a las 16 teclas existentes, servirá para
//identificar correctamente la tecla pulsada
char matriz[4][4]={{'1','2','3', 'F'},{'4','5','6', 'E'},{'7','8','9', 'D'},{'A','0','B', 'C'}};


//Configura e inicializa como entradas y salidas los pines corespondientes al control del teclado
void iniciaTeclado(void)
{
            GPIO_InitTypeDef GPIO_InitStructure;
            //Configurar pines PA1, 2, 3 y 5 como ENTRADAS
            /* Enable GPIOA clock */
             RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
                        //habilita el reloj del GPIOA
             /* Configure PA1, 2, 3 & 5 pin as input floating */
             GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 |GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_5;
          //selecciona los cuatro pines correspondientes
             GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
                                    //modo entarda
             GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
                        //modo salida
             GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
                        //configura la velocidad del puerto
             GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
                        //modo no-pull
             GPIO_Init(GPIOA, &GPIO_InitStructure);
                        //inicializa la variable que se acaba de rellenar en el puerto GPIOA
```

```
	//Configurar pines PC0 al 3 como SALIDAS
	/* GPIOC Periph clock enable */
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC, ENABLE);   //habilita el reloj del GPIOC

	/* Configure PC0, 1, 2 & 3 in output pushpull mode */
	//selecciona los cuatro pines correspondientes
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1 |GPIO_Pin_2|GPIO_Pin_3;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;          //modo salida
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;         //modo push-pull
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;      //configura la velocidad del puerto
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;           //modo pull-up
	//inicializa la variable que se acaba de rellenar en el puerto GPIOC
	GPIO_Init(GPIOC, &GPIO_InitStructure);

	//Poner los pines de salida a 0 para que el teclado no funcione
	GPIO_LOW(GPIOC, GPIO_Pin_0);
	GPIO_LOW(GPIOC, GPIO_Pin_1);
	GPIO_LOW(GPIOC, GPIO_Pin_2);
	GPIO_LOW(GPIOC, GPIO_Pin_3);

}


//Devuelve la fila del botón pulsado, o -1 en caso de que no haya ninguna pulsado
int dimeFila()
{
	//inicializa las variables
	int ReturnCode = -1;
	uint8_t bitstatus0 = 0;
	uint8_t bitstatus1 = 0;
	uint8_t bitstatus2 = 0;
	uint8_t bitstatus3 = 0;

	//Lee los cuatro pines de entrada
	bitstatus0=GPIO_ReadInputDataBit(GPIOA, ColumnasTeclado[0]);
	bitstatus1=GPIO_ReadInputDataBit(GPIOA, ColumnasTeclado[1]);
	bitstatus2=GPIO_ReadInputDataBit(GPIOA, ColumnasTeclado[2]);
	bitstatus3=GPIO_ReadInputDataBit(GPIOA, ColumnasTeclado[3]);

	//Comprueba las cuatro situaciones posibles que implican que se ha pulsado una tecla: 0111,
	//1011, 1101 y 1110
	if( (bitstatus0==0)&&(bitstatus1!=0)&&(bitstatus2!=0)&&(bitstatus3!=0))
	{
		ReturnCode=0;
	}
	else if( (bitstatus0!=0)&&(bitstatus1==0)&&(bitstatus2!=0)&&(bitstatus3!=0))
	{
		ReturnCode=1;
	}
	else if( (bitstatus0!=0)&&(bitstatus1!=0)&&(bitstatus2==0)&&(bitstatus3!=0))
	{
		ReturnCode=2;
	}
	else if( (bitstatus0!=0)&&(bitstatus1!=0)&&(bitstatus2!=0)&&(bitstatus3==0))
	{
		ReturnCode=3;
	}
	else
	{
		ReturnCode=-1;
	}

	//Devuelve la posicion de la fila que se encuentra activa (a nivel bajo)
	return(ReturnCode);
}
```

```
//Devuelve la tecla pulsada, o -1 en caso de que no se haya pulsado ninguna
signed char dimeTecla(void)
{
	//inicializa las variables
	int columna=-1;
	int  i =0;
	int fila=-1, fila2=-1;
	signed char ReturnCode = -1;

	//bucle for, se repite cuatro veces el proceso para revisar todas las teclas
	for(i=0; i<4; i++)
	{
		//Primero se ponen los 4 pines a nivel alto para limpiar el estado anterior
		GPIO_HIGH(GPIOC, GPIO_Pin_0);
		GPIO_HIGH(GPIOC, GPIO_Pin_1);
		GPIO_HIGH(GPIOC, GPIO_Pin_2);
		GPIO_HIGH(GPIOC, GPIO_Pin_3);

		//pequeño retardo
		retardoLED();

		//Se va poniendo cada uno de los pines a nivel bajo para ver si hay una tecla de esa
		//columna pulsada
		GPIO_LOW(GPIOC, FilasTeclado[i]);

		//pequeño retardo
		retardoLED();
		//lee los cuatro pines
		fila= dimeFila();
		//Si hay una tecla pulsada, espera a que se suelte y devuelve su valor, si no devuelve -1
		if(fila!=-1)
		{
			//guarda la columna que se estaba comprobando
			columna=i;
			// Comprueba que se ha soltado la tecla
			do{fila2= dimeFila();}while(fila2!=-1);
			//asigna a la variable de salida el carácter correspondiente a la tecla pulsada
			ReturnCode = matriz[fila][columna];
		}
	}

	//devuelve el carácter correspondiente
	return(ReturnCode);

}


//Espera a que se pulse alguna tecla y devuelve su valor
signed char esperaTecla(void)
{
	//inicializa la variable
	signed char tecla = -1;

	//llama a la funcion anterior hasta que ésta le diga que se ha pulsado (y soltado) una tecla
	while(tecla < 0)
	{
		tecla=dimeTecla();
	}
	return tecla;
}
```

ANEXO 3: EJEMPLOS DE CÓDIGO: LIBRERÍA DEL DISPLAY LCD

La siguiente librería que se incluye a modo de ejemplo es la del display LCD, algo más compleja que la anterior pero también interesante.

Los comandos en hexadecimal que se envían al controlador del LCD corresponden con lo detallado en la tabla del anexo 5.

```
/*
 * Libreria_LCD.c
 *
 *  Created on: 06/06/2012
 *      Author: Norberto Rivillo
 */


#include "Libreria_timer.h"
#include "Libreria_LCD.h"
#include "stm32l1xx.h"
#include "stm32l1xx_conf.h"
#include <stddef.h>
#include "stm32l1xx.h"
#include "discover_board.h"
#include "Libreria_matrizLED.h"

#include "string.h"
#include "math.h"

//Definición de los pines utilizados por el LCD, para simplificar el uso de bucles cuando se quiere trabajar
//con todos ellos
uint16_t LCD_pines[8]={GPIO_Pin_11, GPIO_Pin_12, GPIO_Pin_10, GPIO_Pin_11, GPIO_Pin_12,
GPIO_Pin_13, GPIO_Pin_14, GPIO_Pin_15};
GPIO_TypeDef* LCD_GPIOs[8]={GPIOA, GPIOA, GPIOB, GPIOB, GPIOB, GPIOB, GPIOB, GPIOB};

int i =0;


//envía una señal al LCD, que puede ser un comando o un caracter a representar, según se especifique
void enviar_LCD(int com, int select)                    //select: 0=comando, 1 =letra
{
        //inicializa las varibles
        int comm=0;
        comm=com<<1;
    int i =0;

        //Configurar pines GPIO del LCD como salida (no se explica cada punto porque ya está
        //explicado en la funcion de configuración del teclado)
        GPIO_InitTypeDef GPIO_InitStructure;
        /* GPIOA Periph clock enable */
        RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);

        /* Configure PA8, 11 & 12 in output pushpull mode */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 |GPIO_Pin_11|GPIO_Pin_12;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
        GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
        GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
	/* GPIOB Periph clock enable */
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);

	/* Configure PB10, 11, 12, 13, 14, 15 in output pushpull mode */
	GPIO_InitStructure.GPIO_Pin =
GPIO_Pin_10|GPIO_Pin_11|GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
	GPIO_Init(GPIOB, &GPIO_InitStructure);

	/* GPIOC Periph clock enable */
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC, ENABLE);

	/* Configure PC6 & 7 in output pushpull mode */
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
	GPIO_Init(GPIOC, &GPIO_InitStructure);

	if(select==1)
	{GPIO_HIGH(GPIOC, GPIO_Pin_6);}		//Activa RS para enviar un carater
	else if(select==0)
	{GPIO_LOW(GPIOC, GPIO_Pin_6);}		//Desactiva RS para enviar un comando

	GPIO_LOW(GPIOC, GPIO_Pin_7);		//Desactiva el R/W

	//Pone los pines a 0(para limpiar el estado anterior)
	for(i =0; i<8; i++)
	{
		GPIO_LOW(LCD_GPIOs[i], LCD_pines[i]);
	}

	//Pone a 1 los pines q indique el comando
	for(i =0; i<8 ; i++)
	{comm=comm>>1;
		if (comm & 0x01)
		{
			GPIO_HIGH(LCD_GPIOs[i], LCD_pines[i]);
		}
	}

	retardoLED();				//pequeño retardo
	GPIO_HIGH(GPIOA, GPIO_Pin_8);		//Activa el Enable
	retardoLED();				//pequeño retardo
	GPIO_LOW(GPIOA, GPIO_Pin_8);		//Desactiva el Enable
	retardoLED();				//pequeño retardo
}

//Envía el comando especificado al LCD
void escribeComando_LCD(int com)
{
	enviar_LCD(com, 0);
}

//Envía al LCD la orden de representar un carácter a partir de una variable de tipo integer que representa
//la posición del mismo en la tabla ASCII
void escribeLetra_LCD(int com)
{
	enviar_LCD(com, 1);
}
```

```
//Envía al LCD la orden de representar un número del 0 al 9 a partir de una variable de tipo integer
//correspondiente a ese número
void escribeDigito_LCD(int com)
{
	enviar_LCD(com+48, 1);
}

//Espera mientras que el LCD envíe la señal de "busy", para no enviar comandos mientras el LCD está
//haciendo otra cosa y no los puede recibir
void espera_LCD(void)
{
	//inicializa la vriable
	int comp=0;

	//Configurar pines de control del LCD como salida
	GPIO_InitTypeDef GPIO_InitStructure;

	/* GPIOA Periph clock enable */
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);

	/* Configure PA8 in output pushpull mode */
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 ;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
	GPIO_Init(GPIOA, &GPIO_InitStructure);

	/* GPIOC Periph clock enable */
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC, ENABLE);

	/* Configure PC6 & 7 in output pushpull mode */
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
	GPIO_Init(GPIOC, &GPIO_InitStructure);

	//Configurar pines de datos del LCD como entrada

	/* GPIOC Periph clock enable */
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);

	/* Configure PA11, 12 pin as input floating */
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11|GPIO_Pin_12;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;

	GPIO_Init(GPIOA, &GPIO_InitStructure);


	/* GPIOB Periph clock enable */
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);

	/* Configure PB10, 11, 12, 13, 14, 15 pin as input floating */

	GPIO_InitStructure.GPIO_Pin =
GPIO_Pin_10|GPIO_Pin_11|GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
```

```
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
	GPIO_Init(GPIOB, &GPIO_InitStructure);



	//mientras esté activado el flag de busy(ocupado), no se saldrá de este bucle
	do
	{
		GPIO_LOW(GPIOC, GPIO_Pin_6);		//Desactiva RS
		GPIO_HIGH(GPIOC, GPIO_Pin_7);			//Activa R/W para activar lectura
		retardoLED();
		GPIO_HIGH(GPIOA, GPIO_Pin_8);			//Activa el Enable
		retardoLED();

		if(GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_15)==0x00)		//Coger el dato
		{
			comp=1;
		}

		GPIO_LOW(GPIOA, GPIO_Pin_8);		//Desactivar el enable

	}while(comp!=1);		//Comprobar si esta activado el Busy Flag(LCD ocupado)

}


//Configura e inicializa el LCD
void iniciaLCD(void)
{
	  Delay(60);
	  escribeComando_LCD(0x38);
	  Delay(6);
	  escribeComando_LCD(0x38);
	  Delay(1);
	  escribeComando_LCD(0x38);
	  espera_LCD();
	  escribeComando_LCD(0x38);	// Function Set
	  espera_LCD();
	  escribeComando_LCD(0x0e);	// Display on, cursor on, cursor parpadeante
	  espera_LCD();
	  escribeComando_LCD(0x06);	// Entry Mode Set --> El cursor se mueve incrementando la
posición
	  espera_LCD();
}

//Borra todo el contenido del LCD
void borra_LCD(void)
{
	espera_LCD();				//Comprueba si el LCD está realizando alguna acción
	escribeComando_LCD(0x01);	//Borra el display
}

//Escribe en el LCD una cadena de caracteres
void escribeCadena_LCD( char *cadena)
{

	espera_LCD();				//Comprobar si el LCD está realizando alguna acción
	while(*cadena!=0)		//mientras no llegue al final de la cadena de caracteres sigue
				//presentando en el LCD el contenido de la cadena
	{
		espera_LCD();
		escribeLetra_LCD(*cadena);
		cadena++;
	}
}
```

```
void escribeEntero_LCD( int entero)
{
	int i=0;
	int temp1=0, temp2=0;
	espera_LCD();				//Comprobar si el LCD está relizando alguna acción

//Calcula cada uno de los dígitos //del entero, de más a menos significativo, y los va representando
	for(i=4; i>=0; i--)		//i=4-->0 porque el mayor número que se puede representar con un
					//integer (16 bits), tiene 5 dígitos
	{
		temp1=temp2+temp1*10;
		temp2=(entero-temp1*pow(10, i+1))/(pow(10, i));
		espera_LCD();
		escribeDigito_LCD(temp2);
	}
}

//Posiciona el cursor del LCD en un determinado punto
void posicionCursor_LCD(int x, int y)
{
	int i;
	switch(x)
	{
		case 0:
		{
			espera_LCD();
			escribeComando_LCD(0x80);	//posiciona el cursor en el principio de la
							//primera fila
			if (y<16)
			{
				for(i=0;i<y;i++)
				{
					espera_LCD();
					escribeComando_LCD(0x14);	//avanza el cursor
				}
			}
			break;
		}
		case 1:
		{
			espera_LCD();
			escribeComando_LCD(0xc0);	//posiciona el cursor en el principio de la
							//segunda fila
			if (y<16)
			{
				for(i=0;i<y;i++)
				{
					espera_LCD();
					escribeComando_LCD(0x14);	//avanza el cursor
				}
			}
			break;
		}
		default:
		{
			espera_LCD();
			escribeCadena_LCD("¡Error Cursor!");
			break;
		}
	}
}
```

```
//Escribe en el LCD la cadena en la posición indicada
void escribeCadenaPos_LCD( char *cadena, int x, int y)
{
	posicionCursor_LCD(x,y);
	escribeCadena_LCD(cadena);

}

//Escribe en el LCD el entero en la posición indicada
void escribeEnteroPos_LCD( int entero, int x, int y)
{
	posicionCursor_LCD(x,y);
	escribeEntero_LCD(entero);
}

//hace que el texto se desplace hacia la derecha hasta desaparecer
void escribeCadenaScrollD(char *cadena, int fila)
{
	for(i=0; i<16; i++)
	{
		escribeCadenaPos_LCD(cadena, fila, i);
		Delay(200);
	}
}

//hace que el texto se desplace hacia la izquierda hasta quedar completo en pantalla
void escribeCadenaScrollI(char *cadena, int fila)
{
	for(i=15; i>-1; i--)
	{
		escribeCadenaPos_LCD(cadena, fila, i);
		Delay(200);
	}

}
```

ANEXO 4: EJEMPLOS DE CÓDIGO: FRAGMENTO DE UN PROGRAMA DESARROLLADO PARA COMPROBAR EL FUNCIONAMIENTO DE TODOS LOS PERIFÉRICOS

A continuación se muestra parte del contenido de la función main de este programa, que permite comprobar lo sencillo e intuitivo que resulta controlar los periféricos mediante las funciones desarrolladas:

```
int main(void)
{

        //se inicializan las variables
         int  pwm=100, dato=0; char j=-1, cmot=-1,cir=-1, calt=-1;



          RCC_Configuration();
          Init_GPIOs ();
          Config_Systick();
          EXTI15_10_Config();


          /* Switch on the leds at start */
          GPIO_HIGH(LD_GPIO_PORT,LD_GREEN_GPIO_PIN);
          GPIO_HIGH(LD_GPIO_PORT,LD_BLUE_GPIO_PIN);

          Delay(1000);

          GPIO_LOW(LD_GPIO_PORT,LD_GREEN_GPIO_PIN);
          GPIO_LOW(LD_GPIO_PORT,LD_BLUE_GPIO_PIN);
          GPIO_LOW(GPIOB, GPIO_Pin_5);

          borra_LCD();

          borra_matrizLEDs();
          iniciaLCD();
          iniciaTeclado();
          iniciaMatrizLEDs();
          iniciaTim3Audio();
          //iniciaPWM();
          RCC_Configuration();
          Inicia_I2C();
           /* NVIC configuration -------------------------------------------------------*/
           NVIC_Configuration();
                  Inicia_I2C();


                    escribeCadenaPos_LCD(" DEMO STM32 ", 0, 0);
                    Delay(1000);          // retardo de 1 seg
                    escribeCadenaPos_LCD("  3   ", 1, 4);
                    Delay(1000);
                    escribeCadenaPos_LCD("  2   ", 1, 4);
                    Delay(1000);
                    escribeCadenaPos_LCD("  1   ", 1, 4);
                    Delay(1000);
                    escribeCadenaPos_LCD("  0   ", 1, 4);
                    Delay(1000);
```

```
/* Infinite loop */
while (1)
{
	//inicializa la variable
	ctrl_IR=-1;

	borra_LCD();
	escribeCadenaPos_LCD("PULSA UNA TECLA:", 0, 0);
	Delay(50);
	//espera a que se pulse una tecla
	j=esperaTecla();
	//según el valor de la tecla, reacciona de forma diferente
	switch(j)
	{
	case '1':
	{
		//en este caso se reproducen las notas musicales al pulsar diferentes teclas
		//con una duracion de 0.5 segundos
		borra_LCD();
		escribeCadenaPos_LCD("ALTAVOZ(1/2 seg)", 0, 0);
		escribeCadenaPos_LCD("A: salir", 1, 3);
		calt=-1;
		while(calt!='A')//mientras no se pulse la tecla de salida
		{
			calt=esperaTecla();
		switch(calt)
		{
		case '1':
		{
			//inicia ambos temporizadores
			iniciaTim3Audio();
			iniciaTim2Temp();
			borra_LCD();
			escribeCadenaPos_LCD("DO", 0, 4);
			escribeCadenaPos_LCD("A: salir", 1, 3);
			//envía la señal correspondiente a la
			//nota DO durante 500 ms
			enviaNota(500, DO);break;
		}
		case '2':
		{
			iniciaTim3Audio();
			iniciaTim2Temp();
			borra_LCD();
			escribeCadenaPos_LCD("RE", 0, 4);
			escribeCadenaPos_LCD("A: salir", 1, 3);
			enviaNota(500, RE);break;}
		case '3':
		{
			iniciaTim3Audio();
			iniciaTim2Temp();
			borra_LCD();
			escribeCadenaPos_LCD("MI", 0, 4);
			escribeCadenaPos_LCD("A: salir", 1, 3);
			enviaNota(500,MI);
			break;
		}
		case 'F':
		{
			iniciaTim3Audio();
			iniciaTim2Temp();
			borra_LCD();
			escribeCadenaPos_LCD("FA", 0, 4);
			escribeCadenaPos_LCD("A: salir", 1, 3);
			enviaNota(500,FA);
```

```
                break;
        }
        case '4':
        {
                iniciaTim3Audio();
                iniciaTim2Temp();
                borra_LCD();
                escribeCadenaPos_LCD("SOL", 0, 4);
                escribeCadenaPos_LCD("A: salir", 1, 3);
                enviaNota(500,SOL);
                break;
        }
        case '5':
        {
                iniciaTim3Audio();
                iniciaTim2Temp();
                borra_LCD();
                escribeCadenaPos_LCD("LA ", 0, 4);
                escribeCadenaPos_LCD("A: salir", 1, 3);
                enviaNota(500,LA);
                break;
        }
        case '6':
        {
                iniciaTim3Audio();
                iniciaTim2Temp();
                borra_LCD();
                escribeCadenaPos_LCD("SI", 0, 4);
                escribeCadenaPos_LCD("A: salir", 1, 3);
                enviaNota(500,SI);
                break;
        }
        case 'E':
        {
                iniciaTim3Audio();
                iniciaTim2Temp();
                borra_LCD();
                escribeCadenaPos_LCD("DO'", 0, 4);
                escribeCadenaPos_LCD("A: salir", 1, 3);
                enviaNota(500,DO1);
                break;
        }
        case '7':
        {
                iniciaTim3Audio();
                iniciaTim2Temp();
                borra_LCD();
                escribeCadenaPos_LCD("RE'", 0, 4);
                escribeCadenaPos_LCD("A: salir", 1, 3);
                enviaNota(500,RE1);
                break;
        }
        default:
        {
                borra_LCD();
                escribeCadenaPos_LCD("tecla erronea", 1,0);
                Delay(1000);
                break;
        }
        }

/*SE HA ELIMINADO PARTE DEL CÓDIGO PARA NO EXTENDERSE DEMASIADO */

        case 'F':
```

```
		{			//en este caso se demuestra cómo se puede controlar el
					//motor de CC mediante modulacion PWM
			cmot=-1;
			borra_LCD();escribeCadenaPos_LCD("MOTOR 100%    ", 0, 0);
			ArrancarPWM(100);
			escribeCadenaPos_LCD("F:+/C:-/A:salir", 1, 0);
			//mientras el valor de la tecla no sea el de la tecla de salida:
			while(cmot!='A')
			{
				//se espera a que se pulse una tecla
				cmot=esperaTecla();
				//si es una F, se aumenta el ciclo de trabajo
				if(cmot=='F'){pwm=pwm+10;}
				//si es una C se disminuye el ciclo de trabajo
				else if(cmot=='C'){pwm=pwm-10;}
				//si no es F, C, ni A, la tecla no es válida
				else if(cmot!='A')
				{
					escribeCadenaPos_LCD("TECLA ERRONEA", 0, 0);
					Delay(1000);
				}
				//como el ciclo no puede ser mayor del 100%
				if(pwm>100)
				{
					pwm=100;escribeCadenaPos_LCD("ciclo max: 100%", 0, 0);
					Delay(1000);
				}
				//no menor del 0%
				else if(pwm<0)
				{
					pwm=0;escribeCadenaPos_LCD("ciclo min: 0%", 0, 0);
					Delay(1000);
				}
				borra_LCD();
				iniciaPWM();
				//escribe en el display el % del ciclo de trabajo
				escribeCadenaPos_LCD("MOTOR       ", 0, 0);
				escribeEnteroPos_LCD(pwm, 0, 6);
				escribeCadenaPos_LCD("%", 0, 11);
				escribeCadenaPos_LCD("F:+/C:-/A:salir", 1, 0);
				//acciona el motor segun el ciclo de trabajo configurado
				ArrancarPWM(pwm);
			}
			//una vez fuera del bucle, se detiene el motor
			PararPWM();
			break;
		}

/*SE HA ELIMINADO PARTE DEL CÓDIGO PARA NO EXTENDERSE DEMASIADO */

		case '7':
		{
			//en este caso se activa el motor de CC mediante un mando infrarrojo

			//se inicializan las variables
			cir=-1;
			ctrl_IR=1;
			//se activan las interrupciones del sensor de IR
			iniciaSensorIR();
			//mientras no se pulse el botón de salida:
			while(cir!='A')
			{
				//escribe informacion por pantalla
				borra_LCD();
				escribeCadenaPos_LCD("IR ACTIVA MOTOR", 0, 0);
```

```
                        escribeCadenaPos_LCD("A: SALIR", 1, 0);
                        Delay(500);
                        //y espera a que se pulse el boón de salida
                        cir=esperaTecla();
                    }
                    //desactiva las interrupciones el sensor de IR
                    desactivaSensorIR();
                    break;
                }

            default:
            {
                    borra_LCD();
                    escribeCadenaPos_LCD("tecla erronea", 1,0);
                    Delay(1000);
                    break;
            }
        }

/*SE HA ELIMINADO PARTE DEL CÓDIGO PARA NO EXTENDERSE DEMASIADO */

        }
        return 0;
    }
```

ANEXO 5: FRAGMENTO DEL MANUAL DEL DISPLAY LCD:

Table 2 Instructions

Instruction	Code										Description	Execution time (when fosc is 250 kHz) Note 1	Execution time (when fosc is 160 kHz) Note 2
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0			
Clear display	0	0	0	0	0	0	0	0	0	1	Clears all display and returns the cursor to the home position (Address 0).	82 μs ~ 1.64 ms	120 μs ~ 4.9 ms
Return home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (Address 0). Also returns the display being shifted to the original position. DD RAM contents remain unchanged.	40 μs ~ 1.6 ms	120 μs ~ 4.8 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and specifies or not to shift the display. These operations are performed during data write and read.	40 μs	120 μs
Display ON/ OFF control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of all display (D), cursor ON/OFF (C), and blink of cursor position character (B).	40 μs	120 μs
Cursor and display shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing DD RAM contents.	40 μs	120 μs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL) number of display lines (L) and character font (F).	40 μs	120 μs
Set CG RAM address	0	0	0	1	A_{CG}						Sets the CG RAM address. CG RAM data is sent and received after this setting.	40 μs	120 μs
Set DD RAM address	0	0	1	A_{DD}							Sets the DD RAM address. DD RAM data is sent and received after this setting.	40 μs	120 μs
Read busy flag & address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	1μs	1 μs
Write data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	40 μs	120 μs
Read data to CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40 μs	120 μs
	I/D = 1: Increment (+1) I/D = 0: Decrement (−1) S = 1: Accompanies display shift. S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right. R/L = 0: Shift to the left. DL = 1: 8 bits DL = 0: 4 bits N = 1: 2 lines N = 0: 1 line F = 1: 5 x 10 dots F = 0: 5 x 7 dots BF = 1: Internally operating BF = 0: Can accept instruction										DD RAM: Display data RAM CG RAM: Character generator RAM A_{CG}: CG RAM address A_{DD}: DD RAM address Corresponds to cursor address. AC: Address counter used for both of DD and CG RAM address.	Execution time changes when frequency changes. (Example) When fosc is 270 kHz: $40\ \mu s \times \frac{250}{270} = 37\ \mu s$	

*No effect

Notes 1. Applied to models driven by 1/8 duty or 1/11 duty.
2. Applied to models driven by 1/16 duty.

BIBLIOGRAFÍA:

Libros y documentos utilizados:

[1] Javier García de la Fuente, José Ignacio Rodríguez, Rufino Goñi, Alfonso Brazález, Patxi Funes y Rubén Rodríguez. “Aprenda lenguaje C como si estuviera en Primero”, Abril 1998.

[2] Joseph Yiu,”The Definitive Guide to the ARM Cortex-M3”, Elsevier, 2007.

[3] Juan Vázquez Martínez, “Manual de prácticas de sistemas electrónicos digitales II: 2º curso de Ingeniería Técnica Industrial : Electrónica Industrial “, 2006.

[4] Juan Vázquez Martínez, “Sistemas electrónicos digitales basados en microprocesador ARM7”, 2009.

[5] Michael Victorio García Lorenz, “Ejercicios con ARM7 y entorno [m]Vision”, 2009.

[6] Paul Roukema, “ARM7TDMI vs. ARM Cortex-M3 Microcontroller Cores”, Marzo 2010.

[6] User Manual UM1079 - STM32L-DISCOVERY. Rev 2, Junio 2011.

[7] Reference Manual RM0038 - STM32L151xx and STM32L152xx advanced ARM-based 32-bit MCUs. Rev 6, Julio 2012.

[8] Programming Manual PM0063 - STM32L151xx, STM32L152xx and STM32L162xxFlash and EEPROM programming. Rev 5, Marzo 2012

[9] Datasheet DS6876 - STM32L151xx, STM32L152xx. Rev 6. Enero 2012.

[10] Aplication Note AN3422 - Migrating a microcontroller application from STM32F1 to STM32L1 series. Rev 2, Marzo 2012.

[11] Application Note AN3216 - Getting started with STM32L1xxx hardware development. Rev 5. Junio 2011.

[12] Jorge Caballero Escribano, “Creación de un entorno de desarrolla para aplicaciones basadas en microcontroladores STM32L Cortex-M3”, Proyecto Fin de Carrera, Octubre 2011.

[13] Judith Liu Jiménez, Susana Patón, Juan Vázquez, Raúl Sánchez Reillo, Rodrigo Manzanares Bolea, Michael García Loren, “Manual de la Placa de Periféricos Sistemas Electrónicos Digitales”, Octubre 2006.

[14] Juan Vázquez, “Placa ES de Entrada/Salida para Practicas con MicroControlador”, Octubre 2006.

Páginas web visitadas:

-www.eclipse.org

g-www.chibios.org

- http://forum.chibios.org

-github.com/texane

-dangerousprototypes.com

- http://sistemasembebidos.com.ar/

- http://embeddednewbie.blogspot.com.es

- http://ziblog.ru

- http://www.yagarto.de

- http://www.page.sannet.ne.jp/kenjia

- http://www.rcgroups.com

- http://www.seng.de

- http://www.freertos.org

- http://www.stf12.org

- http://www.st.com

- http://www.mikrocontroller.net/

- http://www.arm.com

- http://www.stm32circle.com

-www.coocox.org

- http://www.versaloon.com/