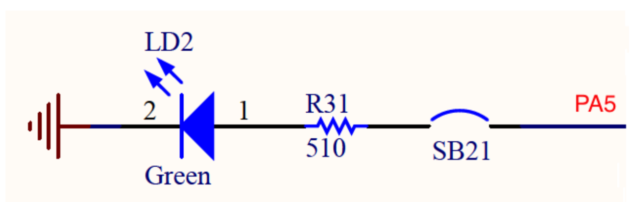
Lab 5: Interfacing Push-button and LED in C

Goals

1. Become familiar with the software development tool (Keil uVision)
2. Create a C program that will run on the board (STM32 NUCLEO-L476RG)
3. Learn the basics of GPIO configuration
4. Use GPIO for digital input (reading input from the push button)
5. Use GPIO for digital output (lighting up a LED)
6. Understand polling I/O (busy waiting) and its inefficiency
7. Understand software debounce techniques

Part A: Lighting up the LED

There is one user-controllable green LED (light-emitting diode) 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, is turned off to save energy.

2. Set the ***mode*** of the corresponding GPIO pin must be set as ***output.*** By default, the mode of all GPIO pins 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)

For this lab, we will program in C.

1. Use C Project Basic Template from https://web.eece.maine.edu/~zhu/book/lab.php

2. See if you can build the template code. It should build, but do not do anything yet.
   (1) Click on the **uvproj** file, and open the project from Keil.
   (2) Click on the "**build**" button to build. This is near the top left and looks like a box with an arrow going into it.
   (3) Now edit the ***main.c*** file in the editor. Modify the C as described below.

3. Modify ***main.c*** to enable the registers you need to turn on the LED. It would help if you had already calculated the values you need to write in advance.

   You will use bitwise operations to set/clear the registers. Some of this might seem repetitive. Use function calls or other more advanced C methods if it helps.

   (1) Program the **AHB2ENR** register to enable the clock of **GPIO Port A**. It would help if you calculated the values for this in the pre-lab.

```
RCC->AHB2ENR
```

   The provided ***stm32l476xx.h*** header file provides some helpers to make this easier. The memory-mapped I/O has been set up to point to the various structures with a volatile pointer. For example, to set the **GPIOCEN** bit (GPIO **Port C** enable) in the **AHB2ENR** register, you would do something like the following:

```
RCC->AHB2ENR |= (1<<2);
```

   The header file ***stm32l476xx.h*** provides:

```
#define  RCC_AHB2ENR_GPIOCEN  ((uint32_t)0x00000004)
```

   Thus, the following is recommended to make your code more readable and easier to debug and maintain.

```
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
```

   Note that the following statement is incorrect because it turns off all the other clocks controlled by this register while enabling the clock of port C. The code should enable the port C clock without impacting the clock settings of the other ports.

```
RCC->AHB2ENR = RCC_AHB2ENR_GPIOCEN;
```

   (2) Program the port A mode register (**MODER**) to set Pin 5 as output.

```
GPIOA->MODER
```

   (3) Program the port A output type register (**OTYPER**) to set Pin 5 as push-pull.

```
GPIOA->OTYPER
```

   (4) Program the port A pull-up/pull-down register (**PUPDR**) to set Pin 5 as no-pull-up no pull-down.

```
GPIOA->PUPDR
```

(5) Finally, program the port A output data register (**ODR**) to set the output of Pin 5 to 1 or 0, which enables or disables the LED, respectively.

```
GPIOA->ODR
```

We did not do this in the pre-lab. To output high (3.0V) on pin 5, just set bit 5 of **ODR** to 1.

(6) At the end of your code, put an infinite loop to keep the code from executing off the end of your program.

```
while(1);
```

4. While doing this, be sure to add comments to your code!

5. Now compile/build your code

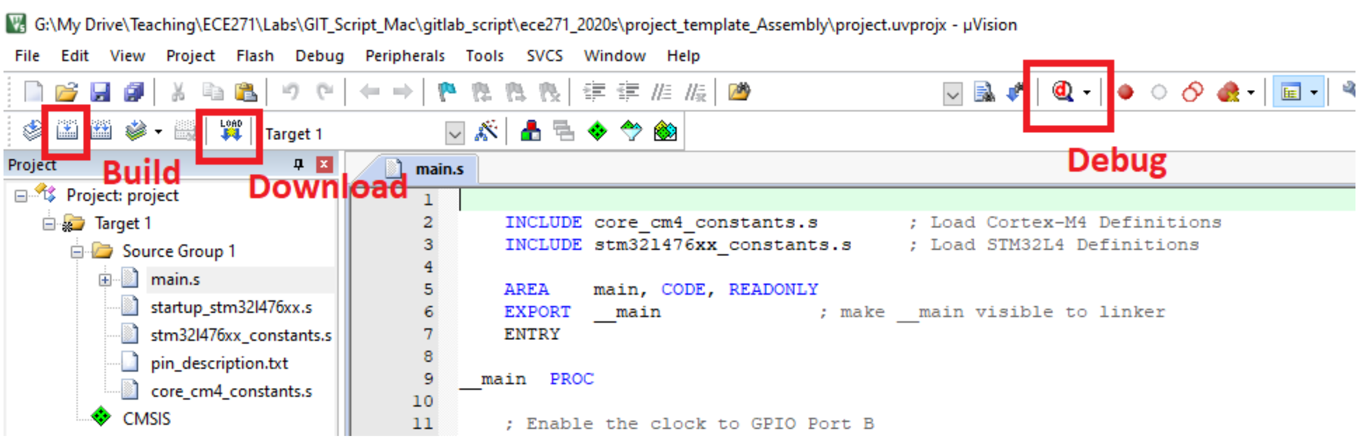
(1) Press the **build** button. Check the window at the bottom for warnings and especially for errors. The IDE will also show little error icons by your code if it detects any.

(2) Then, plug the board into your laptop with the USB cable if you have not already.

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

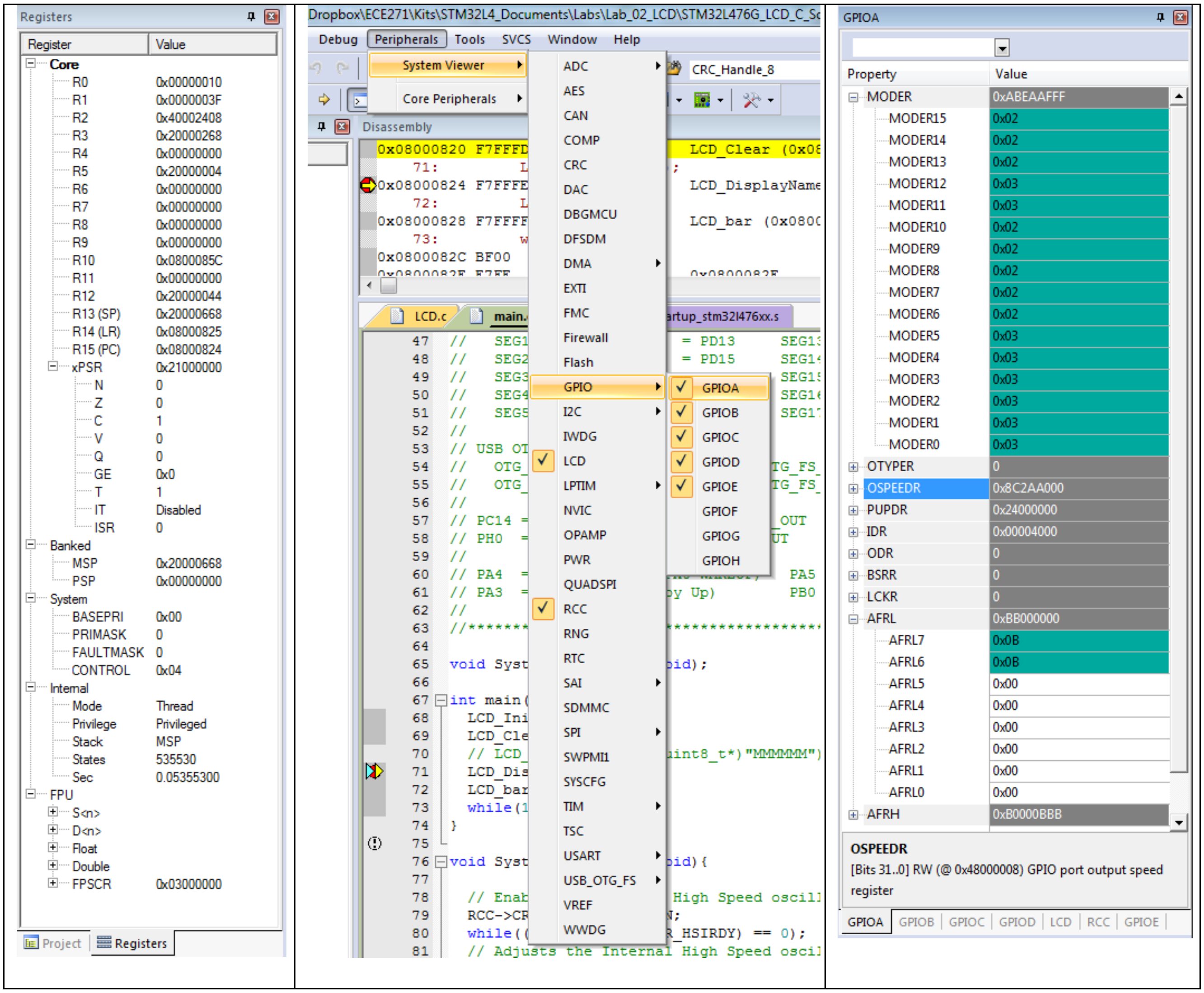
(4) Push the reset button (black) on the board to run your program

> Your code may be buggy, so Keil cannot program the board anymore. In this case, you can use the ***ST-Link Utility*** or ***STM32CubeProgrammer*** to erase the flash memory. See this YouTube tutorial: https://www.youtube.com/watch?v=OiwwB0AvIBI



If the code does not work, you will have to debug your code to find out what is wrong.

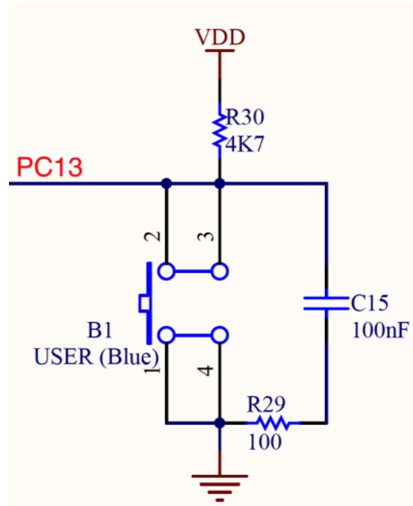
> Review this debug tutorial: https://youtu.be/w4gPcYRk9o8
>
> Keil lets us view the values of Cortex-M registers and all peripheral registers in real time. Click the following: Peripherals, System Viewers, GPIO, and GPIOC


Registers
Register Value
Core
R0 0x00000010
R1 0x0000003F
R2 0x40002408
R3 0x20000268
R4 0x00000000
R5 0x20000004
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x0800085C
R11 0x00000000
R12 0x20000044
R13 (SP) 0x20000668
R14 (LR) 0x08000825
R15 (PC) 0x08000824
xPSR 0x21000000
N 0
Z 0
C 1
V 0
Q 0
GE 0x0
T 1
IT Disabled
ISR 0
Banked
MSP 0x20000668
PSP 0x00000000
System
BASEPRI 0x00
PRIMASK 0
FAULTMASK 0
CONTROL 0x04
Internal
Mode Thread
Privilege Privileged
Stack MSP
States 535530
Sec 0.05355300
FPU
S<n>
D<n>
Float
Double
FPSCR 0x03000000
Project Registers
Debug Peripherals Tools SVCS Window Help
System Viewer
Core Peripherals
ADC AES CAN COMP CRC DAC DBGMCU DFSDM DMA EXTI FMC Firewall Flash GPIO I2C IWDG LCD LPTIM NVIC OPAMP PWR QUADSPI RCC RNG RTC SAI SDMMC SPI SWPMI1 SYSCFG TIM TSC USART USB_OTG_FS VREF WWDG
GPIOA GPIOB GPIOC GPIOD GPIOE GPIOF GPIOG GPIOH
GPIOA
Property Value
MODER 0xABEAAFFF
MODER15 0x02
MODER14 0x02
MODER13 0x02
MODER12 0x03
MODER11 0x03
MODER10 0x02
MODER9 0x02
MODER8 0x02
MODER7 0x02
MODER6 0x02
MODER5 0x03
MODER4 0x03
MODER3 0x03
MODER2 0x03
MODER1 0x03
MODER0 0x03
OTYPER 0
OSPEEDR 0x8C2AA000
PUPDR 0x24000000
IDR 0x00004000
ODR 0
BSRR 0
LCKR 0
AFRL 0xBB000000
AFRL7 0x0B
AFRL6 0x0B
AFRL5 0x00
AFRL4 0x00
AFRL3 0x00
AFRL2 0x00
AFRL1 0x00
AFRL0 0x00
AFRH 0xB0000BBB
OSPEEDR
[Bits 31..0] RW (@ 0x48000008) GPIO port output speed register
GPIOA GPIOB GPIOC GPIOD LCD RCC GPIOE

Part B: Push Button

There is a blue user button on 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.



You will do the above in C.

1. Program **`RCC->AHB2ENR`** to enable the clock to GPIO Port C (by default, they are disabled).
2. Program **`GPIOC->MODER`** to set the mode of Pin PC 13 as **input** (by default, they are analog).
3. Program **`GPIOC->PUPDR`** to set the pull-up/pull-down setting of Pin PC 13 as **no pull-up no pull-down**.
4. Modify your code to have an infinite loop. Each time through the loop, read the status of the USER push button. Each time the button is pressed, the LED is toggled.

   (1) Read **`GPIOC->IDR`** and use a proper bitwise operation to check whether bit 13 is 0. As the hardware diagram shows, the voltage on pin PC 13 is high if the user button is not pressed. It is strongly recommended to use constants defined in the provided ***stm32l476xx.h*** header file to make your code more readable. For example, the bit mask of bit 13 is defined in the header file, equivalent to `1<<13`.

   ```
   // Provided in stm32l476xx.h, no need to redefine it.
   #define GPIO_IDR_IDR_13  ((uint32_t)0x00002000)
   ```

   The following statement waits until the button is pressed.

   ```
   While(GPIOC->IDR & GPIO_IDR_IDR_13);
   ```

   (2) Toggle the LED. Review the bitwise operations regarding how to toggle a bit in a register.

Assignment #2 (Due Date: 04.12.2024)

Using GPIO and LED to send out SOS in Morse code (· · · – – – · · ·) if the user button is pressed. DOT, DOT, DOT, DASH, DASH, DASH, DOT, DOT, DOT. DOT is on for ¼ second, and DASH is on for ½ second, with ¼ second between these light-ons. Search for how to create delays for ½ second and ¼ second in C.

Important Notes:

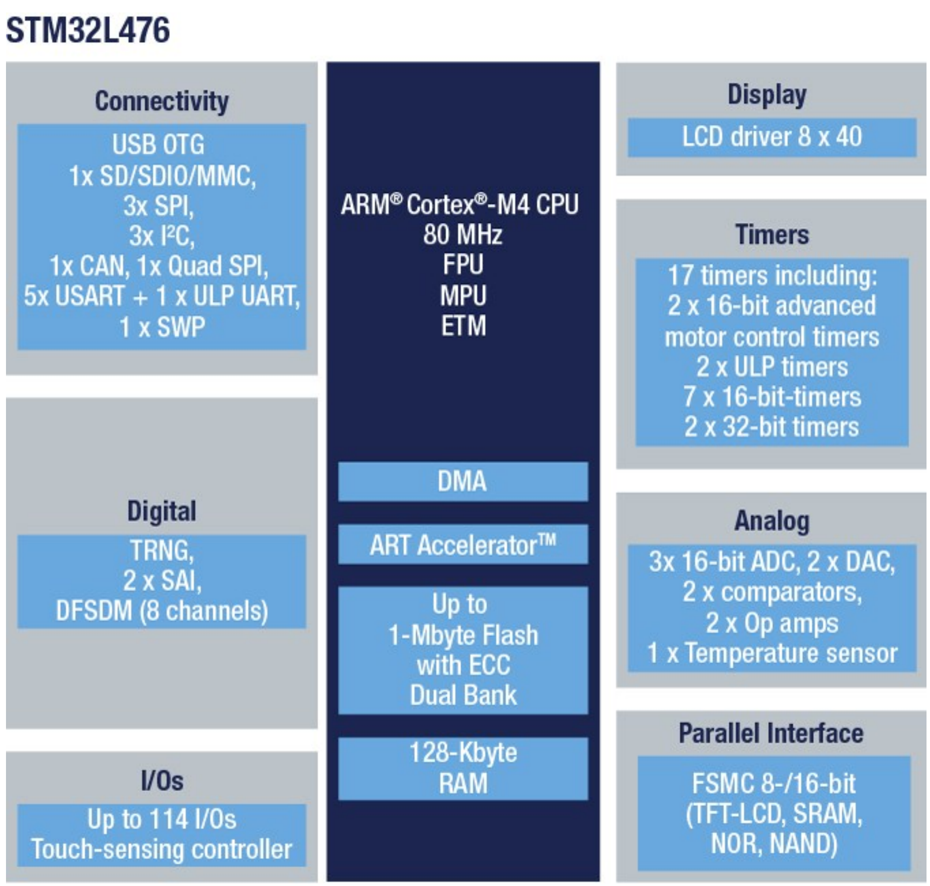
- You are required to work in groups of 4-5 people.
- Please use Keil MDK-ARM IDE, create a project, build, and load the program onto the board successfully.
- Please use the materials that we gave to you for this project.
- You are required to give a demo session for the lab assignments. Demo date and the schedule will be announced later.
- You will have a 30-minutes quiz after submitting the lab assignment. So the assignment quiz for this lab will be on **05.12.2024**.
- Please write necessary comments for the project.
- We use tools that automatically detect plagiarism among the submissions!
- In case of any form of copying and cheating on solutions, you will get **FF** grade from the course! You should submit your own work. In case of any forms of cheating or copying, both giver and receiver are equally culpable and suffer equal penalties.
- Please zip and submit your files using filename StudentNumbersAssignment2.zip to Canvas system (under Assignments tab).
- No late submission will be accepted.

Appendix A: Microcontroller STM32L476

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

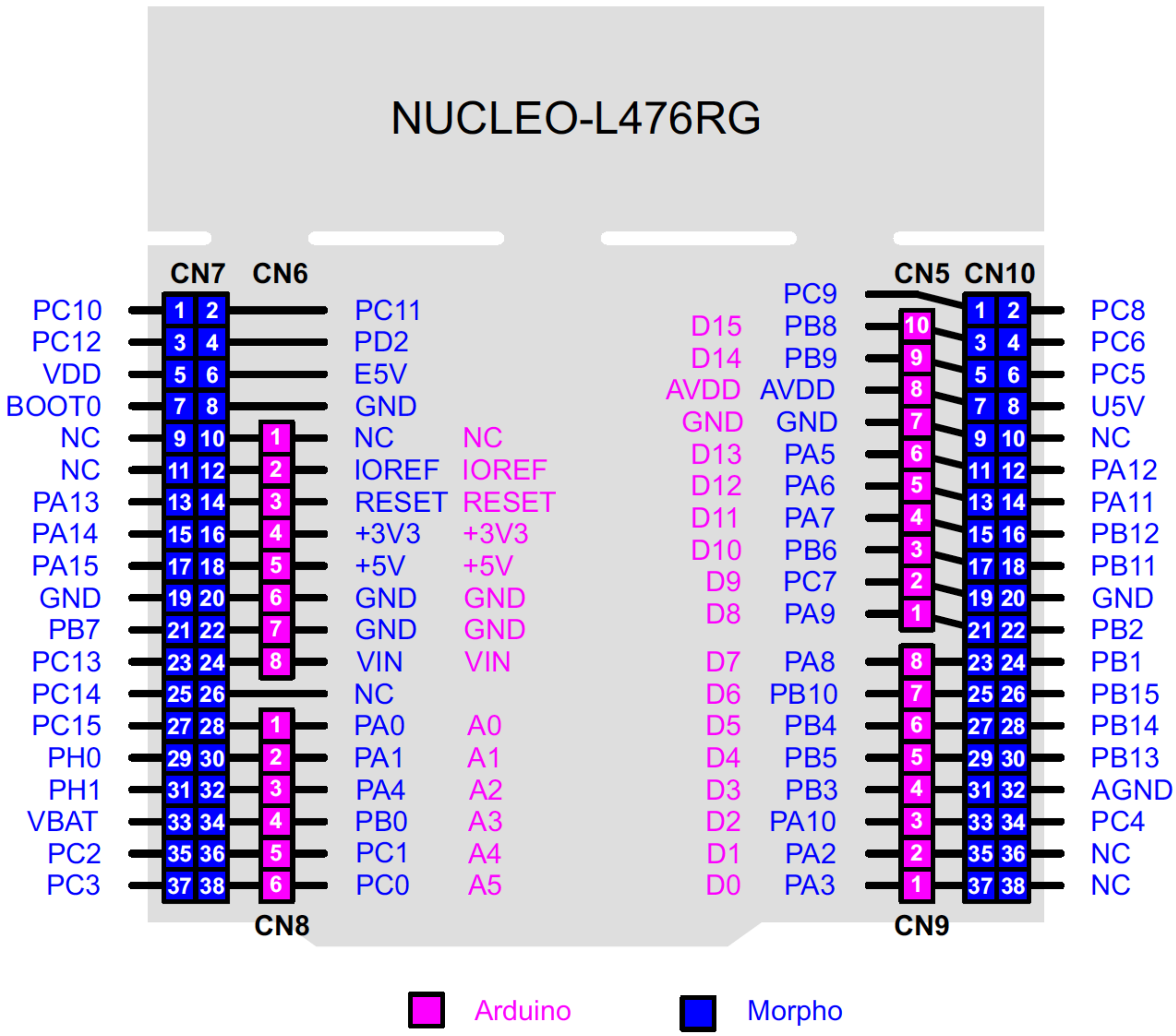
- Part Number: **STM32L476RGT6**
- Total number of pins: 64
- Total number of I/O pins: 51
- Maximum clock frequency: 80MHz
- Program memory size: 1 MB
- Operating supply voltage: 1.71V to 3.6V
- I/O voltage: 3.3V
- Analog supply voltage: 3.3V

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 opened by default
Blue user button	PC 13	Pulled up externally
Black reset button	NRST	Connect to the 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



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, software can select four different clock sources to drive the processor core, 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 inappropriate 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. See the function void `enable_HSI()` for details.

```
// User HSI (high-speed internal) as the processor clock

void enable_HSI(){
  // Enable High Speed Internal Clock (HSI = 16 MHz)
  RCC->CR |= ((uint32_t)RCC_CR_HSION);

  // wait until HSI is ready
  while ( (RCC->CR & (uint32_t) RCC_CR_HSIRDY) == 0 ) {;}

  // Select HSI as system clock source
  RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
  RCC->CFGR |= (uint32_t)RCC_CFGR_SW_HSI;  //01: HSI16 oscillator used as system clock

  // Wait till HSI is used as system clock source
  while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) == 0 ) {;}
}
```

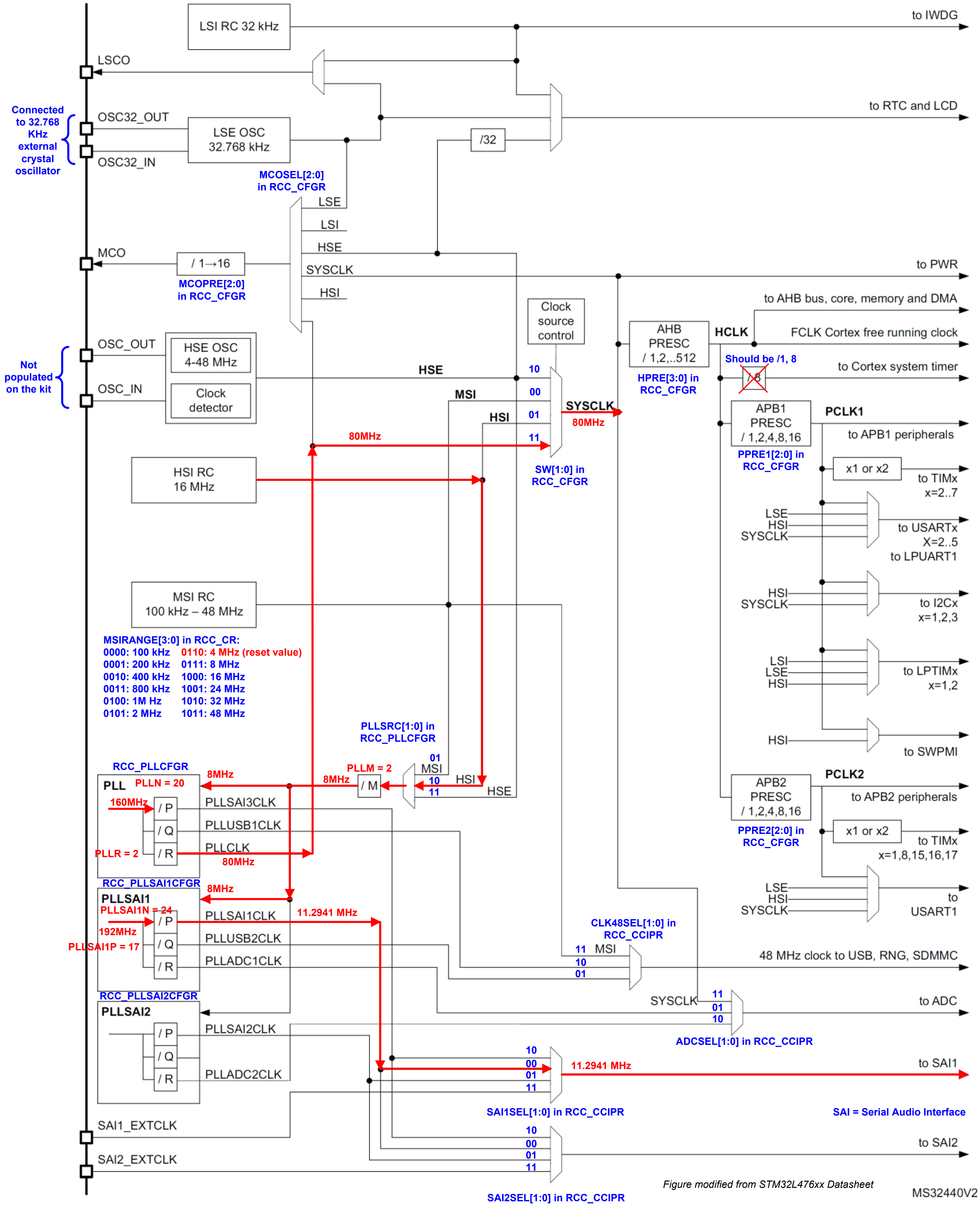


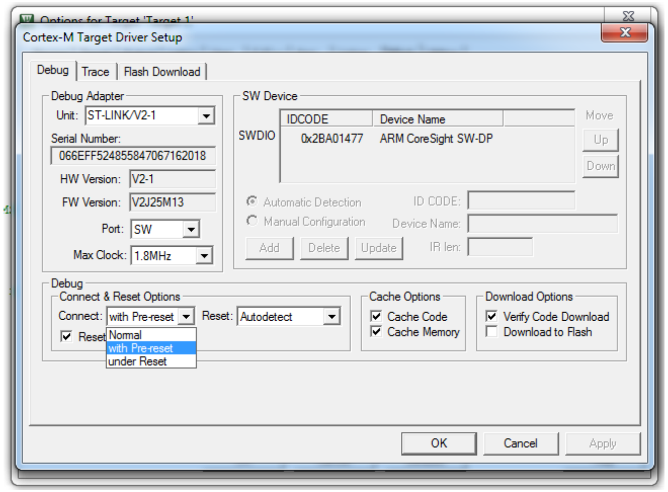
Figure modified from STM32L476xx Datasheet

Appendix D: "Target not found" error

When you use the STM32L4 discovery board for the 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 low-power mode after a reset. There are several ways to solve it. Below are the 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 the 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***"