ECE0202: Embedded Processors and Interfacing

Lab 2: Interfacing Push Button and LED (in C)

Due: TBD

Objectives

- **Get familiar with KEIL uvision software development environment**
- **Create a C project for the Nucleo kit and program the kit**
- **Learn the basics of GPIO input and output configuration**
- **Perform simple digital I/O input (interfacing push button) and output (interfacing LED)**
- **Understand polling I/O (busy waiting) and its inefficiency**

Pre-Lab Reading

- **Read textbook chapter 4.6 to review bitwise operations.**
- **Read textbook chapter 14 to learn about GPIO interfacing.**

Deliverables – total 100 points

- **(50 points) A demonstration of the working pushbutton and LED combination to the TA.**
- **(30 points) A submission of your code with comments.**
- **(20 points) A submission of answers to the pre-lab questions.**
- **(10 points extra credit) Configure the buttons and LEDs to accomplish the following:**
    - **Write a program that sends a Morse Code message "I Love ECE 202" using the LED. You will have to research how to create a delay in C.**
    - **Demonstrate this to a TA and submit your code as a separate file.**

Submit combine pre-lab answers and your code as a *.pdf file and submit it to Canvas.

Jingtong Hu
Based on a lab from ECE271 at the University of Maine by Yifeng Zhu.
2/6/2021

Getting Started

This project is written in the C language. Download the "C Project Template" from Canvas. If you load the project in KEIL, you will see a "main.c" file as part of the project. Open this file and write your project code in the the "USER CODE GOES HERE" area.

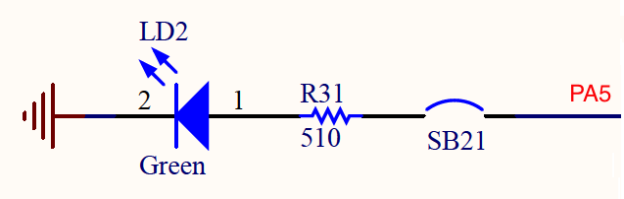
Lab Assignment

- Following Chapter 14, implement a C program that toggles the green LED when the user button is pressed.

LEDs on the Board

Part A: Green LED

There are one user-controllable green LED (light-emitting diodes) on the STM32 NUCLEO-L476RG board. The LED one is connected to the pin GPIO **PA5** (Port A Pin 5), as shown in the figure below.



To turn on a LED, software must at least perform the following five operations:

1. **Enable the clock** of the corresponding GPIO port. By default, the clock to all peripherals, including GPIO ports, are turned off to save the energy.
2. Set the ***mode*** of the corresponding GPIO pin must be set as ***output.*** By default, the mode of all GPIO pin is analog.
3. Set the push-pull/open-drain setting for the GPIO pins to **push-pull**
4. Set the pull-up/pull-down setting for the GPIO pins to **no pull-up and no pull-down**.
5. Finally, set the output of the GPIO pin to have a value of 1 (corresponding to 3.3V)

Compile/build your code

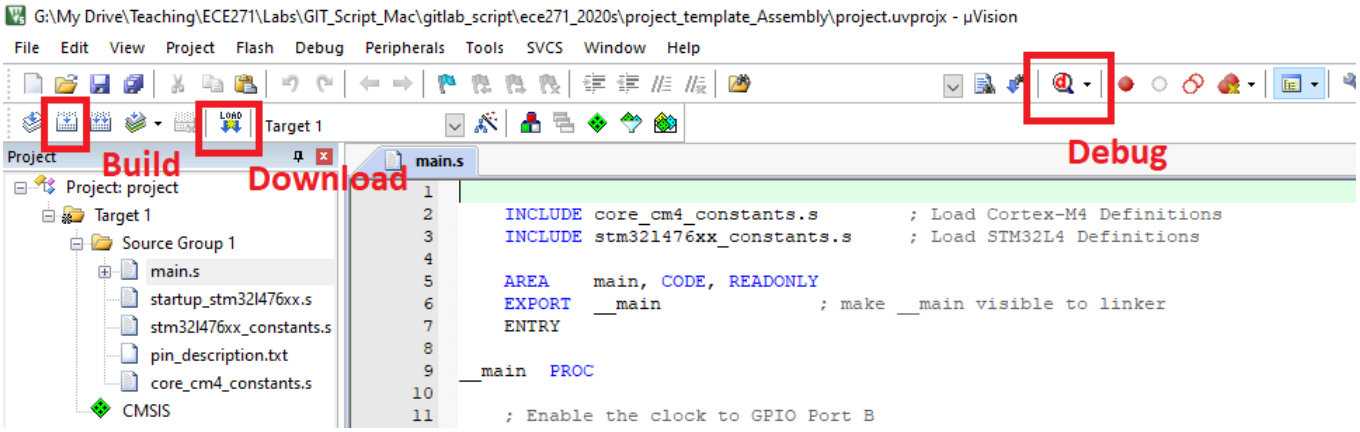
(1) Press the **build** button. Check the window at the bottom for warnings and especially for errors. The IDE will show little error icons by your code too if it detects any.

(2) Then, plug the board into your laptop with the USB cable if you haven't already.

(3) Now download your code to the board by pressing the **download** button.

(4) Push the reset button (black) on the board to run your program

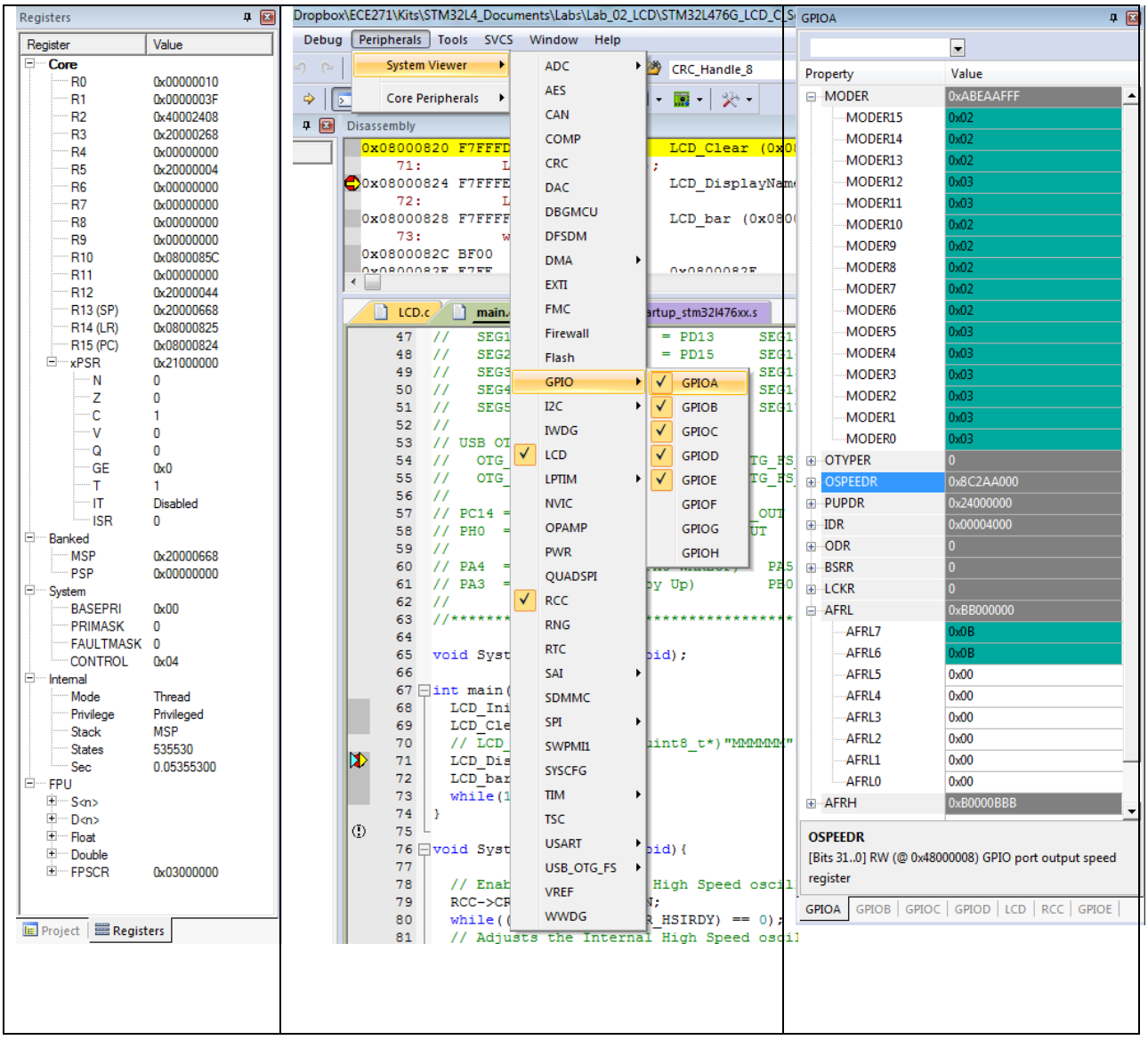
It is possible that your code is buggy so that Keil can't program the board anymore. In this case, you can use ST-Link utility to erase the flash memory. See this YouTube tutorial: https://www.youtube.com/watch?v=OiwwB0AvIBI



If the code doesn't work, you will have to debug your code to find out what is wrong.

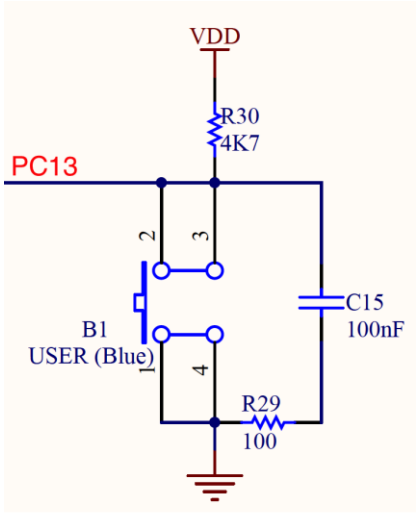
Review this debug tutorial: https://youtu.be/w4gPcYRk9o8

Keil allows us to view the values of Cortex-M registers and all peripheral registers in real-time. Click the following: Peripherals, System Viewers, GPIO, and GPIOC



Part B: Push Button

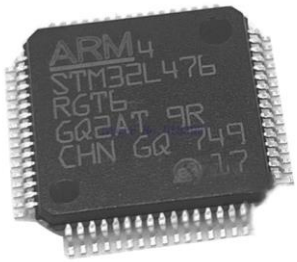
There is a blue user button the board. The button is connected to the microcontroller's **GPIO Port C Pin 13 (PC 13)**, as shown on the right.



- The pin is pulled up to VDD via a resistor (R30).
- When the button is not pressed, the voltage on PC 13 is VDD.
- When the button is pressed down, the voltage on PC 13 is 0.
- The circuit performs ***hardware debouncing*** using a resistor (R29) and a capacitor (C15). It forms a simple RC filter (resistive capacitive filter), which debounces the pushbutton switch.

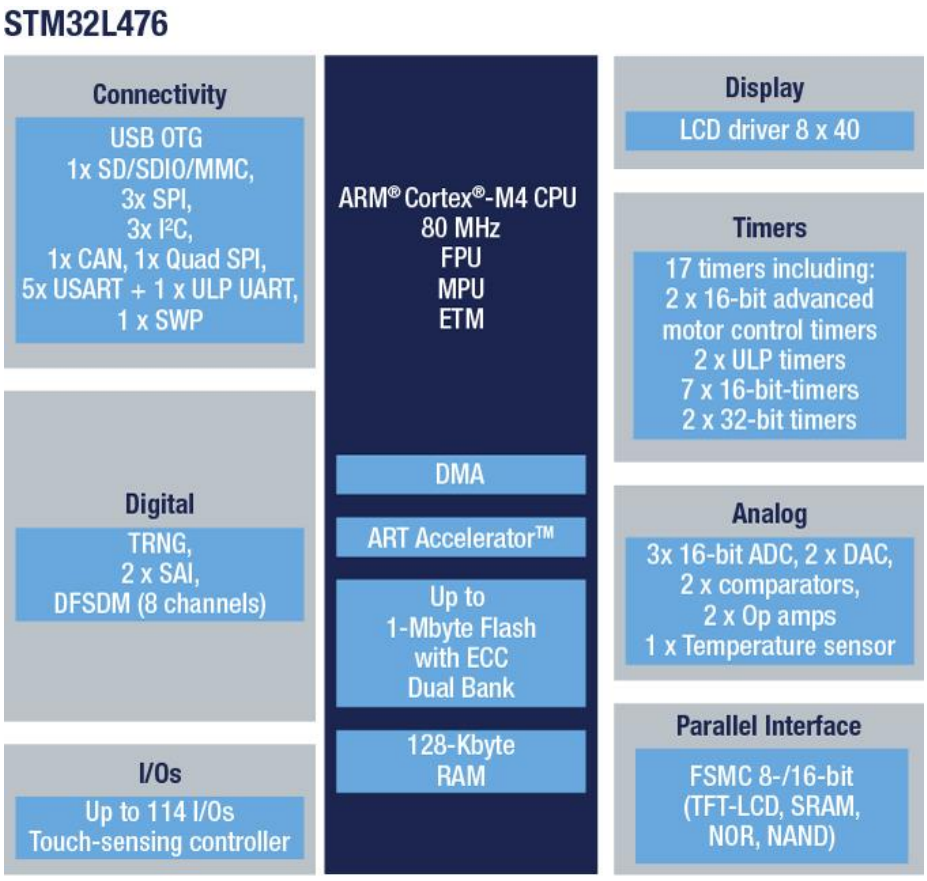
Appendix A: Microcontroller STM32L476

The NUCLEO-L476RG board has a 32-bit Arm Cortex-M4F microcontroller.



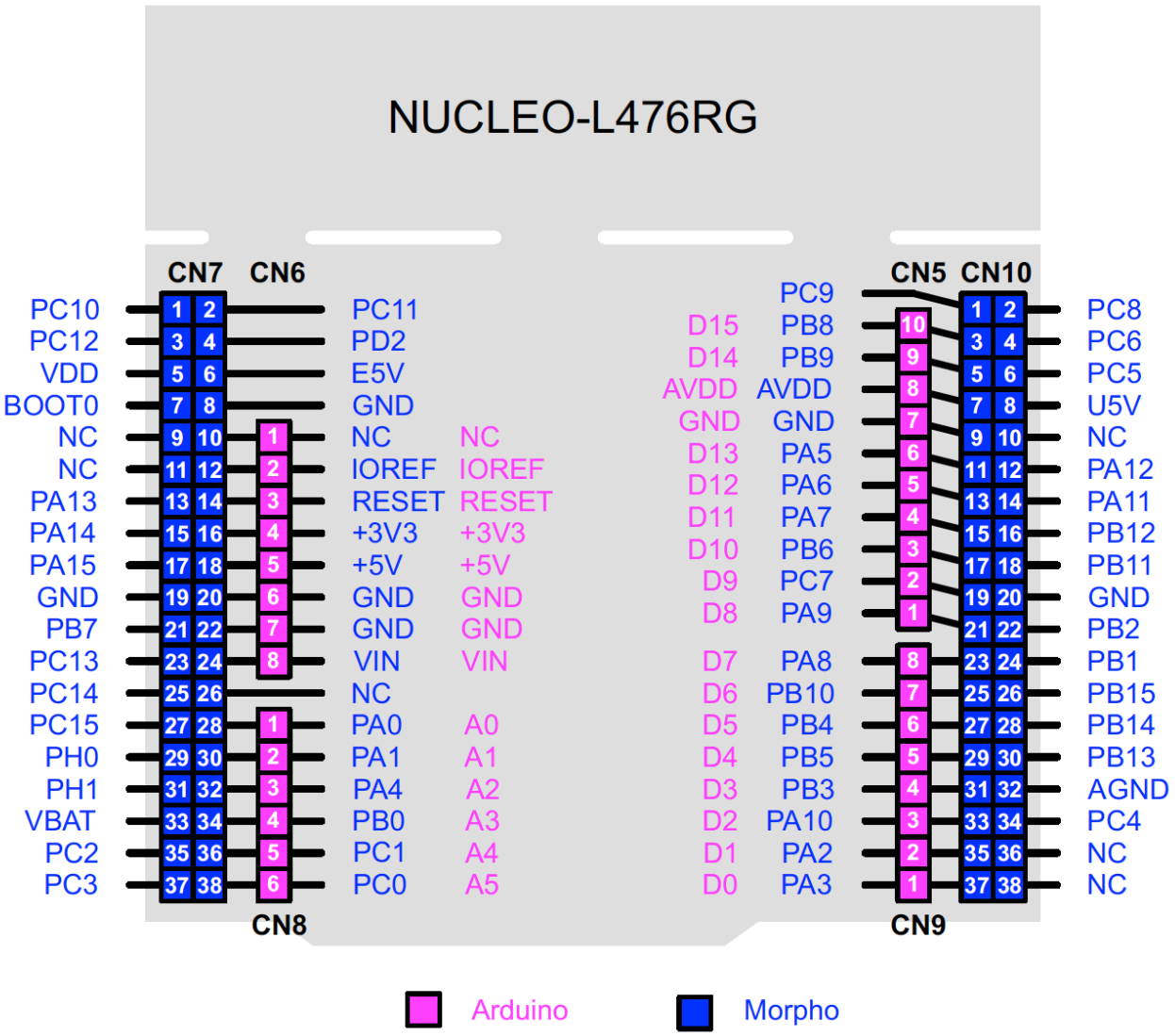
- Part Number: **STM32L476RGT6**
- Total number of pins: 64
- Total number of I/O pins: 51
- Maximum clock frequency: 80MHz
- Program memory size: 1 MB
- Operating supply voltage: 1.71V to 3.6V
- I/O voltage: 3.3V
- Analog supply voltage: 3.3V

The following figure summarizes the I/O connection supported and internal peripherals.



Appendix B: Pin Connections on Nucleo-L476RG

Board Component	Microcontroller Pin	Comment
Green LED	PA 5	SB42 closed and SB29 open by default
Blue user button	PC 13	Pulled up externally
Black reset button	NRST	Connect to ground to reset
ST-Link UART TX	PA 2	STLK_TX
ST-Link UART RX	PA 3	STLK_RX
ST-Link SWO/TDO	PB 3	Trace output pin/Test Data Out pin
ST-Link SWDIO/TMS	PA 13	Data I/O pin/Test Mode State pin
ST-Link SWDCLK/TCK	PA 14	Clock pin/Test Clock pin


NUCLEO-L476RG
CN7
CN6
CN8
CN5
CN10
CN9
PC10
PC12
VDD
BOOT0
NC
NC
PA13
PA14
PA15
GND
PB7
PC13
PC14
PC15
PH0
PH1
VBAT
PC2
PC3
PC11
PD2
E5V
GND
NC
IOREF
RESET
+3V3
+5V
GND
GND
VIN
NC
PA0
PA1
PA4
PB0
PC1
PC0
A0
A1
A2
A3
A4
A5
PC9
D15
D14
AVDD
GND
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
PB8
PB9
AVDD
GND
PA5
PA6
PA7
PB6
PC7
PA9
PA8
PB10
PB4
PB5
PB3
PA10
PA2
PA3
PC8
PC6
PC5
U5V
NC
PA12
PA11
PB12
PB11
GND
PB2
PB1
PB15
PB14
PB13
AGND
PC4
NC
NC
Arduino
Morpho

Appendix C: Clock Configuration

There are two major types of clocks: **system clock** and **peripheral clock**. A video tutorial is given here: https://youtu.be/o6ZWD0PAoJk

- ***System Clock***: To meet the requirement of performance and energy-efficiency for different applications, the processor core can be driven by four different clock sources, including ***HSI*** (high-speed internal) oscillator clock, ***HSE*** (high-speed external) oscillator clock, **PLL** clock, and **MSI** (multispeed internal) oscillator clock. A faster clock provides better performance but usually consumes more power, which is not appropriate for battery-powered systems.
- ***Peripheral Clock***: All peripherals require to be clocked to function. However, ***clocks of all peripherals are turned off by default to reduce power consumption***.

The following figure shows the clock tree of ***STM32L476***, the processor used in the STM32L4 Discovery kit. The clock sources in the domain of Advanced High-performance Bus (***AHB***), low-speed Advanced Peripheral Bus 1 (***APB1***) and high-speed Advanced Peripheral Bus 2 (***APB2***) can be switched on or off independently when it is not used. Software can select various clock sources and scaling factors to achieve desired clock speed, depending on the application's needs.

The software provided in this lab uses the 16MHz HSI as the input to the PLL clock. Appropriate scaling factors have been selected to achieve the maximum allowed clock speed (80 MHz). See the function void **`System_Clock_Init`**() for details.

```
void System_Clock_Init(void) {

   ...


  // Enable the Internal High Speed oscillator (HSI)

  RCC->CR |= RCC_CR_HSION;

  while((RCC->CR & RCC_CR_HSIRDY) == 0);


  RCC->CR    &= ~RCC_CR_PLLON;

  while((RCC->CR & RCC_CR_PLLRDY) == RCC_CR_PLLRDY);


  // Select clock source to PLL

  RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLSRC;

  RCC->PLLCFGR |= RCC_PLLCFGR_PLLSRC_HSI; // 00 = No clock, 01 = MSI, 10 = HSI, 11 = HSE
```

```
// Make PLL as 80 MHz

// f(VCO clock) = f(PLL clock input) * (PLLN / PLLM) = 16MHz * 20/2 = 160 MHz

// f(PLL_R) = f(VCO clock) / PLLR = 160MHz/2 = 80MHz

RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLN) | 20U << 8;

RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLM) | 1U << 4;

RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLR;  // 00: PLLR = 2, 01: PLLR = 4, 10: PLLR = 6, 11: PLLR = 8

RCC->PLLCFGR |= RCC_PLLCFGR_PLLREN; // Enable Main PLL PLLCLK output

RCC->CR   |= RCC_CR_PLLON;

while((RCC->CR & RCC_CR_PLLRDY) == 0);


// Select PLL selected as system clock

RCC->CFGR &= ~RCC_CFGR_SW;

RCC->CFGR |= RCC_CFGR_SW_PLL; // 00: MSI, 01:HSI, 10: HSE, 11: PLL


// Wait until System Clock has been selected

while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL);
 ...
```

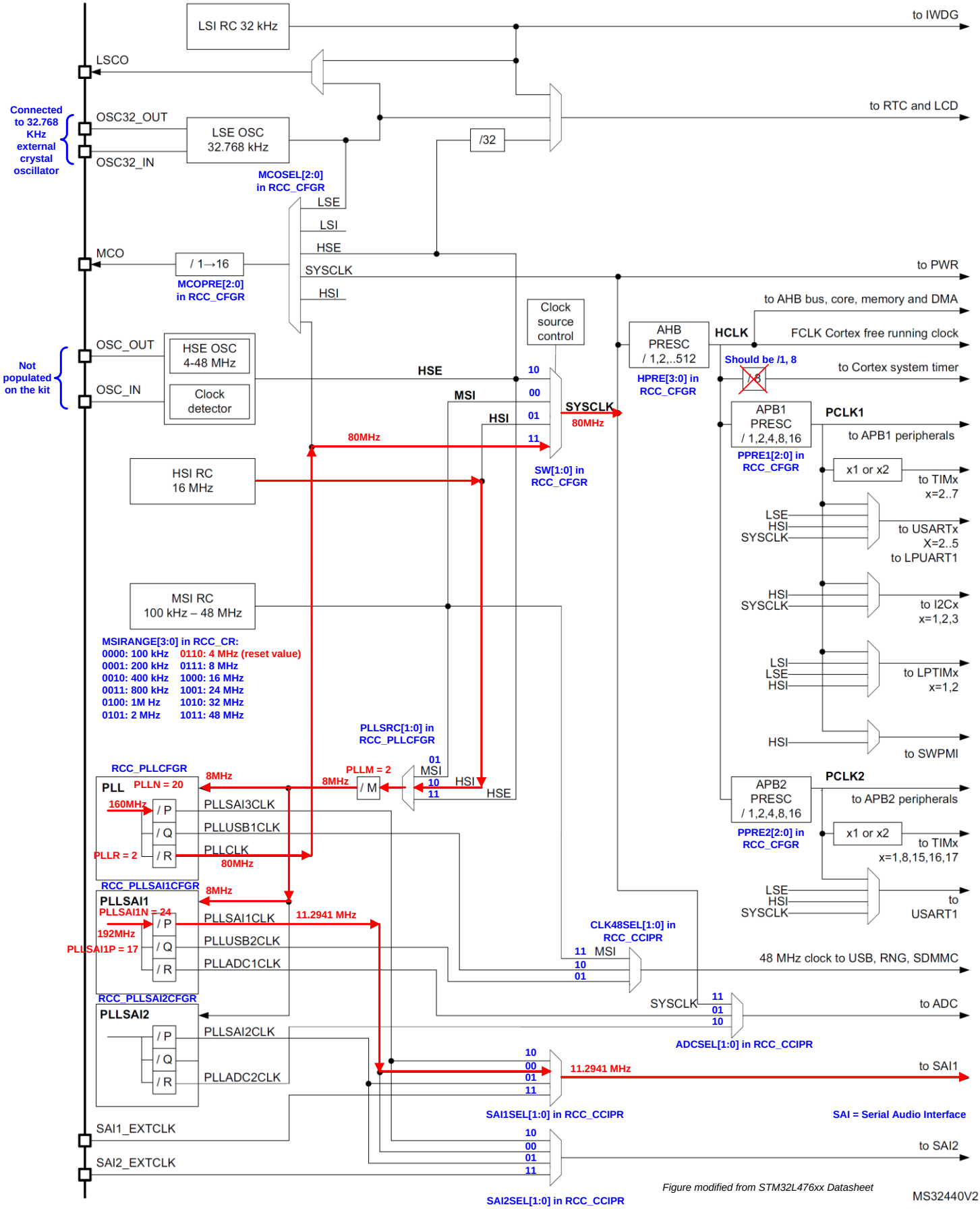
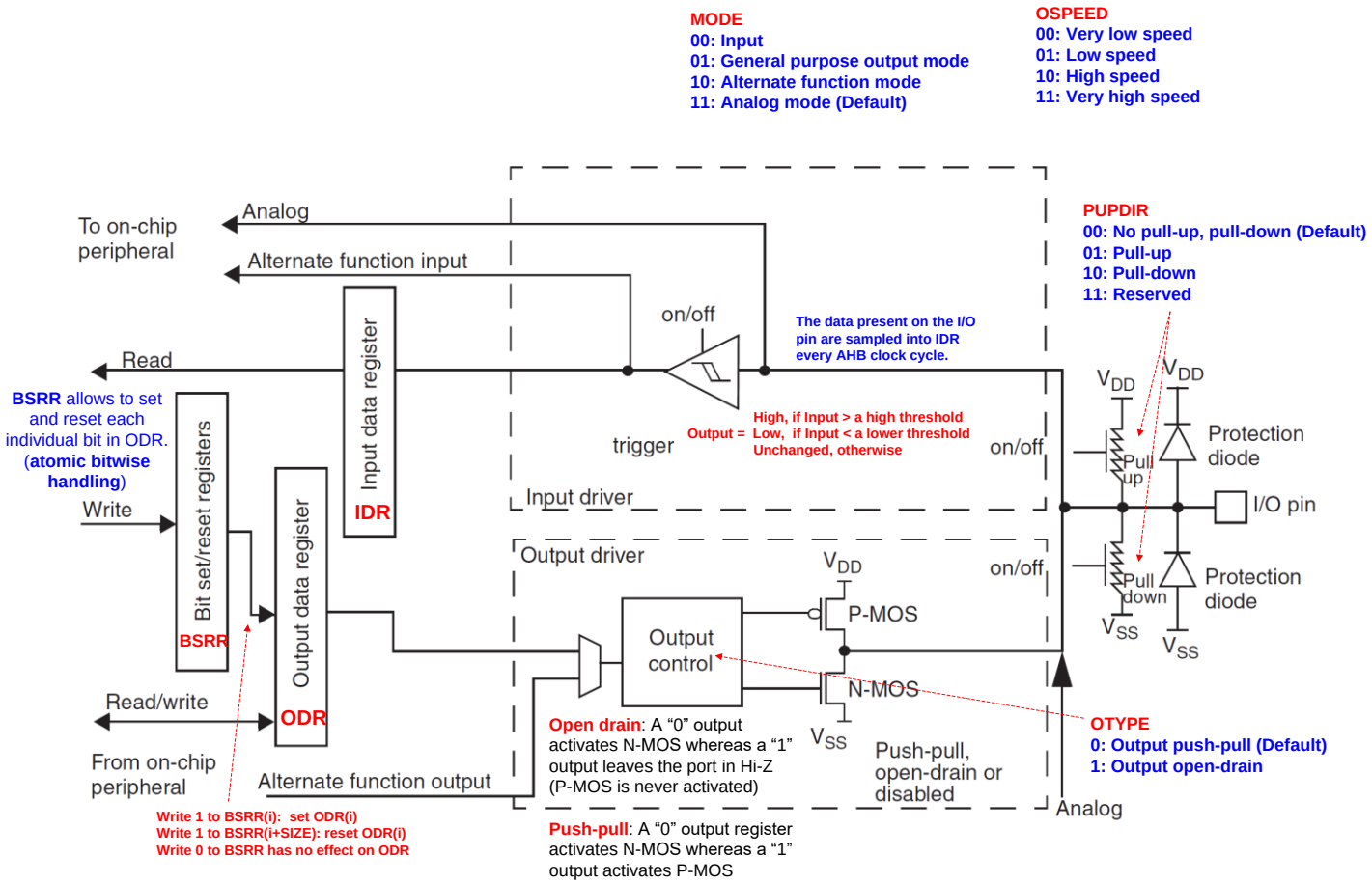


Figure modified from STM32L476xx Datasheet

MS32440V2

Each of the GPIO pins can be configured by software as output (push-pull or open-drain), as input (with or without pull-up or pull-down) or as peripheral alternate function. ***In this lab, we will configure PB2 and PE8 as push-pull output.*** Each general-purpose I/O port x (x = A, B, C, …, H) has

- four 32-bit configuration registers
  - **MODER** (mode register)
  - **OTYPER** (output type register)
  - **OSPEEDR** (output speed register)
  - **PUPDR** (pull-up/pull-down register)
- two 32-bit data registers
  - **IDR** (input data register)
  - **ODR** (output data register)
- a 32-bit set/reset register (**BSRR**).
- a 32-bit locking register (**LCKR**)
- two 32-bit alternate function selection registers
  - **AFRH** (Alternate function high register)
  - **AFRL** (Alternate function low register)



Appendix D: Code Comments and Documentation

Program comments are used to improve code readability, and to assist in debugging and maintenance. A general principal is "***Structure and document your program the way you wish other programmers would***" (McCann, 1997).

The book titled "*The Elements of Programming Style*" by Brian Kernighan and P. J. Plauger gives good advices for beginners.

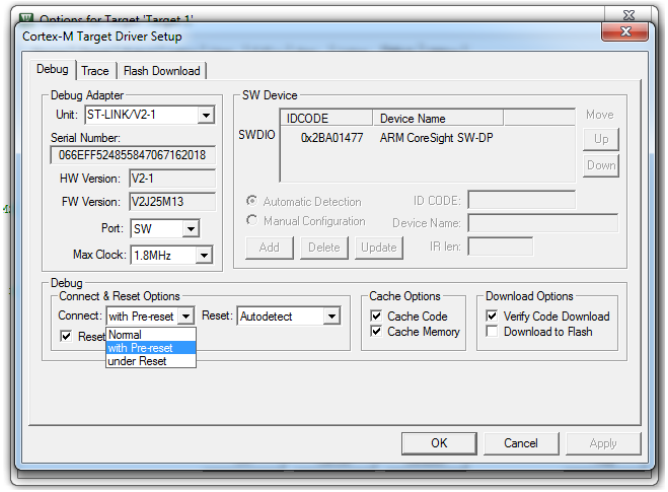
- **Format your code well.** Make sure it's easy to read and understand. Comment where needed but don't comment obvious things it makes the code harder to read. If editing someone else's code, format consistently with the original author.
- Every program you write that you intend to keep around for more than a couple of hours ought to have documentation in it. Don't talk yourself into putting off the documentation. A program that is perfectly clear today is clear only because you just wrote it. Put it away for a few months, and it will most likely take you a while to figure out what it does and how it does it. If it takes you a while to figure it out, how long would it take someone else to figure it out?
- **Write Clearly** - don't be too clever - don't sacrifice clarity for efficiency.
- **Don't over comment.** Use comments only when necessary.
- Format a program to help the reader understand it. **Always Beautify Code.**
- Say what you mean, **simply and directly**.
- **Don't patch bad code** - rewrite it.
- **Make sure comments and code agree.**
- Don't just echo code in comments - **make every comment meaningful.**
- **Don't comment bad code** - rewrite it.
- **The single most important factor in style is consistency.** The eye is drawn to something that "doesn't fit," and these should be reserved for things that are actually different.

Appendix E: "Target not found" error

When you use the STM32L4 discovery board at the very first time, you might not be able to program it in Keil and receive an error of "***Target not found***" when you download the code to the board. This is because the demo program quickly puts the STM32L4 microcontroller into a very low power mode after a reset. There are several ways to solve it. Below are two simplest ones. The error will go away permanently.

Solution 1: In Keil,

1. Click the icon "***Options for Target***"
2. Click "***Debug***" and then "***Settings***"
3. Change the connect from the default value "***normal***" to "***with pre-reset***", as shown below.

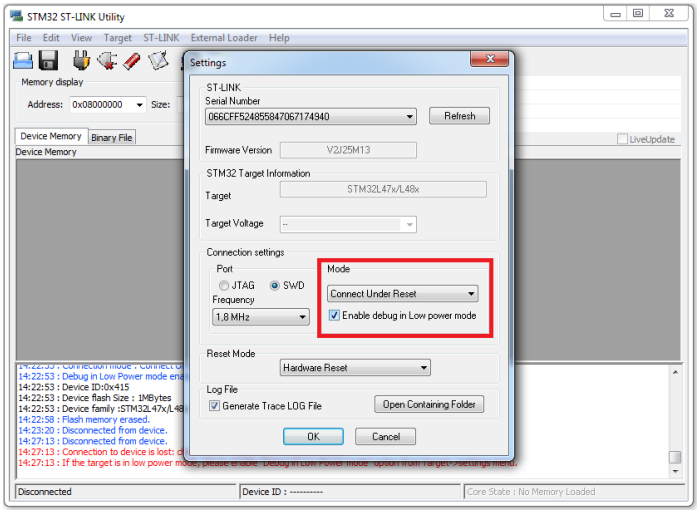


Solution 2: If the previous solution fails, you can download and install ***STM32 ST-Link Utility***

http://www.st.com/en/development-tools/stsw-link004.html

Follow the following steps:

1. Run ST-Link Utility, click menu "***Target***", click "***Settings***"
2. Select "***Connect Under Set***" as the connection mode
3. Click "***Target***" and "***Connect***", and then click "***Target***" and "***Erase Chip***"!
4. Click "***Target***" and "***Disconnect***"


STM32 ST-LINK Utility
File Edit View Target ST-LINK External Loader Help
Memory display
Address: 0x08000000
Size:
Device Memory
Binary File
Device Memory
LiveUpdate
Settings
ST-LINK
Serial Number
066CFF524855847067174940
Refresh
Firmware Version
V2J25M13
STM32 Target Information
Target
STM32L47x/L48x
Target Voltage
Connection settings
Port
JTAG
SWD
Frequency
1,8 MHz
Mode
Connect Under Reset
Enable debug in Low power mode
Reset Mode
Hardware Reset
Log File
Generate Trace LOG File
Open Containing Folder
OK
Cancel
14:22:53 : Debug in Low Power mode
14:22:53 : Device ID:0x415
14:22:53 : Device flash Size : 1MBytes
14:22:53 : Device family :STM32L47x/L48
14:22:58 : Flash memory erased.
14:23:20 : Disconnected from device.
14:27:13 : Disconnected from device.
14:27:13 : Connection to device is lost:
14:27:13 : If the target is in low power
Disconnected
Device ID : ----------
Core State : No Memory Loaded

Lab 1: Interfacing Push-button and LED in C

Pre-Lab Assignment

Student Name: Justin Pacella

TA: _____

Time & Date: 05/27/2022

Video/Textbook Reading

1. Textbook **Chapter 4.6** to review bit-wise operations and **Chapter 14 GPIO.**
2. Watch YouTube Tutorials (http://web.eece.maine.edu/~zhu/book/tutorials.php)
   - Lecture 5: Memory-mapped I/O (8 minutes)
   - Lecture 6: GPIO Output (11 minutes)
   - Lecture 7: GPIO Input (12 minutes)

Summary of Task: Section 1 and section 2.a show (1) how to calculate the mask and the value and (2) how to use them in C code. Follow these examples, find out the mask and value in the rest sections.

1. Enable the clock of GPIO Port A (for Green LED) and Port C (for User Button)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AHB2ENR														RNGEN		AESEN			ADCEN	OTGFSEN					GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
Mask																														1		1
Value																														1		1

In the bitmask (or called **mask**) of a variable, **1** indicates that the corresponding bit in the variable is of interest. In this example, we are interested in bit 0 and bit 2, thus we have the mask as

(1) mask = 0b0101 = 0x5

Mask in hex = 0x00000005 Value in hex = 0x00000005

value = 0b0101 = 0x5

Our C code is like:

```
RCC->AHB2ENR &= ~(mask); // Clear all bits of interests

RCC->AHB2ENR |= value;   // Set bits of interests to target value
```

(2) char a = 0x5;
char mask = a; // = 0x5
a |= mask;

Note 1: Bit-wise Operations

Bit-wise operations update specific bits without affecting the other bits. For example, to set bit 2 of a 32-bit variable aWord, the following is incorrect because it resets all the other bits in this variable.

```
aWord = 4;    // Set bit 2, but also clear all the rest 31 bits
```

The correct one is to use the bit-wise OR operation:

```
aWord |= 4;   // Set bit 2 without affecting the rest
```

Note 2: **Why do we need a bitmask?**

A bitmask is often used for clearing the bits of interest. For example, the following code sets the least significant four bits of variable V to be `0b0101`.

```
Mask = 0xF;  // Mask for least significant four bits

V &= ~Mask;  // Clearing bit 15 and bit 2

V |= 0x5;    // Set least significant four bits to 0x0101
```

The following code, which directly uses bitwise OR without clearing the target bits first, sometimes fails. (why?)

```
V |= 0x5;    // The result is not always correct.
```

(1) mask = 0b1100000000000 = 0xC00
value = 0b0100000000000 = 0x400

2. Pin Initialization for Green LED (PA 5)

a. Set the mode of PA 5 to Output

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mode Register (MODER)	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
Mask																					1	1										
Value																					0	1										

GPIO Mode: Input (00), Output (01), Alternate Function (10), Analog (11, default)

Mask in hex = **0x00000C00** Value in hex = **0x00000400**

char a = 0x400;
char mask = 0xC00;
a |= mask;

Example: The following C code can correctly set the mode of PA 5 to output.

```
GPIOA->MODR &= ~0x00000C00; // Clear bit 10 and bit 11
GPIOA->MODR |=  0x00000400; // Set the mode to output
```

By following the above examples, find out the mask and value in the following sections.

b. Set the output type of PA 5 to Push-Pull

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OTYPER	Reserved																OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
Mask																											1	0	0	0	0	0
Value																	X	X	X		X		X		X	0	X		X		X	X

Push-Pull (0, reset), Open-Drain (1)

don't cares

Mask in hex = 0x 20 Value in hex = 0x 0

c. Set PA 5 to No pull-up No pull-down

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR	PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		**PUPDR5[1:0]**		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
Mask																					1	1	0	0	0							
Value			X				X			X				X					X		0	0					X				X	

No Pullup, No pulldown (00, reset), Pullup (01), Pulldown (10), Reserved (11)

Mask in hex = 0x C00 Value in hex = 0x 0

3. Pin Initialization for the user pushbutton (PC 13)

a. Set the mode of pin PC 13 to Input

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mode Register (MODER)	MODER15[1:0]		MODER14[1:0]		**MODER13[1:0]**		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
Mask					1	1																										
Value					0	0																										

GPIO Mode: Input (00), Output (01), Alternate Function (10), Analog (11, default)

Mask in hex = 0x 0C000000 Value in hex = 0x 0

b. Set PC 13 to no pullup and no pulldown

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR	PUPDR15[1:0]		PUPDR14[1:0]		**PUPDR13[1:0]**		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]		PUPDR7[1:0]		PUPDR6[1:0]		**PUPDR5[1:0]**		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
Mask					1	1																										
Value					0	0																										

No Pullup, No pulldown (00, reset), Pullup (01), Pulldown (10), Reserved (11)

Mask in hex = 0x 0C00 0000 Value in hex = 0x 0000 0000

Lab Demonstration Instructions

Please demonstrate that the green LEs is toggled whenever the pushbutton is pressed. Additionally, please demonstrate that you can toggle the LED state between on and off by directly changing the value of the ODR in the debug environment.

Lab Code Submission Instructions

Submit your code as a text file through Canvas, and please follow the code legibility instructions stated earlier in the lab.

Combine your solutions to the pre-lab assignment and the code as a pdf file. Submit it to Canvas.