# MICROCONTROLADORES

## Práctica No. 2. Entradas Digitales.

### 1. Objetivo

- Entender el funcionamiento de los registros GPIOx_CRL, GPIOx_CRH, GPIOx_IDR y GPIOx_ODR cuando queremos configurar un pin como entrada.
- Configurar un proyecto STM32CubeIDE.
- Conectar un LED en el pin PC13 y un PUSHBUTTON en el pin PA0. Con el botón vamos a encender o apagar el led dependiendo del estado del botón.

### 2. Material y Equipo.

- Computador o laptop con el STM32CubeIDE.
- Un led de 5mm o 3mm.
- Un Push button normalmente abierto.
- Una resistencia de 330Ω o 220Ω.

### 3. Marco de Referencia

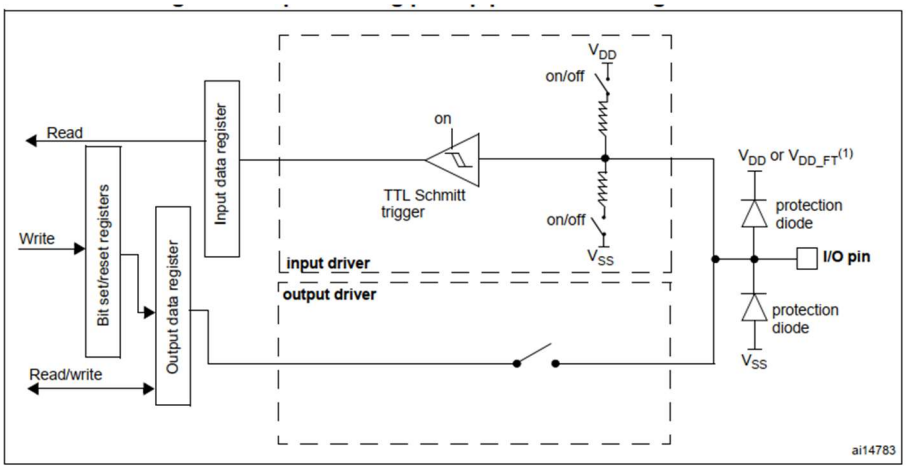
En la practica anterior vimos como configurar los pines del STM32F103C8T6 como salidas para poder parpadear un led a intervalos diferentes de tiempo en esta práctica vamos a ver como configurar cualquier pin del MCU como entrada y de cuantas opciones tenemos cuando un pin esta como entrada. Como en la practica anterior los registros encargados de configurar los pines del MCU como entrada, salida o analógica son el GPIOx_CRL y el GPIOx_CRH. Un pin configurado como entrada tiene las siguientes configuraciones.

- Modo Analogico.
- Entrada Flotante (estado inicial después del reset)
- Entrada con Pull-up o Pull-down.

La siguiente tabla nos muestra como configurar un pin como entrada y como habilitar la resistencia de pull-up o pull-down.

| Configuration mode | | CNF1 | CNF0 | MODE1 | MODE0 | PxODR register |
|---|---|---|---|---|---|---|
| General purpose output | Push-pull | 0 | 0 | 01<br>10<br>11<br>see *Table 21* | | 0 or 1 |
| | Open-drain | | 1 | | | 0 or 1 |
| Alternate Function output | Push-pull | 1 | 0 | | | Don't care |
| | Open-drain | | 1 | | | Don't care |
| Input | Analog | 0 | 0 | 00 | | Don't care |
| | Input floating | | 1 | | | Don't care |
| | Input pull-down | 1 | 0 | | | 0 |
| | Input pull-up | | | | | 1 |

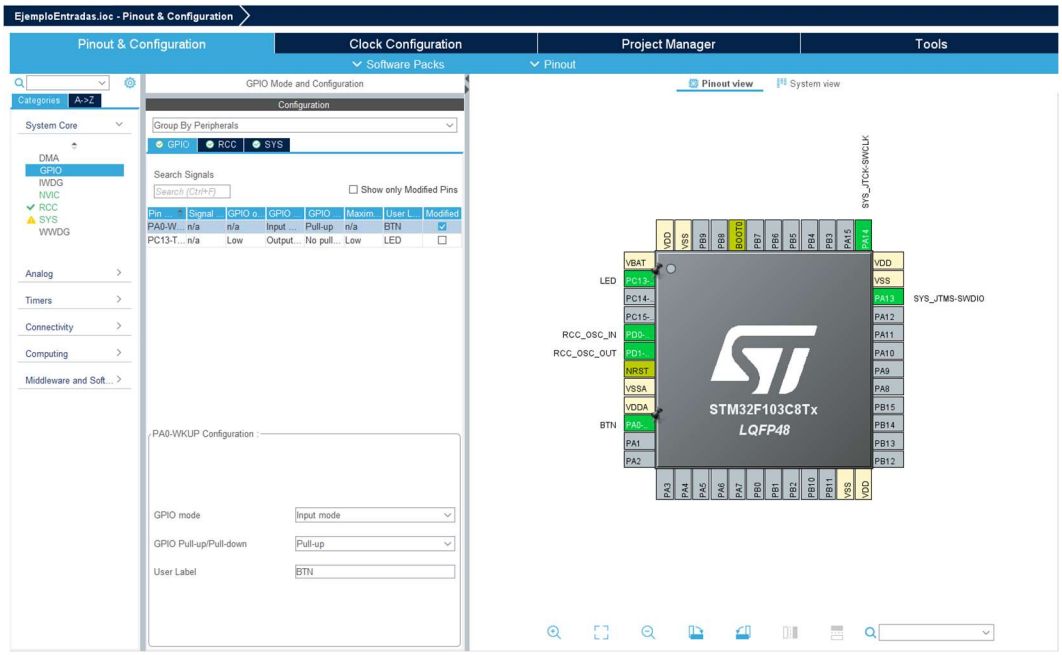
Entonces en modo entrada los bits CNF1 y CNF0 del registro GPIOx_CRL o GPIOx_CRH deben ser "10" respectivamente después los bits MODE1 y MODE0 de dichos registros deben ser "00" de esa forma ya tenemos configurado nuestro(s) pin(es) como entradas flotantes, si queremos habilitar la resistencia de pull-up o pull-down usamos el registro GPIOx_ODR utilizado para sacar un estado alto o bajo en cualquier pin del MCU pero cuando el pin esta configurado como salida un 1 o un 0 en cualquier bit de este registro habilitara la resistencia del pull-up o pull-down respectivamente. La siguiente figura tenemos el diagrama simplificado de un pin del MCU configurado como entrada.



Lo primero que tenemos que notar es que el bloque correspondiente a la configuración de salida (output driver) es abierta (deshabilitada), la entrada de Schmitt Trigger es habilitado y las resistencias de pull-up o pull-down pueden ser habilitadas o deshabilitadas según convenga. El dato presente en el buffer de entrada (Input Data Register) es muestreado cada ciclo del APB2.

## 4. Desarrollo y Procedimiento.

Se creará un proyecto en el STM32CubeIDE como se indicó en la práctica no. 1, le asignamos el nombre al proyecto de "EjemploEntradas" configuramos el oscilador a 72MHz como ya se indicó en la práctica anterior, después asignamos los pines como se muestra a continuación y generamos el código.



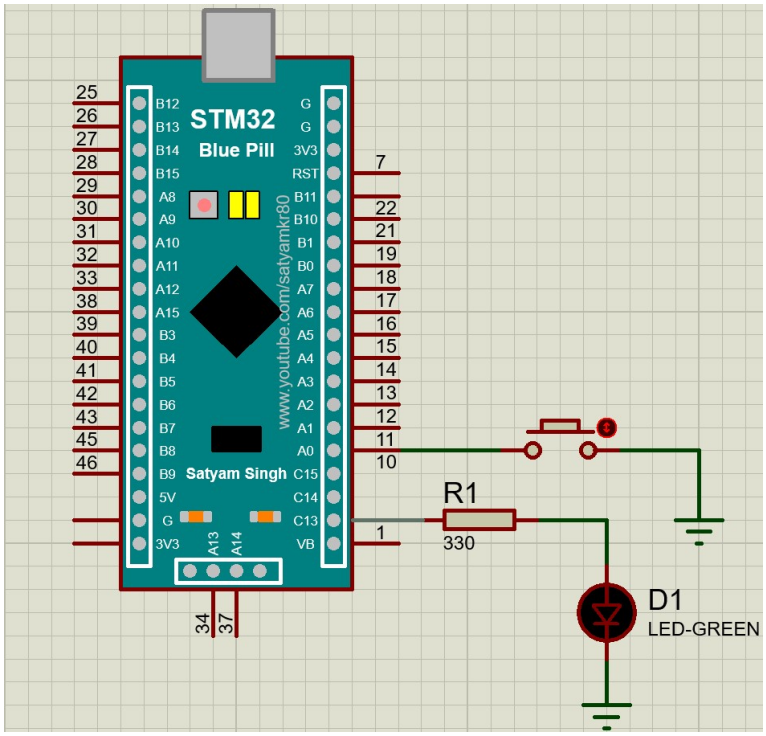
El código de la práctica es el siguiente.

```c
#include "main.h"

void SystemClock_Config(void);
static void MX_GPIO_Init(void);

int main(void)
{
    HAL_Init();

    SystemClock_Config();

    MX_GPIO_Init();

    while (1)
    {
        if(HAL_GPIO_ReadPin(BTN_GPIO_Port, BTN_Pin) == GPIO_PIN_RESET)
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
        else
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
    }
}

void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the CPU, AHB and APB buses 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_DIV2;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
    {
        Error_Handler();
    }
}
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* 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);

    /*Configure GPIO pin : BTN_Pin */
    GPIO_InitStruct.Pin = BTN_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(BTN_GPIO_Port, &GPIO_InitStruct);
}
```

```
84  void Error_Handler(void)
85  {
86    __disable_irq();
87    while (1)
88    {
89    }
90  }
91
92  #ifdef  USE_FULL_ASSERT
93  /**
94    * @brief  Reports the name of the source file and the source line number
95    *         where the assert_param error has occurred.
96    * @param  file: pointer to the source file name
97    * @param  line: assert_param error line source number
98    * @retval None
99    */
100 void assert_failed(uint8_t *file, uint32_t line)
101 {
102   /* USER CODE BEGIN 6 */
103   /* User can add his own implementation to report the file name and line number,
104      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
105   /* USER CODE END 6 */
106 }
107 #endif /* USE_FULL_ASSERT */
108
```

## 5. Simulación y Programación de la tarjeta Arduino.

El diagrama del circuito se muestra a continuación.



## 6. Observaciones.

Esta sección es para que el alumno anote sus observaciones.

## 7. Conclusiones.

Esta sección es para que el alumno anote sus conclusiones.

## 8. Importante.

La práctica deberá ser validad en el salón de clases antes de anexar el reporte al manual de prácticas. Una ves validad realizar el reporte de practica como se indicó en la práctica no. 1 y anexar al manual de practicas que se entregara a final del curso.