

Características generales del sistema de desarrollo con el microcontrolador STM32F103C8T6

En el mercado hay infinidad de microcontroladores para diferentes propósitos. Un microcontrolador se escoge por velocidad, por cantidad de puertos seriales (Uart) para conexión con periféricos, por cantidad de bits del convertidor analógico a digital, por cantidad de pines de entrada y salida, por características de consumo de potencia y alternativas de dormido para bajo consumo, por hardware especializado, como procesamiento de datos, internet en la pastilla, entre muchas otras características. Pero también el precio es importante, sobre todo cuando queremos invertir un tiempo considerable en usar una pastilla en varios propósitos. Dentro del precio se puede incluir, no solo el precio de la pastilla, si no también el precio del programador, que puede ser un hardware adicional, y el precio del programa de compilación. Una de las pastillas más usadas hoy en día es la STM32F103C8T6 que viene en un módulo con la disposición de pines necesarios para empezar a trabajar en un proyecto. Se encuentran en tres versiones, una roja, una azul y una negra (red, blue & black pill). Este libro se enfoca en el uso de las pastillas azul y negra, que se muestran en la siguiente figura, aunque las tres usan el mismo microcontrolador, además que se encuentran en el mercado muchas otras tarjetas que usan el mismo microcontrolador, pero tienen características diferentes.

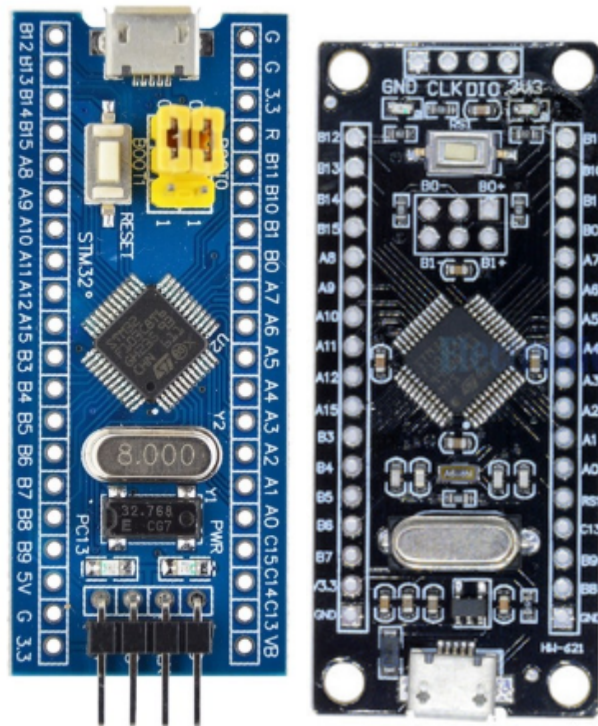
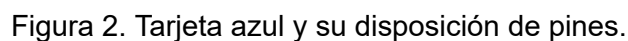


Figura 1. Pastillas azul y negra con el STM32F103C8T6.

El diagrama esquemático y la foto de la tarjeta azul se muestra en las figuras 2, 3 y 4.



La conexión del microcontrolador se presenta en el diagrama esquemático de la figura 3.

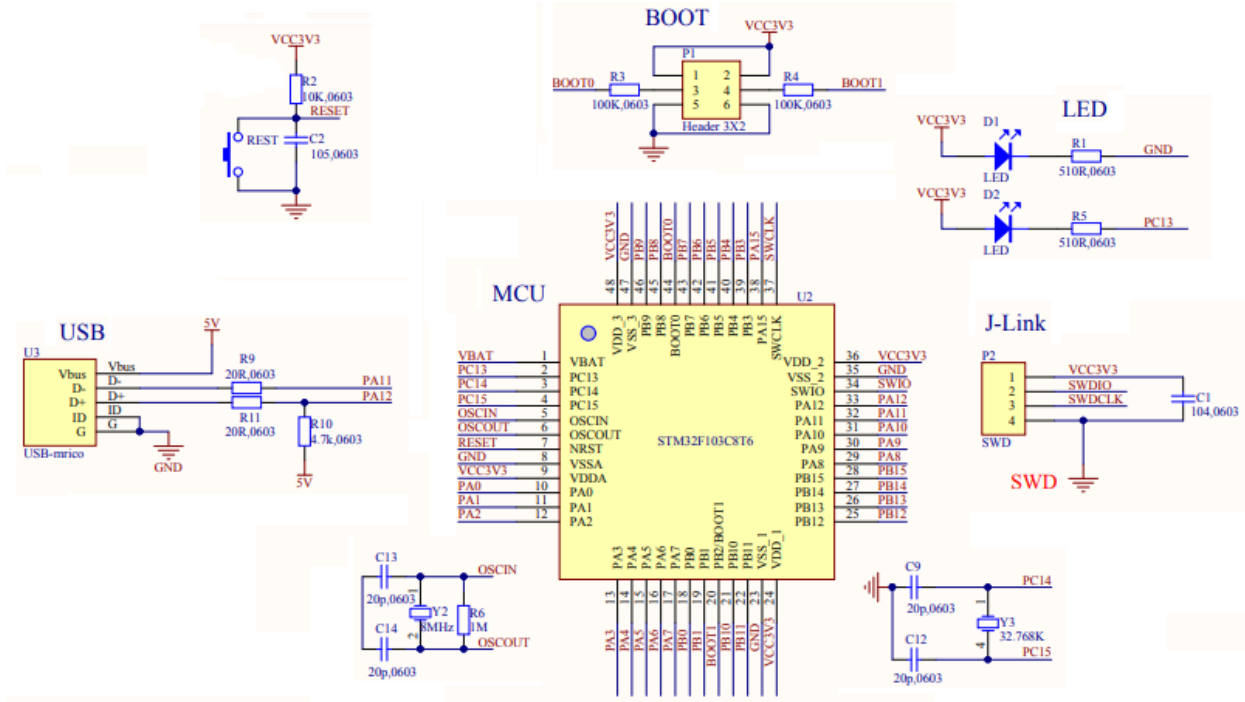


Figura 3. Diagrama esquemático del circuito de la tarjeta azul.
El circuito de potencia se presenta en la figura 4.

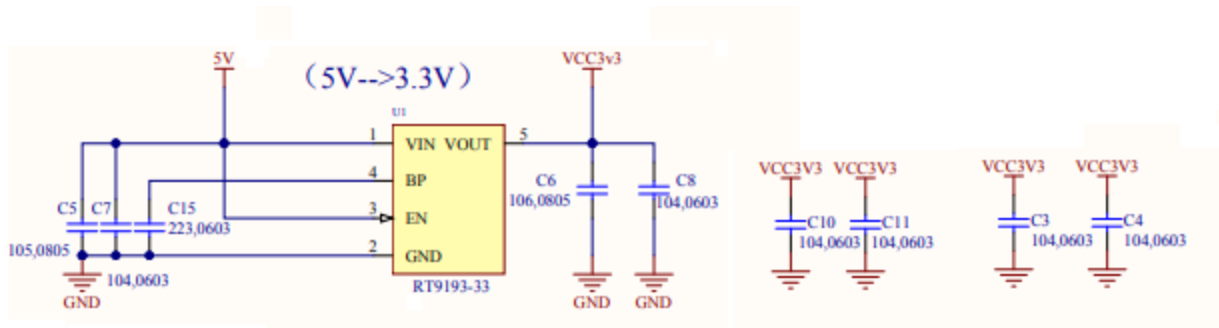


Figura 4. Diagrama esquemático del circuito de potencia de la tarjeta azul.

Para el módulo negro, la imagen de la tarjeta y del diagrama esquemático se muestran en las figuras 5, 6 y 7.

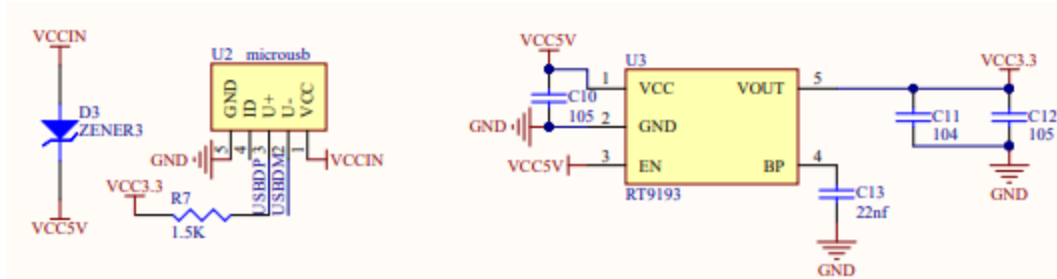


Figura 7. Diagrama esquemático del circuito de potencia de la pastilla negra.

Las diferencias entre las dos tarjetas son mínimas, pues usan el mismo microcontrolador y los mismos circuitos de regulación de voltaje. En resumen las diferencias de las tarjetas azul y negra se muestran a continuación:

1. Resistencia de USB en tarjeta azul es de 4.7k y en la tarjeta negra es de 1.5 K. En la tarjeta azul se debe cambiar el valor de esta resistencia.
2. Disposición de entrada de voltaje para alimentación de memoria en tarjeta azul, pero no en tarjeta negra, Vbat.
3. Disposición de entrada de voltaje de +5 voltios en tarjeta azul, pero no tarjeta negra.
4. Led conectado en PC13 en tarjeta azul y en PB12 en tarjeta negra.
5. 32 pines de puertos en tarjeta azul (PC14 y PC15) y 30 pines de puertos en tarjeta negra.
6. Conector de puertos de 40 pines en tarjeta azul y de 34 en tarjeta negra.
7. Dos pines de V3.3 y tres de GND en tarjeta azul y solo uno de V3.3 y dos de GND en tarjeta negra.
8. La tarjeta negra es más angosta que la tarjeta azul.

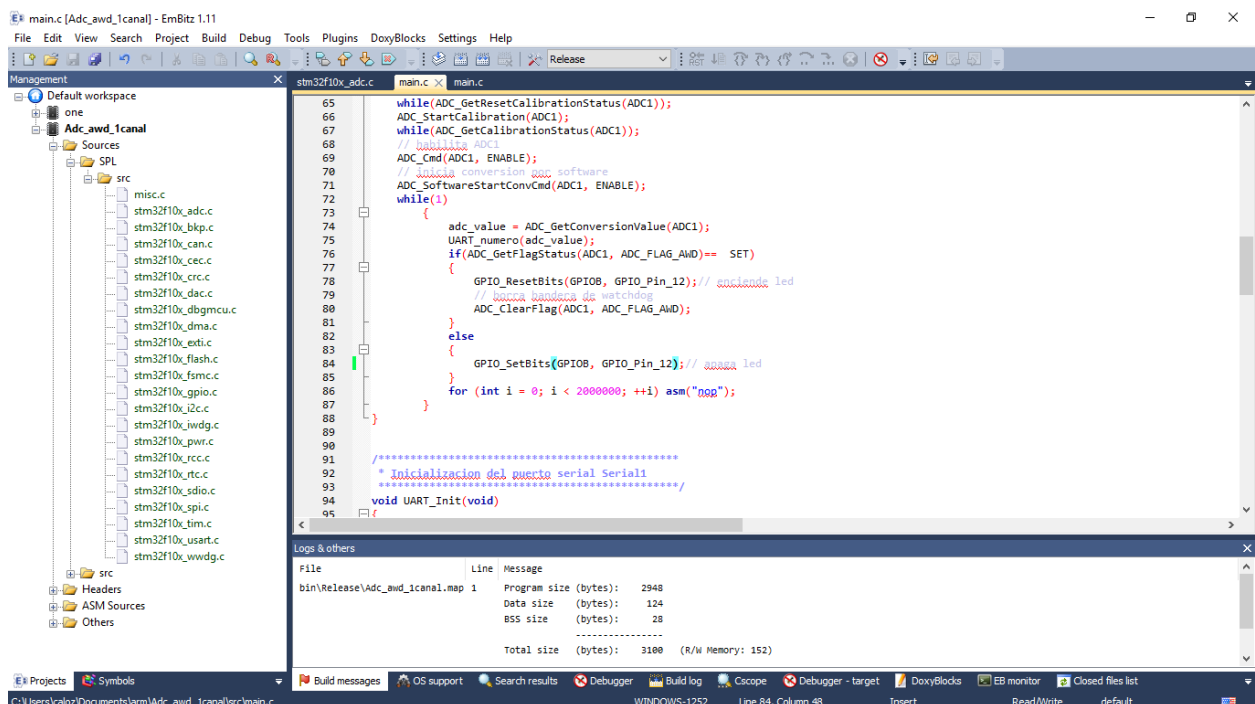
Programas de compilación del STM32F103C8T6

Para programar un microcontrolador, se debe llevar un archivo a la memoria de programa de este con un programa que generalmente es hecho por el mismo fabricante. Este archivo tiene una serie de números en hexadecimal que son los que representan las instrucciones del programa ya convertidas a lenguaje de máquina. Estos archivos generados por el compilador tienen extensión .bin o .hex. Hay varios programas de compilación disponibles para un microcontrolador y que se les denomina IDE, ambiente de desarrollo integrado, por sus siglas en inglés. Hay algunos muy buenos, pero hay que pagar una cierta cantidad de dinero para usarlos, o dan la posibilidad de trabajarlo por un tiempo como prueba y luego se debe comprar. Como la idea es trabajar con una pastilla económica, pues hay que seguir con la misma idea a la hora de buscar un programa IDE. Por esto a continuación se presenta un listado de programas gratuitos muy populares para programar el STM32F103C8T6 y sus principales características.

- **emIDE:** Tiene todas las herramientas posibles para desarrollo de aplicaciones para el microcontrolador. Tiene un diseño intuitivo al usuario no basado en el famoso programa Eclipse. Eclipse es pesado, en el sentido que es un archivo grande para instalar y

requiere mucha memoria RAM. También tiene un editor que ayuda a la escritura de código como complementos de funciones definidas, resaltado de variables, etc. Se puede bajar el programa de instalación en la página emide.org.

- IAR y KEIL: Son dos herramientas de desarrollo muy conocidos en el mundo de los microcontroladores como de los mejores. Aunque estas son versiones comerciales, se puede usar sin pagar cuando el programa no excede un tamaño de hasta 32K. Se puede conseguir IAR en la página <https://www.iar.com> y KEIL en la página <http://www.keil.com>.
- Atollic, o STM32CubeIDE: Es parte de STMicroelectronics, la casa fabricante de la pastilla STM32F103C8T6, y gratis, con todo el soporte del fabricante. Es un programa basado en Eclipse, así que es un poco pesado y lento. Se programa con una especie de librería para definición de registros CMSIS (Cortex Microcontroller Software Interface Standard) y con SPL (Standard Peripheral Library), además de la librería HAL (Hardware Abstraction Layer). Cuando se usa con STM32CubeMX, una herramienta para la configuración de periféricos, se integra con la librería HAL (Hardware Abstraction Layer) o con LL (Low Layer, parecida a SPL). Su página es <https://atollic.com>.
- Coocox: Es una herramienta de desarrollo completo con algunas características que hacen fácil la programación, como configuración de pines, RTOs, debugging. Se encuentra en la página <http://www.coocox.org>
- Embitz: Es el IDE que se describe en este documento, como una herramienta sencilla de usar, gratuita y completa. Soporta la librería SPL (Standard Peripheral Library) de STMicroelectronics, con funciones completas que minimizan la complejidad de configuración de periféricos. La página es <https://www.embitz.org>. Actualmente ya no está soportada, sin embargo la librería SPL todavía se usa en algunos compiladores.



```
65 while(ADC_GetResetCalibrationStatus(ADC1));
66 ADC_StartCalibration(ADC1);
67 while(ADC_GetCalibrationStatus(ADC1));
68 // Calibración ADC1
69 ADC_Cmd(ADC1, ENABLE);
70 // Inicialización de software
71 ADC_SoftwareStartConvCmd(ADC1, ENABLE);
72 while(1)
73 {
74     adc_value = ADC_GetConversionValue(ADC1);
75     UART_numero(adc_value);
76     if(ADC_GetFlagStatus(ADC1, ADC_FLAG_AWD) == SET)
77     {
78         GPIO_ResetBits(GPIOB, GPIO_Pin_12); // enciende led
79         // Inicia temporizador de watchdog
80         ADC_ClearFlag(ADC1, ADC_FLAG_AWD);
81     }
82     else
83     {
84         GPIO_SetBits(GPIOB, GPIO_Pin_12); // apaga led
85     }
86     for (int i = 0; i < 2000000; ++i) asm("nop");
87 }
88
89
90
91
92 /* Inicialización del puerto serial Serial1
93 *****
94 void UART_Init(void)
95 {
```

File	Line	Message
bin\Release\Adc_awd_1canal.map	1	Program size (bytes): 2948
		Data size (bytes): 124
		BSS size (bytes): 28

		Total size (bytes): 3100 (R/W Memory: 152)

Figura 8. Programa IDE de Embitz para programar el STM32F103C8T6.

Otras herramientas de mucho interés y popularidad para programar la pastilla STM32F103C8T6 es mbed, que se encuentra en la página <https://www.mbed.com>. Esta herramienta está en línea y no hay que descargar un archivo para instalación, si no que presenta todas las características de una herramienta de desarrollo en una página de internet. Otra de las ventajas es que se puede usar código público y transformarlo a la medida. Una vez compilado el programa, se crea un archivo con extensión hex o bin para llevarlo a la pastilla. Arduino también soporta la programación de la pastilla STM32F103C8T6 con mucha información sobre programación que se puede usar de otras personas que ya han realizado librerías para periféricos de gran uso.

Programación de la pastilla azul o negra con el programa Embitz.

Hay dos opciones para programar las tarjetas azul o negra con el procesador STM32F103C8T6. Una es usando el STLinkv2, que se encuentra disponible comercialmente en alrededor de \$2 dólares y se muestra en la figura 9.

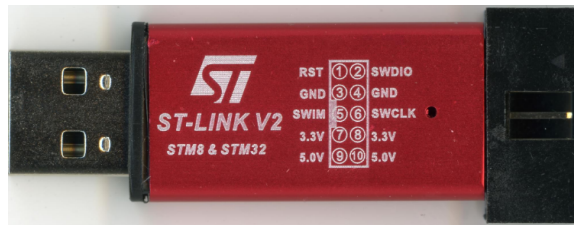


Figura 9. ST-Link V2 para programar la pastilla azul o negra.

De los pines del módulo STLinkV2 se deben usar SWDIO y conectarlo con DIO de la tarjeta, GND con GND, SWCLK con CLK y 3.3V con +3.3V de la tarjeta del procesador.

La otra opción para programar las tarjetas azul y negra es por medio del puerto serial Usart1 que está conectado en los pines A9 y A10 y usar un convertidor TTL a USB, como se muestra en la figura 10 para la tarjeta azul. Con la tarjeta negra es igual, solo que la conexión de +5V no se usa si no la de +3.3V.

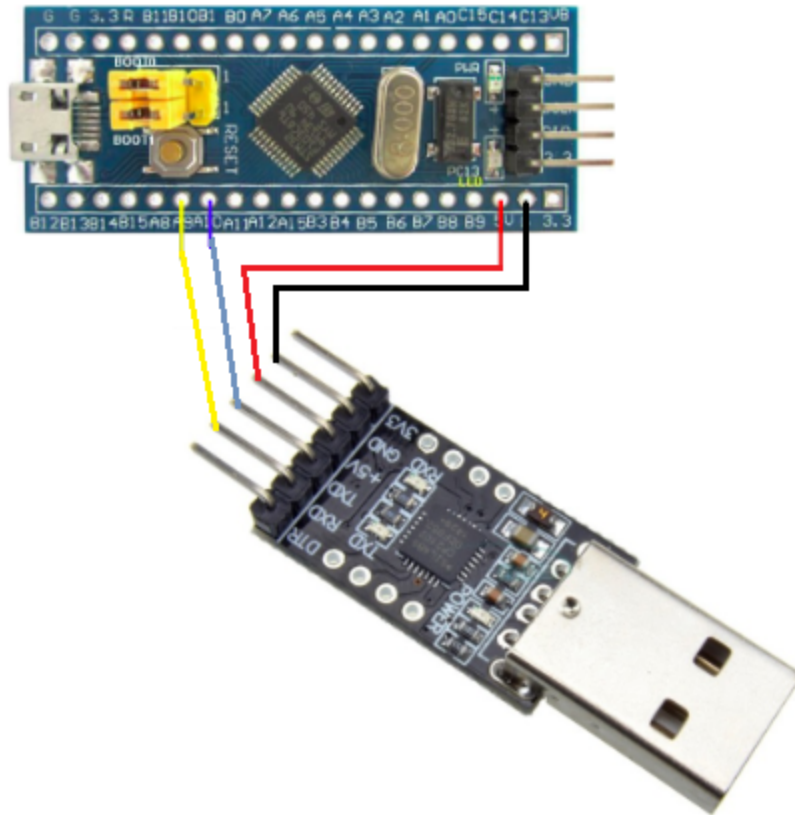


Figura 10. Conexión del módulo USB a TTL para programar la pastilla azul o negra.

Una vez instalado el programa lo que hay que hacer es escoger el integrado que se va a programar, pues este programa Embitz puede programar microcontroladores de varios fabricantes. Para crear un nuevo proyecto se escoge File > New > Proyecto.... Aparecerá la ventana que se muestra en la figura 11.

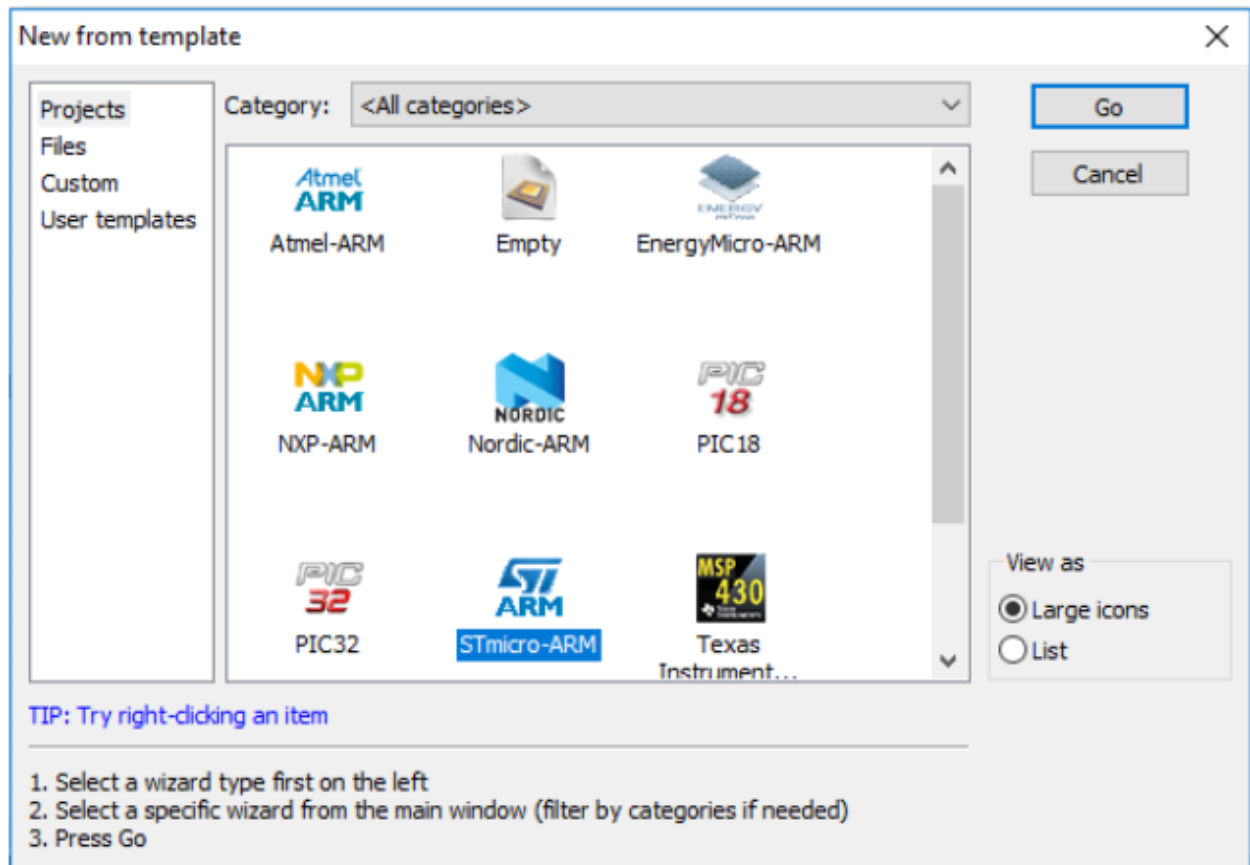


Figura 11. Al crear un nuevo proyecto, se abre esta ventana para elegir el microcontrolador.

De esta ventana se debe escoger ST ARM STmicro-ARM, que es el fabricante del microcontrolador STM32F103C8T6. Luego aparece la ventana que se muestra en la figura 12. Aquí se le da "Next >".

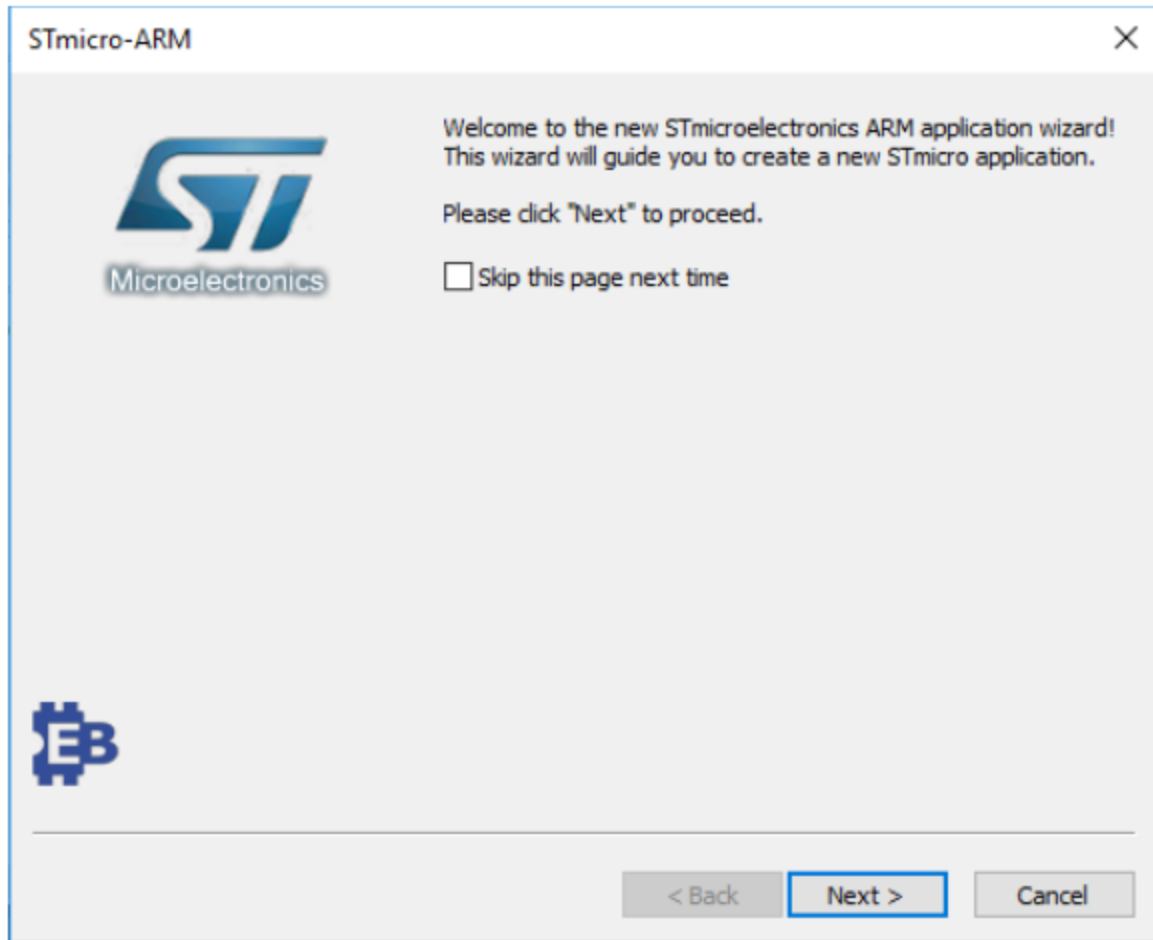



Figura 12. Ventana de bienvenida a proceso de creación del proyecto.

Luego aparecerá una ventana como la de la figura 13, donde pide darle nombre al proyecto que va a crear. Debe llenar este espacio con el nombre que le va a dar a su proyecto y luego presionar "Next >".

STmicro-ARM

Please select the folder where you want the new project to be created as well as its title.




Project title:
ensamble

Folder to create project in:
C:\Users\caloz\Documents\arm\

Project filename:
ensamble.ebp

Resulting filename:
C:\Users\caloz\Documents\arm\ensamble\ensamble.ebp





< Back Next > Cancel

Figura 13. Ventana para el nombre del proyecto.

Aparecerá la ventana que muestra la figura 14.

STmicro-ARM

Please select the compiler to use and which configurations you want enabled in your project.

Compiler:
ARM GCC Compiler (EmBitz - bare-metal) ▾

☒ Create "Debug" configuration: Debug

"Debug" options
Output dir.: bin\Debug\
Objects output dir.: obj\Debug\

☒ Create "Release" configuration: Release

"Release" options
Output dir.: bin\Release\
Objects output dir.: obj\Release\

< Back Next > Cancel

Figura 14. Selección del compilador.

Aquí no se debe modificar nada y se debe presionar "Next >". Aparecerá la ventana de la figura 15.

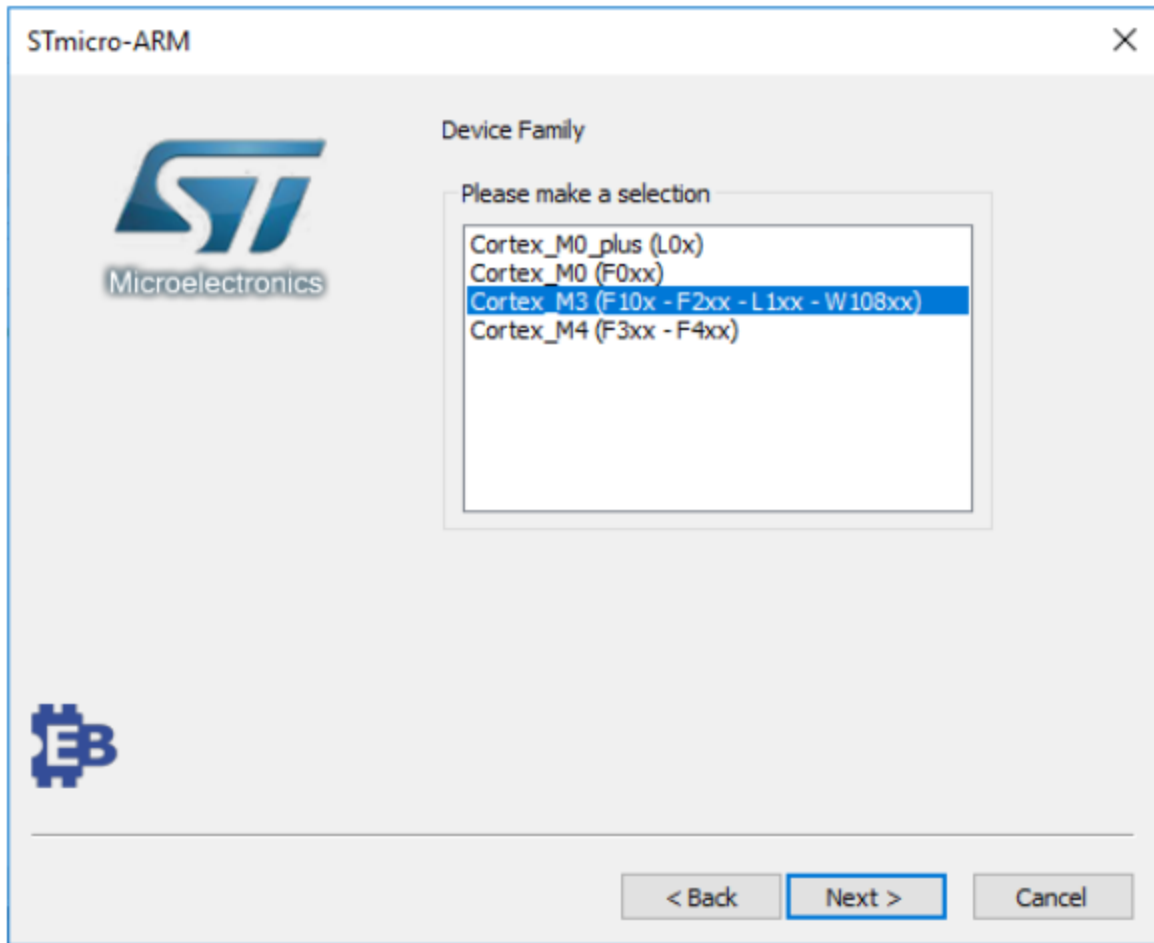


Figura 15. Selección del tipo de procesador ARM.

En esta ventana se escoge Cortex_M3 y se oprime "Next >". Aparecerá la siguiente ventana, como lo muestra la figura 16.

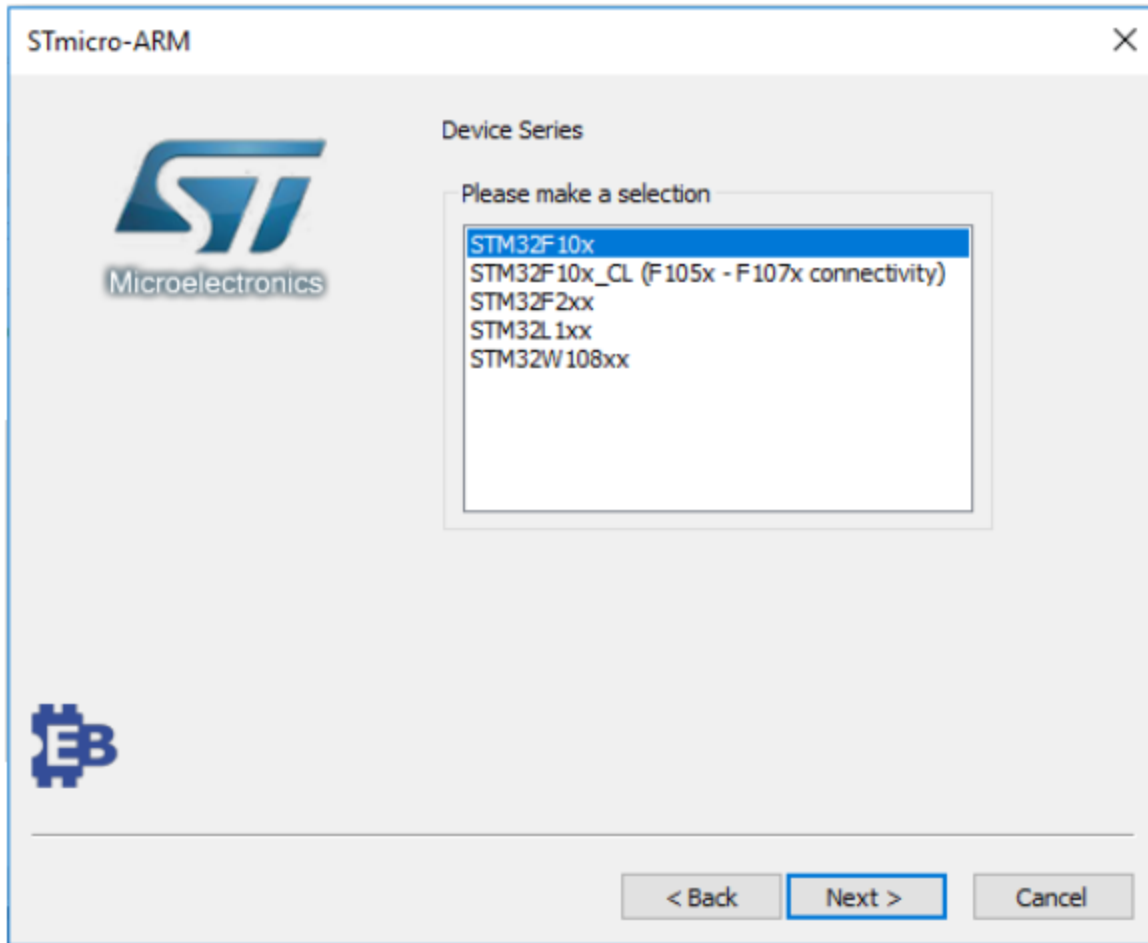


Figura 16. Selección de la familia de procesadores de STM.

Aquí se escoge la primera opción, "STM32F10x" y se oprime "Next >". Aparecerá la ventana que se muestra en la figura 17.

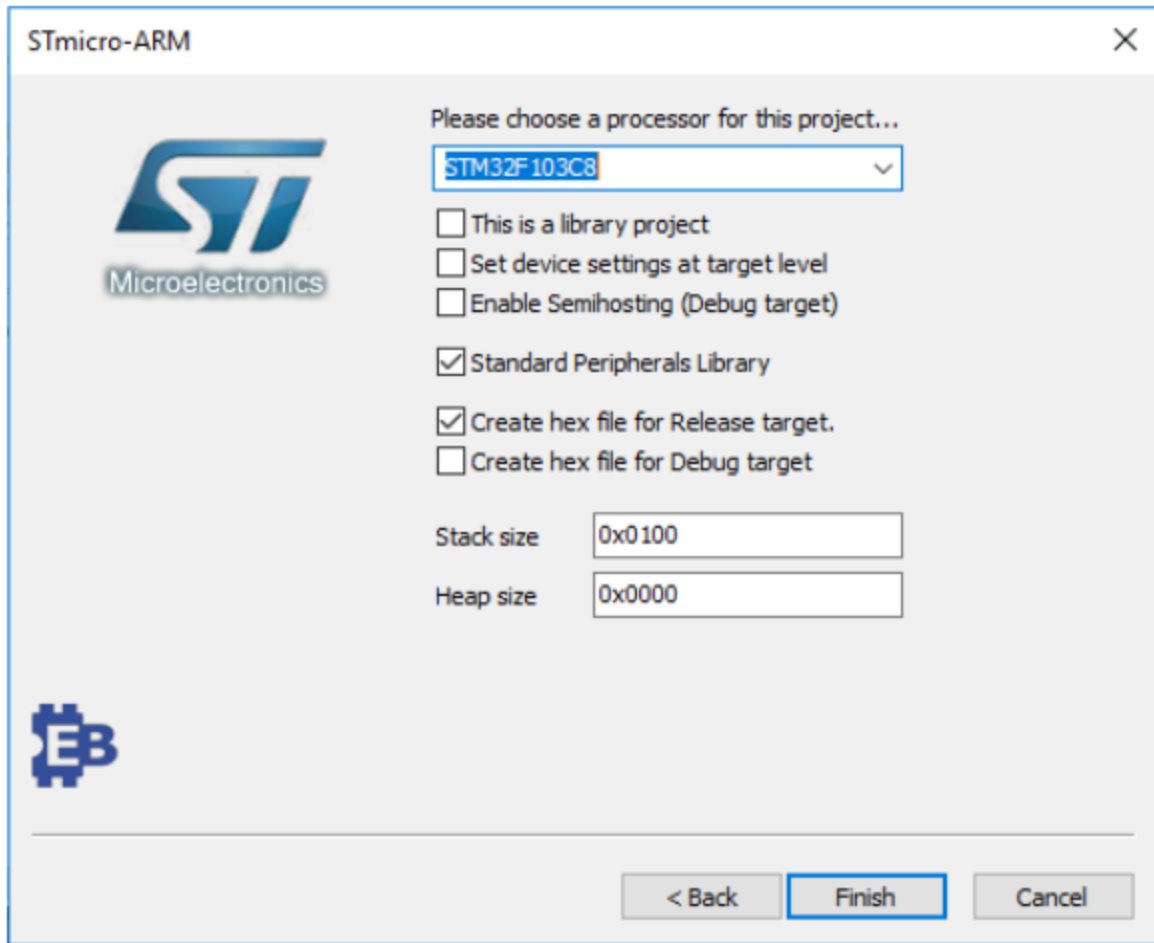


Figura 17. Selección de la referencia del microcontrolador para el proyecto.

En esta ventana se debe escoger el procesador “STM32F103C8” de la lista de opción que aparece. Los demás parámetros no se deben modificar. Se recomienda que la opción “Create hex file for Release target” esté marcada para que genere el archivo con extensión .hex que se debe usar con el programador. Luego presionar “Finish >”. Aparecerá la ventana de la figura 18.

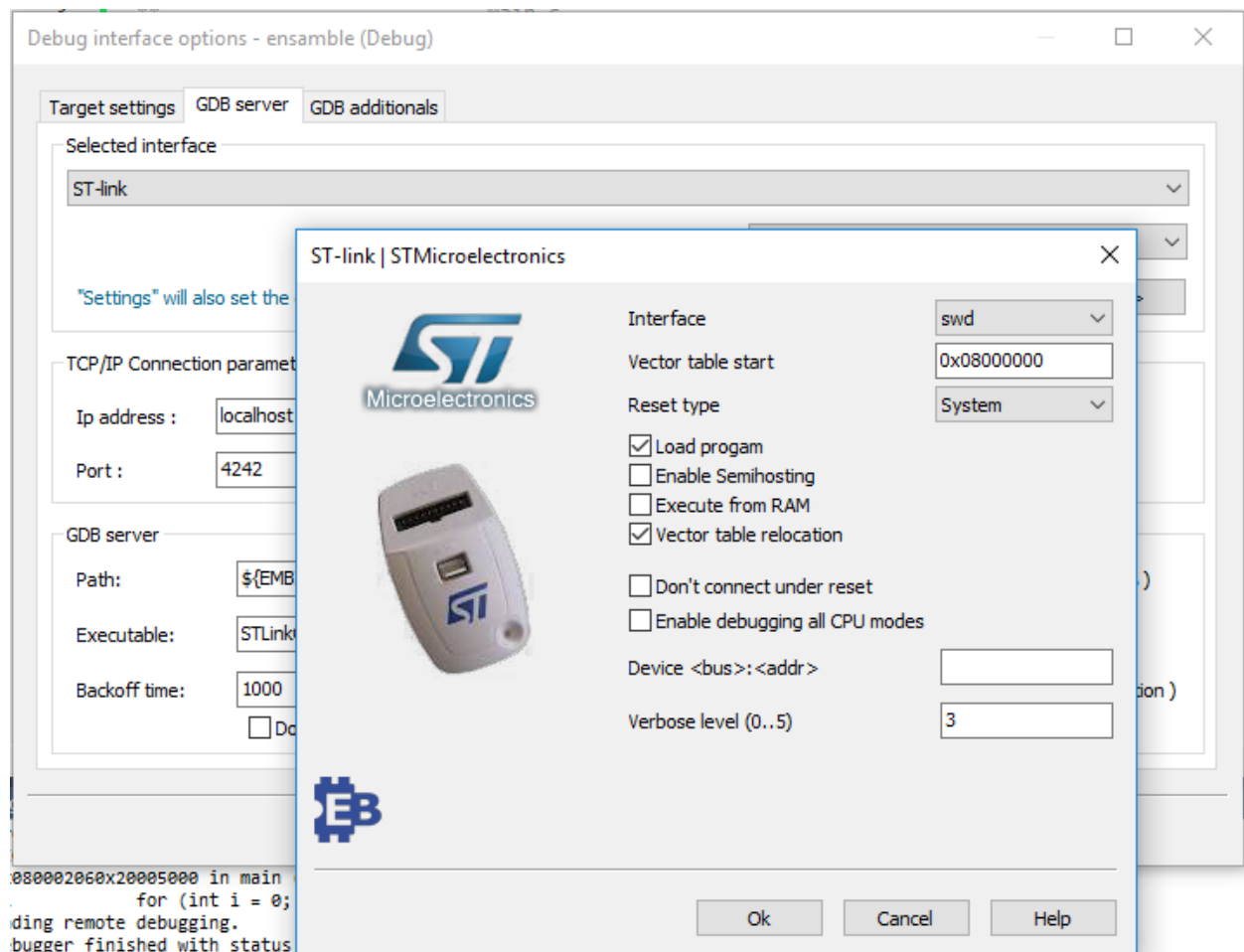


Figura 18. Ventana para seleccionar el ST- Link para programar la tarjeta.

Pulsar “Ok” sin modificar nada. Y luego de nuevo “Ok”. Ya está listo el ambiente de programación para empezar a escribir cualquier programa. Como modo de ejemplo se va a abrir el archivo main.c, tal como aparece en la figura 18.

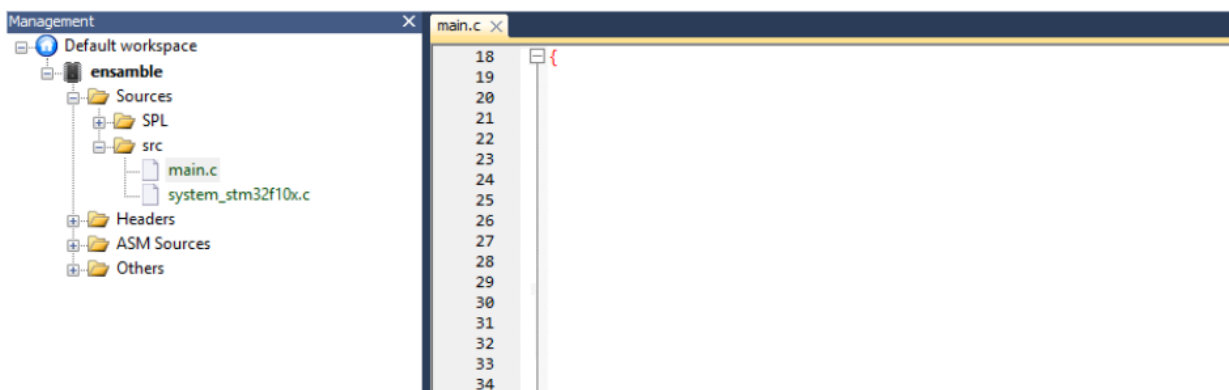


Figura 19. Ambiente Embitx de programación.

En la figura 20 se puede apreciar el diagrama de bloques del microcontrolador STM32F103C8T6.



Descripción del procesador

El procesador del microcontrolador STM32F103C8T6 es un Cortex- M3. Como procesador, tiene 16 registros (R0 a R15) de 32 bits internos que se muestran en la figura 21. Los primeros 12 registros son de propósito general, R13 es el apuntador de la memoria de pila, que es la que sirve para almacenar datos temporales, R14 que sirve para almacenar direcciones de retorno en llamados y saltos a subrutinas, y R15 que es el contador de programa, o sea, guarda la dirección donde está la instrucción que se ejecuta. Además posee registros especiales, como el PSR que tiene bits de estado que muestran el resultado de ejecución de una instrucción, registros de monitoreo en ejecución de instrucciones y un registro de control.

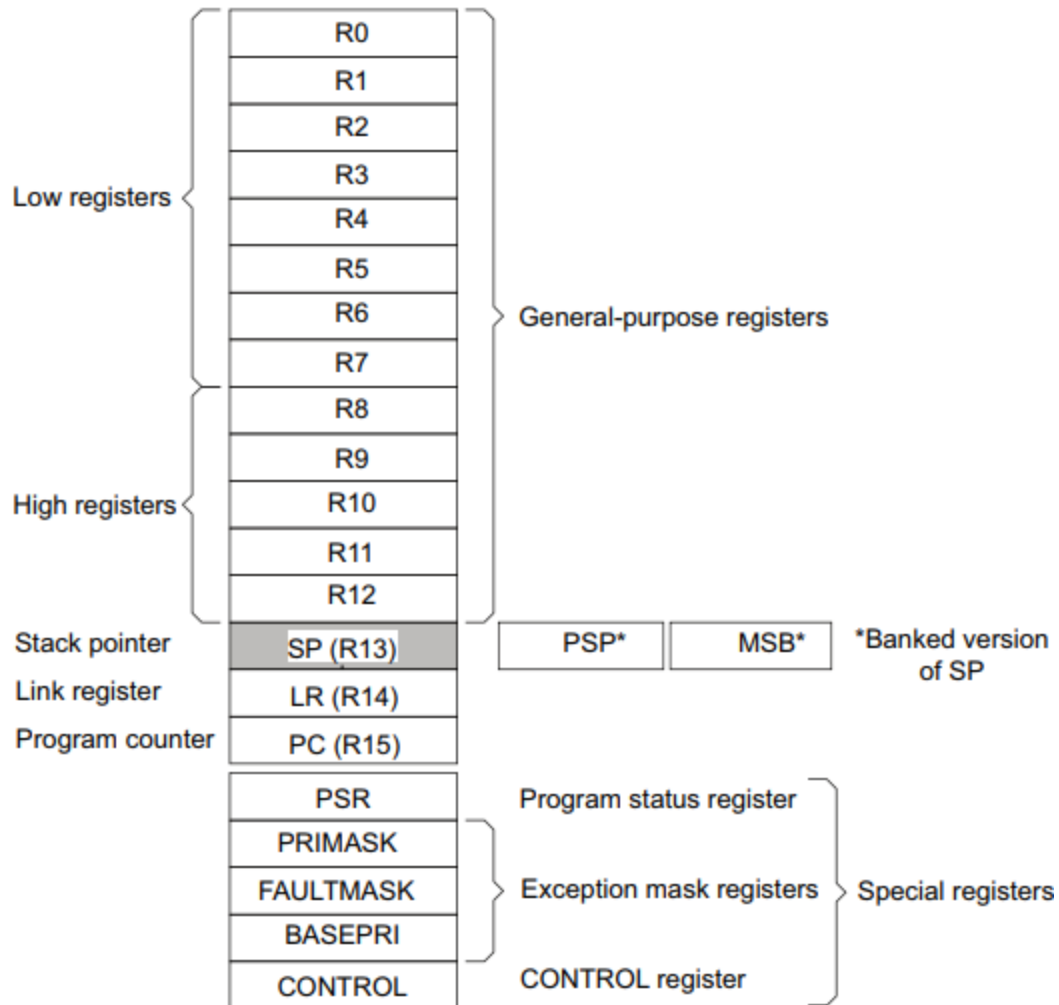


Figura 21. Registros del microprocesador.

El procesador tiene varias instrucciones como instrucciones de suma (ADC, ADCS, ADD, ADDS, ADDW), resta (SUB, SUBS, SUBW), multiplicación (MLA, MLS, MLU, MUL, MULS, UMLAL, UMULL, SMLAL), división (UDIV, SDIV), lógicas (AND, ANDS, EOR, EORS, ORN, ORNS, ORR, ORRS), de desplazamiento del contenido (ASR, LSL, LSR, ROR, RRX), de carga en registros y memoria (ADR, LDR, POP, PUSH, MOVT), condicionales (BL, BLX), entre otras. Estas instrucciones son la base de cualquier programación, pues si el programa se hace en lenguaje ensamblador, se tiene que recurrir al conjunto de instrucciones para hacer el

programa, pero si el lenguaje es, por ejemplo, C, se debe usar un compilador que hace que una sentencia en C se convierta en una serie de instrucciones en lenguaje ensamblador.

Lenguaje ensamblador.

El microprocesador tiene un repertorio de instrucciones en lenguaje ensamblador. A modo de ejercicio, aquí se va a programar el procesador para que haga el encendido y apagado del led de la tarjeta usando algunas instrucciones en lenguaje ensamblador. Como el compilador embitz no compila directamente las instrucciones en lenguaje ensamblador, es necesario usar un comando en C que lo haga. Este comando en C es `asm("instrucción")`, donde "instrucción" va en lenguaje ensamblador. Para entender un poco el funcionamiento de las instrucciones, se sugiere hacer el siguiente procedimiento.

- Crea un proyecto de nombre "ensamble" y abra el archivo main.c, como aparece en la figura 22.

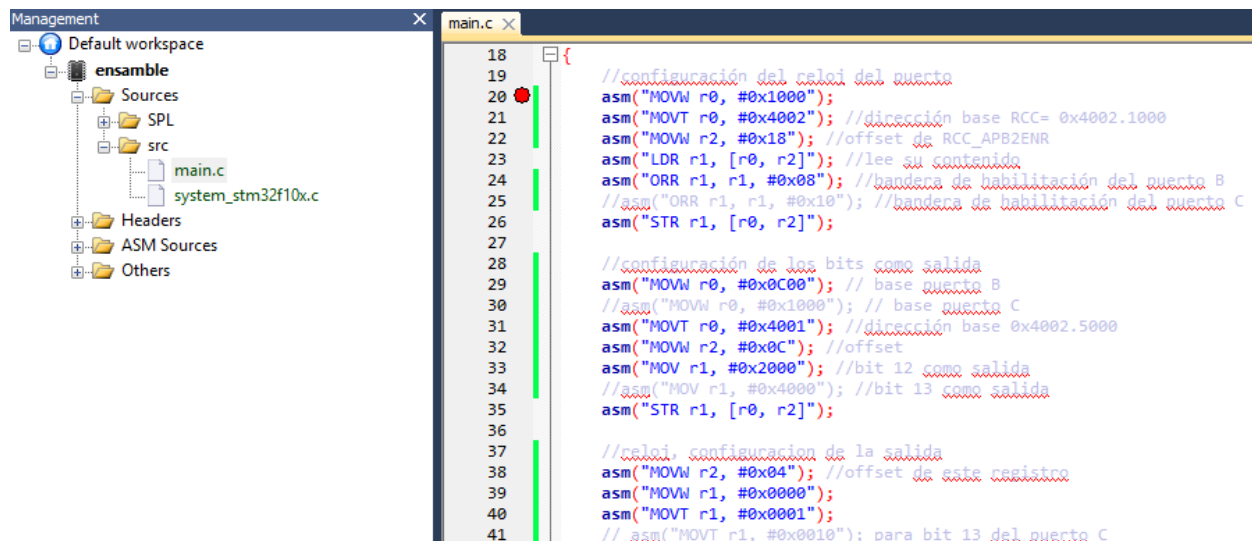


Figura 22. Creación del proyecto "ensamble" y vista del archivo main.c.

- En este archivo se va a escribir el siguiente código dentro de las llaves que aparecen en el "int main(void)":

```
//configuración del reloj del puerto
asm("MOVW r0, #0x1000");
asm("MOVT r0, #0x4002"); //dirección base RCC= 0x4002.1000
asm("MOVW r2, #0x18"); //offset de RCC_APB2ENR
asm("LDR r1, [r0, r2]"); //lee su contenido
asm("ORR r1, r1, #0x08"); //bandera de habilitación del puerto B
asm("STR r1, [r0, r2]");

//configuración de los bits como salida
asm("MOVW r0, #0x0C00"); // base puerto B
asm("MOVT r0, #0x4001"); //dirección base 0x4001.0C00
asm("MOVW r2, #0x04"); //offset del registro GPIOB_CRH
asm("MOVW r1, #0x0000");
```

```

asm("MOVT r1, #0x0001");
asm("STR r1, [r0, r2]");

// encender led
asm("MOVW r2, #0x10"); //offset del registro GPIOB_BSRR
asm("MOVT r1, #0x1000");
asm("STR r1, [r0, r2]");

```

- Y luego escriba el siguiente código dentro de las llaves del “while(1)”:

```

for (uint32 i = 0; i < 2000000; ++i) asm("nop");
// apagar led
asm("MOVW r0, #0x0C00"); // base puerto B
asm("MOVT r0, #0x4001");
asm("MOVW r2, #0x10");
asm("MOVT r1, #0x0000");
asm("MOVW r1, #0x1000");
asm("STR r1, [r0, r2]");
for (uint32 i = 0; i < 2000000; ++i) asm("nop");
// encender led
asm("MOVW r0, #0x0C00"); // base puerto B
asm("MOVT r0, #0x4001");
asm("MOVW r2, #0x10");
asm("MOVW r1, #0x0000");
asm("MOVT r1, #0x1000");
asm("STR r1, [r0, r2]");

```

Este programa es una demostración de programación en lenguaje ensamblador. Lo que hace el programa es encender y apagar el led conectado en PB12 que tiene la tarjeta negra. El procedimiento para encender el led en la tarjeta negra es, primero, habilitar el reloj del puerto, segundo, configurar el reloj y el bit de salida, y por último, encender el led. Para entender el código en lenguaje ensamblador es necesario recurrir al mapa de memoria del procesador. En la figura 23 se muestran las direcciones de ubicación de los módulos conectados al bus AHB. Dentro de este mapa de memoria se ubica RCC (Reset and clock control), que es el que define el reloj de los periféricos. La ubicación de este registro está a partir de la dirección 0x4002.1000. Se menciona que a partir de esta posición porque son varios registros y no uno solo, lo que se conoce como offset.

Campos de memoria	Periféricos	Bus
0xA000 0000 - 0xA000 0FFF	FSMC	AHB
0x5000 0000 - 0x5003 FFFF	USB OTG FS	
0x4003 0000 - 0x4FFF FFFF	Reserved	
0x4002 8000 - 0x4002 9FFF	Ethernet	
0x4002 3400 - 0x4002 7FFF	Reserved	
0x4002 3000 - 0x4002 33FF	CRC	
0x4002 2000 - 0x4002 23FF	Flash memory interface	
0x4002 1400 - 0x4002 1FFF	Reserved	
0x4002 1000 - 0x4002 13FF	Reset and clock control RCC	
0x4002 0000 - 0x4002 0FFF	Reserved	
0x4002 0400 - 0x4002 07FF	DMA2	
0x4002 0000 - 0x4002 03FF	DMA1	
0x4001 8400 - 0x4001 FFFF	Reserved	
0x4001 8000 - 0x4001 83FF	SDIO	

Figura 23. Direcciones de memoria de los módulos del microcontrolador.

Los registros contenidos dentro del campo RCC son los que se muestran en la figura 24, y las direcciones que se suman a la dirección RCC se conocen como offset.

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	RCC_CR	Reserved						PLL RDY	PLL ON	Reserved					CSSON	HSEBYP	HSERDY	HSEON	HSICAL[7:0]					HSITRIM[4:0]					Reserved	HSIRDY	HSION			
	Reset value							0	0						0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
0x04	RCC_CFGR	Reserved						MCO [2:0]	Reserved	USBPRE	PLLMUL [3:0]					PLLXTPRE	PLLSRC	ADC PRE [1:0]	PPRE2 [2:0]	PPRE1 [2:0]	HPRE[3:0]			SWS [1:0]	SW [1:0]									
	Reset value							0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	RCC_CIR	Reserved								CSSC	Reserved	PLLRDYC	HSERDYC	HSIRDYC	LSERDYC	LSIRDYC	Reserved			PLLRDYIE	HSERDYIE	HSIRDYIE	LSERDYIE	LSIRDYIE	CSSF	Reserved			PLLRDYF	HSERDYF	HSIRDYF	LSERDYF		
	Reset value									0		0	0	0	0	0	0				0	0	0	0	0	0				0	0	0	0	
0x0C	RCC_APB2RSTR	Reserved										TIM1RST	TIM10RST	TIM9RST	Reserved			ADC3RST	USART1RST	TIM8RST	SPI1RST	TIM1RST	ADC2RST	ADC1RST	IOPGRST	IOPFRST	IOPHRST	IOPORST	IOPBRST	IOPARST	Reserved		AFIORST	
	Reset value											0	0	0				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x010	RCC_APB1RSTR	Reserved	DACRST	PWRRST	BKPRST	Reserved	CANRST	Reserved	USBRST	I2C2RST	I2C1RST	UART5RST	UART4RST	USART3RST	USART2RST	Reserved	SPI3RST	SPI2RST	Reserved			WWDGRST	Reserved		TIM14RST	TIM13RST	TIM12RST	TIM7RST	TIM6RST	TIM5RST	TIM4RST	TIM3RST	TIM2RST	
	Reset value		0	0	0		0		0	0	0	0	0	0	0		0	0				0			0	0	0	0	0	0	0	0	0	
0x14	RCC_AHBENR	Reserved																SDIOEN		Reserved	FSMCEN		Reserved	CRCEEN	Reserved	FLITFEN	Reserved	SRAIEN	DMA2EN	DMA1EN				
	Reset value																	0			0			0		0		0		0	0	0	0	
0x18	RCC_APB2ENR	Reserved										TIM11EN	TIM10EN	TIM9EN	Reserved			ADC3EN	USART1EN	TIM8EN	SPI1EN	TIM1EN	ADC2EN	ADC1EN	IOPGEN	IOPFEN	IOPEEN	IOPDEN	IOPCEN	IOPBEN	IOPAEN	Reserved		AFOEN
	Reset value											0	0	0				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	RCC_APB1ENR	Reserved	DACEN	PWREN	BKPEN	Reserved	CANEN	Reserved	USBEN	I2C2EN	I2C1EN	UART5EN	UART4EN	USART3EN	USART2EN	Reserved	SPI3EN	SPI2EN	Reserved			WWDGEN	Reserved		TIM14EN	TIM13EN	TIM12EN	TIM7EN	TIM6EN	TIM5EN	TIM4EN	TIM3EN	TIM2EN	
	Reset value		0	0	0		0		0	0	0	0	0	0	0		0	0				0			0	0	0	0	0	0	0	0	0	
0x20	RCC_BDCR	Reserved														BDRST	RTCEEN	Reserved					RTC SEL [1:0]		Reserved					LSEBYP	LSERDY	LSEON		
	Reset value															0	0													0	0	0		
0x24	RCC_CSR	LPWRSTF	WWDGRSTF	IWDGRSTF	SFTRSTF	PORRSTF	PINRSTF	Reserved	RMVF	Reserved																		LSIRDY	LSION					
	Reset value	0	0	0	0	1	1		0																			0	0	0				

Figura 24. Registros y descripción de bits de los registros de control RCC.

- El registro RCC_CR, con offset 0x00, es un registro para el control y la habilitación de los relojes.
- El registro RCC_CFGR, con offset 0x04, es un registro para la configuración de los relojes.

- El registro RCC_CIR, con offset 0x08, es un registro de control de interrupciones de los módulos de reloj, como PLL, HSI, HSE, LSI, LSE.
- El registro RCC_APB2RSTR, con offset 0x0C, registro para inicio del bus APB2. Reinicia los periféricos TIM11, TIM10, TIM9, TIM8, TIM1, ADC3, ADC2, ADC1, USART1, SPI1, funciones alternas y los puertos de entrada/ salida A a G.
- El registro RCC_APB1RSTR, con offset 0x10, registro para reinicio del bus APB1. Reinicia los periféricos DAC, CAN, USB, I2C2, I2C1, USART5, USART4, USART3, USART2, SPI3, SPI2, watchdog por ventana, TIM14, TIM13, TIM12, TIM7, TIM6, TIM5, TIM4, TIM3, y TIM2.
- El registro RCC_AHBENR, con offset 0x14, controlador del reloj del bus AHB. Controla la habilitación de los periféricos SDIO, FSMC, CRC, FLITF, SRAM, DMA2 y DMA1.
- ***El registro RCC_APB2ENR, con offset 0x18, Controlador del reloj del bus APB2. Controla la habilitación de los periféricos TIM11, TIM10, TIM9, TIM8, TIM1, ADC3, ADC2, ADC1, USART1, SPI1, funciones alternas y los puertos de entrada/ salida A a G.***
- El registro RCC_APB1ENR, con offset 0x1C, Controlador del reloj del bus APB1. Controla la habilitación de los periféricos DAC, CAN, USB, I2C2, I2C1, USART5, USART4, USART3, USART2, SPI3, SPI2, watchdog por ventana, TIM14, TIM13, TIM12, TIM7, TIM6, TIM5, TIM4, TIM3, y TIM2.
- El registro RCC_BDCR, con offset 0x20, Registro de control del dominio de guardado (Backup Domain Control Register). Controla el RTC y el oscilador externo de 32.768 hz.
- El registro RCC_CSR, con offset 0x24, Registro de control y estado. Controla el reset, los watchdogs y el oscilador de 40 Khz.

Para habilitar el puerto B, hay que ubicar el registro APBENR dentro de los registros contenidos en RCC. En la figura 24 se encuentra este registro con el offset 0x18.

Este offset , 0x18, registro APB2ENR, es lo que se le suma a la dirección de inicio de la dirección donde se ubica el registro RCC, 0x4000.1000. En la misma figura 24 se observa que el bit 3 es de habilitación del puerto B. Entonces, lo que se debe hacer para habilitar el puerto B es poner en "1" lógico el bit 3 del registro APB2ENR que está en la dirección del registro RCC + 0x18. En lenguaje ensamblador se usan los registros de propósito general R0 a R12. Cada registro es de 32 bits, pero las instrucciones en lenguaje ensamblador cargan de a 16 bits al registro. La instrucción MOVW carga los 16 bits menos significativos y la instrucción MOVT carga los 16 bits más significativos. Para habilitar el puerto B, en lenguaje ensamblador se escribe la secuencia de instrucciones,

```
asm("MOVW r0, #0x1000");
asm("MOVT r0, #0x4002"); //dirección base RCC= 0x4002.1000
asm("MOVW r2, #0x18"); //offset de RCC_APB2ENR
asm("LDR r1, [r0, r2]"); //lee su contenido
asm("ORR r1, r1, #0x08"); //bandera de habilitación del puerto B
asm("STR r1, [r0, r2]"); // se guarda de nuevo
```

El segundo paso es configurar el bit 12 del puerto B como salida. Para esto es necesario ver el mapa de memoria de los periféricos conectados al bus APB2, que es donde están los puertos de entrada salida y que se muestran en la figura 25.

Campo de memoria	Periférico	Bus
0x4001 5800 - 0x4001 7FFF	Reserved	APB2
0x4001 5400 - 0x4001 57FF	TIM11 timer	
0x4001 5000 - 0x4001 53FF	TIM10 timer	
0x4001 4C00 - 0x4001 4FFF	TIM9 timer	
0x4001 4000 - 0x4001 4BFF	Reserved	
0x4001 3C00 - 0x4001 3FFF	ADC3	
0x4001 3800 - 0x4001 3BFF	USART1	
0x4001 3400 - 0x4001 37FF	TIM8 timer	
0x4001 3000 - 0x4001 33FF	SPI1	
0x4001 2C00 - 0x4001 2FFF	TIM1 timer	
0x4001 2800 - 0x4001 2BFF	ADC2	
0x4001 2400 - 0x4001 27FF	ADC1	
0x4001 2000 - 0x4001 23FF	GPIO Port G	
0x4001 1C00 - 0x4001 1FFF	GPIO Port F	
0x4001 1800 - 0x4001 1BFF	GPIO Port E	
0x4001 1400 - 0x4001 17FF	GPIO Port D	
0x4001 1000 - 0x4001 13FF	GPIO Port C	
0x4001 0C00 - 0x4001 0FFF	GPIO Port B	
0x4001 0800 - 0x4001 0BFF	GPIO Port A	
0x4001 0400 - 0x4001 07FF	EXTI	
0x4001 0000 - 0x4001 03FF	AFIO	

Figura 25. Mapa de memoria de los periféricos conectados al bus APB2.

Cada puerto GPIO tiene varios registros para su configuración. Los siguientes son los registros.

- Port configuration register low (GPIOx_CRL) (x=A..G), dirección de offset: 0x00.
- Port configuration register high (GPIOx_CRH) (x=A..G), dirección de offset: 0x04.
- Port input data register (GPIOx_IDR) (x=A..G), dirección de offset: 0x08.
- Port output data register (GPIOx_ODR) (x=A..G), dirección de offset: 0x0C.
- Port bit set/reset register (GPIOx_BSRR) (x=A..G), dirección de offset: 0x10.
- Port bit reset register (GPIOx_BRR) (x=A..G), dirección de offset: 0x14.
- Port configuration lock register (GPIOx_LCKR) (x=A..G), dirección de offset: 0x18.

El puerto B (GPIO Port B) está ubicado a partir de la dirección 0x4001.0C00. La configuración del puerto B con el bit 12 de la tarjeta negra donde está ubicado el led, se hace con el registro GPIOB_CRH, porque contiene la configuración de los bits 8 a 16. Este registro está descrito a continuación en la figura 26.

Port configuration register high (GPIOx_CRH) (x=A..G)

Address offset: 0x04

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, **CNFy[1:0]**: Port x configuration bits (y= 8 .. 15)
 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 These bits are written by software to configure the corresponding I/O port.
 Refer to [Table 20: Port bit configuration table](#).

In input mode (MODE[1:0]=00):

- 00: Analog mode
- 01: Floating input (reset state)
- 10: Input with pull-up / pull-down
- 11: Reserved

In output mode (MODE[1:0] > 00):

- 00: General purpose output push-pull
- 01: General purpose output Open-drain
- 10: Alternate function output Push-pull
- 11: Alternate function output Open-drain

Bits 29:28, 25:24, **MODEy[1:0]**: Port x mode bits (y= 8 .. 15)
 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 These bits are written by software to configure the corresponding I/O port.
 Refer to [Table 20: Port bit configuration table](#).

- 00: Input mode (reset state)
- 01: Output mode, max speed 10 MHz.
- 10: Output mode, max speed 2 MHz.
- 11: Output mode, max speed 50 MHz.

Figura 26. Descripción del registro del puerto B para configuración de sus pines 8 a 15.

De este registro se tienen en cuenta CNF12 y MODE12, que corresponden al bit 12. De la descripción del registro GPIOB_CRH se observa que el offset de este registro es 0x04, que los bits 17 y 16 definen si el bit 12 es de entrada o de salida y también la velocidad del puerto, que para este ejemplo se escoge de 10 Mhz, o sea, estos bits deben estar en 01, respectivamente. También se observa que los bits 19 y 18 del registro son para programar el bit 12 del puerto B como salida en modo push- pull, o sea, bits en 00, respectivamente. Los bits 19, 18, 17 y 16 quedan respectivamente en binario como b0001= 0x1. Entonces, la segunda parte del programa almacena el dato de 32 bits 0x0001.0000 en la dirección (0x4001.0C00 + 0x04). Esto se puede observar en el siguiente código en lenguaje ensamblador.

```
asm("MOVW r0, #0x0C00"); // base puerto B
asm("MOVT r0, #0x4001"); // dirección base 0x4001.0C00
asm("MOVW r2, #0x04"); // offset del registro GPIOB_CRH
asm("MOVW r1, #0x0000");
asm("MOVT r1, #0x0001"); // bit 12 de salida en push- pull con 10Mhz.
```

```
asm("STR r1, [r0, r2]"); // almacena la información en el registro GPIOB_CRH
```

Lo que queda es encender el led, o sea, escribir un cero lógico en el bit 12 del puerto B. Se enciende poniendo un cero lógico porque el led está conectado a +3.3 voltios. Aquí se usa el registro GPIOB_BSRR que se describe a continuación en la figura 27.

Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy**: Port x Reset bit y (y= 0 .. 15)

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy**: Port x Set bit y (y= 0 .. 15)

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Set the corresponding ODRx bit

Figura 27. Descripción del registro para poner cero o uno en un pin del puerto B.

Este registro puede poner un cero o un uno lógico en un bit del puerto, poniendo un uno lógico en BR, el bit del puerto se apaga, y un uno lógico en BS, el bit del puerto se enciende. El registro GPIOB_BSRR tiene un offset de 0x10, y según la descripción, el bit 28 de este registro se debe poner en **uno** lógico para encender el led 12 del puerto B y en uno lógico el bit 12 para apagarlo. Esto se hace con el programa en lenguaje ensamblador de la siguiente manera.

```
asm("MOVW r0, #0x0C00"); // dirección base puerto B
asm("MOVT r0, #0x4001");
asm("MOVW r2, #0x10"); // offset del registro GPIOB_BSRR
asm("MOVW r1, #0x0000");
asm("MOVT r1, #0x1000"); // bit 28 en uno lógico para encender led
asm("STR r1, [r0, r2]");
```

Para alternar el encendido y apagado del led se hace dentro del while(1). Es necesario insertar un retardo y la forma más fácil es usando un for() con una variable de decremento. Lo que puede ocurrir con una instrucción en C mezclandolas con instrucciones en lenguaje ensamblador es que las instrucciones en C pueden usar cualquier registro de propósito general R0 hasta R12, por tanto hay que volver a cargar los valores de estos registros.

Taller 1.

1. Para la tarjeta azul haga las modificaciones necesarias para que se encienda el led conectado al pin C13 y escriba aquí las líneas que modifica.
2. Calcular el valor que debe tener el registro GPIOA_CRH para configurar el bit 8 del puerto A como entrada flotante.
3. Calcular el valor que debe tener el registro GPIOD_CRH para configurar el bit 11 del puerto D como salida de drenaje abierto con una frecuencia de 2 Mhz.
4. Calcular el valor que debe tener el registro GPIOB_BSRR para poner los bits 1, 3, 5, 7, 9, 11, 13 y 15 del puerto B en uno lógico.
5. Calcular el valor que debe tener el registro GPIOC_BSRR para poner los bits 0, 1, 2, 3 y 4 del puerto C en cero lógico.