Understanding the STM32F0′s GPIO part 2 (./stm32f0-gpio-tutorial-part-2.html)

This entry is a continuation of the GPIO Tutorial found here (./stm32f0-gpio-tutorial-part-1.html). We will spend a bit more time discussing the project structure for register based projects that do not require the STM32F0 Peripheral Library.

We will then use our knowledge of GPIO control to blink an RGB LED. Each time the RGB LED blinks, it lights up a randomly selected color. The RGB LED can be used to recreate 7 main colors; red, green, blue, yellow,cyan, magenta and white.

Finally we will demonstrate how to read the state of an input pin and modify the RGB LED code to blink in a specific color sequence when the "USER" button on the STM32F0Discovery board is pressed.

Non-Peripheral Library based Project Folder Structure

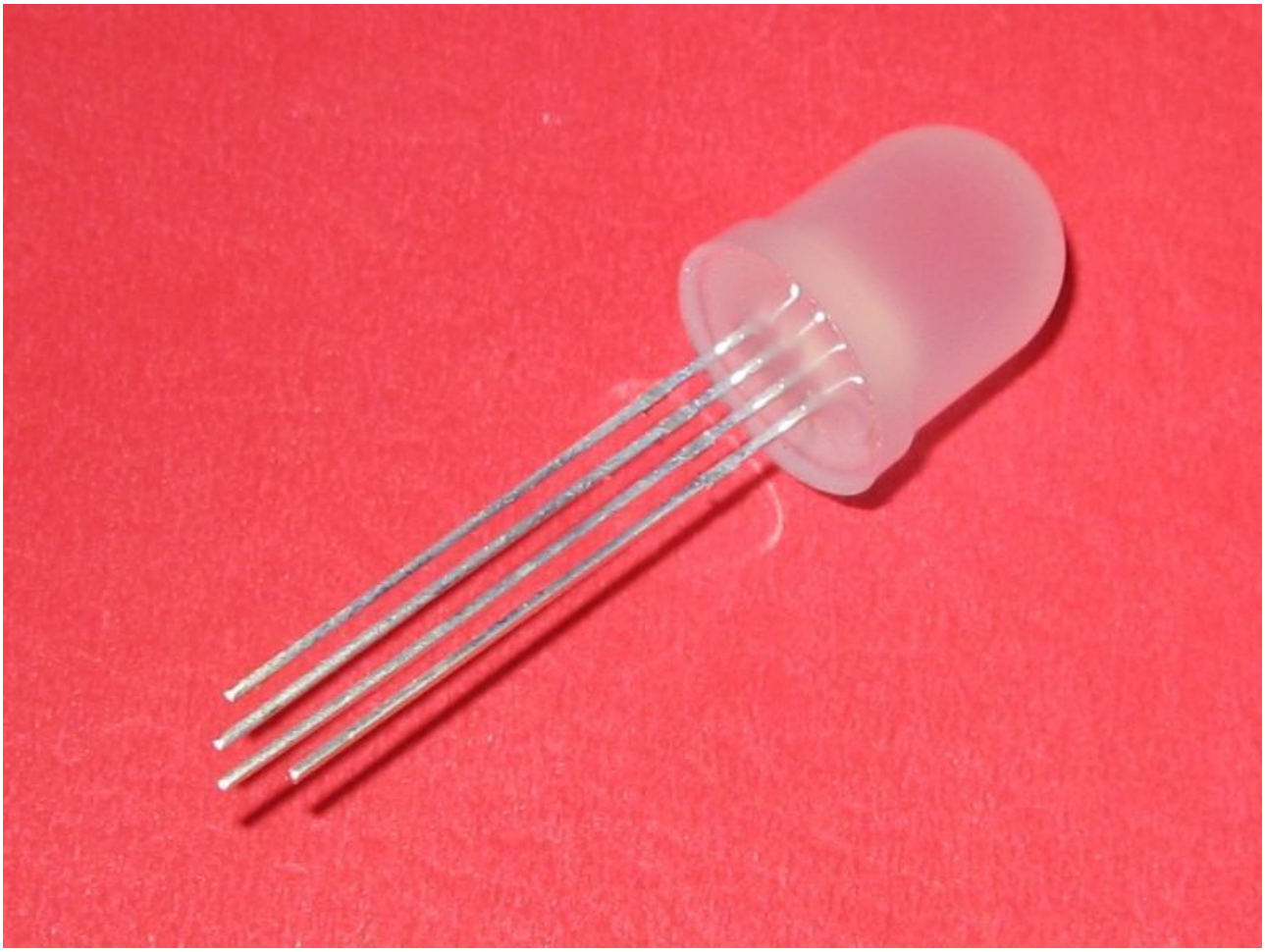
The project structure for the gpiotogglep1 project (files/OTHERFILES/gpiotogglep1.zip) discussed in part 1 (./stm32f0-gpio-tutorial-part-1.html) is shown below. This structure will be used with ALL FUTURE PROJECTS. It is very similar to the iotogglem0 project structure but with some additions in the include ("inc") directory. In the "inc" folder, the following two header files were added; "stm32f0xx.h" "system_stm32f0xx.h".

```
-> gpiotogglep1 folder
    -> Makefile
    -> startup folder
        ->startup_stm32f0xx.s
    -> linker folder
        ->stm32f0_linker.ld
    -> src folder
        -> main.c
        -> stm32f0xx_it.c
        -> system_stm32f0xx.c
    -> inc folder
        ->stm32f0xx.h
        ->stm32f0xx_it.h
        ->stm32f0xx_conf.h
        ->system_stm32f0xx.h
        ->CMSIS folder
```

Also the contents of the "CMSIS "folder found in the "inc" folder were taken from the following section of the STM32F0Discovery firmware package: "STM32F0-Discovery_FW_V1.0.0LibrariesCMSISInclude" was added and renamed "CMSIS". This "CMSIS" folder contains all of the CMSIS library and is required for the proper operation of the "stm32f0xx.h" header file.

Changes where also made to the makefile to reflect the new project structure. You will notice that it has no dependencies outside of the project folder.

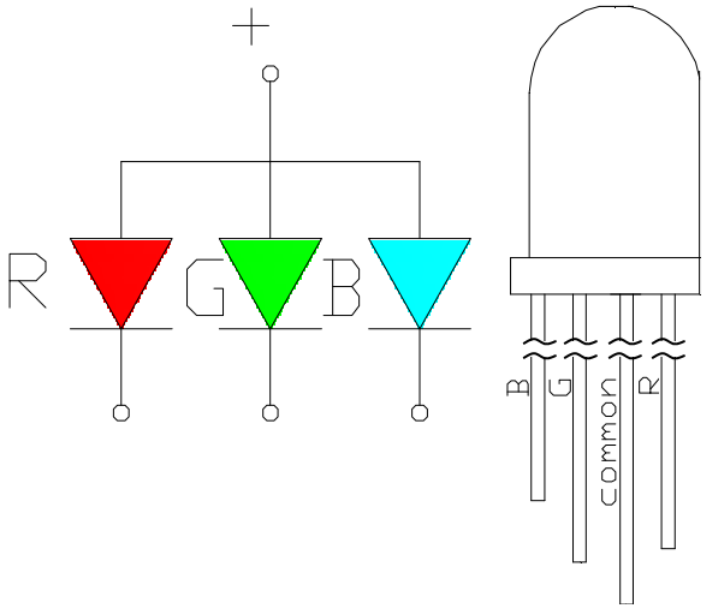
RGB Example



(files/uploads/2012/09/LED10AD1.jpg)

Figure 1. Diffused 10mm RGB LED Common Anode

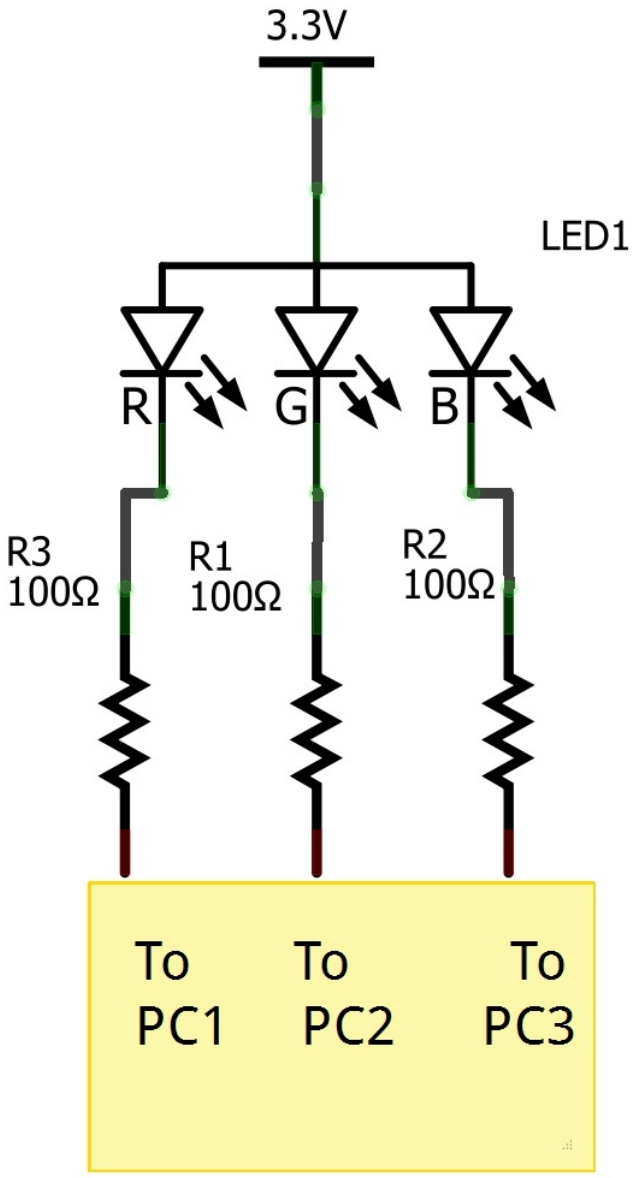
For this example we will be using a 10mm diffused RGB LED bought from dipmicro.com (http://dipmicro.com/store/LED10AD) for 51 cents! Cheapest jumbo size RGB LED that I could find so far! So if you need lots of RGB LEDs..checkout dipmicro.com.



(files/uploads/2012/09/LED-RGB1.png)

Figure 2. RGB LED Pinout

This particular RGB LED is setup in a common anode configuration which means that the anode pin is typically set to VDD, whereas the R, G and B cathode pins will be connected to the GPIO pins of the microcontroller via 100 ohm series resistors. We will connect PC1 to the red cathode pin, PC2 to the green cathode pin and PC3 to the blue cathode pin as shown in figure 3.



(files/uploads/2012/09/Untitled-Sketch_schem1.jpg)

Figure 3. RGB Connection to STM32F0

As I had mentioned briefly before, we want to blink an RGB LED with randomly selected colors. The code is provided below . Note that I have not separated the code into a main.c source files and a library.c/.h files because the code is still rather simple and not too long. I've included the project folder for this particular example here (gpiotogglep2) (files/OTHERFILES/gpiotogglep2.zip).

```
/******************************************************************************
  * @file    gpiotogglep2/main.c
  * @author  Hussam Al-Hertani
  * @version V1.0.0
  * @date    22-July-2012
  * @brief   Main program body
  ******************************************************************************/

/* Includes ------------------------------------------------------------------*/
#include "stm32f0xx.h"

unsigned long int next = 1;
void srand(unsigned int seed);
int rand(void);
void delay (int a);
/**
  * @brief  Main program.
  * @param  None
  * @retval None
  */
int main(void)
{
    int rNum = 0;
  /*!< At this stage the microcontroller clock setting is already configured,
       this is done through SystemInit() function which is called from startup
       file (startup_stm32f0xx.s) before to branch to application main.
       To reconfigure the default setting of SystemInit() function, refer to
       system_stm32f0xx.c file
     */

  /* GPIOC Periph clock enable */
    RCC->AHBENR |= RCC_AHBENR_GPIOCEN;

    GPIOC->MODER |= (GPIO_MODER_MODER1_0 | GPIO_MODER_MODER2_0|GPIO_MODER_MODER3_0) ;
    /* Configure PC8 and PC9 in output  mode  */

    GPIOC->OTYPER &= ~(GPIO_OTYPER_OT_1 | GPIO_OTYPER_OT_2 | GPIO_OTYPER_OT_3) ;
    // Ensure push pull mode selected--default

    GPIOC->OSPEEDR |= (GPIO_OSPEEDER_OSPEEDR1|GPIO_OSPEEDER_OSPEEDR2|GPIO_OSPEEDER_OSPEED
R3);
    //Ensure maximum speed setting (even though it is unnecessary)

    GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPDR1|GPIO_PUPDR_PUPDR2|GPIO_PUPDR_PUPDR3);
    //Ensure all pull up pull down resistors are disabled

    srand(26745);

    while (1)
    {

        rNum = (rand() % 7 + 1) & 7;  //create random  between 1 and 7
        switch(rNum)
```

```
        {
            case 1:
                GPIOC->ODR = ( (1<<3) |(1<<2) ); // RED
                break;

            case 2:
                GPIOC->ODR = ( (1<<3) |(1<<1) ); //GREEN
                break;

            case 3:
                GPIOC->ODR = ( (1<<2) |(1<<1) ); //BLUE
                break;

            case 4:
                GPIOC->ODR = ( (1<<3) ); //YELLOW
                break;

            case 5:
                GPIOC->ODR = ( (1<<2)); //MAGENTA
                break;

            case 6:
                GPIOC->ODR = ( (1<<1) ); //CYAN
                break;

            case 7:
                GPIOC->ODR = 0; //WHITE
                break;

            default:
                GPIOC->ODR = ( (1<<3) |(1<<2) | (1<<1) );
                break;
        }

        delay(1000000);

    }

    return 0;
}

void delay (int a)
{
    volatile int i,j;

    for (i=0 ; i < a ; i++)
    {
        j++;
    }

    return;
}

/* K&R rand function : return pseudo-random integer on 0..32767 */
```

```
int rand(void)
{
next = next * 1103515245 + 12345;
return (unsigned int)(next/65536) % 32768;
}

/* srand: set seed for rand() */
void srand(unsigned int seed)
{
next = seed;
}
```

By this point in time you should be familiar with the gpio initialization code so I will explain the other aspects of the program. In order to randomly select one of the seven possible colors that the RGB LED can produce, I had to rely on a random number generation function. Initially I tried to use the srand/rand function included in Newlib C standard library that's part of the gcc-arm compiler. To do this simply add "#include <stdlib.h> and then go ahead and use srand() to determine the seed and then rand() to create the number. While this worked, the compiled code ended up using 1612 bytes of Flash program memory and a whopping 1092 bytes of static data RAM memory. I'm not quite sure why this is the case but it seems that Newlib is very bloated. So I decided to utilize the srand/rand functions from the Kernighan and Ritchie's "The C programming Language book". With these functions the same program compiled into 1520 bytes of Flash program memory (almost 100 bytes saving) and only 24 bytes of static data RAM memory (more than 1Kbyte saving!!!) usage which is a lot better!

The program starts off by enabling the clock to the GPIOC peripheral and then configuring pins PC1, PC2 and PC3 to output in a similar manner to what we've seen in the first part of the tutorial. The srand() function is then called to initialize the random number generation function with the chosen seed.

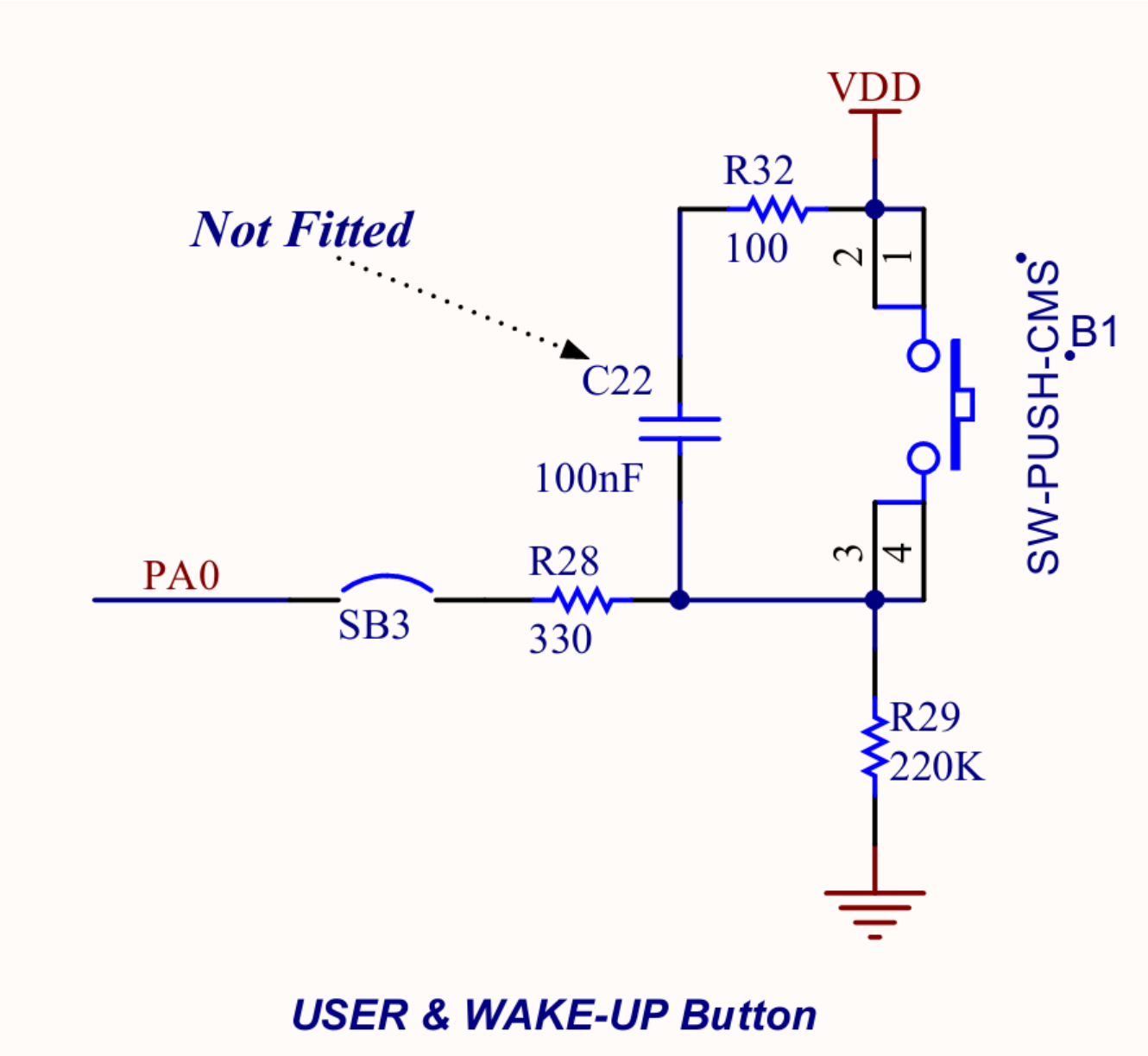
Inside the endless while loop we then generate a random number between 1 to 7 and using a switch case statement, based on the value of this number we turn on/off the appropriate pins to generate a particular color.

Because the RGB LED is common anode, the logic is inverted i.e if I turn on PC1 connected to the RED Cathode only, then the RED LED is OFF and the GREEN and BLUE LEDs are ON producing a CYAN light. Also note that I'm writing directly to the ODR register.

After the switch case statement is executed, a simple cycle delay function is called to simulate a delay that can be visualized.

Thats it!

Reading State of Input Pins



(files/uploads/2012/09/art5image4.png)

Figure 4. Schematic of the USER pushbutton on the STM32F0Discovery board

Now we would like to modify the previous RGB LED example such that the color of the RGB LED changes after the "USER" push button on the STM32F0Discovery has been pressed. A quick look at the STM32F0Discovery manual reveals that the "USER" push button is connected to pin PA0 on GPIOA. It is also connected via a pull down resistor as shown in Figure 4. Meaning that if the button is not pressed, pin PA0 will be at '0' volts and bit '0' of the GPIOA IDR register will be at '0'. If however the button is pressed, pin PA0 will be at VDD and bit '0' of the GPIOA IDR register will be at '1'. To read the state of bit '0' of GPIOA's IDR register we can use the following line of C code:

```
if(GPIOA->IDR & GPIO_IDR_0)
{
   .....
```

Here we basically perform a bitwise AND of the contents of the IDR register with a mask (GPIO_IDR_0... "see stm32f0xx.h") "0b0000000000000001" (the location of bit '0' is set to 1). If the result is non-zero, then bit '0' of GPIOA's IDR register is '1' indicating that the push button is indeed pressed. If however the result is zero, then bit '0' of GPIOA's IDR register is '0' indicating that the push button is not pressed.

To build upon this further we create a function "ButtonPressed()" :

```
int buttonPressed()
{
    int returnVal = 0;
    if(GPIOA->IDR & GPIO_IDR_0)
    {
        delay(50000);
        if(GPIOA->IDR & GPIO_IDR_0)
        {
            returnVal = 1;
            while(GPIOA->IDR & GPIO_IDR_0){}
        }

    }

    return returnVal;
}
```

This function first checks if bit '0' of GPIOA's IDR register is '1' or not. If it is indeed '1' then it waits a few milliseconds and checks again to ensure that this change was not due to a switch debounce fluctuation. If bit '0' of GPIOA's IDR register is still '1' then this is definitely due to a valid button press and not switch debounce, at this point set the function's return value to '1' to indicate that a valid pushbutton press has occured. The while loop ensures that code execution exits the function only once the push button has been released. This ensures that each single button press causes a single RGB LED color transition.

Now in this example I opted to have the RGB LED switch colors in a fixed order and not randomly. This requires the use of a finite state machine.Whenever a finite state machine is being coded in C, the states ought to be defined as an enumeration data type. In our case, we will create an "enum" data type called "color" and instantiate a "state" variable of type "color" as shown below:

```
enum color {RED,GREEN,YELLOW,BLUE,MAGENTA,CYAN,WHITE};
enum color state = RED;
```

The state variable is initialized to RED.

We then need to code the state machine itself. We do this in the runRGBStateMachine() function. This function takes in an enumerated datatype denoting the current state and returns another enumerated data type denoting the next state:

```
enum color runRGBStateMachine(enum color a)
{
    switch(a)
    {
            case RED:
                GPIOC->ODR = ( (1<<2) | (1<<3) );
                a = GREEN;
                break;

            case GREEN:
                GPIOC->ODR = ( (1<<1) | (1<<3));
                a = BLUE;
                break;

            case BLUE:
                GPIOC->ODR = ( (1<<1) |(1<<2) );
                a = YELLOW;
                break;

            case YELLOW :
                GPIOC->ODR = (1<<3);
                a = MAGENTA;
                break;

            case MAGENTA:
                GPIOC->ODR = (1<<2);
                a = CYAN;
                break;

            case CYAN:
                GPIOC->ODR = (1<<1);
                a = WHITE;
                break;

            case WHITE:
                GPIOC->ODR = 0;
                a = RED;
                break;

            default:
                GPIOC->ODR = ( (1<<3) |(1<<2) | (1<<1) );
                a = WHITE;
                break;
    }
    return a;
}
```

The state machine function shown above takes the "current state of the RGB LED" (or the current color of the RGB LED) as a parameter, and sets the LED to light up that particular color. It then returns the future state of the RGB LED.

The RGB LED follows the following sequence RED-> GREEN-> BLUE-> YELLOW-> MAGENTA-> CYAN-> WHITE-> RED-> GREEN->....

Now we need to make sure that this state machine iterates only when the push button is pressed:

```
while (1)
    {
        if(buttonPressed())
        {
          state = runRGBStateMachine(state);
        }
    }
    return 0;
```

The complete code is provided below and the complete project folder structure is provided here (gpiotogglep3) (files/OTHERFILES/gpiotogglep3.zip):

```
/******************************************************************************
  * @file    gpiotogglep3/main.c
  * @author  Hussam Al-Hertani
  * @version V1.0.0
  * @date    25-July-2012
  * @brief   Main program body
  ******************************************************************************/

/* Includes ------------------------------------------------------------------*/
#include "stm32f0xx.h"

enum color {RED,GREEN,YELLOW,BLUE,MAGENTA,CYAN,WHITE};
enum color state = RED;

void delay (int a);
enum color runRGBStateMachine(enum color a);
int buttonPressed(void);
/**
  * @brief  Main program.
  * @param  None
  * @retval None
  */

int main(void)
{
  /*!< At this stage the microcontroller clock setting is already configured,
       this is done through SystemInit() function which is called from startup
       file (startup_stm32f0xx.s) before to branch to application main.
       To reconfigure the default setting of SystemInit() function, refer to
       system_stm32f0xx.c file
     */

  /* GPIOC GPIOA Periph clock enable */
    RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;

    GPIOC->MODER |= (GPIO_MODER_MODER1_0 | GPIO_MODER_MODER2_0|GPIO_MODER_MODER3_0) ;
    /* Configure PC1 PC2 and PC3 in output  mode and PA0 in input mode  */
    GPIOA->MODER &= ~(GPIO_MODER_MODER0) ;

    GPIOC->OTYPER &= ~(GPIO_OTYPER_OT_1 | GPIO_OTYPER_OT_2 | GPIO_OTYPER_OT_3) ;
    // Ensure push pull mode selected for output pins--default

    GPIOC->OSPEEDR |= (GPIO_OSPEEDER_OSPEEDR1|GPIO_OSPEEDER_OSPEEDR2|GPIO_OSPEEDER_OSPEED
R3);
    GPIOA->OSPEEDR |= (GPIO_OSPEEDER_OSPEEDR0);
    //Ensure maximum speed setting (even though it is unnecessary)

    GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPDR1|GPIO_PUPDR_PUPDR2|GPIO_PUPDR_PUPDR3);
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR0);
    //Ensure all pull up pull down resistors are disabled PA0 is connected to external pu
lldown on STM32F0Discovery board
```

```
    while (1)
    {
        if(buttonPressed())
        {
          state = runRGBStateMachine(state);
        }
    }
    return 0;
}

void delay (int a)
{
    volatile int i,j;

    for (i=0 ; i < a ; i++)
    {
        j++;
    }

    return;
}

int buttonPressed()
{
    int returnVal = 0;
    if(GPIOA->IDR & GPIO_IDR_0)
    {
        delay(50000);
        if(GPIOA->IDR & GPIO_IDR_0)
        {
            returnVal = 1;
            while(GPIOA->IDR & GPIO_IDR_0){}
        }

    }

    return returnVal;
}

enum color runRGBStateMachine(enum color a)
{
    switch(a)
    {
            case RED:
                GPIOC->ODR = ( (1<<2) | (1<<3) );
                a = GREEN;
                break;

            case GREEN:
                GPIOC->ODR = ( (1<<1) | (1<<3));
                a = BLUE;
                break;

            case BLUE:
```

```
                GPIOC->ODR = ( (1<<1) |(1<<2) );
                a = YELLOW;
                break;

            case YELLOW :
                GPIOC->ODR = (1<<3);
                a = MAGENTA;
                break;

            case MAGENTA:
                GPIOC->ODR = (1<<2);
                a = CYAN;
                break;

            case CYAN:
                GPIOC->ODR = (1<<1);
                a = WHITE;
                break;

            case WHITE:
                GPIOC->ODR = 0;
                a = RED;
                break;

            default:
                GPIOC->ODR = ( (1<<3) |(1<<2) | (1<<1) );
                a = WHITE;
                break;
    }
    return a;
}
```

Note that in the GPIO initialization portion of the code, we now have to enable the clock to both the GPIOA and GPIOC peripherals. We also have to configure PA0 as an input.

This concludes the GPIO tutorial. In my next blog entry, I will briefly discuss how to modify the above code to use external interrupts.

- « Understanding the STM32F0's GPIO part 1 (./stm32f0-gpio-tutorial-part-1.html)
- Part 3 - Setting up Debugging with the Eclipse IDE (OpenOCD 0.6.x) (./part-3-debugging-openocd-0-6-0.html) »

Published

Sep 1, 2012

Category

STM32F0 (/categories.html#STM32F0-ref)

Tags

- GPIO 7 (/tags.html#GPIO-ref)
- STM32F0 11 (/tags.html#STM32F0-ref)

- Powered by Pelican (http://getpelican.com/). Theme: Elegant (http://oncrashreboot.com/pelican-elegant) by Talha Mansoor (http://oncrashreboot.com)