Trabajo de fin de grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Programación de microcontroladores Cortex-M7 usando herramientas de generación de código para el sistema STM32F7 Discovery

Autor: Manuel Ángel Reyes Resta
Tutor: Manuel Ángel Perales Esteve

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo de fin de grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Programación de microcontroladores Cortex-M7 usando herramientas de generación de código para el sistema STM32F7 Discovery

Autor:
Manuel Ángel Reyes Resta

Tutor:
Manuel Ángel Perales Esteve
Profesor Contratado Doctor

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo de fin de grado: Programación de microcontroladores Cortex-M7 usando herramientas de generación de código para el sistema STM32F7 Discovery

Autor: Manuel Ángel Reyes Resta

Tutor: Manuel Ángel Perales Esteve

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mis padres

Agradecimientos

Cuando era un niño siempre decía que quería ser cocinero y me encantaba la historia y la geografía, ciencias que ahora están muy lejos del currículum y la carrera profesional que poco a poco voy formando con el paso de los días. El cambio de rumbo es evidente y tiene culpables, culpables que desde que soy un niño me han hecho crecer, marcándome un camino a seguir.

El primero de ellos fue mi profesor de tecnología que me acompañó durante mis años en el instituto. Su forma de dar las clases, su temario, su forma de evaluar y los proyectos que nos mandaba como tarea me motivaron y forjaron el camino que estoy siguiendo. Muchas gracias Jesús, por tus años de docencia y tu pasión por la ingeniería, aun recuerdo cuando me enseñaste los mapas de Karnaugh que a día de hoy sigo usando, muchísimas gracias por todo.

El segundo culpable es mi tutor, Manuel Ángel Perales. Jamás pensé que una persona podría conseguir que a otra le gustase una llamada a un registro. Por lo visto se puede, y es siempre más fácil con una sonrisa en la cara y las ganas de enseñar. Muchísimas gracias por tu ayuda y por conseguir que me ame lo que estudio.

Para encontrar a los últimos culpables no tengo que salir de mi casa, son mis padres Manuel Ángel y Mercedes. Dos personas que son mucho más ingenieros que yo. Constantes, trabajadores y siempre buscando soluciones. Su forma de vivir y su ambición han conseguido que nunca deje de crecer y siempre quiera más. Muchas gracias.

Para concluir, me gustaría agradecer a mi padre el ejemplo que siempre me ha dado. Debería saber de sobra que sus conocimientos, sus ganas y sus ansias por saber son mayores que cualquier título firmado por un miembro de la Casa Real, es ejemplo todo un ejemplo a seguir para mí, casi un imposible que veo muy lejos y que probablemente nunca acance. Este trabajo, estos años de ingeniería son por él, para que se sienta orgulloso de mí y así agradecerle el ejemplo que siempre me ha dado. No tengo palabras para agradecer y explicar lo que él supone para mí. Voy a ser ingeniero, pero a mi lo que me gustaría es ser como mi padre.

Manuel Ángel Reyes Resta

Alumno del Grado de Ingeniería Electrónica, Robótica y Mecatrónica

Sevilla, 2017

Resumen

En el siguiente informe se realiza un estudio exhaustivo de la placa STM32F7 Discovery, producto de la compañía STMicroelectronics que aun no ha sido usada en ninguna asignatura de la ETSI y cuyo alto potencial, hace de él un kit muy interesante para muchas de las asignaturas del Departamento de Electrónica de la ETSI.

Así pues, este trabajo incluye un amplio estudio teórico del micro y de las novedades del córtex M7 frente a las versiones anteriores, seguido de una explicación detallada de como preparar un ordenador para comenzar a trabajar con esta placa.

Como el groso de las asignaturas que incluyen trabajos y prácticas con microcontroladores lo ocupa la programación de estos, se realiza una explicación minuciosa de la configuración y utilización de los periféricos básicos gracias a la formulación de problemas que son resueltos utilizando las herramientas aportadas por la compañía. En otras palabras, se explica la configuración de algunos periféricos paso a paso, usando las herramientas gratuitas disponibles en la web de STM. Además, se ofrecen varios softwares capaces de crear interfaces para la pantalla incluida en el kit.

Finalmente, en la conclusión, se realiza un breve resumen de las ventajas y desventajas del producto frente a los que la ETSI usa normalmente, así como otra conclusión en la que se establecen posibles caminos que seguir por otros alumnos a partir del trabajo realizado.

Abstract

The following technical report includes an exhaustive study of the board STM32F7 Discovery which is a product of the company STMicroelectronics that has not been used in any course of ETSI yet and whose potential makes it a really interesting kit that could be used in many courses of the Department of Electronics.

It starts with a theoretical description of the changes of the cortex M7, followed by a detailed explanation of how to prepare a computer to start using the board.

As most of the time spent in subjects that include works and practices with microcontrollers is occupied by the programming of this board, it is carried out a detailed explanation of the configuration and use of the basic peripheral through the formulation of problems that are solved using the tools provided by the firm. In other words, it is explained the configuration of some peripheral step by step, using the free tools available in the web of STM. In addition, it is offered some software able to create interfaces for the screen included in the kit.

Finally there is a section that sums up the advantages and disadvantages of the product compared with the other boards that the Department usually uses. There is also a conclusion that explains the doors that this project leaves open for future labworks, others TFGs or even TFMs.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xv**

Índice de Tablas **xix**

Índice de Figuras **xxi**

Notación **xxiv**

1. Introducción al Trabajo **2**

2. El innovador Cortex M7 **3**

2.3. Un paso más cerca del procesador a tiempo real *6*

2.2 Un paso más cerca del Procesamiento Digital de Señal *8*

2.3 Bus AXI-M *10*

2.4 Smart Architecture *12*

2.5 Conclusión del capitulo *13*

3. STM32F7 Discovery **14**

3.1 Características del kit STM32F7 Discovery *15*

3.1.1 Periféricos 15

3.1.2 Pinout 17

4. Entorno de trabajo para el kit STM32F7 Discovery **19**

4.1 Hardware *20*

4.2 Drivers y paquetes *20*

4.2.1 STM32 ST-LINK Utility 20

4.2.2 STM32 Virtual COM Port Driver 21

4.2.3 Paquete Java SE Runtime Enviroment 8 21

4.2.4 Paquete .NET Framework 4.5 21

4.3 Software *21*

4.3.1 STM32CubeMx 22

4.3.2 MDK-ARM V5: Keil µVision5 26

4.3.3 Tera Term 26

5. Programación de periféricos **27**

5.1 Programación de GPIO *28*

5.1.1 Descripción del periférico 28
5.1.2 Librería del periférico 28
5.1.3 Configuración del periférico 28
5.1.4 Ejemplo práctico 30
5.2 Programación de un Timer *35*
5.2.1 Descripción del periférico 35
5.2.2 Librería del periférico 35
5.2.3 Configuración del periférico 35
5.2.4 Ejemplo práctico 36
5.3 Programación de la UART *40*
5.3.1 Descripción del periférico 40
5.3.2 Librería del periférico 40
5.3.3 Configuración del periférico 40
5.3.4 Ejemplo práctico 41
5.4 Programación de la ADC *45*
5.4.1 Descripción del periférico 45
5.4.2 Librería del periférico 45
5.4.3 Configuración del periférico 45
5.4.4 Ejemplo práctico 46
5.5 Programación de la LCD *49*
5.5.1 Descripción del periférico 49
5.5.2 Librería del periférico 49
5.5.3 Configuración del periférico 51
5.5.4 Ejemplo práctico 51
5.6 Programación del Touchscreen *56*
5.6.1 Descripción del periférico 56
5.6.2 Librería del periférico 56
5.6.3 Configuración del periférico 56
5.6.4 Ejemplo práctico 56

6. Creación de interfaces **58**
6.1 Software de creación de interfaces para la STM32F7 Discovery *59*
6.1.1 Embedded Wizard GUI builder 59
6.1.2 TouchGFX 59
6.1.3 STemWin 59
6.2 Conclusiones *59*

7. Conclusión **60**
7.1 Aplicación en asignaturas *61*
7.2 Posibles futuros trabajos *61*

8. Anexo de códigos **62**
8.1 Ejemplo práctico GPIO *63*
8.1.1 main.c 63
8.2 Ejemplo práctico Timer *67*
8.2.1 main.c 67
8.2.2 Vector de interrupciones 72
8.3 Ejemplo práctico UART *75*
8.3.1 main.c 75
8.3.2 Vector de interrupciones 79
8.4 Ejemplo práctico ADC *84*
8.4.1 main.c 84
8.4.2 Vector de interrupciones 89
8.5 Ejemplo práctico LCD *92*
8.5.1 main.c 92
8.6 Ejemplo práctico Touchscreen *101*

8.6.1 main.c 101
8.6.2 Vector de interrupciones 108

ÍNDICE DE TABLAS

Tabla 2-1: Comparación Cortex M4 y M7 4
Tabla 2-2: Funciones principales ITCM y DTCM 7
Tabla 3-3: Componentes del kit 15

ÍNDICE DE FIGURAS

Figura 2-1. Conjunto de instrucciones de los Cortex Mx. 4
Figura 2-2. Arquitectura ARM Cortex M4 y M7. 5
Figura 2-3. Arquitectura del núcleo de un ARM Cortex M7. 6
Figura 2-4. TCM de un ARM Cortex M7. 6
Figura 2-5. Arquitectura del núcleo de un ARM Cortex M7. 8
Figura 2-6. Esquema del Dual-Issue. 9
Figura 2-7. Arquitectura del núcleo de un ARM Cortex M7. 10
Figura 2-8. Interfaz AXI to Multi-AHB. 10
Figura 2-9. Arquitectura del núcleo de un ARM Cortex M7. 11
Figura 2-10. Smart application. 12
Figura 2-11. Kit STM32F7 Discovery. 16
Figura 2-12. Pinout de la placa. 17
Figura 2-13. Conexiones del conector para Arduino UNO. 18
Figura 4-14. Kit STM32F7 Discovery. 20
Figura 4-15. Creación de un nuevo proyecto STM32CubeMx. 22
Figura 4-16. Elección de microcontrolador en STM32CubeMx. 22
Figura 4-17. Pantalla principal de STM32CubeMx. 23
Figura 4-18. Configuración del reloj usando STM32CubeMx. 23
Figura 4-19. Pantalla de configuración de STM32CubeMx. 24
Figura 4-20. Generación del código de un proyecto de STM32CubeMx. 24
Figura 4-21. Apertura de un proyecto generado en STM32CubeMx. 25
Figura 4-22. Pantalla principal Keil µVision5. 25
Figura 5-23. Configuración del reloj cerámico con STM32CubeMx. 30
Figura 5-24. Configuración de la frecuencia del reloj cerámico con STM32CubeMx. 30
Figura 5-25. Configuración del tipo de debug en STM32CubeMx. 31
Figura 5-26. Configuración de un GPIO en STM32CubeMx. 31

Figura 5-27. Selección de la función de un pin en STM32CubeMx. 32
Figura 5-28. Configuración del núcleo en STM32CubeMx. 32
Figura 5-29. Configuración de las interrupciones en STM32CubeMx. 33
Figura 5-30. Configuración de la freuencia del reloj cerámico con STM32CubeMx. 37
Figura 5-31. Activación de un timer con STM32CubeMx. 37
Figura 5-32. Activación de la interrupción de un timer con STM32CubeMx. 37
Figura 5-33. Configuración de la interrupción de un timer con STM32CubeMx. 38
Figura 5-34. Configuración de la frecuencia del reloj con STM32CubeMx. 41
Figura 5-35. Configuración del debug con STM32CubeMx. 42
Figura 5-36. Activación de la UART con STM32CubeMx. 42
Figura 5-37. Configuración de la UART con STM32CubeMx. 42
Figura 5-38. Icono Options for Target en MDK-ARM V5: Keil µVision5. 43
Figura 5-39. Options for Target en MDK-ARM V5: Keil µVision5. 43
Figura 5-40. Options for Target en MDK-ARM V5: Keil µVision5. 44
Figura 5-41. Configuración del pin PF10 para su uso como ADC. 47
Figura 5-42. Configuración del ADC3. 47
Figura 5-43. Configuración de parámetros del ADC3. 48
Figura 5-44. Funciones del LCD. 50
Figura 5-45. Configuración del acelerador de gráficos DMA2D 51
Figura 5-46. Configuración de la SDRAM 51
Figura 5-47. Configuración del I2C 51
Figura 5-48. Configuración del LTCD 52
Figura 5-49. Configuración del SPI 52
Figura 5-50. Configuración del micro 52
Figura 5-51. Icono Options for Target en MDK-ARM V5: Keil µVision5. 53
Figura 5-52. Include paths en MDK-ARM V5: Keil µVision5. 53
Figura 5-53. Incluir path del kit en MDK-ARM V5: Keil µVision5. 54
Figura 5-54. Icono Manage Project Items en MDK-ARM V5: Keil µVision5. 54
Figura 5-55. Librerías de BSP en MDK-ARM V5: Keil µVision5. 54
Figura 5-56. Esquemático del LCD. 57
Figura 6-57. Embedded Wizard GUI builder 59

Notación

TCM	Tightly Coupled Memories
RISC	Reduced Instruction Set Computer
SIMD	Single Instrution, Multiple Data
MPU	Microprocessor
FPU	Float Point Unit
ART	Adaptive Real-Time
DMA	Direct Memory Access
ITCM	Instruction Tightly Coupled Memories
DTCM	Data Tightly Coupled Memories
ALU	Arithmetic Logic Unit
MAC	Media Access Control
RAM	Random Access Memory
MPU	Memory Protection Unit
DMA	Direct Memory Access
UART	Universal Asynchronous Receiver-Transmitter

1. INTRODUCCIÓN AL TRABAJO

Ideas generales del trabajo

En el Departamento de Ingeniería Electrónica de la ETSI de Sevilla, en las asignaturas relacionadas con sistemas electrónicos o microcontroladores en general, es muy común encontrarse con proyectos, prácticas o incluso asignaturas completas centradas en una placa o sistema determinado que permite al alumno conocer la asignatura de una manera mucho más práctica y llevadera. Siempre han existido manuales, puesto que se usaban placas que llevaban años en el mercado, pero este trabajo ha sido una oportunidad de enfrentarme cara a cara con un producto desde cero, todo un reto que, espero, pueda ser compartido por toda la comunidad educativa.

Se basa en una primera toma de contacto con el producto en cuestión, la STM32F7 Discovery de STMicroelectronics. Así pues, a lo largo de este documento se explicarán los conceptos básicos para poder usar tanto la placa, como sus periféricos más básicos.

De esta forma, el informe tendrá un desarrollo escalonado. Primero se describirá la placa, para luego seguir con un resumen del desarrollo del trabajo: desde que el tutor me dio el producto, hasta que comencé a escribir las primeras palabras de este informe, incluyendo todos y cada uno de los problemas que me he encontrado durante su realización. A continuación, en una sección, se incluirán los pasos necesarios que se deben seguir para la instalación de todos los drivers y programas necesarios para su uso con el fin de que cualquier persona que lea esto sea capaz de comenzar a utilizarlo.

Seguidamente se encuentra el punto fuerte del proyecto, la explicación de la configuración, uso y librerías de algunos periféricos, seguidos de un ejemplo válido para que cualquier persona con nociones de programación sea capaz de aprender a usar la STM32F7 Discovery.

Para facilitar el acceso a los pines de la placa se ha diseñado una PCB que extiende éstos de forma que se pueden conectar a una placa de pruebas para así, evitar daños en la pantalla. También se deja diseñada una PCB a la que se le pueden conectar varios sensores o periféricos externos a la placa para que así el alumno, en una posible futura práctica no tenga que hacer uso de las placas de pruebas y componentes sueltos.

Finalmente, como compendio, se incluye un apartado que habla de los posibles usos de este informe en asignaturas, así como en otros TFG o TFM.

2. EL INNOVADOR CORTEX M7

Breve introducción a los cortex Mx

Análisis exhaustivo de las nuevas características del M7

Los microprocesadores ARM se diferencian fundamentalmente por el tipo de cortex que tienen. Como cualquier otra tecnología, éstos avanzan conforme pasa el tiempo, se van añadiendo nuevas características y funcionalidades que se adaptan a las necesidades de la sociedad y/o el usuario. En este capítulo del informe, se realizará una breve comparación entre las dos últimas versiones de los cortex que se han diseñado así como una somera mirada a los primeros modelos. Finalmente, se hará un inciso sobre las características fundamentales del cortex M7, que es el que incluye la placa STM32F7 Discovery, que es la que he estudiado y analizado a fondo para que sea utilizada.

Aunque lo que de verdad nos interesa es centrarnos en el cortex M7, como ya he indicado, realizaré una breve comparación con los cortex primerizos. Todas y cada una de las STM32 de cortex Mx incluyen las siguientes características:

- Arquitectura de 32-bit de tipo RISC.
- Totalmente diseñada para ser programada en lenguaje C.
- Latencia de interrupción muy baja, lo que la hace determinista.
- Código de memoria de tamaño reducido.
- Alta eficiencia energética y soporte para sistemas de baja potencia.
- Sistema en tiempo real.

En cuanto al conjunto de instrucciones ha ido creciendo a la vez que los procesadores (Figura 2-1). Partiendo de la familia M0, la diferencia con la inmediata siguiente, que es la M3, es bastante grande, de hecho esta última casi triplica el número de instrucciones. Los M4 también suponen un gran salto tecnológico, aunque no tan grande como el anterior y, finalmente, los M7 apenas suponen un cambio en comparación con los M4, si hablamos de conjunto de instrucciones.

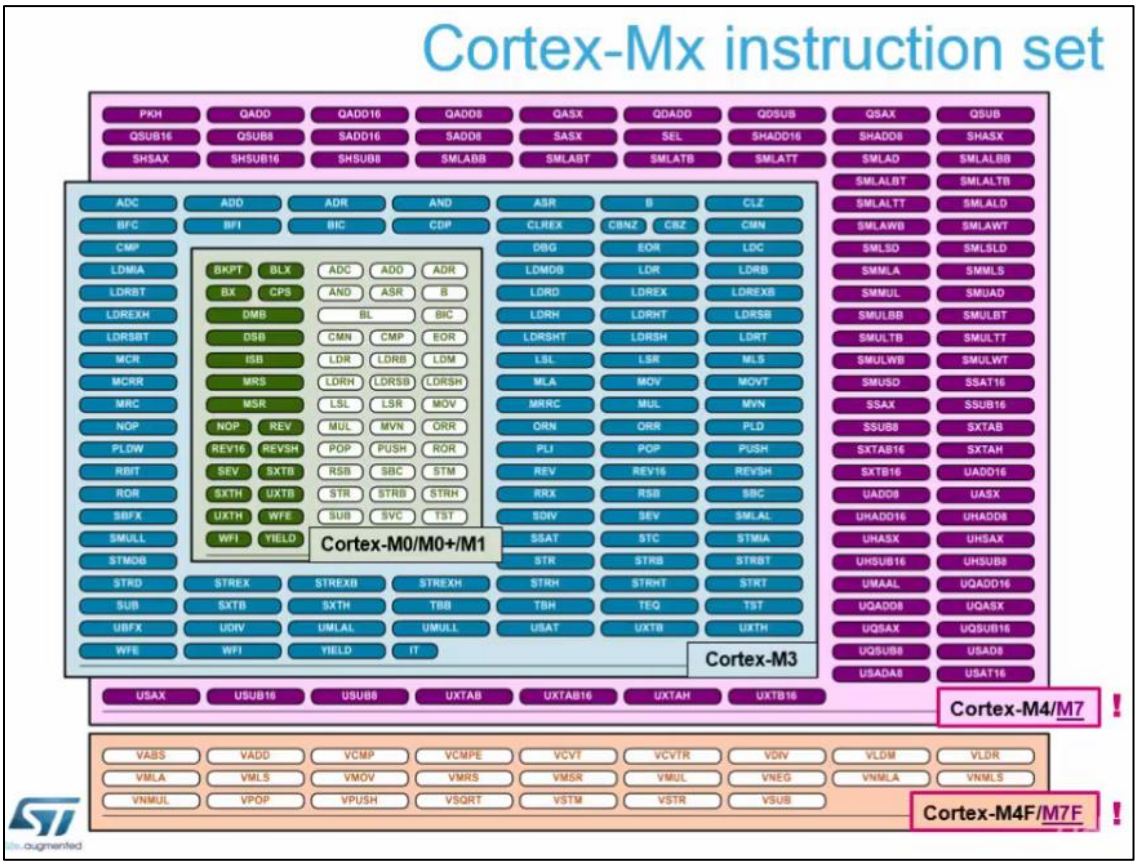

Figura 2-1. Conjunto de instrucciones de los Cortex Mx.[1]

Dejando a un lado el conjunto de instrucciones que, como ya he dicho, es bastante parecido para las dos últimas versiones, el cortex M4 y el M7 tienen diferencias que marcan la distancia entre ambas tecnologías:

Tabla 2-1: Comparación Cortex M4 y M7

Cortex M4	Cortex M7
Arquitectura ARMv7E-M	Arquitectura ARMv7E-M
Arquitectura Harvard	Arquitectura Harvard
Tres etapas segmentadas	Seis etapas segmentadas
DIV en 12 ciclos máximo	DIV en 12 ciclos máximo
Instrucciones SIMD	Instrucciones SIMD
MPU	MPU
FPU	FPU
Latencia de interrupción de 12 ciclos	Latencia de interrupción de 12 ciclos
	Arquitectura superescalar de doble edición

[1] Seminario online de STMicroelectronics sobre la STM32F7. https://st-mooc.udemy.com/stm32f7-hands-on-workshop

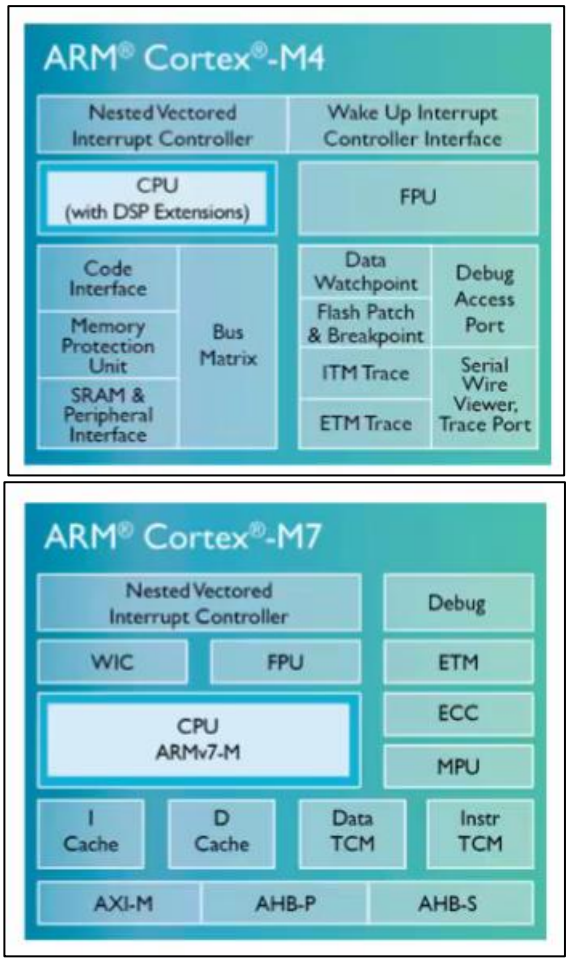

Figura 2-2. Arquitectura ARM Cortex M4 y M7.[2]

Tal y como podemos observar en la tabla 2-1 la única diferencia superficial entre los M4 y M7 son las etapas segmentadas y la arquitectura superescalar. No obstante, aunque sea tan solo una pequeña diferencia, esto supone que el M7 se encuentre un paso más cercano al proceso digital de señal: Carga y almacenamiento en paralelo basado en aritmética e inexistencia de bucles innecesarios. Además, también se encuentra un paso por delante en cuanto a sistemas de tiempo real: Tiene memorias fuertenemente acopladas y una interfaz AXI-M con memoria cache.

[2] Seminario online de STMicroelectronics sobre la STM32F7. https://st-mooc.udemy.com/stm32f7-hands-on-workshop

2.3. Un paso más cerca del procesador a tiempo real

Si echamos un vistazo a la arquitectura del núcleo, fijándonos en la parte del TCM:

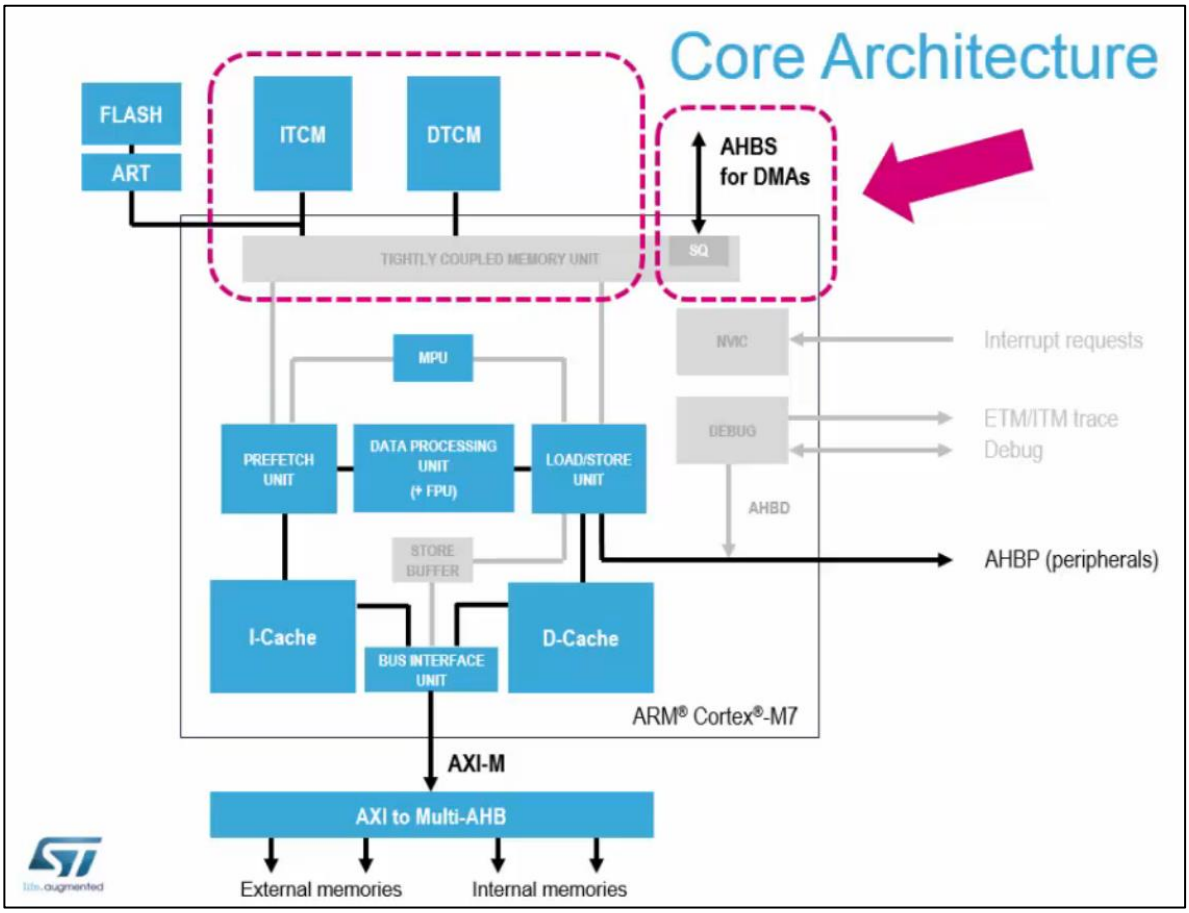

Figura 2-3. Arquitectura del núcleo de un ARM Cortex M7.[3]

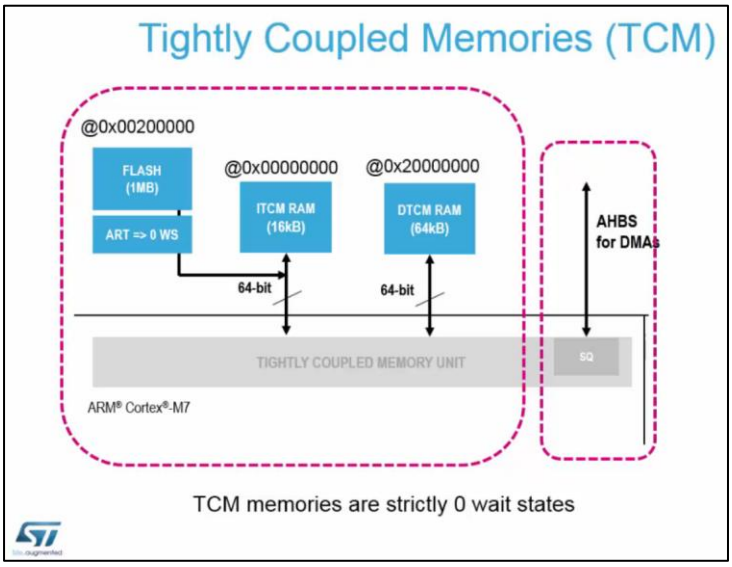

Figura 2-4. TCM de un ARM Cortex M7.[4]

[3] Seminario online de STMicroelectronics sobre la STM32F7. https://st-mooc.udemy.com/stm32f7-hands-on-workshop
[4] Ídem

Resumiendo lo que vemos en la Figura 2-4, podemos decir que tenemos una arquitectura de 64 bits conectada a una TCM que forma parte del núcleo. Dicha arquitectura esta formada por un módulo Flash de 1MB que se encuentra conectado con un acelerador ART. Haciendo una pequeña comparación con el cortex M4, el M7 tiene 64 bits mientras que el M4 tiene 32 bits.

Al TCM se encuentran conectados dos módulos, uno de ellos es el modulo de instrucciones de 16 kB que nos proporciona la seguridad de tener cero estados de espera para el acceso a cualquier rutina almacenada en la RAM. A la vez también se encuentra enlazado un modulo DTCM de 64 kB donde podemos almacenar datos en general, que nos garantiza del mismo modo cero estados de espera en el acceso.

También existe un módulo exclusivamente dedicado para el acceso del DMA.

En cuanto a los bloques de ITCM y DTCM sus funciones principales son:

Tabla 2-2: Funciones principales ITCM y DTCM

ITCM RAM (16kB)	DTCM RAM (64kB)
Modo crítico	Información frecuentemente usada
Rutinas del servicio de interrupción	Apilar/Amontonar
Altamente determinista	Coeficientes del procesamiento digital de señales

2.2 Un paso más cerca del Procesamiento Digital de Señal

Echando un vistazo de nuevo a la arquitectura del núcleo, observamos que en el centro de este se encuentran un bloque de captación previa (Prefetch Unit), uno de procesamiento de datos y uno de carga y almacenaje:

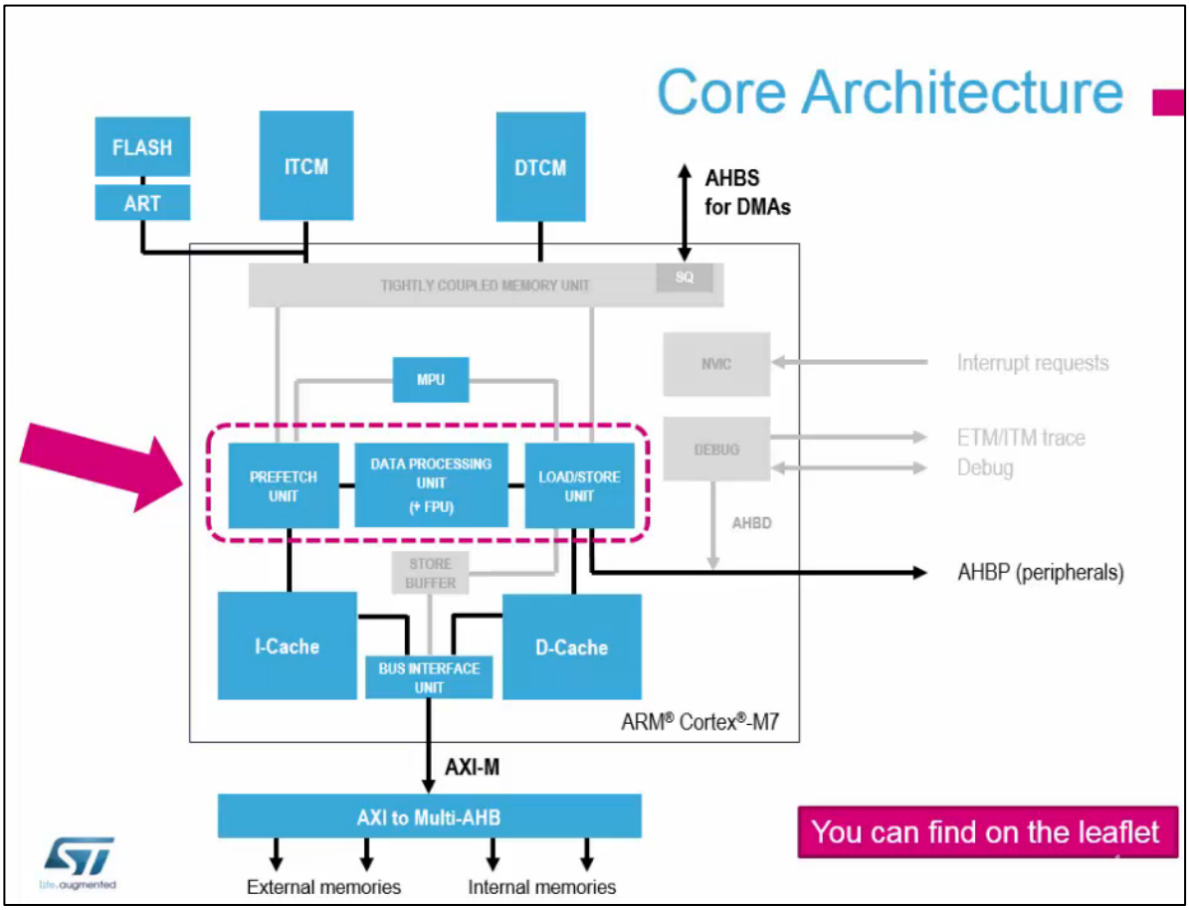

Figura 2-5. Arquitectura del núcleo de un ARM Cortex M7.[5]

Si hacemos zoom sobre la zona en cuestión, observamos que se alimenta un bloque de 32 bits con información con un ancho de 64. Esto se debe a que el bloque Prefetch Unit, separa en dos cadenas de 32 bits la entrada, lo que conocemos como dual-issue. De esta forma podemos cargar o almacenar o realizar dos accesos a la ALU simultáneamente. Del mismo modo es posible mezclar ambas cadenas para formar una MAC, un Branch o una FPU. De esta forma, casi todas las instrucciones pueden ser ejecutadas en paralelo.

[5] Seminario online de STMicroelectronics sobre la STM32F7. https://st-mooc.udemy.com/stm32f7-hands-on-workshop

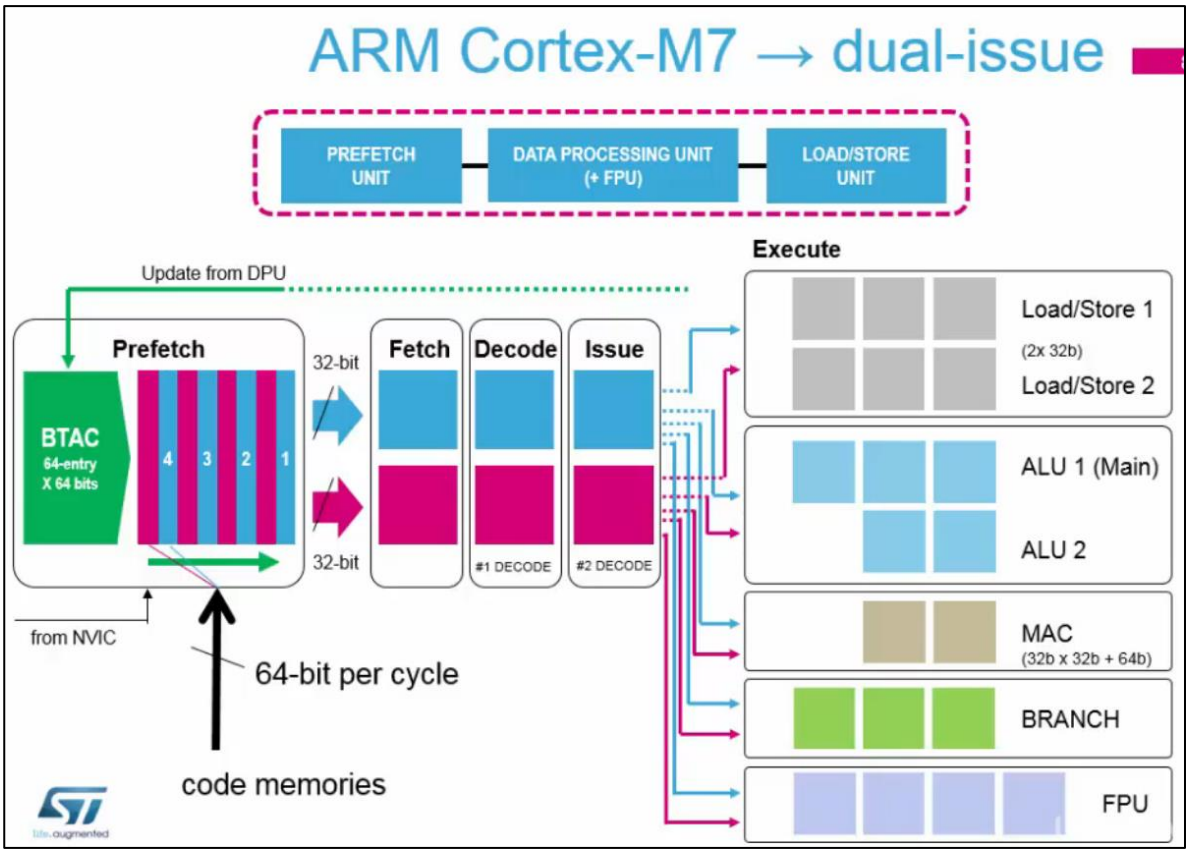

Figura 2-6. Esquema del Dual-Issue.[6]

De esta forma, mientras los Cortex M4 tomaban dos ciclos para almacenar o cargar una instrucción, los M7 lo hacen de forma paralela y sin ningun tipo de esperas. En conclusión, podemos decir que el M7 posee una arquitectura Superscalar que nos permite ejecutar dos instrucciones al mismo tiempo.

En cuanto al tiempo de espera, como hemos dicho antes, son mucho mayores en el M4, ya que toma al menos 3 ciclos, puesto que necesita recibir el resultado de una operación aritmética, incluso con la especulación del branch. Por su parte, el M7, gracias a la prediccion del branch, tan solo toma un ciclo.

[6] Seminario online de STMicroelectronics sobre la STM32F7. https://st-mooc.udemy.com/stm32f7-hands-on-workshop

2.3 Bus AXI-M

En el apartado anterior hemos visto como se accede una pequeña parte de las memorias IRAM y DRAM a través del TCM, pero ¿Qué ocurre con el resto de memorias?, ¿Cómo se accede a ellas? Para ello, el Cortex M7 incluye un nuevo bloque que conforma el bus AXI-M:

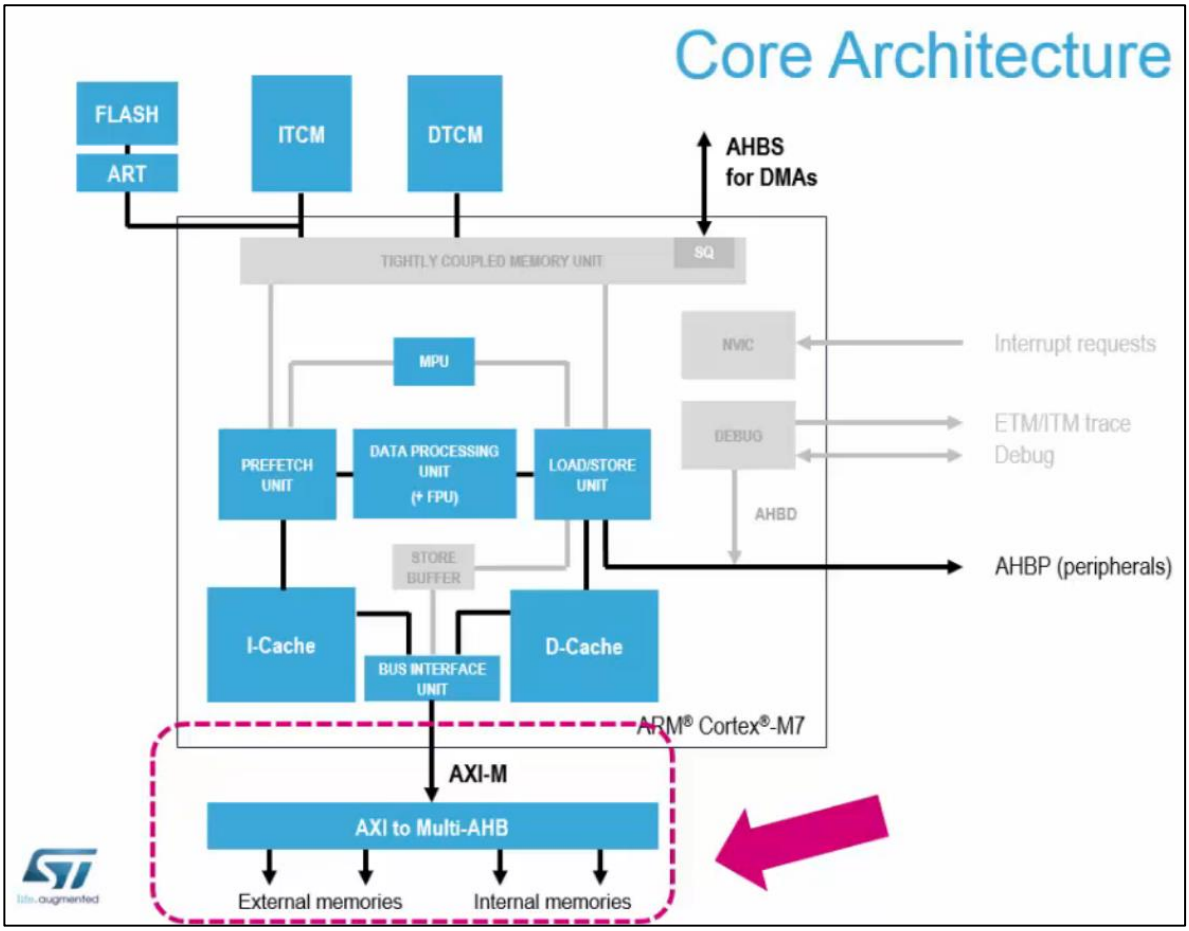

Figura 2-7. Arquitectura del núcleo de un ARM Cortex M7.[7]

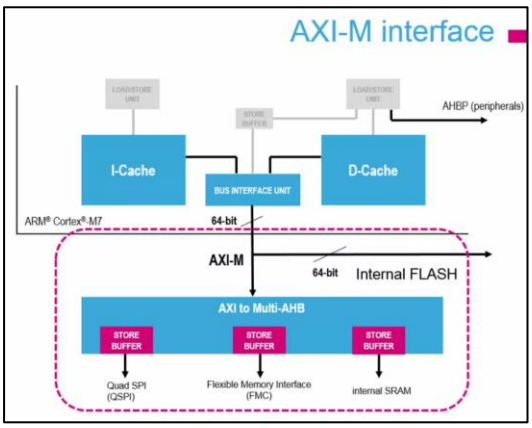

Figura 2-8. Interfaz AXI to Multi-AHB.[8]

[7] Seminario online de STMicroelectronics sobre la STM32F7. https://st-mooc.udemy.com/stm32f7-hands-on-workshop
[8] Ídem

Al igual que el anterior funciona a 64 bits, tal y como se muestra en la Figura 2-9. EL AXI-M no es más que un buffer entre las RAMs externas más lentas y el núcleo en sí.

El bus AXI-M conecta las caché de datos e instrucciones con el bus Multi-AHB que da acceso a las memorias externas e internas. Del mismo modo que en el TCM existe un branch que se dirige a la flash del sistema, en el bus AXI-M ocurre lo mismo, solo que en este caso no pasa por el ART accelerator.

La matriz-bus externa que va por el bus AXI-M es la interfaz que va al núcleo a través de la caché. Es una manera de distribuir todas las instrucciones del sistema, que al fin y al cabo es lo mismo que las cachés que tenemos a continuación:

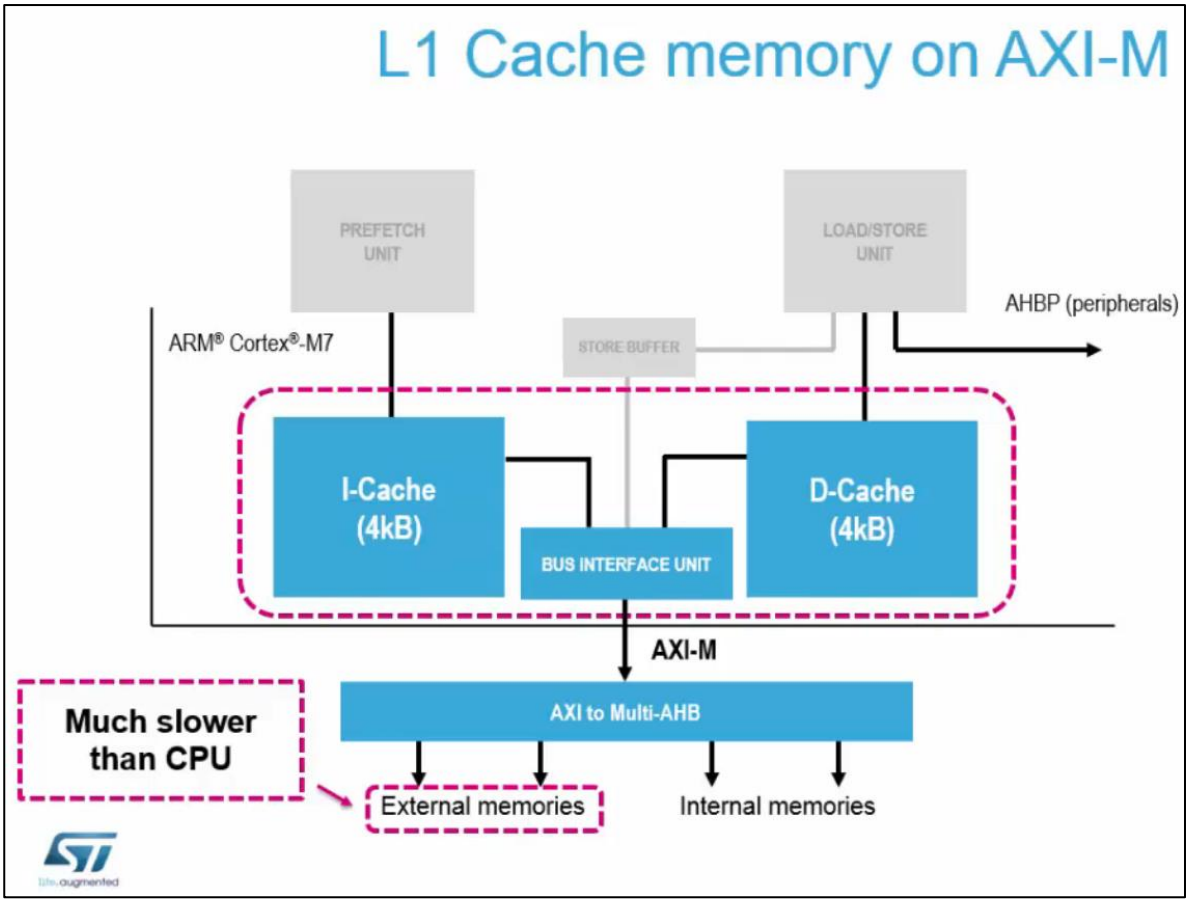

Figura 2-9. Arquitectura del núcleo de un ARM Cortex M7.[9]

Las memorias externas son lentas porque necesitan ser procesadas por la memoria caché para garantizar cero estados de espera. Por su parte, las memorias internas no son tan lentas porque ya se encuentran instaladas en la matriz-bus y, por lo tanto, solo necesita un camino al núcleo.

En resumen, el bus AXI-M es la unión entre una interfaz de las memorias internas y externas con las cachés de instrucciones y datos que lo conectan con el núcleo. Este proceso está establecido de una forma que evita cualquier estado de espera por parte del micro, acelerando los procesos en general.

El tamaño de las cachés del sistema es proporcional a la velocidad del núcleo y está establecida a 4kB tanto para el de instrucciones como para el de datos. Este tamaño cambiará al mismo tiempo que la tecnología y las necesidades del usuario.

[9] Seminario online de STMicroelectronics sobre la STM32F7. https://st-mooc.udemy.com/stm32f7-hands-on-workshop

2.4 Smart Architecture

En los tres anteriores puntos he descrito bloques pertenecientes al núcleo del sistema, pero durante esta sección voy a describir una aplicación de este.

Gracias a la configuración del núcleo y al cortex M7:

- Enviar datos al núcleo usando el ART Accelerator.
- Usando el mismo bus que el ART Accelerator, enviar vectores desde el ITCM al núcleo.
- Se pueden enviar al núcleo de forma paralela datos o instrucciones usando el bus de memoria externa y el caché.
- Usando un bus diferente al ITCM y ART Accelerator, podemos escribir datos sobre el DTCM.
- Gracias a uno de los DMA, es posible escribir sobre la SRAM usando la caché.

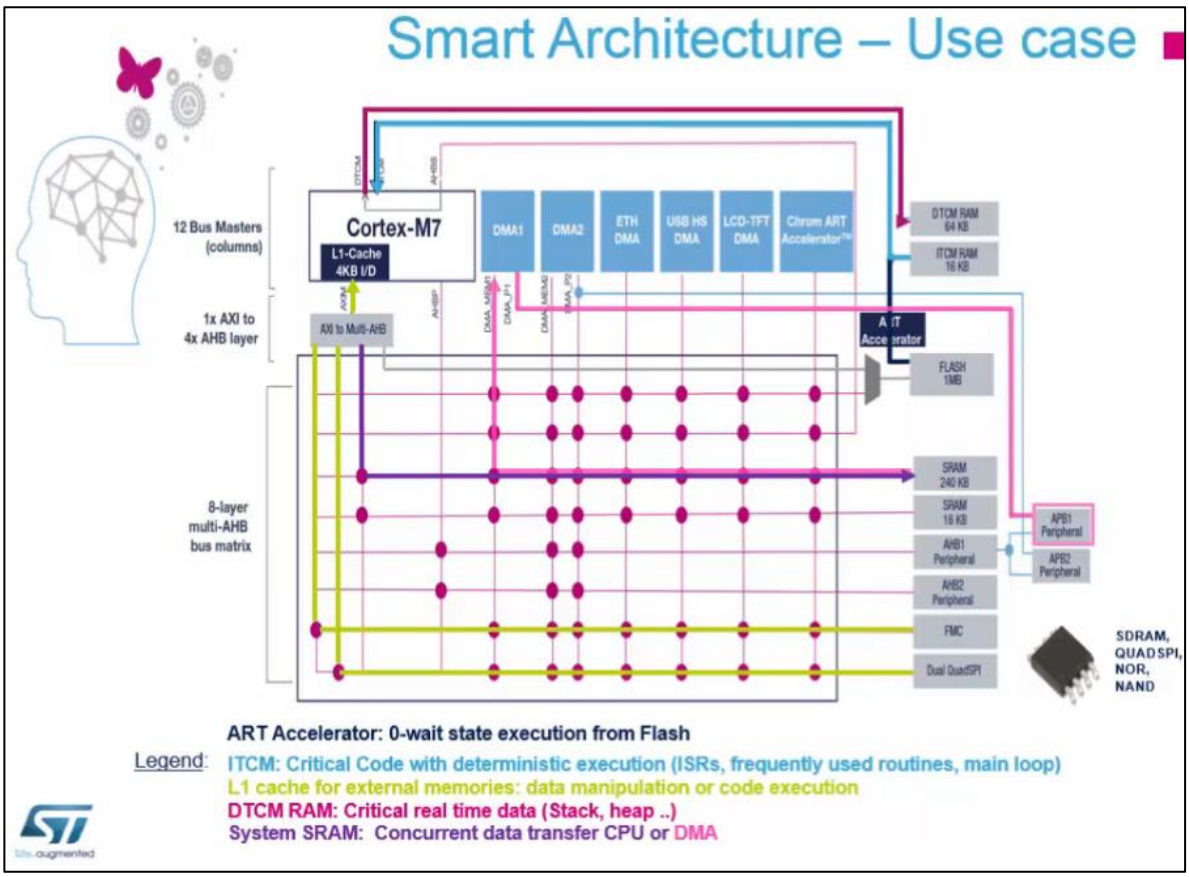

Figura 2-10. Smart application.[10]

[10] Seminario online de STMicroelectronics sobre la STM32F7. https://st-mooc.udemy.com/stm32f7-hands-on-workshop

2.5 Conclusión del capitulo

Así pues, gracias a todo lo explicado en los puntos anteriores, quedan definidas las grandes diferencias del cortex M7 sobre el M4 y el resto de la serie Mx:

- Un superscalar que nos permite ejecutar dos instrucciones por ciclo.
- Realizar branchs en un solo ciclo.
- Dispone de un sistema caché que compensa la lentitud del acceso a las memorias externas.
- Cuenta con un sistema de gran ancho de banda para cualquier aplicación.

3. STM32F7 DISCOVERY

Breve introducción al microcontrolador

Características, periféricos y pinout

La empresa STMicroelectronics lleva apostando unos años por incluir en sus microcontroladores núcleos ARM de cortex Mx, pues su funcionalidad y características otorgan la posibilidad de incluir varios periféricos en la placa para así crear un producto completo que puede ser usado desde el ámbito educativo, al profesional; desde una universidad a una empresa internacional.

El departamento de Ingeniería Electrónica de la ETSI de la Universidad de Sevilla apuesta por los microcontroladores como forma de enseñanza en muchas de sus asignaturas pero, a lo largo de mi vida como estudiante, nunca había usado un núcleo ARM ni siquiera un producto de la empresa STMicroelectronics.

Así pues, en este capítulo se tratarán las características de la placa que ha sido investigada y usada durante la parte prácitca del desarrollo de este trabajo de fin de grado.

El kit STM32F7 Discovery permite a los usuarios desarrollar y compartir aplicaciones con el resto de microcontroladores de la serie STM32F7 con núcleo ARM Cortex M7. Engloba una gran cantidad de aplicaciones gracias a sus sistemas de audio, sensores, gráficos, seguridad, video y conexiones de alta velocidad.

3.1 Características del kit STM32F7 Discovery

3.1.1 Periféricos

El kit incorpora los siguientes componentes:

Tabla 3-3: Componentes del kit

Número	Componente
1	Entrada Ethernet IEEE-802.3-2002
2	Conector para tarjeta microSD
3	Línea de salida para mini-jack
4	Línea de entrada estéreo para mini-jack
5	Pulsador
6	Pulsador RESET
7	Conector para cámara
8	USB OTG HS con conectores Micro-AB
9	USB OTG FS con conectores Micro-AB
10	Puerto debug
11	Alimentación de corriente a 5V
12	Conectores V3 de Arduino UNO
13	STM32F7
14	LCD-TFT de 4.3 pulgadas con pantalla táctil capacitiva
15	Dos micrófonos ST MEMS

A los anteriores componentes se le añaden las siguientes funciones:

- Funciones del USB: Puerto virtual (COM), almacenamiento masivo, puerto para el debug.
- SAI Audio codec.
- Conector de entrada para SPDIF RCA.
- Memoria flash Quad-SPI de 128Mb.
- Conector RF-EEPROM para extensions.
- Cinco opciones de alimentación: ST LINK/V2-1, conectores USB FS o USB HS, conector VIN desde Arduino y conector externo de 5V.

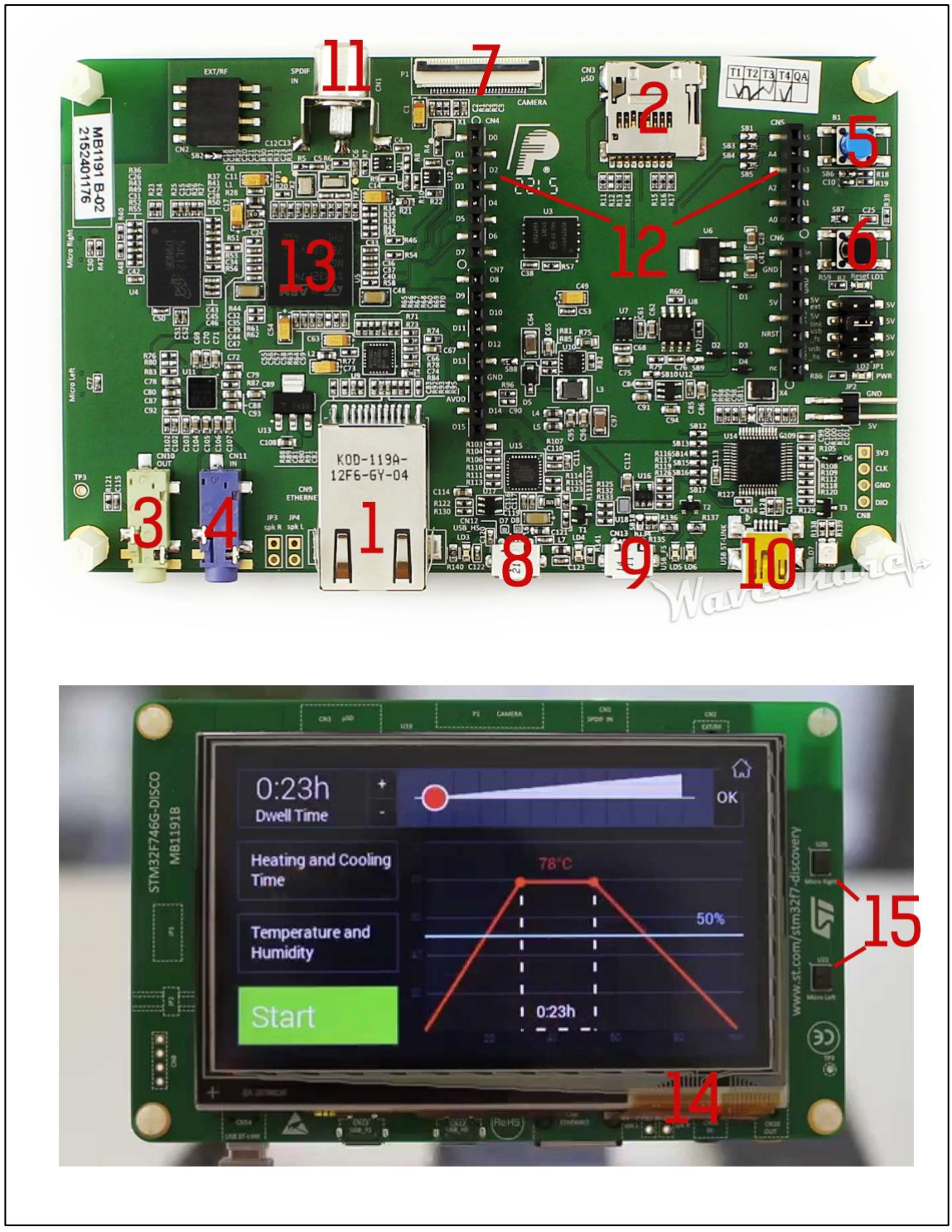

Figura 2-11. Kit STM32F7 Discovery.[11]

[11] Embedded Wizard y eBay

3.1.2 Pinout

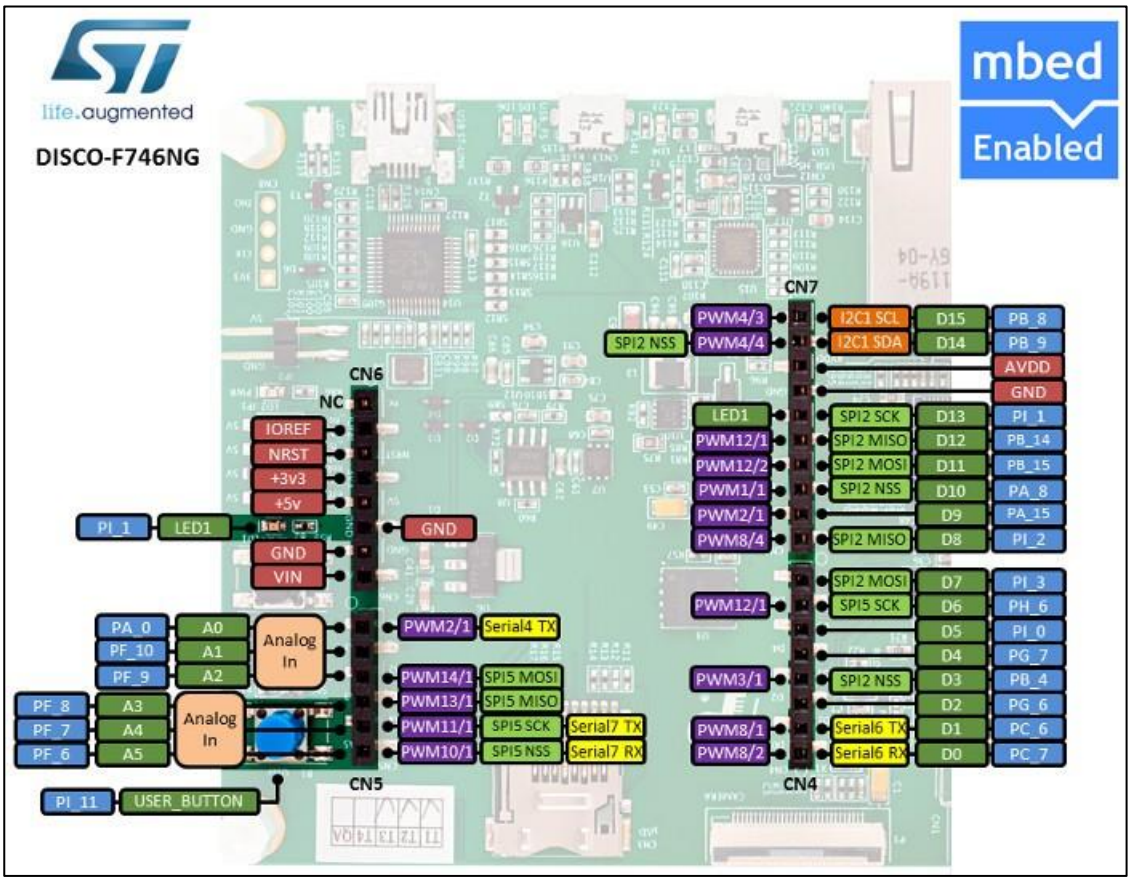

Figura 2-12. Pinout de la placa.[12]

3.1.2.1 Información adicional sobre algunos periféricos no usados en este trabajo

Para facilitar el trabajo de cualquiera que retome estos primeros pasos con el STM32F7, en este punto se recogerán los pines a los que se encuentran conectados algunos periféricos que no son usados en ninguno de los proyectos. La información resumida a continuación viene en los esquemáticos que la compañía facilita. Son de difícil lectura pues la cantidad de pines es alta, así pues, está información puede resultar muy útil:

- El botón de usuario está conectado al pin PI11.
- El LED1 está conetado al pin PI1
- El conector EXT/RF E2P se encuentra conectado a los pines PG3 (EXT_RST), PB8 (EXT_SCL) y PB9 (EXT_SDA).
- El lector de micro SD se encuentra conectado a los pines PD2 (uSD_CMD), PC8 (uSD_D0), PC9 (uSD_D1), PC10 (uSD_D2), PC11 (uSD_D3), PC12 (uSD_CLK) y PC13 (uSD_Detect).
- El conector micro USB OTG FS se encuentra conectado a los pines PD5 (OTG_FS_PowerSwitchOn), PD4 (OTG_FS_OverCurrent), PA9 (OTG_FS_VBUS), PA10 (OTG_FS_ID), PA11 (OTG_FS_N) y PA12 (OTG_FS_P)

[12] Fografía extraída de la página web oficial de STMicroelectronics.

- El conector micro USB OTG HS PHY se encuentra conectado a los pines PE3 (OTG_HS_OverCurrent), PH5 (ULPI_NXT), PC2 (ULPI_DIR), PC0 (ULPI_STP), PA5 (ULPI_CK), PB0 (ULPI_D1), PB1 (ULPI_D2), PB10 (ULPI_D3), PB11 (ULPI_D4), PB12 (ULPI_D5), PB13 (ULPI_D6) y PB5 (ULPI_D7).

- En cuanto a los conectores de Arduino UNO, al tener un gran número de pines, queda mejor detallado con una imagen:

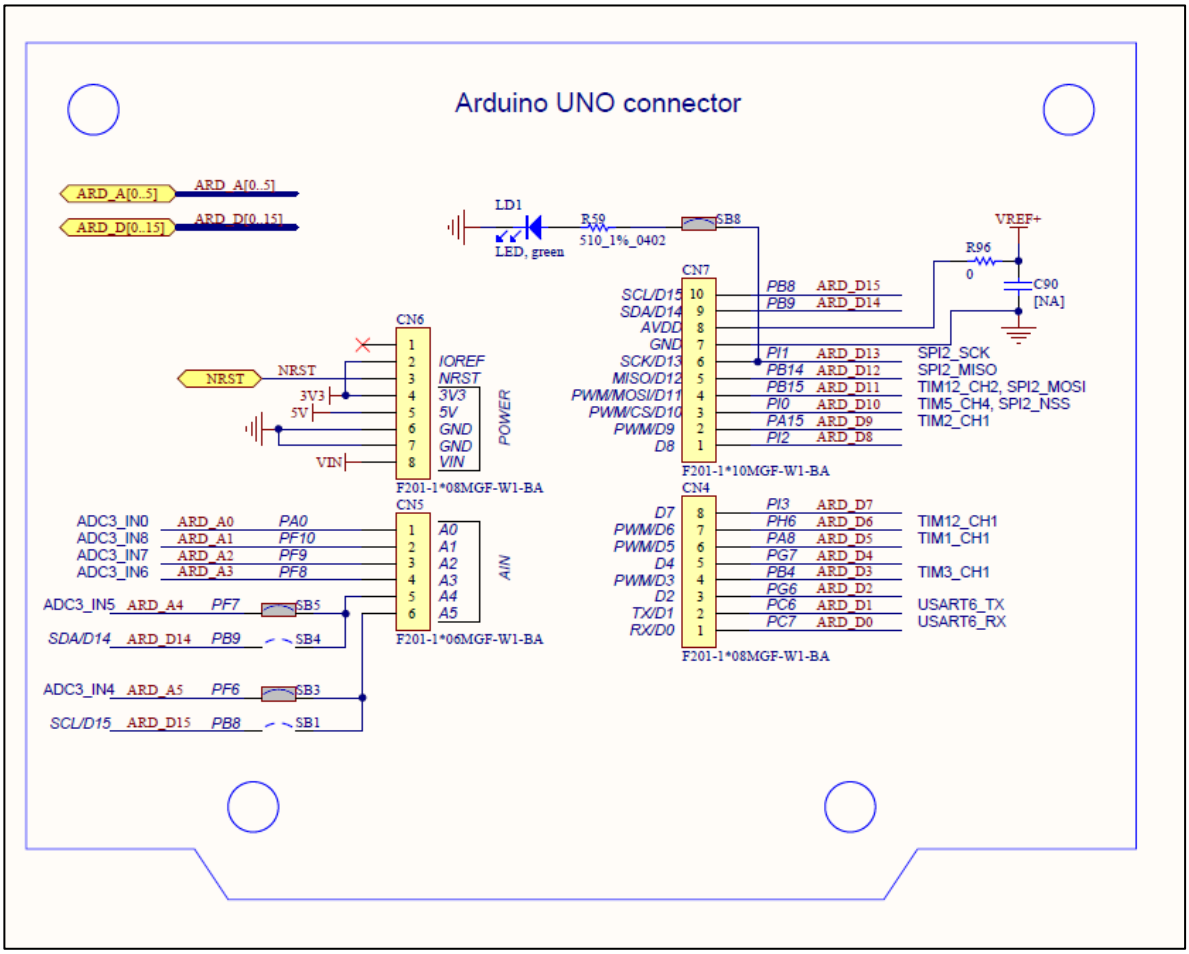

Figura 2-13. Conexiones del conector para Arduino UNO.[13]

Hay otros periféricos, como el Ethernet, la entrada y salida de audio o la pantalla LCD (que aunque se usa no está de más detallar a qué pines está conectada) que son interesantes para continuar el trabajo, pero debido a su gran complejidad y número de conectores, es más sencillo acudir al documento donde aparecen los esquemáticos que leer un resumen. Es por ello por lo que si se desea conocer las conexiones, es necesario consultar el manual de esquemáticos.

[13] Manual de esquemáticos de la placa.

4. ENTORNO DE TRABAJO PARA EL KIT STM32F7 DISCOVERY

Configuración del ordenador para la programación del kit

Software, hardware y drivers necesarios

Como para la programación de cualquier microcontrolador, el kit STM32F7 Discovery también dispone de un entorno de trabajo que es el mismo que para los antecesores de la misma gama. La página web de STMicroelectronics ofrece en la página de cada uno de los microcontroladores una lista de todo lo necesario para comenzar a programar.

4.1 Hardware

Para la programación básica del kit son necesarios:

- Kit STM32F7 Discovery.
- Cable miniUSB.
- PC.

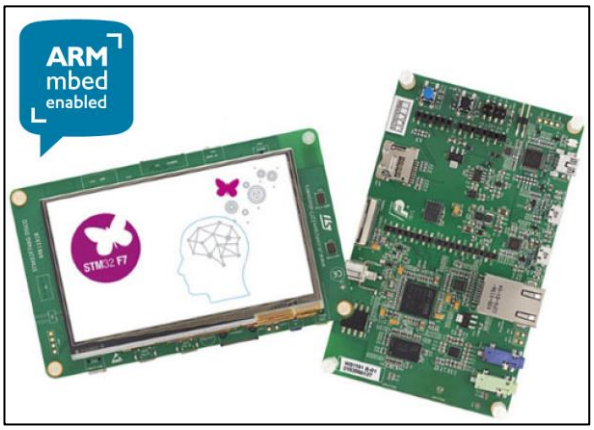

Figura 4-14. Kit STM32F7 Discovery.[14]

4.2 Drivers y paquetes

Son totalmente necesarios para conseguir cargar cualquier programa. Son gratuitos y de libre acceso con una cuenta en la web de STMicroelectronics y su instalación es como la de cualquier otro programa o driver.

4.2.1 STM32 ST-LINK Utility

Se trata de una herramienta que instala los drivers de la placa a la vez que permite leer, escribir y verificar la memoria del dispositivo. La herramienta otorga la posibilidad de programar las memorias internas y verificar el programa que se encuentra corriendo en el dispositivo. Además, permite automatizar la programación. Sus características más importantes son:

- Es gratuito.
- Soporta Motorola S19, Intel HEX y formatos binarios.
- Carga, edita y guarda ejetuables y ficheros de datos generados con compiladores en C o ensamblador.
- Permite la eliminación, programación, vista y verificación del contenido de la memoria flash.
- Automatiza la programación de la gama STM32.
- Programa de una vez la memoria programable.
- Oferta una interfaz de líneas de comando.
- Compara archivos con la memoria de la placa.
- Permite ver el estado de la memoria y el núcleo en tiempo real.

La última versión se puede encontrar en:

http://www.st.com/en/embedded-software/stsw-link004.html

[14] Fografía extraída de la página web oficial de STMicroelectronics.

4.2.2 STM32 Virtual COM Port Driver

Como su propio nombre indica se trata de los drivers del puerto virtual. Permite ver y enviar datos a la placa desde la pantalla usando un software de creación de puertos virtuales que se conecta al microcontrolador usando la UART de este. No es totalmente necesario para la programación del kit, pero si resulta necesario para la resolución de problemas que requieren el uso de la UART. Es una alternativa gratuita a la adquisición de un módulo bluetooth o de cualquier dispositivo que permita el envío y recepción de datos mediante dicho periférico.

La última versión tanto para 32, como para 64 bits se puede encontrar en:

http://www.st.com/en/development-tools/stsw-stm32102.html

4.2.3 Paquete Java SE Runtime Enviroment 8

Totalmente gratuito, fácil de encontrar en la página web de Oracle:

http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html

4.2.4 Paquete .NET Framework 4.5

Al igual que el paquete de Java, está disponible gratis en la página de Microsoft:

https://www.microsoft.com/en-us/download/details.aspx?id=30653

4.3 Software

STMicroelectronics ofrece dos posibildades, dos caminos para elegir: El primero de ellos se trata de un generador de código que inicializa todos los periféricos que le indiquemos, el nombre del software es STM32CubeMx y es gratuito. Antes de generar el código nos pide que indiquemos el compilador que vamos a utilizar:

- EWARM.
- MDK-ARM V5.
- MDK-ARM V4.
- TrueSTUDIO.
- SW4STM32.

Así pues, el uso de la interfaz de generación de código queda limitada a cuatro compiladores. Si se desea realizar la programación con otro compilador, la inicialización de los registros de control debe de hacerse a mano, haciendo el trabajo muchísimo más tedioso, largo y aburrido.

Puesto que permite cargar programas de hasta 32K y crear breakpoints mientras corre el programa, decidí utilizar MDK-ARM V5. Con la limitación indicada, es gratuito y se puede encontrar en la web del desarrollador:

http://www2.keil.com/mdk5/

Antes de elegir MDK-ARM V5 surgió la opción de trabajar con EWARM o TrueSTUDIO. De hecho fueron testeados, pero la interfaz y los paneles de navegación de MDK-ARM V5 me parecieron más sencillos y fáciles de usar.

La segunda opción consiste en un compilador online que permite crear programas para el microcontrolador, pero carece del uso de breakpoints, lo que puede llegar a ser una locura cuando se usan varios periféricos, con sus debidas interrupciones y demás impedimentos que pueden provocar horas de programación buscando errores, cuando un breakpoint puede solucionarlo en minutos. Así pues, esa opción quedó descartada desde el principio.

4.3.1 STM32CubeMx

Se trata de una herramienta de generación de código a partir de especificaciones para todas y cada una las placas de la familia STM32. Es algo parecido a Grace, la herramienta que Code Composer Studio posee para la programación de las placas de la familia MSP y que ya conocemos a través de las diferentes prácticas.

Es una interfaz sencilla, facil de utilizar, que permite configurar cualquier periférico del núcleo que estamos programando.

Aunque el software permite generar un código para cualquier periférico del núcleo, eso no significa que pueda ser usado con el kit, ya que muestra todos los pines configurables para el núcleo que lleva incorporado y no para la placa en sí, es decir, que permite configurar pines que quizás no sean accesibles con el producto. Así pues, es totalmente necesario usar el programa junto con un esquema con el pinout de la placa o con la certeza absoluta de que se conocen los pines accesibles con el kit.

Para comenzar a trabajar con el software es necesario que esté instalado en el ordenador:

http://www.st.com/en/development-tools/stm32cubemx.html?s_searchtype=partnumber

Una vez instalado y abierto, hay que crear un nuevo proyecto:

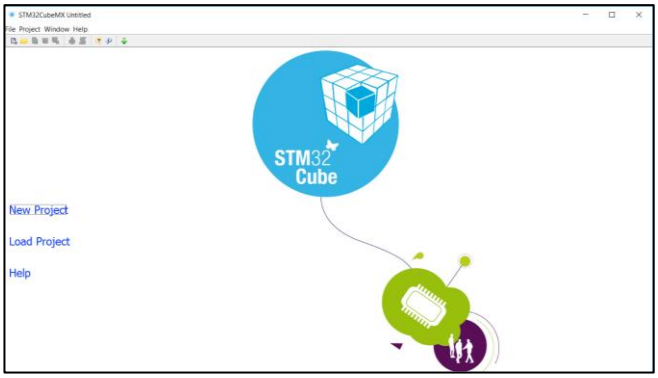

Figura 4-15. Creación de un nuevo proyecto STM32CubeMx.[15]

Una vez se abra la pantalla de elección de placa o núcleo, seleccionamos el producto que se está usando:

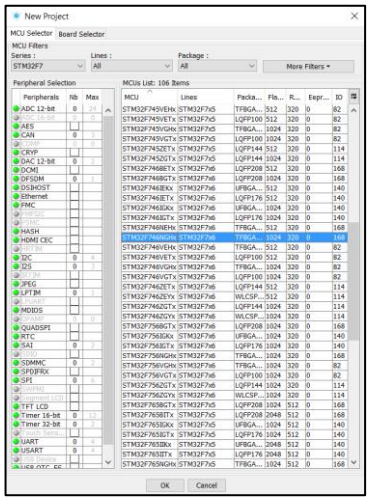

Figura 4-16. Elección de microcontrolador en STM32CubeMx.[16]

[15] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[16] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

Una vez creado el proyecto, se abrirá la siguiente pantalla:

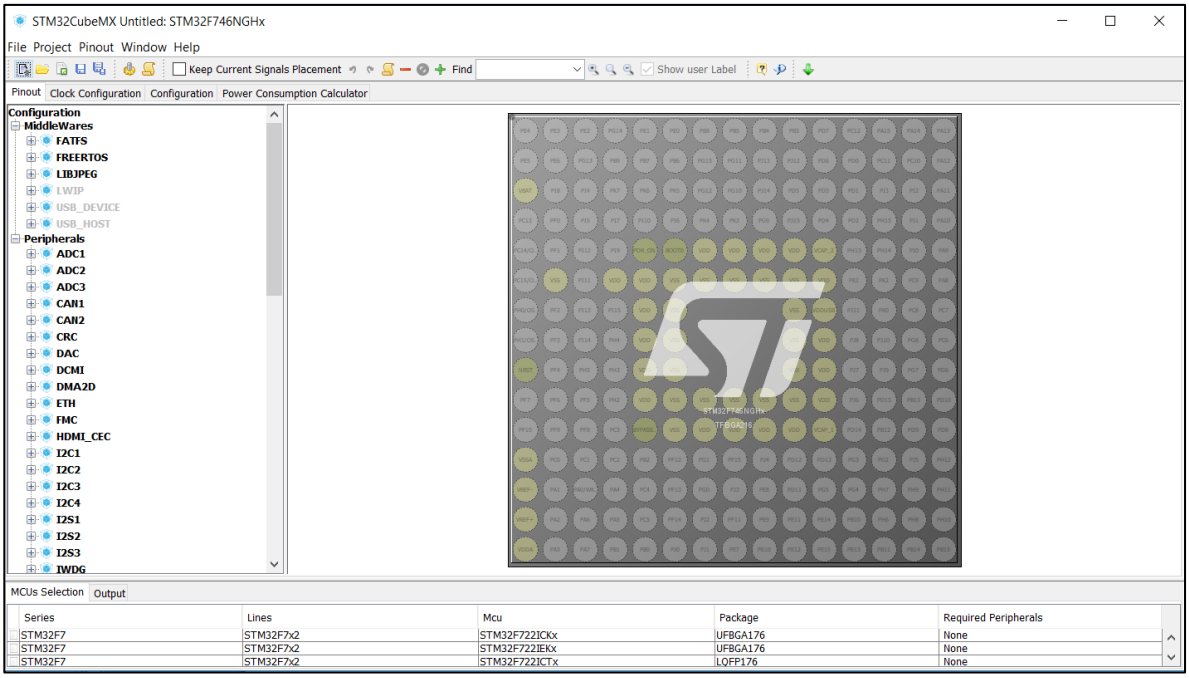

Figura 4-17. Pantalla principal de STM32CubeMx.[17]

En la columna de la izquierda es posible seleccionar cualquiera de los periféricos, activarlo y configurarlo. Del mismo modo, usando el buscador situado en la parte superior de la interfaz, es posible buscar un pin en cuestión. Al escribir el nombre de este y pulsar intro, comenzará a parpadear el circulo correspondiente al pin. Si se pulsa sobre él, aparecen las diferentes opciones disponibles para el pin seleccionado. Así pues, existen dos formas de configurar un periférico: Seleccionándolo en la columna de la izquierda o seleccionando individualemnte cada uno de los pines que se ven envuelto en el periférico.

En la pestaña Clock Configuration, aparecen las opciones del reloj del sistema:

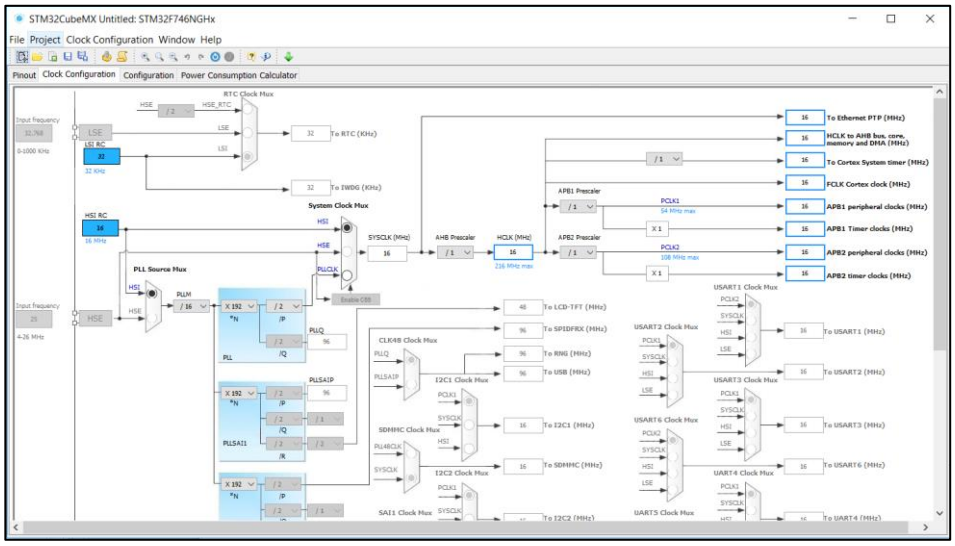

Figura 4-18. Configuración del reloj usando STM32CubeMx.[18]

[17] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[18] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

La pantalla consta de un diagrama de bloques en los que se pueden modificar los valores de los relojes del microcontrolador a gusto del usuario.

Finalmente, en la pestaña Configuration se abren las opciones del sistema. En esta pantalla aparecen las opciones de cada uno de los periféricos seleccionados así como los del núcleo. En esta pantalla es posible configurar lo que se vaya a usar de manera más profunda y especifica:

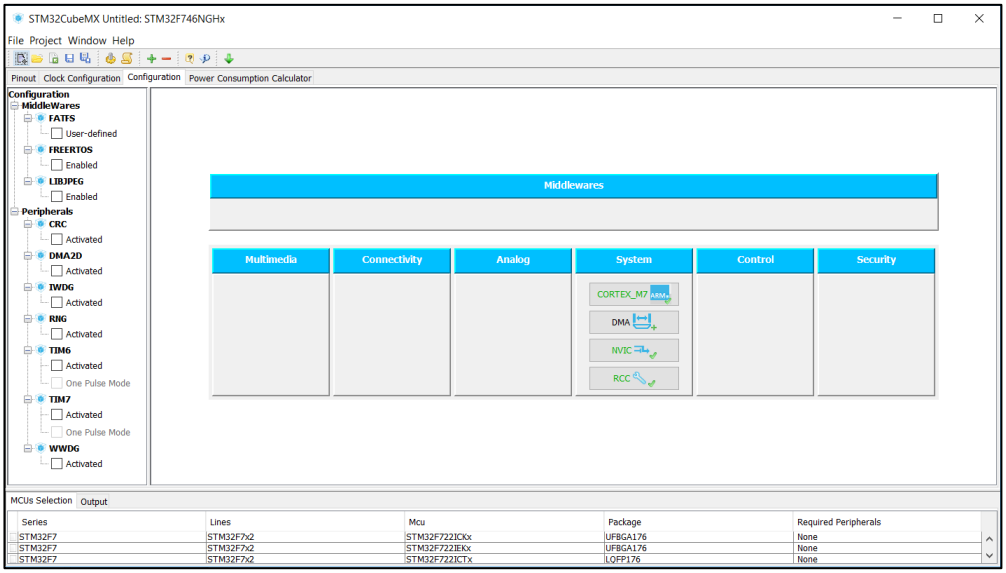

Figura 4-19. Pantalla de configuración de STM32CubeMx.[19]

El botón de CORTEX_M7 permite la configuración del núcleo, NVIC la de las interrupciones, RCC la del reloj y DMA la de la DMA.

Una vez terminada la configuración del proyecto, es necesario generarlos. Para ello, se ha de hacer click sobre Project/Generate Code además de introducir la localización del proyecto, así como el nombre y el compilador:

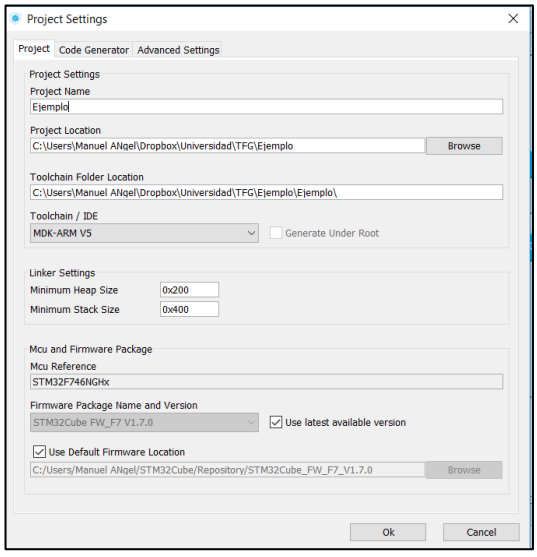

Figura 4-20. Generación del código de un proyecto de STM32CubeMx.[20]

[19] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[20] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

Una vez pulsamos Ok, el programa comienza a generar el código y finalmente da la opción de abrir el proyecto:

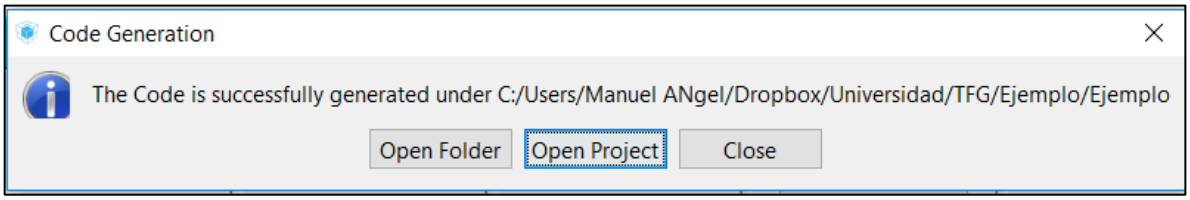

Figura 4-21. Apertura de un proyecto generado en STM32CubeMx.[21]

Al pulsar sobre Open Project, se nos abre la herramienta de programación:

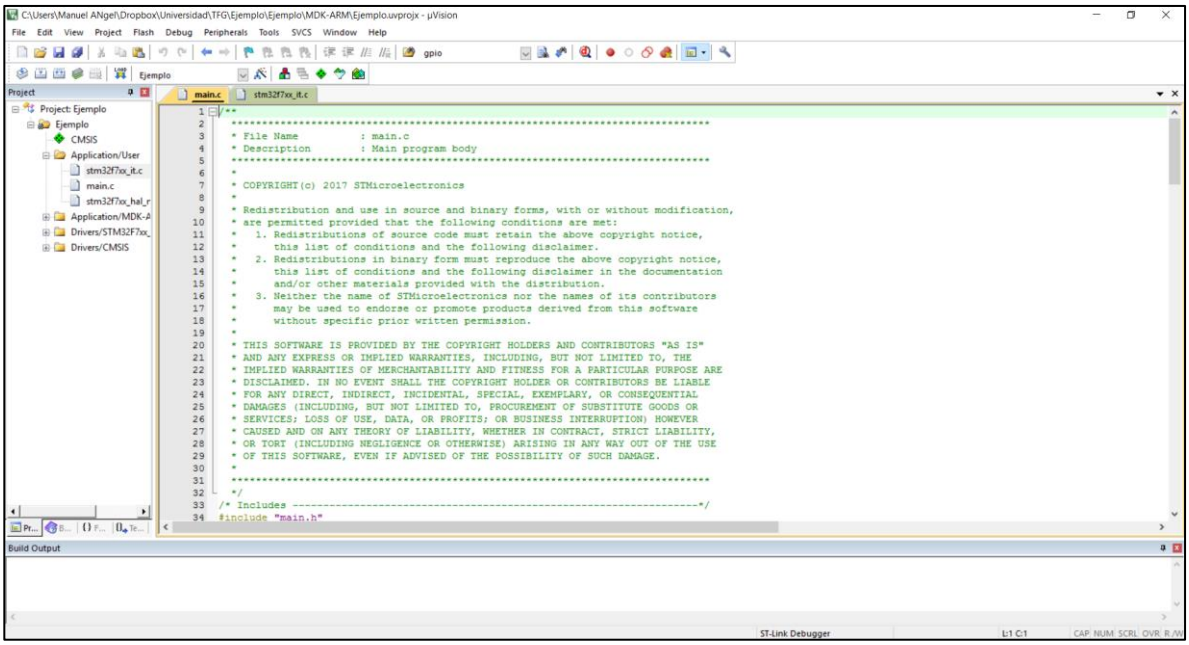

Figura 4-22. Pantalla principal Keil µVision5.[22]

[21] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[22] Imagen adquirida mediante impresión de pantalla durante el uso del programa MDK-ARM V5: Keil µVision5.

4.3.2 MDK-ARM V5: Keil µVision5

Se trata de una herramienta potente de trabajo que permite programar el código generado por el STM32CubeMx, compilarlo y cargarlo en la placa, teniendo la opción de utilizar un visualizador para comprobar cómo cambian las variables, así como el uso de breakpoints.

La gran desventaja que tiene es que para la versión gratuita del programa existe una limitación de 32Kbytes para cualquier programa que se quiera cargar en la placa. No obstante, para el nivel que alcanza este trabajo, es más que suficiente.

Cada vez que se genera un proyecto es necesario reconstruir todos los archivos, así que hay que pulsar Project/Rebuild all Target files. Lleva un poco de tiempo, aunque acorde con el tamaño o complejidad del proyecto.

Por lo demás, el uso del compilador es el mismo o similar que Code Composer Studio, TrueSTUDIO o EWARM, por lo que supongo que el lector de este trabajo tiene el nivel suficiente como para no necesitar indicaciones para construir, cargar programas, crear breakpoints u otras tareas básicas de programación de microcontroladores.

4.3.3 Tera Term

Es un software gratuito que se usa para la generación de puertos virutales para la connexion puerto serie con la placa:

https://osdn.net/projects/ttssh2/releases/

5. PROGRAMACIÓN DE PERIFÉRICOS

Configuración y programación de periféricos

Primeros pasos con el STM32F7

El objetivo de este trabajo de fin de grado es puramente lectivo, por lo que este capítulo engloba la mayor parte del total. Aunque se ha hecho una breve introducción teórica, podemos decir que este es el corazón del trabajo de fin de grado. Supone el resultado de un primer enfrentamiento con un producto que nadie en la ETSI ha usado aun, por lo que ha supuesto también el mayor número de horas de trabajo y quebraderos de cabeza.

Durante las primeras reuniones con el tutor, me dejó claro que su intención era que yo consiguiese programar los periféricos más básicos del kit, hasta llegar al uso de la pantalla para que, en unos años, él o algún compañero lo puedan usar en una asignatura impartida por el departamento.

Junto al tutor establecí unos objetivos que marcan cada uno de los periféricos que se han investigado y programado. Cada apartado sigue el mismo patrón:

- Breve descripción del periférico.
- Librería del periférico.
- Configuración del periférico
- Formulación de un problema que englobe el periférico en cuestión.
- Resolución del problema: Generación paso a paso del proyecto con STM32CubeMx y programación con MDK-ARM V5.
- Código (en el anexo).

5.1 Programación de GPIO

5.1.1 Descripción del periférico

El producto tiene 24 pines in/out accesibles con salida a 3.3V, de los cuales, dos de ellos no son utilizables debido a que se encuentran conectados a un LED y a un interruptor de usuario.

5.1.2 Librería del periférico

Para la utilización de los GPIOs se ha utilizado una de las librerías incluidas en la memoria de la placa. Su nombre es stm32f7xx_hal_gpio.c y consta de las siguientes funciones:

- **HAL_GPIO_DeInit(GPIO_TypeDef *GPIOx, uint32_t GPIO_Pin)**: Permite hacer una reinicialización del GPIO deseado a los valores por defecto.
- **HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)**: Esta función permite conocer el cuándo se está en la línea de código del EXTI.
- **HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)**: Controla la interrupción del EXTI
- **HAL_GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_Init)**: Realiza la inicialización del GPIO de forma acorde con los parámetros especificados.
- **HAL_GPIO_LockPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)**: Bloquea los registros de configuración de un GPIO.
- **HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)**: Se encarga de hacer una lectura de un pin declarado como input. Envía de vuelta un 0 si no está recibiendo nada y un 1 si lo hace.
- **HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)**: Alterna el pin especificado.
- **HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)**: Permite activar o desactivar la salida de un pin declarado como *output*.

5.1.3 Configuración del periférico

Para la configuración del periférico se ha de usar un software que nos proporciona la compañía. Su nombre es *STM32CubeMX* y es parecido al plugin llamado *Grace* incluido en *Code Composer Studio* para la programación de algunas placas de la compañía *Texas Instrument*.

Para la configuración del GPIO es primordial el *pinout*, donde es posible buscar los pines que se desea usar para después, con el software, seleccionar la funcionalidad del pin en cuestión. El software crea automáticamente un proyecto que incluye todas las funciones y configuraciones necesarias para que este cumpla los requisitos marcados en la interfaz de configuración.

Aunque como ya he dicho, el software hace todo el trabajo sucio, voy a explicar las funciones que se deben de llamar en el programa principal para que este configure los pines deseados como GPIO (input u output).

Al comienzo del main del proyecto vemos que quedan declaradas las funciones que inicializan cada uno de los periféricos configurados con STM32CubeMX:

```
/* Private function prototypes -----------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX GPIO Init(void);
```

Todas las funciones son llamadas dentro del programa principal para que se configure al principio de la ejecución de este. En el caso de la función del GPIO, que es la que nos interesa, se realiza un proceso idéntico para cada uno de los pines:

1. Inicialización del reloj de los puertos de los GPIOs para cada base utilizada.
2. Configuración de los GPIOs como de salida.
3. Configuración del modo, velocidad… para cada GPIO precedido de la inicialización de este.

Para un ejemplo en el que se han configurado como entrada el pin PI11 y como salidas PI0, PI2, PI3 y PH6:

```
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOI_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOI,GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_0, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, GPIO_PIN_RESET);

  /*Configure GPIO pins : PI3 PI2 PI0 */
  GPIO_InitStruct.Pin = GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);

  /*Configure GPIO pin : PI11 */
  GPIO_InitStruct.Pin = GPIO_PIN_11;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);

  /*Configure GPIO pin : PH6 */
  GPIO_InitStruct.Pin = GPIO_PIN_6;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOH, &GPIO_InitStruct);
}
```

Como podemos observar, el pin de reset es configurado automáticamente como de salida. Se trata de una configuración impuesta por defecto.

5.1.4 Ejemplo práctico

Utilice el pinout de la STM32F7 Discovery y las librerías necesarias para solucionar el siguiente problema: Utilizando cuatro LEDs (conectados en los pines PI0, PI2, PI3 y PI4) y el pulsador de la placa conectado en el pin PI11, elabore un programa que permita, al presionar el pulsador, encender los LEDs de forma que se vayan iluminando de manera escalonada cada vez que pulsamos y, una vez que estén todos encendidos, se comiencen a apagar de escalonadamente, pero en sentido inverso.

5.1.4.1 Generación del proyecto con STM32CubeMx

Creamos un nuevo proyecto, seleccionamos la placa que estamos utilizando y nos aparecerá la pantalla de selección de pines.

Una vez ahí, se comienza a configurar los pines y todo lo necesario para resolver el ejemplo. Para empezar, lo primero que se configura es el reloj. En la pestaña de la izquierda se ha de abrir el desplegable que dice RCC y en High Speed Clock (HSE), seleccionar la opción Crystal/Ceramic Resonator:

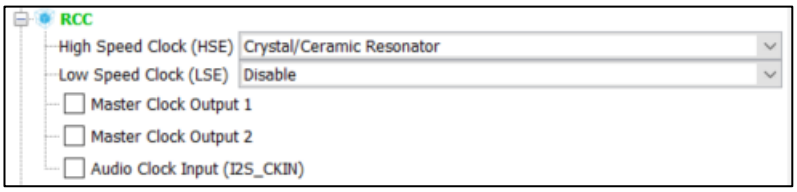

Figura 5-23. Configuración del reloj cerámico con STM32CubeMx.[23]

Una vez seleccionado el reloj hay que configurar la velocidad a la que queremos que funcione. En las pestañas que se encuentran en la parte superior de la pantalla encontramos Clock Configuration. Se nos abrirá una pantalla nueva en la que tenemos que buscar un bloque que nos permite cambiar la velocidad del reloj. El nombre es HCLK (MHz) y suele estar en el centro del diagrama de bloques. Una vez localizado, es posible seleccionar la velocidad deseada, en mi caso el máximo, 216MHz:

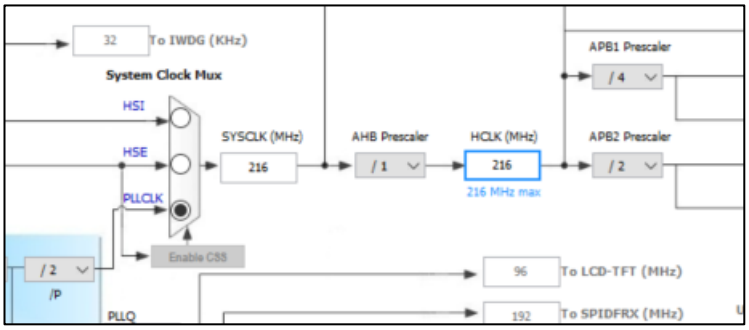

Figura 5-24. Configuración de la frecuencia del reloj cerámico con STM32CubeMx.[24]

[23] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[24] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

Es muy importante pulsar la tecla enter del teclado al escribir la velocidad del reloj para que el programa realice las operaciones necesarias para la configuración del proyecto.

Volviendo a la pantalla inicial pulsando la pestaña pinout continuamos con la configuración del proyecto. Para que sea posible hacer un debug correcto tenemos que configurar en una de las pestañas de la izquierda el modo de debug que se desea. Así pues, en el desplegable Debug de la pestaña SYS se ha de seleccionar el modo Serial Wire:

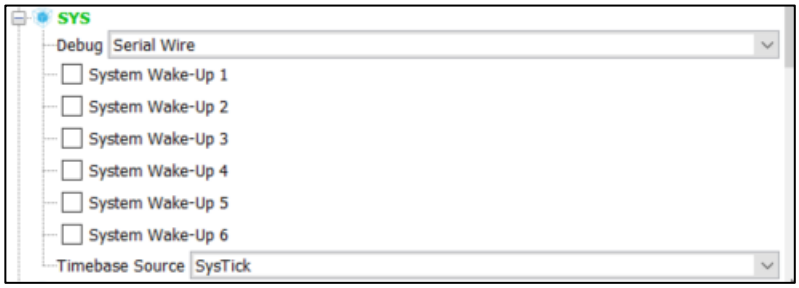

Figura 5-25. Configuración del tipo de debug en STM32CubeMx.[25]

Como es lógico, también se han de configurar los pines para los LEDs, así como para el pulsador, necesarios. En la parte superior de la interfaz encontramos un buscador en la que podemos buscar el pin deseado. Si escribimos el nombre del pin, por ejemplo PI1, y pulsamos enter, el pin en cuestión comenzará a parpadear en la imagen central:

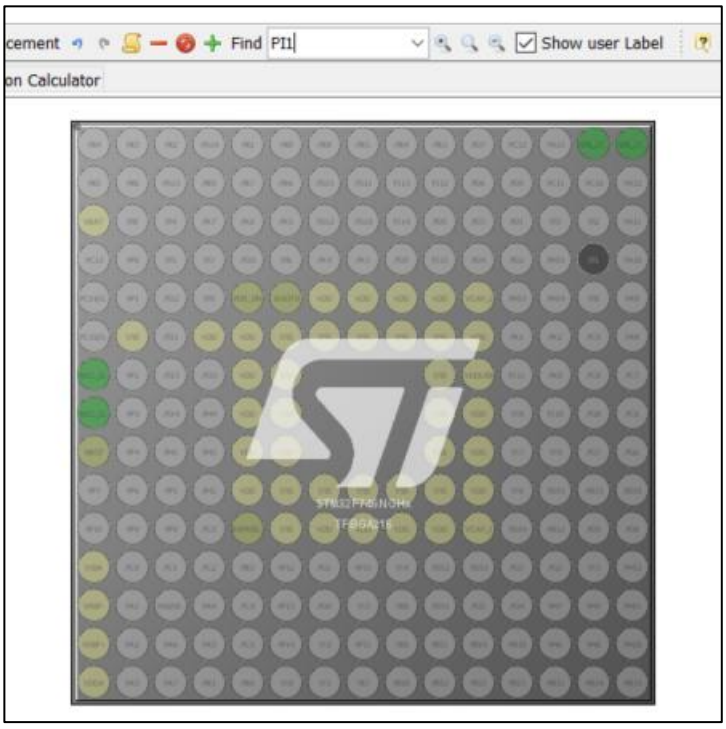

Figura 5-26. Configuración de un GPIO en STM32CubeMx.[26]

[25] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[26] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

Una vez encontrado la pata del micro que corresponde al pin, seleccionamos su función haciendo click izquierdo con nuestro ratón. Aparecerá un desplegable en el que se ha de seleccionar la funcionalidad, en nuestro caso GPIO_Input o GPIO_Output:

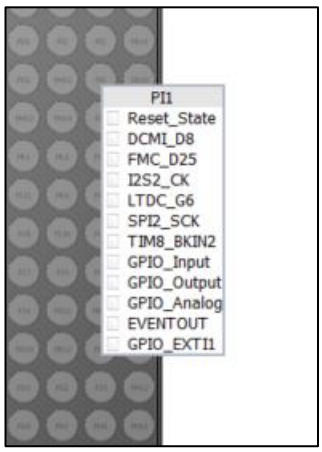

Figura 5-27. Selección de la función de un pin en STM32CubeMx.[27]

Repetimos el proceso para los cuatro pines restantes, recordando que 4 de ellos deben de ser de salida y uno de ellos de entrada.

Finalmente es necesario configurar algunos ajustes del micro así como las interrupciones. Haciendo click en la pestaña "Configuration" nos aparecerá una tabla. En el botón que dice CORTEX_M7 se han de activar el ART Accelerator, el Instruction Prefetch, el CPU ICache y el CPU DCache:

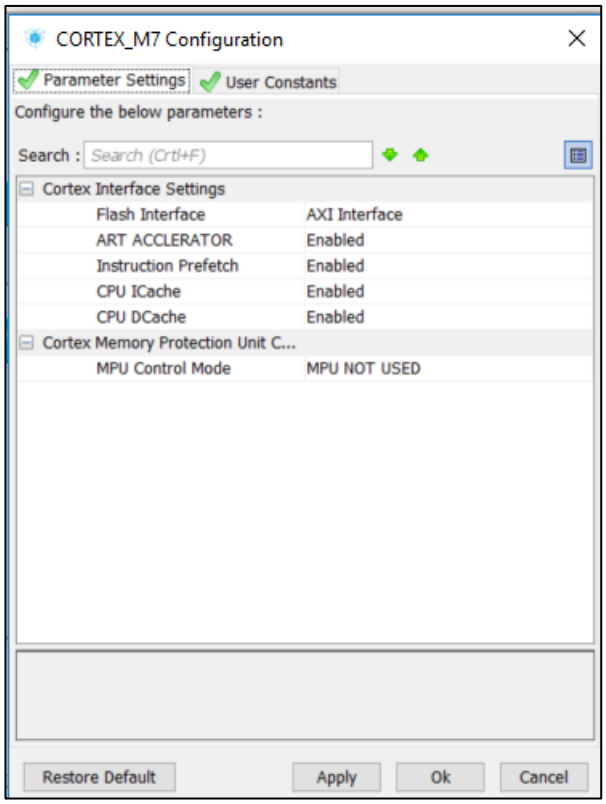

Figura 5-28. Configuración del núcleo en STM32CubeMx.[28]

[27] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[28] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

En el botón que dice NVIC se pueden configurar las interrupciones, de las cuales nos interesa la global del reloj, por lo que la opción correcta es en RCC Global Interrupt:

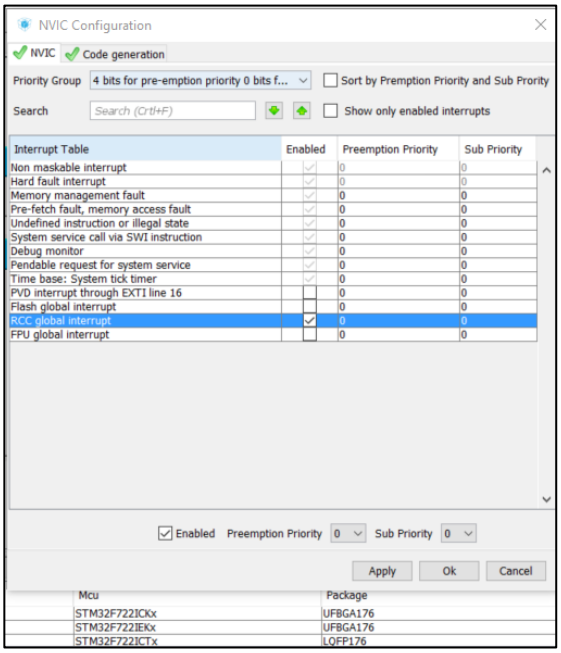

Figura 5-29. Configuración de las interrupciones en STM32CubeMx.[29]

Para terminar, se ha de guardar y generar el proyecto seleccionando como Toolchain la herramienta que estéis utilizando, en mi caso, MDK-ARM V5. Al confirmar, se nos abrirá el Toolchain seleccionado.

Siguiendo los pasos anteriores se nos habrá generado un proyecto que tiene configurado todo lo necesario para poder resolver el problema sin ningún tipo de problemas.

Así pues, en modo resumen, los pasos realizados hasta ahora han sido:

- Configurado el reloj del microcontrolador.
- Configurado el modo de debug del microcontrolador.
- Configurados los pines de entrada/salida.
- Activadas varias opciones del micro.
- Activada la interrupción global del reloj.
- Generado el proyecto.
- Abierto el Toolchain, listo para comenzar a trabajar.

[29] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

5.1.4.2 Resolución del problema con MDK-ARM V5

Al ser un problema diseñado para aprender a manejar los GPIOs de la placa, es lógico que para usarlos el usuario deba buscar en las librerías que manejan los GPIOs para así encontrar la forma de hacer una lectura y escritura. Las librerías se pueden encontrar en la documentación del micro, así como en el MDK-ARM V5 o en el punto 5.1.2 de este trabajo.

Como se puede observar, existe una función de lectura y otra de escritura, a la que tan solo hay que especificarle el número y letra del pin requerido y que se supone configurado previamente. Así pues, para facilitar la programación y lectura del código, en los *define* del programa principal main.c se encuentran definidas palabras tanto para encender y a apagar los LEDs cómo para detectar si se ha pulsado el botón o no:

```
/* Para facilitar la lectura y escritura del codigo, creo varios defines para el encendido/apagado de LEDs*/
#define LED1ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_0, 1)
#define LED2ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_2, 1)
#define LED3ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3, 1)
#define LED4ON HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, 1)
#define LED1OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_0, 0)
#define LED2OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_2, 0)
#define LED3OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3, 0)
#define LED4OFF HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, 0)
#define UP HAL_GPIO_ReadPin(GPIOI, GPIO_PIN_11)
```

Una vez realizado esto, el resto del programa se trata de una máquina de estados sencilla que se encarga de cambiar de estado tras una pulsación y que, cuando llega al final, invierte su sentido. Todo incluido en el típico bucle sin fin de los microcontroladores.

5.2 Programación de un Timer

5.2.1 Descripción del periférico

El producto dispone de 14 timers que se pueden clasificar configurables a diferentes niveles:

- Timers de configuración avanzada (6 canales): TIM1 y TIM8.
- Timers de configuración para propósito general (4 canales): TIM2, TIM3, TIM4 y TIM5.
- Timers de configuración lite (2 canales): TIM9 y TIM12.
- Timers de configuración básica (1 canal): TIM6, TIM7, TIM10, TIM11, TIM13 yTIM14.

5.2.2 Librería del periférico

Para la utilización de los Timers se ha utilizado una de las librerías incluidas en la memoria de la placa. Su nombre es stm32f7xx_hal_tim.c y sus funciones más importantes son:

- **HAL_TIM_Base_DeInit(TIM_HandleTypeDef *htim)**: Permite hacer una reinicialización del Timer deseado a los valores por defecto.
- **HAL_TIM_Base_GetState(TIM_HandleTypeDef *htim)**: Devuelve un valor que indica el estado del Timer.
- **HAL_TIM_Base_Init(TIM_HandleTypeDef *htim)**: Realiza la inicialización del Timer según los valores especificados.
- **HAL_TIM_Base_Stop_IT(TIM_HandleTypeDef *htim)**: Para el Timer en modo interrupción.
- **HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)**: Inicia el Timer en modo interrupción.

5.2.3 Configuración del periférico

Al igual que para el ejemplo del GPIO, se utilizará el software STM32CubeMx para la configuración del Timer.

Debido a que hay más de un Timer disponible, para configurarlo debemos buscar en las características de la placa aquel Timer que se adecua a las necesidades de nuestro problema. Una vez seleccionado, se ha de configurar su interrupción para que salte cada vez que se cumpla un ciclo.

Del mismo modo que en el caso anterior, STM32CubeMX crea todas las funciones necesarias para la configuración del Timer. No obstante, para entender completamente la configuración del periférico, utilizaré las funciones creadas para conocer cómo sería la la configuración manual de un Timer.

Al comienzo del main del proyecto vemos que quedan declaradas las funciones que inicializan cada uno de los periféricos configurados con STM32CubeMX:

```
/* Private function prototypes -----------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_TIM6_Init(void);
```

Tal y como observamos existe una función de inicialización del Timer, que será en la que se indicarán los valores de escalado, contadores, periodos…

```
/* TIM6 init function */
static void MX_TIM6_Init(void)
{

  TIM_MasterConfigTypeDef sMasterConfig;

  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 25000;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 350;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
}
```

La función de inicialización es llamada en el programa para configurar el periférico desde el inicio. De este modo, cada vez que se llama a la función se realiza la configuración de:

1. Selección del Timer que se usa.
2. Configuración del preescalado.
3. Modo de conteo.
4. Periodo.
5. Comprobación de errores.

5.2.4 Ejemplo práctico

El objetivo de este ejemplo es comprender el uso de las funciones básicas de manejo de los Timers de la STM32F7 Discovery, utilizando las librerías incluidas en la propia memoria de la placa.

5.2.4.1 Generación del proyecto con STM32CubeMx

Partiendo del proyecto del ejemplo anterior, en el que teníamos 4 pines de salida (PI0, PI2, PI3 y PH6) y uno de entrada (PI11) con el reloj cerámico a 216MHz y el debug configurado. Vamos a reconfigurarlo añadiendo opciones para ajustarlo a nuestra necesidad.

Para empezar, se ha de reducir el reloj cerámico a 50Mhz para así decrementar el trabajo del micro:

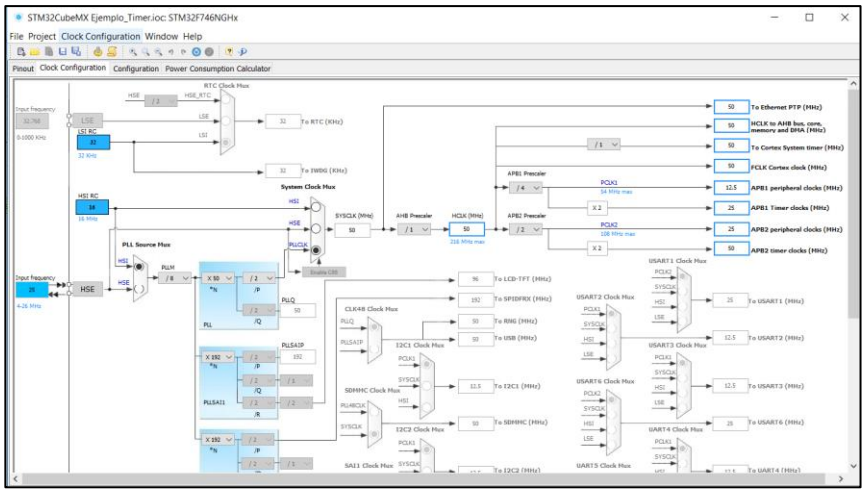

Figura 5-30. Configuración de la freuencia del reloj cerámico con STM32CubeMx.[30]

Una vez reconfigurado el reloj, se activa el timer número 6 buscando dicho periférico en la pestaña del pinout bajo el nombre de TIM6:

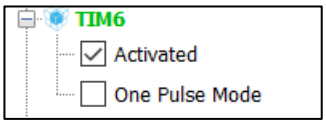

Figura 5-31. Activación de un timer con STM32CubeMx.[31]

Una vez configurado el timer, en la pestaña de configuración, se hará lo mismo que con los vectores de interrupción, tanto del reloj general, como el del nuevo timer. Así pues, haciendo click sobre NVIC se ha de seleccionar la interrupción del timer 6 y deseleccionar la del reloj RCC.

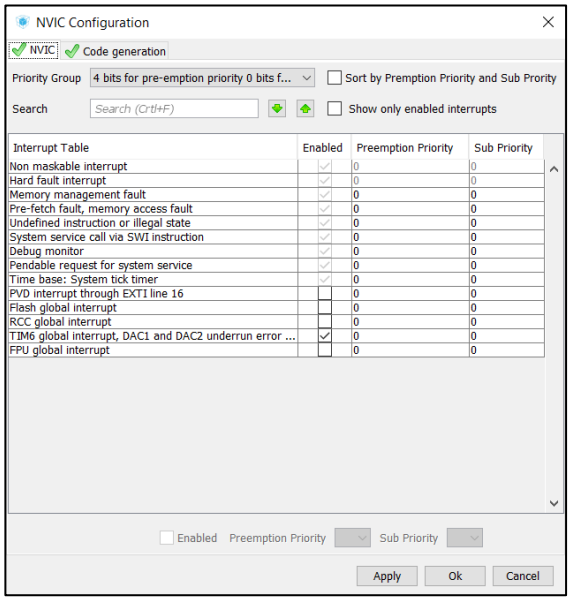

Figura 5-32. Activación de la interrupción de un timer con STM32CubeMx.[32]

Ahora, se ha de configurar el periodo del timer, para ello, en la misma pestaña de configuración se ha de pulsar

[30] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[31] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[32] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

sobre el botón que corresponde a nuestro timer. Dentro de la pantalla observaremos que aparece un valor de preescalado así como otro llamado periodo de conteo. Teniendo en cuenta que la frecuencia de entrada son 25 millones de Hz, para que sea más sencillo de utilizar y elegir un periodo de funcionamiento, vamos a elegir un periodo de 250000Hz de forma que:

$$\frac{250000000}{250000} = 1000$$

Así pues, de esta forma, un valor de 1000 en el valor de conteo corresponderá a 1 segundo. Como nuestro valor deseado son 350ms:

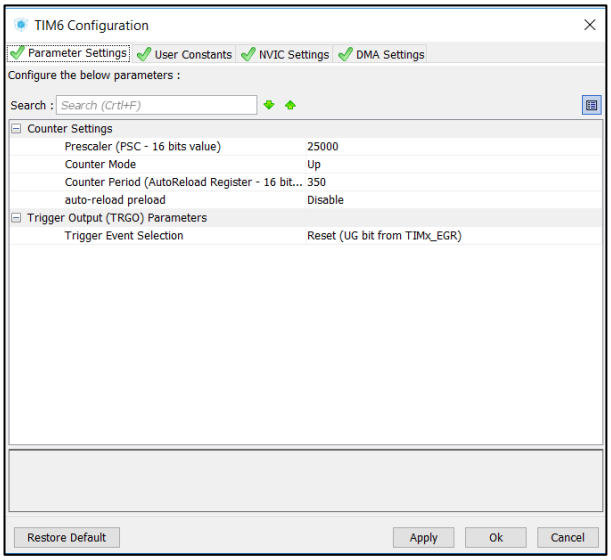

Figura 5-33. Configuración de la interrupción de un timer con STM32CubeMx.[33]

Una vez realizado esto, al igual que en el ejemplo del GPIO, podemos generar el proyecto y abrir µVision5.

Así pues, en modo resumen, se han realizado las siguientes configuraciones:

- Configurado el reloj del microcontrolador a 50Mhz.
- Configurado el modo de debug del microcontrolador.
- Configurados los pines de entrada/salida.
- Activadas varias opciones del micro.
- Desactivada la interrupción global del reloj.
- Activado el timer número 6.
- Configurado el timer a 350ms.
- Generada la interrupción del timer.
- Generado el proyecto.
- Abierto el Toolchain, listo para comenzar a trabajar.

5.2.4.2 Resolución del problema con MDK-ARM V5

Puesto que en el ejemplo anterior se realizaba una lectura continua del pulsador, con el timer configurado a un periodo determinado, se evita que la máquina de estados programada, avance sin control.

De esta forma, vamos a utilizar la función de interrupción del timer de forma que esta, nos permitirá leer el pulsador cuando nosotros queramos.

Así pues, si se abre el archivo stm32f7xx_it.c, se puede observar que existe una función llamada

[33] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

TIM6_DAC_IRQHandler(void). Dicha función es nuestra función de interrupción que saltará cada 350ms. Es ahí donde nos interesa hacer la lectura del pulsador para así aumentar o no la variable de estados.

Puesto que todas las variables se encuentran en el main del proyecto, recurrimos a una función declarada como externa en el vector de interrupciones que será llamada en la misma función de interrupción. De este modo, en el vector de interrupciones tan solo tenemos que declarar y llamar, para así realizar el resto de operaciones en el main, evitando tener que declarar variables globales o como externas para así poder utilizarlas en ambos archivos.

De esta forma, creamos la función void timer_6(void), una función vacía que hace lectura del botón y ejecuta las ordenes pertinentes para incrementar o decrementar la variable de estados.

5.3 Programación de la UART

5.3.1 Descripción del periférico

El producto dispone de 4 UART y 4 USART de las cuales una de ellas, la USART1, se usa para el puerto virtual que permite a la placa comunicarse con el PC mediante el uso de un software tipo TeraTerm.

5.3.2 Librería del periférico

Para la utilización de la UART se ha utilizado una de las librerías incluidas en la memoria de la placa. Su nombre es *stm32f7xx_hal_uart.c* y sus funciones más importantes son:

- **HAL_UART_GetError(UART_HandleTypeDef *huart)**: Devuelve un valor que indica el error que está ocurriendo.
- **HAL_UART_IRQHandler(UART_HandleTypeDef *huart)**: Controla la petición de la interrupción.
- **HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef *huart)**: Realiza la inicialización de la UART según los valores especificados.
- **HAL_UART_StateTypeDef HAL_UART_GetState(UART_HandleTypeDef *huart)**: Devuelve un valor que indica el estado.
- **HAL_StatusTypeDef HAL_UART_DeInit(UART_HandleTypeDef *huart)**: Reinicializa el periférico a los valores por defecto.
- **HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)**: Envía información en bloque.
- **HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)**: Recibe información en modo de interrupción.

5.3.3 Configuración del periférico

Al igual que en los demás ejemplos, se utilizará el software STM32CubeMx para la configuración de la UART.

Hay varias UART disponibles, pero se ha de seleccionar aquella que permite la transmisión y recepción de datos vía USB.

Del mismo modo que en el caso anterior, STM32CubeMX crea todas las funciones necesarias para la configuración de la UART. No obstante, para entender completamente la configuración del periférico, utilizaré las funciones creadas para conocer cómo sería la la configuración manual de la UART.

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART1_UART_Init();
```

Si nos dirijimos a la función que nos interesa, la que inicializa la UART con sus conrrespondientes variables, como el numero de baudios, la longitud, la paridad, el modo…

```
static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

5.3.4 Ejemplo práctico

Utilice el pinout de la STM32F7 Discovery y las librerías necesarias para solucionar el siguiente problema: Utilizando cuatro LEDs conectados a la placa y la UART1 del microcontrolador, elabore un programa que, usando el puerto serie y Tera Term, permita cambiar el estado de una máquina de 4 estados con diferentes configuraciones para los LED.

5.3.4.1 Generación del proyecto con STM32CubeMx

Para este ejemplo se partirá de un proyecto vacío. Antes de nada, se ha de configurar el reloj, para ello en la pestaña Clock Configuration y se ha de escribir 216MHz en el bloque correspondiente:

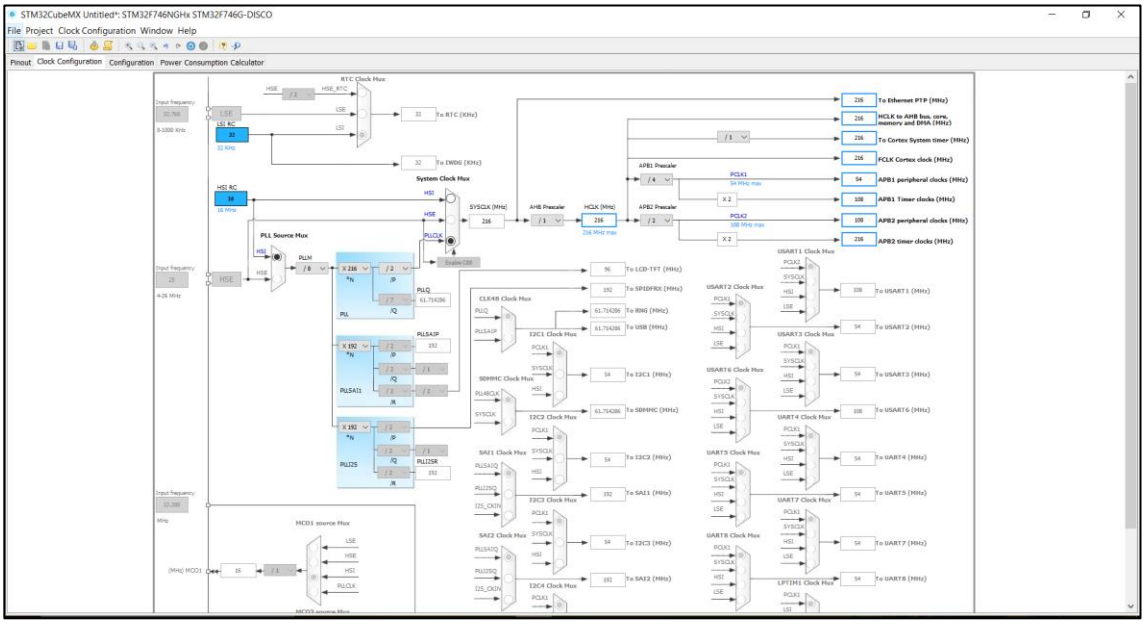

Figura 5-34. Configuración de la frecuencia del reloj con STM32CubeMx.[34]

Volviendo a la pestaña de pinout, en la columna de la izquierda, en la pestaña SYS se configura el Debug como Serial Wire:

[34] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

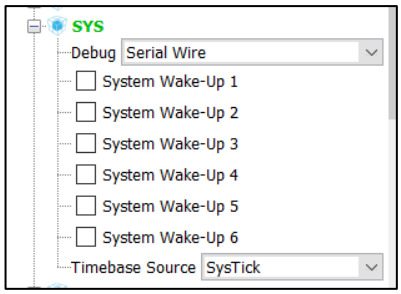

Figura 5-35. Configuración del debug con STM32CubeMx.[35]

En la misma columna, se ha de configurar la UART deseada que aparece bajo el nombre de USART1 y que debe funcionar en modo asíncrono:

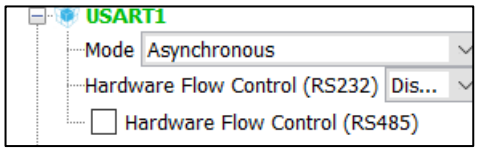

Figura 5-36. Activación de la UART con STM32CubeMx.[36]

En cuanto a los pines de salida, se han de configurar como en el ejemplo del punto 5.1.4 los pines PI0, PI2, PI3 y PH6.

Así como en el ejemplo del timer era necesario deshabilitar la interrupción del reloj, para la UART es necesario activarlo de nuevo, por lo que en la pestaña Configuration, en NVIC, se ha de marcar la interrupción del micro.

En cuanto a la UART, se han configurar los baudios, así como los bits y su interrupción. Para ello se ha de hacer click sobre el botón correspondiente y, en la pantalla que se abre, seleccionar 9600 Bits/s y una longitud de palabra de 8 Bits:

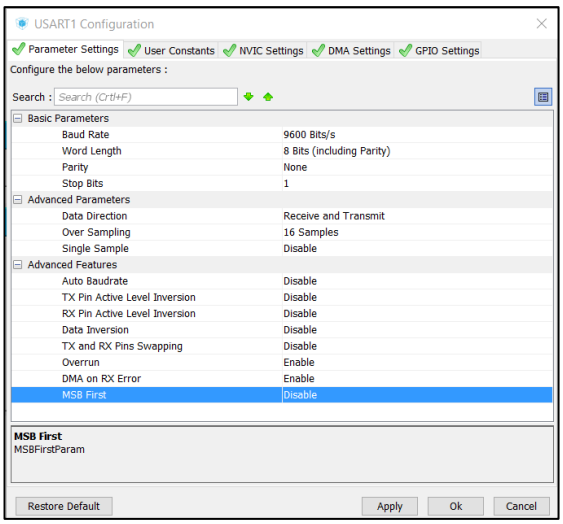

Figura 5-37. Configuración de la UART con STM32CubeMx.[37]

En la misma ventana, en NVIC Settings, es necesario activar su interrupción.

[35] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[36] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[37] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

Para terminar, en CORTEX_M7, hay que activar ART ACCELERATOR, Instruction Prefetch, CPU ICache y CPU DCache.

No nos podemos olvidar de instalar Tera Term y configurar el puerto serie de nuestra placa, a 9600 Bits/s y 8 Bits de ancho de palabra.

Así pues, en modo resumen, se ha:

- Configurado el reloj del microcontrolador a 216MHz.
- Configurado el modo de debug del microcontrolador.
- Configurados los pines de salida.
- Configurada la UART1.
- Activada la interrupción de la UART1.
- Activadas varias opciones del micro.
- Activada la interrupción global del reloj.
- Generado el proyecto.
- Abierto el Toolchain, listo para comenzar a trabajar.

Una vez abierto el proyecto en MDK-ARM V5 es necesario configurar las opciones del debug para que funcione el envío y recepción de datos del puerto serie. Para ello, hay que dirigirse Options for target que se encuentra en la barra superior de µVision5:

Figura 5-38. Icono Options for Target en MDK-ARM V5: Keil µVision5.[38]

Una vez abierto, en la pestaña de debug, hay que hacer click en el botón de Settings que se encuentra a la derecha:

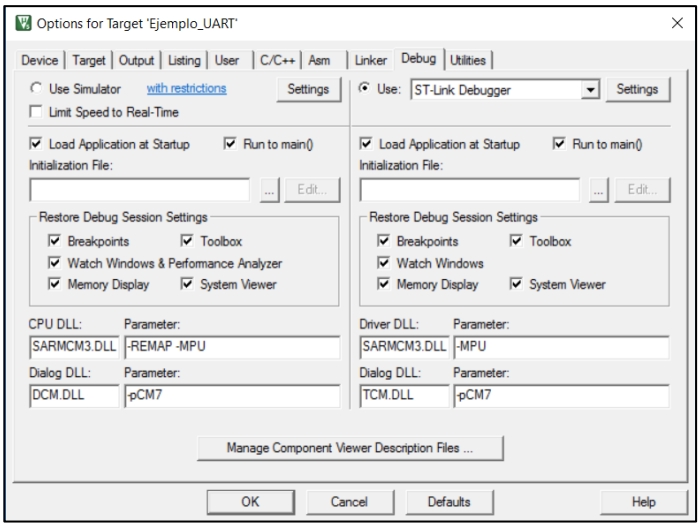

Figura 5-39. Options for Target en MDK-ARM V5: Keil µVision5.[39]

Una vez abierto, en la pestaña de Flash Download, hay que marcar Reset and Run antes de compilar el proyecto; es muy importante hacerlo antes, puesto que si pulsamos build antes de seleccionarlo, no funcionará. Una vez hecho esto, no debería de haber ningún tipo de problema con Tera Term:

[38] Imagen adquirida mediante impresión de pantalla durante el uso del programa MDK-ARM V5: Keil µVision5.
[39] Imagen adquirida mediante impresión de pantalla durante el uso del programa MDK-ARM V5: Keil µVision5.

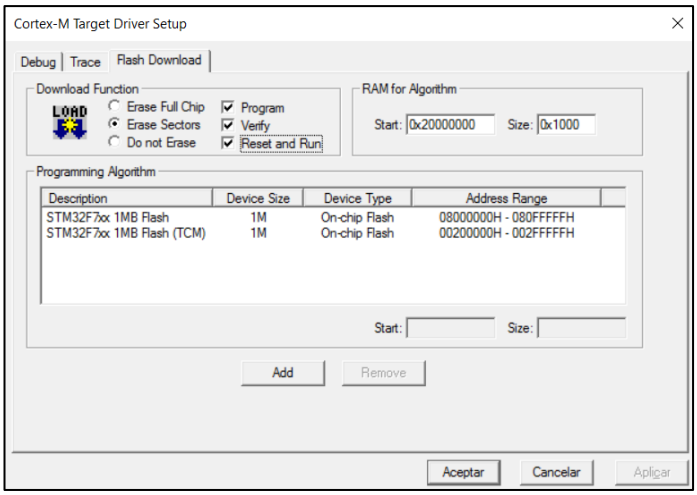

Figura 5-40. Options for Target en MDK-ARM V5: Keil µVision5.[40]

Aunque el envío de caracteres y datos y su impresión por pantalla funciona a la perfección, es cierto que al escribir datos para enviar, estos no aparecen, aunque si son recibidos en el micro.

5.3.4.2 Resolución del problema con MDK-ARM V5

Para la resolución de este problema se utilizan las funciones de transmisión y recepción de la librería. Para ello, en el programa principal se activa la recepción justo antes del bucle infinito.

Una vez se entra en el bucle se utiliza la función de transmisión para pedir por pantalla un número y se levanta una bandera que se utiliza para que no se esté pidiendo constantemente.

En cuanto al vector de interrupciones, se declara y define una función que analiza la variable en la que se almacena el dato recibido y activa o desactiva los pines correspondientes usando una máquina de estados. Cada vez que se recibe un dato se baja la bandera permitiendo que se pida por pantalla un nuevo dato. Al final de la máquina de estado se vuelve a activar la recepción, permitiendo el almacenamiento de un un nuevo dato.

[40] Imagen adquirida mediante impresión de pantalla durante el uso del programa MDK-ARM V5: Keil µVision5.

5.4 Programación de la ADC

5.4.1 Descripción del periférico

La STM32F746G Discovery cuenta con 3 convertidores analógico digitales que tienen desde 16 a 24 canales de entrada con una resolución de 12b y una velocidad máxima de 2.4Msamples/s. Dispone también de un watchdog por ADC. Tienen diferentes modos de funcionamiento: Simple, contínuo, escaneo, discontínuo o inyectado y el procesamiento de datos se realiza mediante interrupciones o el DMA.

5.4.2 Librería del periférico

Para la utilización del ADC se ha utilizado una de las librerías incluidas en la memoria de la placa. Su nombre es stm32f7xx_hal_tim.c y sus funciones más importantes son:

- **HAL_ADC_Start(ADC_HandleTypeDef* hadc)**: Inicializa la conversión del ADC por el canal indicado.
- **HAL_ADC_GetValue(ADC_HandleTypeDef* hadc)**: Toma un valor del canal indicado.
- **HAL_ADC_Init(ADC_HandleTypeDef* hadc)**: Realiza la inicialización del ADC según los valores especificados.
- **HAL_ADC_Stop(ADC_HandleTypeDef* hadc)**: Para la conversion del canal indicado.
- **HAL_ADC_GetError(ADC_HandleTypeDef *hadc)**: Retorna el error del ADC del canal indicado.
- **HAL_ADC_GetState(ADC_HandleTypeDef* hadc)**: Retorna el estado del ADC del canal indicado.
- **HAL_ADC_IRQHandler(ADC_HandleTypeDef* hadc)**: Controla la petición de la interrupción.
- **HAL_ADC_DeInit(ADC_HandleTypeDef* hadc)**: Resetea los valores de configuración a valores por defecto.

5.4.3 Configuración del periférico

Al igual que en los demás ejemplos, hay que utilizar el software STM32CubeMx para la configuración del ADC.

Hay varios ADC disponibles, pero se ha de seleccionar aquel correspondiente al pin que queramos utilizar.

Del mismo modo que en el caso anterior, STM32CubeMX crea todas las funciones necesarias para la configuración del ADC. No obstante, para entender completamente la configuración del periférico, utilizaré las funciones creadas para conocer cómo sería la la configuración manual del ADC.

```
/* Private function prototypes -----------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_ADC3_Init(void);
static void MX_TIM6_Init(void);
```

Sin contar con las dos primeras funciones, que configuran el reloj y la función de detención de erorres, la primera de ellas se encarga de configurar los pines para el reloj, la segunda de configurar la UART, la cuarta el timer y la tercera, y única que nos interesa, el ADC:

```
/* ADC3 init function */
static void MX_ADC3_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock, Resolution, Data
Alignment and number of conversion)
    */
  hadc3.Instance = ADC3;
  hadc3.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc3.Init.Resolution = ADC_RESOLUTION_12B;
  hadc3.Init.ScanConvMode = DISABLE;
  hadc3.Init.ContinuousConvMode = DISABLE;
  hadc3.Init.DiscontinuousConvMode = DISABLE;
  hadc3.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc3.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc3.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc3.Init.NbrOfConversion = 1;
  hadc3.Init.DMAContinuousRequests = DISABLE;
  hadc3.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc3) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure for the selected ADC regular channel its corresponding
rank in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_8;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}
```

Tal y como podemos observar en el trozo de código anterior, la función define los parámetros del ADC, tales como el preescalado, la resolución, el modo, el trigger…

5.4.4 Ejemplo práctico

Utilice el pinout de la STM32F7 Discovery y las librerías necesarias para solucionar el siguiente problema: Utilizando un timer, la UART y el ADC elabore un programa que lea el valor de un potenciómetro en la interrupción del timer y lo imprima por pantalla usando el puerto serie.

5.4.4.1 Generación del proyecto con STM32CubeMx

Este ejemplo es un compendio de los dos puntos anteriores, en los que se configura la UART y un timer. Puesto que ya se ha explicado en apartados anteriores, vamos a suponer por configurados los siguientes pasos:

- Configurado el reloj del microcontrolador a 216MHz.
- Configurado el modo de debug del microcontrolador.
- Configurada la UART1.

- Activadas varias opciones del micro.
- Activada la interrupción global del reloj.
- Activado el timer número 6.
- Configurado el timer a 350ms.
- Generada la interrupción del timer.

En cuanto al ADC es bastante sencillo de configurar. En primer lugar se ha de buscar en el pinout el pin accesible que queremos usar para así saber cual configurar. En esta ocasión, se usará el ADC3 accesible mediante el pin PF10.
Así pues, para la configuración de este seleccionamos el pin correspondiente y activamos el ADC:

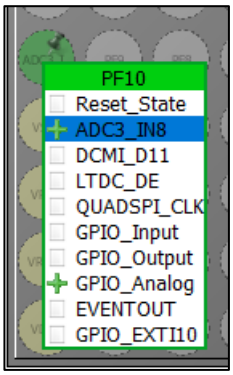

Figura 5-41. Configuración del pin PF10 para su uso como ADC.[41]

Una vez seleccionado el pin nos aseguramos en la columna izquierda del programa que el convertidor analógico digital número tres está activo:

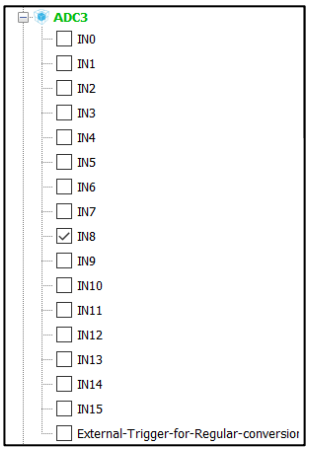

Figura 5-42. Configuración del ADC3.[42]

[41] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[42] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

Finalmente, en la pestaña de configuración, hay dejar la configuración por defecto del ADC:

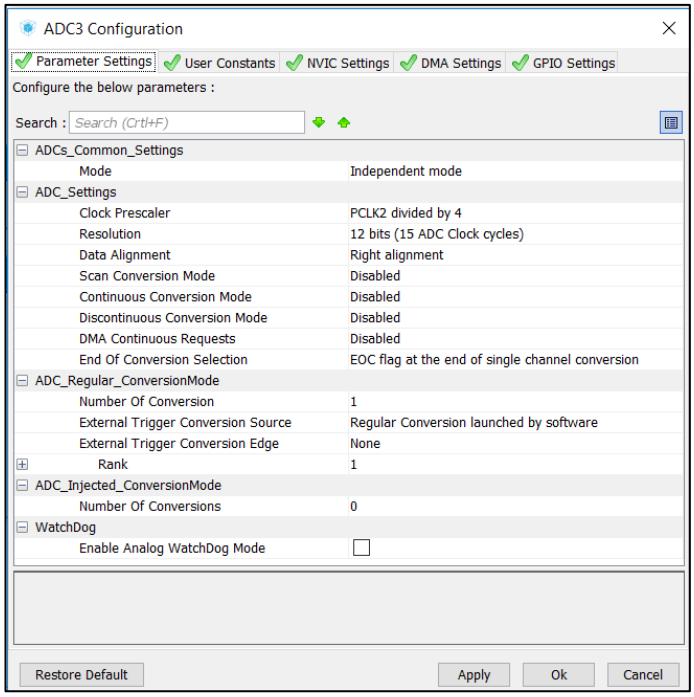

Figura 5-43. Configuración de parámetros del ADC3.[43]

De esta forma, guardando y generando el proyecto queda todo completamente preparado para poder resolver el problema propuesto, habiendo seguido los siguientes pasos:

- Configurado el reloj del microcontrolador a 216MHz.
- Configurado el modo de debug del microcontrolador.
- Configurada la UART1.
- Activadas varias opciones del micro.
- Activada la interrupción global del reloj.
- Activado el timer número 6.
- Configurado el timer a 350ms.
- Generada la interrupción del timer.
- Configurado el ADC3 a través del pin PF10.
- Configurado el ADC3 por defecto.
- Generado el proyecto.
- Abierto el Toolchain, listo para comenzar a trabajar.

5.4.4.2 Resolución del problema con MDK-ARM V5

La resolución de este problema es bastante sencilla aunque algo laboriosa debido al gran número de periféricos que entran en juego.

Se crea una función externa en el vector de interrupciones que salta cada vez que lo hace la interrupción del timer, el cual se inicializa justo antes del bucle infinito.

Dicha función externa se encuentra en el programa principal y se encarga de realizar una lectura usando la librería del ADC y transformado el dato de flotante a carácter, para que este pueda ser impreso por pantalla mediante la UART.

[43] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

5.5 Programación de la LCD

5.5.1 Descripción del periférico

La pantalla LCD del producto es de 4.3 pulgadas y tiene una resolución de 480x272 píxeles. Además, cuenta con una pantalla táctil capacitiva.

5.5.2 Librería del periférico

Como se verá en el punto 5.5.4.1, este periférico requiere de la configuración del path e inclusión de las librerías de la SDRAM, LCD y el propio kit, lo que significa que será el primer ejemplo en el que se trabajará con una librería distinta a la HAL. Es por ello por lo que a las funciones que se muestran a continuación serán un poco diferentes. Hay muchísimas funciones interesantes que permiten dibujar formas, cambiar colores y, en general, dejan al usuario crear interfaces elaboradas. No obstante, a continuación nombraré las usadas en el ejemplo y mostraré un pantallazo de todas ellas, puesto que enumerarlas y explicarlas todas aumentaría de una forma importante el contenido del capítulo, desviándonos del objetivo fundamental de este trabajo. De esta manera, las funciones usadas son:

- **BSP_LCD_Init(void)**: Inicializa el LCD.
- **BSP_LCD_LayerDefaultInit(uint16_t LayerIndex, uint32_t FB_Address)**: Inicializa la capa del LCD en formato ARGB8888, es decir, con 32 bits por píxel.
- **BSP_LCD_SelectLayer(uint32_t LayerIndex)**: Selecciona la capa del LCD.
- **BSP_LCD_Clear(uint32_t Color)**: Limpia la pantalla pintando del color seleccionado el fondo.
- **BSP_LCD_SetBackColor(uint32_t Color)**: Permite seleccionar el color del fondo.
- **BSP_LCD_SetTextColor(uint32_t Color)**: Selecciona el color del texto a escribir.
- **BSP_LCD_DrawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2)**: Dibuja una linea por coordenadas.
- **BSP_LCD_DisplayStringAt(uint16_t Xpos, uint16_t Ypos, uint8_t *Text, Text_AlignModeTypdef Mode):** Imprime una cadena de caracteres indicando las coordenadas, el vector y el modo de impresión.
- **BSP_LCD_DeInit(void)**: Reinicializa el LCD con los valores por defecto.

En cuanto al resto de funciones, las cuales podemos encontrar en los archivos del proyecto, son:

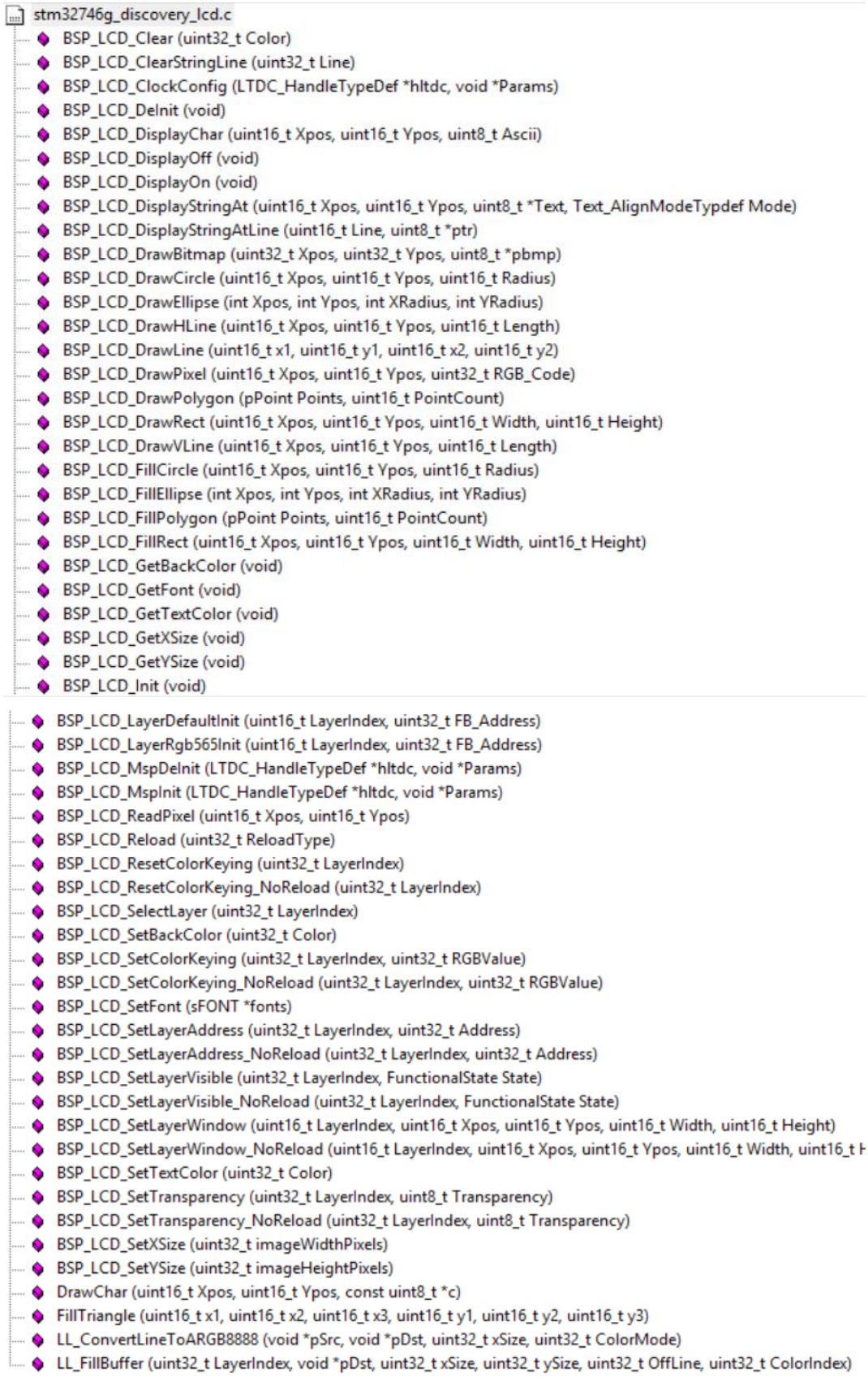

Figura 5-44. Funciones del LCD.[44]

[44] Imagen adquirida mediante impresión de pantalla durante el uso del programa MDK-ARM V5: Keil µVision5.

5.5.3 Configuración del periférico

La configuración del periférico en este ejempo es muy tediosa ya que el número de periféricos que entran en juego es enorme: LCD, SDRAM, ADC, GPIO…

5.5.4 Ejemplo práctico

Utilice el pinout de la STM32F7 Discovery y las librerías necesarias para solucionar el siguiente problema: Utilizando el ADC y la pantalla LCD elabore un programa que lea el valor de un potenciómetro y lo imprima por la pantalla LCD del kit. Para ello, elabore una pequeña interfaz que indique numéricamente el valor, así como un objeto visual que indique el valor.

5.5.4.1 Generación del proyecto con STM32CubeMx

La configuración de los módulos necesarios para este problema es larga ya que se necesitan muchos de ellos. Así pues, para que no sea difícil y quede todo claro, voy a explicar como configurar todo desde el principio.

En primer lugar y antes que nada, se ha de activar el acelerador de gráficos de modo en la columna izquierda se ha de activar el DMA2D:

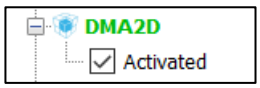

Figura 5-45. Configuración del acelerador de gráficos DMA2D[45]

Para conseguir mantener el estado en el que se encuentra la pantalla debemos configurar la SDRAM. Así pues, en el desplegable FMC hay que hacer click sobre SDRAM1, la cual se ha de configurar tal y como se aprecia a continuación:

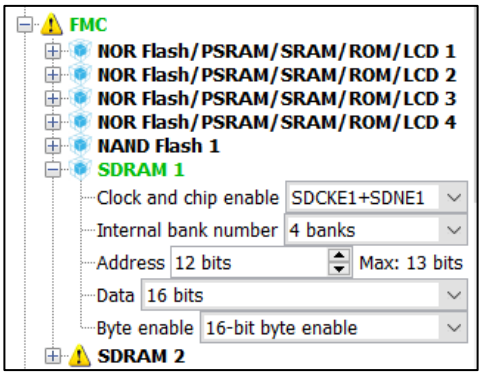

Figura 5-46. Configuración de la SDRAM[46]

El I2C también ha de ser activado:

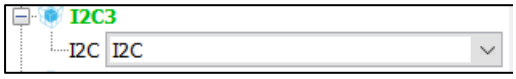

Figura 5-47. Configuración del I2C[47]

[45] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[46] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[47] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

Para activar todos los pines necesarios que permiten el funcionamiento de la pantalla, en la pestaña LTCD y se configura para que funcione a 18 bits:

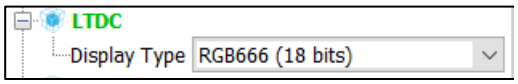

Figura 5-48. Configuración del LTCD[48]

En cuanto al SPI, como Full-Duplex Master:

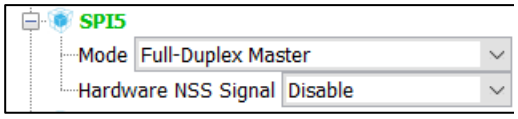

Figura 5-49. Configuración del SPI[49]

Finalmente hay que configurar como en los apartados anteriores la USART6 y el ADC1.

En la pestaña de configuración, se ha de activar el acelerador y las memorias caché haciendo click sobre el botón correspondiente al micro:

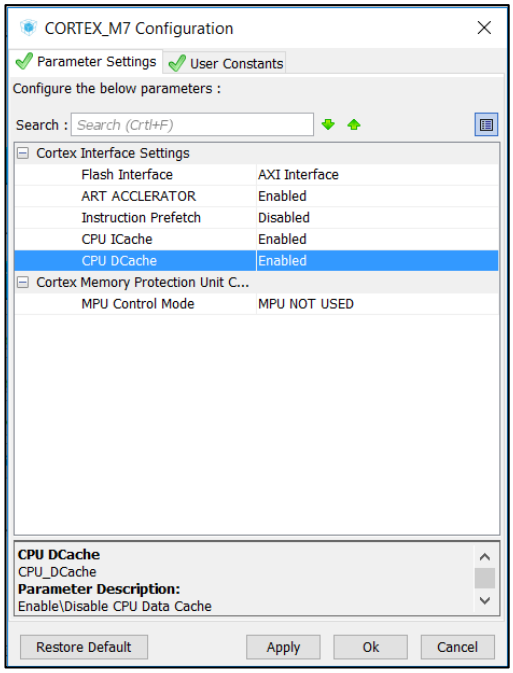

Figura 5-50. Configuración del micro[50]

Así pues, en modo resumen, se ha hecho lo siguiente:

- Configurado el DMAD2.
- Configurada la SDRAM.
- Configurado el I2C.
- Configurado el LTCD.
- Configurado el SPI.

[48] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[49] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.
[50] Imagen adquirida mediante impresión de pantalla durante el uso del programa STM21CubeMx.

- Configurada la USART.
- Configurado el ADC.
- Activados las caché y el acelerador del mircro.
- Generar el proyecto y abrir el MDK-ARM V5

Debido a las librerías y a algunos archivos no incluidos en los paths del MDK-ARM por defecto, es necesario realizar algunos pasos en el mismo para que la pantalla pueda ser utilizada.

Suponiendo que tenemos instalado todo el software requerido en el ordenador, lo primero que hay que hacer es dirigirse a la ruta "C:\Users\UserName\STM32Cube" y hacer click sobre la carpeta "Repository". Dentro de ella hay que seleccionar y abrir la carpeta que contenga la última versión, siendo en mi caso la carpeta "STM32Cube_FW_F7_V1.7.0".

Dentro de esa carpeta hay otras dos carpetas que debemos copiar. La primera de ellas es la carpeta "Utilities" y la segunda de ellas está dentro de "Drivers" y se llama "BSP". De esta forma, lo que tendremos que hacer, en resumen, es acceder a estas direcciones y copiar las carpetas indicadas:

1. En **"C:\Users\UserName\STM32Cube\Repository\STM32Cube_FW_F7_V1.7.0"** copiar "Utilities".
2. En **"C:\Users\UserName\STM32Cube\Repository\STM32Cube_FW_F7_V1.7.0\Drivers"** copiar "BSP".

Para continuar se ha de abrir una ventana con el proyecto generado por el STM32CubeMx en la hay que pegar "Utilities". Dentro de la carpeta del proyecto, en la carpeta "Drivers", se ha de pegar "BSP".

Una vez pegado los archivos, generamos de nuevo el proyecto con el STM32CubeMx y esperamos a que se abra el Keil.

Una vez abierto, en Options for target:

Figura 5-51. Icono Options for Target en MDK-ARM V5: Keil µVision5.[51]

En la pestaña de C/C++ se ha de abrir Include paths:

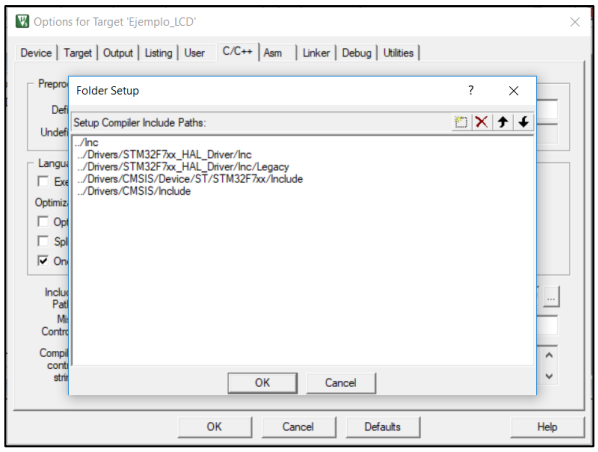

Figura 5-52. Include paths en MDK-ARM V5: Keil µVision5.[52]

[51] Imagen adquirida mediante impresión de pantalla durante el uso del programa MDK-ARM V5: Keil µVision5.
[52] Imagen adquirida mediante impresión de pantalla durante el uso del programa MDK-ARM V5: Keil µVision5.

Y se añade el path que vemos a continuación:

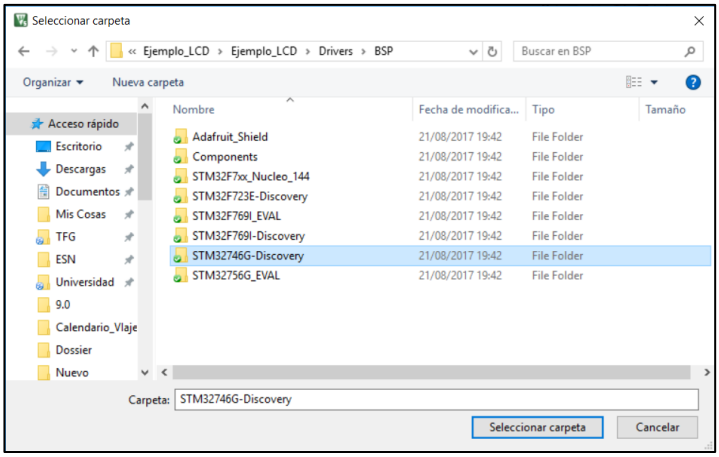

Figura 5-53. Incluir path del kit en MDK-ARM V5: Keil µVision5.[53]

Tras seleccionar la ruta, se guarda el proyecto. En Manage Project Items:

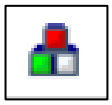

Figura 5-54. Icono Manage Project Items en MDK-ARM V5: Keil µVision5.[54]

Se ha de añadir un grupo cuyo nombre debe de ser "BSP/stm32f746g_Discovery" y una vez creado, se seleccionan los archivos de las librerías incluídos en el camino añadido en Options for Target que interesan: Kit, LCD y SDRAM. Finalmente hay que hacer click en Set as Current Target:

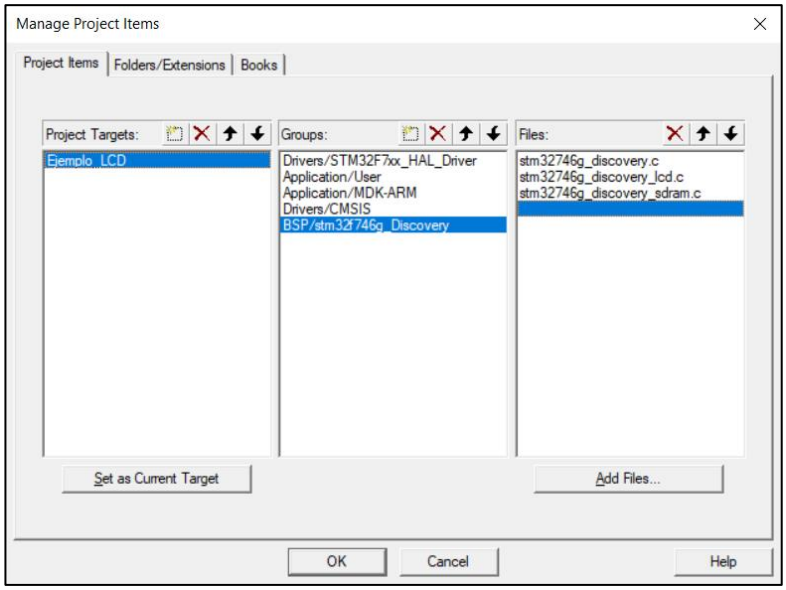

Figura 5-55. Librerías de BSP en MDK-ARM V5: Keil µVision5.[55]

[53] Imagen adquirida mediante impresión de pantalla durante el uso del programa MDK-ARM V5: Keil µVision5.
[54] Imagen adquirida mediante impresión de pantalla durante el uso del programa MDK-ARM V5: Keil µVision5.
[55] Imagen adquirida mediante impresión de pantalla durante el uso del programa MDK-ARM V5: Keil µVision5.

Para terminar no podemos olvidar incluir las librerías en el main del proyecto:

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f7xx_hal.h"
#include "stm32746g_discovery.h"
#include "stm32746g_discovery_lcd.h"
#include "stm32746g_discovery_sdram.h"
```

Una vez llegamos a este punto, las librerías necesarias para utilizar el LCD están disponibles, así como su documentación, en el mismo Keil.

5.5.4.2 Resolución del problema con MDK-ARM V5

La resolución del problema es relativamente sencilla teniendo en cuenta que la mayoría de los periféricos en uso ya se han utilizado en los ejemplos anteriores. Así pues, el núcleo del problema está en el uso de las librerías explicadas anteriormente.

Dentro del programa, antes de entrar en el while se inicializa la pantalla y se selecciona el modo de 32bits por píxeles, además de la capa y se realiza un clear. Tras esto se llama a la función “Interface”, la cual pinta la pantalla de gris (bajo mi punto de vista blanco) y dibuja un cuadro.

Dentro del while se realiza una llamada continua a una función llamada “conversion” que se encarga de inicializar el ADC, tomar el dato, convertirlo y rellenar la cadena de caracteres de manera adecuada para que finalmente sea impresa por pantalla. Tras esta llamada se realiza un delay con la función HAL_Delay para que el ADC no este funcionando de manera contínua.

En cuanto al objeto visual es un rectángulo cuyo ancho varía con el valor del ADC.

5.6 Programación del Touchscreen

5.6.1 Descripción del periférico

La pantalla LCD del producto es de 4.3 pulgadas y tiene una resolución de 480x272 píxeles. Además, cuenta con una pantalla táctil capacitiva.

5.6.2 Librería del periférico

Al igual que en el punto anterior se agregaban las librerías para el LCD, en esta ocasión, además de las añadidas en el proyecto anterior, es necesario añadir las correspondientes al touchscreen. Como la librería correspondiente al LCD quedó explicada en el capitulo anterior, en este punto me limitaré a explicar las funciones principales del táctil de la pantalla:

- **BSP_TS_Init(uint16_t ts_SizeX, uint16_t ts_SizeY)**: Inicializa la pantalla táctil.
- **BSP_TS_GetState(TS_StateTypeDef *TS_State)**: Consigue la posición donde se está tocando la pantalla.

5.6.3 Configuración del periférico

Dado que el proyecto requiere exactamente los mismos periféricos que el ejemplo anterior, carece de sentido repetir la información recogida en el punto 5.5.3.

5.6.4 Ejemplo práctico

Utilice el pinout de la STM32F7 Discovery y las librerías necesarias para solucionar el siguiente problema: Utilizando la pantalla LCD y los sensores táctiles de la misma, elabore un programa que permita dibujar usando el dedo de la mano.

5.6.4.1 Generación del proyecto con STM32CubeMx

Puesto que este proyecto requiere del uso del LCD incluido en el kit, se ha de partir del ejemplo anterior en el que se realizaron las siguientes configuraciones:

- Configurado el DMAD2.
- Configurada la SDRAM.
- Configurado el I2C.
- Configurado el LTCD.
- Configurado el SPI.
- Configurada la USART.
- Configurado el ADC.
- Activados las caché y el acelerador del mircro.
- Generar el proyecto y abrir el MDK-ARM V5

Es necesario mencionar que el proyecto tiene que tener incluidas las carpetas BSP y Utilities que se pegaron en el ejemplo anterior para así poder añadir las librerías.

Si se consultan los esquemáticos del producto, se puede comprobar que la entrada del touchscreen tiene se realiza a través del I2C usando los pines SCL y SDA:

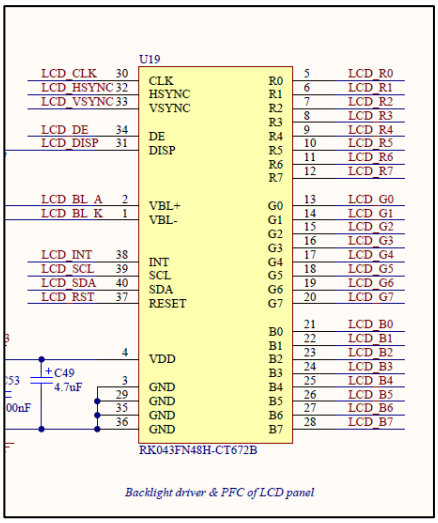

Figura 5-56. Esquemático del LCD.[56]

En el mismo manual, si se busca a qué pines del núcleo se encuentran asignados, se comprueba que lo están a los pines PH7 y PH8.

Al configurar el I2C3, STMCubeMx por defecto habilita otros pines que no corresponden a los que el LCD necesita para la entrada, por lo que deben de ser seleccionados manualmente los pines PH7 y PH8 para que así, al tocar la pantalla, se pueda recoger el gesto. De este modo, queda configurada la pantalla táctil al cien por cien.

Al igual que en el ejemplo anterior hay que añadir librerías al proyecto, para ello, en Project manager se añade la librería stm32746g_discovery_ts.c al grupo BSP/stm32f746g_Discovery que se encuentra en la misma carpeta en la que estaban la del LCD, la SDRAM…

También es necesario crear un nuevo grupo al que llamaremos Drivers/ft5336 en el que es necesario añadir la librería ft5336.c que se encuentra en Drivers/BSP/Componentes/ft5336 en la carpeta del proyecto. Al igual que se hizo para las librerías BSP, el path de la ft5336 debe de ser añadido en "options for target" como se hizo en el punto 5.5.4.1

Para terminar, no hay que olvidar incluir las librerías:

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f7xx_hal.h"
#include "stm32746g_discovery_ts.h"
#include "stm32746g_discovery_sdram.h"
#include "stm32746g_discovery_lcd.h"
#include "stm32746g_discovery.h"
#include "ft5336.h"
```

5.6.4.2 Resolución del problema con MDK-ARM V5

Como se ha mencionado antes, el proyecto es reciclado, por lo que se parte del ejemplo anterior.

La pantalla táctil de la placa funciona de tal manera que cada vez que se toca salta una interrupción que, en este caso, no ha sido configurada con el STM32CubeMx. Por lo tanto, ha sido necesario añadirla manualmente, buscando el handler asociado al touchscreen en el archivo startup_stm32f746xx.s.

Dicho handler ha de ser copiado en el vector de interrupciones con su función IRQhandler correspondiente.

Para pulir más el código, se ha añadido una función callback que es la encargada de realizar una función determinada ante un toque, en nuestro caso, pintar.

[56] Imagen adquirida mediante impresión de pantalla de los esquemáticos del producto disponibles en Keil.

6. CREACIÓN DE INTERFACES

Herramientas, librerías y limitaciones

En este capítulo se recogen algunas de las herramientas que se pueden utilizar para la creación de interfaces y ventanas de usuario para la Discovery. Se ha intentado hacer una interfaz, incluir imágenes y todo lo posible para utilizar las herramientas que se explicarán, pero debido al alto precio de las licencias de las aplicaciones, así como la limitación de la licencia gratuita de MDK-ARM V5, han imposibilitado incluir una explicación completa al igual que se ha hecho con cada periférico

6.1 Software de creación de interfaces para la STM32F7 Discovery

6.1.1 Embedded Wizard GUI builder

Dispone de una version de evaluación que permite al usuario crear interfaces de dos ventanas, insuficiente para cualquier proyecto de nivel medio. Su elevado precio, nada asequible ni si quiera para una institución educativa, hace que deje de ser una opción interesante. No obstante, la variedad de iconos, botones, barras deslizantes, imágenes…, hace de esta herramienta un accesorio muy completo para nuestro producto.

Tal y como podemos apreciar en la imagen que se muestra a continuación, el acabado de las interfaces es muy bueno y muy profesional, tanto, que es usado en muchas empresas que utilizan la Discovery:

Figura 6-57. Embedded Wizard GUI builder[57]

- Web: https://www.embedded-wizard.de/
- Precio versión starter: 2990€ (excl. VAT)

6.1.2 TouchGFX

Es prácticamente igual que el product anterior, pero en esta ocasión la versión de evaluación incluye más cosas, como una librería de ejemplos y demos que dan una idea más general del producto que incluso dispone de un simulador de interfaces para el PC. De nuevo el precio se convierte en hándicap.

- Web: http://touchgfx.com/
- Precio version para líneas de producción pequeñas: 5000€ (excl. VAT)

6.1.3 STemWin

Es la librería de gráficos que proporciona la propia STM. Da un resultado bastante bueno para el ámbito educacional y su configuración es bastante sencilla. No obstante, la versión gratuita del MDK-ARM V5 no permite compilar un proyecto superior a 32Kbytes,

6.2 Conclusiones

Dado que ninguna de las herramientas expuestas anteriormente satisface el objetivo, pienso que elaborar una librería propia con las funciones y librerías explicadas en los ejemplos del LCD y el Touchscreen. Es un trabajo no muy difícil aunque tedioso, que daría una solución al problema.

[57] Imagen adquirida de la página web oficial de Embedded Wizard GUI builder.

7. CONCLUSIÓN

Aplicaciones del producto, continuación del trabajo

Tras realizar una explicación teórica y comprobar el funcionamiento del kit STM32F7 Discovery se pueden sacar varias conclusiones que se resumirán en este capítulo. En esta parte del trabajo, se podrá obtener una idea de la aplicación inmediata en el ámbito educacional y los posibles caminos que se pueden seguir para así, buscar nuevas ideas para otros trabajos de fin de grado o máster.

7.1 Aplicación en asignaturas

En la ETSI, existen diferentes asignaturas del departamento de electrónica en las que se utilizan microcontroladores.

Tanto en el Grado de Ingeniería Industrial, como el Grado en Ingeniería Electrónica, Robótica y Mecatrónica existen las asignaturas de Sistemas Electrónicos y Sistemas Electrónicos para la Automatización Industrial. En ambas se utilizan microcontroladores de Texas Instruments muy limitados que, por una parte enseñan al alumno a optimizar el espacio y apreciar los recursos, pero por otra parte establecen una frontera entre la creatividad del alumno y las capacidades del núcleo.

El kit STM32F7 Discovery tiene un núcleo que, comparado con cualquiera de los microcontroladores usados en esos grados, es mucho más potente y ofrece mucha más posibilidades. Además, mientras los de Texas Instrument incluían LEDs, pulsadores y como mucho una entrada Ethernet, el STM32F7 Discovery incluye una pantalla táctil capacitiva, incontables canales de ADC, entrada y salida de audio, 2 mini USB, entrada de cámara, microSD… Un auténtico mundo de posiblidades.

No obstante, tiene una desventaja muy grande. Mientras los precios de las Texas Instrument no superaban los 20€, el producto en cuestión tiene un coste que no todo el mundo puede permitirse, por lo que el departamento tendría que disponer de un número considerable de placas disponibles, lo que supondría un desembolso considerable.

Como conclusión, siempre y cuando el alumno aprenda a utilizar un micro poco potente previamente y que el departamento disponga de suficiente dinero, utilizaría este kit en alguna asignatura de la ETSI, especialmente en Electrónica de Consumo, ya que tiene un bloque dedicado al audio.

7.2 Posibles futuros trabajos

La creación de interfaces ha sido un problema que ha estado presente durante la realización del trabajo y que nunca se ha podido resolver debido a la limitación de capacidad de la versión gratuita del compilador. Siempre y cuando el departamento adquiriese una versión completa del compilador, un alumno podría continuar con el proyecto investigando cómo crear interfaces a partir de un software o herramienta gratuita como STemWin o µGUI.

La investigación del funcionamiento de los periféricos expuestos en este trabajo ha supuesto muchísimo tiempo, por lo que he tenido que dejar algunos periféricos atrás. Continuar investigando acerca de los demás periféricos es otro posible trabajo de fin de grado.

En conclusión, esta placa ofrece un sinfín de posibilidades y, al ser un producto tan nuevo, la información es escasa, por lo que cualquier aspecto de ella es un posible reto para ser investigado y conformar un trabajo.

8. ANEXO DE CÓDIGOS

Códigos de los proyectos que resuelven los ejemplos prácticos

En este anexo se adjuntan los códigos de cada uno de los proyectos realizados para completar la explicación de los periféricos recogidos en este trabajo. Con ellos se intenta completar la información recogida en cada uno de los puntos para que así el lector pueda comprobar, sin necesidad de usar un ordenador, cómo se han utilizado las funciones de las librerías explicadas en cada uno de los puntos.

Existirá un subcapítulo por periférico, cada uno de los cuales contendrá a su vez otro subcapítulo, en el caso de que entre en juego el vector de interrupciones u otro archivo de importancia reseñable.

Cada línea de código está comentada o bien por el generador de código en inglés o por mi mismo en español.

8.1 Ejemplo práctico GPIO

8.1.1 main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f7xx_hal.h"

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);

/* Para facilitar la lectura y escritura del codigo, creo varios defines para
el encendido/apagado de LEDs*/
#define LED1ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_0, 1)
#define LED2ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_2, 1)
#define LED3ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3, 1)
#define LED4ON HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, 1)
#define LED1OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_0, 0)
#define LED2OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_2, 0)
#define LED3OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3, 0)
#define LED4OFF HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, 0)
#define UP HAL_GPIO_ReadPin(GPIOI, GPIO_PIN_11)

/*Declaracion de variables necesarias*/
int input;              //Variable que controlara la recepcion de señal desde
el boton
int state = 0;          //Variable de la maquina de estado
int flag_end = 0;       //Variable bandera fin de tarea

int main(void)
{
  /* Enable I-Cache-------------------------------------------------------------*/
  SCB_EnableICache();

  /* Enable D-Cache-------------------------------------------------------------*/
  SCB_EnableDCache();

  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();

  while (1)
  {
          switch (state){ //Máquina de estados
                case 0:
                      LED1OFF;
                      LED2OFF;
                      LED3OFF;
                      LED4OFF;
                      flag_end = 0;//Si ha vuelto a llegar al principio,
bajamos la bandera
                      break;
```

```
                    case 1:
                          LED1ON;
                          LED2OFF;
                          LED3OFF;
                          LED4OFF;
                          break;
                    case 2:
                          LED1ON;
                          LED2ON;
                          LED3OFF;
                          LED4OFF;
                          break;
                    case 3:
                          LED1ON;
                          LED2ON;
                          LED3ON;
                          LED4OFF;
                          break;
                    case 4:
                          LED1ON;
                          LED2ON;
                          LED3ON;
                          LED4ON;
                          flag_end = 1;//Si hemos llegado al final levantamos la
bandera
                          break;
              }
              if(UP == 1 && flag_end == 0) state++;//Si aun no hemos llegado al
final, continuamos aumentando state
              if(UP == 1 && flag_end == 1) state--;//Si hemos llegado al final,
cambiamos de sentido
              HAL_Delay(100);//Pequeño delay para poder manejar bien el pulsador

    }
}

/** System Clock Configuration*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

    /**Configure the main internal regulator output voltage
    */
  __HAL_RCC_PWR_CLK_ENABLE();

  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 216;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
```

```
    Error_Handler();
  }

    /**Activate the Over-Drive mode
    */
  if (HAL_PWREx_EnableOverDrive() != HAL_OK)
  {
    Error_Handler();
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOI_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_0, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, GPIO_PIN_RESET);

  /*Configure GPIO pins : PI3 PI2 PI0 */
  GPIO_InitStruct.Pin = GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);

  /*Configure GPIO pin : PI11 */
  GPIO_InitStruct.Pin = GPIO_PIN_11;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);

  /*Configure GPIO pin : PH6 */
  GPIO_InitStruct.Pin = GPIO_PIN_6;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOH, &GPIO_InitStruct);

}

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state
*/
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */

}
#endif

/**
  * @}
  */

/**
  * @}
*/
```

8.2 Ejemplo práctico Timer

8.2.1 main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f7xx_hal.h"

TIM_HandleTypeDef htim6;

/* Para facilitar la lectura y escritura del codigo, creo varios defines para
el encendido/apagado de LEDs*/
#define LED1ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_0, 1)
#define LED2ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_2, 1)
#define LED3ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3, 1)
#define LED4ON HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, 1)
#define LED1OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_0, 0)
#define LED2OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_2, 0)
#define LED3OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3, 0)
#define LED4OFF HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, 0)
#define UP HAL_GPIO_ReadPin(GPIOI, GPIO_PIN_11)

/*Declaracion de variables necesarias*/
int input;                    //Variable que controlara la recepcion de señal
desde el boton
int state = 0;          //Variable de la maquina de estado
int flag_end = 0; //Variable bandera fin de tarea

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_TIM6_Init(void);
void timer_6(void); //Función de interrupción del timer asociada al vector de
interrupciones

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* Enable I-Cache---------------------------------------------------------*/
  SCB_EnableICache();

  /* Enable D-Cache---------------------------------------------------------*/
  SCB_EnableDCache();

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM6_Init();
```

```
      HAL_TIM_Base_Start_IT(&htim6); //Inicialización del timer

while (1)
  {
            switch (state){ //Máquina de estados
                  case 0:
                        LED1OFF;
                        LED2OFF;
                        LED3OFF;
                        LED4OFF;
                        flag_end = 0;//Si ha vuelto a llegar al principio,
bajamos la bandera
                        break;
                  case 1:
                        LED1ON;
                        LED2OFF;
                        LED3OFF;
                        LED4OFF;
                        break;
                  case 2:
                        LED1ON;
                        LED2ON;
                        LED3OFF;
                        LED4OFF;
                        break;
                  case 3:
                        LED1ON;
                        LED2ON;
                        LED3ON;
                        LED4OFF;
                        break;
                  case 4:
                        LED1ON;
                        LED2ON;
                        LED3ON;
                        LED4ON;
                        flag_end = 1;//Si hemos llegado al final levantamos la
bandera
                        break;
            }
  }
}

void timer_6(void) //Función de interrupción/lectura del pulsador
{
      if(UP == 1 && flag_end == 0) state++;//Si aun no hemos llegado al
final, continuamos aumentando state
      if(UP == 1 && flag_end == 1) state--;//Si hemos llegado al final,
cambiamos de sentido
}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

    /**Configure the main internal regulator output voltage
    */
  __HAL_RCC_PWR_CLK_ENABLE();
```

```
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 50;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* TIM6 init function */
static void MX_TIM6_Init(void)
{

  TIM_MasterConfigTypeDef sMasterConfig;

  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 25000;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 350;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
```

```
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOI_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_0, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, GPIO_PIN_RESET);

  /*Configure GPIO pins : PI3 PI2 PI0 */
  GPIO_InitStruct.Pin = GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);

  /*Configure GPIO pin : PG7 */
  GPIO_InitStruct.Pin = GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

  /*Configure GPIO pin : PH6 */
  GPIO_InitStruct.Pin = GPIO_PIN_6;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOH, &GPIO_InitStruct);

}

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
```

```
  /* User can add his own implementation to report the HAL error return state
*/
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/
```

8.2.2 Vector de interrupciones

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f7xx_hal.h"
#include "stm32f7xx.h"
#include "stm32f7xx_it.h"

/* External variables --------------------------------------------------------*/
extern TIM_HandleTypeDef htim6;
extern void timer_6(void); //Función externa de interrupción
/******************************************************************************/
/*            Cortex-M7 Processor Interruption and Exception Handlers         */
/******************************************************************************/
/**
* @brief This function handles Non maskable interrupt.
*/
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */

  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
* @brief This function handles Hard fault interrupt.
*/
void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN HardFault_IRQn 1 */

  /* USER CODE END HardFault_IRQn 1 */
}

/**
* @brief This function handles Memory management fault.
*/
void MemManage_Handler(void)
{
  /* USER CODE BEGIN MemoryManagement_IRQn 0 */

  /* USER CODE END MemoryManagement_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN MemoryManagement_IRQn 1 */

  /* USER CODE END MemoryManagement_IRQn 1 */
}

/**
* @brief This function handles Pre-fetch fault, memory access fault.
*/
```

```
void BusFault_Handler(void)
{
  /* USER CODE BEGIN BusFault_IRQn 0 */

  /* USER CODE END BusFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN BusFault_IRQn 1 */

  /* USER CODE END BusFault_IRQn 1 */
}

/**
* @brief This function handles Undefined instruction or illegal state.
*/
void UsageFault_Handler(void)
{
  /* USER CODE BEGIN UsageFault_IRQn 0 */

  /* USER CODE END UsageFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN UsageFault_IRQn 1 */

  /* USER CODE END UsageFault_IRQn 1 */
}

/**
* @brief This function handles System service call via SWI instruction.
*/
void SVC_Handler(void)
{
  /* USER CODE BEGIN SVCall_IRQn 0 */

  /* USER CODE END SVCall_IRQn 0 */
  /* USER CODE BEGIN SVCall_IRQn 1 */

  /* USER CODE END SVCall_IRQn 1 */
}

/**
* @brief This function handles Debug monitor.
*/
void DebugMon_Handler(void)
{
  /* USER CODE BEGIN DebugMonitor_IRQn 0 */

  /* USER CODE END DebugMonitor_IRQn 0 */
  /* USER CODE BEGIN DebugMonitor_IRQn 1 */

  /* USER CODE END DebugMonitor_IRQn 1 */
}

/**
* @brief This function handles Pendable request for system service.
*/
void PendSV_Handler(void)
{
  /* USER CODE BEGIN PendSV_IRQn 0 */

  /* USER CODE END PendSV_IRQn 0 */
```

```
  /* USER CODE BEGIN PendSV_IRQn 1 */

  /* USER CODE END PendSV_IRQn 1 */
}

/**
* @brief This function handles System tick timer.
*/
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}

/******************************************************************************/
/* STM32F7xx Peripheral Interrupt Handlers
*/
/* Add here the Interrupt Handlers for the used peripherals.
*/
/* For the available peripheral interrupt handler names,
*/
/* please refer to the startup file (startup_stm32f7xx.s).
*/
/******************************************************************************/

/**
* @brief This function handles TIM6 global interrupt, DAC1 and DAC2 underrun
error interrupts.
*/
void TIM6_DAC_IRQHandler(void)
{
  /* USER CODE BEGIN TIM6_DAC_IRQn 0 */

  /* USER CODE END TIM6_DAC_IRQn 0 */
  HAL_TIM_IRQHandler(&htim6);
  /* USER CODE BEGIN TIM6_DAC_IRQn 1 */
  timer_6();//Llamada a la función del main/función de interrupción
  /* USER CODE END TIM6_DAC_IRQn 1 */
}
```

8.3 Ejemplo práctico UART

8.3.1 main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f7xx_hal.h"

UART_HandleTypeDef huart1;

uint8_t ask[40] = "Introduce un numero del 0 al 4: \n\r"; //Variable que
pregunta a usuario
uint8_t receive[1];          //Buffer para almacenar la respuesta
uint8_t flag = 0;            //Variable bandera para preguntar sólo una vez

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);

int main(void)
{
  /* Enable I-Cache-------------------------------------------------------------*/
  SCB_EnableICache();

  /* Enable D-Cache-------------------------------------------------------------*/
  SCB_EnableDCache();

  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();

  HAL_UART_Receive_IT(&huart1, receive, 1);//Inicialización de la UART para
recepción

  while (1)
  {
          if(flag == 0)                 //Si la bandera es cero
          {
            flag = 1;                   //La levantamos
            HAL_UART_Transmit(&huart1, ask, 40, 100);//Pedimos número por
pantalla
          }
  }
}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
```

```
RCC_ClkInitTypeDef RCC_ClkInitStruct;
RCC_PeriphCLKInitTypeDef PeriphClkInitStruct;

  /**Configure the main internal regulator output voltage
  */
__HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /**Initializes the CPU, AHB and APB busses clocks
  */
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
RCC_OscInitStruct.HSIState = RCC_HSI_ON;
RCC_OscInitStruct.HSICalibrationValue = 16;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
RCC_OscInitStruct.PLL.PLLM = 8;
RCC_OscInitStruct.PLL.PLLN = 216;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLQ = 2;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
  Error_Handler();
}

  /**Activate the Over-Drive mode
  */
if (HAL_PWREx_EnableOverDrive() != HAL_OK)
{
  Error_Handler();
}

  /**Initializes the CPU, AHB and APB busses clocks
  */
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                            |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7) != HAL_OK)
{
  Error_Handler();
}

PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_USART1;
PeriphClkInitStruct.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
{
  Error_Handler();
}

  /**Configure the Systick interrupt time
  */
HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

  /**Configure the Systick
  */
HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

/* SysTick_IRQn interrupt configuration */
```

```
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* USART1 init function */
static void MX_USART1_UART_Init(void)
{

  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOI_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_0, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, GPIO_PIN_RESET);

  /*Configure GPIO pins : PI3 PI2 PI0 */
  GPIO_InitStruct.Pin = GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);

  /*Configure GPIO pin : PH6 */
  GPIO_InitStruct.Pin = GPIO_PIN_6;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOH, &GPIO_InitStruct);

}
```

```
/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state
*/
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/
```

8.3.2 Vector de interrupciones

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f7xx_hal.h"
#include "stm32f7xx.h"
#include "stm32f7xx_it.h"

/* External variables --------------------------------------------------------*/
extern UART_HandleTypeDef huart1;

extern uint8_t receive[1];  //Variable externa buffer de datos
extern uint8_t flag;                    //Bandera para evitar envio continuo
de pregunta

/* Local variables and functions ---------------------------------------------*/
void maquina(void);                     //Función de máquina de estados
uint8_t error[50] = "El numero introducido no es correcto...\n\r"; //Mensaje
de error


/* Para facilitar la lectura y escritura del codigo, creo varios defines para
el encendido/apagado de LEDs*/
#define LED1ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_0, 1)
#define LED2ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_2, 1)
#define LED3ON HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3, 1)
#define LED4ON HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, 1)
#define LED1OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_0, 0)
#define LED2OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_2, 0)
#define LED3OFF HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3, 0)
#define LED4OFF HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, 0)

/******************************************************************************/
/*            Cortex-M7 Processor Interruption and Exception Handlers
*/
/******************************************************************************/

/**
* @brief This function handles Non maskable interrupt.
*/
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */

  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
* @brief This function handles Hard fault interrupt.
*/
void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN HardFault_IRQn 1 */

  /* USER CODE END HardFault_IRQn 1 */
```

```
}

/**
* @brief This function handles Memory management fault.
*/
void MemManage_Handler(void)
{
  /* USER CODE BEGIN MemoryManagement_IRQn 0 */

  /* USER CODE END MemoryManagement_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN MemoryManagement_IRQn 1 */

  /* USER CODE END MemoryManagement_IRQn 1 */
}

/**
* @brief This function handles Pre-fetch fault, memory access fault.
*/
void BusFault_Handler(void)
{
  /* USER CODE BEGIN BusFault_IRQn 0 */

  /* USER CODE END BusFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN BusFault_IRQn 1 */

  /* USER CODE END BusFault_IRQn 1 */
}

/**
* @brief This function handles Undefined instruction or illegal state.
*/
void UsageFault_Handler(void)
{
  /* USER CODE BEGIN UsageFault_IRQn 0 */

  /* USER CODE END UsageFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN UsageFault_IRQn 1 */

  /* USER CODE END UsageFault_IRQn 1 */
}

/**
* @brief This function handles System service call via SWI instruction.
*/
void SVC_Handler(void)
{
  /* USER CODE BEGIN SVCall_IRQn 0 */

  /* USER CODE END SVCall_IRQn 0 */
  /* USER CODE BEGIN SVCall_IRQn 1 */

  /* USER CODE END SVCall_IRQn 1 */
}
```

```
/**
* @brief This function handles Debug monitor.
*/
void DebugMon_Handler(void)
{
  /* USER CODE BEGIN DebugMonitor_IRQn 0 */

  /* USER CODE END DebugMonitor_IRQn 0 */
  /* USER CODE BEGIN DebugMonitor_IRQn 1 */

  /* USER CODE END DebugMonitor_IRQn 1 */
}

/**
* @brief This function handles Pendable request for system service.
*/
void PendSV_Handler(void)
{
  /* USER CODE BEGIN PendSV_IRQn 0 */

  /* USER CODE END PendSV_IRQn 0 */
  /* USER CODE BEGIN PendSV_IRQn 1 */

  /* USER CODE END PendSV_IRQn 1 */
}

/**
* @brief This function handles System tick timer.
*/
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}

/******************************************************************************/
/* STM32F7xx Peripheral Interrupt Handlers
*/
/* Add here the Interrupt Handlers for the used peripherals.
*/
/* For the available peripheral interrupt handler names,
*/
/* please refer to the startup file (startup_stm32f7xx.s).
*/
/******************************************************************************/

/**
* @brief This function handles RCC global interrupt.
*/
void RCC_IRQHandler(void)
{
  /* USER CODE BEGIN RCC_IRQn 0 */

  /* USER CODE END RCC_IRQn 0 */
  /* USER CODE BEGIN RCC_IRQn 1 */
```

```
  /* USER CODE END RCC_IRQn 1 */
}

/**
* @brief This function handles USART1 global interrupt.
*/
void USART1_IRQHandler(void)
{
  /* USER CODE BEGIN USART1_IRQn 0 */

  /* USER CODE END USART1_IRQn 0 */
  HAL_UART_IRQHandler(&huart1);
  /* USER CODE BEGIN USART1_IRQn 1 */
      maquina();//Iniciamos la función de la máquina de estados cada vez que
recibimos dato
  /* USER CODE END USART1_IRQn 1 */
}

void maquina(void) //Función máquina de estados
{
      switch(receive[0]) //En función del caracter recibido
      {
            case '0':
                  LED1OFF;
                  LED2OFF;
                  LED3OFF;
                  LED4OFF;
              flag = 0; //Bajamos la bandera otra vez
            break;

            case '1':
                  LED1ON;
                  LED2OFF;
                  LED3OFF;
                  LED4OFF;
              flag = 0; //Bajamos la bandera otra vez
            break;

            case '2':
                  LED1ON;
                  LED2ON;
                  LED3OFF;
                  LED4OFF;
            break;

            case '3':
                  LED1ON;
                  LED2ON;
                  LED3ON;
                  LED4OFF;
            break;

            case '4':
                  LED1ON;
                  LED2ON;
                  LED3ON;
                  LED4ON;
            break;

            default:    //Si el carácter introducido no es correcto
                  HAL_UART_Transmit(&huart1, error, 50, 100); //Transmitimos
error
```

```
            break;
      }

      HAL_UART_Receive_IT(&huart1, receive, 1); //Volvemos a activar la
recepcion
}
```

8.4 Ejemplo práctico ADC

8.4.1 main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f7xx_hal.h"

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc3;

TIM_HandleTypeDef htim6;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
uint8_t muestra[40];    //Variable que pregunta a usuario
uint8_t salto[5] = "mV\n\r";//Cadena de caracteres
int conv;               //Variable para la conversión
void lectura(void);     //Función que realiza la lectura del ADC
uint32_t conversion;    //Función que realiza la conversion
float voltaje;          //Variable donde se almacena el voltaje
/* USER CODE END PVze */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_ADC3_Init(void);
static void MX_TIM6_Init(void);

int main(void)
{

  /* Enable I-Cache-------------------------------------------------------------*/
  SCB_EnableICache();

  /* Enable D-Cache-------------------------------------------------------------*/
  SCB_EnableDCache();

  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  MX_ADC3_Init();
  MX_TIM6_Init();

HAL_TIM_Base_Start_IT(&htim6); //Inicialización del timer
  /* Infinite loop */
  while (1)
  {
  }
```

```
}

void lectura(void){
            HAL_ADC_Start(&hadc3);              //Inicializa el ADC
            conversion = HAL_ADC_GetValue(&hadc3);//Almacena en una var. el
valor del ADC
            voltaje = (float) conversion/819;   //Se realiza la conversión
            voltaje = (float) voltaje*1000;
            conv = (int) voltaje;               //Se coge la parte entera
            sprintf(muestra, "%d", conv);       //Se transforma a cadena de
caracteres
            HAL_UART_Transmit(&huart1, muestra, 40, 100);//Transmite por la
UART
            HAL_UART_Transmit(&huart1, salto, 5, 100);//Escribe mV y da un
salto de línea
}


/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct;

    /**Configure the main internal regulator output voltage
    */
  __HAL_RCC_PWR_CLK_ENABLE();

  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 216;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Activate the Over-Drive mode
    */
  if (HAL_PWREx_EnableOverDrive() != HAL_OK)
  {
    Error_Handler();
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
```

```
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7) != HAL_OK)
  {
    Error_Handler();
  }

  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_USART1;
  PeriphClkInitStruct.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* ADC3 init function */
static void MX_ADC3_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock, Resolution, Data
Alignment and number of conversion)
    */
  hadc3.Instance = ADC3;
  hadc3.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc3.Init.Resolution = ADC_RESOLUTION_12B;
  hadc3.Init.ScanConvMode = DISABLE;
  hadc3.Init.ContinuousConvMode = DISABLE;
  hadc3.Init.DiscontinuousConvMode = DISABLE;
  hadc3.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc3.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc3.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc3.Init.NbrOfConversion = 1;
  hadc3.Init.DMAContinuousRequests = DISABLE;
  hadc3.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc3) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure for the selected ADC regular channel its corresponding rank
in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_8;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
```

```
}

/* TIM6 init function */
static void MX_TIM6_Init(void)
{

  TIM_MasterConfigTypeDef sMasterConfig;

  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 25000;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 350;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }

}

/* USART1 init function */
static void MX_USART1_UART_Init(void)
{

  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
```

```
  __HAL_RCC_GPIOF_CLK_ENABLE();

}

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state
*/
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/
```

8.4.2 Vector de interrupciones

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f7xx_hal.h"
#include "stm32f7xx.h"
#include "stm32f7xx_it.h"

/* External variables --------------------------------------------------------*/
extern TIM_HandleTypeDef htim6;
extern void lectura(void);//Función externa del main para la lectura del ADC
/******************************************************************************/
/*            Cortex-M7 Processor Interruption and Exception Handlers
*/
/******************************************************************************/

/**
* @brief This function handles Non maskable interrupt.
*/
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */

  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
* @brief This function handles Hard fault interrupt.
*/
void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN HardFault_IRQn 1 */

  /* USER CODE END HardFault_IRQn 1 */
}

/**
* @brief This function handles Memory management fault.
*/
void MemManage_Handler(void)
{
  /* USER CODE BEGIN MemoryManagement_IRQn 0 */

  /* USER CODE END MemoryManagement_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN MemoryManagement_IRQn 1 */

  /* USER CODE END MemoryManagement_IRQn 1 */
}

/**
* @brief This function handles Pre-fetch fault, memory access fault.
*/
```

```
void BusFault_Handler(void)
{
  /* USER CODE BEGIN BusFault_IRQn 0 */

  /* USER CODE END BusFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN BusFault_IRQn 1 */

  /* USER CODE END BusFault_IRQn 1 */
}

/**
* @brief This function handles Undefined instruction or illegal state.
*/
void UsageFault_Handler(void)
{
  /* USER CODE BEGIN UsageFault_IRQn 0 */

  /* USER CODE END UsageFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN UsageFault_IRQn 1 */

  /* USER CODE END UsageFault_IRQn 1 */
}

/**
* @brief This function handles System service call via SWI instruction.
*/
void SVC_Handler(void)
{
  /* USER CODE BEGIN SVCall_IRQn 0 */

  /* USER CODE END SVCall_IRQn 0 */
  /* USER CODE BEGIN SVCall_IRQn 1 */

  /* USER CODE END SVCall_IRQn 1 */
}

/**
* @brief This function handles Debug monitor.
*/
void DebugMon_Handler(void)
{
  /* USER CODE BEGIN DebugMonitor_IRQn 0 */

  /* USER CODE END DebugMonitor_IRQn 0 */
  /* USER CODE BEGIN DebugMonitor_IRQn 1 */

  /* USER CODE END DebugMonitor_IRQn 1 */
}

/**
* @brief This function handles Pendable request for system service.
*/
void PendSV_Handler(void)
{
  /* USER CODE BEGIN PendSV_IRQn 0 */
```

```
  /* USER CODE END PendSV_IRQn 0 */
  /* USER CODE BEGIN PendSV_IRQn 1 */

  /* USER CODE END PendSV_IRQn 1 */
}

/**
* @brief This function handles System tick timer.
*/
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}

/******************************************************************************/
/* STM32F7xx Peripheral Interrupt Handlers
*/
/* Add here the Interrupt Handlers for the used peripherals.
*/
/* For the available peripheral interrupt handler names,
*/
/* please refer to the startup file (startup_stm32f7xx.s).
*/
/******************************************************************************/

/**
* @brief This function handles RCC global interrupt.
*/
void RCC_IRQHandler(void)
{
  /* USER CODE BEGIN RCC_IRQn 0 */

  /* USER CODE END RCC_IRQn 0 */
  /* USER CODE BEGIN RCC_IRQn 1 */

  /* USER CODE END RCC_IRQn 1 */
}

/**
* @brief This function handles TIM6 global interrupt, DAC1 and DAC2 underrun
error interrupts.
*/
void TIM6_DAC_IRQHandler(void)
{
  /* USER CODE BEGIN TIM6_DAC_IRQn 0 */

  /* USER CODE END TIM6_DAC_IRQn 0 */
  HAL_TIM_IRQHandler(&htim6);
  /* USER CODE BEGIN TIM6_DAC_IRQn 1 */
  lectura();//Realiza la lectura del ADC
  /* USER CODE END TIM6_DAC_IRQn 1 */
}
```

8.5 Ejemplo práctico LCD

8.5.1 main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f7xx_hal.h"
#include "stm32746g_discovery.h"              //Librería de la placa
#include "stm32746g_discovery_lcd.h"          //Librería del LCD
#include "stm32746g_discovery_sdram.h"        //Librería de la SDRAM

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

DMA2D_HandleTypeDef hdma2d;

I2C_HandleTypeDef hi2c3;

LTDC_HandleTypeDef hltdc;

SPI_HandleTypeDef hspi5;

UART_HandleTypeDef huart6;

SDRAM_HandleTypeDef hsdram1;

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_DMA2D_Init(void);
static void MX_FMC_Init(void);
static void MX_I2C3_Init(void);
static void MX_LTDC_Init(void);
static void MX_SPI5_Init(void);
static void MX_USART6_UART_Init(void);
static void MX_ADC1_Init(void);


/* Private function prototypes -----------------------------------------------*/
void Interface(void);         //Función que dibuja la interfaz
void conversion(void);        //Función que realiza la conversión

uint32_t conv;
                              //Buffer del ADC
uint32_t Voltaje;             //Variable convertida del buffer
uint8_t Medidas[5];           //Vector de impresión

int main(void)
{

  /* Enable I-Cache---------------------------------------------------------*/
  SCB_EnableICache();

  /* Enable D-Cache---------------------------------------------------------*/
  SCB_EnableDCache();

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();
```

```
  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA2D_Init();
  MX_FMC_Init();
  MX_I2C3_Init();
  MX_LTDC_Init();
  MX_SPI5_Init();
  MX_USART6_UART_Init();
  MX_ADC1_Init();

      BSP_LCD_Init();               //Inicializa la pantalla
      BSP_LCD_LayerDefaultInit(1,SDRAM_DEVICE_ADDR);//Configuramos la pantala
a 32bits
      BSP_LCD_SelectLayer(1);       //Seleccionamos la capa
      BSP_LCD_Clear(LCD_COLOR_BLACK);//Limpiamos la pantalla
      Interface();                  //Llamada a la función Interface

  while (1)
  {
          conversion();             //Llamada a la función conversión
          HAL_Delay(5);             //Delay
          BSP_LCD_Clear(LCD_COLOR_LIGHTGRAY); //Limpia la pantalla
  }
}

void Interface(void)
{
      BSP_LCD_SelectLayer(1);             //Seleccionamos capa
      BSP_LCD_Clear(LCD_COLOR_LIGHTGRAY); //Limpiamos y pintamos la pantalla
      BSP_LCD_SetBackColor(LCD_COLOR_LIGHTGRAY);//Color de fondo para el
texto
      BSP_LCD_SetTextColor(LCD_COLOR_BLACK);    //Color del texto
      BSP_LCD_DrawLine(24,24,456,24);           //Dibujamos un marco
      BSP_LCD_DrawLine(456,24,456,272-24);
      BSP_LCD_DrawLine(456,272-24,24,272-24);
      BSP_LCD_DrawLine(24,272-24,24,24);
}

void conversion(void)
{
      Interface();
      BSP_LCD_SelectLayer(1);
            //Seleccionamos capa
      BSP_LCD_SetBackColor(LCD_COLOR_LIGHTGRAY);//Color de fondo para el
texto
      HAL_ADC_Start(&hadc1);                    //Iniciamos ADC

      conv = HAL_ADC_GetValue(&hadc1);          //Obtenemos el valor
      Voltaje = conv/12.4;                      //Realizamos conversión
      Medidas[4] = 'V';                         //Etiqueta de VOltios
      Medidas[3] = 0x30 + (Voltaje%10);         //Centésimas
      Medidas[2] = 0x30 + (Voltaje/10)%10;      //Décimas
      Medidas[1] = '.';                         //Punto
      Medidas[0] = 0x30 + (Voltaje/100);        //Unidades
      BSP_LCD_DisplayStringAt(50,120, Medidas,LEFT_MODE);//Imprimimos valor

      int x = conv*100/4095;                    //Realiza una conversión
```

```
      BSP_LCD_FillRect(300, 120, x, 30);        //Dibuja un rectángulo
proporcional a la medida
}


/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct;

    /**Configure the main internal regulator output voltage
    */
  __HAL_RCC_PWR_CLK_ENABLE();

  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 216;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Activate the Over-Drive mode
    */
  if (HAL_PWREx_EnableOverDrive() != HAL_OK)
  {
    Error_Handler();
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7) != HAL_OK)
  {
    Error_Handler();
  }

  PeriphClkInitStruct.PeriphClockSelection =
RCC_PERIPHCLK_LTDC|RCC_PERIPHCLK_USART6
                              |RCC_PERIPHCLK_I2C3;
  PeriphClkInitStruct.PLLSAI.PLLSAIN = 50;
  PeriphClkInitStruct.PLLSAI.PLLSAIR = 2;
```

```
  PeriphClkInitStruct.PLLSAI.PLLSAIQ = 2;
  PeriphClkInitStruct.PLLSAI.PLLSAIP = RCC_PLLSAIP_DIV2;
  PeriphClkInitStruct.PLLSAIDivQ = 1;
  PeriphClkInitStruct.PLLSAIDivR = RCC_PLLSAIDIVR_2;
  PeriphClkInitStruct.Usart6ClockSelection = RCC_USART6CLKSOURCE_PCLK2;
  PeriphClkInitStruct.I2c3ClockSelection = RCC_I2C3CLKSOURCE_PCLK1;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* ADC1 init function */
static void MX_ADC1_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock, Resolution, Data
Alignment and number of conversion)
    */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure for the selected ADC regular channel its corresponding rank
in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* DMA2D init function */
```

```
static void MX_DMA2D_Init(void)
{

  hdma2d.Instance = DMA2D;
  hdma2d.Init.Mode = DMA2D_M2M;
  hdma2d.Init.ColorMode = DMA2D_OUTPUT_ARGB8888;
  hdma2d.Init.OutputOffset = 0;
  hdma2d.LayerCfg[1].InputOffset = 0;
  hdma2d.LayerCfg[1].InputColorMode = DMA2D_INPUT_ARGB8888;
  hdma2d.LayerCfg[1].AlphaMode = DMA2D_NO_MODIF_ALPHA;
  hdma2d.LayerCfg[1].InputAlpha = 0;
  if (HAL_DMA2D_Init(&hdma2d) != HAL_OK)
  {
    Error_Handler();
  }

  if (HAL_DMA2D_ConfigLayer(&hdma2d, 1) != HAL_OK)
  {
    Error_Handler();
  }

}

/* I2C3 init function */
static void MX_I2C3_Init(void)
{

  hi2c3.Instance = I2C3;
  hi2c3.Init.Timing = 0x20404768;
  hi2c3.Init.OwnAddress1 = 0;
  hi2c3.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c3.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c3.Init.OwnAddress2 = 0;
  hi2c3.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c3.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c3.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c3) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure Analogue filter
    */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c3, I2C_ANALOGFILTER_ENABLE) !=
HAL_OK)
  {
    Error_Handler();
  }

    /**Configure Digital filter
    */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c3, 0) != HAL_OK)
  {
    Error_Handler();
  }

}

/* LTDC init function */
static void MX_LTDC_Init(void)
{
```

```
LTDC_LayerCfgTypeDef pLayerCfg;
LTDC_LayerCfgTypeDef pLayerCfg1;

hltdc.Instance = LTDC;
hltdc.Init.HSPolarity = LTDC_HSPOLARITY_AL;
hltdc.Init.VSPolarity = LTDC_VSPOLARITY_AL;
hltdc.Init.DEPolarity = LTDC_DEPOLARITY_AL;
hltdc.Init.PCPolarity = LTDC_PCPOLARITY_IPC;
hltdc.Init.HorizontalSync = 7;
hltdc.Init.VerticalSync = 3;
hltdc.Init.AccumulatedHBP = 14;
hltdc.Init.AccumulatedVBP = 5;
hltdc.Init.AccumulatedActiveW = 654;
hltdc.Init.AccumulatedActiveH = 485;
hltdc.Init.TotalWidth = 660;
hltdc.Init.TotalHeigh = 487;
hltdc.Init.Backcolor.Blue = 0;
hltdc.Init.Backcolor.Green = 0;
hltdc.Init.Backcolor.Red = 0;
if (HAL_LTDC_Init(&hltdc) != HAL_OK)
{
  Error_Handler();
}

pLayerCfg.WindowX0 = 0;
pLayerCfg.WindowX1 = 0;
pLayerCfg.WindowY0 = 0;
pLayerCfg.WindowY1 = 0;
pLayerCfg.PixelFormat = LTDC_PIXEL_FORMAT_ARGB8888;
pLayerCfg.Alpha = 0;
pLayerCfg.Alpha0 = 0;
pLayerCfg.BlendingFactor1 = LTDC_BLENDING_FACTOR1_CA;
pLayerCfg.BlendingFactor2 = LTDC_BLENDING_FACTOR2_CA;
pLayerCfg.FBStartAdress = 0;
pLayerCfg.ImageWidth = 0;
pLayerCfg.ImageHeight = 0;
pLayerCfg.Backcolor.Blue = 0;
pLayerCfg.Backcolor.Green = 0;
pLayerCfg.Backcolor.Red = 0;
if (HAL_LTDC_ConfigLayer(&hltdc, &pLayerCfg, 0) != HAL_OK)
{
  Error_Handler();
}

pLayerCfg1.WindowX0 = 0;
pLayerCfg1.WindowX1 = 0;
pLayerCfg1.WindowY0 = 0;
pLayerCfg1.WindowY1 = 0;
pLayerCfg1.PixelFormat = LTDC_PIXEL_FORMAT_ARGB8888;
pLayerCfg1.Alpha = 0;
pLayerCfg1.Alpha0 = 0;
pLayerCfg1.BlendingFactor1 = LTDC_BLENDING_FACTOR1_CA;
pLayerCfg1.BlendingFactor2 = LTDC_BLENDING_FACTOR2_CA;
pLayerCfg1.FBStartAdress = 0;
pLayerCfg1.ImageWidth = 0;
pLayerCfg1.ImageHeight = 0;
pLayerCfg1.Backcolor.Blue = 0;
pLayerCfg1.Backcolor.Green = 0;
pLayerCfg1.Backcolor.Red = 0;
if (HAL_LTDC_ConfigLayer(&hltdc, &pLayerCfg1, 1) != HAL_OK)
{
  Error_Handler();
}
```

```
}

/* SPI5 init function */
static void MX_SPI5_Init(void)
{

  hspi5.Instance = SPI5;
  hspi5.Init.Mode = SPI_MODE_MASTER;
  hspi5.Init.Direction = SPI_DIRECTION_2LINES;
  hspi5.Init.DataSize = SPI_DATASIZE_4BIT;
  hspi5.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi5.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi5.Init.NSS = SPI_NSS_SOFT;
  hspi5.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi5.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi5.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi5.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi5.Init.CRCPolynomial = 7;
  hspi5.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi5.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
  if (HAL_SPI_Init(&hspi5) != HAL_OK)
  {
    Error_Handler();
  }

}

/* USART6 init function */
static void MX_USART6_UART_Init(void)
{

  huart6.Instance = USART6;
  huart6.Init.BaudRate = 115200;
  huart6.Init.WordLength = UART_WORDLENGTH_7B;
  huart6.Init.StopBits = UART_STOPBITS_1;
  huart6.Init.Parity = UART_PARITY_NONE;
  huart6.Init.Mode = UART_MODE_TX_RX;
  huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart6.Init.OverSampling = UART_OVERSAMPLING_16;
  huart6.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart6.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart6) != HAL_OK)
  {
    Error_Handler();
  }

}
/* FMC initialization function */
static void MX_FMC_Init(void)
{
  FMC_SDRAM_TimingTypeDef SdramTiming;

  /** Perform the SDRAM1 memory initialization sequence
  */
  hsdram1.Instance = FMC_SDRAM_DEVICE;
  /* hsdram1.Init */
  hsdram1.Init.SDBank = FMC_SDRAM_BANK2;
  hsdram1.Init.ColumnBitsNumber = FMC_SDRAM_COLUMN_BITS_NUM_8;
  hsdram1.Init.RowBitsNumber = FMC_SDRAM_ROW_BITS_NUM_13;
  hsdram1.Init.MemoryDataWidth = FMC_SDRAM_MEM_BUS_WIDTH_16;
  hsdram1.Init.InternalBankNumber = FMC_SDRAM_INTERN_BANKS_NUM_4;
```

```
  hsdram1.Init.CASLatency = FMC_SDRAM_CAS_LATENCY_1;
  hsdram1.Init.WriteProtection = FMC_SDRAM_WRITE_PROTECTION_DISABLE;
  hsdram1.Init.SDClockPeriod = FMC_SDRAM_CLOCK_DISABLE;
  hsdram1.Init.ReadBurst = FMC_SDRAM_RBURST_DISABLE;
  hsdram1.Init.ReadPipeDelay = FMC_SDRAM_RPIPE_DELAY_0;
  /* SdramTiming */
  SdramTiming.LoadToActiveDelay = 16;
  SdramTiming.ExitSelfRefreshDelay = 16;
  SdramTiming.SelfRefreshTime = 16;
  SdramTiming.RowCycleDelay = 16;
  SdramTiming.WriteRecoveryTime = 16;
  SdramTiming.RPDelay = 16;
  SdramTiming.RCDDelay = 16;

  if (HAL_SDRAM_Init(&hsdram1, &SdramTiming) != HAL_OK)
  {
    Error_Handler();
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOG_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOI_CLK_ENABLE();
  __HAL_RCC_GPIOK_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOJ_CLK_ENABLE();

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state
*/
  while(1)
  {
  }
```

```
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/
```

8.6 Ejemplo práctico Touchscreen

8.6.1 main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f7xx_hal.h"
#include "stm32746g_discovery_ts.h"            //Librería del touchscreen
#include "stm32746g_discovery_sdram.h"         //Librería de la SDRAM
#include "stm32746g_discovery_lcd.h"           //Librería del LCD
#include "stm32746g_discovery.h"               //Librería de la placa
#include "ft5336.h"


/* Private variables ---------------------------------------------------------*/

DMA2D_HandleTypeDef hdma2d;

I2C_HandleTypeDef hi2c3;

LTDC_HandleTypeDef hltdc;

SPI_HandleTypeDef hspi5;

UART_HandleTypeDef huart6;

SDRAM_HandleTypeDef hsdram1;

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_DMA2D_Init(void);
static void MX_FMC_Init(void);
static void MX_LTDC_Init(void);
static void MX_SPI5_Init(void);
static void MX_USART6_UART_Init(void);
static void MX_I2C3_Init(void);


int main(void)
{

  /* Enable I-Cache---------------------------------------------------------*/
  SCB_EnableICache();

  /* Enable D-Cache---------------------------------------------------------*/
  SCB_EnableDCache();

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA2D_Init();
  MX_FMC_Init();
```

```
  MX_LTDC_Init();
  MX_SPI5_Init();
  MX_USART6_UART_Init();
  MX_I2C3_Init();

  /* USER CODE BEGIN 2 */
      BSP_LCD_Init();

      //Inicializa la pantalla
      BSP_LCD_LayerDefaultInit(1,SDRAM_DEVICE_ADDR); //Configuramos la
pantala a 32bits
      BSP_LCD_SelectLayer(1);             //Seleccionamos la capa
      BSP_LCD_Clear(LCD_COLOR_WHITE);     //Limpiamos la pantalla

      BSP_TS_Init(BSP_LCD_GetXSize(),BSP_LCD_GetYSize());  //Habilita el
touchscreen de toda la pantalla
      BSP_TS_ITConfig();                  //La habilita como interrupción

  while (1)
  {
  }
}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct;

    /**Configure the main internal regulator output voltage
    */
  __HAL_RCC_PWR_CLK_ENABLE();

  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 216;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Activate the Over-Drive mode
    */
  if (HAL_PWREx_EnableOverDrive() != HAL_OK)
  {
    Error_Handler();
  }

    /**Initializes the CPU, AHB and APB busses clocks
```

```
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7) != HAL_OK)
  {
    Error_Handler();
  }

  PeriphClkInitStruct.PeriphClockSelection =
RCC_PERIPHCLK_LTDC|RCC_PERIPHCLK_USART6
                              |RCC_PERIPHCLK_I2C3;
  PeriphClkInitStruct.PLLSAI.PLLSAIN = 50;
  PeriphClkInitStruct.PLLSAI.PLLSAIR = 2;
  PeriphClkInitStruct.PLLSAI.PLLSAIQ = 2;
  PeriphClkInitStruct.PLLSAI.PLLSAIP = RCC_PLLSAIP_DIV2;
  PeriphClkInitStruct.PLLSAIDivQ = 1;
  PeriphClkInitStruct.PLLSAIDivR = RCC_PLLSAIDIVR_2;
  PeriphClkInitStruct.Usart6ClockSelection = RCC_USART6CLKSOURCE_PCLK2;
  PeriphClkInitStruct.I2c3ClockSelection = RCC_I2C3CLKSOURCE_PCLK1;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* DMA2D init function */
static void MX_DMA2D_Init(void)
{

  hdma2d.Instance = DMA2D;
  hdma2d.Init.Mode = DMA2D_M2M;
  hdma2d.Init.ColorMode = DMA2D_OUTPUT_ARGB8888;
  hdma2d.Init.OutputOffset = 0;
  hdma2d.LayerCfg[1].InputOffset = 0;
  hdma2d.LayerCfg[1].InputColorMode = DMA2D_INPUT_ARGB8888;
  hdma2d.LayerCfg[1].AlphaMode = DMA2D_NO_MODIF_ALPHA;
  hdma2d.LayerCfg[1].InputAlpha = 0;
  if (HAL_DMA2D_Init(&hdma2d) != HAL_OK)
  {
    Error_Handler();
  }

  if (HAL_DMA2D_ConfigLayer(&hdma2d, 1) != HAL_OK)
  {
    Error_Handler();
  }
```

```
}

/* I2C3 init function */
static void MX_I2C3_Init(void)
{

  hi2c3.Instance = I2C3;
  hi2c3.Init.Timing = 0x00303D5B;
  hi2c3.Init.OwnAddress1 = 0;
  hi2c3.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c3.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c3.Init.OwnAddress2 = 0;
  hi2c3.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c3.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c3.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c3) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure Analogue filter
    */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c3, I2C_ANALOGFILTER_ENABLE) !=
HAL_OK)
  {
    Error_Handler();
  }

    /**Configure Digital filter
    */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c3, 0) != HAL_OK)
  {
    Error_Handler();
  }

}

/* LTDC init function */
static void MX_LTDC_Init(void)
{

  LTDC_LayerCfgTypeDef pLayerCfg;
  LTDC_LayerCfgTypeDef pLayerCfg1;

  hltdc.Instance = LTDC;
  hltdc.Init.HSPolarity = LTDC_HSPOLARITY_AL;
  hltdc.Init.VSPolarity = LTDC_VSPOLARITY_AL;
  hltdc.Init.DEPolarity = LTDC_DEPOLARITY_AL;
  hltdc.Init.PCPolarity = LTDC_PCPOLARITY_IPC;
  hltdc.Init.HorizontalSync = 7;
  hltdc.Init.VerticalSync = 3;
  hltdc.Init.AccumulatedHBP = 14;
  hltdc.Init.AccumulatedVBP = 5;
  hltdc.Init.AccumulatedActiveW = 654;
  hltdc.Init.AccumulatedActiveH = 485;
  hltdc.Init.TotalWidth = 660;
  hltdc.Init.TotalHeigh = 487;
  hltdc.Init.Backcolor.Blue = 0;
  hltdc.Init.Backcolor.Green = 0;
  hltdc.Init.Backcolor.Red = 0;
  if (HAL_LTDC_Init(&hltdc) != HAL_OK)
  {
```

```
    Error_Handler();
  }

  pLayerCfg.WindowX0 = 0;
  pLayerCfg.WindowX1 = 0;
  pLayerCfg.WindowY0 = 0;
  pLayerCfg.WindowY1 = 0;
  pLayerCfg.PixelFormat = LTDC_PIXEL_FORMAT_ARGB8888;
  pLayerCfg.Alpha = 0;
  pLayerCfg.Alpha0 = 0;
  pLayerCfg.BlendingFactor1 = LTDC_BLENDING_FACTOR1_CA;
  pLayerCfg.BlendingFactor2 = LTDC_BLENDING_FACTOR2_CA;
  pLayerCfg.FBStartAdress = 0;
  pLayerCfg.ImageWidth = 0;
  pLayerCfg.ImageHeight = 0;
  pLayerCfg.Backcolor.Blue = 0;
  pLayerCfg.Backcolor.Green = 0;
  pLayerCfg.Backcolor.Red = 0;
  if (HAL_LTDC_ConfigLayer(&hltdc, &pLayerCfg, 0) != HAL_OK)
  {
    Error_Handler();
  }

  pLayerCfg1.WindowX0 = 0;
  pLayerCfg1.WindowX1 = 0;
  pLayerCfg1.WindowY0 = 0;
  pLayerCfg1.WindowY1 = 0;
  pLayerCfg1.PixelFormat = LTDC_PIXEL_FORMAT_ARGB8888;
  pLayerCfg1.Alpha = 0;
  pLayerCfg1.Alpha0 = 0;
  pLayerCfg1.BlendingFactor1 = LTDC_BLENDING_FACTOR1_CA;
  pLayerCfg1.BlendingFactor2 = LTDC_BLENDING_FACTOR2_CA;
  pLayerCfg1.FBStartAdress = 0;
  pLayerCfg1.ImageWidth = 0;
  pLayerCfg1.ImageHeight = 0;
  pLayerCfg1.Backcolor.Blue = 0;
  pLayerCfg1.Backcolor.Green = 0;
  pLayerCfg1.Backcolor.Red = 0;
  if (HAL_LTDC_ConfigLayer(&hltdc, &pLayerCfg1, 1) != HAL_OK)
  {
    Error_Handler();
  }

}

/* SPI5 init function */
static void MX_SPI5_Init(void)
{

  hspi5.Instance = SPI5;
  hspi5.Init.Mode = SPI_MODE_MASTER;
  hspi5.Init.Direction = SPI_DIRECTION_2LINES;
  hspi5.Init.DataSize = SPI_DATASIZE_4BIT;
  hspi5.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi5.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi5.Init.NSS = SPI_NSS_SOFT;
  hspi5.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi5.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi5.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi5.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi5.Init.CRCPolynomial = 7;
  hspi5.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi5.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
```

```
  if (HAL_SPI_Init(&hspi5) != HAL_OK)
  {
    Error_Handler();
  }

}

/* USART6 init function */
static void MX_USART6_UART_Init(void)
{

  huart6.Instance = USART6;
  huart6.Init.BaudRate = 115200;
  huart6.Init.WordLength = UART_WORDLENGTH_7B;
  huart6.Init.StopBits = UART_STOPBITS_1;
  huart6.Init.Parity = UART_PARITY_NONE;
  huart6.Init.Mode = UART_MODE_TX_RX;
  huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart6.Init.OverSampling = UART_OVERSAMPLING_16;
  huart6.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart6.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart6) != HAL_OK)
  {
    Error_Handler();
  }

}
/* FMC initialization function */
static void MX_FMC_Init(void)
{
  FMC_SDRAM_TimingTypeDef SdramTiming;

  /** Perform the SDRAM1 memory initialization sequence
  */
  hsdram1.Instance = FMC_SDRAM_DEVICE;
  /* hsdram1.Init */
  hsdram1.Init.SDBank = FMC_SDRAM_BANK2;
  hsdram1.Init.ColumnBitsNumber = FMC_SDRAM_COLUMN_BITS_NUM_8;
  hsdram1.Init.RowBitsNumber = FMC_SDRAM_ROW_BITS_NUM_12;
  hsdram1.Init.MemoryDataWidth = FMC_SDRAM_MEM_BUS_WIDTH_16;
  hsdram1.Init.InternalBankNumber = FMC_SDRAM_INTERN_BANKS_NUM_4;
  hsdram1.Init.CASLatency = FMC_SDRAM_CAS_LATENCY_1;
  hsdram1.Init.WriteProtection = FMC_SDRAM_WRITE_PROTECTION_DISABLE;
  hsdram1.Init.SDClockPeriod = FMC_SDRAM_CLOCK_DISABLE;
  hsdram1.Init.ReadBurst = FMC_SDRAM_RBURST_DISABLE;
  hsdram1.Init.ReadPipeDelay = FMC_SDRAM_RPIPE_DELAY_0;
  /* SdramTiming */
  SdramTiming.LoadToActiveDelay = 16;
  SdramTiming.ExitSelfRefreshDelay = 16;
  SdramTiming.SelfRefreshTime = 16;
  SdramTiming.RowCycleDelay = 16;
  SdramTiming.WriteRecoveryTime = 16;
  SdramTiming.RPDelay = 16;
  SdramTiming.RCDDelay = 16;

  if (HAL_SDRAM_Init(&hsdram1, &SdramTiming) != HAL_OK)
  {
    Error_Handler();
  }

}
```

```
/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOG_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOI_CLK_ENABLE();
  __HAL_RCC_GPIOK_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();

}

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state
*/
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */

}

#endif

/**
```

```
  * @}
  */

/**
  * @}
*/
```

8.6.2 Vector de interrupciones

```
/* Includes ------------------------------------------------------------------
-*/
#include "stm32f7xx_hal.h"
#include "stm32f7xx.h"
#include "stm32f7xx_it.h"
#include "stm32746g_discovery_ts.h"
#include "stm32746g_discovery_sdram.h"
#include "stm32746g_discovery_lcd.h"
#include "stm32746g_discovery.h"
#include "ft5336.h"

TS_StateTypeDef *Posicion;          //Puntero para la posición

/* External variables ---------------------------------------------------------*/

/******************************************************************************/
/*            Cortex-M7 Processor Interruption and Exception Handlers
*/
/******************************************************************************/

/**
* @brief This function handles Non maskable interrupt.
*/
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */

  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
* @brief This function handles Hard fault interrupt.
*/
void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN HardFault_IRQn 1 */

  /* USER CODE END HardFault_IRQn 1 */
}

/**
* @brief This function handles Memory management fault.
*/
void MemManage_Handler(void)
```

```
{
  /* USER CODE BEGIN MemoryManagement_IRQn 0 */

  /* USER CODE END MemoryManagement_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN MemoryManagement_IRQn 1 */

  /* USER CODE END MemoryManagement_IRQn 1 */
}

/**
* @brief This function handles Pre-fetch fault, memory access fault.
*/
void BusFault_Handler(void)
{
  /* USER CODE BEGIN BusFault_IRQn 0 */

  /* USER CODE END BusFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN BusFault_IRQn 1 */

  /* USER CODE END BusFault_IRQn 1 */
}

/**
* @brief This function handles Undefined instruction or illegal state.
*/
void UsageFault_Handler(void)
{
  /* USER CODE BEGIN UsageFault_IRQn 0 */

  /* USER CODE END UsageFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN UsageFault_IRQn 1 */

  /* USER CODE END UsageFault_IRQn 1 */
}

/**
* @brief This function handles System service call via SWI instruction.
*/
void SVC_Handler(void)
{
  /* USER CODE BEGIN SVCall_IRQn 0 */

  /* USER CODE END SVCall_IRQn 0 */
  /* USER CODE BEGIN SVCall_IRQn 1 */

  /* USER CODE END SVCall_IRQn 1 */
}

/**
* @brief This function handles Debug monitor.
*/
void DebugMon_Handler(void)
{
  /* USER CODE BEGIN DebugMonitor_IRQn 0 */
```

```
  /* USER CODE END DebugMonitor_IRQn 0 */
  /* USER CODE BEGIN DebugMonitor_IRQn 1 */

  /* USER CODE END DebugMonitor_IRQn 1 */
}

/**
* @brief This function handles Pendable request for system service.
*/
void PendSV_Handler(void)
{
  /* USER CODE BEGIN PendSV_IRQn 0 */

  /* USER CODE END PendSV_IRQn 0 */
  /* USER CODE BEGIN PendSV_IRQn 1 */

  /* USER CODE END PendSV_IRQn 1 */
}

/**
* @brief This function handles System tick timer.
*/
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}

void EXTI15_10_IRQHandler(void)     //Handler asociado a la pantalla táctil
{
      HAL_GPIO_EXTI_IRQHandler(TS_INT_PIN);//Función que controla la
intierrupción
      HAL_GPIO_EXTI_Callback(TS_INT_PIN); //Función callback
}


void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) //Callback asociado a la
interrupción que se produce con un toque
{
      BSP_TS_GetState(Posicion);//Función que detecta la posición del toque
      if(Posicion -> touchY[0] < 15 ||
         Posicion -> touchX[0] < 15 ||
         Posicion -> touchY[0] > 260||
         Posicion -> touchX[0] > 465)//Para evitar errores, no se pinta en la
frontera exterior
      {
            return;
      }
      BSP_LCD_SetTextColor(LCD_COLOR_RED);      //Elige el color
      BSP_LCD_FillCircle(Posicion ->touchX[0],Posicion -> touchY[0],7);
//Dibuja donde se toca
}

/************************************************************************/
```

```
/* STM32F7xx Peripheral Interrupt Handlers
*/
/* Add here the Interrupt Handlers for the used peripherals.
*/
/* For the available peripheral interrupt handler names,
*/
/* please refer to the startup file (startup_stm32f7xx.s).
*/
/******************************************************************************/
```