

MICROCONTROLADORES

Práctica No. 3. Lectura y Escritura de un Puerto.

1. Objetivo

- Transferir el contenido de un puerto configurado como entrada a otro puerto configurado como salida.
- Configurar un proyecto en el STM32CubeIDE.
- Conectar 8 Leds a los pines PA0 – PA7 y un DIPSWITCH de 8 posiciones conectado a los pines PB8 – PB15.

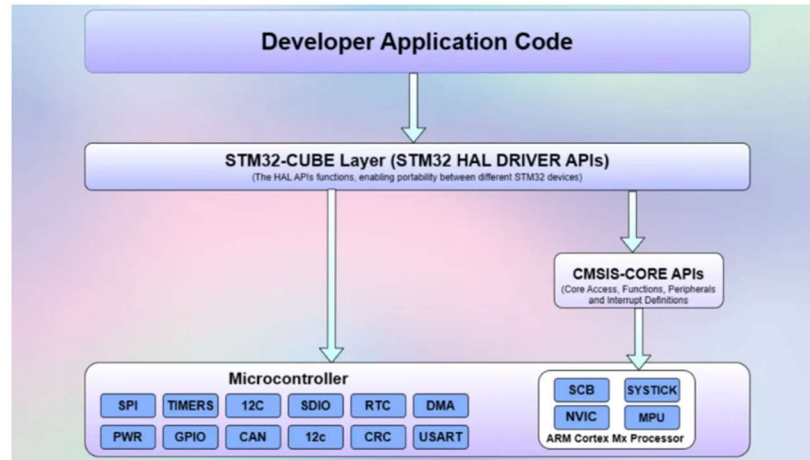
2. Material y Equipo.

- Computador o laptop con el STM32CubeIDE.
- Ocho leds de 5mm o 3mm.
- Un DIPSWITCH de 8 posiciones.
- Ocho resistencias de 330Ω o 220Ω.

3. Marco de Referencia

El STM32CubeIDE es un IDE basado en eclipse con mucha potencia para el desarrollo de programas y con una gran facilidad para depurarlos. De lado del programador tenemos una ayuda extra ya que junto con el STM32CubeIDE se integra el STM32CubeMX que es la herramienta visual que usamos para configurar el hardware del STM32 y después generar el código esto es de gran ayuda por que realizamos la inicialización del STM32 de una forma rápida e intuitiva en lugar de estudiar semanas la hoja de datos para entender la organización de memoria, configuración del oscilador, etc.

Otra herramienta muy importante del CubeMX es que cuando generamos nuestro código se crea una estructura o esqueleto de nuestro programa y el código generado se crea a partir de unas librerías denominadas HAL (Hardware Abstraction Layer), nuestro programa se crea como se muestra en la siguiente figura.



Tenemos básicamente cuatro bloques que van desde el bloque de mas alto nivel que es la aplicación del usuario dos bloques intermedios y un bloque inferior. Las librerías HAL no ayudan a abstraer toda la capa inferior del usuario en otras palabras a no tener que trabajar a nivel registro con los periféricos del microcontrolador funcionando de la siguiente manera.

El “Developer Application Code” es nuestra aplicación final por ejemplo un medidor de temperatura donde leemos un LM35 y desplegamos la temperatura en un LCD alfanumérico en este punto solo nos preocupamos por el código de nuestra aplicación y el programador se centra en desarrollarlo. La capa intermedia nos ayuda a conectar nuestro programa (capa superior) con el hardware o periféricos del STM32 (capa inferior) es donde nosotros usamos las librerías HAL tenemos librerías para cada periférico integrado en el STM32 (GPIO, ADC, SPI, I2C, etc.) esto acelera mas el desarrollo de nuestra aplicación por que solo usamos funciones para leer o escribir en cierto periférico si tener que ir directamente al hardware y ver registros, banderas de estado, hacer lazos de espera a que el periférico termine de realizar cierta tarea de eso ya se encarga la función que

nosotros estemos utilizando y después de la llamada a la función podemos o no tener un resultado de regreso.

Otro bloque muy importante es el CMSIS (Common Microcontroller Software Interface Standard) dentro de este bloque se encuentran unas librerías comunes a todos los ARM-Cortex que hay en el mercado no importa su fabricante estas librerías son desarrolladas directamente por ARM y son específicas del procesador (Cortex-M0/M0+, Cortex-M3, Cortex-M4, Cortex-M7, etc.) dentro de estas librerías tenemos instrucciones propias de cada procesador y también librerías específicas para ciertas aplicaciones avanzadas como el CMSIS-DSP o el CMSIS-RTOS, etc.

El bloque del CMSIS es importante ya que las librerías HAL hacen uso de el llamando a funciones del CMSIS desde la capa de las HAL por ejemplo para hacer un reset por software necesitamos modificar un bit de un registro directo del procesador y esta función esta dentro del CMSIS y dentro de las HAL se hace la llamada a la función de reset de las librerías del CMSIS.

A nivel de las librerías HAL los GPIOs tienen los siguientes componentes dentro de sus archivos.

Para los GPIO tenemos dos estructuras básicas y varias funciones de las cuales hemos usado unas en las practicas anteriores. Primero tenemos que ver las estructuras que maneja las HAL a la hora de generar el código con el CubeMX.

```
typedef struct {  
    volatile uint32_t MODER;  
    volatile uint32_t OTYPER;  
    volatile uint32_t OSPEEDR;  
    volatile uint32_t PUPDR;  
    volatile uint32_t IDR;  
    volatile uint32_t ODR;  
    volatile uint32_t BSRR;  
    volatile uint32_t LCKR;  
    volatile uint32_t AFR[2];  
    volatile uint32_t BRR;  
} GPIO_TypeDef;
```

Esta primera estructura es la que no da acceso a los registros de los GPIOs que trae cada STM32. Lo siguiente es crear una variable tipo puntero del tipo de dato (GPIO_TypeDef) de la estructura anterior y se inicializa con la dirección de memoria del puerto que vayamos a usar de la siguiente forma.

```
GPIO_TypeDef *GPIOA = 0x48000000;
```

De esta forma creamos una variable tipo puntero y esta apunta a la dirección 0x48000000 que corresponde al GPIOA y así se hace lo mismo para cada puerto que tenga disponible el STM32 con el que estemos trabajando.

La segunda estructura es la que el CubeMX usa a la hora de generar el código y es la que nosotros también podemos usar si deseamos realizar un cambio en un pin (entrada o salida) del STM32 cuando se esté ejecutando la aplicación final.

```
typedef struct {  
    uint32_t Pin;  
    uint32_t Mode;  
    uint32_t Pull;  
    uint32_t Speed;  
    uint32_t Alternate;  
} GPIO_InitTypeDef;
```

Como ejemplo de esta estructura tenemos el siguiente código.

```
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0}; ← ①

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin : LED_Pin */
    GPIO_InitStruct.Pin = LED_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(LED_GPIO_Port, &GPIO_InitStruct); ②
}
```

Lo primero que tenemos que hacer es crear una variable del tipo de la estructura anterior (GPIO_InitTypeDef) como se muestra en la declaración marcada con el número 1, después usamos esta variable e inicializamos la estructura con los valores deseados para el pin que vayamos a usar dependiendo del comportamiento que queramos que tenga al final llamamos a la función “HAL_GPIO_Init” que tiene la siguiente sintaxis.

```
void HAL_GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_Init)
```

Esta función nos pide dos parámetros el primero un puntero al puerto que vamos a usar creado con la primera estructura que vimos (GPIO_TypeDef), el segundo argumento es un puntero a la segunda estructura que vimos (GPIO_InitTypeDef) en esta le pasamos la configuración que queremos para uno o varios pines.

Las macros que podemos usar en la estructura “GPIO_InitTypeDef” en el miembro de la estructura “Mode” son las siguientes.

MODO	DESCRIPCION
GPIO_MODE_INPUT	Entrada Flotante
GPIO_MODE_OUTPUT_PP	Salida Push-Pull
GPIO_MODE_OUTPUT_OD	Salida Open Drain
GPIO_MODE_AF_PP	Funcion Alterna Push-Pull
GPIO_MODE_AF_OD	Funcion Alterna Open Drain
GPIO_MODE_AF_INPUT	Entrada con Funcion Alterna
GPIO_MODE_ANALOG	Entrada Analogica
GPIO_MODE_IT_RISING	Interrupcion Externa con deteccion de flanco de subida
GPIO_MODE_IT_FALLING	Interrupcion Externa con deteccion de flanco de bajada
GPIO_MODE_IT_RISING_FALLING	Interrupcion Externa con deteccion de flanco de subida y bajada
GPIO_MODE_EVT_RISING	Evento Externo con deteccion de flanco de subida
GPIO_MODE_EVT_FALLING	Evento Externo con deteccion de flanco de bajada
GPIO_MODE_EVT_RISING_FALLING	Evento Externo con deteccion de flanco de subida y bajada

Las macros que podemos usar en el miembro de la estructura “Pull” (GPIO_InitTypeDef) son las siguientes.

MODO	DESCRIPCION
GPIO_NOPULL	Pull-Up y Pull-Down deshabilitadas
GPIO_PULLUP	Pull-Up Activada
GPIO_PULLDOWN	Pull-Down Activada

Las macros que podemos usar en el miembro de la estructura “Speed” (GPIO_InitTypeDef) son las siguientes.

MODO	DESCRIPCION
GPIO_SPEED_FREQ_LOW	Pin en Modo de Velocidad Baja (2 MHz)
GPIO_SPEED_FREQ_MEDIUM	Pin en Modo de Velocidad Media (5 MHz)
GPIO_SPEED_FREQ_HIGH	Pin en Modo de Velocidad Alta (10 MHz)

Las funciones disponibles para los GPIOs son las siguientes.

Para leer el estado de un pin del STM32.

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

Para cambiar el estado (Encendido o Apagado) de un pin del STM32.

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
```


Para cambiar el estado de un pin (Toggleo) del STM32.

```
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

Para evitar cambiar el estado de un pin (bloquear).

```
HAL_StatusTypeDef HAL_GPIO_LockPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

Por último, tenemos una función contraria al “HAL_GPIO_Init” usada para resetear el estado del GPIO (regresarlo a su estado inicial después del reset como entrada flotante).

```
void HAL_GPIO_DeInit(GPIO_TypeDef *GPIOx, uint32_t GPIO_Pin)
```

Desafortunadamente no tenemos una función de las HAL que nos permita modificar un puerto completo o leer un puerto completo para estos casos usamos las definiciones de los puertos creados a partir de la estructura de los registros de cada puerto (primera estructura vista) que ya viene definida en las librerías del CMSIS.

```
#define GPIOA      ((GPIO_TypeDef *)GPIOA_BASE)
#define GPIOB      ((GPIO_TypeDef *)GPIOB_BASE)
#define GPIOC      ((GPIO_TypeDef *)GPIOC_BASE)
#define GPIOD      ((GPIO_TypeDef *)GPIOD_BASE)
#define GPIOE      ((GPIO_TypeDef *)GPIOE_BASE)
```

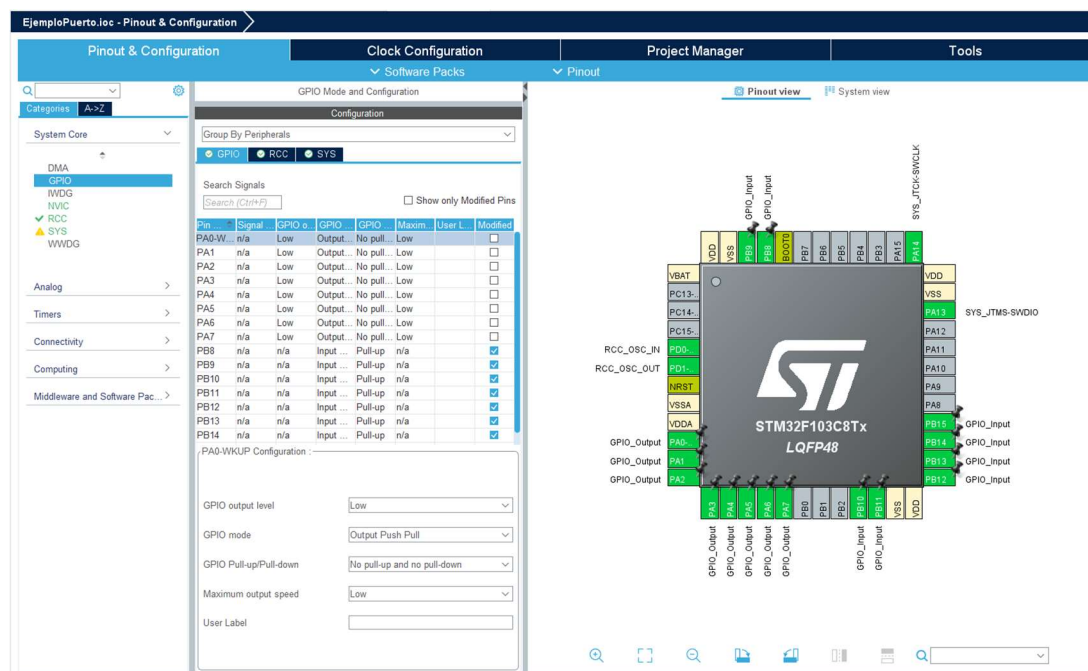
Usando estas definiciones podemos tener acceso a los registros de cada puerto y podemos usar los registros que nos dan acceso a la lectura o escritura usando el operador flecha (->) ya que cada definición como se puede ver es un puntero a la estructura “GPIO_TypeDef”. La forma de usarlas se muestra a continuación.

```
GPIOA->MODER |= 0x400;
GPIOA->ODR   |= 0x20;
```

Ya que tenemos un conocimiento de la configuración de los pines usando las librerías HAL junto con las del CMSIS vamos a realizar un programa que haga una transferencia de un dato ingresado por un DIPSWITCH conectado a los pines PA0 – PA7 y lo que leamos será transferido a los pines PB8 – PB15 donde habrá 8 leds conectados.

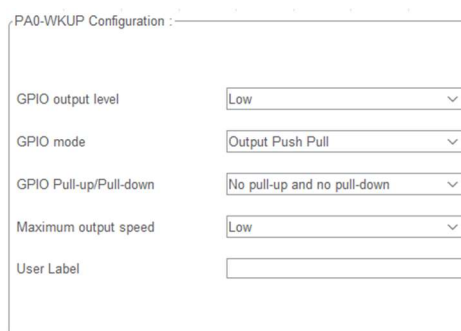
4. Desarrollo y Procedimiento.

Se creará un proyecto en el STM32CubeIDE como se indicó anteriormente. La configuración del STM32 queda de la siguiente manera.



Como se puede ver en esta practica no se usarán etiquetas para cada pin ya que vamos a leer 8 bits de un puerto y escribir esos 8 bits en otro puerto y no usaremos etiquetas individuales de pines.

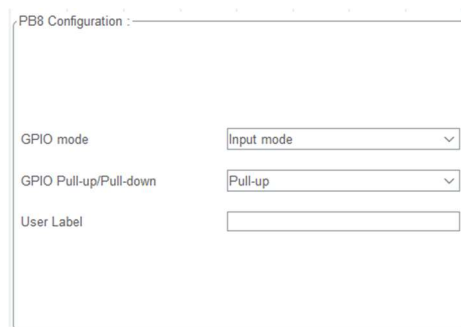
La siguiente imagen muestra la configuración de cada pin que será usado como salida.



PA0-WKUP Configuration :

GPIO output level	Low
GPIO mode	Output Push Pull
GPIO Pull-up/Pull-down	No pull-up and no pull-down
Maximum output speed	Low
User Label	

La siguiente imagen muestra la configuración de cada pin que será usado como entrada.



PB8 Configuration :

GPIO mode	Input mode
GPIO Pull-up/Pull-down	Pull-up
User Label	

La siguiente imagen muestra el código principal del archivo main.c. Recuerde que el siguiente código debe estar entre los comentarios “USER CODE BEGIN” y “USER CODE END”.

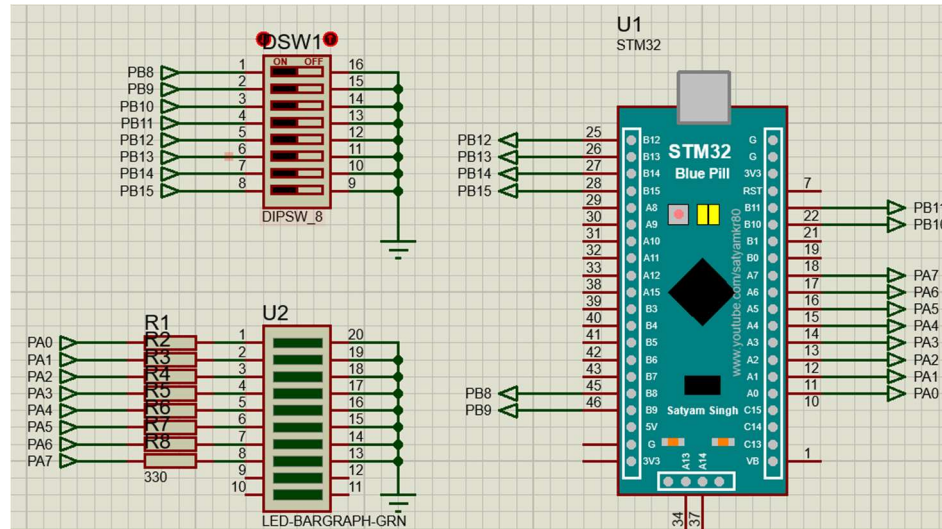
```
1 #include "main.h"
2
3 void SystemClock_Config(void);
4 static void MX_GPIO_Init(void);
5
6 int main(void)
7 {
8     HAL_Init();
9
10    SystemClock_Config();
11
12    MX_GPIO_Init();
13
14    while (1)
15    {
16        GPIOA->ODR = GPIOB->IDR >> 8;
17    }
18 }
```

A continuación, se muestra la imagen del segmento de código de la configuración de los pines usando las estructuras y funciones mencionadas anteriormente.

```
64 static void MX_GPIO_Init(void)
65 {
66     GPIO_InitTypeDef GPIO_InitStruct = {0};
67
68     /* GPIO Ports Clock Enable */
69     __HAL_RCC_GPIOC_CLK_ENABLE();
70     __HAL_RCC_GPIOA_CLK_ENABLE();
71     __HAL_RCC_GPIOB_CLK_ENABLE();
72
73     /*Configure GPIO pin Output Level */
74     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
75                       |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_RESET);
76
77     /*Configure GPIO pins : PA0 PA1 PA2 PA3
78                          PA4 PA5 PA6 PA7 */
79     GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
80                       |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
81     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
82     GPIO_InitStruct.Pull = GPIO_NOPULL;
83     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
84     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
85
86     /*Configure GPIO pins : PB10 PB11 PB12 PB13
87                          PB14 PB15 PB8 PB9 */
88     GPIO_InitStruct.Pin = GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13
89                       |GPIO_PIN_14|GPIO_PIN_15|GPIO_PIN_8|GPIO_PIN_9;
90     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
91     GPIO_InitStruct.Pull = GPIO_PULLUP;
92     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
93 }
```

5. Esquemático del circuito.

El circuito de la práctica se muestra a continuación.



6. Mejora.

Modifique el programa para que después de leer el puerto le suma una constante (número entero) y lo despliegue en los leds.

7. Observaciones.

Esta sección es para que el alumno anote sus observaciones.

8. Conclusiones.

Esta sección es para que el alumno anote sus conclusiones.

9. Importante.

La práctica deberá ser validada en el salón de clases antes de anexar el reporte al manual de prácticas. Una vez validada realizar el reporte de práctica como se anteriormente y anexar al manual de prácticas que se entregará a final del curso.