Project 1.1: Controlling LED on the STM32F407G-DISC1 Board

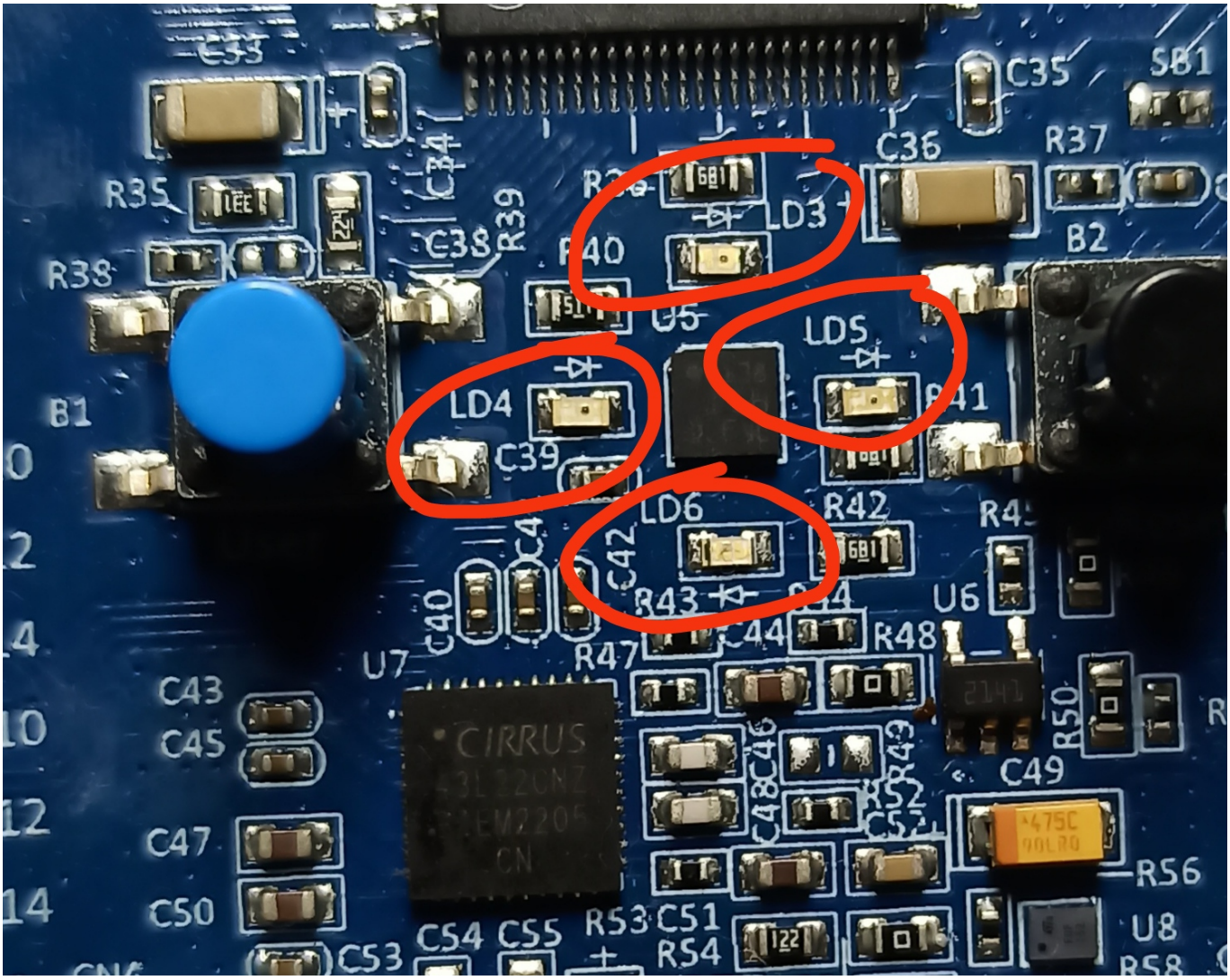
In this sub-project we are going to control one of the LEDs present on STM32F407G-DISC1 Board.

Upon completion of this project, you will learn.

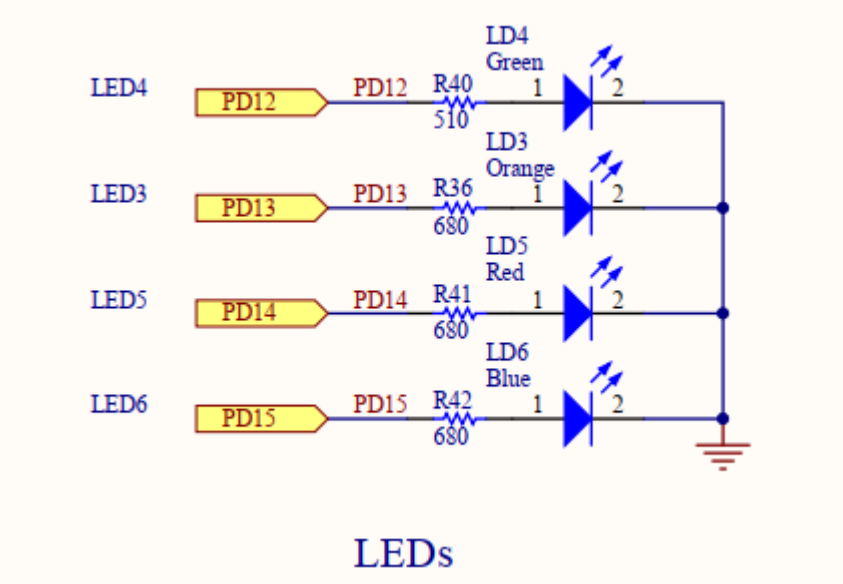
- To identify the Correct GPIO pin connect to the LED
- How LED is connected to Microcontroller.
- To refer datasheet and user manual when required.
- What is Hardware Abstraction.
- To work with GPIO registers.
- To work with C programming.

From STM32F407G-DISC1 user manual(section 6.3) we know that there are four user LEDs(LD3 to LD7).

1. User LD3: orange LED is a user LED connected to the I/O PD13 of the STM32F407VGT6.
2. User LD4: green LED is a user LED connected to the I/O PD12 of the STM32F407VGT6.
3. User LD5: red LED is a user LED connected to the I/O PD14 of the STM32F407VGT6.
4. User LD6: blue LED is a user LED connected to the I/O PD15 of the STM32F407VGT6.



Below is the schematic extract for the LED circuit:



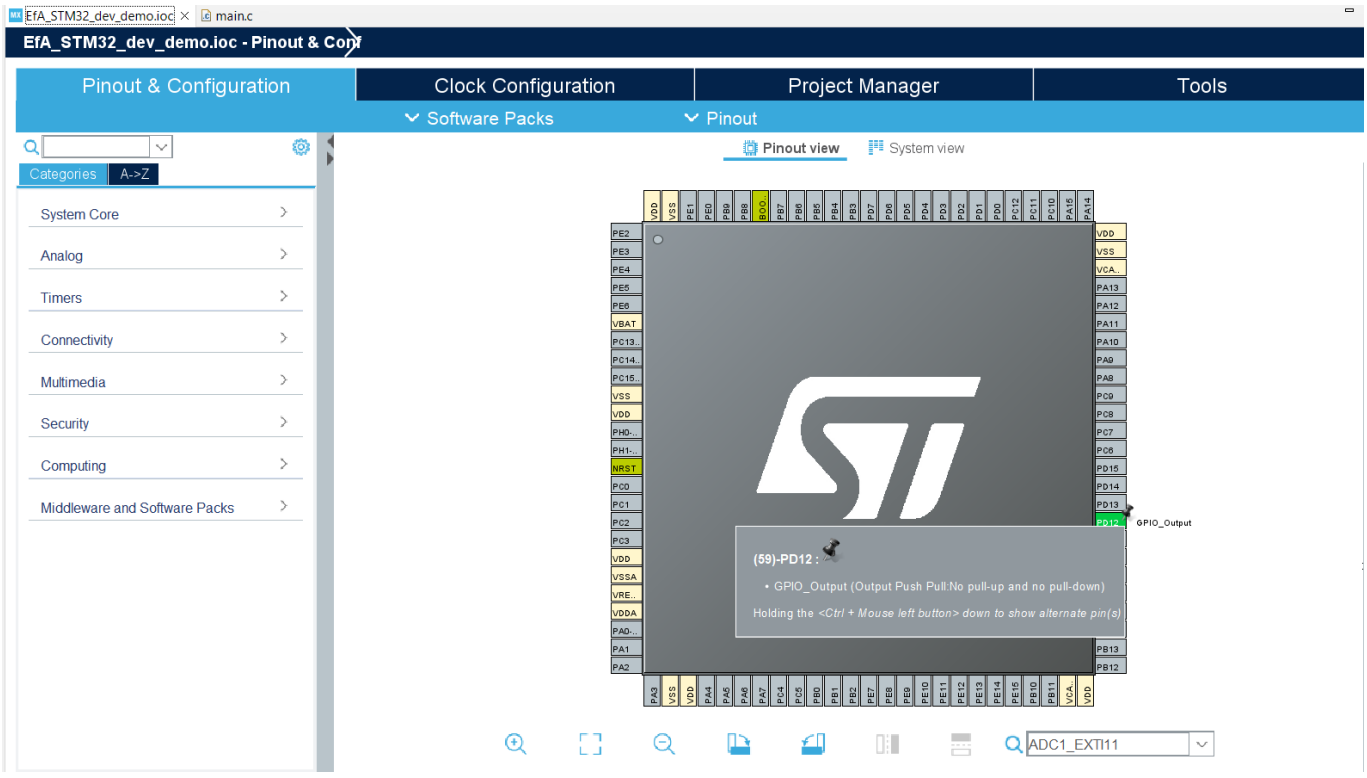
Notice the resistor and LED symbols.

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

In this project we will control User LD4: green LED is a user LED connected to the I/O PD12 of the STM32F407VGT6.

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 PD12 in the chip diagram and select GPIO_Output.



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 LED control 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 "led_control.h" in /Core/Inc folder with following content.

```
#ifndef INC_LED_CONTROL_H_
#define INC_LED_CONTROL_H_

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

#define DELAY_IN_MS 1000

/*
 * Function to toggle GPIO pin 12.
 * Input:
 *      Delay - Delay in milliseconds
 */
void Led_Toggle_GPIO_PIN_12(uint32_t Delay);

#endif /* INC_LED_CONTROL_H_ */
```

In this file we:

1. Define Header Guards.

   ```
   #ifndef INC_LED_CONTROL_H_
   #define INC_LED_CONTROL_H_


   /* Header contents here */

   #endif /* INC_LED_CONTROL_H_ */
   ```

   To prevent multiple inclusions of the same header file, especially when headers are included in multiple source files, you should use header guards. Header guards are conditional preprocessor directives that ensure a header file is included only once in a translation unit.

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. Define a constant DELAY_IN_MS with 1000 decimal value. #define is use to define a constant. In C programming, we use the '#define' directive to define macros. The syntax is

```
#define MACRO_NAME value
* **MACRO_NAME** is the name of the macro, and it is conventionally written in uppercase.
* **value** is the code snippet or constant that you want the macro to represent.
```

4. Declare a function with prototype:

```
void Led_Toggle_GPIO_PIN_12(uint32_t Delay);
```

Name of the function is *Led_Toggle_GPIO_sPIN_12*. It takes one parameter called *Delay* in milliseconds which is of type *uint32_t*(more about types in C later) and return type is *void* means returns nothing.

Here's the syntax for declaring a function in C:

```
return_type function_name(parameters);
```

- return_type: Specifies the data type of the value that the function will return. For functions that don't return a value, you use the void keyword.
- function_name: The name of the function, which should be unique within your program.
- parameters: A list of input parameters or arguments that the function accepts, along with their data types. If the function takes no parameters, you should use (void) or an empty parameter list.

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

```
#include "led_control.h"

void Led_Toggle_GPIO_PIN_12(uint32_t Delay)
{
    /* Use the HAL library function to introduce delay */
    HAL_Delay(Delay);
    /* Use the HAL library function to toggle GPIO */
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
}
```

In this file we:

1. Include the "led_control.h".
2. Define the Led_Toggle_GPIO_PIN_12() function which we had declared in the "led_control.h"
    - Inside the function call HAL_Delay(Delay). It takes one parameter of type *uint32_t* and introduces a delay of so many microseconds as the input argument. We can see definition of this function in

stm32f4xx_hal.c file.

- Then call HAL_GPIO_TogglePin() with parameters
    1. GPIOD is a macro which expands to base address of GPIOD (((GPIO_TypeDef *) ((0x40000000UL + 0x00020000UL) + 0x0C00UL))) We can check the same from reference manual. Table 1. STM32F4xx register boundary addresses 0x4002 0C00 - 0x4002 0FFF for GPIOD
    2. GPIO_PIN_12(0x1000) which is 12th pin.

Now lets update the main.c file:

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

    ```
    #include "led_control.h"
    ```

    By including the "led_control.h" header, the compiler knows about the Led_Toggle_GPIO_PIN_12 function's prototype (declaration), allowing you to call it in your main.c file even though the actual implementation of Led_Toggle_GPIO_PIN_12 is in a different source file i.e *led_control.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.

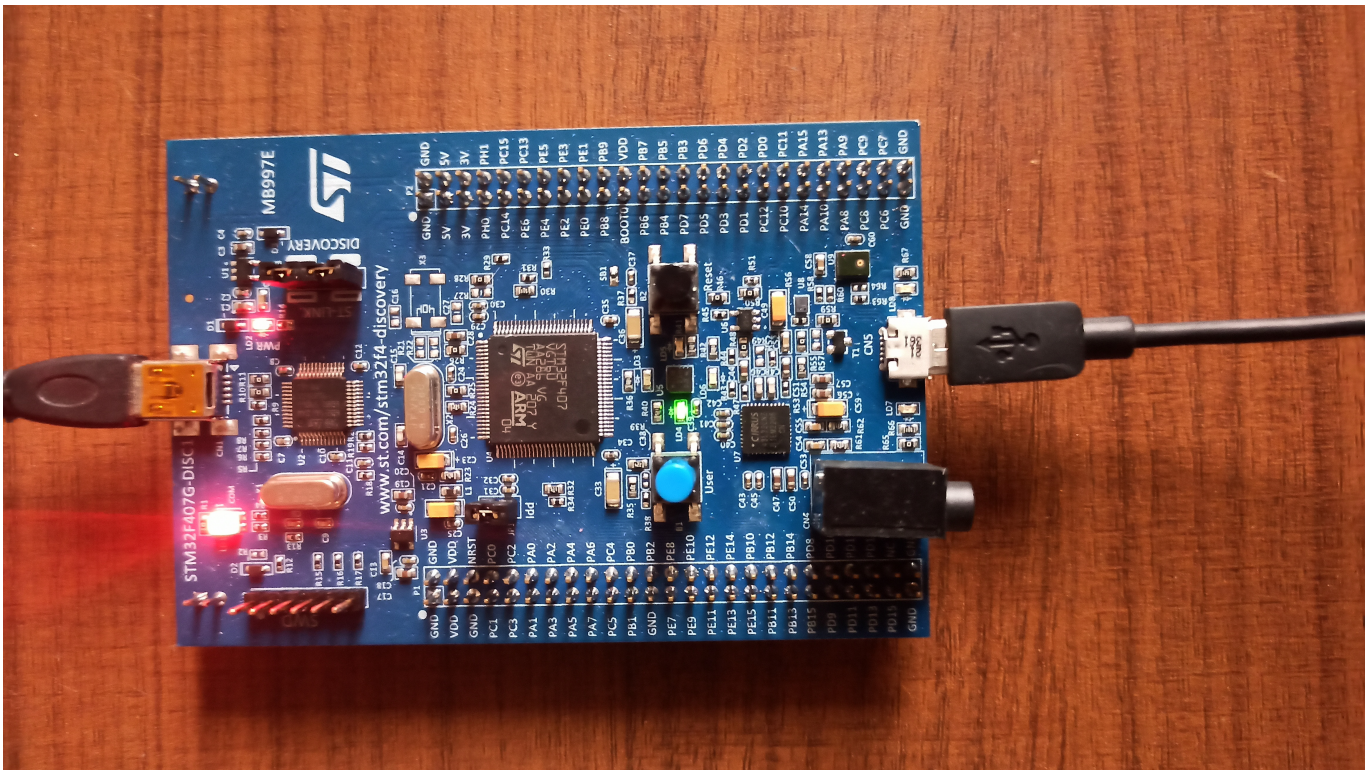
5. Then **MX_GPIO_Init()** which initialize all configured peripherals.

6. call the Led_Toggle_GPIO_PIN_12() function as below in while(1) loop.

    ```
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        /* USER CODE BEGIN 3 */
        Led_Toggle_GPIO_PIN_12(DELAY_IN_MS);
        /* 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 blinking every 1 sec (1 sec ON and 1 sec OFF).



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