Project 1.2: Read status of push button switch and write to LED (on/off) on STM32F407G-DISC1 Board

In this sub-project we are going to:

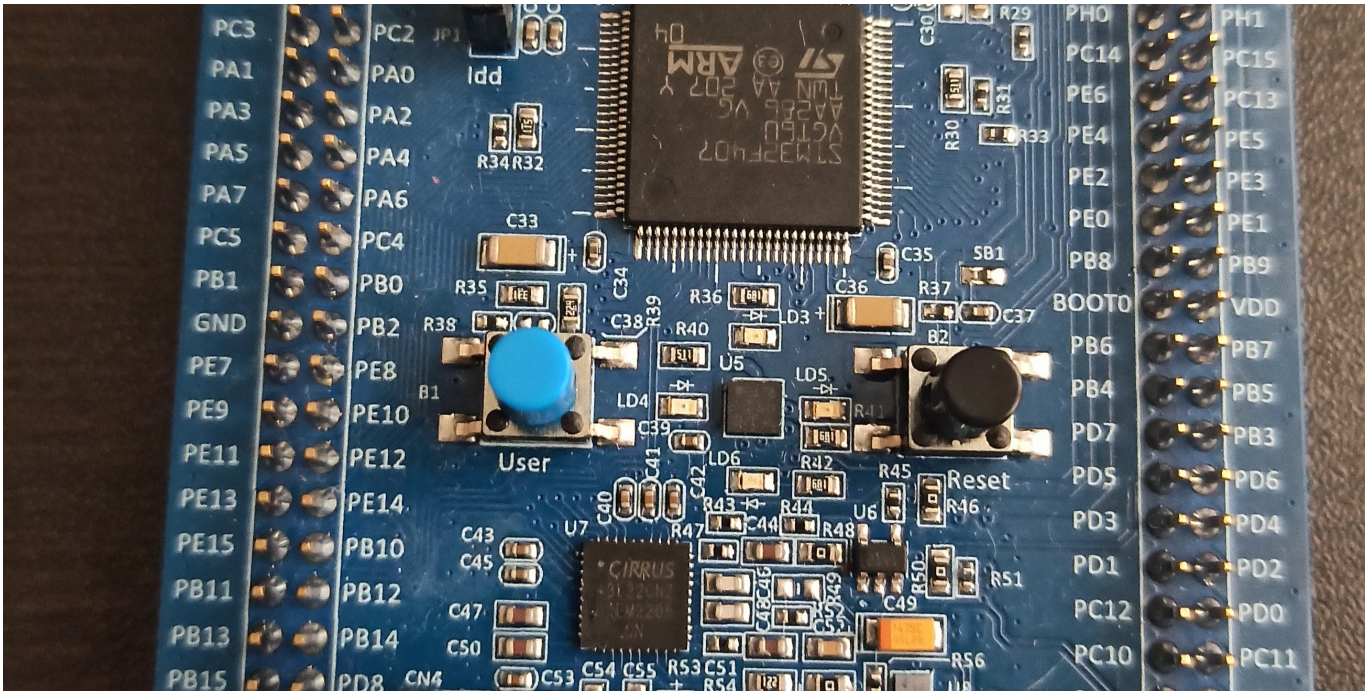
1. Read the status(on/off) of push button switch
2. Reflect the status of the switch to the LED on STM32F407G-DISC1 Board.

Upon completion of this project, you will learn.

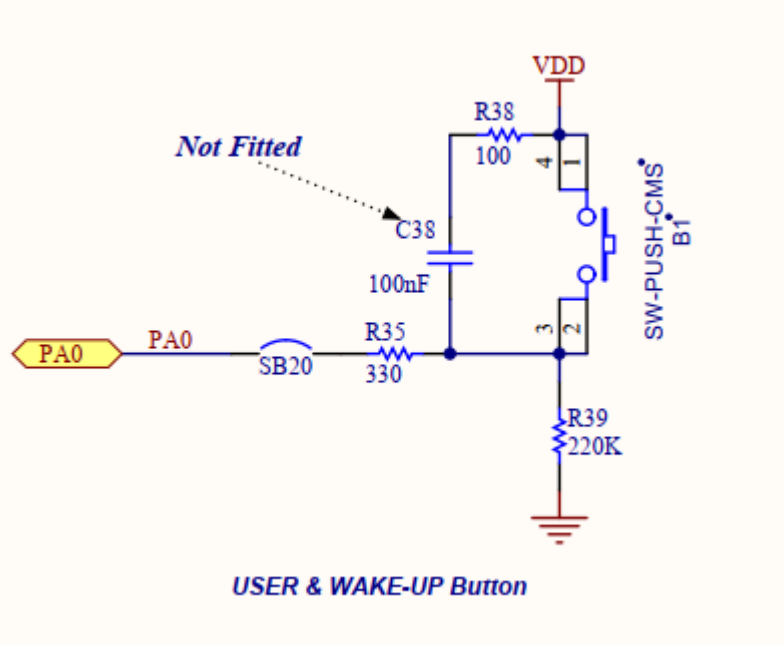
- To identify the Correct GPIO pin connect to the push button switch
- How push button switch is connected to Microcontroller.
- To refer datasheet and user manual when required.
- To work with GPIO registers.
- To work with C programming.

From STM32F407G-DISC1 user manual(section 6.4) we know that there are two user push buttons (B1 and B2).

1. B1 USER (Blue): User and Wake-Up buttons are connected to the I/O PA0 of the STM32F407VG.
2. B2 RESET (Black): Push button connected to NRST is used to RESET the STM32F407VG.



Below is the schematic extract for the push button(B1) circuit:



Notice the resistor and Switch symbols.

Switch circuit is directly connected to micro-controller ports hence microcontroller pins are driving the switch.

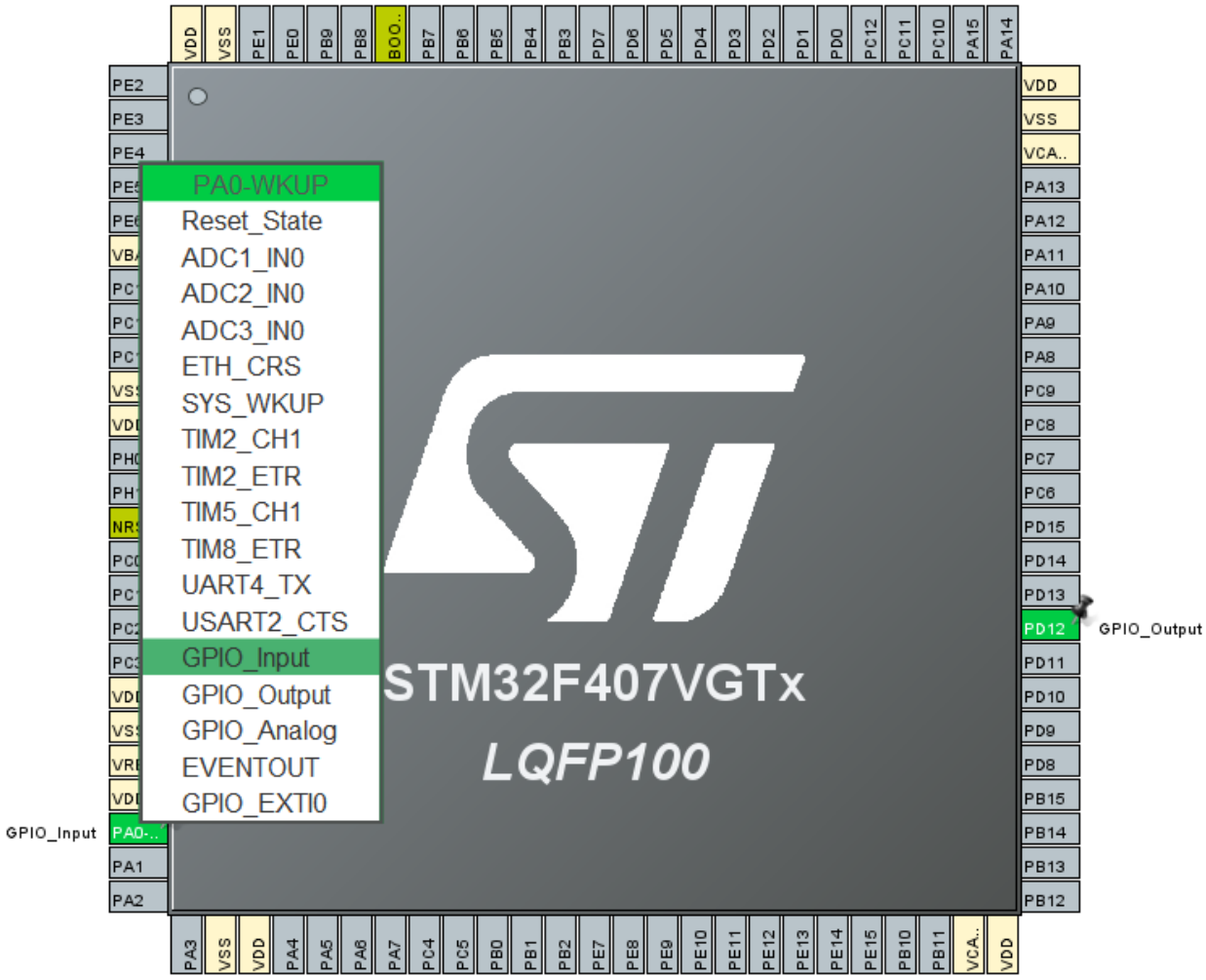
It works in the following way:

1. When the push button is open - the port PA0 pin sees 0 : Logic Low (~0V).
2. When the push button is close - the port PA0 pin sees 1 :Logic High (~3.3V/VDD) due to resistor R39(220K ohm).

We are going to use the same LED, User LD4 which we used in Project 1.1 with the same configuration.

As we have covered setting up of dev env earlier, we should be having our development environment to work with STM32.

First we need to open EfA_STM32_dev.ioc(open .ioc file of your project). Click on PA0 in the chip diagram and select GPIO_Input.



Click Save and hit Yes to enable generating code automatically. The STM32CubeIDE (from now we simply call it as IDE) generates the code for us to put the user code in appropriate sections. Then we build, flash code to run on the target device.

As main.c file gets regenerated every time we update the .ioc file, let's not update the main.c file with our user code directly.

Let's put our Switch status to LED functionality in a separate file within a function and call this function in the main() function of main.c file. By separating we make our code **modular**, which makes it easy to **extend and maintain**. This concept of modularity is very important when we work with large projects.

Create file "switch_read_led_write.h" in /Core/Inc folder with following content.

```
#ifndef INC_SWITCH_READ_LED_WRITE_H_
#define INC_SWITCH_READ_LED_WRITE_H_


/* Includes ------------------------------------------------------------*/
#include "stm32f4xx_hal.h"

/**
  * @brief  GPIO Bit SET and Bit RESET enumeration
  */
```

```
typedef enum
{
  PIN_RESET = 0,
  PIN_SET
}PinState;

/*
 * Function to Read from GPIOA PIN 0 and Write to GPIOD PIN 12.
 */
void Read_GPIOA_PIN_0_Write_GPIOD_PIN_12(void);


#endif /* INC_SWITCH_READ_LED_WRITE_H_ */
```

In this file we:

1. Define enumerator(enum) to contain pin states i.e RESET(0) and SET(1).

   ```
   typedef enum
   {
       PIN_RESET = 0,
       PIN_SET
   }PinState;
   ```

   In C, an enumeration (enum) is a user-defined data type that consists of a set of named integer constants. Enums are used to make code more readable and maintainable by giving meaningful names to integer values, instead of using raw numbers. Here's the basic syntax for defining an enum in C:

   ```
   enum enum_name
   {
   constant1,
   constant2,
   constant3,
   // ...
   };
   ```

2. Include "stm32f4xx_hal.h" file to use the functionalities provided by the HAL Library. In C programming, #include is a preprocessor directive used to include the contents of another file into your source code.

3. Declare a function with prototype:

   ```
   void Read_GPIOA_PIN_0_Write_GPIOD_PIN_12(void);
   ```

   Name of the function is *Read_GPIOA_PIN_0_Write_GPIOD_PIN_12*. Its argument list is *void* type means takes noting and return type is *void* means returns nothing.

Then, create file "switch_read_led_write.c" in /Core/Src folder with following content.

```
#include "switch_read_led_write.h"

/*
 * Function to Read from GPIOA PIN 0 and Write to GPIOD PIN 12.
 */
void Read_GPIOA_PIN_0_Write_GPIOD_PIN_12(void)
{
    /* declare variable to store the GPIO pin status*/
    uint32_t status;
    /*
    * Read the pin status of GPIO port A and Pin 0
    * and store the same in status variable.
    */
    status = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);

    /*
    * Check if pin status is SET or REST
    */
    if(status == PIN_SET)
    {
        /* Write the 1 (SET) to GPIOD pin 12*/
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, PIN_SET);
    }
    else
    {
        /* Write the 0 (RESET) to GPIOD pin 12*/
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, PIN_RESET);
    }
}
```

In this file we:

1. Include the "switch_read_led_write.h".
2. Define the Read_GPIOA_PIN_0_Write_GPIOD_PIN_12() function which we had declared in the "switch_read_led_write.h"
    - Declared variable "status" to store the GPIO pin status.
    - Read the pin status of GPIO port A and Pin 0 using HAL library function call *HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)* and assign the return result to status variable.
        1. GPIOA is a macro which expands to base address of GPIOD ((GPIO_TypeDef *) ((0x40000000UL + 0x00020000UL) + 0x0000UL)) We can check the same from reference manual. Table 1. STM32F4xx register boundary addresses 0x4002 0000 - 0x4002 03FF for GPIOA
        2. GPIO_PIN_0(0x0001) which is 1st pin.
    - Check if pin status is SET or REST
        1. If pin status is SET Write the 1 (SET) to GPIOD pin 12 using HAL library function call *HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, PIN_SET);*

2. If pin status is RESET Write the 0 (RESET) to GPIOD pin 12 using HAL library function call *HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, PIN_RESET);*

Now lets update the main.c file:

1. Include the *switch_read_led_write.h* file at the top.

```
#include "switch_read_led_write.h"
```

By including the "switch_read_led_write.h" header, the compiler knows about the Read_GPIOA_PIN_0_Write_GPIOD_PIN_12 function's prototype (declaration), allowing you to call it in your main.c file even though the actual implementation of Read_GPIOA_PIN_0_Write_GPIOD_PIN_12 is in a different source file i.e *switch_read_led_write.c*.

2. Program starts from beginning of main() function and executes in sequentially till the end of main.
3. The first function call is **HAL_Init()**. This function is used to initialize the HAL(Hardware Abstraction Layer) Library. it performs the following:
    - Configure the Flash prefetch, instruction and Data caches.
    - Configures the SysTick to generate an interrupt each 1 millisecond, which is clocked by the HSI
    - Set NVIC Group Priority to 4
    - Calls the HAL_MspInit() callback function defined in user file "stm32f4xx_hal_msp.c" to do the global low level hardware initialization.
4. Then we call **SystemClock_Config()** which configure the system clock.
5. Then **MX_GPIO_Init()** which initialize all configured peripherals. Notice the following code:

```
  /*Configure GPIO pin : PA0 */
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /*Configure GPIO pin : PD12 */
    GPIO_InitStruct.Pin = GPIO_PIN_12;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

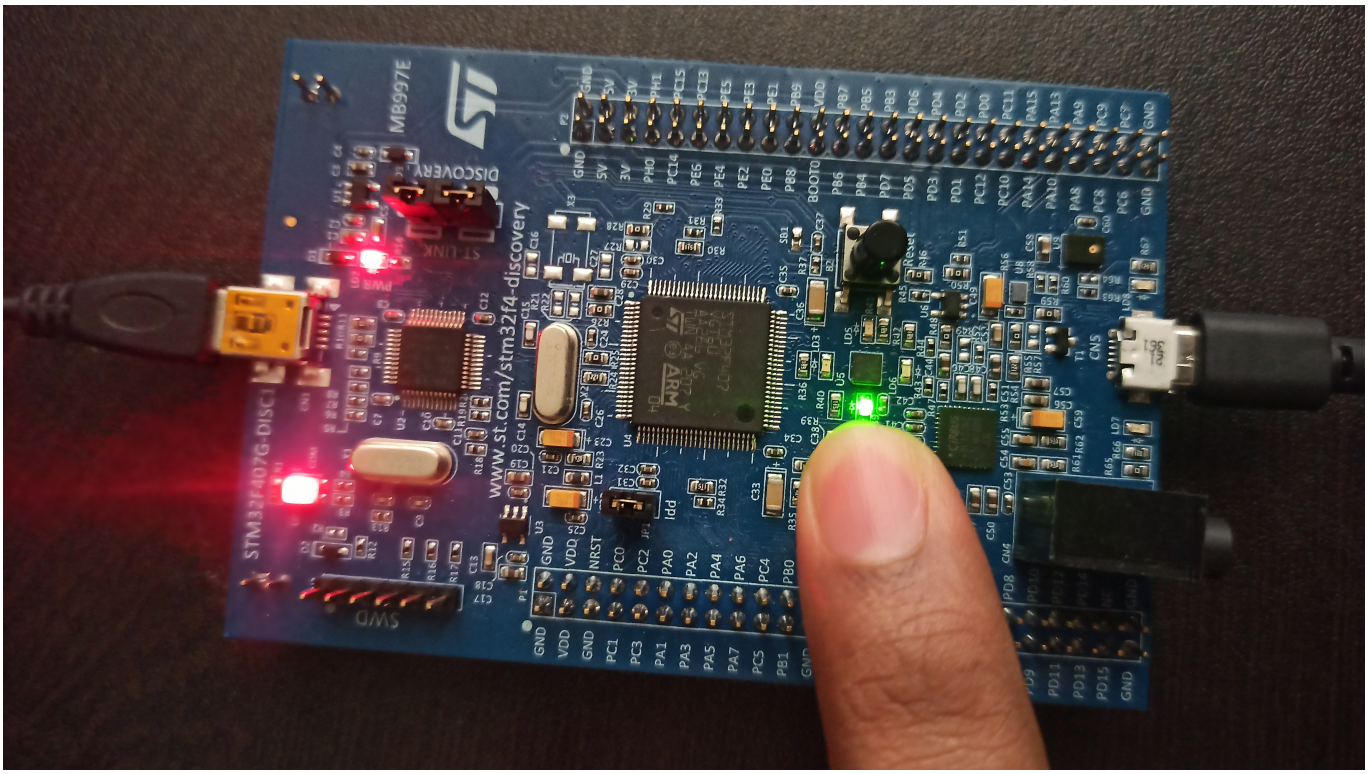
6. call the Read_GPIOA_PIN_0_Write_GPIOD_PIN_12() function as below in while(1) loop.

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
```

```
    /* USER CODE BEGIN 3 */
    Read_GPIOA_PIN_0_Write_GPIOD_PIN_12();
    /* USER CODE END 3 */
  }
  /* USER CODE END WHILE */
```

Save main.c file. Build, Flash and Run the program on the target.

We should see Green LED is OFF when you push-button is not pressed and LED is ON when you push the the push-button.



Source code is available at :
https://github.com/embeddedforallefa/embedded_sys_dev_with_stm32_code/tree/main/projects/EfA_STM32_dev/Core