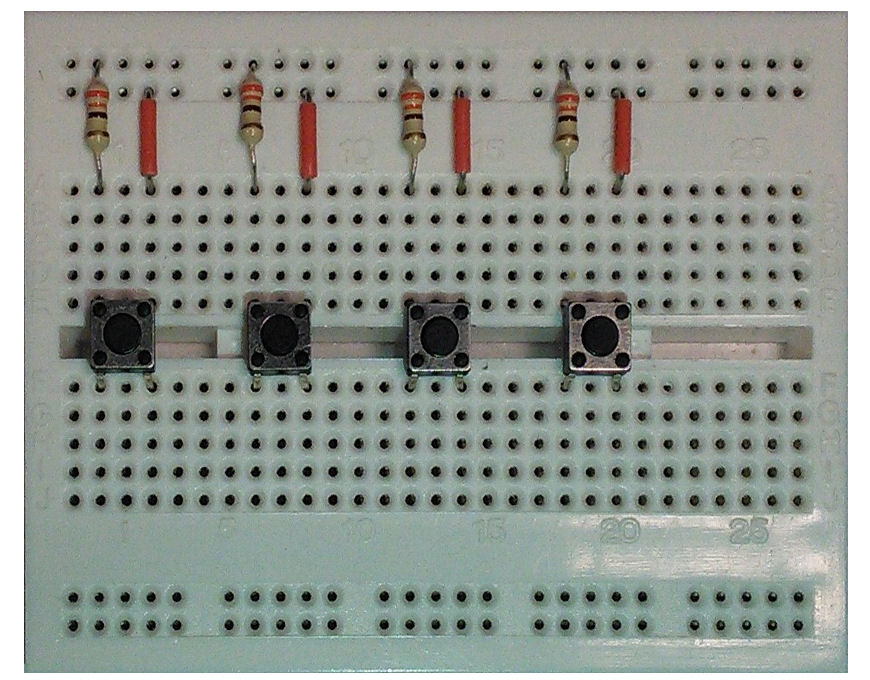
Lab Exercise:  
digital input/ output and GPIO

# Overview

In this lab, you will learn how to configure a General Purpose Input Output (GPIO) peripheral in a low-level (register-level) in practice.



Vcc

Buttons 1-4

PA\_10

PB\_3

PB\_5

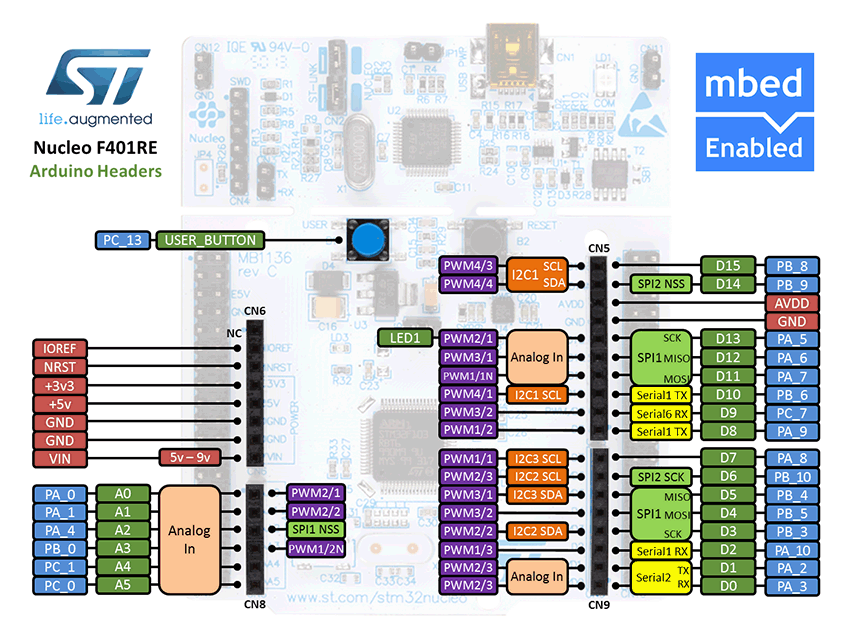
PB\_4

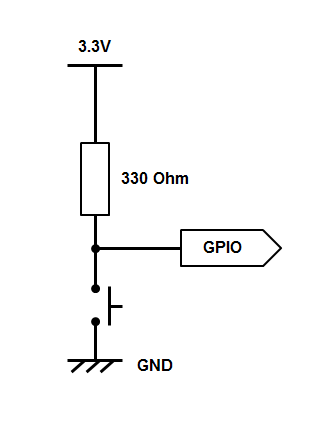
# implementation details

## Hardware

### Nucleo F401RE board

The Nucleo F401RE board pin descriptions are shown below:



Connect the buttons according to the diagram on the left. Your buttons should pull the GPIO pin low.

|  |  |
| --- | --- |
| **Pin** | **mbed API** |
| Button 1 | PA\_10 |
| Button 2 | PB\_3 |
| Button 3 | PB\_5 |
| Button 4 | PB\_4 |
| Red LED | PB\_10 |
| Green LED | PA\_8 |
| Blue LED | PA\_9 |

ARM

Processor

MCU

Board

Button 2

Button 1

Button 3

Green LED

Red LED

Button 4

Bus Interface

GPIO

…..

Blue LED

Once the status of a switch is changed, a bit of the GPIO input register will be set. It will allow the program to know which button was pressed.

## Software

### configure the GPIO peripheral

#### Bit operations

There are several ways how you can write to a register. It is very important to use the correct one to achieve the desired result. To illustrate this example, let’s use a fictional register REG.

REG = 0x04; //set bit[2] and clear other bits

REG |= 0x04; //set bit[2] and keep others unchanged

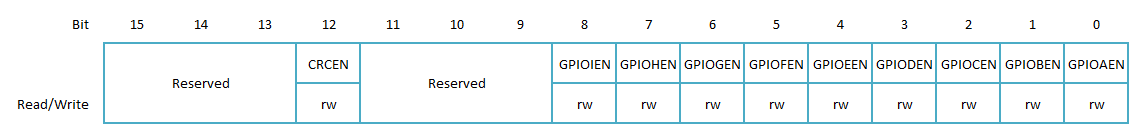
REG = ~0x04; //clear all bits except bit[2]

REG &= ~0x04; //clear bit[2] and keep others unchanged

((REG->DIR & (1 << 2))>> 2); //read bit[2] of the DIR register

#### Clock

Firstly, we have to turn on the clock signal to GPIOA, GPIOB, GPIOC and GPIOD ports. This is done by writing to the RCC AHB1 peripheral clock enable register (AHB1ENR).



For example, to turn on the clock signal to GPIOB, we need to write 0x02 to the RCC\_AHB1ENR register, or use shifting operation such as:

RCC\_AHB1ENR |= 0x02; //turn on clock to PORTA

Or

RCC\_AHB1ENR |= 0x01 << 2; //turn on clock to PORTA

#### GPIO REGISTERS

Each GPIO port has 10 32bit registers to configure every pin.

* **GPIO port mode register (GPIOx\_MODER)**

These bits are written by software to configure the I/O direction mode

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

* **GPIO port output type register (GPIOx\_TYPER)**

These bits are written by software to configure the output type of the I/O port

0: Output push-pull (reset state)

1: Output open-drain

* **GPIO port output speed register (GPIOx\_OSPEEDR)**

These bits are written by software to configure the I/O output speed

00: 2 MHz Low speed

01: 25 MHz Medium speed

10: 50 MHz Fast speed

11: 100 MHz High speed on 30pF

* **GPIO port pull-up/pull-down register (GPIOx\_PUPDR)**

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

* **GPIO port input data register (GPIOx\_IDR)**

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port

* **GPIO port output data register (GPIOx\_ODR)**

These bits can be read and written by software

* **GPIO port bit set/reset register (GPIOx\_BSRR)**

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000

* **GPIO port configuration lock register (GPIOx\_LCKR)**

This register is used to lock the configuration of the port bits when a correct write sequence is applied to bit 16 (LCKK). The value of bits [15:0] is used to lock the configuration of the GPIO. During the write sequence, the value of LCKR[15:0] must not change. When the LOCK sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next reset.

* **GPIO alternate function low register (GPIOx\_AFRL)**

These bits are written by software to configure alternate function I/Os

* **GPIO alternate function high register (GPIOx\_AFRH)**

These bits are written by software to configure alternate function I/Os

You can find a detailed GPIO register map on pages 203 – 205 of the Reference Manual.

#### Register Addresses

All the registers are one-to-one mapped to the 32-bit global memory space.

The base addresses for GPIO ports are listed below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Peripheral | RCC | GPIOA | GPIOB | GPIOC | GPIOD |
| Base Address | 0x4002 3800 | 0x4002 0000 | 0x4002 0400 | 0x4002 0800 | 0x4002 0C00 |

Address offsets for the GPIO registers are listed below:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Register | MODER | TYPER | OSPEED | PUPDR | IDR | ODR | BSRR | LCKR | AFRL | AFRH |
| Offset | 0x00 | 0x04 | 0x08 | 0x0C | 0x10 | 0x14 | 0x18 | 0x1C | 0x20 | 0x24 |

#### Define GPIO Structure

To define and use a GPIO peripheral, it is better to define them into a structure and map it to the memory, for example:

typedef struct{

volatile unsigned int MODER; //offset 0x00

volatile unsigned int TYPER; //offset 0x04

volatile unsigned int OSPEED; //offset 0x08

…

} GPIO\_Type

#define GPIOA\_BASE (0x40020000)

#define GPIOB\_BASE (0x40020400)

…

#define GPIOA (GPIO\_Type \*)GPIOA\_BASE)

#define GPIOB (GPIO\_Type \*)GPIOB\_BASE)

…

To access a register in the peripheral, you can simply write:

GPIOA->MODER &= ~(0x01 << 4) //clear bit 4 of the GPIOA MODER

For more information you can refer to the Nucleo F401RE reference manual at: http://www.st.com/web/en/resource/technical/document/reference\_manual/DM00096844.pdf

## your Application Code

In this lab, you need to complete the code to perform the following functions:

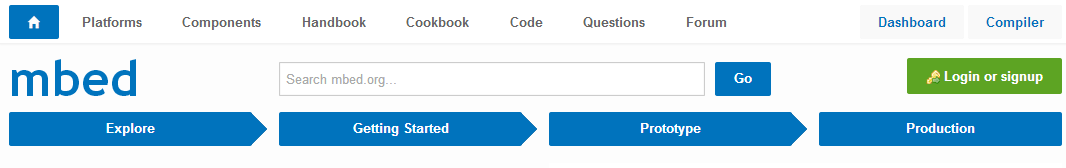
* Implement the following files:
  + switches.c
  + leds.c
  + main.c
* Use the GPIOs to read the status of each button
* Using GPIO to control the RGB LEDs according to the status of buttons, for example:
  + Button 1 – red LED
  + Button 2 – blue LED
  + Button 3 – green LED
  + Button 4 – white (red, green and blue altogether)

# use mbed online compiler

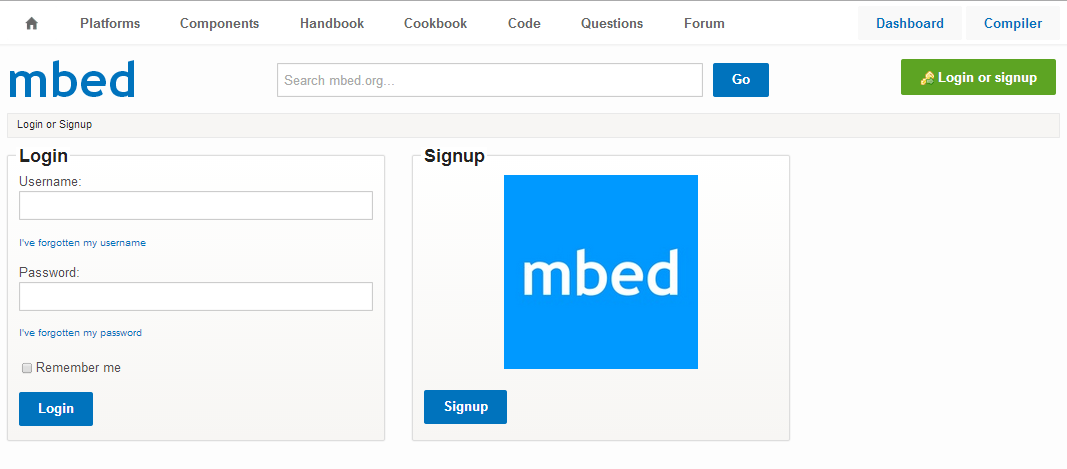
The mbed online Compiler provides a lightweight online C/C++ IDE that is pre-configured to let you quickly write programs, compile and download them to run on your mbed Microcontroller. The mbed online compiler is web based hence you don't have to install or set up anything to get running with mbed.

## create your first mbed online project

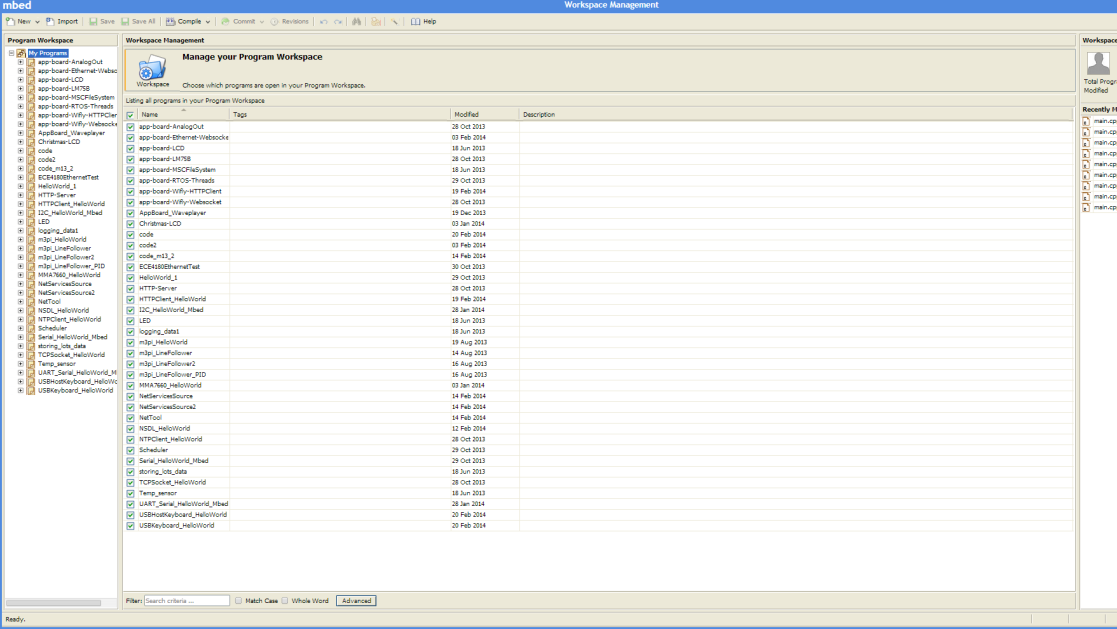
1. Go to mbed.org, and click “compiler”



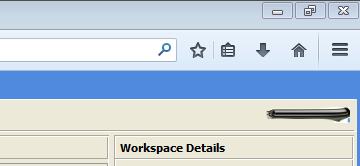
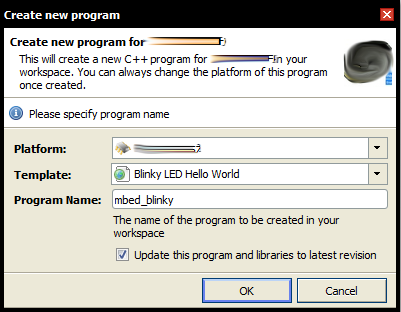
1. Register an account and then login



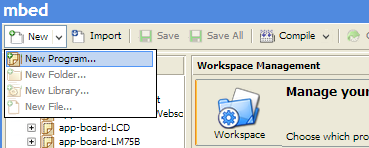
1. The main IDE of mbed online compiler will be displayed



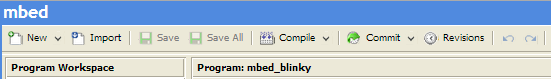
You can select the board by clicking option on the top-right. In our case, we use Nucleo F401RE.



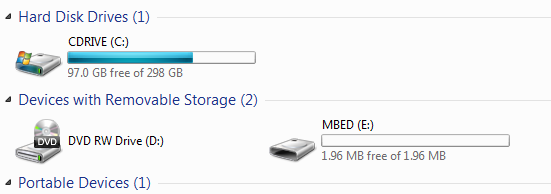
1. Create a helloworld project (blinky LED program)



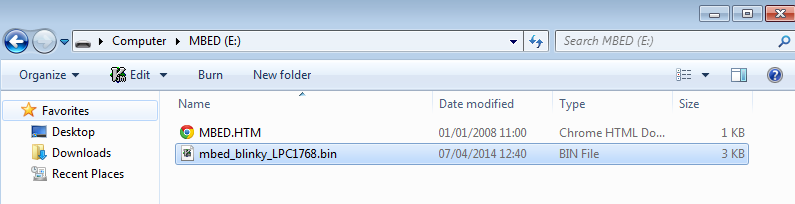
1. Compile the program



1. The program file will be generated and downloaded to your default download directory (set by your web browser)
2. Connect the Nucleo F401RE board to your PC via an USB cable, the mbed will appear as a removable storage device



1. Copy the downloaded program file to the mbed root directory



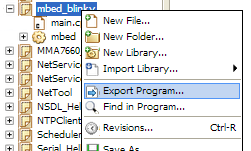
1. Reset the mbed board, the latest copied program file will be the default program to run.

More detail can be found at <https://mbed.org/handbook/mbed-Compiler-Getting-Started>

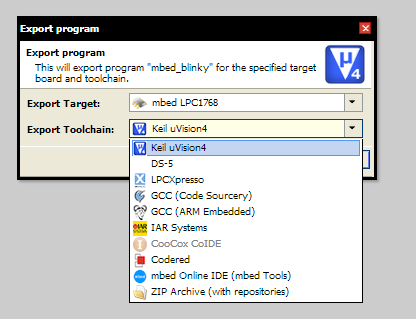
## Exporting to offline toolchains

The online project can be exported to offline toolchains, such as Keil MDK in our case.

1. Right click the project and select “export program”



1. Select Keil MDK project, and click “export”, the project folder will be downloaded to your local machine



Note: The local Keil MDK project allows you to download and debug your program (see getting started). In this set of teaching material, all the lab projects have been exported to local Keil MDK project, making it easier for you to start with.