UNIVERSIDAD NACIONAL DE COLOMBIA

Primer proyecto
Led ON (GPIO)

Gustavo Adolfo Osorio - Samuel Esteban Reyes - Santiago Matta Amador

Universidad Nacional de Colombia sede Manizales Email: gaosoriol@unal.edu.co sreyesn@unal.edu.co smatta@unal.edu.co

Introducción

El primer código que se estudiará en el curso encenderá un led usando los pines de salida del microcontrolador. Para esto tendremos que familiarizarnos con los registros que controlan los GPIO (General purpose input/output) e identificar la distribución de los pines en la tarjeta, la cual cuenta con varios puertos, pines y algunos elementos propios. Los documentos que se usarán en esta guía se encuentran en las referencias, junto con un link para abrirlos.

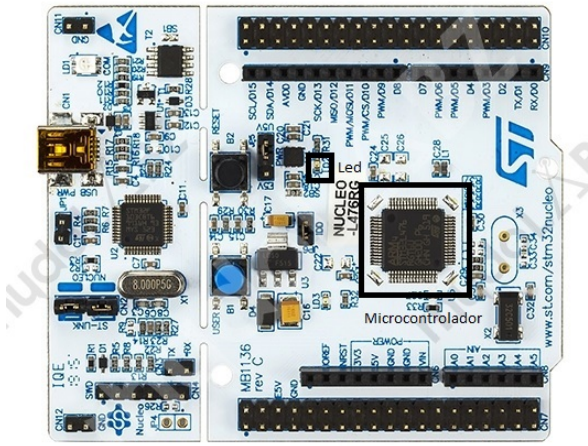
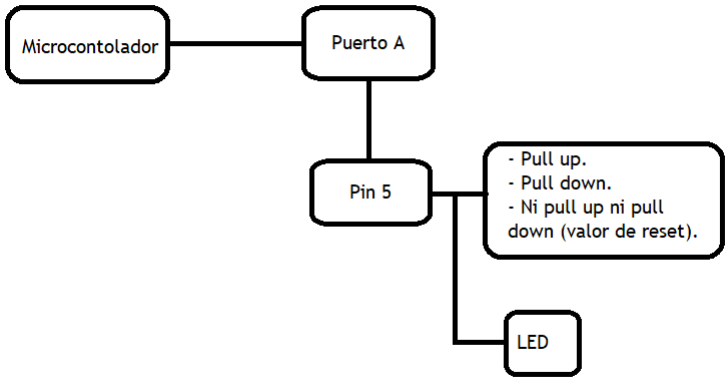


Figure 1. STM32L476


Microcontolador
Puerto A
Pin 5
- Pull up.
- Pull down.
- Ni pull up ni pull down (valor de reset).
LED

Contenido

1 Registros

Los registros son elementos de memoria con direcciones que almacenan la configuración de diferentes funciones del microcontrolador. Determinan los parámetros de: timers, interrupciones internas y externas, así como periféricos. Los registros se componen de 32 bits (algunos de los cuales son fijos y no se pueden modificar) que corresponden a una cierta característica o función según su posición, estos bits se cambian a través de operaciones de enmascaramiento para que así tomen los valores requeridos. Los registros tienen un valor de reset y un offset que servirá para inicializarlos.

+ RCC (Reset and clock control) address: 0x4002 1000
+ GPIOA (General purpose input/output)address: 0x4800 0000

La inicialización se hace sumando a la dirección del primer registro (RCC, GPIOA) el valor de offset del segundo (ODR,MODER, AHB2ENR).

1.1 RCC_AHB2ENR

AHB2 peripherial clock enable register, registro que nos permite activar el reloj para ciertas secciones de la tarjeta, en este caso se usará para activar el reloj del puerto del pin que estará conectado al led.

+ Adress offset: 0x4C
+ Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RNG EN	HASHE N	AESEN (1)
													rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DCMIE N	ADCEN	OTGFS EN	Res.	Res.	Res.	GPIOIE N	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
	rw	rw	rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 2. Distribución bits AHB2ENR, RM0351 Reference Manual, Pag. 251

Para el ejercicio el usuario usará el led 1 que se encuentra en la tarjeta, que está conectado al pin 5 del puerto A, lo cual podemos ver si accedemos al documento MB1136 schematic board que se encuentra en el link:
https://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf
Donde encontrarán la distribución y conexiones de la tarjeta en la página 63 (Electrical schematics). Teniendo en cuenta que se usará el puerto A, cargaremos al registro el valor correspondiente para activar el reloj del puerto.

1.2 GPIOx_MODER

GPIO port mode register, con este registro se configura el modo de funcionamiento de los pines de un puerto específico. Con este el usuario establecerá el pin 5 del puerto A como una salida. La x en el nombre del registro se cambia por la letra del puerto que se va a usar (A) y se compone de 32 bits que se agrupan en pares, cada par corresponde a un pin del puerto, por ejemplo los bits con el nombre MODE3 corresponden al pin 3.

+ Adress offset: 0x00
+ Reset value puerto A: 0xABFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[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
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 3. Distribución bits MODER, RM0351 Reference Manual, Pag. 303

Modos:
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode

Para el ejercicio se necesita configurar el pin 5 como una salida y así poder encender el led, por lo cual es necesario cargar en MODE5 los bits 01 sin alterar lo valores que se encuentran en los otros bits, para lo cual utilizaremos enmascaramiento.

Figure 26. Output configuration

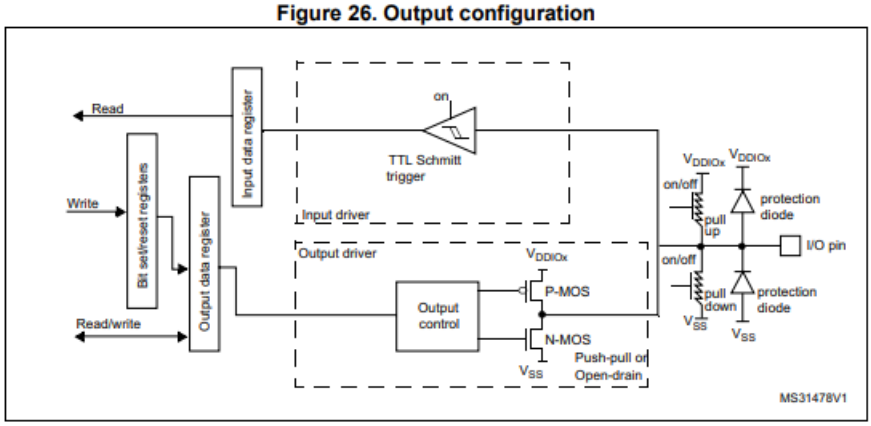


Figure 4. Output configuration, RM0351 Reference Manual, Pag. 301

1.3 GPIOx_ODR

GPIO port output data register, con este registro se manejan las salidas de un puerto (determinado por la letra que se ponga en el lugar de la x), escogiendo cuales están en HIGH y cuales en LOW (en otras palabras en que salidas hay voltaje y en cuales no). Solo se trabaja con los 16 primeros bits que están enumerados de 0 a 15, que corresponde al pin con el mismo número.

+ Adress offset: 0x14
+ Reset value puerto A: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OD15	OD14	OD13	OD12	OD11	OD10	OD9	OD8	OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 5. Distribución bits ODR, RM0351 Reference Manual, Pag. 305

Para poner en HIGH un pin específico basta con escribir un uno en el bit cuya posición corresponda con el número del pin.

2 Enmascaramiento

El uso de enmascaramiento es escencial en la programación en ensamblador y en C, es una operación usada para modificar los valores que se encuentran en los registros de una manera más precisa, alterando los bits que necesitamos sin cambiar los demás. Esto es útil en la medida en que es posible que no se conozca el valor que tenía el registro antes de modificarlo. El enmascaramiento se hace principalmente con las operaciones lógicas and y or.

2.1 AND

El enmascaramiento con AND permite convertir cualquier valor que se encuentra en un bit (0,1) en un 0, por lo tanto se usa para convertir los bits en cero cuando sea necesario y para los demas bits se aplica el AND con unos, para que mantengan su valor original.

AND	0	1
0	0	0
1	0	1

Table 1. Combinaciones AND.

En la tabla 1 se ilustra los resultados a las posibles operaciones AND entre bits, con lo cuál queda más claro, al hacer un AND con un cero cualquier valor se vuelve cero y al hacer AND con un uno, el valor del bit permanece igual.

EJM: Es necesario cambiar los bits 3 y 4 de un registro de 16 bits por 0 pero no conocemos que valores tiene ya (0xXXXX), entonces realizamos un enmascaramiento con and:
Como se van a cambiar los bits 3 y 4 y no se van a alterar los demas, escogemos el valor binario con ceros en los bits 3 y 4 y 1 en el resto: 0b1111111111100111 el cual corresponde en hexadecimal a 0xFFE7. al realizar el and obtenemos que:
0bXXXXXXXXXXXXXXXX AND 0b1111111111100111 = 0bXXXXXXXXXXX00XXX

2.2 OR

El enmascaramiento con OR se usa para convertir cualquier valor que se encuentre en un bit en un 1, por lo tanto se usa para forzar el valor de los bits a uno mientras que los bits que no se necesitan modificar se les aplica el OR con cero para que permanezcan igual.

OR	0	1
0	0	1
1	1	1

Table 2. Combinaciones OR.

En la tabla 2 se ilustra cuales son los resultados a los posibles operaciones OR entre bits, con lo cual queda más claro, al hacer un OR con un uno cualquier valor se vuelve uno y al hacer OR con un cero, el valor del bit permanece igual.
EJM: Es necesario cambiar los bits 6 y 7 de un registro de 16 bits por 1 pero no conocemos que valores tiene ya (0xXXXX), entonces realizamos un enmascaramiento con or:

Como se van a cambiar los bits 6 y 7 y no se van a alterar los demas, escogemos el valor binario con unos en los bits 6 y 7 y 0 en el resto: 0b0000000011000000 el cual corresponde en hexadecimal a 0x00C0. al realizar el OR obtenemos que:
0bXXXXXXXXXXXXXXXX OR 0b0000000011000000 = 0bXXXXXXXX11XXXXXX

3 Código Led-on (Assembler)

El código para encender el led se puede dividir en tres partes: inicialización, configuración de pines y puertos y la salida.

A continuación se mostraran 3 codigos diferentes con 3 formas diferentes para prende el Led del microcontrolador, el cual corresponde al puerto A y pin 5.

3.1 Codigo Led-On-As (Sin variables y funciones)

Todos los códigos como se mencionó anteriormente se pueden dividir en tres partes: inicialización, configuración de puertos y pines y su salida; en este primer código no vamos a encontrar una parte de inicialización, directamente vamos a entrar en la configuración de los registros, pines y salida para que el led configurado se encienda, no se van a utilizar variables, solo registros que nos van a ayudar a configurar los registros AHB2ENR, MODER y ODR.

Lo primero a configurar es el registro RCC-AHB2ENR, que es el registro encargado de encender o activar el clock del puerto que necesitamos en este caso el puerto A.
La explicación de cada línea de código está en la documentación del mismo, sin embargo, no es tan complejo entender cómo funciona, en la línea 17 utilizamos un primer registro (r6) para cargar la dirección RCC-BASE que es la dirección de registro que vamos a trabajar (con 'ldr' le asignamos a r6 el valor del registro) , en la segunda línea de código es decir en la numero 18, encontramos unos corchetes con r6 y el valor de offset del registro AHB2ENR, lo que se hace al meter estos dos valores entre corchetes es sumarlos, para luego con 'ldr' cargarlos a un segundo registro (r5), en la línea 19 se realiza un enmascaramiento ya explicado con 'orr' para colocar un 1 en el bit que necesitamos, en este caso el reloj o clock del puerto A se activa colocando un 1 en el bit 0 en RCC-AHB2ENR que es el valor cargado en el segundo registro (r5), en la línea 20 con 'str' almacenamos el valor de r5 en r6 que es el registro que contiene la dirección base es decir RCC-BASE.

```
7 .syntax unified
8
9 .global main
10
11
12 .section .text.main
13 .type   main,%function
14 main:
15
16 // ACTIVAR RELOJ DEL PUERTO A (bit 0 in AHB2ENR register) (RM0351, page 251)
17     ldr r6, =0x40021000  // Carga en el registro 6 (r6) la direccion RCC_BASE (0x40021000)
18     ldr r5, [r6,#0x4C]  // Utilizamos otro registro (r5) para sumarle el offset del registro AHB2ENR (0x4C)
19     orr r5, 0x00000001 // Activamos el reloj del puerto A colocando un 1 en el bit 0 utilizando un enmascaramiento con orr
20     str r5, [r6,#0x4C] // Almacenamos el valor del segundo registro en el primer registro el cual contiene la direccion del registro AHB2ENR
21
```

En la segunda parte del código se realiza algo similar para configurar el pin como salida, lo primero que tenemos que conocer antes de configurar el pin es saber a que pin y puerto esta conectado el led que vamos a prender, en este caso el led está conectado al pin PA5 es decir puerto A y pin 5, el puerto A ya lo activamos con las anteriores líneas de código, nos falta establecer la configuración del pin que recordemos según el libro puede tener varias configuraciones, en este caso lo configuraremos como salida.

De manera similar tenemos la misma secuencia que la anterior porción de código, a diferencia que en esta porción en la primera línea de código es decir en la línea 23 cargamos en r6 el valor del registro GPIOA BASE, de la misma manera en la línea 24 le sumamos el offset del registro MODER en los corchetes para realizarle un enmascaramiento, en las líneas 25 y 26 se realizan los respectivos enmascaramientos a los bits del registro GPIOA-MODER que es el cargado en r5, recordemos que para este registro los pines se manejan en pares de bits y que para colocar un pin como OUTPUT tenemos que colocar el par de bits correspondientes a ese pin en 01 en este

caso los bits que le corresponden al pin 5 son los bits 10 y 11, hay que colocar el pin 11 en 1 y el pin 10 en 0 para que el pin quede configurado como salida, este proceso se realiza en las líneas 25 y 26, en la última línea de código cargamos el valor configurado de r5 en r6 para configurar el registro GPIOA-MODER y establecer el pin 5 como salida.

```
21
22// CONFIGURAR EL PIN 5 COMO SALIDA (01) (bits 1:0 in MODER register)
23    ldr r6, =0x48000000    // Cargamos en el registro 6 (r6) la direccion GPIOA (0x48000000)
24    ldr r5, [r6,#0x00]    // Utilizamos otro registro(r5) para sumarle el offset del registro MODER(0x00)
25    orr r5, #(1<<10)     //  los bits 10 y 11 que corresponden al pin 5 (RM0351, page 303)
26    bfc r5, #11, #1     // Colocamos el pin 11 como 0 y el pin 10 como 1 para configurar el pin 5 como salida (RM0351, page 303)
27    str r5, [r6,#0x00]  // Cargamos el valor del segundo registro en el primero el cual contiene la direccion GPIOA MODER register
28
```

La última porción de código está dedicada a colocar el pin 5 en HIGH, a través del registro GPIOA-ODR, de igual manera, es similar a las anteriores porciones del código, carga, enmascaramiento y regreso, se carga en r6 el valor de GPIOA base, se suma el offset del registro ODR, se hace el enmascaramiento al bit 5 que corresponde al pin 5 en esta dirección de registro (GPIOA-ODR), y regresamos el valor a r6 para que se configure el registro GPIOA-ODR y el pin se establezca en HIGH y se prenda el led.

```
21
22// CONFIGURAR EL PIN 5 COMO SALIDA (01) (bits 1:0 in MODER register)
23    ldr r6, =0x48000000    // Cargamos en el registro 6 (r6) la direccion GPIOA (0x48000000)
24    ldr r5, [r6,#0x00]    // Utilizamos otro registro(r5) para sumarle el offset del registro MODER(0x00)
25    orr r5, #(1<<10)     //  los bits 10 y 11 que corresponden al pin 5 (RM0351, page 303)
26    bfc r5, #11, #1     // Colocamos el pin 11 como 0 y el pin 10 como 1 para configurar el pin 5 como salida (RM0351, page 303)
27    str r5, [r6,#0x00]  // Cargamos el valor del segundo registro en el primero el cual contiene la direccion GPIOA MODER register
28
```

De la línea 38 a 43 es una porción de código a no tener en cuenta en estos momentos.

3.2 Codigo Led-On-Bs (Con variables y sin funciones)

Este código funciona de la misma manera que el código anterior, la única diferencia es que en este código si hay una inicialización de variables al principio del código.

En la imagen siguiente, encontraremos la inicialización del código utilizando variables, estas variables se establecen utilizando el código '.equ' el cual asigna valores a algunas variables que queramos usar o vayamos a usar, como vemos utilizamos los valores y variables ya explicadas en el anterior código, por ejemplo el valor de RCC-BASE ya lo declaramos o lo guardamos en una variable llamada de la misma manera 'RCC-BASE' utilizando el '.equ' en la línea 18, esto nos facilitara trabajar en las siguientes porciones del código, pues ya trabajamos con los nombres de las variables mas no con los valores de los registros que muchas veces son muy fáciles de confundir.

```
 9 .syntax unified
10
11 .global main
12
13 // Constants defined in file stm32l476xx_constants.S
14 //   RCC   base address is 0x40021000
15 //   AHB2ENR register offset is 0x4C
16 //   RCC   base address is 0x40021000
17 //   APB2ENR register offset is 0x60
18 .equ    RCC_BASE,       0x40021000
19 .equ    RCC_AHB2ENR,    0x4C // RCC AHB2 peripheral clock reg (RM0351, page 251)
20
21
22
23 //   GPIOA base address is 0x48000000
24 //   MODER register offset is 0x00
25 //   ODR   register offset is 0x14
26 .equ    GPIOA_BASE,     0x48000000 // GPIO BASE ADDRESS (RM0351, page 78)
27
28 .equ    GPIO_MODER,     0x00 // GPIO port mode register (RM0351, page 303)
29 .equ    GPIO_ODR,       0x14 // GPIO outpu data register (RM0351, page 306)
30
```

La siguiente porción del código es exactamente igual a lo explicado en el punto 3.1, tiene la misma secuencia, las mismas variables y el mismo orden a excepción de que en esta ocasión,

utilizamos las variables declaradas al inicio del código, por ejemplo para cargar el valor del registro RCC-BASE en r6 ya no colocamos el valor 0x40021000 debido a que este valor ya se lo declaramos a la variable 'RCC-BASE', de la misma manera en los corchetes el valor del offset ya esta dado por la variable 'RCC-AHB2ENR', el enmascaramiento es igual y la salida de cada configuración de registro sigue siendo de la misma manera que en el código del punto 3.1

```
.section .text.main
.type   main,%function
main:

// Enable GPIOA Peripheral Clock (bit 0 in AHB2ENR register) (RM0351, page 251)
    ldr r6, =RCC_BASE       // Load peripheral clock reg address to r6
    ldr r5, [r6,#RCC_AHB2ENR]   // Read its content to r5
    orr r5, 0x00000001  // AHB2 peripheral clock enable register (RCC_AHB2ENR) (RM0351, PAGE 251
    str r5, [r6,#RCC_AHB2ENR]

// Make GPIOA Pin5 as output pin (bits 1:0 in MODER register)
    ldr r6, =GPIOA_BASE       // Load GPIOA base register address to r6
    ldr r5, [r6,#GPIO_MODER]    // Read the content the MODER register to r5
    orr r5, #(1<<10)            // Set bits 11, 10 for P5 (RM0351, page 303)
    bfc r5, #11, #1             // Write 01 to bits 11, 10 for P5 (RM0351, page 303)
    str r5, [r6,#GPIO_MODER]    // Store result in GPIOA MODER register


//Set pin A5
    ldr r6, =GPIOA_BASE       // Load GPIOA MODER register address to r6
    ldr r5, [r6,#GPIO_ODR]    // Read ODR content to r5
    orr r5, #(1<<5)             // Set P5 (RM0351, page 306)
    str r5, [r6,#GPIO_ODR]  // Store result in GPIOA MODER register

```

3.3 Codigo Led-On-Cs (Con funciones)

4 Referencias

+ PM0214 Programming manual, STM32 Cortex-M4 MCUs and MPUs programming manual.
+ RM0351 Reference manual, STM32L4x5 and STM32L4x6 advanced Arm-based 32-bit MCUs.
+ Repositorio GITHUB