

UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIONES
SISTEMAS DE TELECOMUNICACIONES



PROYECTO FINAL DE CARRERA

CREACIÓN DE UN ENTORNO DE DESARROLLO PARA
APLICACIONES BASADAS EN MICROCONTROLADORES
STM32L CORTEX-M3

AUTOR: JORGE CABALLERO ESCRIBANO
TUTOR: MICHAEL GARCÍA LORENZ

Octubre de 2011

TÍTULO: *CREACIÓN DE UN ENTORNO DE DESARROLLO PARA APLICACIONES BASADAS EN MICROCONTROLADORES SMT32L CORTEX-M3.*

AUTOR: *JORGE CABALLERO ESCRIBANO*

TUTOR: *MICHAEL GARCÍA LORENZ*

La defensa del presente Proyecto Fin de Carrera se realizó el día 28 de Octubre de 2011; siendo calificada por el siguiente tribunal:

PRESIDENTE:

SECRETARIO

VOCAL

Habiendo obtenido la siguiente calificación:

CALIFICACIÓN:

Presidente

Secretario

Vocal

Agradecimientos

Quiero agradecer este proyecto de fin de carrera a mi tutor Michael, por su ayuda, paciencia, y respuesta cuando lo he necesitado.

Quisiera agradecer no sólo este proyecto sino también toda mi carrera a mis padres, por la educación que me han dado, sin la cuál no sería quien soy, gracias. También me acuerdo de todos mis familiares que encendían velas en mis exámenes, en especial a mis abuelos Félix y María, mi espejo en la vida. Y no puedo olvidarme de mi primo Rubén, que me ha enseñado, ha compartido estudios conmigo y hemos conseguido una gran amistad.

No puedo olvidarme de mi otra familia la cuál ha sido un soporte fundamental en estos años, en concreto todos aquellos que han compartido tardes en la biblioteca, Christian, Patri, Victor, Rubén, etc. Pero en especial me gustaría agradecer la ayuda incondicional de Raúl, gracias por tu apoyo, comprensión y tus charlas en la calle, y por preguntarme un día, si quería ser ingeniero o instalador de antenas.

También quiero recordar mis compañeros de clase, e@synet, con los cuales he pasado muy buenos momentos y el estudio fue menos doloroso gracias a su compañía, y gracias a ellos me voy de la universidad con una amistad para siempre.

Mi más sincero agradecimiento a mis compañeros de Indra en Aranjuez, en especial a Carlos por haberme enseñado a no ser un ingenierito y a ser mejor persona.

Y por último a Andoni, mi mejor apoyo en estos últimos meses de carrera y tener la paciencia que tienes conmigo. Gracias por ayudarme en los malos momentos y compartir los buenos.

Índice general

1. INTRODUCCIÓN	21
1.1. PROBLEMAS	21
1.2. OBJETIVOS	22
2. ESTADO DE LA TÉCNICA	23
2.1. LPC2129 y μ Vision de Keil	23
2.1.1. LPC2129	24
2.1.2. μ Vision de Keil	24
2.2. ARDUINO	25
2.2.1. PLACA ARDUINO	26
2.2.2. ARDUINO IDE	27
2.2.3. Programar el microcontrolador ATmega directamente.	28
2.3. THE MAPLE	29
2.4. LPCXPRESSO	31
2.4.1. LPCXpresso IDE	31
2.4.2. LPCXpresso HW	31
2.4.3. LPC1769	33
2.5. STM32L DISCOVERY	35
2.5.1. IDEs	39

3. SOLUCIÓN ESCOGIDA	45
3.1. ELECCIÓN DE LA PLACA	45
3.2. ELECCIÓN DEL TOOLCHAIN	46
3.3. CORTEX M3	48
3.3.1. INTRODUCCIÓN	48
3.3.2. VENTAJAS DEL CORTEX-M3	50
3.4. CMSIS	51
3.4.1. ¿QUÉ ES CMSIS?	51
3.4.2. CAPAS DE SOFTWARE	51
3.4.3. CMSIS PARA UN CORTEX	52
3.5. STM32L152RBT6	53
3.6. LIBRERÍAS	55
3.6.1. INTRODUCCIÓN	55
3.6.2. CONTENIDO DE LA LIBRERÍA	56
3.6.3. NOMENCLATURA	58
3.6.4. DESCRIPCIÓN DE LOS FICHEROS DE LA LIBRERÍA	59
3.6.5. ¿CÓMO UTILIZAR LA LIBRERÍA?	61
4. GUÍA DE PERIFÉRICOS	65
4.1. RESET AND CLOCK CONTROL (RCC)	65
4.1.1. Relojes	65
4.2. GENERAL PURPOSE I/Os (GPIO)	68
4.3. SYSTEM TIMER (SysTick)	71
4.3.1. CONFIGURACIÓN	71
4.4. DISPLAY LCD	73

4.4.1. Reloj del LCD	73
4.4.2. CONFIGURACIÓN	73
4.5. EXTERNAL INTERRUPT (EXTI) y NVIC	76
4.6. ANALOG-TO-DIGITAL CONVERTER (ADC)	80
4.6.1. CONFIGURACIÓN	81
4.7. TIMER	87
4.7.1. Configuración Input Capture	88
4.7.2. Configurar Output Compare	92
4.7.3. External Trigger	94
4.8. PULSE WIDTH MODULATOR (PWM)	95
4.9. DIGITAL-TO-ANALOG CONVERTER (DAC)	97
4.9.1. Onda Triangular	100
4.9.2. Onda senoidal	102
5. EVALUACIÓN	107
5.1. GPIO	107
5.1.1. ¿CÓMO UTILIZAR LA APLICACIÓN?	108
5.1.2. DESCRIPCIÓN DEL SOFTWARE	108
5.1.3. CÓDIGO DE LA APLICACIÓN	109
5.2. INTERRUPCIÓN EXTERNA	110
5.2.1. ¿CÓMO UTILIZAR LA APLICACIÓN?	110
5.2.2. DESCRIPCIÓN DEL SOFTWARE	110
5.2.3. CÓDIGO DE LA APLICACIÓN	112
5.3. ADC	113
5.3.1. ¿CÓMO UTILIZAR LA APLICACIÓN?	114

5.3.2. DESCRIPCIÓN DEL SOFTWARE	115
5.3.3. CÓDIGO DE LA APLICACIÓN	116
5.4. TIMER	117
5.4.1. ¿CÓMO UTILIZAR LA APLICACIÓN?	118
5.4.2. DESCRIPCIÓN DEL SOFTWARE	118
5.4.3. CÓDIGO DE LA APLICACIÓN	119
5.5. PWM	120
5.5.1. ¿CÓMO UTILIZAR LA APLICACIÓN?	120
5.5.2. DESCRIPCIÓN DEL SOFTWARE	120
5.5.3. CÓDIGO DE LA APLICACIÓN	122
5.6. DAC: ONDA TRIANGULAR	123
5.6.1. ¿CÓMO UTILIZAR LA APLICACIÓN?	123
5.6.2. DESCRIPCIÓN DEL SOFTWARE	124
5.6.3. CÓDIGO DE LA APLICACIÓN	125
5.7. DAC: ONDA SENOIDAL Y SAWTOOTH	126
5.7.1. ¿CÓMO UTILIZAR LA APLICACIÓN?	127
5.7.2. DESCRIPCIÓN DEL SOFTWARE	127
5.7.3. CÓDIGO DE LA APLICACIÓN	128
6. CONCLUSIONES	129
7. TRABAJO FUTURO	131
8. PRESUPUESTO DEL PROYECTO	133
8.1. Fases del Proyecto	133
8.2. Coste del Personal Empleado	133

8.3. Coste del material empleado	134
8.4. Coste del Software empleado	134
8.5. Presupuesto Total	135
APÉNDICES	139
A. ATOLLIC TRUESTUDIO	139
A.1. INSTALACIÓN	139
A.2. PRIMEROS PASOS	146
A.2.1. Creando un nuevo proyecto	147
A.2.2. Compilando el proyecto	149
A.2.3. Modo Debug	151
A.3. Gestionando Proyectos	155
B. ECLIPSE	161
C. MAPEADO DE PINES	171

Lista de Figuras

2.1. <i>Entorno de trabajo</i>	23
2.2. <i>Arduino Uno</i>	26
2.3. <i>The Maple - Leaf Labs</i>	29
2.4. <i>LPCXPRESSO IDE</i>	32
2.5. <i>LPCXPRESSO BOARD</i>	32
2.6. <i>STM32L-DISCOVERY</i>	36
3.1. <i>Estructura de CMSIS</i>	52
3.2. <i>Diagrama de bloques de STM32L152RBT6</i>	54
3.3. <i>STM32L1xx Standard Peripherals Library</i>	56
3.4. <i>Visión Global</i>	59
3.5. <i>Relación Librería</i>	60
4.1. <i>Onda Triangular</i>	98
4.2. <i>Onda Senoidal</i>	102
5.1. <i>Potenciómetro conectado a placa STM32L-DISCOVERY</i>	113
A.1. <i>Descarga Atollic TrueSTUDIO Lite</i>	139
A.2. <i>Instalación Atollic TrueSTUDIO Lite</i>	140
A.3. <i>Generación Clave PC</i>	140
A.4. <i>Registro Web Atollic</i>	141

A.5. <i>Registro Completo</i>	141
A.6. <i>Clave Atollic TrueSTUDIO Lite</i>	142
A.7. <i>Instalación ST-Link</i>	142
A.8. <i>Instalación Completa</i>	143
A.9. <i>Cable USB</i>	143
A.10. <i>Instalacion ST-Link Dongle</i>	144
A.11. <i>Descarga ST-Link V2</i>	144
A.12. <i>Instalación ST-Link Utility</i>	145
A.13. <i>Ventana ST-Link Utility</i>	145
A.14. <i>ST-Link Upgrade</i>	146
A.15. <i>Actualización Completa</i>	146
A.16. <i>Selección Workspace</i>	147
A.17. <i>Perspectiva C/C++</i>	148
A.18. <i>Creando un nuevo proyecto</i>	149
A.19. <i>Editor de Texto</i>	150
A.20. <i>Errores de Compilación</i>	151
A.21. <i>Build Automatically</i>	151
A.22. <i>Compilando el proyecto</i>	152
A.23. <i>Run Debug</i>	152
A.24. <i>Debug Configuration</i>	153
A.25. <i>Debug Perspective</i>	154
A.26. <i>Controles Modo Debug</i>	154
A.27. <i>Insertar Breakpoint</i>	155
A.28. <i>Step Into y Step Over</i>	156
A.29. <i>Add Watch Expression</i>	157

A.30. <i>Finalizar Modo Debug</i>	157
A.31. <i>Varios Proyectos Abiertos</i>	158
A.32. <i>Close Project</i>	158
A.33. <i>Open Project</i>	159
B.1. <i>Descarga Eclipse for C/C++ Developers</i>	161
B.2. <i>Sourcery ++ Lite for ARM EABI</i>	162
B.3. <i>Selección Workspace</i>	163
B.4. <i>Pantalla Bienvenida Eclipse</i>	163
B.5. <i>Pantalla Principal Eclipse</i>	164
B.6. <i>Añadir fuente CDT</i>	164
B.7. <i>Añadir fuente GNU ARM Eclipse</i>	165
B.8. <i>Instalación de nuevas fuentes</i>	165
B.9. <i>Instalación de nuevas fuentes</i>	165
B.10. <i>Nuevo Proyecto C</i>	166
B.11. <i>Proyecto Template</i>	167
B.12. <i>Botón Apply</i>	167
B.13. <i>Preprocessor GCC Assembler</i>	168
B.14. <i>Directories GCC Assembler</i>	168
B.15. <i>Preprocessor GCC C Compiler</i>	168
B.16. <i>Directories GCC C Compiler</i>	169
B.17. <i>Optimization GCC C Compiler</i>	169
B.18. <i>General GCC C Linker</i>	170
B.19. <i>Compilación Correcta</i>	170
C.1. <i>Conectores en STM32L-Discovery</i>	171

Lista de Tablas

4.2. <i>Amplitudes Señal Triangular</i>	101
4.4. <i>Valores digitales y analógicos</i>	103
8.2. <i>Fases del proyecto junto con el tiempo aproximado empleado.</i>	134
8.4. <i>Presupuesto del proyecto: coste del personal empleado en el proyecto.</i>	134
8.6. <i>Coste del material empleado en el proyecto.</i>	134
8.8. <i>Coste del software empleado en el proyecto</i>	135
8.10. <i>Coste total.</i>	135
C.2. <i>Pines en el conector 1</i>	172
C.4. <i>Pines en el conector 2</i>	173

Resumen

El siguiente proyecto plantea un nuevo escenario de desarrollo para aplicaciones de microcontroladores enfocado a un entorno académico. El principal objetivo es la utilización de microcontroladores de 32 bits para el aprendizaje de programación de microcontroladores. El handicap del proyecto es la búsqueda y sustitución de hardware de alto coste por un nuevo hardware de bajo coste, altas prestaciones y que utilice herramientas GNU.

El hardware seleccionado es la placa de STMicroelectronics STM32L-Discovery que tiene un microprocesador STM32L152RBT6 de 32 bits basado en el ARM Cortex-M3. El IDE destinado al desarrollo de estas aplicaciones es Eclipse y ATollie TrueSTUDIO. En las diferentes etapas de este proyecto, se realiza primero un estudio de las características de la placa de desarrollo, así como de ambos IDEs. A continuación presenta una guía de configuración de los principales periféricos de esta placa. Al final, se realizan varias aplicaciones para evaluar la funcionalidad de la placa y la compatibilidad con un entorno de enseñanza.

INTRODUCCIÓN

La motivación de este proyecto es la búsqueda de una alternativa al entorno de prácticas en una asignatura de enseñanza de programación de microcontroladores. Para realizar un estudio de la situación actual, partimos de lo que ya conocemos y esto es la placa MCB2100 la cuál incluye el microcontrolador LPC2129 de NXP. Para el desarrollo de aplicaciones con esta placa se utiliza el IDE μ Vision de Keil y adaptador JTAG externo ULINK. Partiendo de este entorno, realizaremos un estudio y definiremos qué es lo que queremos.

1.1. PROBLEMAS

Gracias a la experiencia de estos años empleando el mismo entorno, conocemos cuáles son algunos de los principales problemas a la hora de desarrollar con esta configuración. A continuación, presentamos cuáles son estos inconvenientes:

- Elevado precio. El precio de una placa MCB2100 es de 217 €(unitario). A esto tenemos que sumarle el precio del adaptador USB JTAG ULINK que es de 26 €.
- Requiere una placa de periféricos externos de E/S para interactuar con el código. Si deseamos trabajar simplemente con la placa y el adaptador JTAG ULINK tendremos problemas para ver resultados tangibles de fácil acceso.
- El IDE para trabajar con ella es propietario de KEIL. Tenemos acceso a una versión gratuita o de evaluación pero tiene una limitación de 6 meses de uso y una limitación de código de 32 KB. Existe la posibilidad de comprar una licencia pero tiene un precio bastante elevado. Para un entorno académico donde un estudiante puede utilizar la placa durante más de 6 meses, supone un problema tener que estar renovando la licencia gratuita, por ejemplo, si estas versiones están instaladas en laboratorios.
- El IDE no es multiplataforma, y sólo está disponible para S.O. Windows. No es posible desarrollar bajo Linux, o Mac OS.
- El conjunto del equipo para desarrollar es bastante grande e imposibilita el poder llevarlo de un lado a otro para poder trabajar tanto en un laboratorio como trabajar desde casa.

- La placa MCB2100 está descatalogada y algunas de las actualizaciones del IDE afectan al funcionamiento de aplicaciones ya programadas en la placa.

1.2. OBJETIVOS

Una vez analizado el entorno en el que partimos, nuestro objetivo es la búsqueda de una nueva tarjeta con un microcontrolador que sea como mínimo de similares características al actual o superior. Ya que vamos a cambiar el microcontrolador debemos estudiar en el mercado qué tarjetas hay que se adapten a nuestros requisitos.

- Esta nueva tarjeta debe suponer un cambio progresivo al modelo de estudio que teníamos. Estábamos trabajando con ARM7, por lo tanto deberíamos buscar algo similar. No es viable un cambio a PIC o AVR ya que el salto de modelo de programación es bastante considerable.
- Debe ser low-cost. Es muy interesante que la tarjeta pueda ser adquirida por cualquier estudiante o grupo. Este hecho facilita que pueda ser adquirida a bajo precio y con ello se pueda trabajar en casa.
- Sus dimensiones deben de ser reducidas.
- Es interesante que posea elementos E/S para su interacción con la aplicación que haya descargada en la placa.
- Debe ser capaz de programarse bajo herramientas GNU tales como compiladores GCC o debuggers GDB.

Este análisis de la situación actual nos hace plantearnos proponer una nuevo entorno de desarrollo donde no existan estos problemas. Para ello, es necesario la redacción de unos objetivos. Una vez establecidos, podremos realizar un estudio en el mercado de qué placas se adaptan mejor a nuestro problema y elegir la más conveniente.

ESTADO DE LA TÉCNICA

En este capítulo se mostrará el estudio realizado a diferentes tarjetas de microcontroladores que hay actualmente en el mercado. Analizaremos cada una de ellas, observando sus ventajas y desventajas. También observaremos sus IDEs y sus especificaciones técnicas.

2.1. LPC2129 y μ Vision de Keil

El hardware del que disponemos está formado por:

- MCB2100: Placa de desarrollo de Keil que incorpora un LPC2129 de NXP que permite crear y testear programas. Se conecta al PC a través de una interfaz USB JTAG. (Figura 2.1)
- ULINK USB-JTAG: Adaptador DEBUG que conecta el puerto USB del PC con la tarjeta a través de JTAG y permite depurar aplicaciones embebidas.
- Placa de Periféricos de E/S: incluye un teclado matricial, un LCD, un sensor de infrarrojos, un altavoz, y periféricos adicionales.
- PC con S.O. Microsoft Windows instalado.

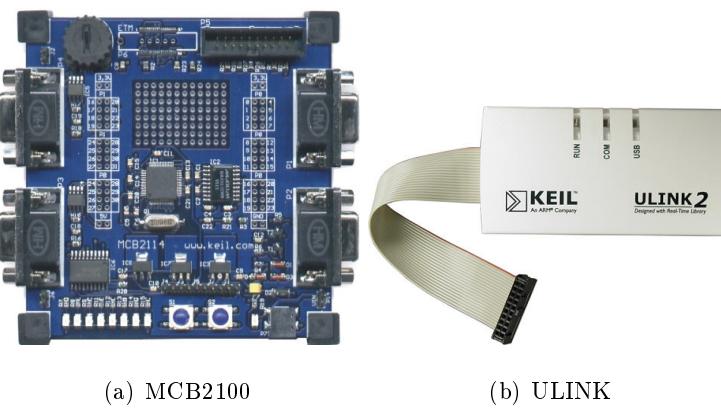


Figura 2.1: *Entorno de trabajo*

El software del que disponemos está compuesto por:

- Keil μ Vision: Es el IDE del fabricante de la placa MCB2100, KEIL SOFTWARE. Este IDE nos permite la creación de proyectos, ensamblado y simulación, y empleando el adaptador

USB JTAG, se descargan programas a la placa y debugger.

2.1.1. LPC2129

La tarjeta dispone de las siguientes características:

- Microcontrolador: LPC2129 de 32 bits
- 256 KB de memoria RAM
- 16 KB memoria SRAM
- Reloj 60 Mhz
- 46 I/O Digitales
- Vectored Interrupt Controller VIC
- 2 TIMERS de 32 BITS
- PWM, Watchdog, y RTC
- ADC 10 bits de resolución
- 2 UARTs
- I^2C 400 kbps
- 2 interfaces SPI

2.1.2. μ Vision de Keil

El μ Vision de Keil es un SW de la compañía Keil. Es propietario. Está disponible en versión evaluación (6 meses con limitación de código a 32 KB) o también se pueden comprar licencias. Permite simulación, y debugger.

Estas razones son algunas de las que han motivado la búsqueda de una solución. Debemos buscar una placa de coste mucho más bajo, que no requiera de herramientas externas para su programación, y que sea capaz programarse con compiladores GCC, para evitar estar ligados a un IDE propietario.

La búsqueda de esta solución comenzó con el OPEN HARDWARE, y uno de sus máximos exponentes, ARDUINO.

2.2. ARDUINO

La primera idea es aproximarnos al movimiento Open Source Hardware. Éste tiene un gran impacto en la actualidad tecnológica y está cambiando el panorama empresarial. Uno de los mayores exponentes del Open Source Hardware es la plataforma Arduino que fue nuestra primera placa de estudio.



Arduino es una plataforma Open Source Hardware de desarrollo para microcontroladores AVR ATmega. Es una placa sencilla de E/S y un entorno de desarrollo que implementa el lenguaje PROCESSING. Arduino puede ser usada para desarrollar aplicaciones sencillas y sus tarjetas pueden ser compradas ya ensambladas o ensamblarlas a mano. El IDE de Arduino es Open Source y puede ser descargado gratuitamente desde su página web.

Algunos de sus puntos fuertes son los siguientes:

- Arduino es multiplataforma, puede correr en Windows, Linux o Mac OS.
- Está basado en el lenguaje PROCESSING, muy fácil de usar por desarrolladores y aficionados al hobby de la electrónica.
- Se programa a través de USB, no requiere de puerto serie.
- Arduino es Open Source hardware y software, por lo que podemos descargar los planos y fabricar nuestras propias placas Arduino sin tener que pagar a sus creadores.
- La placa es de bajo coste, tan sólo 20€ y se puede reemplazar su microcontrolador por tan sólo 5€.
- Está soportada por una comunidad online bastante activa, por lo que es bastante sencillo obtener ayuda.
- El Proyecto Arduino ha sido pensado para entornos educativos, y es adecuada para iniciarse en el mundo de los microcontroladores.

Arduino se encuentra en constante evolución y sus placas cada vez cuentan con mejores prestaciones. Al comienzo del proyecto las placas eran de comunicación serie disponiendo de un puerto RS-232 para su comunicación externa con el PC (versión Diecimilla). A día de hoy la placa más avanzada de Arduino es la UNO (Figura 2.2), que es la última versión en el mercado y es a partir de la cuál se va a continuar desarrollando el proyecto. En ella desaparece la conexión serie y se comunica por USB. La placa Arduino es escalable a través de Shields que amplían su potencial.

Muchas Shields están disponibles para añadirlas a nuestra placa UNO. Entre las más destacadas y que mayor funcionalidad ofrecen están modulos de GPS, Bluetooth, RFID, XBee, etc.

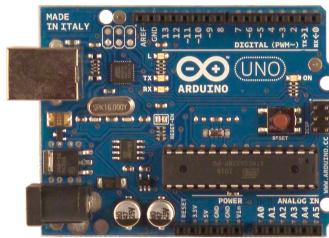


Figura 2.2: *Arduino Uno*

2.2.1. PLACA ARDUINO

La placa Arduino UNO posee un ATmega 328. Sus características más importantes son:

- Microcontrolador: ATmega328 de 8 bits
- 32 KB memoria Flash (0.5 KB son utilizados por el bootloader)
- 2 KB memoria SRAM
- 1 KB EEPROM
- Reloj 16 Mhz
- 14 I/O Digitales (de los cuales 6 pueden ser PWM)
 - 1 Com. Serie: conectados al ATmega USB-to-TTL

- 2 Interrupciones Externas
 - PWM 8 bits
 - 4 pines comunicación SPI
 - 1 LED
- 6 pines analógicos con 10 bits de resolución
 - I^2C utilizando WireLibrary

Arduino dispone de un bootloader o gestor de arranque pregrabado para realizar la descarga de código a la placa sin necesidad de disponer de un programador externo. Sin embargo, también es posible la programación mediante un ISP externo utilizando los pines adecuados para ellos que están disponibles en la placa.

2.2.2. ARDUINO IDE

La última versión del IDE es la 0022. Es un IDE escrito en JAVA y es multiplataforma. Tiene licencia GPL (General Public License) lo que significa que es de libre distribución, modificación y uso. Está basado en Processing, y avr-gcc.

Ventajas para nuestro proyecto:

1. Arduino es una placa de muy bajo coste (22€), Open Source Hardware por lo que podemos ensamblarla nosotros mismos.
2. Su IDE es Open Source, por lo tanto podemos descargarlo gratuitamente y programar sin limitaciones.
3. Es muy portable y de dimensiones reducidas. Un grupo de estudiantes puede disponer de su propia placa y trabajar en casa con ella, escribir código y volcarlo directamente para ver su funcionamiento, sin necesidad de simular. Ante cualquier fallo, el micro es muy económico de sustituir.
4. No es necesario un programador, se carga el código a través del USB gracias al bootloader.
5. No dispone de un puerto de comunicaciones serie, pero si virtualiza uno gracias al VIRTUALCOM.

6. Fuerte comunidad online, y continuamente actualizándose el firmware de Arduino, su IDE y sus prestaciones.

Desventajas para nuestro proyecto:

1. Para utilizar el entorno de desarrollo de Arduino (su IDE) se escribe en PROCESSING que es un lenguaje de muy alto nivel y se abstrae mucho de la filosofía de C utilizada con anterioridad. Esto no permite conocer cómo funciona exactamente el microcontrolador.
2. Su IDE es poco potente, no tiene muchas funcionalidades.
3. No dispone de debugger, por lo tanto, no se puede depurar el código.
4. El Hardware de Arduino es de menores prestaciones que el LPC2129, estaríamos utilizando micros de 8 bits en lugar de 32 bits.
5. No disponemos de periféricos para E/S, es necesario crear una placa de periféricos.

Debido a este estudio no creemos que sea muy favorable sustituir el actual entorno por este nuevo. Sin embargo, es posible una alternativa.

2.2.3. Programar el microcontrolador ATmega directamente.

Para poder programar el microcontrolador directamente es necesario disponer de un programador como el AVR-USB-JTAG que implementa el protocolo JTAG-ICE de Atmel. Además necesitaríamos un nuevo IDE ya que el de Arduino no es viable. Como primera alternativa planteamos AVR-Studio.

Si a tal efecto esto ocurriera, sería necesario la compra de material adicional para poder funcionar adecuadamente dicho entorno de trabajo, con lo cual el coste total del material aumentaría.

Al mismo tiempo, sería un coste alto para poder trabajar con una herramienta de bajo nivel como es un microcontrolador de 8 bits.

El problema resulta que con este paso estaríamos alejándonos de la filosofía de open source. Si utilizásemos un programador para acceder directamente al microcontrolador de Atmel, Arduino perdería todo el sentido, su lenguaje, su IDE y su filosofía.

Por lo tanto esta opción queda descartada.

2.3. THE MAPLE

Tras ARDUINO, las especificaciones empezaban a ser más claras:

- El HW debe de ser similar o superior al que ya disponemos.
- Debe ser una plataforma de bajo coste.
- Debe ser una plataforma pequeña, con alta portabilidad.
- Debe tener un IDE gratuito.
- Debe tener la capacidad de Debugger.

Buscando HW similar a Arduino pero superior, se llegó a MAPLE LEAFS. La placa MAPLE (Figura 2.3) posee un microprocesador de 72Mhz ARM Cortex M3. La compañía LeafLabs (Open Electronics) distribuye esta placa con un micro muy potente en el mercado y lo acerca al mundo educacional implementando un entorno de programación muy intuitivo y familiar con Arduino.

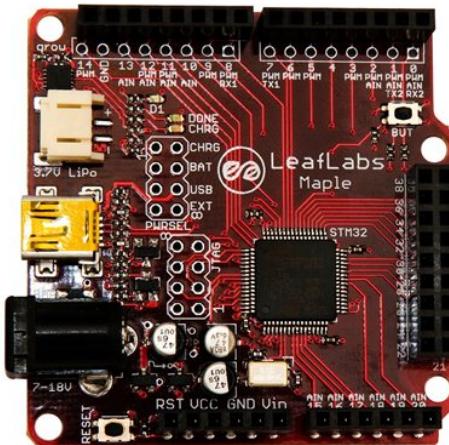


Figura 2.3: *The Maple - Leaf Labs*

La placa está basada en un microprocesador STM32F103RB de 32 bits.

- Microprocesador: STM32F103RB

- 128 KB memoria Flash
- 20 KB memoria SRAM
- Reloj 72 Mhz
- 39 I/O Digitales
- 3 USARTs
- 64 Canales de NVIC
- USB nativo
- Dos Comunicación SPI
- I^2C 400 kbps
- 4 TIMERs de 16 bits
- 16 pines analógicos
- PWM 16 bits
- A/D converter 12 bits

El sistema de programación está basado en Arduino, es Open Source. El chip viene programado con un bootloader permitiendo descargar el código utilizando la interfaz USB sin necesidad de un programador externo. También se puede programar el microcontrolor directamente como sucedía con el AVR de Arduino utilizando una interfaz JTAG.

Ventajas:

1. Mantenemos la filosofía Open Source Hardware, pero cambiando el microcontrolador por uno más potente.
2. Utilizamos un ARM por lo tanto, no existe tanto salto tecnológico.
3. Es un ARM Cortex M3 que es uno de los más puntero del mercado.

Desventajas:

1. El IDE no depura.
2. Necesidad de un hw externo para depurar y programar el microcontrolador.
3. Es más costoso que Arduino.

2.4. LPCXPRESSO

LPCXpresso es una plataforma de desarrollo de muy bajo coste desarrollada por NXP y Code Red Technologies. El proyecto consiste en dos partes: Software y Hardware.

2.4.1. LPCXpresso IDE

El IDE (Figura 2.4) consiste en una versión mejorada de ECLIPSE, un compilador GNU de C, linker, librerías y un mejorado debugger GDB. Este software ha sido desarrollado por Code Red Technologies (www.code-red-tech.com/lpcxpresso/), y está basado en la plataforma de desarrollo Eclipse a la que se le ha incluido mejoras para LPC. El compilador es GNU, utiliza una librería de C optimizada. La única limitación del IDE es una restricción de código de 128 kB, aunque con actualizaciones de pago del IDE ésta puede ser anulada.

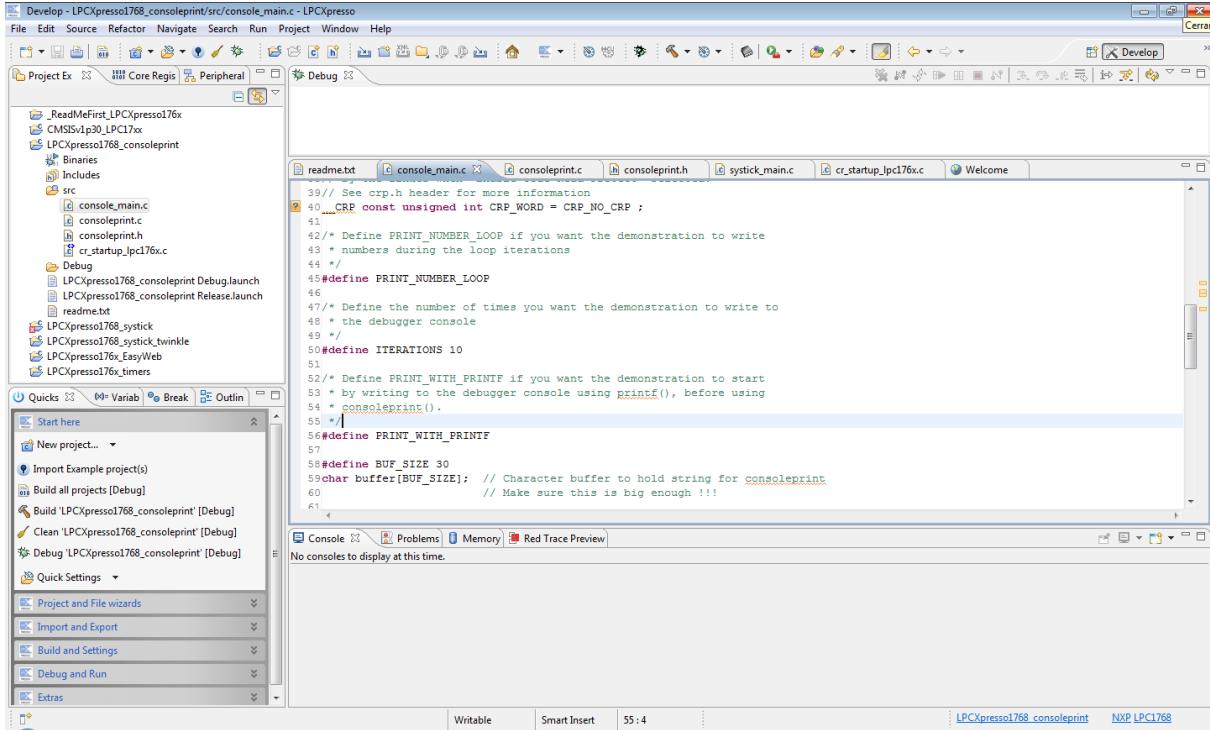
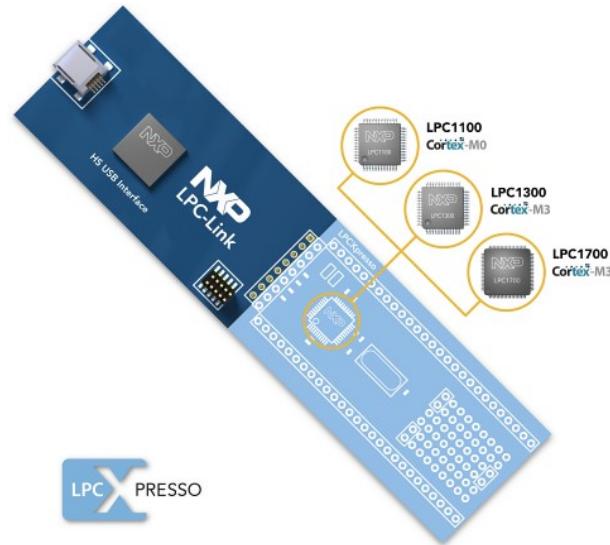
2.4.2. LPCXpresso HW

La tarjeta de LPCXpresso (Figura 2.5) ha sido desarrollada en conjunto por NXP, Code Reed Technologies y Embedded Artist (<http://www.embeddedartists.com/products/lprcspresso/>). Incluye un JTAG debugger (LPC-Link) por lo que no hay necesidad de un JTAG externo. El LPC-Link debugger provee conexión USB a la JTAG/SWD interfaz en el IDE, y además puede ser configurado para trabajar como un programador externo para otras tarjetas.

Las familias que están disponibles son:

- LPC1100: ARM Cortex M0
- LPC1300: ARM Cortex M3
- LPC1700: ARM Cortex M3

Podemos ampliar la funcionalidad de esta tarjeta conectándola a una placa de periféricos como la desarrollada por EmbbededArtist, y explorar todas las posibilidades que ofrece.

Figura 2.4: *LPCXPRESSO IDE*Figura 2.5: *LPCXPRESSO BOARD*

Con una sola tarjeta podemos desarrollar código gracias a la MCU y el IDE de LPCXpresso. Podemos descargar el código y probar su funcionamiento en la placa de periféricos. Una vez que

esté finalizado el proceso de desarrollo y pruebas de la aplicación, utilizando su programador, podemos descargar el código en cualquier tarjeta compatible con JTAG. De este modo cubrimos todo el proceso de creación y desarrollo de la aplicación.

LPC-LINK JTAG/SWD Debugger

El LPCXpresso contiene un JTAG/SWD debugger conocido como LPC-LINK. El LPC-LINK tiene una cabecera JTAG de 10 pines y se conecta vía USB a la tarjeta. Es posible hacer funcionar por sí solo al LPC-LINK independientemente de la tarjeta, y convertirlo en un programador individual.

2.4.3. LPC1769

La placa de LPCXpresso elegida para el estudio ha sido el modelo LPCXpresso LPC1769 de 32 bits

Características:

- Microprocesador: LPC1769FBD100
- 512 KB memoria Flash
- 64 KB memoria SRAM
- Reloj 120 Mhz
- 70 I/O pines Digitales
- 4 USARTs
- 64 Canales de NVIC
- Comunicación SPI
- 3 I^2C hasta 1 Mbit/s
- I^2S
- 4 TIMERS de 16 bits
- 6 PWM 16 bits

- ADC converter 12 bits
- DAC converter 10 bits
- Dos canales CAN
- Ethernet MAC
- USB interface Decive/Host/OTG
- Quadrature Encoder Interface

Aplicaciones:

- eMeeting
- Iluminación
- Redes Industriales
- Sistemas de Alarma
- Electrodomésticos
- Control de motores

Ventajas:

1. Es un ARM Cortex M3, que es de lo mejor del mercado actual, un micro muy potente y de bajo consumo.
2. Implementaciones propias de ARM: MPU, NVIC, WIC, DMA...
3. 15 interfaces serie: SSPI, UARTs, CAN, I2C, I2S, SPI, Ethernet, USB, TIMERS, PWM, RTC,
4. Migración fácil desde ARM7 que es de dónde venimos.
5. La filosofía de programación es la misma ya que seguimos trabajando bajo un LPC, antes LPC2129 y ahora LPC1769.

6. La propia empresa nos facilita gratuitamente el IDE de manera gratuita con una única limitación de 128 KB.
7. Soporte online y foros sobre LPCXpresso.
8. Muy bajo coste, tan sólo 25€.
9. Su IDE está basado en Eclipse.
10. Es muy pequeña, portable, y de fácil conexión, mediante USB.
11. No requiere programador externo.

Desventajas:

1. No tiene periféricos E/S para interactuar con la placa, como pulsadores, leds, lcd... Si deseamos la placa de periféricos de EmbbededArtists, es demasiado cara.
2. Proporcionan un IDE gratuito pero corremos el riesgo de que limiten el uso cualquier momento.

2.5. STM32L DISCOVERY

El kit de desarrollo STM32L-DISCOVERY (Figura 2.6) es una tarjeta bajo coste con capacidad para desarrollar, evaluar y crear proyectos con un STM32. Está basado en un STM32L152RBT6 y incluye un ST-Link/V2 (interfaz debugger), un LCD de 24 segmentos, LEDs, pulsadores y un sensor lineal.

El kit de desarrollo tiene las siguientes características:

- Microcontrolador STM32L152RBT6 de 32 bits, 128 KB de memoria Flash, 16 KB de RAM, 4 KB de EEPROM.
- Dispone de ST-Link/V2, programador y debugger, con capacidad de funcionar con autonomía.
- Alimentación 3.3V o 5V a través de USB o una fuente externa
- LCD: DIP28, 24 segmentos y 4 comunes



Figura 2.6: *STM32L-DISCOVERY*

- Cuatro LEDs:
 - LD1(rojo/verde): indica comunicación USB
 - LD2(rojo): indicador de alimentación
 - Leds de usuario: LD3 (verde) y LD4 (rojo).
- Dos pulsadores (reset y de usuario)
- Un sensor lineal y cuatro teclas táctiles
- Medidor de Corriente Idd

La familia de MCU STM32L están basadas en el ARM Cortex M3 y poseen juego de instrucciones Thumb-2. Combinan un alto rendimiento con un bajo consumo, utilizando una arquitectura avanzada.

La familia STM32L15xxx compatibiliza la capacidad de conexión de USB con el alto rendimiento del ARM Cortex M3 de 32 bits, trabajando a un reloj de 32 Mhz, con Unidad de Protección de Memoria (MPU), memorias embebidas de alta velocidad (memoria flash hasta 128

KB y RAM hasta 16 KB), y una gran diversidad de I/Os y periféricos conectados a dos buses APB.

Entre sus características más avanzadas destaca: 12 bit ADC, 2 DAC y dos comparadores, 6 TIMERs de propósito general de 16 bit y dos básicos. En cuanto a interfaces de comunicación, el STM32L15XXX incorpora dos I^2C y SPIs, tres USARTs y USB. Además incluyen RTC y registro de control del LCD.

Trabaja en un rango de 1.8V a 3.6V y en un rango de temperatura de -40 a +85 °C

El microcontrolador que incorpora este kit de desarrollo es el STM32L152RBT6 que posee un ARM Cortex M3 de 32 bits. Sus principales funcionalidades son:

- ARM Cortex M3 de 32 bits 130nm
- 128 KB memoria Flash
- 16 KB memoria RAM
- 4 KB memoria EEPROM
- Reloj 32 Mhz
- Interrupciones NVIC
- 83 I/O pines Digitales (73 de los cuales son tolerantes a 5V). Todos ellos mapeables a 16 canales de interrupciones externas.
- 6 TIMERs de 16 bits, 2 TIMERs básicos, y 2 Watchdog TIMERs.
- ADC de 12 bits de resolución con sensor de temperatura
- 2 DAC de 12 bits
- 3 USARTs
- 2 interfaces SPI (16 Mbps)
- 2 I^2C (SMBus / PMBus)
- Interfaz USB 2.0

- Controlador LCD 8x40 o 4x44
- Soporte de desarrollo por SWD o JTAG.

El precio unitario de este producto es de: 15€ aproximadamente.

STMicroelectronics no facilita un IDE propietario para el desarrollo de sus productos sin embargo si recomienda algunas suites de desarrollo de terceras empresas. Los IDE que recomiendan son: MDK-ARM, IAR Embedded Workbench y Atollic TrueSTUDIO.

Ventajas de esta placa:

1. Tal y como buscábamos es una placa más actual que la que disponíamos y de mejores prestaciones.
2. Tiene un ARM Cortex M3 de 32 bits.
3. Es de muy bajo coste, tan sólo 15€ por ser una versión de evaluación.
4. Incorpora un programador en la propia tarjeta SWD/JTAG ST-LINK/V2, por lo que no hace falta ninguna herramienta externa.
5. Es bastante portátil y fácil de transportar.
6. Incorpora un LCD, botones de usuario, y sensor lineal, por lo tanto las capacidades de interacción con el código se multiplican.
7. Implementaciones propias de ARM: MPU, NVIC, WIC, DMA...
8. Interfaces serie: UARTs, I2C, SPI, USB, TIMERS, PWM, RTC etc.
9. Migración fácil desde ARM7 que es de donde venimos.

Desventajas de esta placa:

1. Existe un salto de programación desde el LPC2129 al STM32L152xxx, que es un valor añadido al aprendizaje por parte del estudiante.
2. Es un microcontrolador mucho más potente y complejo que el LPC2129, y esto implica que los periféricos requieren más conocimientos para configurar sus registros.

3. No nos facilitan un propio IDE y tenemos que recurrir software de terceros que por lo general son de pago y nos facilitan versiones de evaluación y limitadas.

2.5.1. IDEs

MDK-ARM

MDK-ARM es un software de desarrollo para Windows que combina un robusto y moderno editor con un gestor de proyectos. Integra todas las herramientas necesarias para el desarrollo de aplicaciones embebidas, incluyendo un compilador de C/C++, macro assembler, linker, y debugger. Esta suite de desarrollo también es conocida como μ Vision de Keil.

Este software es el mismo que se estuvo utilizando para el desarrollo con la placa MCB2100 y su microcontrolador LPC2129. Como es lógico, aunque cambiemos de microcontrolador, las limitaciones que la versión gratuita nos ofrece son las mismas que en el caso anterior. La versión actual es la 4.20.03.0. Esta versión incluye soporte para ST-LINK/V2.

Ventajas:

1. Contiene todo lo necesario para trabajar: compilador, linker, debugger etc.
2. Es compatible en su totalidad con ST-Link V2.

Desventajas:

1. No es multiplataforma, solamente desarrollo en Windows.
2. Requiere licencia, aunque también está disponible una versión gratuita por una duración de 6 meses con limitación de 32 KB.
3. Requiere registro para obtener la licencia gratuita.

IAR Embedded Workbench

IAR Embedded Workbench for ARM es una software de desarrollo para Windows. Integra una plataforma de desarrollo con un gestor de proyectos. Reúne herramientas como compilador de aplicaciones en C/C++, macro assembler, linker y generador de ficheros HEX. Está optimizado para C y C++ para ARM.

La versión actual de este producto es la 6.21.1.2846

Existe una versión de pago aunque también existe una versión "kickstar" gratuita previo registro en su web. Esta edición gratuita tiene bastantes limitaciones como un límite de código de 32 KB, limitada en herramientas de desarrollo, y sin debugger.

Ventajas:

1. Compatible con ST-Link/V2.

Desventajas:

1. Requiere licencia. Disponible versión gratuita pero no incorpora debugger.
2. Muy pocas funcionalidades.
3. No es multiplataforma. Solamente se encuentra disponible en Windows.

Atollic TrueSTUDIO

Atollic True Studio es software de desarrollo basado en Eclipse que incluye un compilador de C/C++, un editor del estado del arte, y un debugger profesional. Incluye todas las herramientas necesarias para desarrollar aplicaciones embebidas. Algunas de sus principales características son un editor de código muy potente, gestor de proyectos, consola MS/DOS integrada, herramientas de programación flash para descargar la aplicación en la ROM, debugger, etc.

Este producto cuenta con diferentes versiones en función de la MCU que utilicemos. Encontraremos versiones disponibles para Atmel AVR UC3, Atmel AT91SAM, Texas Instruments Stellaris, STMicroelectronics STM32 y Toshiba TX.

En nuestro caso, el interés cae sobre la versión para STM32. Esta versión cuenta con soporte para toda la familia de microcontroladores STM32 y para kit de desarrollos como nuestro STM32L DISCOVERY. También cuenta con soporte para ST-Link V2 así como ejemplos.

Un punto fuerte de este IDE reside en el hecho de contar auto generador de proyectos para los diferentes tipos de microcontroladores que deseemos usar en nuestro proyecto. De esta forma, cuando iniciemos un nuevo proyecto, el software nos preguntará por la placa que vamos a utilizar para la aplicación, y tras esto, configurará todos los parámetros del programa para generar

un template para compilar y depurar. Otro factor importante es el hecho de estar basado en ECLIPSE y utilizar compiladores, y herramientas GNU.

Se encuentra disponible en una versión de pago y otra gratuita. Pero en esta suite, la diferencia respecto a otros IDEs que hemos visto es que la versión gratuita no tiene limitación de código ni de tiempo de uso. Las limitaciones se encuentran en el número de breakpoints a la hora de utilizar el debugger, y que antes de arrancar el debugger aparece una ventana para actualizar el producto. La versión gratuita del producto requiere registro para obtener la licencia gratuita.

Ventajas:

1. Basado en Eclipse y utiliza herramientas GNU.
2. Optimizado para STM32, su asistente de proyectos configura automáticamente la mejor configuración para nuestro modelo de placa.
3. La versión gratuita (Lite) no contiene limitaciones de código ni de uso.
4. Debugger profesional.

Desventajas:

1. El debugger sólo permite un breakpoint.
2. Mensaje de “publicidad” cada vez que hacemos uso de la herramienta debugger.
3. La versión gratuita no permite escribir proyectos en C++.
4. No es multiplataforma.

ECLIPSE + GCC + GDB

Por último reside la opción más efectiva a la hora de realizar un proyecto de bajo coste, y es el empleo de aplicaciones GNU y herramientas GNU. En este caso la aplicación escogida es el IDE Eclipse en su versión para desarrollo de aplicaciones C/C++. Eclipse es un software GNU por lo que no es necesario ningún tipo de licencia y podemos descargarnos su versión completa. Además Eclipse es multiplataforma por lo que una vez instalado y configurado para el desarrollo

para STM32, podemos trabajar tanto en Windows, como Linux, como en Mac OS. También cabe destacar que es muy configurable para cualquier dispositivo y tenemos a disposición una amplia gama de plugins para aumentar su potencial.

A Eclipse hay que sumarle compiladores y linkers GNU como el proyecto GCC, CodeSourcery, ARM EABI, etc.

El valor añadido que tiene ECLIPSE es que también es portable por lo que no es necesario instalarlo. Basta con descargarlo y configurarlo. Podemos copiar la carpeta a un pendrive y trabajar con nuestro IDE personalizado en cualquier parte.

Una vez ECLIPSE es configurado adecuadamente para nuestro tarjeta, no debemos preocuparnos por actualizaciones del producto puesto que sabemos que esa configuración es válida para desarrollar aplicaciones para nuestra tarjeta.

Algunos de los toolchain que podemos utilizar para Eclipse:

1. GNU ARM toolkit: disponible para Linux, Mac OS y Windows (bajo CygWin). Es un toolchain gratuito bajo licencia GNU.
2. CodeSourcery G++ para ARM EABI: disponible para Linux y Windows. Es un compilador de línea de comandos gratuito.
3. GCC: Linux, Mac OS y Windows.

El principal problema de este entorno de trabajo es el debugger que trae incorporado la tarjeta, el ST-Link V2. El protocolo del ST-Link V2 no está abierto por lo que solamente los productos de terceros mencionados anteriormente tienen capacidad para utilizarlo. Estos productos son exclusivamente MDK-ARM, IAR Embedded Workbench, y Atollic True Studio. Por ello no podemos configurar el ST-Link V2 en Eclipse ni otro IDE que no sea alguno de esos tres.

El protocolo de ST-Link V2 se basa en almacenamiento masivo y sólo acceso de alto nivel. El problema es que el ST-Link no tiene acceso de bajo nivel JTAG o SWD.

Dos posibles soluciones:

1. Generar un fichero compilado con Eclipse y grabarlo en la memoria de la placa con la ayuda de la herramienta externa ST-LINK Utility. Sin embargo, esto sólo nos permite descargar el código, y no depurarlo.

2. Utilizar un programador externo JTAG en el que podamos comunicarnos a través de Eclipse. No obstante, esto implica una inversión mayor de dinero en el proyecto, y no nos interesa aumentar los costes.

SOLUCIÓN ESCOGIDA

En la anterior sección realizamos un estudio tanto de las placas de desarrollo como de los toolchain. En esta sección indicaremos cuál es la solución escogida y cuáles son los motivos para ello.

3.1. ELECCIÓN DE LA PLACA

La placa elegida es el modelo **STM32L-DISCOVERY**. Esta placa es la que mejor se adapta a los requisitos planteados al principio de este proyecto. Los principales motivos por los que se ha elegido este modelo y no otro se detallan a continuación:

- De todas las placas analizadas, es la que mejor prestaciones ofrece y la más potente. Posee un ARM Cortex-M3 de 32 bits. La placa Arduino poseía un ATmega de tan sólo 8 bits. Partíamos de una situación en la que disponíamos de una placa con un LPC2129 con una procesador ARM7TDMI-S de 32 bits. Era inviable plantear una solución en la que el modelo fuese de inferior potencia que el antiguo. Respecto al resto de modelos estudiados, tanto The Mapple como LPCXpresso también tenían procesadores de ARM Cortex-M3. Además, al tratarse de un ARM Cortex M-3, cuenta con implementaciones propias como NVIC, MPU, DMA etc. Por lo tanto descartamos la opción de Arduino.
- La tarjeta incorpora un programador SWD/JTAG ST-Link V2 implementado en la propia placa. No es necesario ningún programador externo como en nuestro anterior entorno de trabajo, donde necesitábamos del Keil Ulink. El ST-Link/V2 se conecta al PC mediante un cable USB, por lo que la conexión es muy sencilla. La placa Arduino y The Mapple no disponían de debugger. Para cargar el código utilizaban un bootloader. Sin embargo, LPCXpresso incorporaba un programador en la propia placa al igual que el STM32L-DISCOVERY. Este hecho facilita que el estudiante pueda trabajar desde casa sin necesidad de ningún hardware externo adicional. Además, el coste unitario del entorno de trabajo no se eleva, ya que por lo general, estas herramientas externas son bastante caras.
- Gran cantidad de periféricos E/S incorporados en la propia placa. Con STM32L-Discovery no necesitamos una placa externa de periféricos para poder interactuar con la aplicación ya

que incorpora periféricos muy sencillos y de bastante utilidad como dos leds, un botón de usuario, un display LCD y un sensor lineal táctil. Como estamos viendo, las prestaciones entre The Mapple, LPCXpresso y STM32L-DISCOVERY son muy similares. Sin embargo, las dos primeras no poseen periféricos de E/S como STM32L-DISCOVERY, por lo que deberíamos comprar hardware externo o diseñar una placa de periféricos nosotros mismos.

- Precio de la placa. El precio por unidad de trabajo tan sólo es de 15€ que es el precio del dispositivo. Al no necesitar programador externo, ni placa de periféricos, sólo necesitamos nuestra placa y un PC para diseñar, desarrollar y descargar el código. El precio de las restantes placas, aún siendo también económicos, superan los 15€, como Arduino con 22€ o LPCXpresso con 33€ .

3.2. ELECCIÓN DEL TOOLCHAIN

La elección del Toolchain está ligada a la elección de la placa de desarrollo. Al habernos decantado por STM32L-DISCOVERY, el abanico de toolchains se estrecha. Los criterios de selección son los mismos que en la elección de la placa, abaratar costes. Para ello la primera opción es utilizar Eclipse configurado con compiladores GCC y herramientas de depuración GDB. Con esta configuración estaremos utilizando Eclipse que posee licencia pública, y compiladores y debuggers GNU. Esto implica utilizar un software que no tiene ningún coste.

El objetivo es conseguir una plataforma de desarrollo completamente funcional y que nos permita desarrollar sin ninguna limitación las aplicaciones que desarrollemos. Para ello utilizaremos la versión “Eclipse for C/C++ Developers” que es la que necesitamos para desarrollar en lenguaje C. Luego necesitaremos de diversos plugins para poder utilizarlo para desarrollar para STM32L-DISCOVERY.

Como complemento adicional a Eclipse necesitaremos instalar “Sourcery G++ Lite for ARM EABI”. Este software es gratuito. Sourcery G++ Lite incluye todo lo necesario para el desarrollo de aplicaciones, incluidos compiladores en C y C++, ensambladores, linkers y librerías. En concreto contiene:

- CodeSourcery Common Startup Code Sequence
- CodeSourcery Debug Sprite for ARM

- GNU Binary Utilities (Binutils)
- GNU C Compiler (GCC)
- GNU C++ Compiler (G++)
- GNU C++ Runtime Library (Libstdc++)
- GNU Debugger (GDB)
- Newlib C Library

Como plugins para Eclipse, necesitaremos de los siguientes:

- GNU ARM C/C++ Development Suppor
- C/C++ Development Tools
- C/C++ Development Platform
- C/C++ DSF GDB Debugger Integration
- C/C++ GDB HW Debugger
- GNU Toolchain Build Support
- GNU Toolchain Debug Support
- P2 C/C++ Toolchain Installer

Algunos de estos plugins ya vienen instalados en la versión C/C++ Developers de Eclipse, mientras que otros deberemos descargarlos e instalarlos mediante el asistente de instalación de nuevo software de Eclipse.

En el Anexo B viene detallado el proceso de instalación y configuración de Eclipse, así como el de sus plugins. En este anexo también se muestra como configurar un template y compilar un proyecto.

Pero este nuevo entorno tenía un inconveniente. Como se comentó en un principio al analizar este toolchain, el protocolo ST-Link/V2 no está abierto por lo que no se ha podido integrar un debugger GDB funcional en esta configuración de Eclipse. Ante tal problema, optamos por utilizar el IDE de Atollic TrueSTUDIO. Los motivos son los siguientes:

- Atollic TrueSTUDIO está basado en Eclipse. Toda su plataforma como se podrá comprobar es exactamente igual que Eclipse. La compañía Atollic ha cogido una distribución de Eclipse y la ha configurado para que sea operativa al cien por cien con las placas STM32.
- Atollic TrueSTUDIO utiliza compiladores, ensambladores y linkers GNU, por lo que se mantiene la filosofía inicial.
- Este software es uno de los elegidos por STMicroelectronics para dar soporte al ST-Link/V2, al igual que otros IDEs como MDK-ARM, IAR, etc. Eso supone que tiene un debugger profesional configurado para el ST-Link/V2.
- Dispone de una versión gratuita sin limitaciones de uso, ni de código. La única limitación mas importante es la restricción de un único breakpoint. Esto no nos supone ningún problema puesto que el objetivo de este entorno de trabajo es el de aprendizaje.
- En un tiempo existirán herramientas de depuración GDB para el ST-Link/V2 compatibles en Eclipse. Llegado a ese día, la migración desde Atollic a Eclipse será muy sencilla ya que los dos son la misma plataforma.

En este proyecto se ha decidido presentar la configuración para Eclipse para poder desarrollar aplicaciones en C para el STM32. En el Anexo B se puede consultar este trabajo. Esto facilita que el día en que exista la posibilidad de integrar el ST-Link/V2 en Eclipse, ya existirá una configuración para desarrollar y la mitad del trabajo estará realizado, que sería configurar el IDE, y el aprendizaje de su uso (es el mismo que en Eclipse).

Ahora que tenemos planteada la solución, es de momento de conocer un poco más en detalle la placa en la que vamos a trabajar y algunas de sus facetas importantes.

3.3. CORTEX M3

3.3.1. INTRODUCCIÓN

Nuestra placa tiene un procesador ARM Cortex-M3. Antes de comenzar a ver el microcontrolador de STM32L-DISCOVERY, debemos estudiar las funcionalidades y capacidades de la CPU.

El procesador ARM Cortex-M3, el primero de la generación Cortex fue lanzado por ARM en 2006, y fue diseñado principalmente con destino al mercado de microprocesadores de 32 bits. El procesador ARM Cortex-M3 ofrece un rendimiento excelente en cuanto al bajo número de compuertas e incorpora muchas características nuevas anteriormente sólo disponibles en procesadores de gama alta.

El procesador Cortex-M3 afronta los requisitos actuales de mercado de la siguiente manera:

- **Gran eficiencia de rendimiento.** Esta eficiencia permite realizar más tareas sin aumentar los requisitos de frecuencia.
- **Bajo consumo energético.** Permite una mayor duración de batería especialmente en dispositivos portátiles.
- **Mejorado determinismo.** Esto garantiza que las tareas críticas e interrupciones puedan ser atendidas en el menor tiempo posible.
- **Mejorada densidad del código.** El código se ajusta a pequeñas porciones de memoria.
- **Facilidad de uso.** Programación y depuración mas fácil.
- **Soluciones de menor coste.** El precio de los microprocesadores de 32 bits se aproxima al de 8 y 16 bits.
- **Gran variedad de herramientas de desarrollo.** Disponibilidad de compiladores de bajo coste, software libre y grandes herramientas de desarrollo.

Debido a que los procesadores Cortex-M3 pueden ser escritos en lenguaje C, y están basados en una arquitectura adecuada, su código puede ser portado y reutilizado fácilmente, reduciendo de esta manera el tiempo de desarrollo.

El Cortex-M3 surge a partir del éxito del ARM7, mejorando su programación y depuración, así como ofreciendo mayor capacidad de procesamiento. Además ofrece mejoras tecnológicas como una unidad de protección de memoria, manipulaciones atómicas de bits, interrupciones no-enmascarables o vectores de interrupción anidados.

3.3.2. VENTAJAS DEL CORTEX-M3

Alto Rendimiento

- Muchas instrucciones como la multiplicación son de un solo ciclo.
- El bus de datos y de instrucción son independientes lo que permite realizar accesos simultáneos a código y datos.
- Set de instrucciones Thumb-2. No es necesaria cambiar entre estados ARM (32 Bits) y Thumb (16 bits), por lo que se reducen los ciclos de instrucción y el tamaño del programa.
- Gracias al juego de instrucciones Thumb-2, Cortex-M3 ofrece mayor densidad de código y reduce los requisitos de memoria.
- Cortex-M3 implementa relojes que operan a más de 100 Mhz. Además, tiene una relación reloj por instrucción (CPI) mejor que el resto de procesadores.

Funciones Avanzadas en el Manejo de Instrucciones

- El NVIC que incorpora (Nested Vectored Interrupt Controller) soporta hasta 240 interrupciones externas de entrada. La función interrupción vectorizada reduce la latencia de interrupción, no es necesario software para determinar el manejador de IRQ.
- El NVIC puede programar la prioridad de interrupción individualmente. Maneja hasta 8 niveles de prioridad.
- En la recepción de una NMI (NonMaskeable Interrupt) está garantizada la ejecución del manejador. Esto es importante en aplicaciones de seguridad críticas.

Bajo Consumo de Energía

- Es óptimo para diseños de bajo consumo debido al recuento de baja de la puerta.
- Soporta los modos de ahorro de energía SLEEPING y SLEEPDEEP. El procesador puede despertarse de ellos por interrupción o por evento.

Características del Sistema

- El sistema proporciona operación bit-band, acceso a datos no alineados y modo big-endian byte-invariante.
- MPU: Unidad de Protección de Memoria. Desarrollo de software más robusto y confiable.

Soporte Depuración

- Soporta JTAG y SWD.
- Soporte integrado para seis breakpoints y cuatro watchpoints.
- Coresight: podemos acceder al contenido de memoria cuando el núcleo está ejecutando.

3.4. CMSIS

3.4.1. ¿QUÉ ES CMSIS?

CMSIS (Cortex Microcontroller Software Interface Standard) es una capa de abstracción para todos los Cortex-M. CMSIS permite interfaces sencillas y consistentes para el procesador y periféricos, simplificando la reutilización de software y reduciendo la curva de aprendizaje para desarrolladores de aplicaciones para microcontroladores.

El desarrollo de software es conocido como un coste alto en la industria de software embebido. Mediante la estandarización de la interfaz software en los Cortex-M por los vendedores de silicio, este coste se reduce significativamente, especialmente cuando se crean nuevos proyectos o se migra software ya existente a nuevos dispositivos.

3.4.2. CAPAS DE SOFTWARE

CMSIS (ver Figura 3.1) consiste en los siguientes componentes:

- **Peripheral Register and Interrupt Definitions.** Una interfaz robusta para registros de dispositivo e interrupciones. Cada periférico consiste en una estructura que define los registros del periférico. Todas las interrupciones de un dispositivo son definidas robustamente conforme a CMSIS.

- **Core Peripheral Functions.** Proporciona un método de inicio del sistema y funciones para acceder a características específicas del procesador y funcionalidades del núcleo. El vendedor de silicio podría añadir funcionalidades específicas para su dispositivo.
- **DSP Library.** Algoritmos óptimos de procesamiento de señal y soporte para instrucciones SIMD del Cortex-M4.
- **System View Description (SVD).** fichero XML que describe los periféricos del dispositivo e interrupciones.

El estándar es escalable para asegurar que es viable para cualquier procesador Cortex-M desde el más pequeño con 8 KB hasta dispositivos más sofisticados con periféricos de comunicación como Ethernet o USB. Los requerimientos de memoria son de 1 KB de código y menos de 10 Bytes de RAM.

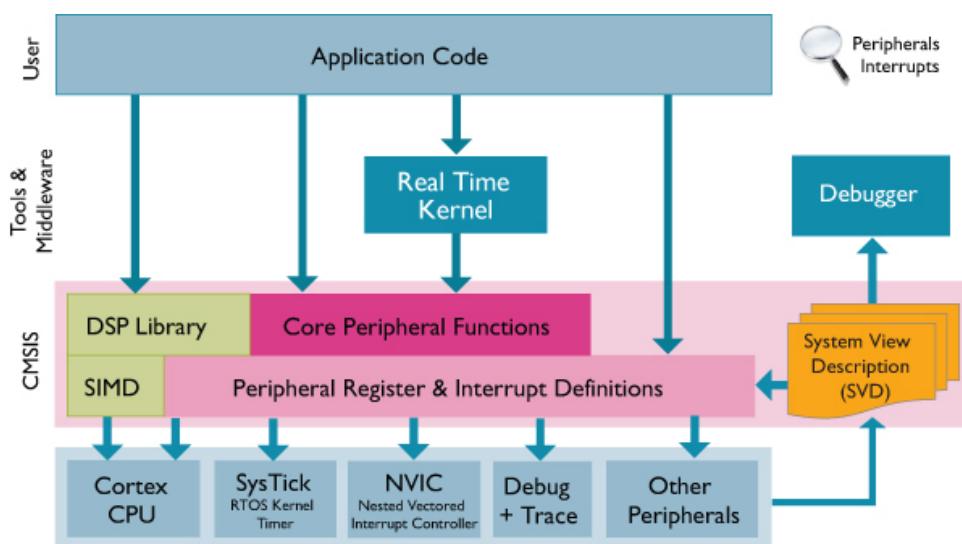


Figura 3.1: Estructura de CMSIS

Definido por ARM en cooperación con vendedores de Silicio y de Software. Entre los vendedores de silicio destacan por importancia: Actel, Atmel, NXP, STMicroelectronics, Texas Instruments y Toshiba. Entre los vendedores de software: Code_Red, Doulos, IAR Systems, Keil, etc.

3.4.3. CMSIS PARA UN CORTEX

CMSIS define:

- Una manera común de acceso a registros de periféricos del Core y de definir vectores de excepción.
- Los nombres de los registros de los periféricos del Core y de los vectores de excepción del mismo.
- Una interfaz independiente del dispositivo para RTOS Kernels incluyendo un canal de depuración.
- Mediante el uso de software CMSIS compatible, el usuario puede fácilmente reutilizar el código. CMSIS tiene por objeto permitir la combinación de componentes de software de múltiples proveedores de componentes.

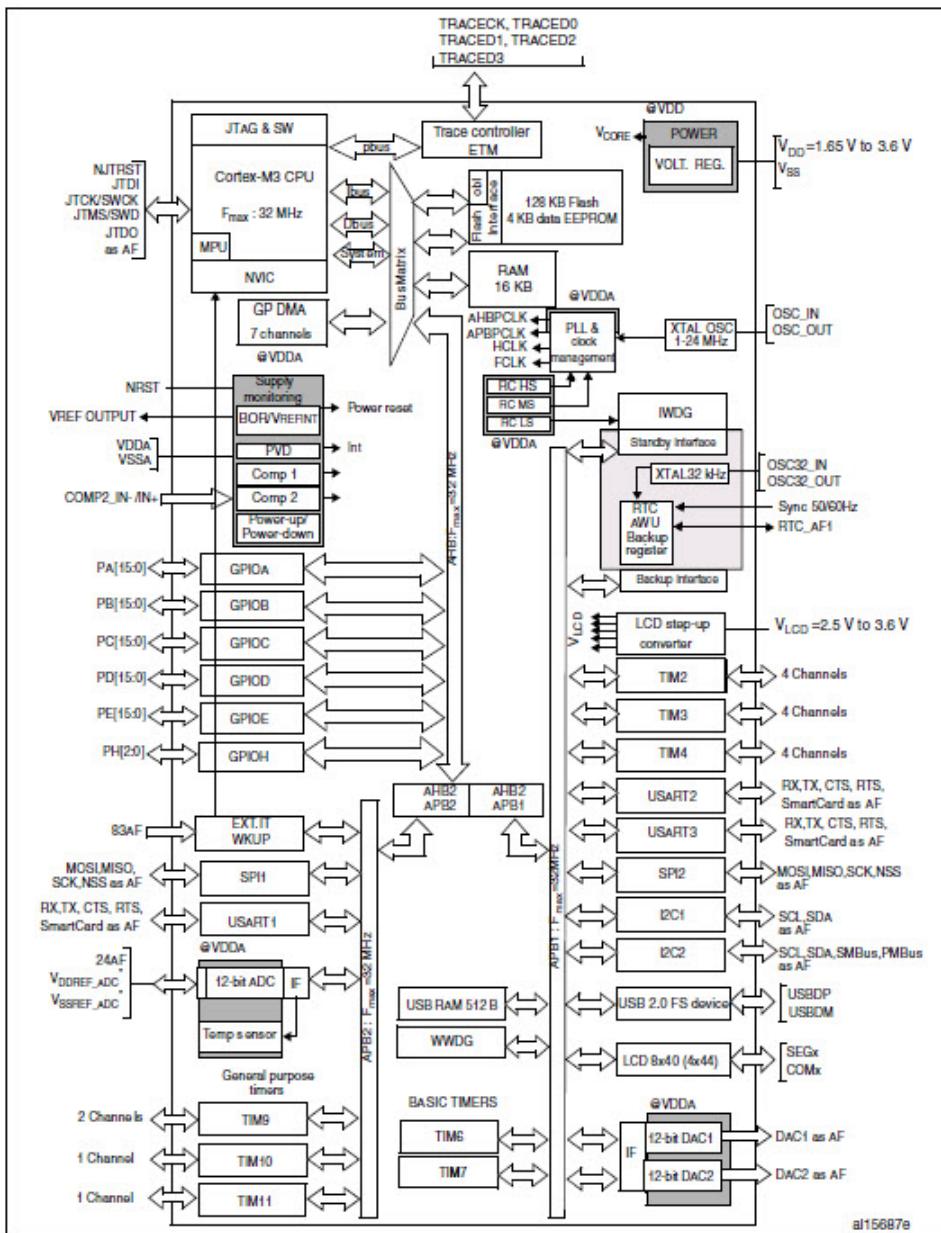
3.5. STM32L152RBT6

La placa STM32L-DISCOVERY incorpora un microcontrolador STM32L152RBT6 de 32 bits. Posee 128 KB de Flash, 16 KB de RAM, 4 KB de EEPROM, RTC, controlador LCD, timers, USART, I²C, SPI, ADC, DAC y Comparadores.

Esta MCU pertenece a la familia STM32L. Esta familia presenta un compromiso con el bajo consumo, perteneciendo a la plataforma EnergyLite. Combina una gran eficiencia con un bajo consumo dando lugar a un gran ahorro energético. Esta plataforma presentan MCUs con una tecnología de 130 nm. Las dos principales procesadores de esta familia son el STM32L152 y el STM32L151. La diferencia entre estos dos modelos es que el STM32L152 presenta un controlador LCD.

Esta familia esta pensada para las siguientes aplicaciones:

- Sistemas de Alarma.
- Telefonía Móvil.
- Sensores.
- Dispositivos Portátiles.
- Instrumentación médica.



1. AF = alternate function on I/O port pin.

Figura 3.2: Diagrama de bloques de STM32L152RBT6

El STM32L152 tiene 6 modos de baja potencia:

1. “Run Mode” a $230 \mu\text{A}/\text{MHz}$
2. “Run Mode” a $186 \mu\text{A}/\text{MHz}$
3. “Low-power Run Mode” a $10,4 \mu\text{A}/\text{MHz}$ a 32 kHz

4. "Low-power Sleep Mode" a $6,1 \mu\text{A}/\text{MHz}$ a $32 \text{ kHz} + 1$ timer
5. "Stop with/without RTC" a $1,3 \mu\text{A}/\text{MHz}$ / $0,43 \mu\text{A}/\text{MHz}$
6. "Standby with/without RTC" a $1,0 \mu\text{A}/\text{MHz}$ / $0,27 \mu\text{A}/\text{MHz}$

En la sección anterior ya se vieron los periféricos y prestaciones de cada uno de ellos. Sin embargo, se muestra a continuación el diagrama de bloques del microcontrolador en la (Figura 3.2)

3.6. LIBRERÍAS

3.6.1. INTRODUCCIÓN

STMicroelectronics facilita una librería descargable desde su web llamada **STM32L1xx Standard Peripherals library**. Esta cubre dos niveles de abstracción:

- Un completo mapeo de registros con todos los bits, campos de bits y registros declarados en lenguaje C. Esto evita conocer a fondo cada periférico ya que son de elevada complejidad y con esto se facilita el aprendizaje en fases iniciales.
- Colección de funciones y estructuras de datos que cubren todas las funcionalidades de los periféricos.

Cada driver consiste en un conjunto de funciones cubriendo todas las funcionalidades de periféricos. El desarrollo de cada driver está basado en un API común (Application Programming Interface) que estandariza la estructura del driver.

El código de los drivers ha sido escrito en C. Está documentado y es conforme a los requisitos MISRA-C 2004. La librería es independiente del toolchain empleado, solo los ficheros de inicio startup dependen del toolchain.

Nota: MISRA-C 2004 (Motor Industry Software Reliability Association's Guidelines for the use of the C language in critical systems). La programación en lenguaje C cada vez tiene mayor importancia y se encuentra en crecimiento para sistemas embebidos pero en lenguaje tiene muchos inconvenientes. Este lenguaje no es muy apropiado para el desarrollo de sistemas de seguridad críticos. MISRA-C 2004 ha especificado una normativa para el uso de C en sistemas críticos de seguridad.

Standar Peripherals Library también es conforme a CMSIS.

La libreria Standard Peripherals Library tiene implementada detección de fallo por paso de parámetros en todas las funciones. Ésto hace que el software sea más robusto.

La ventaja de utilizar esta librería reside en el tiempo que se necesita para comenzar a desarrollar aplicaciones para STM32L1xx. Gracias a ella, no necesitamos conocer en profundidad todas las facetas y posibilidades del periférico que deseamos programar. Sin embargo, tiene la desventaja que no optimiza el tamaño de la aplicación ni la velocidad de ejecución. En casos en los que tengamos restricciones de tamaño o tiempo, es conveniente utilizar la librería como una referencia para aprender cómo programar el periférico.

3.6.2. CONTENIDO DE LA LIBRERÍA

Podemos descargar de la página web de STMicroelectronics la librería en un fichero comprimido zip. La extracción de este fichero generará la carpeta: **STM32L1xx_StdPeriph_Lib_VX.Y.Z**. Los Parámetros X, Y, y Z hacen referencia a la versión. Actualmente se encuentra en la versión 1.0.0. El contenido de esta carpeta es el que se muestra en la (Figura 3.3).

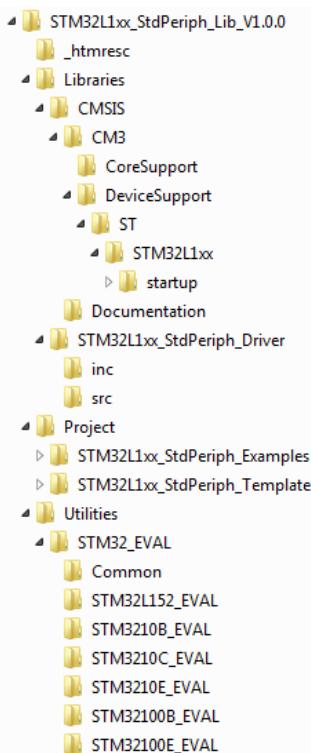


Figura 3.3: *STM32L1xx Standard Peripherals Library*

- **_htmresc.** Esta carpeta contiene todas las páginas de recursos de html de la librería.
- **Libraries.** Contiene todos los ficheros CMSIS y STM32L1xx Standard Peripheral's Drivers.
 - **CMSIS.** Contiene todos los ficheros CMSIS de STM32L1xxxx: device peripheral access layer y core peripheral access layer.
 - **STM32L1xx_StdPeriph_Driver.** Contiene todos los subdirectorios y ficheros que constituyen el núcleo de la librería:
 - **inc.** Contiene los ficheros de cabecera de los drivers de periféricos. Contiene un fichero cabecera por cada driver de periférico.
 - **src.** Contiene los ficheros fuente de los drivers de periféricos. Contiene un fichero fuente por cada driver de periférico.
- **Project.** Esta carpeta contiene los template para diferentes toolchain y ejemplos de cada periférico
 - **STM32L1xx_StdPeriph_Examples.** Esta carpeta contiene una subcarpeta por cada periférico, y dentro de ella, ejemplos de código para aprender a usar dicho periférico.
 - **STM32L1xx_StdPeriph_Templates.** Esta carpeta contiene templates para proyectos para los toolchain: EWARM, MDK-ARM, RIDE, HiTOP, y TrueSTUDIO.
- **Utilities.**
 - **STM32_EVAL.** Implementa una capa de abstracción para que el usuario interactue con la placa: botones, LEDs, LCD, puertos COM, etc. Contiene varias subcarpetas en función de la placa de evaluación que estemos utilizando.

Esta estructura deberemos copiarla en nuestro Workspace. Sin embargo, nuestro toolchain elegido es Atollic TrueSTUDIO. Este toolchain está configurado para crear proyectos a través de un tutorial. Cuando creamos un proyecto nuevo, el propio toolchain nos generará un árbol de directorios igual a este que hemos explicado, y no será necesario copiarlo a nuestro Workspace. No obstante, es necesario tener esta librería descargada ya que viene documentada y tendremos que recurrir a ella en numerosas ocasiones.

3.6.3. NOMENCLATURA

Los drivers de la librería utilizan la siguientes convenciones:

- **PPP** se refiere a un periférico, por ejemplo ADC.
- Los ficheros de fuente y cabecera vienen precedidos por el prefijo **stm32l1xx_**
- Las constantes son definidas en mayúscula. Las constantes definidas en un archivo fuente, son definidas sólo en ese fichero. Las constantes definidas en ficheros de cabeceras, son usadas en mas de un fichero.
- Los registros son considerados constantes. Sus nombres aparecen en mayúscula. La mayoría de las veces son los mismos nombres que los utilizados en el STM32L1xx Reference Manual.
- Las funciones de periféricos se nombran con el nombre del periférico en mayúscula seguido de un guión bajo. A continuación se escribe el nombre de la función, con la primera letra en mayúscula. Por ejemplo USART_SendData.
- Las funciones que inicializan un periférico se nombran **PPP_Init** como ADC_Init.
- Las funciones que resetean los registros de un periférico a sus valores por defecto se nombran **PPP_DeInit** como TIM_DeInit.
- Las funciones que rellenan una estructura con sus valores por defecto se nombran **PPP_StructInit**.
- Las funciones que habilitan o deshabilitan un periférico se nombran **PPP_Cmd**.
- Las funciones que activan o desactivan una fuente de interrupción se nombran como **PPP_ITConfig**
- Las funciones que configuran una función de un periférico siempre terminan en **Config** como **GPIO_PinAFConfig**.
- Las funciones que comprueban si un flag está a uno o cero se nombran **PPP_GetFlagStatus**.
- Las funciones que borran flags se llaman **PPP_ClearFlag**.
- Las funciones que comprueban si una interrupción ha ocurrido o no se nombran **PPP_GetITStatus**.
- Las funciones que borran un bit pendiente de interrupción se nombran **PPP_ClearITPendingBit**.

3.6.4. DESCRIPCIÓN DE LOS FICHEROS DE LA LIBRERÍA

STM32L1xx Standard Peripheral's Library Architecture

En la (Figura 3.4) se muestra una visión global de la librería, uso e interacción con otros componentes.

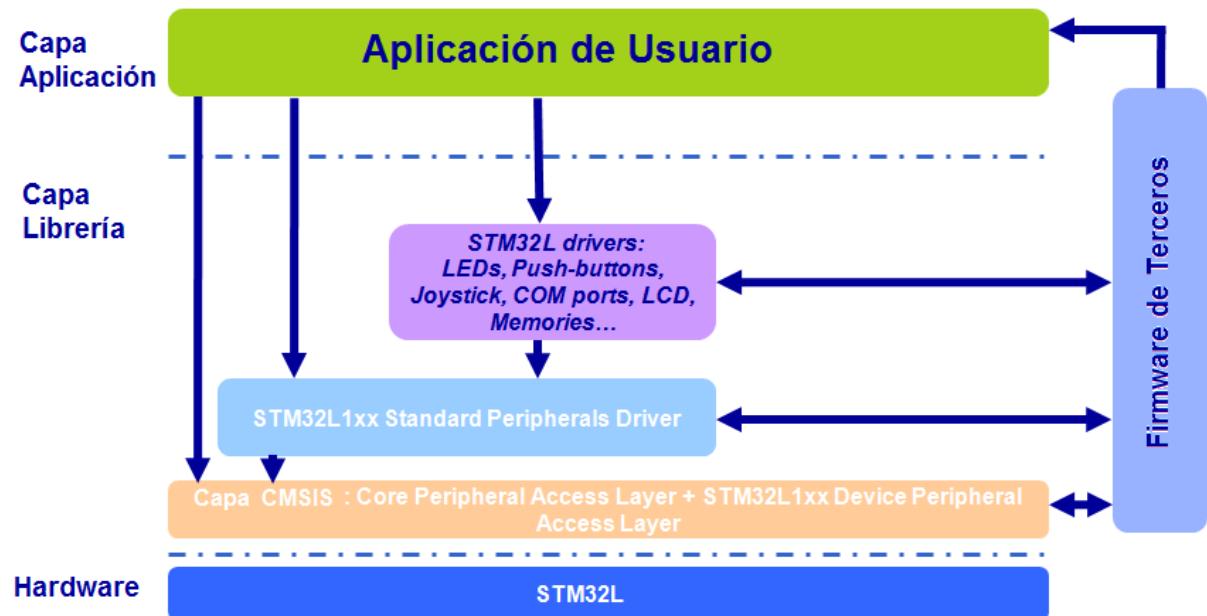


Figura 3.4: Visión Global

Descripción de los drivers de STM32L1xx Standard Peripheral's

En la (Figura 3.5) se muestra la relación de la inclusión de ficheros

Cada periférico tiene un fichero fuente **stm32l1xx_ppp.c** y un fichero cabecera **stm32l1xx_ppp.h**. El fichero **stm32l1xx_ppp.c** contiene todas las funciones necesarias para usar el periférico PPP.

Sólo un fichero de mapeo de memoria **stm32l1xx.h** es necesario para todos los periféricos. Éste fichero contiene todas las declaraciones de registros y las definiciones de bit. Éste es el único fichero que debe ser incluido en la aplicación de usuario para interaccionar con la librería.

El fichero **stm32l1xx_conf.h** se utiliza para especificar el conjunto de parámetros para interactuar con la librería de periféricos antes de correr cualquier aplicación. Se puede activar o desactivar la inclusión de determinados periféricos.

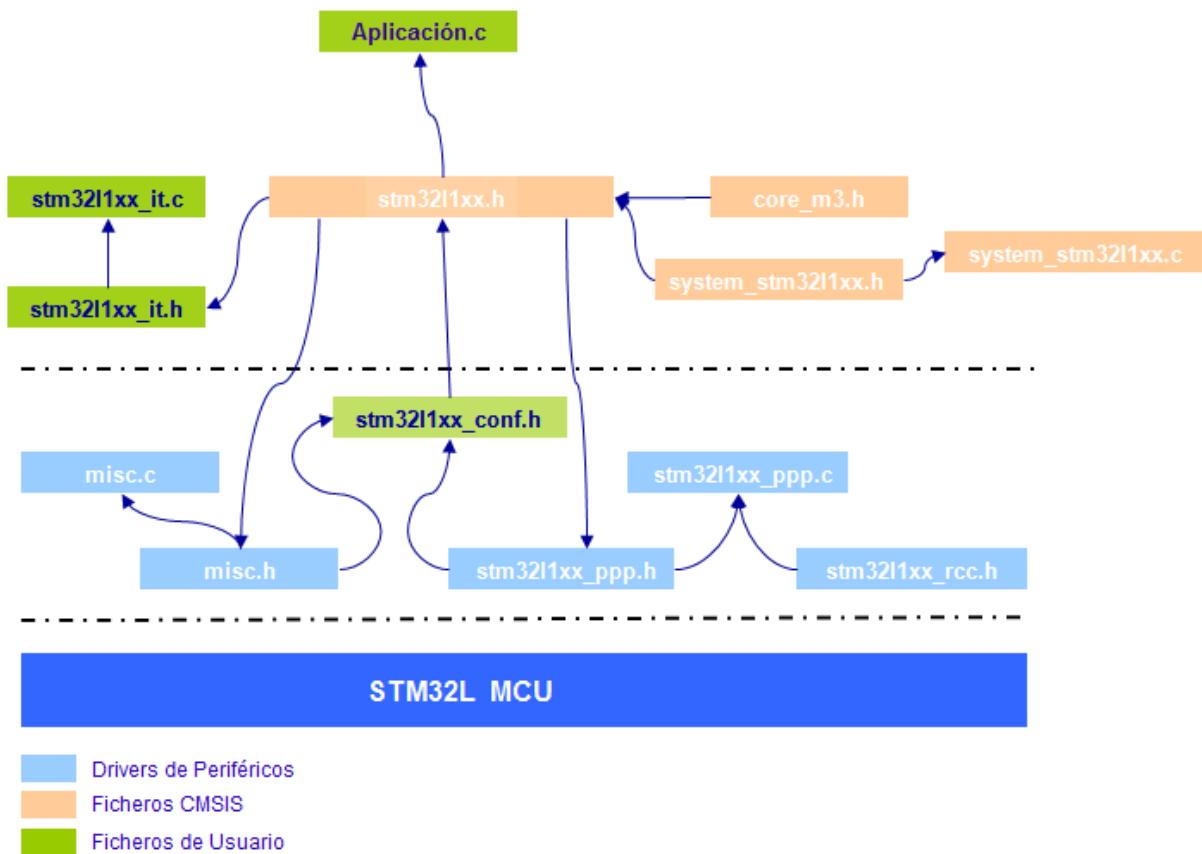


Figura 3.5: Relación Librería

Luego existen los ficheros de interrupción. El fichero fuente `stm32l1xx_it.c` es un fichero que contiene rutinas de interrupción para las excepciones del Cortex-M3. Ademas se pueden añadir nuevas para los diferentes periféricos. El fichero `stm32l1xx_it.h` es el fichero de cabecera con las declaraciones de las rutinas de interrupción.

Descripción de los ficheros CMSIS

- **stm32l1xx.h** Es el fichero de cabecera de CMSIS Cortex M3 STM32L1xxxx de la capa de acceso. Este fichero es el único que hay que incluir en el código de la aplicación, típicamente en el `main.c`. Este fichero contiene:

- Sección de configuración que permite:
 - Seleccionar el dispositivo a utilizar
 - Seleccionar el uso o no de librerías. Se controla con la sentencia:

```
#define USE_STDPERIPH_DRIVER
```

- Cambiar determinados parámetros como puede ser la frecuencia del cristal HSE.
 - Estructuras de datos y mapeo de direcciones para todos los periféricos.
 - Declaración de registros de periféricos y definiciones de bit.
 - Macros para acceso del registro de periféricos.
- **system.stm32l1xx.h** Fichero de cabecera de CMSIS Cortex M3 STM32L1xx de la capa de acceso
 - **system.stm32l1xx.c** Fichero fuente de CMSIS Cortex M3 STM32L1xx de la capa de acceso
 - **startup_stm32l1xx_md.s** Fichero Startup para la familia de microcontroladores STM32L Ultra Low Power Medium density

3.6.5. ¿CÓMO UTILIZAR LA LIBRERÍA?

¿CÓMO UTILIZAR STANDARD PERIPHERAL'S LIBRARY?

En el caso de que utilicemos Eclipse, deberemos utilizar el Template creado para aplicaciones con la placa STM32L-DISCOVERY creado a tal efecto en este proyecto. También se puede utilizar el template proporcionado con la librería STANDARD PERIPHERAL'S LIBRARY en el directorio (Project/STM32L1xx_StdPeriph/Template)

- Seleccionar el correspondiente fichero startup en función a nuestro dispositivo:

```
startup_stm32l1xx_md.s
```

En nuestro fichero fuente donde se encuentre el main, deberemos incluir el fichero de cabeceras stm32l1xx.h y configurarlo:

- Seleccionar nuestro la familia de microcontroladores. Para ello deberemos comentar/descomentar la correspondiente línea:

```
#if !defined (STM32L1XX_MD)  
#define STM32L1XX_MD /*!< STM32L1XX_MD */  
#endif
```

- Seleccionar si usar o no los drivers de periféricos:
 - Código basado en los drivers de periféricos de Standard's Peripheral Library.
 - Descomentar `#define USE_PERIPH_LIBRARY`
 - En `stm32l1xx_conf.h`, seleccionar los periféricos a incluir
 - Utilizar los drivers para desarrollar la aplicación.
 - Código basado en el acceso directo a los registros de periféricos.
 - Comentar `#define USE_PERIPH_LIBRARY`
 - Utilizar los registros y definiciones de bit disponibles en `stm32l1xx.h` para desarrollar la aplicación.

INICIALIZACIÓN Y CONFIGURACIÓN DE PERIFÉRICOS

A continuación se describe paso a paso como inicializar y configurar cualquier periférico utilizando los drivers de periféricos. Como explicamos en un principio, nos referiremos al periférico como PPP.

1. En el fichero que contenga el main, declarar una estructura **PPP_InitTypeDef**, como por ejemplo: ***PPP_InitTypeDef PPP_InitStructure***.

Dicha estructura es una variable localizada en memoria que permite inicializar una o más instancias del periférico PPP.

2. Rellenar la variable **PPP_InitStructure** con valores permitidos de dicha estructura. Para ello se deberá consultar la librería. Existen dos maneras de hacer esto:

- Configurando toda la estructura:

```
PPP_InitTypeDef PPP_InitStructure = { val1, val2, ... valN }
```

- Configurando sólo algunos miembros de la estructura y el resto a sus valores por defecto. Para ello utilizamos la función **PPP_StructInit(...)** que inicializa la función a sus valores por defecto. Después modificamos los valores que deseemos individualmente.

```
PPP_StructInit (&PPP_InitStructure);
PPP_InitStructure.memberX = valX;
PPP_InitStructure.memberY = valY;
```

3. Inicializamos el periférico PPP llamando a la función PPP_Init(...).

```
PPP_Init(PPP, &PPP_InitStructure);
```

4. El periférico ya está inicializado y podemos habilitarlo utilizando la función PPP_Cmd(...).

```
PPP_Cmd(PPP, ENABLE);
```

En este momento el periférico ya está configurado y habilitado y puede ser utilizado a través de las distintas funciones escritas en su driver. Cada periférico tiene sus propias funciones específicas.

Nota: Antes de configurar cualquier periférico, es necesario activar su reloj utilizando una de las siguientes funciones:

- RCC_AHBPeriphClockCmd(RCC_AHBPeriph_PPPx, ENABLE);
- RCC_APB1PeriphClockCmd(RCC_APB1Periph_PPPx, ENABLE);
- RCC_APB2PeriphClockCmd(RCC_APB2Periph_PPPx, ENABLE);

GUÍA DE PERIFÉRICOS

En este capítulo se mostrará una guía de configuración de los principales periféricos que se utilizan en las aplicaciones de evaluación que se han desarrollado bajo este proyecto. Con esta guía se intenta abordar la configuración y modo de empleo de cada periférico estudiado. Para un estudio detallado se recomienda la lectura y consulta del Reference Manual.

4.1. RESET AND CLOCK CONTROL (RCC)

4.1.1. Relojes

En el stm32l152xx hay diferentes fuentes de reloj (internos y externos) así como un PLL.

- **HSI (High-Speed Internal).** Reloj interno de 16 Mhz a través de un oscilador RC. Puede ser usado como System Clock o como entrada al PLL. No requiere de componentes externos.
- **MSI (Multi-Speed Internal).** Reloj interno a través de un oscilador RC que proporciona siete frecuencias: 65.536 kHz, 131.072 kHz, 262.144 kHz, 524.288 kHz, 1.048 MHz, 2.097 Mhz (valor por defecto) y 4.194 Mhz. El reloj MES es usado como System Clock después del Reset, wake-up desde Stop, y modo Standby low power. El valor por defecto con el que arranca es 2.097 MHz.
- **LSI (Low-Speed Internal).** Es un reloj RC de 37 kHz y se utiliza para el IWDG (Independent Watchdog) o como reloj para el RTC (Real Time Clock).
- **HSE (High-Speed External).** Proporciona desde 1 hasta 24 MHz de un oscilador de cristal. También funciona como reloj del PLL o el RTC.
- **LSE (Low-Speed External).** Reloj que proporciona 32.768 kHz a partir de un resonador cerámico o un cristal externo. Utilizado como fuente para el reloj RTC y el reloj del LCD.
- **PLL.** La fuente de reloj proviene del HSI o el HSE. Sirve como System Clock (entre 2 y 24 MHz) o para generar los 48 MHz para el periférico USB.
- **CSS (Clock Security System.**

El RTC y el LCD utilizan el mismo reloj (puede ser el LSE, LSI o el HSE a 1 Mhz). Nosotros escogemos el LSE como fuente para estos dos periféricos. Comparten configuración de reloj.

Después de reiniciar el controlador, el dispositivo tiene configurado el reloj MSI como System Clock. En este punto, todos los periféricos están desactivados excepto la memoria SRAM, Flash y JTAG. Además, no hay preescalado para el AHB ni los buses APB, todos ellos corren a la misma velocidad que el MSI. Al mismo tiempo, todos los puertos GPIO están configurados como input floating.

Una vez arrancado el dispositivo, deberemos:

- Seleccionar una fuente de reloj para el System Clock en caso de que necesitemos mayores frecuencia que la que nos proporciona el MSI. Para el desarrollo de las aplicaciones creadas, no existían requisitos de frecuencia por lo que no se ha configurado otro reloj como System Clock.
- Configurar el preescalado de los buses AHB y APB si se desea. En nuestro caso, tampoco se ha optado por esta medida.
- Habilitar los relojes de los periféricos que se utilizan.
- Configurar los relojes de aquellos periféricos que utilizan un reloj independiente, como es el caso del RTC/LCD o el IWDG.

Los periféricos del microcontrolador estan conectados a tres buses ya mencionados con anterioridad y que describimos a continuación:

- **Advanced High-Performance (AHB)**. A él estan conectados: GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOH, RCC, DMA, entre otros.
- **Advanced Peripheral Bus (APB1)**. Trabaja como máximo a 36 Mhz. Se conectan a él: DAC, PWR, RTC, LCD, TIM7, TIM6, etc.
- **Advanced Peripheral Bus (APB2)**. Trabaja como máximo a 36 Mhz. Estan conectados a el: ADC, TIM9, TIM10, TIM11, EXTI, SYSCFG, etc.

Los periféricos más comunes se encuentran conectados a uno de estos buses y por defecto están desactivados para evitar el consumo de energía. Debido a esto, deberemos activarles la señal de reloj utilizando la función **RCC_APB1PeriphClockCmd()**, **RCC_APB1PeriphClockCmd()**, o **RCC_APB1PeriphClockCmd()**, tal y como corresponda en su caso. Siempre que deseemos utilizar un periférico, deberemos consultar en el Reference Manual a qué bus está conectado para proceder a activarlo.

4.2. GENERAL PURPOSE I/Os (GPIO)

GPIO es uno de los periféricos más sencillos de utilizar, pero al igual que otras muchas facetas de un ARM Cortex M3, tiene bastantes parámetros que configurar. Para configurar un puerto GPIO se utiliza los drivers de la librería de STMicroelectronics. Dicho driver, proporciona una estructura para el periférico GPIO, una función que da valores por defecto a dicha estructura y una función para inicializar el periférico. En el caso de GPIO la estructura se llama `GPIO_InitTypeDef`, la función que da valores se llama `GPIO_StructInit()`, y la que inicializa el periférico `GPIO_Init()`. Esta nomenclatura es común al resto de periféricos, lo único que cambia es el nombre del periférico.

En el caso de GPIO, la estructura tiene 3 campos a configurar:

- `GPIO_Pin` Con este campo referenciamos al pin que queremos inicializar. Podemos inicializar varios pines a la vez utilizando el operador | (OR) o directamente inicializar todos los pines de un mismo puerto con la sentencia `GPIO_Pin_All`.
- `GPIO_Mode` Configura el modo de dirección.
 - `GPIO_Mode_In` Modo de entrada
 - `GPIO_Mode_OUT` Modo salida
 - `GPIO_Mode_AF` Modo Función Alternativa
 - `GPIO_MODE_AN` Modo Analog: para un ADC Channel, DAC Output o Comparador.
- `GPIO_Speed` Define la velocidad máxima de trabajo. Puede ser: `GPIO_Speed_400Khz`, `GPIO_Speed_2MHz`, `GPIO_Speed_10Mhz`, `GPIO_Speed_40MHz`.
- `GPIO_OType` Establece el modo de operación en modo salida. Puede ser: `GPIO_OType_PP` (Push-Pull) o `GPIO_OType_OD` (Open Drain).
- `GPIO_PuPd` Establece si el pin opera en pull-up o pull-down. Puede ser: `GPIO_PuPd_NOPULL`, `GPIO_PuPd_UP`, `GPIO_PuPd_DOWN`.

El siguiente paso es llamar a la función `GPIO_Init()` para que inicialice el periférico con los datos que tenemos que configurar. En la estructura hemos determinado los datos para esos pines

y su modo de operación, velocidad, etc. A la hora de llamar a `GPIO_Init()` debemos indicar a qué puerto queremos inicializarlo. Un ejemplo de llamada sería:

```
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

Al que le hemos pasado el puerto al que pertenecen los pines que acabábamos de configurar y la estructura anteponiéndole `&`. Ya tenemos configurados los pines con sus parámetros deseados y podemos comenzar con operaciones de set/reset. Para realizar estas operaciones utilizaremos directamente los registros del microcontrolador ya que es muy sencillo. Estos registros son:

- **`GPIOx_BSRR`**: set al pin indicado. El bit 0 es el pin 0, el bit 1 es el pin 1, y así sucesivamente.
- **`GPIOx_BSRRH`**: reset al bit indicado.

Nota: No confundir la L y la H de los anteriores registros con poner el bit a Low o High. Este es una convención heredada del antiguo registro de GPIO. El antiguo registro de GPIO (el que aparece en el REFERENCE MANUAL en la página 122) está desactualizado. Con ese mismo registro llamado `GPIOx_BSRR` podíamos hacer set y reset, ya que los 16 primeros bits eran el set y los 16 restantes para el reset. La L y la H de los nuevos registros hacen referencia a los bits más significativos y los menos significativos.

A continuación se muestra un ejemplo del uso de estos registros:

```
GPIOB->BSRRL = 0x00000080; // set al pin PB7
GPIOB->BSRRH = 0x00000040; // reset al pin PB6
temp=GPIOA->IDR; //leemos el valor de los pines de entrada del puerto A
temp=GPIOB->ODR; //leemos el valor de los pines de salida del puerto B
```

Cuando configuramos un pin en función alternativa, después de inicializarlo con la llamada `GPIO_Init()`, debemos indicar cuál de las funciones alternativas de ese pin va a llevar a cabo. En el datasheet STM32L151xx STM32L152xx podemos consultar todas y cada una de las funciones alternativas de cada pin. Para indicarlo, basta solamente con la instrucción:

`GPIO_PinAFConfig(GPIOx, GPIO_PinSourcex, GPIO_AF)` donde:

- `GPIOx`: es el puerto del pin, por ejemplo `GPIOA`, `GPIOB`, etc.

- **GPIO_PinSource_x**: es el pin, por ejemplo `GPIO_PinSource1`, `GPIO_PinSource2`, etc.
- **GPIO_AF**: es la función que va a desempeñar. Por ejemplo `GPIO_AF_TIM2`, `GPIO_AF_LCD`, etc. Para una lista detallada consultar el driver de GPIO `stm32l1xx_gpio.c`

4.3. SYSTEM TIMER (SysTick)

El ARM Cortex tiene un timer interno llamado System Timer (SysTick). Este timer es muy apropiado para generar ticks o medir delays. Una de las grandes ventajas es que es común a todos los Cortex M3 por lo que cualquier procesador de la serie M3 funcionará idénticamente.

El método de operación detallado viene explicado en la documentación de ARM Cortex M3, pero nosotros no vamos a profundizar en su uso, sino que vamos a ver una aplicación concreta, insertar delays.

4.3.1. CONFIGURACIÓN

El objetivo es programar el SysTick para generar un tiempo base de 1ms. Gracias a él y a la función Delay que programaremos, generaremos Delays de manera muy sencilla.

El SysTick está manejado por el reloj AHB clock (HCLK).

Fuera del main, programamos las funciones:

```
volatile static uint32_t TimingDelay;
RCC_ClocksTypeDef RCC_Clocks;

/* Función que configura el Systick a 1ms */
void Config_Systick(void) {
    RCC_GetClocksFreq(&RCC_Clocks);
    SysTick_Config(RCC_Clocks.HCLK_Frequency / 1000);
}

/* Función para implementar delays de nTime ms */
void Delay(uint32_t nTime) {
    TimingDelay = nTime;
    while(TimingDelay != 0);
}

/* Función que será llamada cada vez que salte la interrupción del SysTick */
void TimingDelay_Decrement(void) {
    if (TimingDelay != 0x00) {
        TimingDelay--;
    }
}
```

La función Config_Systick() deberá ser llamada dentro del main para configurar el SysTick.

Además, deberemos programar la RAI para la interrupción del SysTick en el fichero `stm32l1xx_it.c`.

```
void SysTick_Handler(void)
{
    TimingDelay_Decrement();
}
```

De esta manera ya tenemos configurado nuestro timer con una base de tiempos de 1 ms. Para generar un delay, bastan con hacer una llamada a la función `Delay()` pasándole por parámetro el tiempo de ese delay expresado en ms, por ejemplo, `Delay(500)` generará un delay de 0,5 segundos.

El funcionamiento es el siguiente: al llamar a `Delay`, entraremos en un bucle while (`TimingDelay != 0`). Cada ms, la interrupción de SysTick salta, y lo que hace su RAI es decrementar el valor de la variable `TimingDelay` unitariamente. Por lo tanto, con cada excepción, el valor de `TimingDelay` se irá decrementando hasta llegar a 0, momento en el finaliza el `Delay`.

4.4. DISPLAY LCD

En la placa STM32L-DISCOVERY hay dos elementos que tenemos que saber identificar. Por un lado está el driver del controlador LCD y por otro lado el driver para el display LCD que viene montado en la placa.

El controlador LCD es un driver para controlar displays pasivos de cristal líquido (LCD) de hasta 44 segmentos. El número exacto depende del pinout.

El controlador del display LCD es un driver para controlador el display que viene montado en nuestra placa.

A continuación se muestra una guía de como configurar el LCD utilizando el driver tanto del display como del controlador, para poder mostrar mensajes por pantalla.

4.4.1. Reloj del LCD

La frecuencia del generador permite lograr varios frame rates para el LCD partiendo del LCDCLK que varía desde 32Khz hasta 1Mhz. El LCD y el RTC comparten la misma fuente de reloj. Hay disponibles 3 fuentes para el LCDCLK/RTCCLK

- 32 Khz LSE
- 37 Khz LSI
- 1 hasta 24 Mhz HSE dividiendo por factores 2, 4, 8 y 16.

4.4.2. CONFIGURACIÓN

Llamada a la función **RCC_Configuration()**. Dicha función no pertenece a ningún driver de las librerías. Esta rutina configura diferentes elementos de reloj de los periféricos que son necesarios para arrancar el LCD, y habilita el reloj LSE que es el que domina al LCD.

- Arranca los relojes de los periféricos que se van a utilizar como GPIOA, GPIOB, y GPIOC en el bus AHB; reloj del LCD y PWR en el bus APB1; y el reloj de SYSCFG en el bus APB2.
- Habilitamos acceso al dominio RTC, y resetteamos el dominio RTC.

- Habilitamos el reloj LSE, que es el reloj que domina al RTC y al LCD.
- Una vez activado el reloj LSE, hay que esperar a que sea estable.
- Por último, seleccionamos el reloj LSE como reloj fuente de reloj del RTC y LCD.

Llamada a la función **Init_GPIO_LCD()**. Esta función no pertenece a ningún driver de la librería. Su misión es configurar los pines de I/O a los que están conectados el display LCD. Estos pines pertenecen a PORT A, PORT B, y PORT C, por eso era necesario activar el reloj de cada una en la función **RCC_Configuration()**.

El funcionamiento interno de la función configura los pines necesarios como **función alternativa**. Una vez configurados de tal modo, hay que indicar uno a uno cuales de las funciones alternativas van a utilizar. Los pines de cada puerto que son necesarios configurar son:

- PORT A: PA1 PA2 PA3 PA8 PA9 PA10 PA15
- PORT B: PB3 PB4 PB5 PB8 PB9 PB10 PB11 PB12 PB13 PB14 PB15
- PORT C: PC0 PC1 PC2 PC3 PC6 PC7 PC8 PC9 PC10 PC11

En la página 23 del USER MANUAL (UM1079) de STM32L-Discovery vienen indicados las conexiones del LCD con los puertos I/O. Siempre y cuando el LCD esté conectado, dichos puertos no podrán ser utilizados como GPIO.

En la tabla 5 del Datasheet de STM32L151xx y STM32L152xx, vienen indicadas las funciones alternativas de cada pin de I/O.

Llamada a la función **LCG_GLASS_Init()**. Esta función pertenece al driver del controlador del display LCD que se encuentra en la ruta

`STM32L1xx_StdPeriph_Lib_V1.0.0/Utilities/STM32_EVAL/stm32l_discovery_lcd.c`.

El funcionamiento de esta función es el siguiente:

- Configuración del LCD: prescalado, LCD divider, duty, bias, y fuente del voltaje.
- Habilitamos el Mux Segment.
- Establecemos un nivel de contraste de 2.99 V sin dead time

- Asignamos una duración del pulso de 4/CK_PS (Reloj Prescalado del LCD).
- Esperamos a que se sincronicen los registros del LCD y habilitamos el LCD.
- Una vez habilitado, esperamos al flag del LCD y al flag READY. Tras esto ya podemos escribir mediante funciones de escritura y borrado.

Para poder escribir en el LCD utilizamos las siguientes funciones principalmente:

- **LCD_GLASS_DisplayString(uint8_t* ptr)**. Escribe un string en el display LCD. Recibe como parámetro un puntero a un String.
- **LCD_GLASS_Clear()**. Esta función borra la memoria RAM del LCD.
- **LCD_GLASS_ScrollSentence(uint8_t* ptr, uint16_t nScroll, uint16_t ScrollSpeed)**. Utilizamos esta función cuando deseamos mostrar un String en modo scroll. Recibe como parámetro el puntero al string, cuantas veces va a ser el string pasado por pantalla, y la velocidad de paso en ms. Para poder utilizar esta función el display debe ser borrado con anterioridad.
- **LCD_GLASS_DisplayStrDeci(uint16_t* ptr)**. Esta función escribe un char en la memoria RAM del LCD. Requiere que se le pasa un array de caracteres en ASCII.

Alternativamente, se puede configurar para que el LCD parpadee. Esto se consigue con la función **LCD_BlinkConfig(uint32_t LCD_BlinkMode, uint32_t LCD_BlinkFrequency)**.

- A través:
de LCD_BlinkConfig(LCD_BlinkMode_AllSEG_AllCOM, LCD_BlinkFrequency_Div512)
configuramos el LCD para que parpadee todos los segmentos.
- Gracias a LCD_BlinkConfig(LCD_BlinkMode_Off, LCD_BlinkFrequency_Div512) deshabilitamos el parpadeo.

4.5. EXTERNAL INTERRUPT (EXTI) y NVIC

El NVIC (Nested Vectored Interrupt Controller) es el periférico que gestiona las interrupciones. Pertenece al núcleo de ARM, es capaz de gestionar 45 canales enmascarables de interrupción sin contar las 16 líneas de interrupción propias de ARM y puede asignar 16 niveles de prioridad.

El controlador de interrupciones externas (EXTI) consiste en 23 detectores para generar interrupciones. Cada línea de entrada puede ser configurada individualmente y también ser enmascarada. Además, se pueden configurar 23 interrupciones generadas por software y un registro de estado controla cada línea con un bit por cada una.

A continuación se muestra como configurar una línea externa para generar una interrupción.

Primero, se debe configurar la línea de interrupción y habilitarla. Cuando el determinado pulso tiene lugar en la línea de interrupción, una interrupción es generada y su bit correspondiente es puesto a 1. La RAI (Rutina de Atención a la Interrupción) procesa el código suspendiendo la ejecución normal de nuestro programa.

En nuestro STM32L15xx los 83 pines GPIO están conectados a 16 líneas de interrupción externa, de esta manera conseguimos total libertad para configurar nuestra línea de interrupción externa, pudiendo asignar cualquier pin a nuestra línea de interrupción. Estas son 16 líneas pero en total EXTI gestiona 23 líneas de interrupción. El resto están conectadas como se indica a continuación:

- EXTI16 está conectada a la salida de PVD
- EXTI17 está conectada a RTC_ALARM
- EXTI18 está conectada a USB Device FS Wakeup
- EXTI19 está conectada a RTC Tamper y TimeStamp
- EXTI20 está conectada a RTC Wakeup
- EXTI21 y EXTI22 están conectadas a Comparator 1 y 2

Para asignar las líneas de interrupción a los pines de GPIO tenemos el periférico **SYSCFG** (System Configuration controller). Este periférico es el encargado de mapear las líneas de interrupción externa a los puertos GPIO. Es necesario que arranquemos su reloj con la instrucción:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG , ENABLE);
```

Además de arrancar el reloj de SYSCFG, debemos inicializar el pin GPIO que queremos utilizar. El procedimiento para ello es inicializar el pin que deseemos en modo entrada, sin pull-up ni pull-down. Para realizarlo, basta con ver la guía de configuración de GPIO.

Al igual que el driver de GPIO, necesitamos crear una estructura para nuestra línea de interrupción. Esta es EXTI_InitTypeDef. Al mismo tiempo, como indicamos con anterioridad, tenemos que indicar qué pin está asociado a qué línea externa. Para ello utilizaremos el periférico SYSCFG con la función:

```
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOx, EXTI_PinSourcey)
```

Con ello indicamos que el pin Pyx tendrá asociada una línea de interrupción. Gracias al mapeado de pines y puertos de la imagen, sabemos a qué línea está asociada a ese pin de tal puerto.

Ahora tenemos que dar valores a los parámetros de la estructura que creamos con anterioridad. En el caso de EXTI la estructura tiene cuatro parámetros:

- **EXTI_LINE**. Indicamos qué línea de interrupción es la que deseamos configurar, por ejemplo, EXTI_Line0.
- **EXTI_Mode**. Aquí seleccionamos si deseamos una interrupción o evento. Nosotros trabajaremos siempre por interrupción: EXTI_Mode_Interrupt.
- **EXTI_Trigger**. En este parámetro debemos seleccionar el flanco de detección. Puede ser:
 - EXTI_Trigger_Rising. Flanco de subida
 - EXTI_Trigger_Falling. Flanco de bajada
 - EXTI_Trigger_Rising_Falling. Ambos
- **EXTI_LineCmd**. Indicamos activar la línea de interrupción

Tras configurar la estructura, debemos inicializarla a través de EXTI_Init(). Un ejemplo sería el siguiente, suponiendo que EXTI_InitStructure sea la estructura que configuramos anteriormente:

```
EXTI_Init(&EXTI_InitStructure);
```

A estas alturas vemos que la filosofía de configuración es la misma que en el caso de GPIO. Esta filosofía se mantendrá con todos los periféricos, por lo tanto en cuanto aprendamos a programar dos o más periféricos tendremos bastante soltura para programar el resto de ellos, simplemente tendremos que consultar en los drivers los parámetros para configurarlos.

Una vez inicializada la línea externa de interrupción, tenemos que configurar el canal IRQ NVIC que gestionar la línea de interrupción externa utilizando `NVIC_Init()`. Para el periférico NVIC la estructura es `NVIC_InitTypeDef`. El driver de NVIC viene en la librería en el fichero **misc.c**. Los parámetros que tenemos que configurar en la estructura NVIC son los siguientes:

- **NVIC_IRQChannel**. Especifica el Canal IRQ a ser activado
- **NVIC_IRQChannelPreemptionPriority**. Especifica la prioridad pre-emption para el canal IRQ. Este valor puede ser cualquier valor entre 0 y 15
- **NVIC_IRQChannelSubPriority**. Especifica el nivel de subprioridad para el canal IRQ. Puede ser un valor entre 0 y 15.
- **NVIC_IRQChannelCmd**. Especifica si activa o desactiva el canal. Admite: ENABLE o DISABLE.

Para inicializar la estructura debemos llamar a `NVIC_Init()` pasándole por parámetro nuestra estructura completada con la configuración deseada. Un ejemplo es:

```
NVIC_Init(&NVIC_InitStructure);
```

Nota: El parámetro que admite `NVIC_IRQChannel` debe ser un canal IRQ valido. Para consultar la lista debemos ver el fichero `stm32l1xx.h`

Con esto ya tenemos configurada la línea de interrupción externa y configurada la IRQ de NVIC. Ahora tenemos que crear la RAI. El fichero `stm32l1xx_it.c` es el encargado de gestionar todas las interrupciones y es en él donde debemos añadir nuestra RAI.

El objetivo de la RAI es comprobar que sea la línea de interrupción configurada es la responsable de la interrupción y de ser así, realizar lo que deseemos para esa RAI y antes de terminar su ejecución, borrar el bit de interrupción. A continuación se muestra un ejemplo de RAI para la línea de interrupción externa EXTI0.

```
void EXTI0_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_Line0) != RESET)
    {
        /* Código que deseemos que se ejecute en la RAI */

        /* Clear the EXTI line 0 pending bit */
        EXTI_ClearITPendingBit(EXTI_Line0);
    }
}
```

El nombre de la rutina de atención a la interrupción no puede ser trivial y se tiene que corresponder con el nombre designado por ARM Cortex M3 en el vector table que aparece en:

Libraries\CMSIS\CM3\DeviceSupport\ST\STM32L1xx\startup\TrueSTUDIO\startup_stm32xx_md.s

De todas formas para las líneas de interrupción externa que son las que son de nuestro interés en este momento, los nombres admitidos para la RAI son los siguientes:

- EXTI0_IRQHandler
- EXTI1_IRQHandler
- EXTI2_IRQHandler
- EXTI3_IRQHandler
- EXTI4_IRQHandler

Para una lista completa de todas las RAI para todos los periféricos disponibles en STM32L15xx, es aconsejable consultar el archivo arriba indicado.

4.6. ANALOG-TO-DIGITAL CONVERTER (ADC)

El ADC que dispones el STM32L15xx es un ADC de 12 bits de resolución con capacidad para medir 26 canales, de los cuales 24 externos y 2 internos (corriente y temperatura). Sus modos de conversión son simple, continuo, scan y modo discontinuo. Además incorpora un watchdog tanto para umbrales altos como umbrales bajos. Tanto ADC como su watchdog generan interrupciones al final de conversiones o cuando el voltaje de entrada sobrepasa alguno de sus umbrales. Las conversiones son siempre realizadas a la máxima velocidad pudiendo preescalar esta velocidad.

Nuestro ADC trabaja con reloj HSI (High-Speed Internal) a 16 MHz para trabajar de manera independiente. Si deseamos trabajar a velocidades más bajas existe la posibilidad de aplicar un divisor de reloj para trabajar a media velocidad (8 MHz) y a baja velocidad (2 Mhz). El hecho de que nuestro ADC trabaje a velocidades tan altas puede ocasionar un problema con las conversiones. El problema reside en el hecho de que nuestro APB clock puede ser de velocidades inferiores. Nada más arrancar el dispositivo, éste trabaja con MSI a 2 Mhz (aprox.) mientras que si configuramos el ADC a 16 MHz existirá un problema de velocidades, en el que se pierdan datos convertidos. Para ello existe la posibilidad de insertar delays de tiempo entre conversión y conversión para darle suficiente tiempo al sistema para leer y salvar el dato convertido hasta la siguiente conversión.

Hay 16 canales multiplexados y existe la posibilidad de organizar las conversiones en dos grupos (conjunto de secuencias de conversión que pueden ser realizados en cualquier orden, por ejemplo: Ch1, Ch3, Ch8, Ch2, Ch15):

- **Regular Group:** compuestos por un máximo de 27 conversiones. El orden de conversiones se puede elegir.
- **Injected Group:** compuesto por un máximo de 4 conversiones. El orden de conversiones se puede elegir. Los Injected Channels no pueden ser configurados en modo continuo.

Para la configuración que vamos a mostrar, se empleará un Regular Group que es más genérico.

Los modos de operación son:

- **Single Conversion Mode.** El ADC solo realiza una conversión. Se realiza conversión

por software o mediante una señal externa. Una vez que la conversión ha sido realizada, se almacena el dato convertido en un registro de 16 bits, se pone a 1 el flag End Of Conversion, y una interrupción es generada si se ha activado previamente.

- **Continuous Conversion Mode.** El ADC comienza una nueva conversión tan pronto como ha terminado la anterior. Se realiza conversión por software o por señal externa. Al igual que en el modo sencillo, el dato se almacena en un registro de 16 bits, el flag End Of Conversion se pone a 1 y se genera una interrupción si anteriormente se habilitó.

Paralelamente, otros dos modos de operación son posibles:

- **Scan Mode.** Este método se utiliza para convertir todos los canales de un mismo grupo. Una conversión es realizada por cada canal del grupo. Después de cada conversión, el siguiente canal del grupo se convierte automáticamente. Si nos encontramos en modo continuo, cuando finalice el último canal del grupo, volverá a comenzar, sino, terminará las conversiones con el último canal del grupo.
- **Discontinuous Mode.** En este modo sólo se convierten hasta un máximo de 8 canales dentro de un grupo.

4.6.1. CONFIGURACIÓN

Para configurar este periférico, primeramente deberemos configurar otros periféricos y elementos de nuestro dispositivo. Como indicamos anteriormente, el ADC utiliza el reloj HSI que deberemos habilitar a través de la sentencia `RCC_HSICmd(ENABLE)`. Una vez activado, al igual que ocurría con el LSE para el RTCCLK/LCDCLK, tendremos que esperar al flag de ready. Al mismo tiempo deberemos activar el reloj del ADC que está conectado al bus APB2 mediante. Como último paso previo, deberemos escoger que pin GPIO vamos a utilizar para el canal analógico de entrada del ADC. Para ello configuraremos el pin seleccionado siguiendo la guía descrita previamente, en modo analógico. Un resumen en código de este proceso es el siguiente:

```
RCC_HSICmd(ENABLE) ;  
while(RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET) ;  
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE) ;
```

```

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
GPIO_Init(GPIOA, &GPIO_InitStructure);

```

El siguiente paso es configurar el ADC. Al igual que anteriores periféricos, el driver de ADC dispone de su propia estructura de configuración conocida como **ADC_InitTypeDef**. Los parámetros a configurar para esta estructura son los siguientes:

- **ADC_Resolution.** Es la resolución del ADC. Admite:
 - ADC_Resolution_12b
 - ADC_Resolution_10b
 - ADC_Resolution_8b
 - ADC_Resolution_6b
- **ADC_ScanConvMode.** Especifica si la conversión es en Modo Scan o Sencilla. Admite ENABLE o DISABLE.
- **ADC_ContinuousConvMode.** Especifica si la conversión es realizada en modo continuo o discontinuo. Admite ENABLE o DISABLE.
- **ADC_ExternalTrigConvEdge.** Selecciona si hay señal externa de conversión, y si es así, el flanco de activación.
 - ADC_ExternalTrigConvEdge_None. No hay conversión por señal externa.
 - ADC_ExternalTrigConvEdge_Rising. Activo por flanco de subida.
 - ADC_ExternalTrigConvEdge_Falling. Activo por flanco de bajada.
 - ADC_ExternalTrigConvEdge_RisingFalling. Activo por ambos.
- **ADC_DataAlign.** Alineación de los datos. Por defecto siempre diremos ADC_DataAlign_Right.
- **ADC_NbrOfConversion.** Número de conversiones a realizar de cada grupo de canales. Admite un número entre 1 y 27.

Una vez definida la estructura, pasamos a inicializarla con **ADC_Init()**. Un ejemplo sería:

```
ADC_Init(ADC1, &ADC_InitStructure);
```

Nota: Aunque nuestro dispositivo STM32L15xx sólo tiene un ADC, la librería actúa como si existieran dos ADC. Siempre deberemos indicar ADC1 como si estuviéramos utilizando el ADC número uno.

Tras configurar nuestro ADC, debemos asignar algunos parámetros adicionales de configuración que no aparecen contemplados en la estructura del driver del ADC. Algunos de ellos son los siguientes:

- ADC-RegularChannelConfig(ADC1, ADC_Channel_x, rank, ADC_SampleTime_zCycles).

Con esta instrucción determinamos:

- ADC1. Como indicamos antes, este parámetro es fijo puesto que sólo tenemos un ADC.
- ADC_Channel_x. Es el canal ADC donde x es el número del canal, pudiendo tomar valores desde 0 a 25.
- rank. Es el orden del canal dentro del grupo. Para canales regulares puede tomar valores desde 1 hasta 26.
- ADC_SampleTime_zCycles Indica el sample time el canal seleccionado. Z puede tomar valores entre 4, 9, 16, 24, 48, 96, 192 y 384.

Es necesaria además la configuración del delay que mencionamos al principio. A la hora de configurarlo se utiliza la función ADC_DelaySelectionConfig(). El delay se puede configurar de muchos tiempos con divisores del APB clock pero la mejor opción es configurarlo para que no comience una conversión hasta que la anterior haya sido leída. Esto se hace con la siguiente configuración:

```
ADC_DelaySelectionConfig(ADC1, ADC_DelayLength_Freeze);
```

El Watchdog requiere de varias instrucciones para ser configurado. Entre las acciones que debemos realizar están establecer sus umbrales, y seleccionar si se desea activar para un canal o todos los de un grupo. A través de:

```
ADC_AnalogWatchdogThresholdsConfig(ADC1, umbralH, umbralL);
```

configuramos los umbrales. Los parámetros son umbralH y umbralL que pueden tomar valores 0 y 4096 (12 bits) y son el umbral superior y el inferior respectivamente. Luego tenemos que indicar si queremos activar el watchdog para un sólo canal o todos los de su grupo. Esto lo realizamos a través de la función:

```
ADC_AnalogWatchdogCmd(ADC1, ADC_AnalogWatchdog);
```

dónde ADC_AnalogWatchdog puede ser ADC_AnalogWatchdog_SingleRegEnable para un solo canal, o también ADC_AnalogWatchdog_AllRegEnable para todos los canales del grupo. Si elegimos la primera opción, antes de esta sentencia debemos decirle al Watchdog que canal es el que va a analizar. Esto se consigue a través de:

```
ADC_AnalogWatchdogSingleChannelConfig(ADC1, ADC_Channel_x);
```

dónde x es el número del canal pudiendo tomar valores entre 0 y 25.

La última sección de la configuración del ADC es la gestión de interrupciones. El ADC puede generar varias interrupciones como el watchdog, final de conversión, etc. Primero tenemos que configurar su IRQ. Este proceso se realiza exactamente igual que en el caso de la interrupción externa, creando una estructura, dando valores a sus parámetros e inicializándola. A modo de ejemplo se adjunta la siguiente configuración:

```
NVIC_InitTypeDef NVIC_InitStructure;
NVIC_InitStructure.NVIC_IRQChannel = ADC1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Tras esto tenemos que seleccionar que interrupciones habilitar para la IRQ del ADC. Las interrupciones que utilizaremos son las siguientes:

- ADC_IT_OVR. Overrun: cuando un dato convertido es perdido.
- ADC_IT_EOC. End Of Conversion: para indicar que la conversión de un canal o un grupo ha terminado.
- ADC_IT_AWD. Analog Watchdog: indica si el voltaje de entrada excede los umbrales por encima o por debajo.

Para habilitar cualquiera de estas tres interrupciones se utiliza ADC_ITConfig(ADC1, ADC_IT, ENABLE), dónde ADC_IT puede ser cualquiera de las tres fuentes de interrupción arriba indicadas. Al igual que pueden habilitarse, pueden deshabilitarse en cualquier momento indicando DISABLE como tercer parámetro.

Al igual que con el resto de interrupciones de periféricos, la RAI se encuentra en el fichero stm32l1xx_it.c, en ella deberemos escribir la RAI que gestiona la interrupción del ADC. Al igual que otros periféricos, esta RAI es exclusiva del periférico pero puede atender a varias fuentes de interrupción, por ejemplo, en el caso del ADC podríamos activar interrupci por AWD y EOC. Para ello dentro de la RAI deberemos leer el flag de interrupción para identificar la fuente. A continuación vemos un ejemplo para el caso de una interrupción AWD. Lo primero que se realiza es identificar la fuente la interrupción, después se ejecuta el código y por último antes de salir de la RAI, se borra el bit de esa interrupción.

```
void ADC1_IRQHandler(void)
{
    if(ADC_GetITStatus(ADC1, ADC_IT_AWD) != RESET)
    {
        /* Código que ejecuta la RAI cuando hay interrupción */

        ADC_ClearITPendingBit(ADC1, ADC_IT_AWD);
    }
}
```

Al igual que todos las RAIs, el nombre que recibe no puede ser trivial y debe ser consultado en el fichero startup. En el caso del ADC el nombre que debe recibir es el que se muestra en el ejemplo.

Por último, una vez que hemos configurado el ADC, el watchdog, la IRQ, falta activar el ADC y comenzar a convertir. Como no hemos configurado por señal externa su activación, deberemos realizar esa tarea por software. Con las siguientes sentencias finalizaríamos de configurar y activar el ADC.

```
ADC_Cmd(ADC1, ENABLE);

/* Wait until the ADC1 is ready */
while(ADC_GetFlagStatus(ADC1, ADC_FLAG_ADONS) == RESET) {}

/* Start ADC1 Software Conversion */
ADC_SoftwareStartConv(ADC1);
```

Para obtener el valor convertido utilizamos la sentencia `ADC_GetConversionValue(ADC1)` que nos devuelve el valor convertido almacenado en un registro de 16 bits.

4.7. TIMER

El STM32L15xx posee varios timers:

- **TIM2-TIM4:** Timers de propósito general de 16 bits con contador auto-recargable con preescalado programable. Son contadores ascendentes, descendentes y ascendentes/descendentes. El preescalado también es de 16 bits. Poseen 4 canales independientes para: Input Capture, Output Compare, PWM, y One-Pulse. Entre sus funciones destacan la de generar ondas (PWM y Output Compare) y medir duraciones de pulso de señales (Input Compare).
- **TIM9/10/11:** Timers de propósito general de 16 bits con contador auto-recargable con preescalado programable. Entre sus funciones destacan generar ondas (PWM y Output Compare) y medir duraciones de pulso de señales (Input Compare).
 - TIM9: Contador ascendente de 16 bits con preescalado de 16 bits. Dos canales independientes para: Input Compare, Output Compare, PWM, y One-Pulse.
 - TIM10/11: Contadores ascendentes de 16 bits con preescalado de 16 bits. Un canal independiente para: Input Compare, Output Compare y PWM.
- **TIM6&TIM7:** Timers básicos de 16 bits con preescalado programable de 16 bits. Contador ascendente de 16 bits. Se utilizan para generar bases de tiempos y para dirigir el DAC (Digital-to-Analog Converter).

En esta sección se mostrará cómo configurar un timer para operaciones básicas como Input Capture y Output Compare. Se utilizarán los Timers 2 al Timer 4 ya que son los más completos.

El principal bloque del timer es un contador de 16 bits con registro auto-recargable. El contador puede contar ascendentemente, descendentemente o ambos a la vez. El counter clock puede ser dividido por un preescalado. El preescalado divide la frecuencia del reloj por un factor entre 1 y 65536 (16 bits).

El reloj principal del Timer es el CK_INT que es el reloj del bus al que está conectado el timer, en nuestro caso el APB1. Por defecto, al arrancar el dispositivo el reloj del sistema es el MSI (2 MHz aprox.) y el bus APB1 trabaja con el mismo reloj puesto que no está dividido por ningún factor. Por lo tanto, el CK_INT tiene una frecuencia de 2 Mhz. Este reloj es el que dirige

al preescalado. A su vez, el reloj del preescalado CK_CNT, es el que dirige al contador del timer. Si no deseamos preescalado en nuestra aplicación, deberemos utilizar un preescalado de valor 1. Por contra, si deseamos que nuestro contador tenga una frecuencia menor de 2 MHz deberemos configurar el preescalado. Ello se consigue a través de la fórmula:

$$\text{PrescalerValue} = \frac{\text{CK_INT}}{\text{CK_CNT}} - 1$$

Por ejemplo, si deseamos que nuestro contador trabaje a una frecuencia de 10 kHz, el valor del preescalado sería el siguiente:

$$\text{PrescalerValue} = \frac{2097000}{10000} - 1 = 208,7$$

Entre otras funciones adicionales, también es posible la división del reloj por un factor 2, 3, 4, 9, 10 o 11. Además es posible utilizar otros timers como preescalados de timer, o utilizar señales de reloj externas para dirigir al timer pero no detallaremos esa configuración.

Cuando utilizamos Timers frecuentemente vamos a tener que hacer uso de sus canales, ya sea para entrada o salida. Para consultar qué pines están conectados a qué canales debemos consultar el datasheet de STM32L151x y STM32L152x. En él vienen indicados en la tabla 5 qué pin está conectado a qué canal de qué timer. Pero en ocasiones deseamos utilizar un pin que no tiene asignado un canal de un timer por requisitos de la aplicación. Para solucionar esto disponemos de **RI (Routing Interface)**.

RI permite un remapeo de canales de entrada a los TIM2/TIM3/TIM4. Por defecto, los cuatro canales de captura de estos tres timers aparecen conectados a los puertos GPIO especificados en el datasheet. Gracias a RI esto se puede cambiar pudiendo enrutar cualquier canal de entrada al timer, a cualquier puerto GPIO. La única limitación es que esta capacidad sólo la puede ejercer un solo timer al mismo tiempo, pero este timer, puede mapear sus cuatro canales. Esto es posible a una matriz de control del ADC.

4.7.1. Configuración Input Capture

En modo captura, los registros de comparación son utilizados para capturar el valor del contador cuando una transición es detectada en una señal. Cuando ocurre una captura, se produce

una interrupción si está habilitada. A continuación se muestra como configurar el periférico para programar un timer en modo input capture.

Lo primero que debemos realizar es activar el reloj del timer que estemos configurando. Esto lo realizamos a través de la función `RCC_APBxPeriphClockCmd(RCC_APBxPeriph_TIMx, ENABLE)` donde x es el timer que deseamos utilizar. Después, una vez hemos decidido qué canal del timer utilizar, debemos mirar en el datasheet qué pin es el que tiene asignado dicho canal. Deberemos inicializar el pin GPIO configurándolo como función alternativa. Para ello procedemos como indicamos al principio de esta sección, mediante una estructura `GPIO`, dando valores a los parámetros, inicializando la estructura, e indicando la función alternativa de dicho pin. En el siguiente código mostramos un ejemplo de configuración del pin PA0 como canal de entrada del TIM2.

```
GPIO_InitTypeDef GPIO_InitStructure;

/* Configuramos PA0 como TIM2 Channel 1 */
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_PinAFConfig(GPIOA, GPIO_PinSource0, GPIO_AF_TIM2);
```

Una vez realizado esto, deberemos comenzar a configurar el timer. Lo primero que debemos hacer es configurar la base de tiempos. Para ello tenemos la estructura `TIM_TimeBaseInitTypeDef` que dispone de los siguientes parámetros:

- **TIM_Period.** Es el valor de autorecarga, el valor en el que el contador se reinicia y comienza a contar desde cero de nuevo.
- **TIM_Prescaler.** Es el valor del prescalado. Admite valores comprendidos entre 1 y 65536 (16 bits)
- **TIM_ClockDivision.** Es el divisor del contador que puede tomar: `TIM_CKD_DIV1`, `TIM_CKD_DIV2` o `TIM_CKD_DIV4`.
- **TIM_CounterMode.** Indica el modo de contador. Puede ser: `TIM_CounterMode_Up`

- o TIM_CounterMode_Down.

Tras parametrizar la estructura, procedemos a inicializarla mediante `TIM_TimeBaseInit()`.

Un ejemplo de su uso es el siguiente:

```
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
```

Después de configurar la base de tiempos, configuramos el modo del timer, que en este caso es Input Capture. Para configurarlo, disponemos de otra estructura llamada `TIM_ICInitTypeDef`. Esta estructura tiene como parámetros:

- **TIM_Channel**. Seleccionamos el canal de entrada. Acepta como parámetro `TIM_Channel_x` donde x es el número del canal (admite del 1 al 4).
- **TIM_ICPolarity**. Indica el tipo de flanco de la señal de captura. Puede ser:
 - `TIM_ICPolarity_Rising`. Activo por flanco de subida.
 - `TIM_ICPolarity_Falling`. Activo por flanco de bajada.
 - `TIM_ICPolarity_BothEdge`. Activo por ambos flancos.
- **TIM_ICSelection**. Define la dirección del canal (entrada/salida) así como el tipo de entrada. Por defecto, lo configuramos como `TIM_ICSelection_DirectTI`.
- **TIM_ICPrescaler**. Indica cuantos eventos tienen que ocurrir para que se produzca la captura.
 - `TIM_ICPSC_DIV1`: no hay preescalado
 - `TIM_ICPSC_DIV2`: captura cada 2 eventos
 - `TIM_ICPSC_DIV4`: captura cada 4 eventos
 - `TIM_ICPSC_DIV8`: captura cada 8 eventos
- **TIM_ICfilter**. Indica la frecuencia con que la entrada es sampleada y la duración del filtro digital. Esto es válido cuando la señal de entrada no es estable. Por defecto, la configuramos como `0x0` que indica que no hay filtro.

Una vez configurada la estructura, procedemos a inicializarla mediante `TIM_ICInit()` de igual modo que todas las estructuras anteriormente vistas. El siguiente paso es activar el timer, instrucción que conseguimos mediante `TIM_Cmd(TIMx, ENABLE)` donde x es el timer que deseamos habilitar. Con esto, nuestro timer ya está funcionando según la configuración programada.

Tras configurar la base de tiempos, y el timer, debemos configurar las interrupciones si deseamos habilitarlas. El procedimiento es muy similar al indicado en el periférico ADC. Debemos habilitar la interrupción, asignar prioridad, y elegir la fuente de interrupción. Si más de una interrupción ha sido configurada, deberemos identificar el origen en la RAI. Para habilitar la interrupción lo hacemos parametrizando la estructura de NVIC como muestra el ejemplo a continuación:

```
NVIC_InitTypeDef NVIC_InitStructure;

NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Para seleccionar la fuente de interrupción utilizamos la función:

`TIM_ITConfig(TIMx, TIM_IT_CCy, ENABLE)`

donde x es el timer que deseamos configurar, e y el canal asignado.

En el fichero `stm32l1xx_it.c` debemos incluir la RAI para la interrupción del timer. El nombre de esta interrupción debe estar contemplado en el fichero `startup`. Para el caso del Timer es `TIMx_IRQHandler` donde x es el timer seleccionado. Un ejemplo de RAI se muestra a continuación:

```
void TIM2_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET) /* Channel 1 IC */
    {
        /* código que deseamos que realice la RAI */
        TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
    }
}
```

Ya tenemos configurado el timer y sus interrupciones asignadas. El último paso es la lectura

del valor capturado. Esto lo conseguimos con la función: `TIM_GetCapturex(TIMx)` donde x es el timer seleccionado.

4.7.2. Configurar Output Compare

El modo Output Compare lo utilizamos para generar una onda o cuándo un periodo de tiempo ha ocurrido. Cuando ocurre una coincidencia entre el valor almacenado y el contador, el timer genera una interrupción si está habilitada, y asigna a un pin que esté configurado un determinado valor, pudiendo hacer un set, un reset, alternar su anterior valor, y no hacer nada.

Para configurarlo, partimos del caso anterior. En él teníamos nuestro timer configurado como Input Capture en uno de los canales (debemos recordar que los timer TIM2/3/4 tienen 4 canales independientes cada uno). Vamos a utilizar esa configuración para además configurar en otro canal un Capture Compare. Hay determinadas tareas que no debemos configurarlas ya que ya lo están. Vamos a utilizar un mismo timer y una misma base de tiempos para añadir otra canal. Por lo tanto, ya tenemos configurado la base de tiempos que es un contador trabajando a una frecuencia de 10 kHz. El reloj del timer ya lo teníamos activado pero lo que sí debemos realizar es la configuración de un puerto GPIO ya que el anterior pertenecía a otro canal. Para ello configuramos como anteriormente, procurando no volver a usar el mismo del canal ya utilizado, puesto que un determinado pin GPIO sólo puede realizar un cometido al mismo tiempo. No se detalla cómo configurar un puerto GPIO como función alternativa ya que se vio recientemente.

Para configurar el timer en modo Output Compare, disponemos de otra estructura llamada `TIM_OCInitTypeDef`. Los parámetros que recibe esta estructura son:

- `textbf{TIM_OCMODE}`. Especifica el modo del Timer. Puede ser:
 - `TIM_OCMODE_Timing`. Cuando hay coincidencia, no ocurre nada en el pin de salida.
 - `TIM_OCMODE_Active`. Cuando hay coincidencia, set al pin.
 - `TIM_OCMODE_Inactive`. Cuando existe coincidencia, reset al pin.
 - `TIM_OCMODE_Toggle`. Cuando existe coincidencia, se alterna el valor anterior.
 - `TIM_OCMODE_PWM1`. Modo PWM (se ve en la sección siguiente).
- `textbf{TIM_OutputState}` Especifica si hay salida en el pin del canal seleccionado. Admite `ENABLE` o `DISABLE`.

- **textbfTIM_Pulse** Especifica el valor que se compara con el contador
- **textbfTIM_OCPolarity**. Especifica la polaridad de salida.
 - **TIM_OCPolarity_High**. Activo por nivel alto.
 - **TIM_OCPolarity_Low**. Activo por nivel bajo.

Una vez parametrizada la estructura, procedemos a inicializarla mediante:

```
TIM_OCxInit(TIMy, &TIM_OCInitStructure);
```

donde x es el canal del timer utilizado para Output Compare, e y es el timer que estamos utilizando. Es importante darse cuenta de que a la hora de inicializar la estructura para el Output Compare, es el único momento donde determinamos que canal estamos configurando. El modo Input Capture especificaba el canal en la estructura mediante un parámetro.

Si deseamos activar la interrupción del deberemos activar la fuente de interrupción por Output Compare deberemos hacerlo con:

```
TIM_ITConfig(TIMx, TIM_IT_CCy, ENABLE);
```

donde x es el timer que deseamos configurar, e y el canal asignado.

La IRQ no es necesario configurarlo ya que es el mismo que configuramos antes. En un proyecto donde un mismo timer utilice dos canales, ambos utilizan la misma IRQ pero cada uno con su fuente de interrupción. Si en un proyecto tenemos solo un canal en modo Output Compare, sí deberíamos configurar la IRQ de la misma forma que hicimos anteriormente.

Ya sólo hace falta modificar la RAI que escribimos antes.

```
void TIM2_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET) /* Channel 1 IC */
    {
        /* código que deseamos que realice la RAI */
        /* Borramos el bit pendiente de interrupción */
        TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);

    }
    else { /* Channel 2 OC */

        /* código para la interrupción por OCC */
        /* Borramos el bit pendiente de interrupción */
    }
}
```

```
    TIM_ClearITPendingBit (TIM2, TIM_IT_CC2) ;  
}  
}
```

Nota: Para configurar los canales del timer, así como parámetros del mismo el timer debe estar deshabilitado.

4.7.3. External Trigger

Podemos utilizar el Timer para tareas de sincronización como dirigir las conversiones del ADC o el DAC. Para ello debemos utilizaremos la función:

```
TIM_SelectOutputTrigger(TIMx, TIM_TRGOSource);
```

donde:

- x puede tomar el valor 2, 3, 4, 6, 7 o 9 para seleccionar el timer.
- TIM_TRGO puede tomar el valor:
 - TIM_TRGOSource_Reset: El bit UG se utiliza como trigger output (TRGO).
 - TIM_TRGOSource_Enable: El Counter Enable CEN es utilizado como trigger outputThe Counter Enable CEN is used as the trigger output.
 - TIM_TRGOSource_Update: El reseteo del contador por auto-recarga se utiliza como trigger output.

4.8. PULSE WIDTH MODULATOR (PWM)

Pulse Width Modulator (PWM) nos permite generar señales con una frecuencia determinada y un ciclo de trabajo determinado. Aunque está dentro del Timer, se ha decidido tratarlo como si fuera un periférico individual. Conociendo la configuración del timer en Output Compare, es relativamente sencillo configurarlo para el modo PWM.

Tal y como vimos al comienzo de la sección del timer, sólo poseen modo PWM los timers TIM2/3/4 y TIM9/10/11. Suponemos que estamos configurando un timer del primer grupo. Para el lector, suponemos también que se conoce el funcionamiento de un PWM y se obviará su explicación funcional.

Para configurar el timer en modo PWM debemos:

- Habilitar el reloj del timer que vayamos a utilizar
- Habilitar el reloj del puerto GPIO del pin que vayamos a utilizar como canal de salida.
Además, deberemos configurar este pin GPIO en modo función alternativa.
- Configurar la base de tiempos, configurando preescalado si deseamos. En el parámetro periodo deberemos indicar el periodo de nuestra señal.

Ahora tenemos que configurar el canal en modo Output PWM. Primero deberemos crear la estructura y configurar tal y como se muestra a continuación:

```
TIM_OCInitTypeDef  TIM_OCInitStructure;

TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = Duty_Cycle;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OC1Init(TIM1, &TIM_OCInitStructure);
```

Este es un ejemplo donde se configura el canal 1 en modo Output PWM. El PWM seleccionado es el 1 (Los timer TIM2/3/4 tiene dos PWM). Activamos la salida en el pin donde se encuentra el canal del timer y activo por nivel alto. El único parámetro que hay darle valor de los que aparecen en el ejemplo es el Duty_Cycle que deberá ser una fracción del periodo de nuestra

señal. Admite cualquier valor pero lógicamente, este deberá ser menor al del periodo de nuestra señal o el PWM no funcionará correctamente.

En el caso del PWM, antes de poder activar el Timer debemos realizar dos operaciones previas. Éstas son activar el Preload Register y activar el Auto-reload Register. Con el siguiente código se muestra cómo realizar dicha tarea:

```
TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);  
TIM_ARRPreloadConfig(TIM1, ENABLE);  
TIM_Cmd(TIM1, ENABLE);
```

En el caso del PWM, si deseamos cambiar algún parámetro, no sólo deberemos deshabilitar el timer, sino también estos dos registros.

El funcionamiento de las excepciones funciona exactamente igual que en el caso de Output Compare.

4.9. DIGITAL-TO-ANALOG CONVERTER (DAC)

El DAC (Digital-to-Analog Converter) es un dispositivo que nos permite realizar la función opuesta al ADC, es decir, convertir palabras digitales en voltaje.

El DAC del STM32L15xx es un conversor de 12 u 8 bits, con dos canales de salida, cada uno de ellos con su propio conversor. En modo dual las conversiones pueden realizarse en modo simultaneo o independiente. Entre sus principales aplicaciones están generar ondas o control de ingeniería.

El DAC además integra dos buffers de salida (uno por cada canal) que se puede utilizar para reducir la impedancia de salida, y no tener la necesidad de utilizar amplificadores operaciones.

Los formatos de datos que admite el DAC son:

- 8 bits con alineación a la derecha.
- 12 bits con alineación a la izquierda.
- 12 bits con alineación a la derecha. Utilizaremos este formato por defecto.

La inicialización de la conversión puede realizarse por:

- Por software.
- Mediante la salida de un timer. La interfaz del DAC detecta un flanco en el TRGO (trigger Output) del timer seleccionado y realiza una conversión.
- A través de una línea de interrupción externa.

Las entradas digitales son convertidas en voltajes mediante una conversión lineal entre 0 y Vref. El voltaje de salida queda determinado por:

$$DACoutput = Vref * \frac{Ydigital}{4095}$$

donde Ydigital es el valor digital almacenado.

El DAC puede utilizarse en conjunto con el DMA (Direct Memory Access). El uso del DMA se explica porque es necesario evitar tener que hacer operaciones de traspaso de datos entre memoria y periférico. Gracias al DMA, es éste el que se encarga de realizar dichas operaciones, y no la propia aplicación por software.

El DAC puede generar dos formas de onda por sí solo. Una de ellas es la triangular y otra la generación de ruido blanco.

White Noise Generation

Para generar pseudo código de amplitud variable, un LFSR (Linear Feedback Shift Register) está disponible. Genera secuencias de 2^{n-1} números. El ruido producido puede ser considerado ruido blanco aunque tiene distribución uniforme.

Onda Triangular

Con el DAC es posible también generar ondas triangulares de amplitud, frecuencia y offset variable. Para variar la frecuencia debemos variar la frecuencia del timer que controla el DAC.

El modo de generación de onda triangular funciona de la siguiente manera: El modo triangular tiene un contador interno. Por cada tick recibido del external output del timer, este contador se incrementa unitariamente. El valor de inicio es el offset establecido para la señal. A partir de este valor, el contador irá incrementando con cada tick hasta llegar al valor de amplitud máxima también determinado en la configuración del DAC. Una vez alcanzado este valor, comenzará a descender hasta llegar al valor de offset y así sucesivamente.

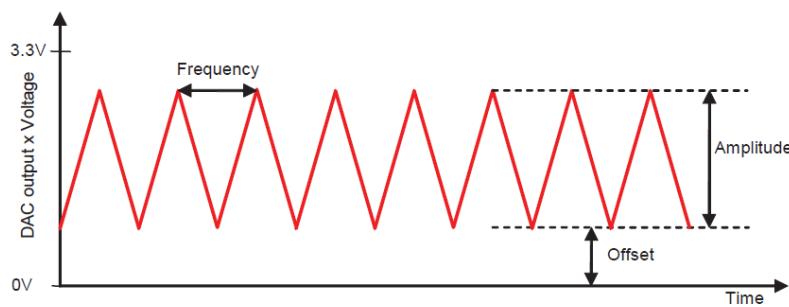


Figura 4.1: *Onda Triangular*

Por este motivo, la frecuencia queda determinada tanto por el tick del timer, como por el valor de la amplitud, puesto que a mayor amplitud, menor frecuencia ya que el contador interno

del modo triangular necesitará más ticks hasta llegar al valor de amplitud máxima. Debido a que STMicroelectronics no facilita información más detallada sobre este funcionamiento, no es posible determinar la frecuencia de la onda de forma teórica.

A continuación se muestra cómo configurar el periférico DAC. Primeramente, debemos:

- Activar el reloj del puerto GPIO del pin que vayamos a utilizar. El pin que utilicemos será la salida del canal del DAC. Estos serán o bien el pin PA4 o el PA5. Deberemos configurarlos en modo analógico.
- Activar el reloj del periférico DAC a través de:

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
```

Tras esto, deberemos decidir sobre cuál será la fuente que controle al DAC. Debemos recordar que es posible iniciar las conversiones a través del timer, de una línea de interrupción externa, y por software.

Para configurar el timer, deberemos crear su escala de tiempos. El timer controlará la frecuencia de la onda generada por el DAC. Para configurar un timer y su escala de tiempos, debemos seguir los pasos indicados en la sección timer. Antes de habilitar el timer, deberemos configurar su salida como trigger external, como se vió en la sección timer. Un ejemplo para el TIM2 y una frecuencia de 180 kHz se muestra a continuación:

```
/* Activamos el reloj del timer */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

/* Time base configuration */
TIM_TimeBaseStructure.TIM_Period = 10;
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

/* Configuramos el timer como trigger external */
TIM_SelectOutputTrigger(TIM2, TIM_TriggerSource_Update);

/* Habilitamos el TIM2 */
TIM_Cmd(TIM2, ENABLE);
```

La frecuencia del tick del trigger output queda determinada por el valor de auto-recarga del contador:

$$TICK_TRGO = \frac{CK_CNT}{ARR + 1}$$

que en nuestro caso:

$$TICK_TRGO = \frac{2MHz}{ARR + 1} = 180kHz \Rightarrow ARR = 10$$

Una vez configurado el timer para controlar las conversiones del DAC, explicamos como configurar el DAC.

4.9.1. Onda Triangular

Para generar una onda triangular podemos utilizar las funciones del propio periférico. Lo primero que deberemos hacer al igual que el resto de los periféricos ya vistos, es crear su estructura, darle valores e inicializarla. La estructura para el DAC es DAC_InitTypeDef y un ejemplo es el siguiente:

```
DAC_InitTypeDef      DAC_InitStructure;

/* Configuracion DAC Channel 1 */
DAC_InitStructure.DAC_Trigger = DAC_Trigger_T2_TRGO;
DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_Triangle;
DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude = DAC_TriangleAmplitude_1023;
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;

DAC_Init(DAC_Channel_1, &DAC_InitStructure);
```

La estructura puede recibir los siguientes parámetros:

- **DAC_Trigger**. Selecciona la fuente trigger. Sus valores pueden ser:
 - DAC_Trigger_Tx_TRGO donde x puede ser 2, 4, 7 o 9.
 - DAC_Trigger_Ext_IT9
 - DAC_Trigger_Software
- **DAC_WaveGeneration**. Selecciona la onda a generar: ruido, triangular o ninguna.

- DAC_WaveGeneration_None
 - DAC_WaveGeneration_Noise
 - DAC_WaveGeneration_Triangle
- **DAC_LFSRUnmask_TriangleAmplitude.** Selecciona la amplitud de la onda triangular. Consultar Tabla 4.2 para una Vref de 3V.
- **DAC_Outputbuffer.** Habilita o deshabilita el buffer de salida del canal seleccionado. Admite DAC_OutputBuffer_Enable o DAC_OutputBuffer_Disable.

Amplitud Digital	Amplitud Analógica (V)
DAC_TriangleAmplitude_1	0.0016
DAC_TriangleAmplitude_3	0.0032
DAC_TriangleAmplitude_7	0.0064
DAC_TriangleAmplitude_15	0.0128
DAC_TriangleAmplitude_31	0.0257
DAC_TriangleAmplitude_63	0.0515
DAC_TriangleAmplitude_127	0.1031
DAC_TriangleAmplitude_255	0.2062
DAC_TriangleAmplitude_511	0.4125
DAC_TriangleAmplitude_1023	0.8250
DAC_TriangleAmplitude_2047	1.6483
DAC_TriangleAmplitude_4095	3.300

Tabla 4.2: Amplitudes Señal Triangular

Después de configurar e inicializar el periférico DAC debemos habilitarlo mediante:

`DAC_Cmd(DAC_Channel_x, ENABLE)`

donde x puede ser 1 o 2 haciendo referencia al canal del DAC. Al activar el timer, el pin asociado al canal (PA4 o PA5) se conectará automáticamente al DAC converter. Después debemos indicar el valor base para la señal de salida, es decir, el valor mínimo que tomará la salida del DAC.

Para establecer este valor deberemos obedecer la fórmula:

$$DACoutput = Vref * \frac{DOR}{4095}$$

Si deseamos un valor de DACoutput de 0.1 Voltio, deberemos escribir en el DOR un valor 124, asumiendo:

- el formato de datos es de 12 bits, por eso el valor máximo es 4095.
- Vref son 3.3 Voltios (por defecto).

El siguiente código muestra un ejemplo para el canal 1:

```
/*Activamos el DAC Channel 1 */
DAC_Cmd(DAC_Channel_1, ENABLE);

/* Establecemos el DAC channel 1 DHR12RD register */
DAC_SetChannel1Data(DAC_Align_12b_R, 0x124);
```

Con esto ya tenemos el DAC configurado para una onda triangular.

4.9.2. Onda senoidal

Con el DAC podemos generar ondas senoidales de un tono. Para ello, no deberemos configurar el DAC para generar ninguna señal puesto que no nos interesa ni onda triangular ni ruido. Para la generación de este tipo de onda, nos ayudaremos del siguiente ejemplo, en el que generaremos una onda senoidal a partir de 10 muestras de una onda que varía desde 0 a 2*PI.

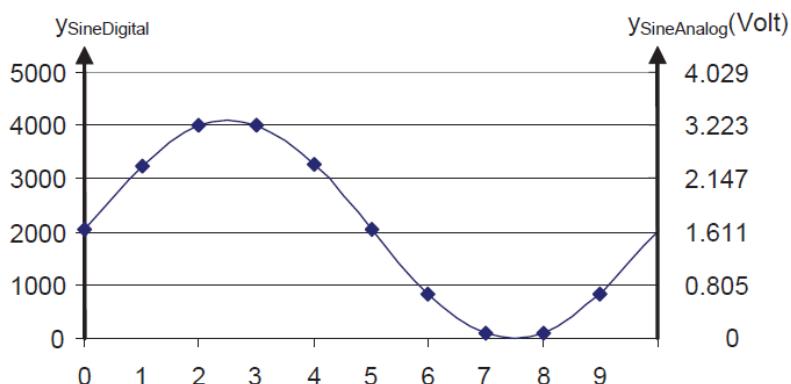


Figura 4.2: Onda Senoidal

El paso de muestreo es $(2\pi)/n_s$ (número de muestras).

El resultado de $\sin(x)$ se encuentra acotado entre -1 y 1, y debemos recalibrar a valores positivos, haciendo que varia entre valores 0 y 0xFFFF, y que se corresponda al rango entre 0 y 3.3 Voltios (Vref).

$$Y_{digital}(x) = (\sin((x * \frac{2\pi}{n_s}) + 1))(\frac{(0xFFFF + 1)}{2})$$

La onda analógica se pueden determinar mediante la siguiente relación:

$$Y_{analogica}(x) = 3,3 * \frac{Y_{digital}(x)}{0xFFFF}$$

Dando valores a x para nuestras 10 muestras obtenemos una relación entre valores digitales y valores de salida analógicos. Estos quedan reflejados en Tabla 4.4

Sample(x)	Ydigital(x)	Yanalógica(x)
Sample(x)	Ydigital(x)	Yanalógica(x)
0	2048	1.650
1	3251	2.620
2	3995	3.219
3	3996	3.220
4	3253	2.622
5	2051	1.653
6	847	0.682
7	101	0.081
8	98	0.079
9	839	0.676

Tabla 4.4: Valores digitales y analógicos

La frecuencia de la onda senoidal queda determinada por la frecuencia del tick del timer, y por el número de muestras mediante la siguiente relación:

$$F_{seno} = \frac{F_{TRGO}}{n_s}$$

Para tener mayor resolución de la onda, es conveniente aumentar el número de muestras.

Por lo tanto, para crear una onda que no contemple el DAC basta con crear un vector de muestras y pasárselo al DAC. Para generar un vector de muestras nos podemos ayudar de cualquier herramienta como "Matlab" o "Octave". Basta con generar un número de muestras suficientes para representar dicha función en un periodo.

Dicho vector de muestras deberemos transferirlo de la memoria al DAC mediante el DMA (Direct Memory Access). El paso de muestras se realiza con la misma frecuencia que el trigger output del timer.

Para configurar el DAC para generar una onda senoidal procederemos como se detalla a continuación.

Lo primero que deberemos hacer al igual que el resto de los periféricos ya vistos, es crear su estructura, darle valores e inicializarla. La estructura para el DAC es DAC_InitTypeDef y un ejemplo es el siguiente:

```
DAC_InitTypeDef      DAC_InitStructure;

/* Configuracion DAC Channel 1 */
DAC_InitStructure.DAC_Trigger = DAC_Trigger_T2_TRGO;
DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;

DAC_Init(DAC_Channel_1, &DAC_InitStructure);
```

Los parámetros de la estructura son los mismos que vimos anteriormente para la onda triangular pero esta vez no deseamos que el propio DAC genere ninguna onda.

El siguiente paso es configurar el DMA. El DMA es un periférico bastante complejo y no se contempla en este proyecto estudiarlo al igual que el resto de periféricos. Sin embargo, es necesario utilizarlo para poder generar este tipo de ondas. Por ese motivo, se proporciona un ejemplo a continuación que sirve para generar este tipo de ondas, y solamente es necesario cambiar algún parámetro.

```

DMA_DeInit(DMA1_Channel2);
DMA_InitStructure.DMA_PeripheralBaseAddr = Memory_Address; /* parámetro modificable */
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&Vector_muestras; /* parametro
modificable */
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
DMA_InitStructure.DMA_BufferSize = num_muestras; /* parámetro modificable */
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA1_Channel2, &DMA_InitStructure);

```

Como vemos, el funcionamiento de inicialización es el mismo que con el resto de periféricos ya estudiados. Los parámetros que editaremos son los siguientes:

- **DMA_InitStructure.DMA_PeripheralBaseAddr**. Aquí debemos indicar la dirección donde el DAC debe escribir los datos del vector de muestras. Recordamos que teníamos varios formatos de datos: 8b alineados a la derecha, 12b alineados a la derecha y 12b alineados a la izquierda. Además, teníamos los modos de configuración dual y sencillo. En este parámetro deberemos indicarle la dirección de memoria donde escribir. Por ejemplo: Si estamos utilizando 12 bits alineado a la izquierda en modo sencillo, le pasaremos la dirección de memoria del registro DAC_DHR12R1. En nuestro caso siempre configuraremos este modo cuya dirección en memoria es: 40007408.
- **DMA_InitStructure.DMA_MemoryBaseAddr**. En este parámetro debemos indicar el vector de muestras de nuestra señal, y debe contener tantos elementos como indique el parámetro DMA_BufferSize. Debe ser un vector ya creado en memoria, cuyos valores no pueden exceder los 12 bits indicados en el formato de datos.
- **DMA_InitStructure.DMA_BufferSize**. Aquí debemos indicar el numero de muestras de nuestro vector.

Es importante conocer qué canal DMA debemos asignar al DAC. Cada canal DAC tiene su propio canal DMA, y no podemos asignar uno cualquiera.

- Si estamos trabajando con el DAC_Channel1, usaremos el DMA_Channel2.
- Si estamos trabajando con el DAC_Channel2, usaremos el DMA_Channel3.

El resto de parámetro no es necesario modificarlos para el propósito de generar ondas mediante el DAC. Con este ejemplo de código tendríamos inicializado la estructura. Ahora debemos inicializar el DAC. Para ello, las siguientes instrucciones nos permiten arrancar el DAC y el canal DMA asociado a él.

```
/* Activamos DMA1\_\_Channel2 */
DMA_Cmd(DMA1_Channel2, ENABLE);

/* Activamos DAC\_\_Channel2 */
DAC_Cmd(DAC_Channel_1, ENABLE);

/* Activamos sincronización entre DMA y DAC */
DAC_DMACmd(DAC_Channel_1, ENABLE);
```

Con ello ya tendríamos el DAC generando la onda seleccionado por el pin que configuramos anteriormente.

Si deseamos detener el DMA para reconfigurar alguno de sus parámetros, como pasarle un vector de muestras diferente, lo haremos con la siguiente función:

DMA_DeInit(DMA1_Channelx)

donde x es el canal DMA seleccionado.

Si lo que deseamos es detener el DAC, primero deberemos detener el canal DAC configurado, y luego el periférico:

```
DAC_Cmd(DAC_Channel_x, DISABLE);
DAC_DeInit();
```

EVALUACIÓN

Para realizar una evaluación de la placa se ha decidido realizar varios aplicaciones utilizando diversos periféricos. Estos periféricos han sido configurados para realizar tareas concretas y sencillas, puesto que el interés en esta evaluación es demostrar la capacidad del microcontrolador para desarrollar ejemplos y aplicaciones sencillas que puedan servir de ejemplo al estudio del periférico a la hora de desarrollar aplicaciones mayores.

Para todos los ejemplos mostrados a continuación la configuración de la placa STM32L-DISCOVERY debe ser la siguiente:

- El jumper JP1 debe estar en posición ON.
- Los jumpers en CN3 deben estar puestos para permitir la comunicación del STM32L152x con el debugger ST-Link mediante interfaz SWD.

5.1. GPIO

VISIÓN GENERAL

El objetivo de la aplicación desarrollada es alternar el encendido de los leds LD3 y LD4 de la placa STM32L-Discovery. Tal efecto se consigue pulsando el botón USER, de tal forma que cada vez que se pulse este botón, uno de los leds se encenderá y el otro se apagará. Al mismo tiempo, se implementará un contador de pulsaciones que se mostrará de forma numérica por el display LCD.

Para llevar a cabo tal funcionamiento, se han seguido los siguientes hitos:

- Encendido y apagado de leds a través de los registros BSRR (Port bit set/reset register).
- Reconocimiento del botón USER para detectar si está pulsado o no.
- Implementación de un sistema antirrebote a través del Systick, que contemple la auto-repetición (detectar si la tecla se mantiene pulsada).
- Generación de mensajes en el display LCD, en modo estático y en modo scroll.

- Implementación de un contador para el botón USER, que cuente el número de veces que el botón ha sido pulsado.

Descripción funcional: La aplicación arrancará mostrando un mensaje en el display LCD en modo scroll indicando “EJEMPLO GPIO”. Tras esto, otro mensaje en el display indicando “Pulsa” pedirá que pulsemos el botón USER. En el momento en el que pulsemos el botón, uno de los leds (LD3 o LD4) se encenderá, y el display LCD mostrará el número 1. Si volvemos a pulsar el botón USER, el led que estaba encendido se apagará y el restante se encenderá, mostrando el display LCD el número 2, y así sucesivamente.

5.1.1. ¿CÓMO UTILIZAR LA APLICACIÓN?

Seleccionar el workspace utilizado en este proyecto como el workspace de Atollic. Una vez seleccionado y con Atollic arrancado, abrir el proyecto GPIO. Para arrancar el programa basta con compilar todos los ficheros y entrar en modo Debug.

Alternativamente, se puede crear un nuevo proyecto con Atollic utilizando el tutorial (ver tutorial Atollic) y sustituir la carpeta /src por la carpeta GPIO/src.

5.1.2. DESCRIPCIÓN DEL SOFTWARE

Periféricos utilizados por la aplicación

Esta aplicación utiliza los siguientes periféricos con la configuración detallada a continuación:

- **GPIO:**
 - PA0 como entrada floating pin que está conectado al botón USER
 - PB6 (LED4 azul) y PB7 (LED3 verde) como salida push-pull
 - PA1, PA2, PA3, PA8, PA9, PA10, PA15, PB3, PB4, PB5, PB8, PB9, PB10, PB11, PB12, PB13, PB14, PB15, PC0, PC1, PC2, PC3, PC6, PC7, PC8, PC9, PC10 y PC11 están configurados como función alternativa, segmentos del LCD.
- **LCD:** Diferentes funciones disponibles del driver del controlador LCD son utilizadas para inicializar, borrar, mostrar strings y mostrar strings en modo scroll.
- **SYSTICK Timer:** El timer Systick es usado para generar un delays con un tiempo base de 1ms.

- **Reloj:** El oscilador MSI (Multi Speed Internal oscillator) es seleccionado como reloj.

Contenido del directorio

- GPIO/src/stm32l1xx_conf.h Fichero configuración de la librería
- GPIO/src/stm32l1xx_it.c Gestión de interrupciones
- GPIO/src/stm32l1xx_it.h Cabeceras del fichero de manejo de interrupciones
- GPIO/src/main.c Aplicación principal. Contiene el main.
- GPIO/src/system_stm32l1xx.c STM32L1xx fichero de sistema
- GPIO/src/gpio_user.c Fichero de usuario con funciones de gpio
- GPIO/src/gpio_user.h Fichero de cabeceras del fichero gpio de usuario
- GPIO/src/lcd_user.c Fichero de usuario con funciones del display LCD
- GPIO/src/lcd_user.h Fichero de cabeceras del fichero lcd de usuario
- GPIO/src/systick_user.c Fichero de funciones del systick de ARM
- GPIO/src/systick_user.h Fichero de cabeceras del fichero systick de usuario

5.1.3. CÓDIGO DE LA APLICACIÓN

El código de la aplicación se encuentra bajo la carpeta GPIO/src y está disponible en el CD que se adjunta con la memoria.

5.2. INTERRUPCIÓN EXTERNA

VISIÓN GENERAL

El objetivo de esta aplicación es el mismo que el del ejemplo anterior. La diferencia reside en que en este caso se utilizará el periférico EXTINT y el controlador de interrupciones NVIC para detectar cuando el botón USER esté pulsado.

Para llevar a cabo tal funcionamiento, se han seguido los siguientes hitos:

- Se configura una línea de interrupción externa asociada al pin PA0 para detectar cuando la tecla está pulsada.
- Configuración de una IRQ para la línea de interrupción externa.

Descripción funcional: La aplicación arrancará mostrando un mensaje en el display LCD en modo scroll indicando “EJEMPLO GPIO”. Tras esto, otro mensaje en el display indicando “Pulsa” pedirá que pulsemos el botón USER. En el momento en el que pulsemos el botón, uno de los leds (LD3 o LD4) se encenderá, y el display LCD mostrará el número 1. Si volvemos a pulsar el botón USER, el led que estaba encendido se apagará y el restante se encenderá, mostrando el display LCD el número 2, y así sucesivamente.

5.2.1. ¿CÓMO UTILIZAR LA APLICACIÓN?

Seleccionar el workspace utilizado en este proyecto como el workspace de Atollic. Una vez seleccionado y con Atollic arrancado, abrir el proyecto EXTI. Para arrancar el programa basta con compilar todos los ficheros y entrar en modo Debug.

Alternativamente, se puede crear un nuevo proyecto con Atollic utilizando el tutorial (ver tutorial Atollic) y sustituir la carpeta /src por la carpeta EXTI/src.

5.2.2. DESCRIPCIÓN DEL SOFTWARE

Periféricos utilizados por la aplicación

Esta aplicación utiliza los siguientes periféricos con la configuración detallada a continuación:

- **GPIO:**

- PA0 mapea la línea de interrupción externa EXTI0
- PB6 (LED4 azul) y PB7 (LED3 verde) como salida push-pull
- PA1, PA2, PA3, PA8, PA9, PA10, PA15, PB3, PB4, PB5, PB8, PB9, PB10, PB11, PB12, PB13, PB14, PB15, PC0, PC1, PC2, PC3, PC6, PC7, PC8, PC9, PC10 y PC11 están configurados como función alternativa, segmentos del LCD.

- **LCD:** Diferentes funciones disponibles del driver del controlador LCD son utilizadas para inicializar, borrar, mostrar strings y mostrar strings en modo scroll.

- **SYSTICK Timer:** El timer Systick es usado para generar un delays.

- **Reloj:** El oscilador MSI (Multi Speed Internal oscillator) es seleccionado como reloj.

- **EXTI** Línea de interrupción externa EXTI0 mapeado en el pin PA0. Genera una interrupción por flanco de subida al pulsar el botón USER.

- **NVIC** Configura una IRQ para la línea de interrupción externa EXTI0.

Contenido del directorio

- EXTI/src/stm32l1xx_conf.h Fichero configuración de la librería
- EXTI/src/stm32l1xx_it.c Manejador de interrupciones
- EXTI/src/stm32l1xx_it.h Cabeceras del fichero de manejo de interrupciones
- EXTI/src/main.c Aplicación principal. Contiene el main.
- EXTI/src/system_stm32l1xx.c STM32L1xx fichero de sistema
- EXTI/src/gpio_user.c. Fichero de usuario con funciones de gpio
- EXTI/src/gpio_user.h Fichero de cabeceras del fichero gpio de usuario
- EXTI/src/lcd_user.c Fichero de usuario con funciones del display LCD
- EXTI/src/lcd_user.h Fichero de cabeceras del fichero lcd de usuario
- EXTI/src/systick_user.c Fichero de funciones del systick de ARM

- EXTI/src/systick_user.h Fichero de cabeceras del fichero systick de usuario
- EXTI/src/exti_user.c Fichero de funciones del periférico EXTI con funciones de usuario.
- EXTI/src/exti_user.h Fichero de cabeceras del fichero exti de usuario

5.2.3. CÓDIGO DE LA APLICACIÓN

El código de la aplicación se encuentra bajo la carpeta EXTI/src y está disponible en el CD que se adjunta con la memoria.

5.3. ADC

VISIÓN GENERAL

El objetivo de esta aplicación es la utilización del ADC para convertir un voltaje de entrada y mostrarlo por pantalla. Para ello, será necesaria la conexión de la placa con un divisor de tensión, a través de un potenciómetro de $10\text{ k}\Omega$ con una patilla a tierra, tal y como muestra la (Figura 5.1).

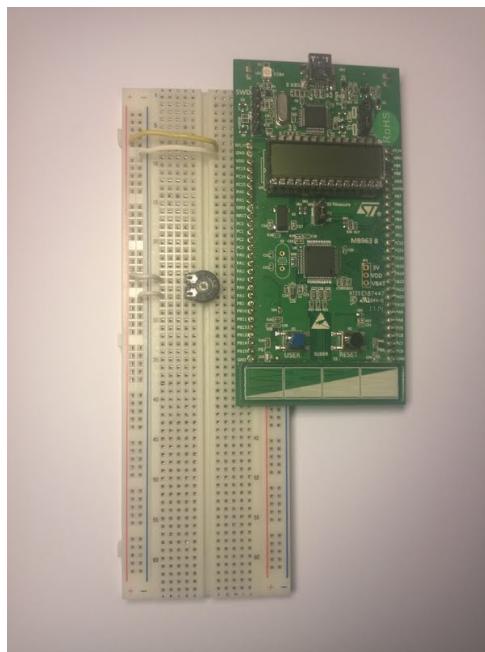


Figura 5.1: Potenciómetro conectado a placa STM32L-DISCOVERY

Para llevar a cabo tal funcionamiento, se han seguido los siguientes hitos:

- Conexionado de la placa STM32L-Discovery al circuito divisor de tensión arriba indicado.
Comprobar mediante voltímetro el correcto funcionamiento electrónico de dicho circuito.
- Configuración de un canal de entrada del ADC, configurándolo en modo continuo.
- Configuración del watchdog del ADC para detectar umbrales mínimos y máximos de tensión de entrada mediante interrupción.
- Mostrar por pantalla LCD el valor nominal de la tensión convertido a través de los registros del ADC.

Configuración ADC

Se muestreará de forma continua la entrada analógica. Para ello, se configurará un pin GPIO en modo analógico y se configurará el canal del ADC para muestrear la entrada analógica de este pin, que a su vez está conectado a un potenciómetro conectado entre la tensión de alimentación EXT_3V y masa.

En el caso de que la tensión de entrada supere se salga del rango admitido de 0,5 V a 2.75 V, el watchdog del ADC se activará por interrupción.

Descripción funcional: La aplicación arranca mostrando un mensaje por el display LCD en modo scroll indicando “EJEMPLO ADC”. Tras ello comenzará a mostrar el voltaje de entrada por pantalla. Este voltaje debe ser un valor entre 0 y 3 Voltios. Además, hay implementado un analog watchdog con dos umbrales configurados, uno inferior y otro superior. Estos umbrales están configurados a 2.75 Voltios el superior, y 0.5 Voltios el inferior. Si el voltaje excede por encima o por debajo de estos umbrales, una interrupción es lanzada pidiéndole al usuario que vuelva a alimentar el pin analógico con una tensión admitida. Para ello deberá introducir un valor correcto con el potenciómetro y pulsar el botón usuario, para volver al funcionamiento normal de la aplicación. Si se pulsa el botón USER con una tensión fuera de rango, el sistema seguirá sin mostrar el voltaje por pantalla.

5.3.1. ¿CÓMO UTILIZAR LA APLICACIÓN?

Seleccionar el workspace utilizado en este proyecto como el workspace de Atollic. Una vez seleccionado y con Atollic arrancado, abrir el proyecto ADC. Para arrancar el programa basta con compilar todos los ficheros y entrar en modo Debug.

Alternativamente, se puede crear un nuevo proyecto con Atollic utilizando el tutorial (ver tutorial Atollic) y sustituir la carpeta /src por la carpeta ADC/src.

Para esta aplicación, deberemos conectar un potenciómetro de $10\text{ k}\Omega$ entre la patilla EXT_3V, GND y el pin PA4 donde tenemos configurado la entrada analógica del ADC.

5.3.2. DESCRIPCIÓN DEL SOFTWARE

Periféricos utilizados por la aplicación

Esta aplicación utiliza los siguientes periféricos con la configuración detallada a continuación:

- **GPIO:**
 - PA4 configurado en modo analógico como canal de entrada al ADC.
 - PA0 mapea la línea de interrupción externa EXTI0
 - PA1, PA2, PA3, PA8, PA9, PA10, PA15, PB3, PB4, PB5, PB8, PB9, PB10, PB11, PB12, PB13, PB14, PB15, PC0, PC1, PC2, PC3, PC6, PC7, PC8, PC9, PC10 y PC11 están configurados como función alternativa, segmentos del LCD.
- **LCD:** Diferentes funciones disponibles del driver del controlador LCD son utilizadas para inicializar, borrar, mostrar strings y mostrar strings en modo scroll.
- **SYSTICK Timer:** El timer Systick es usado para generar un delays.
- **Reloj:** El oscilador MSI (Multi Speed Internal oscillator) es seleccionado como reloj.
- **EXTI:** Línea de interrupción externa EXTI0 mapeado en el pin PA0. Genera una interrupción por flanco de subida al pulsar el botón USER.
- **ADC:** ADC configurado en modo continuo con analog watchdog. Configurado un canal analógico en el pin PA4.
- **NVIC** Configura una IRQ para la línea de interrupción externa EXTI0 y otra para el analog watchdog del ADC.

Contenido del Directorio

- ADC/src/stm32l1xx_conf.h Fichero configuración de la librería
- ADC/src/stm32l1xx_it.c Manejador de interrupciones
- ADC/src/stm32l1xx_it.h Cabeceras del fichero de manejo de interrupciones
- ADC/src/main.c Aplicación principal. Contiene el main.

- ADC/src/system_stm32l1xx.c STM32L1xx fichero de sistema
- ADC/src/gpio_user.c. Fichero de usuario con funciones de gpio
- ADC/src/gpio_user.h Fichero de cabeceras del fichero gpio de usuario
- ADC/src/lcd_user.c Fichero de usuario con funciones del display LCD
- ADC/src/lcd_user.h Fichero de cabeceras del fichero lcd de usuario
- ADC/src/systick_user.c Fichero de funciones del systick de ARM
- ADC/src/systick_user.h Fichero de cabeceras del fichero systick de usuario
- ADC/src/exti_user.c Fichero de funciones del periférico EXTI con funciones de usuario.
- ADC/src/exti_user.h Fichero de cabeceras del fichero exti de usuario
- ADC/src/adc_user.c Fichero de funciones del periférico ADC con funciones de usuario.
- ADC/src/adc_user.h Fichero de cabeceras del fichero ADC de usuario

5.3.3. CÓDIGO DE LA APLICACIÓN

El código de la aplicación se encuentra bajo la carpeta ADC/src y está disponible en el CD que se adjunta con la memoria.

5.4. TIMER

VISIÓN GENERAL

El objetivo de esta aplicación es la implementación de un cronómetro, con capacidad para medir tiempos entre 0 y 60 segundos con resolución de 1 ms. Para llevar a cabo tal funcionamiento, se han seguido los siguientes hitos:

- Configuración del timer para generar una base de tiempos de 1 KHz, utilizando para ello preescalado.
- Configuración de un canal Input Capture asociado al botón USER. El timer capturará el tiempo del contador cuando se pulse el botón.
- Mostrar por display LCD el valor capturado por el Input Capture.
- Configuración de un canal Outuput Compare, implementando un watchdog para avisar el overflow del contador.
- Implementar una cuenta atrás de 3 a 1 antes de que el timer comience a funcionar para avisar del inicio del cronómetro.

Configuración del TIMER

Para configurar adecuadamente el TIMER se deberá configurar su escala de tiempos. El contador del timer deberá tener una frecuencia de 1 kHz y ser capaz de contar hasta 60 segundos.

Para configurar el canal Input Capture se deberá configurar el pin GPIO en modo función alternativa para este canal del timer.

El canal Output Compare estará configurado sin salida hardware.

Descripción Funcional: La aplicación arranca mostrando un mensaje por el display LCD en modo scroll indicando “EJEMPLO TIMER”. Tras ello, en el display LCD aparecerá una cuenta atrás desde 3 a 1 para avisar del comienzo del contador. Inmediatamente después, en el display aparecerá el mensaje “PULSA” y contador se pondrá en marcha. Cada vez que pulsemos el botón USER se mostrará el tiempo transcurrido en la pantalla, con una resolución de 3 decimales. En

el segundo 50 aparecerá un mensaje “ALERTA” indicando overflow. Transcurridos 60 segundos, el contador volverá a comenzar desde cero.

5.4.1. ¿CÓMO UTILIZAR LA APLICACIÓN?

Seleccionar el workspace utilizado en este proyecto como el workspace de Atollic. Una vez seleccionado y con Atollic arrancado, abrir el proyecto TIM. Para arrancar el programa basta con compilar todos los ficheros y entrar en modo Debug.

Alternativamente, se puede crear un nuevo proyecto con Atollic utilizando el tutorial (ver tutorial Atollic) y sustituir la carpeta /src por la carpeta TIM/src.

5.4.2. DESCRIPCIÓN DEL SOFTWARE

Periféricos utilizados por la aplicación

Esta aplicación utiliza los siguientes periféricos con la configuración detallada a continuación:

- **GPIO:**
 - PA0 configurado como como GPIO función alternativa: TIM2 Channel 1.
 - PA1, PA2, PA3, PA8, PA9, PA10, PA15, PB3, PB4, PB5, PB8, PB9, PB10, PB11, PB12, PB13, PB14, PB15, PC0, PC1, PC2, PC3, PC6, PC7, PC8, PC9, PC10 y PC11 están configurados como función alternativa, segmentos del LCD.
- **LCD:** Diferentes funciones disponibles del driver del controlador LCD son utilizadas para inicializar, borrar, mostrar strings y mostrar strings en modo scroll.
- **SYSTICK Timer:** El timer Systick es usado para generar un delays.
- **Reloj:** El oscilador MSI (Multi Speed Internal oscillator) es seleccionado como reloj.
- **TIMER:** El timer configurado para que la base de tiempos, el contador tenga una frecuencia de 1 kHz. Un canal Input Capture configurado en el TIM2 Channel 1 y un canal Output Compare sin correspondencia en ningún pin GPIO.
- **NVIC** Configura una IRQ para el TIM2 con dos fuentes de interrupción, una por canal.

Contenido del Directorio

- TIM/src/stm32l1xx_conf.h Fichero configuración de la librería
- TIM/src/stm32l1xx_it.c Manejador de interrupciones
- TIM/src/stm32l1xx_it.h Cabeceras del fichero de manejo de interrupciones
- TIM/src/main.c Aplicación principal. Contiene el main.
- TIM/src/system_stm32l1xx.c STM32L1xx fichero de sistema
- TIM/src/gpio_user.c Fichero de usuario con funciones de gpio
- TIM/src/gpio_user.h Fichero de cabeceras del fichero gpio de usuario
- TIM/src/lcd_user.c Fichero de usuario con funciones del display LCD
- TIM/src/lcd_user.h Fichero de cabeceras del fichero lcd de usuario
- TIM/src/systick_user.c Fichero de funciones del systick de ARM
- TIM/src/systick_user.h Fichero de cabeceras del fichero systick de usuario
- TIM/src/tim_user.c Fichero de funciones del periférico TIM con funciones de usuario.
- TIM/src/tim_user.h Fichero de cabeceras del fichero TIM de usuario

5.4.3. CÓDIGO DE LA APLICACIÓN

El código de la aplicación se encuentra bajo la carpeta TIM/src y está disponible en el CD que se adjunta con la memoria.

5.5. PWM

VISIÓN GENERAL

La siguiente aplicación muestra como configurar el timer visto en el anterior ejemplo, en modo PWM para hacer variar la intensidad del led LD3. Para la realización de la aplicación se han cumplido los siguientes hitos:

- Configurar la base de tiempos del timer a 12 Khz sin preescalado.
- Configurar el timer en modo PWM con salida HW en el led LD3.
- Implementar una función que tras pulsar el botón USER, cambie el ciclo de trabajo del PWM.
- Mostrar por pantalla LCD el ciclo de trabajo activo en ese momento.

Descripción Funcional: La aplicación arrancará mostrando un mensaje por el display LCD en modo scroll indicando “EJEMPLO PWM”. Tras el mensaje, se iluminará el LD3 que está conectado como salida del PWM. Cada vez que pulsemos el botón USER la intensidad del led se verá modificada. El display LCD mostrará el ciclo de trabajo configurado en ese momento, pudiendo ser 50 %, 25 %, 12 %, y 6 %.

5.5.1. ¿CÓMO UTILIZAR LA APLICACIÓN?

Seleccionar el workspace utilizado en este proyecto como el workspace de Atollic. Una vez seleccionado y con Atollic arrancado, abrir el proyecto PWM. Para arrancar el programa basta con compilar todos los ficheros y entrar en modo Debug.

Alternativamente, se puede crear un nuevo proyecto con Atollic utilizando el tutorial (ver tutorial Atollic) y sustituir la carpeta /src por la carpeta PWM/src.

5.5.2. DESCRIPCIÓN DEL SOFTWARE

Periféricos utilizados por la aplicación

Esta aplicación utiliza los siguientes periféricos con la configuración detallada a continuación:

- **GPIO:**

- PA0 configurado como entrada GPIO. - PB7 configurado como función alternativa TIM4 Channel 2, salida del PWM
- PA1, PA2, PA3, PA8, PA9, PA10, PA15, PB3, PB4, PB5, PB8, PB9, PB10, PB11, PB12, PB13, PB14, PB15, PC0, PC1, PC2, PC3, PC6, PC7, PC8, PC9, PC10 y PC11 están configurados como función alternativa, segmentos del LCD.

- **LCD:** Diferentes funciones disponibles del driver del controlador LCD son utilizadas para inicializar, borrar, mostrar strings y mostrar strings en modo scroll.
- **SYSTICK Timer:** El timer Systick es usado para generar un delays.
- **Reloj:** El oscilador MSI (Multi Speed Internal oscillator) es seleccionado como reloj.
- **TIMER:** El timer TIM4 configurada su base de tiempos para generar una onda de 12 kHz. Canal 2 configurado como Output PWM con un ciclo de trabajo variable.

Contenido del Directorio

- PWM/src/stm32l1xx_conf.h Fichero configuración de la librería
- PWM/src/stm32l1xx_it.c Manejador de interrupciones
- PWM/src/stm32l1xx_it.h Cabeceras del fichero de manejo de interrupciones
- PWM/src/main.c Aplicación principal. Contiene el main.
- PWM/src/system_stm32l1xx.c STM32L1xx fichero de sistema
- PWM/src/gpio_user.c. Fichero de usuario con funciones de gpio
- PWM/src/gpio_user.h Fichero de cabeceras del fichero gpio de usuario
- PWM/src/lcd_user.c Fichero de usuario con funciones del display LCD
- PWM/src/lcd_user.h Fichero de cabeceras del fichero lcd de usuario
- PWM/src/systick_user.c Fichero de funciones del systick de ARM
- PWM/src/systick_user.h Fichero de cabeceras del fichero systick de usuario

- PWM/src/pwm_user.c Fichero de funciones del periférico PWM con funciones de usuario.
- PWM/src/pwm_user.h Fichero de cabeceras del fichero PWM de usuario

5.5.3. CÓDIGO DE LA APLICACIÓN

El código de la aplicación se encuentra bajo la carpeta TIM/src y está disponible en el CD que se adjunta con la memoria.

5.6. DAC: ONDA TRIANGULAR

VISIÓN GENERAL

La siguiente aplicación muestra como configurar el DAC para generar una onda triangular de frecuencia y amplitud variable por su canal de salida. Para la realización de la aplicación se han cumplido los siguientes hitos:

- Configurar el timer para que el contador trabaje a una frecuencia de 180 kHz, con salida trigger external activa.
- Configurar el DAC en modo onda triangular, sin offset, y amplitud 200 mV.
- Implementar una función que con cada pulsación del botón USER cambie la amplitud de la señal triangular hasta en 4 ocasiones.
- Mostrar la frecuencia de la onda por pantalla LCD.

Descripción Funcional: La aplicación arrancará mostrando un mensaje por el display LCD en modo scroll indicando “EJEMPLO DAC”. Tras el mensaje, deberá aparecer en pantalla LCD la frecuencia de la primera onda. Midiendo con un osciloscopio en el pin analógico asociado al canal de salida del DAC, podremos ver la onda generada. Tras pulsar el botón USER, la amplitud y frecuencia de la onda representada en el osciloscopio cambiará, y en el display LCD se mostrará la nueva frecuencia. Esto sucederá para los 4 valores determinados, y una vez finalice el último, volverá a comenzar de nuevo.

5.6.1. ¿CÓMO UTILIZAR LA APLICACIÓN?

Seleccionar el workspace utilizado en este proyecto como el workspace de Atollic. Una vez seleccionado y con Atollic arrancado, abrir el proyecto ADC_TRI. Para arrancar el programa basta con compilar todos los ficheros y entrar en modo Debug.

Alternativamente, se puede crear un nuevo proyecto con Atollic utilizando el tutorial (ver tutorial Atollic) y sustituir la carpeta /src por la carpeta ADC_TRI/src.

5.6.2. DESCRIPCIÓN DEL SOFTWARE

Periféricos utilizados por la aplicación

Esta aplicación utiliza los siguientes periféricos con la configuración detallada a continuación:

- **GPIO:**
 - PA0 configurado como línea de interrupción externa EXTI0. - PA4 configurado en modo analógico como DAC_CH1. - PA1, PA2, PA3, PA8, PA9, PA10, PA15, PB3, PB4, PB5, PB8, PB9, PB10, PB11, PB12, PB13, PB14, PB15, PC0, PC1, PC2, PC3, PC6, PC7, PC8, PC9, PC10 y PC11 están configurados como función alternativa, segmentos del LCD.
- **LCD:** Diferentes funciones disponibles del driver del controlador LCD son utilizadas para inicializar, borrar, mostrar strings y mostrar strings en modo scroll.
- **SYSTICK Timer:** El timer Systick es usado para generar un delays.
- **Reloj:** El oscilador MSI (Multi Speed Internal oscillator) es seleccionado como reloj.
- **TIMER:** El timer TIM2 configurada su base de tiempos para generar un tick en la salida trigger output, a una frecuencia de 180 kHz.
- **DAC:** DAC configurado su canal DAC_CH1 para generar una onda con el vector de muestras que le pasa el DMA.
- **DMA:** DMA configurado para pasarle un vector de muestras al DAC en modo circular.

Contenido del Directorio

- DAC_TRI/src/stm32l1xx_conf.h Fichero configuración de la librería
- DAC_TRI/src/stm32l1xx_it.c Manejador de interrupciones
- DAC_TRI/src/stm32l1xx_it.h Cabeceras del fichero de manejo de interrupciones
- DAC_TRI/src/main.c Aplicación principal. Contiene el main.
- DAC_TRI/src/system_stm32l1xx.c STM32L1xx fichero de sistema
- DAC_TRI/src/gpio_user.c. Fichero de usuario con funciones de gpio

- DAC_TRI/src/gpio_user.h Fichero de cabeceras del fichero gpio de usuario
- DAC_TRI/src/lcd_user.c Fichero de usuario con funciones del display LCD
- DAC_TRI/src/lcd_user.h Fichero de cabeceras del fichero lcd de usuario
- DAC_TRI/src/systick_user.c Fichero de funciones del systick de ARM
- DAC_TRI/src/systick_user.h Fichero de cabeceras del fichero systick de usuario
- DAC_TRI/src/dac_user.c Fichero de funciones del periférico DAC con funciones de usuario.
- DAC_TRI/src/dac_user.h Fichero de cabeceras del fichero DAC de usuario

5.6.3. CÓDIGO DE LA APLICACIÓN

El código de la aplicación se encuentra bajo la carpeta DAC_TRI/src y está disponible en el CD que se adjunta con la memoria.

5.7. DAC: ONDA SENOIDAL Y SAWTOOTH

VISIÓN GENERAL

La siguiente aplicación muestra como configurar el DAC para generar una onda senoidal de frecuencia fija y una onda sawtooth o diente de sierra. Para la realización de la aplicación se han cumplido los siguientes hitos:

- Configurar el timer para que genere un tick en su salida trigger output a una frecuencia de 1 kHz.
- Configurar el DAC sin generación de onda triangular.
- Configurar el periférico DMA junto al DAC para pasarle un vector de muestras al DAC, con las muestras de la onda senoidal.
- Realizar el mismo paso anterior pero esta vez pasándole un vector de muestras de una señal sawtooth.
- Implementar una función que con cada pulsación del botón USER, commute entre la generación de una onda senoidal y una de diente de sierra.
- Mostrar el tipo de onda que está activa en el display LCD.

Generación ondas

Para generar ondas, basta con crear un vector de muestras. Para ello, con cualquier herramienta tipo "Matlab" podemos programar una función para generar un número determinado de muestras de un periodo de la función que deseemos. Una vez generadas estas muestras, debemos pasarselas al DAC mediante el periférico DMA. El número de muestras que tomemos de la señal determinará junto con la frecuencia del timer a la frecuencia de la onda que generemos.

Descripción Funcional: La aplicación arrancará mostrando un mensaje por el display LCD en modo scroll indicando “EJEMPLO DAC”. Tras el mensaje, aparecerá en pantalla LCD el tipo de onda que se está generando en el DAC. Midiendo con un osciloscopio en el pin analógico asociado al canal de salida del DAC, podremos ver la onda generada. Tras pulsar el botón

USER, conmutaremos entre onda senoidal y diente de sierra. Como ambas están generadas con el mismo tick de la señal trigger output del timer, tendrán frecuencias similares.

5.7.1. ¿CÓMO UTILIZAR LA APLICACIÓN?

Seleccionar el workspace utilizado en este proyecto como el workspace de Atollic. Una vez seleccionado y con Atollic arrancado, abrir el proyecto DAC. Para arrancar el programa basta con compilar todos los ficheros y entrar en modo Debug.

Alternativamente, se puede crear un nuevo proyecto con Atollic utilizando el tutorial (ver tutorial Atollic) y sustituir la carpeta /src por la carpeta DAC/src.

5.7.2. DESCRIPCIÓN DEL SOFTWARE

Periféricos utilizados por la aplicación

Esta aplicación utiliza los siguientes periféricos con la configuración detallada a continuación:

- **GPIO:**

- PA0 configurado como línea de interrupcion externa EXTI0. - PA4 configurado en modo analógico como DAC_CH1. - PA1, PA2, PA3, PA8, PA9, PA10, PA15, PB3, PB4, PB5, PB8, PB9, PB10, PB11, PB12, PB13, PB14, PB15, PC0, PC1, PC2, PC3, PC6, PC7, PC8, PC9, PC10 y PC11 están configurados como función alternativa, segmentos del LCD.

- **LCD:** Diferentes funciones disponibles del driver del controlador LCD son utilizadas para inicializar, borrar, mostrar strings y mostrar strings en modo scroll.

- **SYSTICK Timer:** El timer Systick es usado para generar un delays.

- **Reloj:** El oscilador MSI (Multi Speed Internal oscillator) es seleccionado como reloj.

- **TIMER:** El timer TIM2 configurada su base de tiempos para generar un tick en la salida trigger output, a una frecuencia de 180 kHz.

- **DAC:** DAC configurado su canal DAC_CH1 en modo onda triangular para generar una onda de amplitud y frecuencia variable.

Contenido del Directorio

- DAC/src/stm32l1xx_conf.h Fichero configuración de la librería
- DAC/src/stm32l1xx_it.c Manejador de interrupciones
- DAC/src/stm32l1xx_it.h Cabeceras del fichero de manejo de interrupciones
- DAC/src/main.c Aplicación principal. Contiene el main.
- DAC/src/system_stm32l1xx.c STM32L1xx fichero de sistema
- DAC/src/gpio_user.c Fichero de usuario con funciones de gpio
- DAC/src/gpio_user.h Fichero de cabeceras del fichero gpio de usuario
- DAC/src/lcd_user.c Fichero de usuario con funciones del display LCD
- DAC/src/lcd_user.h Fichero de cabeceras del fichero lcd de usuario
- DAC/src/systick_user.c Fichero de funciones del systick de ARM
- DAC/src/systick_user.h Fichero de cabeceras del fichero systick de usuario
- DAC/src/dac_user.c Fichero de funciones del periférico DAC con funciones de usuario.
- DAC/src/dac_user.h Fichero de cabeceras del fichero DAC de usuario

5.7.3. CÓDIGO DE LA APLICACIÓN

El código de la aplicación se encuentra bajo la carpeta DAC/src y está disponible en el CD que se adjunta con la memoria.

CONCLUSIONES

Tal y como se recogió al principio de esta memoria, el objetivo principal de este proyecto era el planteamiento de un nuevo entorno de trabajo constituido por una placa de desarrollo y un IDE que se ajustara al mismo, todo ello con un presupuesto ajustado.

Para llegar a la solución ha sido necesario un estudio de diversas placas y diferentes toolchains. En este estudio se ha trabajado con varias tecnologías desde plataformas de hardware libre como “Arduino” o “The Maple”, a placas de desarrollo económicas por parte de grandes empresas del sector como “LPCXpresso” o “STM32L-Discovery”. Al mismo tiempo se ha trabajado con diferentes arquitecturas, como microcontroladores AVR de Arduino a los potentes ARM Cortex M3. Este estudio ha servido para conocer diferentes arquitecturas y posibilidades a la hora de diseñar un nuevo proyecto de sistemas electrónicos embebidos.

En la búsqueda por el mejor toolchain, la mejor opción era lograr una configuración de Eclipse con compiladores GCC y herramientas de depuración GDB pero este último punto complicó el planteamiento de este entorno de trabajo. Por ello, se buscó y optó por Atollic TrueSTUDIO Lite que es un IDE gratuito, sin limitaciones, y 100 % compatible con la placa elegida, STM32L-DISCOVERY. Además, un motivo importante que apoyó la apuesta por el IDE de Atollic fue el hecho de estar basado en Eclipse y utilizar compiladores GCC, hecho que hará muy fácil la migración de Atollic a Eclipse en un futuro.

Una vez elegida la solución, ésta estaba formada por la placa STM32L-DISCOVERY y el IDE Atollic TrueSTUDIO. Con ello, se elaboró una guía de desarrollo para diferentes periféricos, ofreciendo la posibilidad al lector de este proyecto, ser capaz de desarrollar aplicaciones en un tiempo record. Esta guía se acompaña de diferentes ejemplos de código para dichos periféricos lo que aporta mucha más base para el estudio de esta tecnología.

Gracias a este nuevo entorno de trabajo, se ha conseguido solucionar el problema inicial, en el que teníamos herramientas de desarrollo de pago, y placas de desarrollo de precio bastante más elevado. Con este proyecto, solamente es necesario una placa como la que hemos utilizado, cuyo precio en el mercado no supera los 15€, además de ofrecer mucha más ventajas como incorporar un LCD, leds, botón de usuario, sensor deslizante, tener un tamaño pequeño que facilite su transporte, y sencilla conexión al PC mediante USB sin necesidad de programadores externos. A

esto hay que sumarle la utilización de un IDE completamente gratuito sin ningún tipo de coste y limitación de uso temporal.

También es importante destacar los trabajos futuros que ofrece esta placa, como se analizarán a continuación, y es que este proyecto abre la puerta a muchos otros proyectos, no sólo en seguir buscando un nuevo entorno para esta configuración, sino en utilizar la placa como herramienta de desarrollo para nuevas aplicaciones de investigación.

TRABAJO FUTURO

En esta sección se plantearán las nuevas líneas de investigación que se abren con este proyecto, recogiendo de esta forma las ideas que durante la elaboración y desarrollo del mismo han surgido.

1. El primer punto de investigación sería la búsqueda de un servidor debugger compatible con ST-LINK/V2 y Eclipse. En este proyecto se ha conseguido configurar Eclipse con herramientas de compilación GCC para que pueda compilar un proyecto para la placa STM32L-DISCOVERY, pero por problemas del protocolo ST-LINK/V2 no se ha podido depurar con herramientas GDB. Por lo tanto, se abre una vía de estudio en la que se consiga plantear un entorno de desarrollo totalmente GNU. Esto implicaría el salto de Atollic TrueSTUDIO a Eclipse, labor que sería sencilla puesto que Atollic está basado en Eclipse.
2. En este proyecto no hemos trabajado con todos los periféricos y son muchos los periféricos y modos de operación que hay en la placa de desarrollo. Una nueva línea de investigación sería elaborar ficheros de configuración para estos periféricos, como son DAC (Digital-to-Analog Converter), USART (Universal Synchronous/Asynchronous Receiver Transmitter), Comparador, I²C (Inter-Integrated Circuit), etc.
3. Con a este proyecto, cualquier estudiante es capaz de estudiar el microcontrolador, y con los ejemplos de configuración, aprender a desarrollar para STM32L utilizando un Cortex M3. Esto abre la puerta a muchos otros proyectos de fin de carrera, en el que el microcontrolador elegido sea un ARM Cortex M3 y este proyecto sirva de guía y base para el aprendizaje de este procesador.
4. Diseño de una tarjeta expansora que permita la conexión de la placa STM32L-Discovery a la placa de periféricos E/S disponible en los laboratorios del departamento de electrónica de la universidad Carlos III. Esto ampliaría las posibilidades del microcontrolador, pudiendo aumentar la gama de periféricos E/S.

PRESUPUESTO DEL PROYECTO

En la siguiente sección se muestra justificado el coste global de la realización de este proyecto.

8.1. Fases del Proyecto

En la Tabla 8.2 se muestran las fases del proyecto y el tiempo estimado para cada una de ellas. Para la realización del proyecto se ha necesitado disponer de un ingeniero. Se estima una fase inicial de **planificación** de 20 horas. En esta fase se estudia las diferentes posibilidades y se estudió el mercado en busca de placas sustitutivas a la que disponíamos.

Tras esta primera fase, comienza una fase de **evaluación y documentación** de las placas que habíamos seleccionado. Se debía estudiar la capacidad de cada una, las ventajas e inconvenientes para finalmente poder realizar una elección. Se estima una duración de 80 horas.

Una vez se escogió la placa, comenzó una etapa de **estudio** de la arquitectura de la placa, documentación e instalación del software necesario para poder desarrollar. Se estima una duración de 80 horas.

La fase de **diseño y desarrollo** de las aplicaciones es una de las más extensas junto con la de documentación. En esta fase se incluye el diseño inicial de las aplicaciones, su desarrollo en código y su posterior evaluación. Se estima una duración de 160 horas.

La última etapa es la de **documentación** del proyecto, etapa en la que se reflexiona sobre el trabajo realizado, se establecen conclusiones y se elabora una memoria que refleje el trabajo realizado. Se estima una duración de 160 horas.

La duración total es de 500 horas que se extendieron en un periodo de 4 meses. A continuación se detalla en la siguiente tabla:

8.2. Coste del Personal Empleado

Para la realización del proyecto se ha necesitado cierto material y personal. Como refleja la Tabla 8.4 se ha necesitado un ingeniero técnico que ha llevado a cabo la implementación, evaluación, y documentación del proyecto. Suponiendo el sueldo medio de un Ingeniero Técnico de Telecomunicaciones de 25€/hora el coste de personal es 12.500€.

Fase	Tiempo (horas)
Planificación	20
Evaluación y Documentación	80
Estudio e Instalación	80
Desarrollo	160
Documentación	160

Tabla 8.2: *Fases del proyecto junto con el tiempo aproximado empleado.*

Personal	Coste(€)
Ingeniero Técnico Telecomunicaciones	12.500

Tabla 8.4: *Presupuesto del proyecto: coste del personal empleado en el proyecto.*

8.3. Coste del material empleado

Además del personal se ha necesitado ciertos gastos materiales, que quedan reflejados en la Tabla 8.6.

Material	Coste(€)
PC portatil	499
Placa STM32L-Discovery	14,80
Placa LPCXPresso 1769	33,69
Placa Arduino UNO	22
Material Oficina	100
Conexión Internet 4 meses	160

Tabla 8.6: *Coste del material empleado en el proyecto.*

8.4. Coste del Software empleado

En cuanto al software utilizado durante el proyecto, queda reflejado en la siguiente Tabla 8.8:

Software	Coste(€)
S.O. Windows 7	200
Eclipse IDE	Gratis
Atollic TrueSTUDIO Lite	Gratis
Texmaker	Gratis
LPCXPresso IDE	Gratis
Arduino IDE	Gratis
Compiladores GCC	Gratis

Tabla 8.8: *Coste del software empleado en el proyecto*

8.5. Presupuesto Total

En la Tabla 8.10 queda reflejado el resumen del coste total del proyecto.

Concepto	Importe(€)
Personal	12.500
Material	829,49
Software	200
IVA (18 %)	2.399,30
TOTAL	15.728,80

Tabla 8.10: *Coste total.*

APÉNDICES

APÉNDICE A

ATOLLIC TRUESTUDIO

A.1. INSTALACIÓN

Este tutorial muestra como instalar el software ATOLLIC TRUE STUDIO Lite (su version gratuita).

Entramos en su página web:

<http://atollic.com/index.php/download/downloadstm32>

y descargamos la version “Atollic TrueSTUDIO / STM32 Lite v2.10”

Nota: Atollic está basado en Eclipse y Eclipse está escrito en Java por lo que necesitaremos de la última versión de la máquina virtual de Java. La podemos descargar desde su web:

<http://www.java.com>

Pinchamos sobre “Free Download” (Figura A.1).

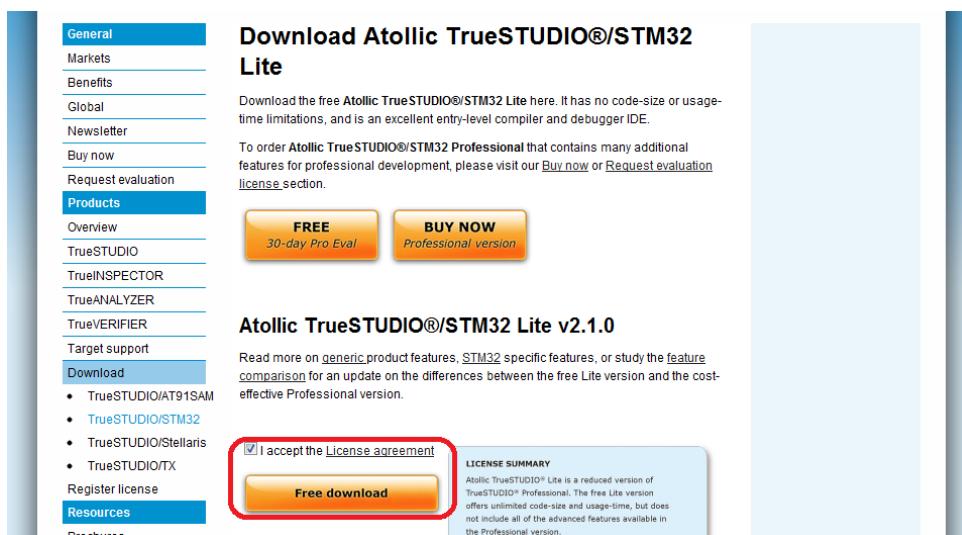


Figura A.1: Descarga Atollic TrueSTUDIO Lite

Una vez descargado el ejecutable, procedemos a su instalación. Marcamos la casilla de verificación para asegurarnos de que Atollic busque actualizaciones (Figura A.2).

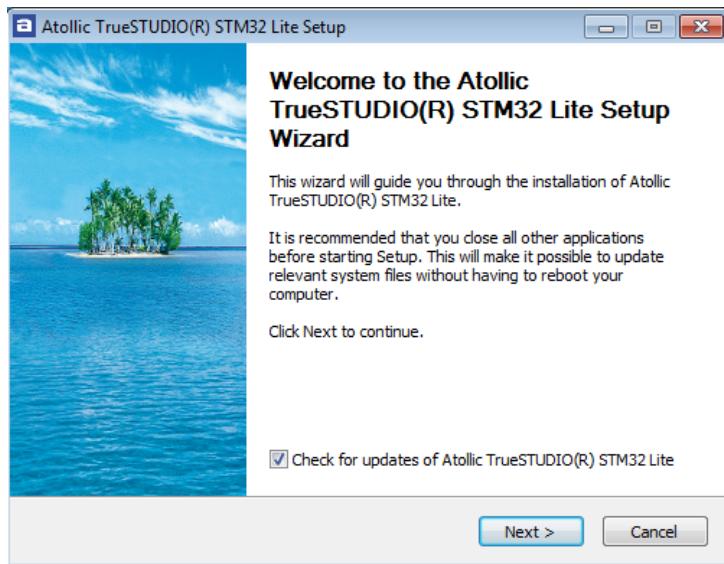


Figura A.2: *Instalación Atollic TrueSTUDIO Lite*

En la siguiente ventana se mostrará una clave que identifica a nuestro PC. Al mismo tiempo, se abrirá una ventana en el navegador en la que se mostrará una página para el registro del producto. (Figura A.3) (Figura A.4)

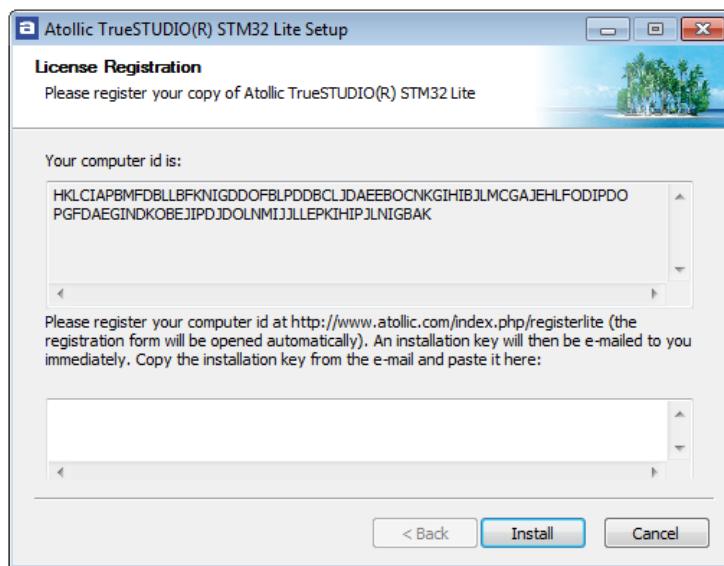


Figura A.3: *Generación Clave PC*

Debemos copiar la clave de nuestro PC y pegarla en el registro para así obtener una licencia para el producto. Podemos llenar el formulario de registro con los datos del correo electrónico de la universidad (Figura A.5).

Register Lite version

Error: You need to fill in all fields!

Please enter your contact information and Computer id (generated by the installation program) in the form below. Then press the "Register" button below to register your free product license.

An Installation key is then generated and e-mailed to you automatically. You must enter the Installation key in the installation program to complete the product installation.

First name	Benito
Last name	Pérez Galdós
E-mail	10007777@alumnos.uc3m.es
Company	UC3M
Address	Avda Universidad 30
Postal code	28911
City	Leganés
State	Madrid
Phone	916 249 153
Country	Spain
Customer type	Academic use
Computer id (from installer)	HKLCIAPPBMFDLBFKNIGDDOFBLPDBCLJDAEEBOCNKGHIHBJLMC CAJEHLFODIPDO PGFDAEGINDKOBEPJDJDOLNMJJLLEPKIHIPJLNIGBAK

Register

Figura A.4: Registro Web Atollic

Register Lite version

Thank you for registering your product!

An e-mail with your Installation key has been sent to: david87guetta@hotmail.com

If you have not received an e-mail with the installation key within some minutes after registering, try again or contact support@atollic.com. It is not uncommon that the auto-generated e-mails are deleted by spam filters.

Figura A.5: Registro Completo

Una vez relleno el formulario se nos enviará un email con nuestro código de activación, que deberemos copiarlo y pegarlo en la ventana de instalación, justo en el recuadro blanco destinado para ello (Figura A.6).

Tras esto ya habremos instalado el producto.

A continuación el instalador nos preguntará si deseamos instalar el driver de ST-Link. Seleccionaremos la casilla de verificación y comenzara la instalación del driver.(Figura A.7)

Tras instalar el driver, volveremos al instalador de Atollic y finalizaremos la instalación.

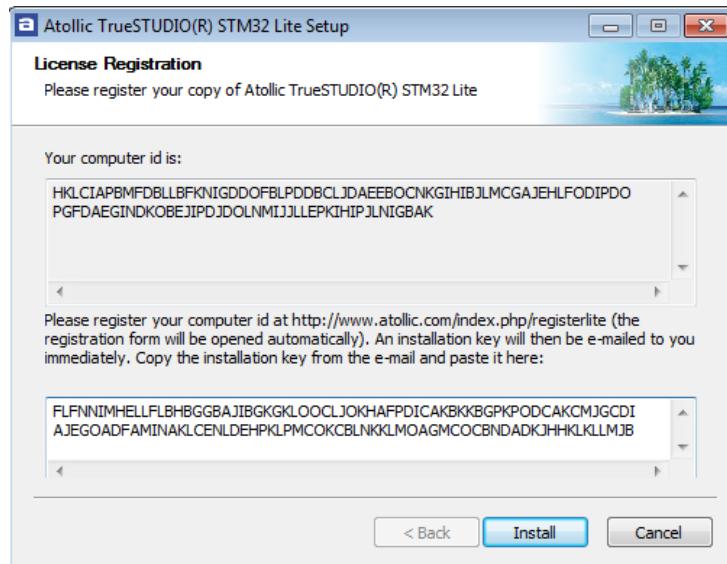


Figura A.6: Clave Atollic TrueStudio Lite

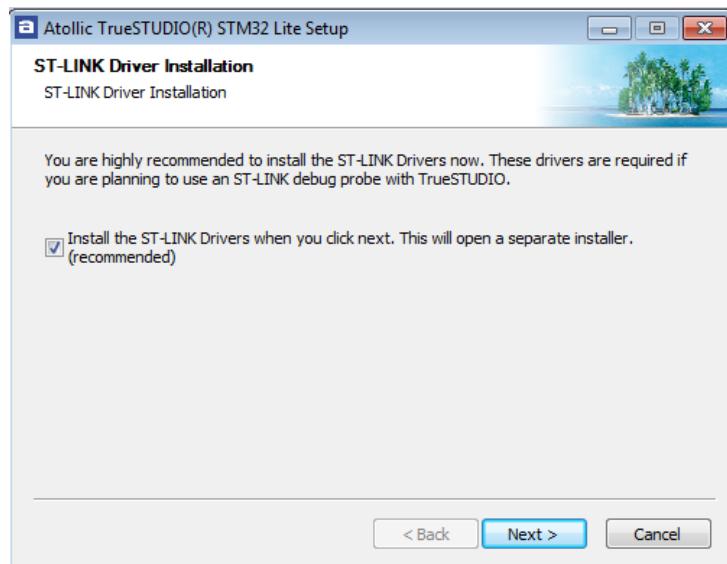


Figura A.7: Instalación ST-Link

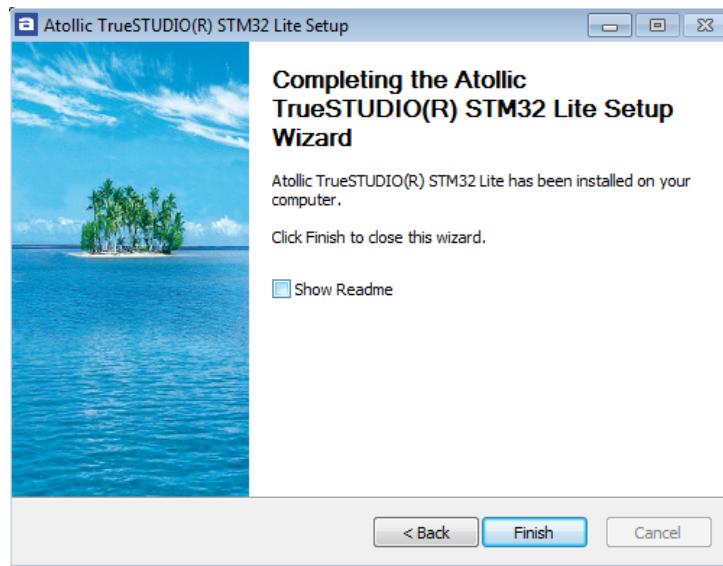


Figura A.8: *Instalación Completa*

(Figura A.8)

Una vez instalado Atollic y el driver de ST-Link, es el momento de conectar la tarjeta a nuestro PC a través del cable USB. Para ello será necesario un cable USB Standard-A a Mini-B (Figura A.9).



Figura A.9: *Cable USB*

Nada más conectarlo, Windows comenzará a instalar el software controlador automáticamente. Sabremos que el dispositivo está correctamente conectado cuando los leds LD1 y LD2 permanezcan encendidos de manera continua.

Windows nos informará de que el dispositivo “STMicroelectronics STLink dongle” ha sido instalado correctamente (Figura A.10).

También es necesario descargar el firmware actualizado del STLink y la herramienta para

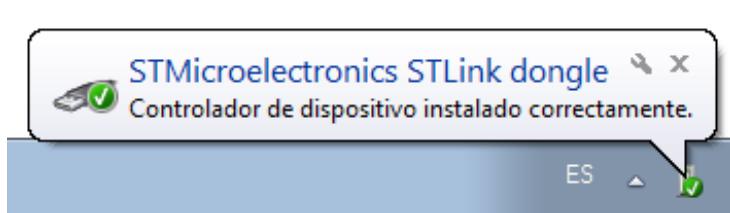


Figura A.10: *Instalacion ST-Link Dongle*

actualizar. Ambos están disponibles en la web de STMicroelectronics a través de este enlace:

<http://www.st.com/internet/evalboard/product/251168.jsp>

Nota: Es importante no confundir el STLink con el STLink V2 que es el que lleva la placa STM32L-DISCOVERY.

Una vez en la web, entramos en la pestaña Design Support y ahí dispondremos del Software necesario (Figura A.11).



Figura A.11: *Descarga ST-Link V2*

Los ficheros a descargar son:

- STM32 ST-LINK utility
- ST-LINK/V2 USB driver for Windows 7, Vista and XP

El ST-Link V2 driver ya le tenemos instalado pues en la instalación de Atollic se nos preguntó si deseábamos instalar dicho driver. Sin embargo, es conveniente descargarlo y guardararlo junto con el resto de documentos y aplicaciones de Atollic y ST por si tuviéramos problemas de conectividad con la placa, poder reinstalarlo directamente a través de este ejecutable. No obstante, si más tarde comprobamos que tenemos conectividad, no será necesaria su reinstalación.

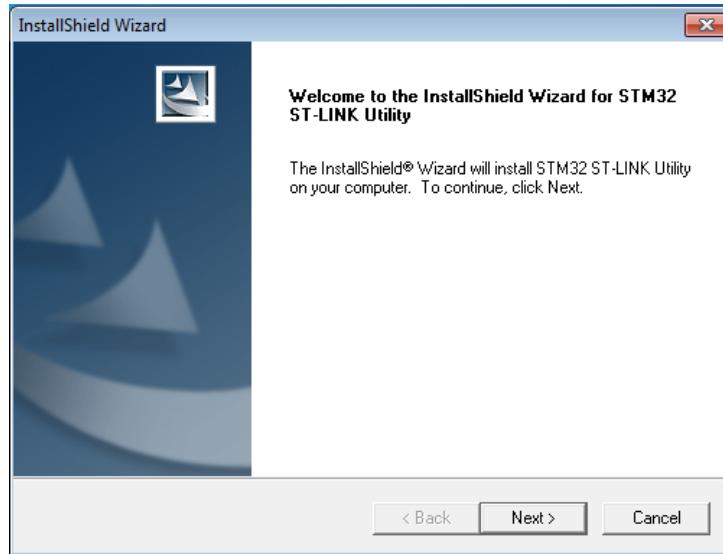


Figura A.12: *Instalación ST-Link Utility*

El segundo paso sera instalar el ST-LINK Utility (Figura A.12).

El propio proceso instalara la versión más actual del producto. Actualmente la 1.04.0000

Una vez instalado, un acceso directo se creará en nuestro escritorio llamado “STM32 ST-LINK Utility”. La funcionalidad de esta aplicación es la de descargar el código en la memoria de la tarjeta pero éste no es nuestro objetivo. La funcionalidad es de nuestro interés reside en la pestaña “ST-LINK” ->“Firmware Update”. Con esta herramienta podremos actualizar el firmware. (Figura A.13) (Figura A.14).

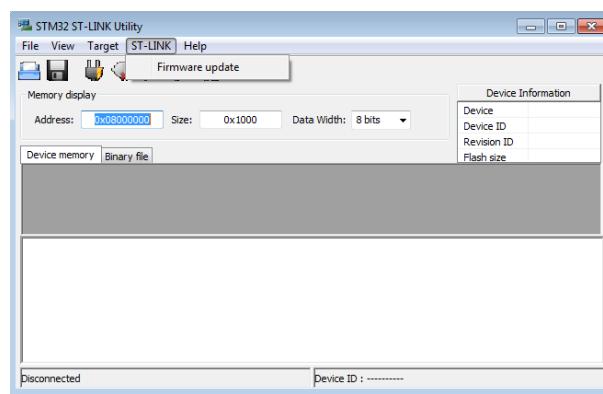
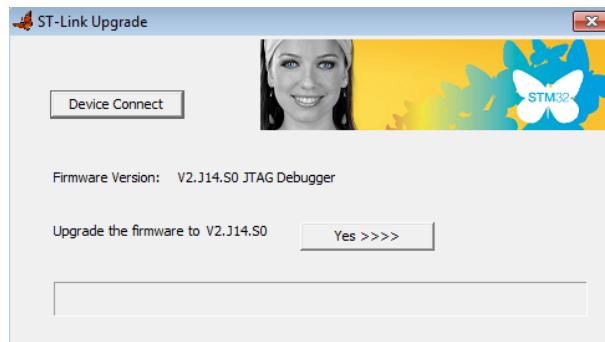
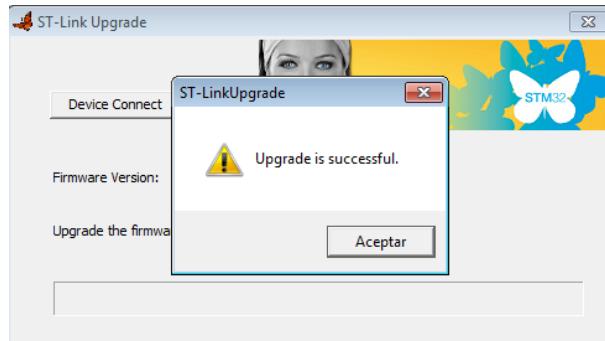


Figura A.13: *Ventana ST-Link Utility*

Seleccionamos “Device Connect”. Si la conectividad es correcta, nos informará de la version del firmware que dispone nuestra tarjeta y de la versión más reciente disponible para actualizar.

Figura A.14: *ST-Link Upgrade*

Tras pulsar sobre “Yes”, el dispositivo se actualizará (Figura A.15).

Figura A.15: *Actualización Completa*

En este momento ya tenemos nuestro dispositivo y ordenador completamente configurados y listos para poder trabajar. En la placa viene por defecto una aplicación de muestra cargada. Esta muestra el funcionamiento de algunas de las funciones disponibles, como el LCD, encendido de LEDS, botones táctiles y sensor lineal, medidor de corriente, etc. Podemos interactuar con los diferentes periféricos y ver el funcionamiento de la aplicación de muestra.

A.2. PRIMEROS PASOS

Una vez instalado Atollic y configurado la placa STM32L-Discovery es el momento de arrancar Atollic TrueStudio. Una vez abierto, nos preguntará por donde deseamos crear el Workspace. El Workspace es nuestra carpeta donde se guardarán todos los proyectos que realicemos con Atollic para una determinada tarjeta, o un determinado proyecto. Podemos crear este Workspace en nuestra carpeta de usuario (Figura A.16).

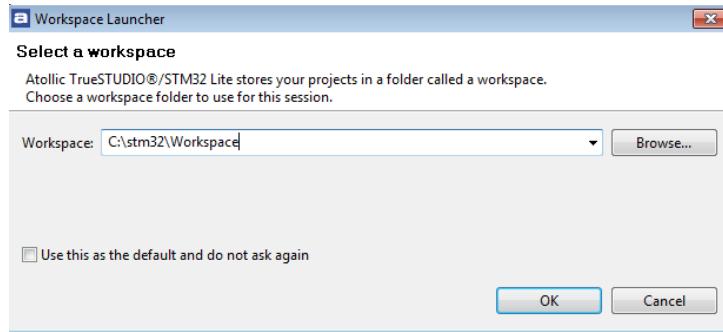


Figura A.16: *Seleccion Workspace*

Tras instalarlo, nos aparecerá una ventana de publicidad, anunciándonos que actualicemos a la versión PRO de Atollic. Basta con cerrar la ventana pulsando en “Close Windows”. Esta ventana es la misma que aparecerá cuando entremos en modo debug. La primera vez que arranquemos Atollic nos mostrará una página de bienvenida. En ella podemos acceder a información sobre el producto. Podemos saltar esta introducción pulsando “Start using TrueStudio”

Después de saltar esa ventana, el entorno de desarrollo aparecerá. La ventana de Atollic pasará a **Perspectiva C/C++**. Atollic al igual que Eclipse tiene varias perspectivas. Esta primera es la que utilizamos para trabajar con el código (Figura A.17).

A.2.1. Creando un nuevo proyecto

Para crear un nuevo proyecto es necesario seguir los siguientes pasos (Figura A.18):

- Seleccionar File, New, C Project en la barra de herramientas. (captura 21)
- La página configuración **C Project** debemos dejarla como se muestra a continuación. Deberemos introducir un nombre de proyecto. Éste deberá ser identificativo al tipo de aplicación que vayamos a escribir. Seleccionar STM32 C Project y pulsar next.
- En la ventana **Build Settings** seleccionar en Evaluation Board nuestra tarjeta, STM32L-Discovery. El resto dejarlo como se muestra en la imagen. Pulsar Next.
- En la ventana **Misc Settings** seleccionar ST-LINK como la interfaz debug. La versión Atollic Lite sólo soporta esta interfaz mientras que la profesional admite un mayor rango de interfaces. Pulsar Next.
- La ventana **Select Configurations** deberemos dejarla tal cuál se presenta. Pulsar Finish.

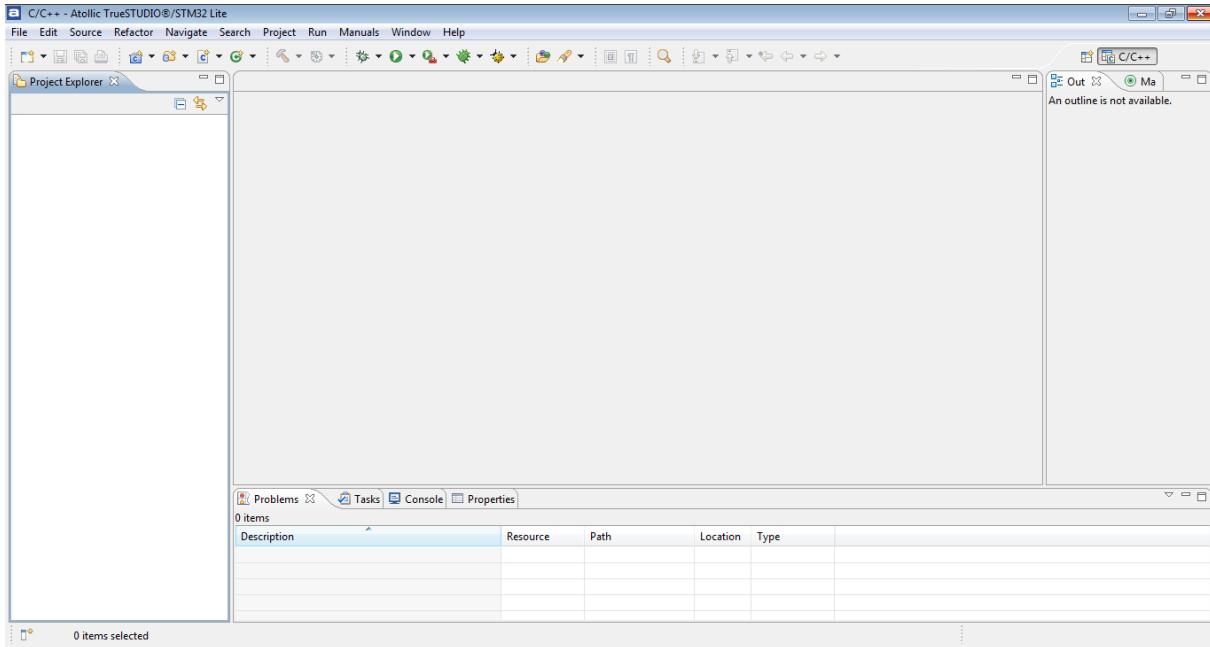


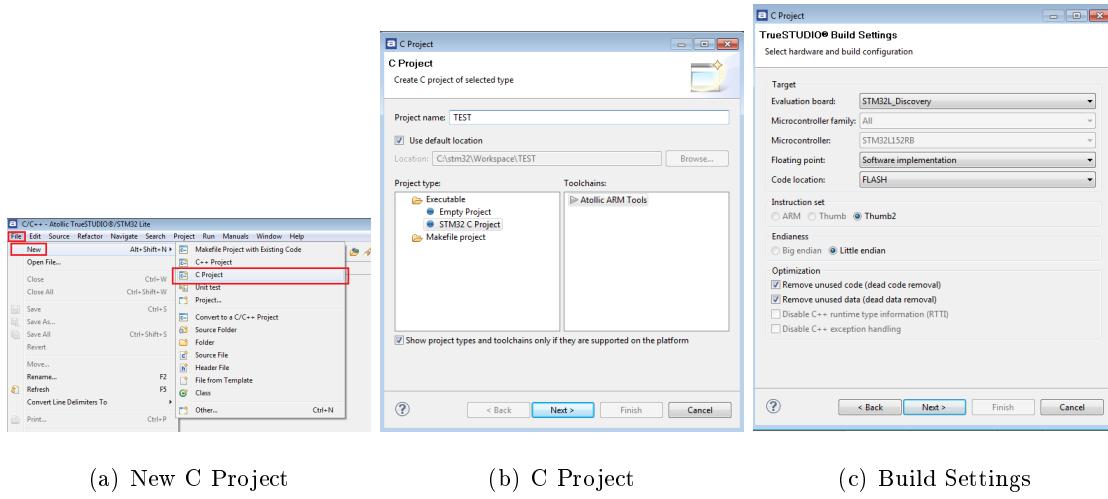
Figura A.17: Perspectiva C/C++

De esta forma habremos creado un nuevo proyecto C para STM32L-Discovery utilizando el Wizard que nos ofrece Atollic. También es posible crear un proyecto en blanco para configurarlo nosotros pero es de nuestro interés que el propio programa configure sus parámetros para nuestra placa.

El propio Atollic crea ejemplos de prueba para facilitar el desarrollo de nuevas aplicaciones.

A continuación vamos a ver cuáles son las diferentes partes de este programa.

A la izquierda tenemos el explorador de Proyecto. En él se encuentra una carpeta llamada TEST que es nuestro proyecto. La desplegamos y vemos que contiene varias carpetas en su interior. En el interior de la carpeta src se encuentra el código de nuestro programa: main.c. Haciendo doble click en él aparecerá el editor de texto en la ventana central (Figura A.19). El resto de carpetas en el explorador son las mismas que vemos en el directorio del proyecto en el workspace. La estructura de ficheros y el sentido de los mismos se explica en la SECCIÓN XX de esta memoria. Además de estas carpetas, también aparecen los despegables **Binaries** y **Includes**.

Figura A.18: *Creando un nuevo proyecto*

En la ventana inferior, tenemos varias pestañas: Problems, Tasks, Console y Properties. Explicamos a continuación las que son de nuestro interés (Figura A.20).

- **Problems:** en ella aparecen los errores y warnings sufridos tras compilación. En el editor de texto se indicará el error al lado del número de línea para facilitar su identificación. En el explorador de proyectos se mostrará también qué ficheros es el que genera el error.
- **Console:** es la salida de la consola. La función make que utiliza el compilador es por línea de comandos y muestra su salida en esta pestaña. Los errores que se muestren aquí son los mismos que se muestran en la pestaña Problems.

A.2.2. Compilando el proyecto

Atollic TrueSTUDIO por defecto compila el proyecto automáticamente en cuanto cualquier fichero del proyecto es actualizado. Esto se muestra dentro de la pestaña **Project** en la barra de herramientas (Figura A.21).

Para compilar el proyecto podemos utilizar el acceso directo debajo de la barra de herramientas con forma de martillo **Build**. Con esta herramienta lo que le decimos al programa es que compile exclusivamente aquellos ficheros que necesiten ser compilados porque tengan alguna modificación.

Si en lugar de este botón pulsamos el botón de **Build All** lo que haremos será recompilar

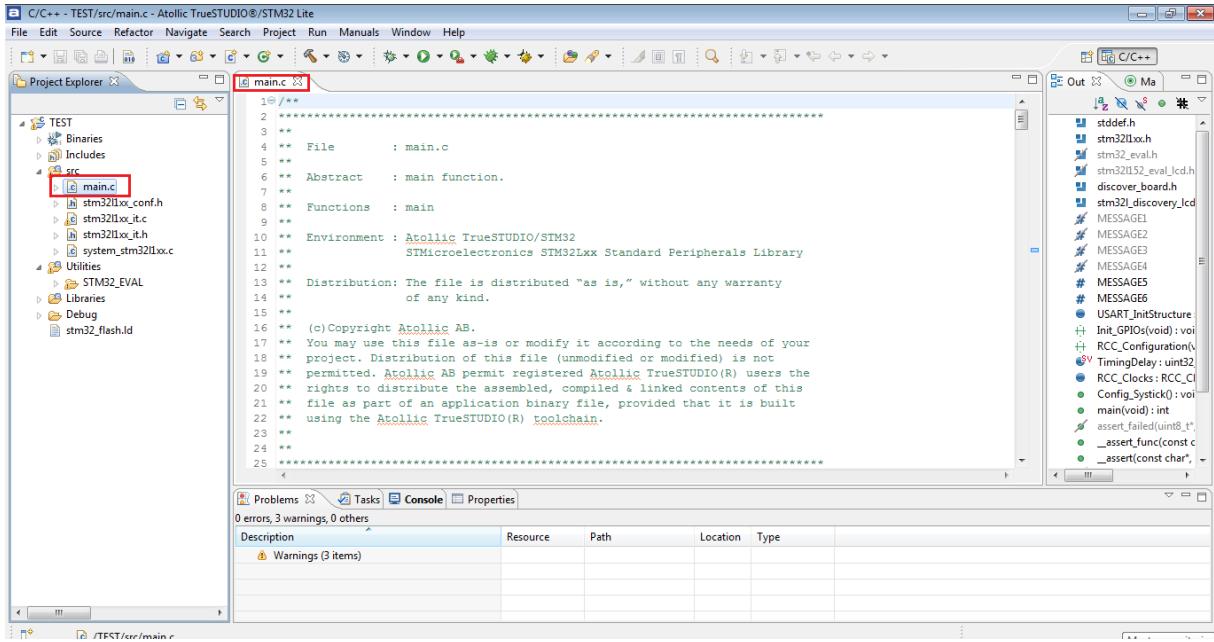


Figura A.19: *Editor de Texto*

todo el proyecto entero. Para ver la salida del compilador podemos abrir la pestaña **Console** en la parte inferior, y ver la salida en formato texto (Figura A.22).

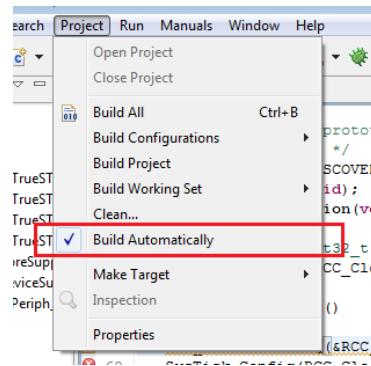
Una vez tenemos compilado el proyecto, podemos entrar al modo Debug. A partir de este momento es necesario tener la placa conectada a nuestro PC a través del cable USB. Para arrancar el modo Debug debemos pulsar sobre el acceso directo debajo de la barra de herramientas, con forma de insecto (Figura A.23). La primera vez que entremos en modo Debug nos aparecerá una pantalla de configuración para crear un perfil de Debug. Al igual que otras muchas facetas de Atollic para STM32, en el momento que seleccionamos nuestra tarjeta STM32L-Discovery en el asistente de creacion de proyecto al principio, no es necesario configurar ningun parámetro pues Atollic ya lo hizo por nosotros. Solamente tenemos que pulsar OK y lo tendremos configurado (Figura A.24). Tras ello, nos aparecerá como ya habíamos comentado la ventana para actualizar a la version PRO. Esta nos aparecerá cada vez que entremos a modo Debug pero basta con cerrarla.

Importante: Es posible que la primera vez que entremos en modo Debug con nuestra placa en el PC salte el firewall de Windows pidiendo autorización para el ST-Link. Debemos autorizar su acceso. Sólo será necesario realizar esta operación la primera vez que usemos ST-Link en

The screenshot shows the Atollic TrueSTUDIO IDE interface. On the left, the project tree displays a 'src' folder containing 'main.c'. The code editor window shows a portion of 'main.c' with several lines highlighted in blue. A red box highlights the line 'SysTick_Config(RCC_Clocks.HCLK_Frequency / 1000);'. Below the code editor is a 'Problems' view showing one error: 'expected ';' before 'SysTick_Config''. A red arrow points to this error message.

```

58 #ifndef USE_SIM32L_DISCOVERY
59 void Init_GPIOs (void);
60 void RCC_Configuration(void);
61
62 static volatile uint32_t TimingDelay;
63 RCC_ClocksTypeDef RCC_Clocks;
64
65 void Config_Systick()
{
66     RCC_GetClocksFreq(&RCC_Clocks);
67     SysTick_Config(RCC_Clocks.HCLK_Frequency / 1000);
68 }
69 /**
70 ****
71 ** Abstract: main program
72 /**
73 ****
74 /**
75 ** Abstract: main program
76 /**
77 ****
78 */
79 int main(void)
    
```

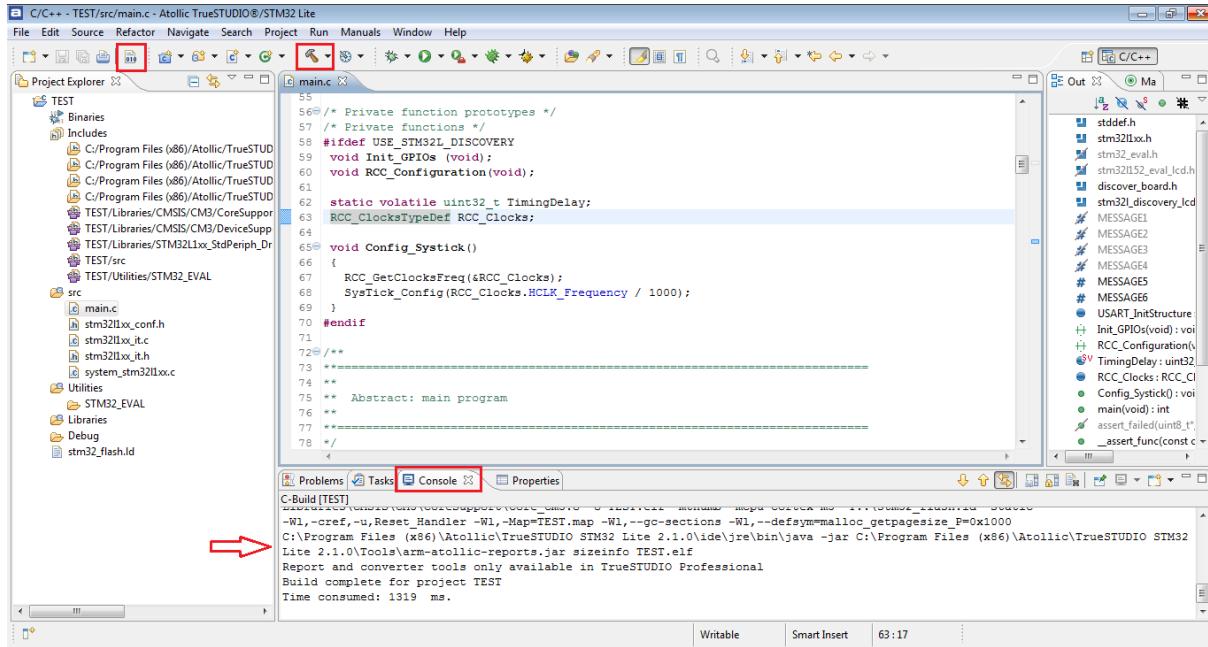
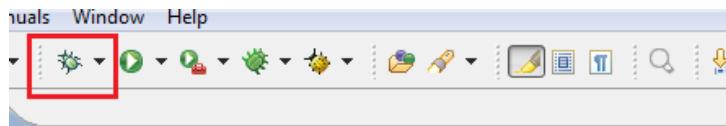
Figura A.20: *Errores de Compilación*Figura A.21: *Build Automatically*

nuestro PC.

A.2.3. Modo Debug

La ventana de Atollic cambiará. Esto es debido a que hemos cambiado la perspectiva, hemos pasado de Perspectiva de C/C++ a **Perspectiva Debug** (Figura A.25). En esta perspectiva aparecen nuevas ventanas. En el menú **Run** de la barra de herramientas aparecerán activos varios botones necesarios para correr el código. Sin embargo los mas importantes aparecen en accesos directos en la ventana izquierda **Debug** (Figura A.26).

- La ventana de la izquierda muestra la Ventana Debug. En ella la barra de control tiene los

Figura A.22: *Compilando el proyecto*Figura A.23: *Run Debug*

principales controles: correr el código, pausa, stop, reinicio, step into, step over, etc.

- A la derecha tenemos una ventana con varias pestañas. Las que nos interesan son **variables** y **breakpoints**. La primera de ellas muestra el valor de las variables en el momento en el que paramos el código. La segunda muestra los breakpoints que tenemos habilitados. En la versión Lite de Atollic sólo está permitido un breakpoint. Como es lógico, para poder mostrar el código, la aplicación debe estar pausada.
- Más abajo aparece el editor de texto. Éste sirve para analizar el código mientras la aplicación corre o insertar y eliminar breakpoints. Cualquier cambio que realicemos en el código en esta perspectiva no tendrá efecto ya el Debug necesita del fichero objeto que genera el compilador.

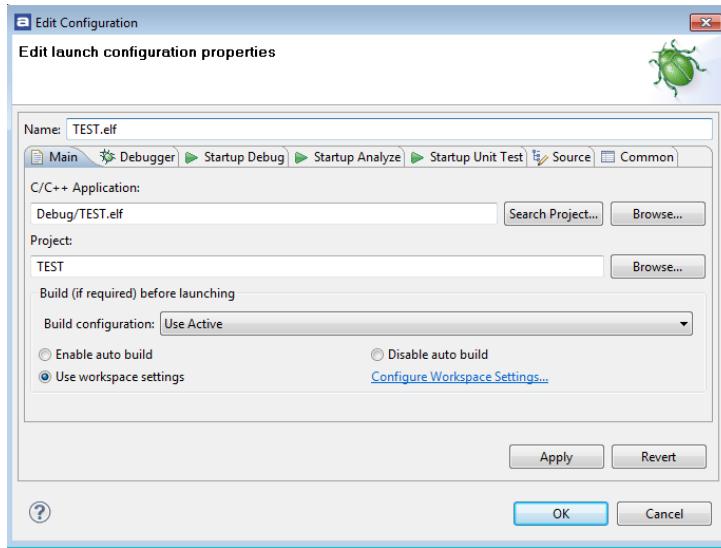


Figura A.24: *Debug Configuration*

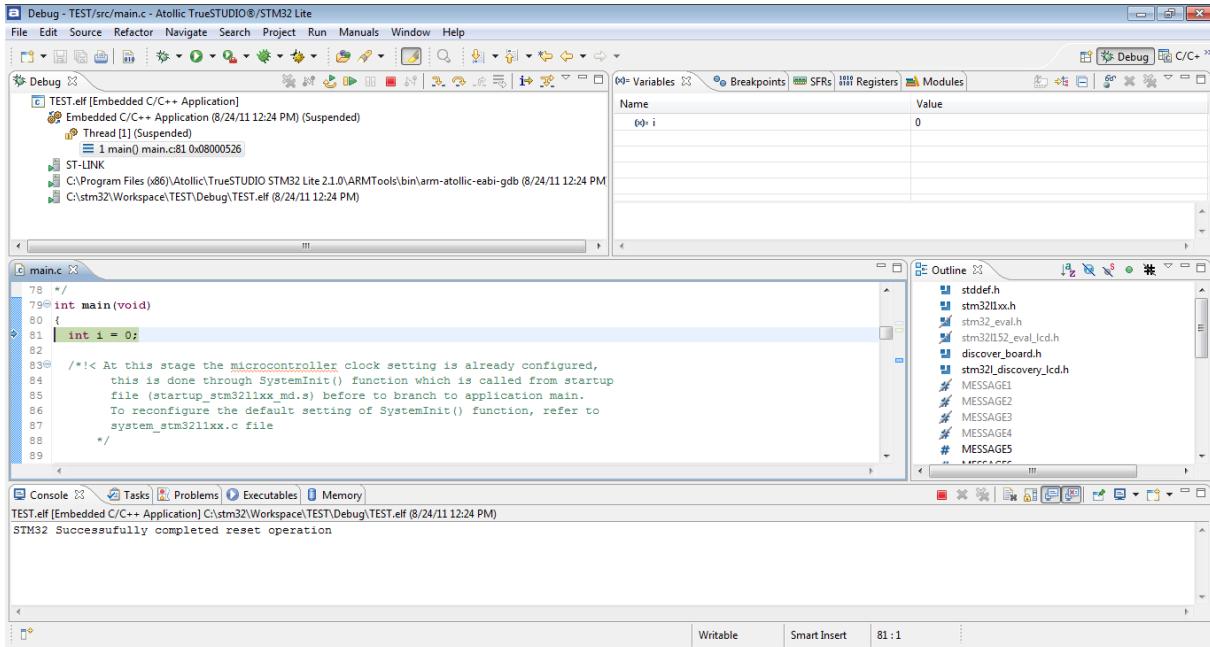
- La última ventana inferior muestra la salida por consola, tasks, problems, y dos pestañas nuevas: **Executables** y **Memory**.

Por defecto, Atollic inserta un breakpoint en el main, por lo que tenemos que arrancar la aplicación. Para arrancar, pulsamos el botón **Resume** o la tecla **F8**. En dicho momento, la aplicación arrancará en nuestra placa y comenzará a realizar la función para la cuál ha sido programada. En el ejemplo que viene en Atollic y estamos corriendo es el parpadeo de dos leds.

Una vez arrancado, podemos parar el código de dos formas: insertando un breakpoint en la línea de código que queramos parar o deteniéndolo manualmente pulsando el botón **pause**. Los breakpoint son muy útiles cuando se están intentando comprender el funcionamiento de algunas partes del código o detectar por qué nuestro código falla en determinado momento.

Para insertar un breakpoint basta con situarnos con el cursor encima de la línea en la que deseemos el breakpoint, pulsar el botón derecho y seleccionar **Toggle Breakpoint**. También podemos hacer doble click en el numero de línea y de esta manera insertaremos el breakpoint de manera más rápida. Para eliminar el breakpoint proceder igual que para insertarlo, seleccionando **Toggle** en el menú desplegable o con doble click (Figura A.27).

Nota: Si alguna vez tenemos problema con el ST-Link de no conectividad, ir a la perspectiva de Debug y eliminar todos los breakpoint que haya abiertos. También podemos hacer doble click

Figura A.25: *Debug Perspective*Figura A.26: *Controles Modo Debug*

en el número de línea y de esta manera insertaremos el breakpoint de manera más rápida.

Cuando tenemos el código pausado, podemos avanzar lentamente para observar el progreso de la aplicación mas exhaustivamente. Para ello tenemos las herramientas **Step into** y **Step Over**. Con la primera de ellas el debugger ejecuta una línea de código, y si hay una llamada a una función, el debugger irá dentro de esa función. Sin embargo, con la segunda, el debugger ejecutará la línea de código y si hay una llamada a una función, el debugger realizará el cometido de esa función sin detenerse. También existe la herramienta **Step Return**. Su funcionamiento es el siguiente: si estamos dentro de una función y queremos que el debugger salga de esa función y vaya a la línea de código que llamó a tal función, usaremos dicha herramienta. En la ventana Debug podemos ir viendo que en qué profundidad de código nos encontramos (Figura A.28).

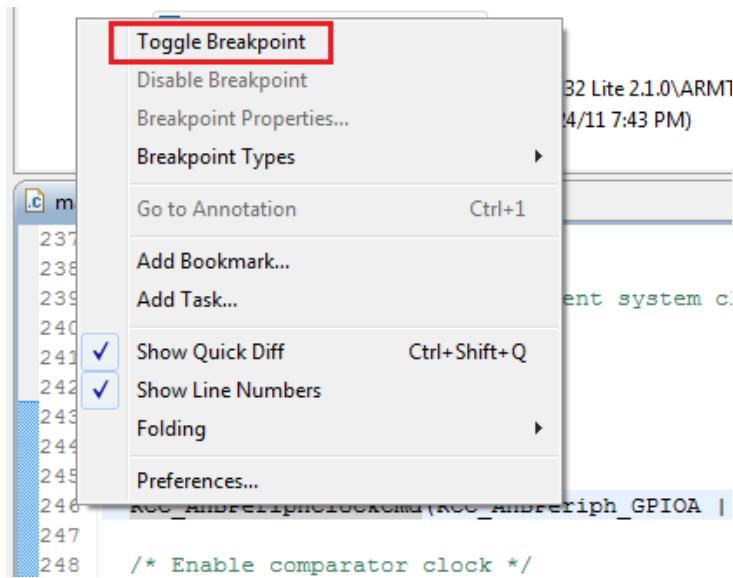


Figura A.27: Insertar Breakpoint

En muchas ocasiones es interesante introducir una variable para observar su variación a medida que se ejecuta el código, por ejemplo un contador. Para ello, seleccionamos la variable en la ventana de edición del código y con la palabra seleccionado, abrimos el menú despegable y seleccionamos **Add Watch Expression**. Ahora en la pestaña Expressions podremos ver su valor siempre y cuando la aplicación se encuentre detenida (Figura A.29).

Para salir del modo Debug basta con pulsar sobre el botón **Terminate** (Figura A.30). Si simplemente realizamos un cambio de perspectiva, el modo Debug seguirá activo y la siguiente vez que deseemos arrancarlo no podremos puesto que ya hay una sesión activa. Tras pulsar el botón Terminate, volveremos a la perspectiva C/C++ automáticamente.

A.3. Gestionando Proyectos

Una de las grandes facetas de Atollic es su gestor de proyectos. Esta característica permite agrupar varios proyectos en el mismo Workspace, todos bajo el mismo entorno de trabajo.

Para gestionar varios proyectos vamos a crear uno nuevo, siguiendo los pasos que explicamos a comienzo de este capítulo, y lo llamaremos “TEST2”.

Si observamos en el explorador de proyectos, aparecerán dos proyectos, TEST y TEST2 (Figura A.31). Podemos ver los ficheros de cada uno de ellos en el editor de texto y trabajar

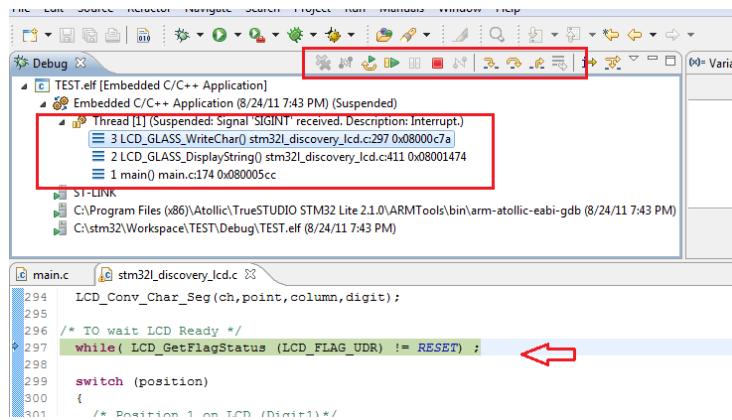


Figura A.28: *Step Into* y *Step Over*

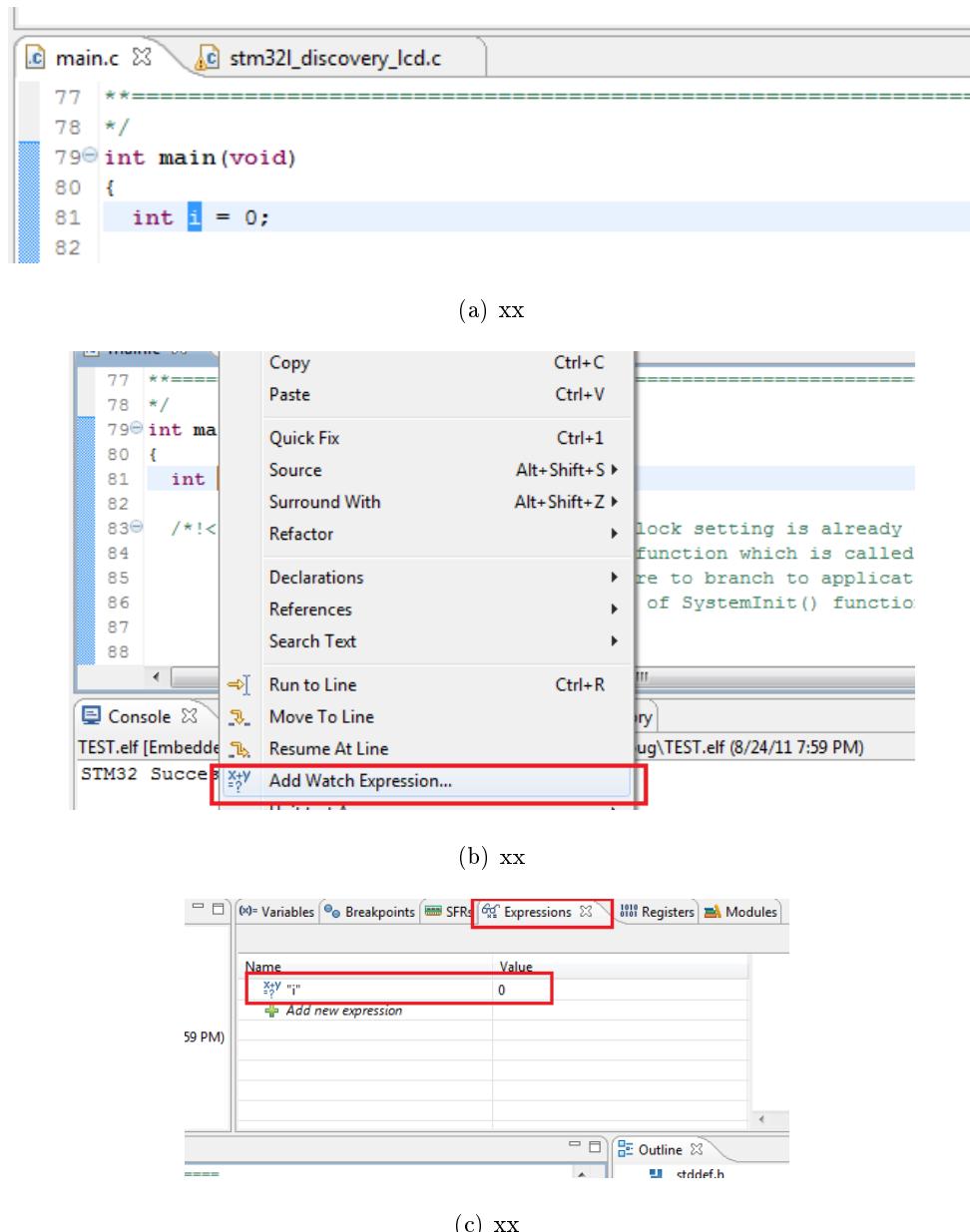
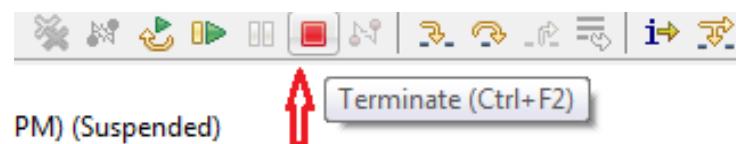
como si tuviésemos uno sólo.

Tenemos que tener cuidado de que a la hora de compilar, tengamos en el explorador de proyectos seleccionado el proyecto que deseamos y no uno diferente. Otro error sería querer arrancar el debugger para un proyecto y tener seleccionado otro. Siempre hay que tener seleccionado el proyecto que deseemos antes de depurar y/o compilar.

Para evitar este tipo de errores están las herramientas de **Close Project** y **Open Project**. Estas herramientas permiten cerrar proyectos deshabilitándolos de forma temporal, de esta manera estará inactivo y no podremos trabajar con él, pero seguirá estando su información ahí. Cuando deseemos volver a trabajar con él, bastara con abrirlo. Para cerrar un proyecto seleccionamos el que deseamos cerrar y con el menú desplegable seleccionamos Close Project. También se puede realizar la misma operación desde el menú Project en la barra de herramientas (Figura A.32).

Observar cómo una vez cerrado el proyecto, no se puede acceder a sus archivos y el símbolo de su carpeta pasa de abierta a cerrada y en color azul. Para volver abrirla basta seleccionar el proyecto y seleccionar Open Project o desde la barra de herramientas en el menú Project (Figura A.33).

Es importante destacar que cada proyecto requiere de un perfil de debug, por lo que la primera vez que entremos en perspectiva debug con un nuevo proyecto, la ventana de configuración nos aparecerá para crear un nuevo perfil.

Figura A.29: *Add Watch Expression*Figura A.30: *Finalizar Modo Debug*

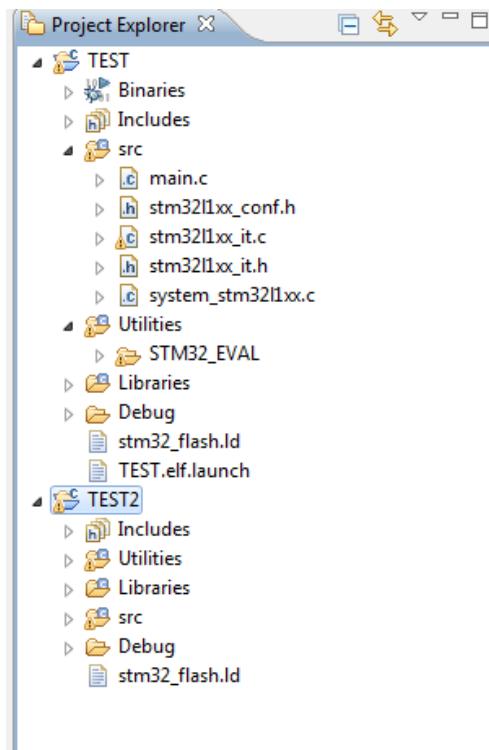


Figura A.31: Varios Proyectos Abiertos

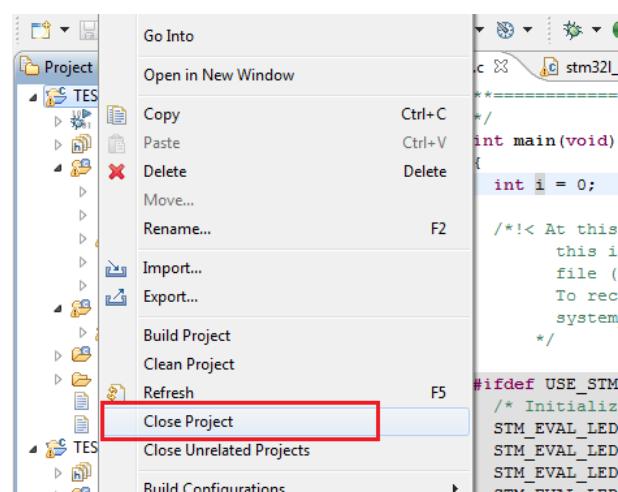


Figura A.32: Close Project

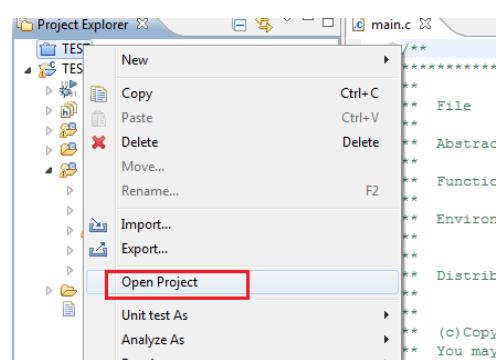


Figura A.33: *Open Project*

APÉNDICE B

ECLIPSE

En este anexo vamos a mostrar como configurar Eclipse para desarrollar aplicaciones para la placa STM32L-DISCOVERY. A la hora de leer este anexo se realizan las dos siguientes suposiciones:

- En el PC están instalado los drivers del ST-Link/V2 y actualizado el firmware del mismo mediante su aplicación, tal y como explica el anexo de instalación y configuración de Atollic TrueSTUDIO.
- El anexo sobre Atollic TrueSTUDIO ha sido leído y por lo tanto se conoce el funcionamiento del programa.

Vamos a utilizar "Eclipse IDE for C/C++ Developers", en concreto el paquete "Helios SR2" que está disponible en su página web para descargar:

<http://www.eclipse.org/downloads/packages/release/helios/sr2>



Figura B.1: Descarga Eclipse for C/C++ Developers

También debemos descargar el paquete "Sourcery G++ Lite for ARM EABI". La versión actual es 2011.03-42 lanzada en Mayo de 2011.

<http://www.codesourcery.com/sgpp/lite/arm/portal/subscription?@template=lite>

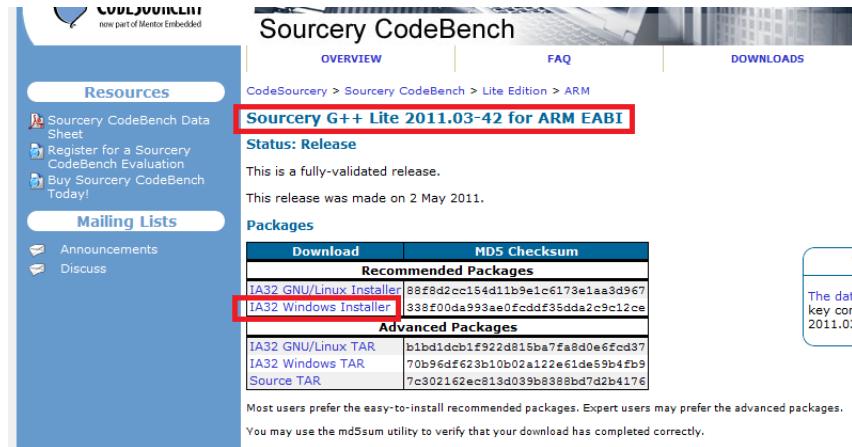


Figura B.2: *Sourcery ++ Lite for ARM EABI*

Una vez instalado procedemos a instalar Sourcery ++ Lite en primer lugar. Para ello abrimos el ejecutable. Debemos realizar la instalación típica. Se recomienda instalar el programa en la misma carpeta dónde vayamos a instalar posteriormente Eclipse. Para ello, a modo de ejemplo, hemos creado una carpeta llamado ST32L-DISCOVERY en C:\|. En dicha carpeta, crearemos una carpeta donde instalar Sourcery, otra donde descomprimiremos Eclipse, y otra donde tendremos el Workspace de Eclipse. La estructura de directorios quedará de la siguiente forma:

```
.
| -- STM32L_DISCOVERY
|   | -- Sourcery_Lite
|   | -- Eclipse
|   ' -- Workspace
```

Seguiremos el asistente de instalación asegurándonos de modificar el PATH para el usuario actual al menos. Cuando finalice el asistente ya dispondremos de Sourcery ++ Lite instalado.

Eclipse no tiene un instalador. El archivo que nos descargamos es un fichero comprimido que deberemos descomprimir en el directorio anteriormente creado a tal efecto. El directorio Workspace también creado será donde ubiquemos los proyectos que realicemos con Eclipse.

La primera vez que abramos Eclipse nos aparecerá una ventana para seleccionar el directorio del Workspace. Seleccionamos el directorio que creamos anteriormente e indicamos que utilice dicho directorio por defecto (Figura B.3).

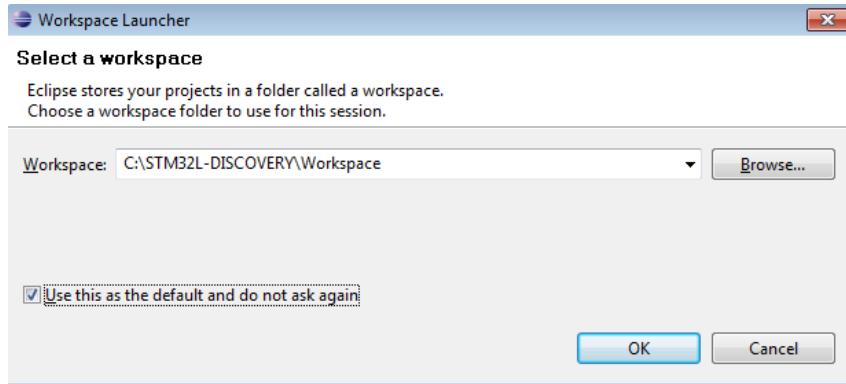


Figura B.3: Selección Workspace

Después de seleccionar el Workspace, nos aparecerá una pantalla inicial. Para saltarla deberemos pinchar en el ícono de la derecha que indica “Go to Workbench”. Esta pantalla solo aparecerá la primera vez que ejecutemos Eclipse (Figura B.4).



Figura B.4: Pantalla Bienvenida Eclipse

La siguiente (Figura B.5) muestra la perspectiva C/C++. Como se puede observar, no tiene diferencias con la pantalla de Atollic TrueSTUDIO. Tanto esta perspectiva, como la perspectiva Debug tiene la misma presentación. Todas las funciones y el modo de operación de Atollic está en Eclipse puesto que Atollic está basado en Eclipse.

Ahora es necesario instalar unas extensiones para Eclipse. Para ello tenemos que ir a “Help ->Install New Software...”. Una vez dentro pinchamos en “Available Software Sites”

En ella, en el cuadro de texto introducimos “CDT” y la fuente que aparece, debemos seleccionarla y pinchar en OK para añadirla como fuente de recursos software (Figura B.6). Debemos

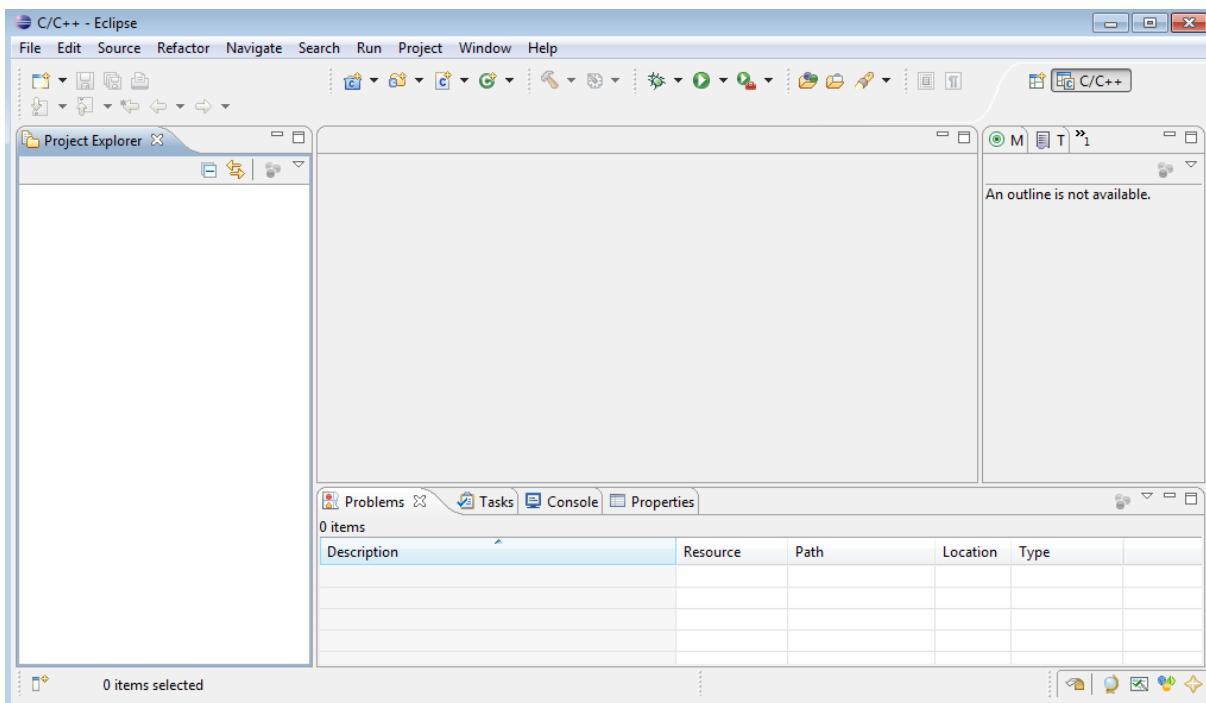


Figura B.5: Pantalla Principal Eclipse

hacer los mismo con otra fuente pero ésta no aparece por defecto, debemos añadirla. Para ello pinchamos en el botón ADD, y en la dirección añadimos <http://gnuarmeclipse.sourceforge.net/updates> (Figura B.7). Pulsamos OK y de nuevo OK para volver a la pantalla inicial de instalación de nuevo software.

De nuevo en la pantalla inicial para instalar nuevas fuentes, en “Work with” debemos indicar “All Available Sites”. La lista que se desplegará debajo puede que tarde en salir. Una vez aparezca, seleccionar las siguientes (Figura B.8) y (Figura B.9).

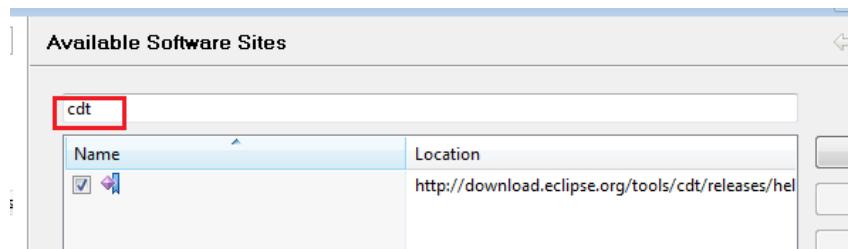


Figura B.6: Añadir fuente CDT

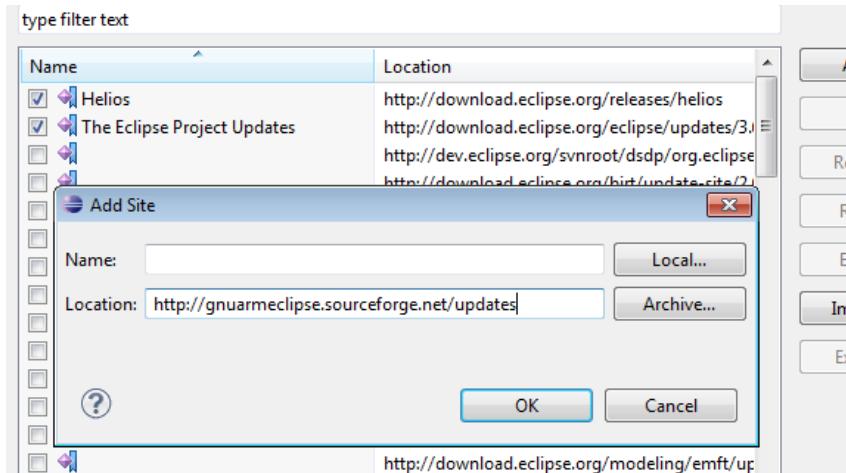


Figura B.7: *Añadir fuente GNU ARM Eclipse*

▷	<input type="checkbox"/> Business Intelligence, Reporting and Charting	
◀	<input checked="" type="checkbox"/> CDT GNU Cross Development Tools	
	<input checked="" type="checkbox"/> GNU ARM C/C++ Development Support	0.5.3.201010141144
◀	<input type="checkbox"/> CDT Main Features	
	<input type="checkbox"/> C/C++ Development Tools	7.0.2.201102110609
	<input checked="" type="checkbox"/> C/C++ Development Tools SDK	7.0.2.201102110609

Figura B.8: *Instalación de nuevas fuentes*

◀	<input type="checkbox"/> CDT Optional Features	
	<input type="checkbox"/> C/C++ Code Analysis Framework (Codan)	1.0.0.201102110609
	<input type="checkbox"/> C/C++ Debugger Services Framework (DSF) Examples	2.1.0.201102110609
	<input type="checkbox"/> C/C++ Development Platform	7.0.2.201102110609
	<input type="checkbox"/> C/C++ DSF GDB Debugger Integration	2.1.0.201102110609
	<input type="checkbox"/> C/C++ GCC Cross Compiler Support	1.0.0.201102110609
	<input checked="" type="checkbox"/> C/C++ GDB Hardware Debugging	7.0.0.201102110609

Figura B.9: *Instalación de nuevas fuentes*

Una vez seleccionadas, pulsamos next hasta que el proceso de instalación finalice. Eclipse nos pedirá reiniciar. Aceptamos y esperamos a que vuelva a arrancar (lo realizará automáticamente).

Nota: Si durante la instalación nos aparece una ventana emergente informándonos de que estamos instalando software sin firmar, diremos que aun así deseamos seguir con la instalación.

A continuación, vamos a aprender a configurar nuestro proyecto para la placa STM32L-DISCOVERY. Para ello, crearemos un nuevo proyecto C con Eclipse como aprendimos en el anterior anexo. Una vez tengamos este proyecto creado, lo utilizaremos como plantilla, ya que contiene todos los ficheros y librerías necesarios para que el código funcione. Además, al crear el proyecto, Atollic genera un main.c con un ejemplo sencillo que utilizaremos para comprobar que Eclipse compilar correctamente después de su configuración.

En Eclipse, creamos un nuevo proyecto mediante pulsaremos en “File ->New ->C Project”. Una vez en la ventana, dejamos la configuración tal y como muestra la (Figura B.10) y pulsamos Finish.

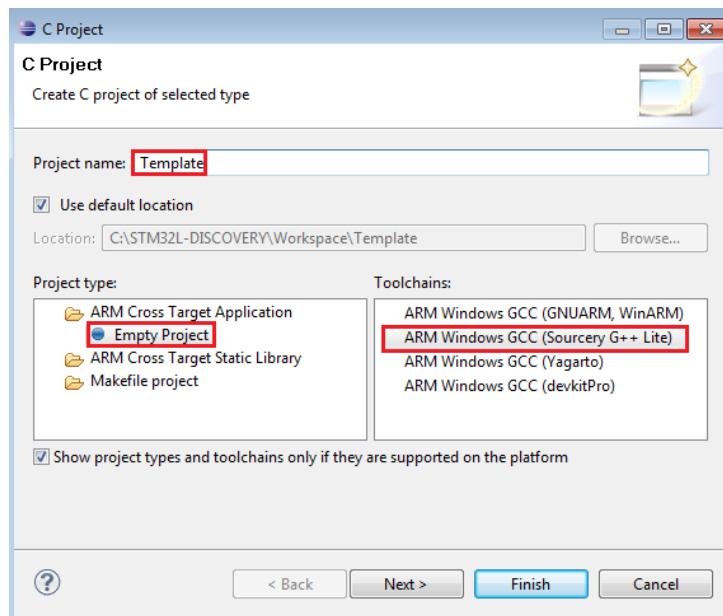


Figura B.10: Nuevo Proyecto C

En el explorador de proyectos veremos que se ha creado un nuevo proyecto llamado "Template" (Figura B.11). Ahora deberemos copiar todas las carpetas generadas con Atollic dentro de la carpeta Template. Para ello, utilizamos el explorador de Windows, en lugar de realizarlo dentro de Eclipse, pues es mas rápido. Deberemos copiar todos los archivos excepto .cproject

y .project. Una vez copiados, pulsamos F5 en Eclipse para actualizar, y todas las carpetas y ficheros aparecerán dentro de Template. Ahora Eclipse deberá quedar como a continuación:

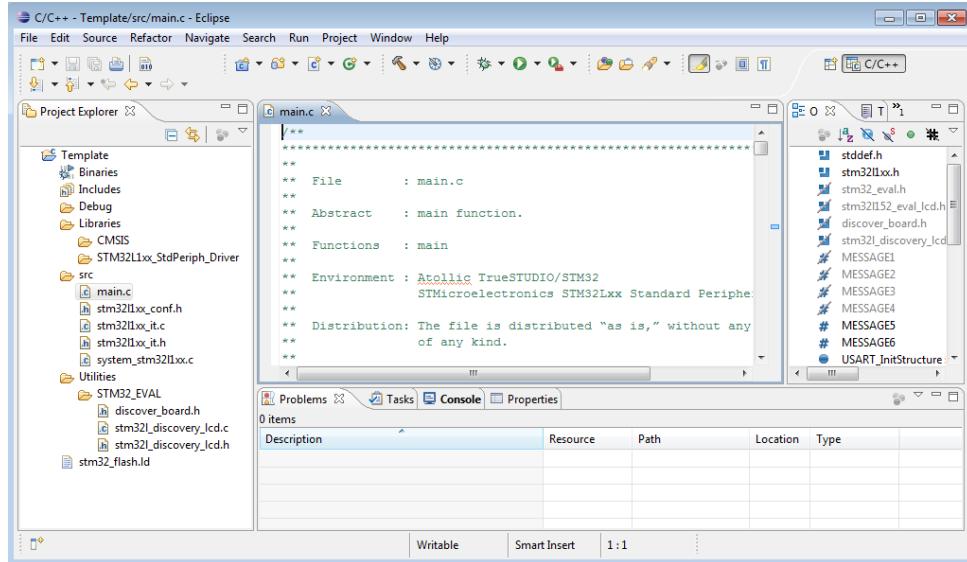


Figura B.11: *Proyecto Template*

Ahora debemos configurar el proyecto. Para ello pulsamos en “Project ->Properties”. Nuestro punto de interés es “C/C++ Build” y dentro de esta pestaña, la sección “Settings”. La siguiente batería de imágenes muestra como debemos dejar la configuración. Debemos configurar solamente las ventanas que se muestren a continuación, y seleccionar las mismas opciones.

Importante: Cada vez que hagamos un cambio en una de estas ventanas, hay que pulsar en “Apply” (Figura B.12)

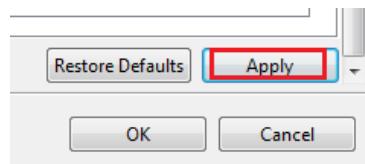
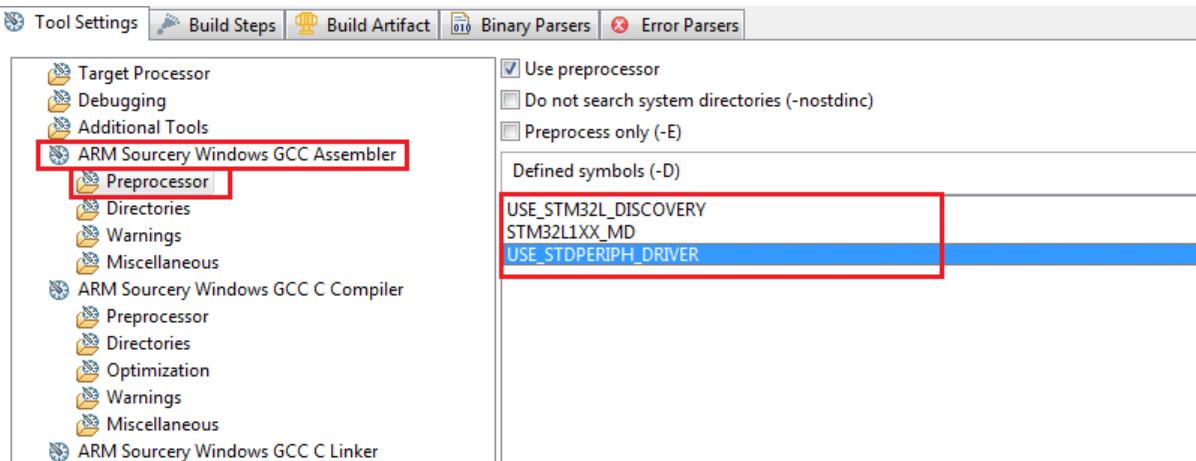
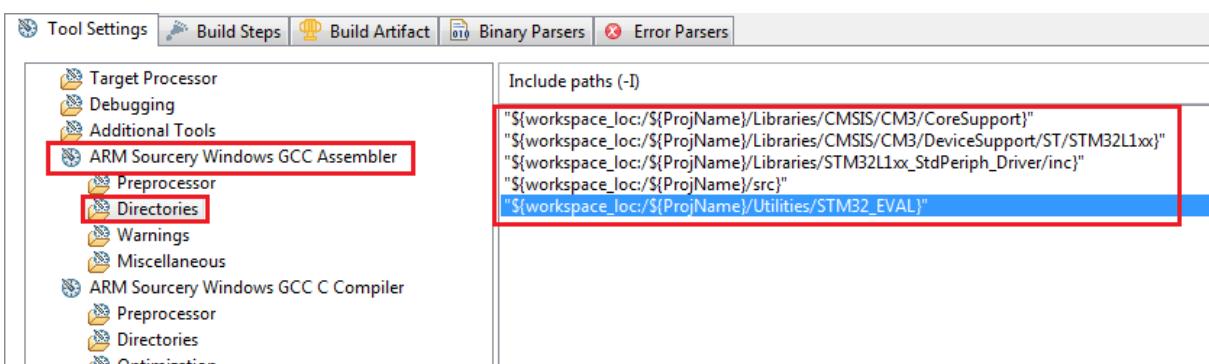
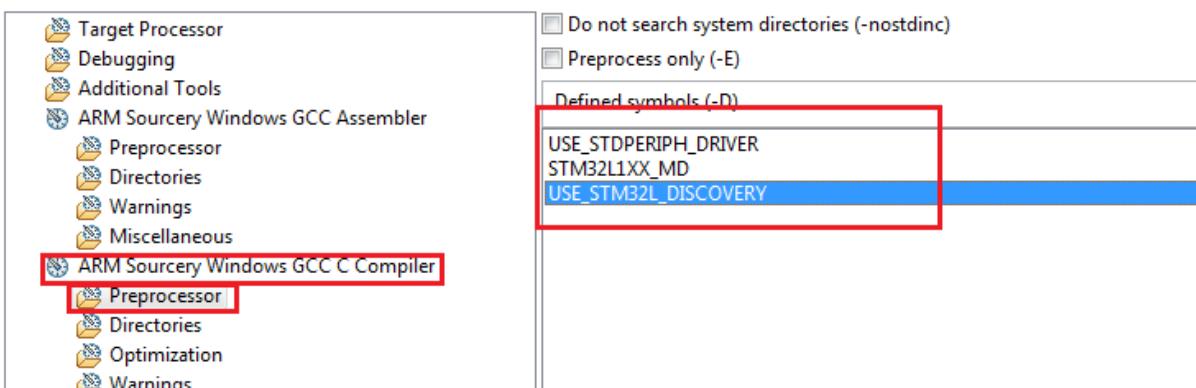
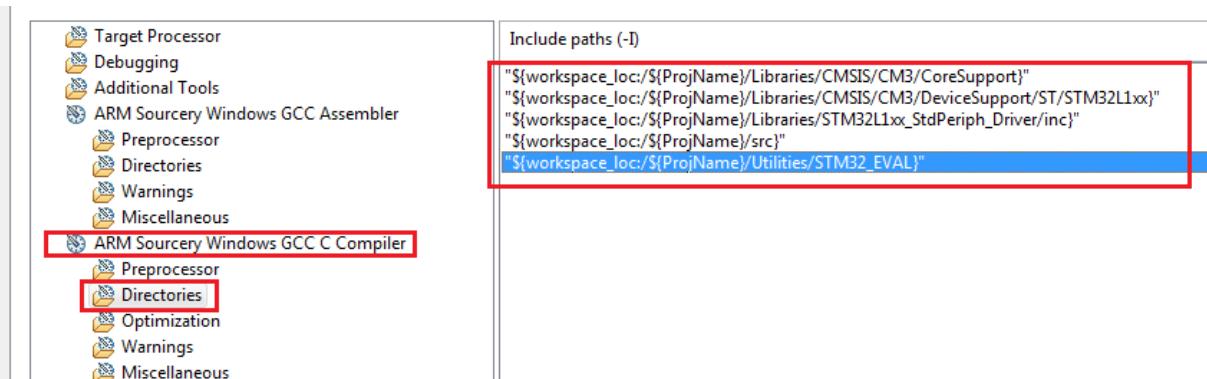
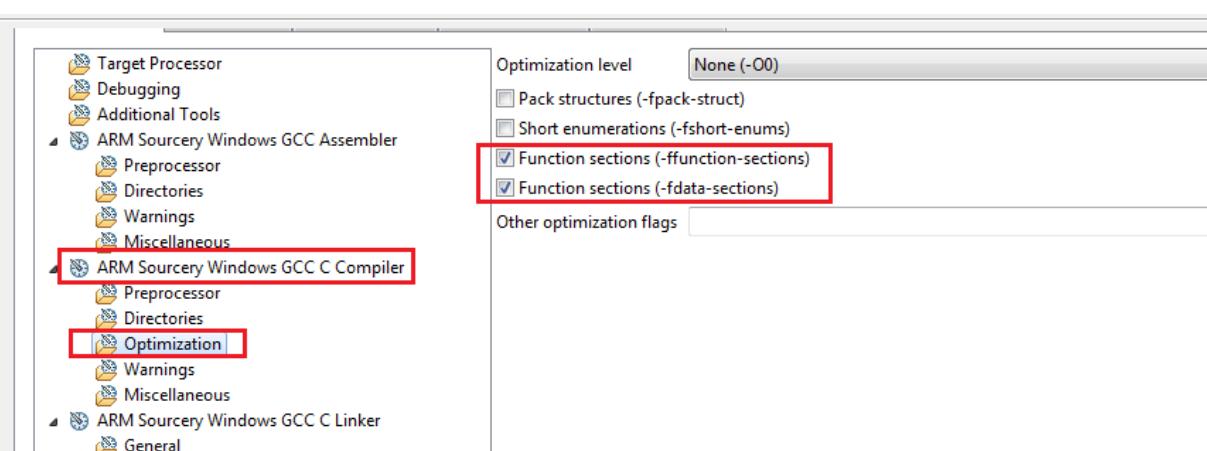
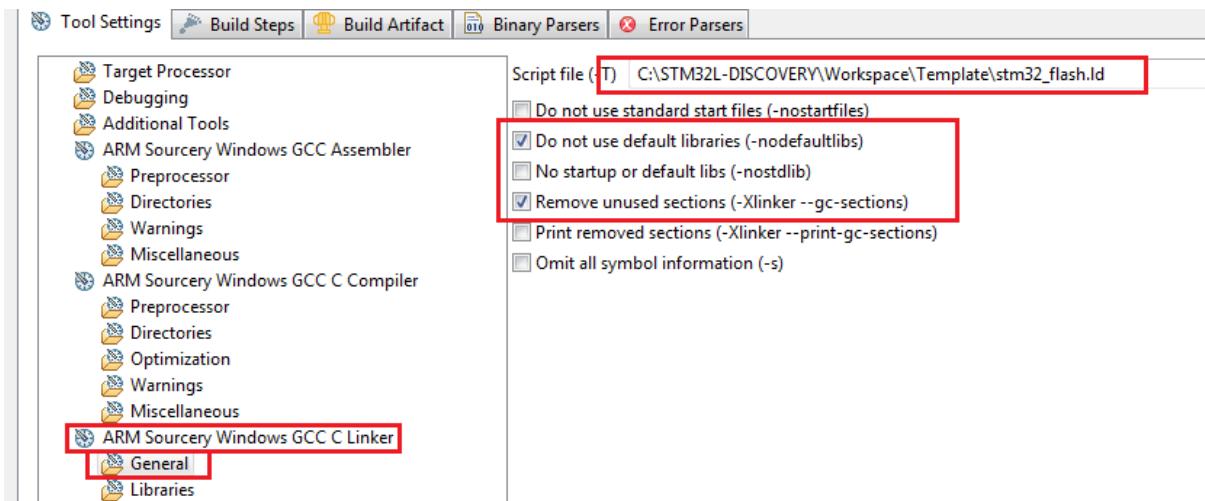


Figura B.12: *Botón Apply*

Figura B.13: *Preprocessor GCC Assembler*Figura B.14: *Directories GCC Assembler*Figura B.15: *Preprocessor GCC C Compiler*

Figura B.16: *Directories GCC C Compiler*Figura B.17: *Optimization GCC C Compiler*

Figura B.18: *General GCC C Linker*

```
**** Build of configuration Debug for project Template ****

cs-make all
Invoking: ARM Sourcery Windows GNU Print Size
arm-none-eabi-size --format=berkeley Template.elf
    text     data      bss      dec      hex filename
      72         8     132     212      d4 Template.elf
Finished building: Template.siz
```

Figura B.19: *Compilación Correcta*

Una vez hemos configurado nuestro proyecto, podemos compilarlo. Para ello, actuamos igual que vimos con Atollic. Tras pulsar el botón veremos en la consola que el resultado ha sido correcto (Figura B.18). Ahora ya sabemos como crear proyectos con Eclipse para STM32L-DISCOVERY.

Este Template que hemos creado podemos utilizarlo como template para nuevos proyectos en Eclipse.

APÉNDICE C

MAPEADO DE PINES

El siguiente anexo refleja una tabla de los pines utilizados en el proyecto. Aunque el STM32L152 tiene 83 pines I/Os, en la placa STM32L-Discovery tan sólo tenemos disponibles 56, distribuidos por igual en dos conectores, P1 y P2 (Figura C.1). Estos conectores pueden ser conectados a una protoboard, y los pines medidos mediante osciloscopio, analizador lógico o un voltímetro.

Estos pines son pines GPIO configurables, pudiendo tomar funciones alternativas como ya hemos hablado en la sección de GPIO. Sin embargo, 28 de ellos son utilizados por el display LCD que viene montado en la placa. Para poder utilizar estos pines como GPIO, o cualquiera de sus funciones alternativas deberemos extraer el display LCD de la placa.

Las siguientes tablas muestran que pines quedan libres después de utilizar los periféricos que han sido utilizados en el proyecto. El objetivo es mostrar la capacidad de la placa para ser ampliable mediante una placa externa de periféricos.

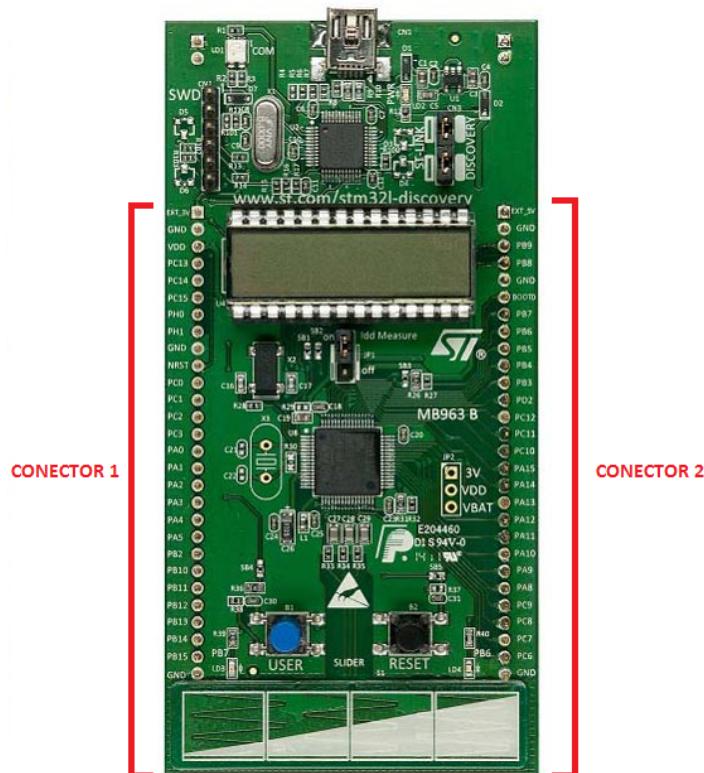


Figura C.1: *Conectores en STM32L-Discovery*

Conecotor 1	MCU PIN	Función
1	-	EXT_3V
2	-	GND
3	-	VDD
4	PC13	LIBRE
5	PC14	LIBRE
6	PC15	LIBRE
7	PH0	LIBRE
8	PH1	LIBRE
9	-	GND
10	-	NRST
11	PC0	LCD SEG14
12	PC1	LCD SEG15
13	PC2	LCD SEG16
14	PC3	LCD SEG17
15	PA0	USER BUTTON / TIM2_CH1 / EXTI
16	PA1	LCD SEG1
17	PA2	LCD SEG2
18	PA3	LCD SEG3
19	PA4	ADC_IN4 / DAC_OUT1
20	PA5	LIBRE
21	PB2	LIBRE
22	PB10	LCD SEG6
23	PB11	LCD SEG7
24	PB12	LCD SEG8
25	PB13	LCD SEG9
26	PB14	LCD SEG10
27	PB15	LCD SEG11
28	-	GND

Tabla C.2: *Pines en el conector 1*

Conecotor 2	MCU PIN	Función
1	-	EXT _5V
2	-	GND
3	PB9	LCD COM3
4	PB8	LCD SEG13
5	-	GND
6	-	BOOT0
7	PB7	LED3 / TIM4_CH2
8	PB6	LED4
9	PB5	LCD SEG5
10	PB4	LCD SEG4
11	PB3	LCD SEG3
12	PD2	LIBRE
13	PC12	LIBRE
14	PC11	LCD SEG23
15	PC10	LCD SEG22
16	PA15	LCD SEG12
17	PA14	LIBRE
18	PA13	LIBRE
19	PA12	LIBRE
20	PA11	LIBRE
21	PA10	LCD COM2
22	PA9	LCD COM1
23	PA8	LCD COM0
24	PC11	LCD SEG21
25	PC10	LCD SEG20
26	PC9	LCD SEG19
27	PC8	LCD SEG18
28	-	GND

Tabla C.4: *Pines en el conector 2*

Bibliografía

- [1] Web Arduino (<http://arduino.cc>). Accedido en Mayo de 2011.
- [2] Leaf Labs - The Mapple (<http://leaflabs.com/devices/maple/>). Accedido en Mayo de 2011.
- [3] LPCXpresso (<http://ics.nxp.com/lpcxpresso/>). Accedido en Junio de 2011.
- [4] Programming Arduino with AVR-GCC
(http://www.javiervalcarce.eu/wiki/Program_Arduino_with_AVR-GCC). Accedido en Junio 2011.
- [5] GCC, GNU Compiler Collection (<http://gcc.gnu.org/>). Accedido en Junio 2011.
- [6] GDB, GNU Project Debugger (<http://www.gnu.org/s/gdb/>). Accedido en Julio 2011.
- [7] STMicroelectronics - STM32L-Discovery
(<http://www.st.com/internet/evalboard/product/250990.jsp>). Accedido en Julio 2011.
- [8] STMicroelectronics - STM32L152RB (<http://www.st.com/internet/mcu/product/248820.jsp>). Accedido en Julio 2011.
- [9] 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.
- [10] Walter Mora F. and Alex Borbón. Edición de textos científicos L^AT_EX. Escuela de Matemáticas, Instituto Tecnológico de Costa Rica, 2009.
- [11] User Manual UM1079 - STM32L-DISCOVERY Rev 2. Junio 2011.
- [12] Reference Manual RM0038 - STM32L151xx and STM32L152xx advanced ARM-based 32-bit MCUs. Rev 4. Febrero 2011.
- [13] Datasheet DS6876 - STM32L151xx, STM32L152xx. Rev 5. Junio 2011.

- [14] Application Note AN3216 - Getting started with STM32L1xxx hardware development. Rev 5. Junio 2011.
- [15] Getting started with CMSIS - Douslos (http://www.douslos.com/knowhow/arm/CMSIS/CMSIS_Douslos_Tutorial.pdf). Accedido en Agosto 2011.
- [16] Atollic TrueSTUDIO STM32 Quickstart Guide. Rev 5. Septiembre 2010.
- [17] Atmega48PA/88PA/168PA/328PA Datasheet. Rev 8161A. Octubre 2009.
- [18] UM10360 LP17xx User Manual. Rev 2. 19 Agosto 2010.
- [19] LPCXpresso Getting started with NXP LPCXpresso. Rev 10. 7 Abril 2011.
- [20] STM32L1xx Standard Peripherals Library (StdPeriph_Lib). Rev 1.0.0. 31 Diciembre 2010.
- [21] Application Note AN3126 - Audio and waveform generation using the DAC. Rev 1. Mayo 2010.