LAB: GPIO Digital InOut(eval board)

LAB: GPIO Digital InOut

Date: 2025-09-17
Github: https://github.com/passtock/EC-jylee-561/tree/main
Demo Video: https://youtube.com/shorts/exlSgk05boc?feature=share, https://youtube.com/shorts/UrGJzFBJATw?feature=share, https://youtube.com/shorts/5_jPjStjiNE?feature=share
PDF version:

Introduction

In this lab, you are required to create a simple program that toggle multiple LEDs with a push-button input. Create HAL drivers for GPIO digital in and out control and use your library.

You must submit

- LAB Report (*.pdf)
- Zip source files(main*.c, ecRCC.h, ecGPIO.h etc...).
  - Only the source files. Do not submit project files

Requirement

Hardware

- MCU
  - NUCLEO-F411RE
- Eval Board
- Actuator/Sensor/Others:

Software

- PlatformIO, CMSIS, EC_HAL library

.

Problem 1: Create EC_HAL library

Procedure

Library Header Files

Create the library directory `\repos\EC\include\`.

- **For uVision User:** Save your header library files in this directory. See here for detail.

DO NOT make duplicates of library files under each project folders

- **For VS.Code User:** Save your header library files in this directory. See here for detail.

Download necessary library files: Download library files from here

- `ecRCC2.h, ecRCC2.c`
- `ecPinNames.h, ecPinNames.c`

Create your own library for GPIO Digital_In and Out

- First, download: `ecGPIO2_student.h, ecGPIO2_student.c`

- Download library file from here
- Then, modify it as ecGPIO2.c, ecGPIO2.h

ecRCC2.h (provided)

```
void RCC_HSI_init(void);
void RCC_GPIOA_enable(void);
void RCC_GPIOB_enable(void);
void RCC_GPIOC_enable9void);


```

ecGPIO2_student.h

```
void GPIO_init(PinName_t pinName, int mode);
void GPIO_write(PinName_t pinName, int Output);
int  GPIO_read(PinName_t pinName);
void GPIO_mode(PinName_t pinName, int mode);
void GPIO_ospeed(PinName_t pinName, int speed);
void GPIO_otype(PinName_t pinName, int type);
void GPIO_pupd(PinName_t pinName, int pupd);

/*
// Version 1
void GPIO_init(GPIO_TypeDef *Port, int pin,    int mode);
void GPIO_write(GPIO_TypeDef *Port, int pin,    int output);
int  GPIO_read(GPIO_TypeDef *Port, int pin);
void GPIO_mode(GPIO_TypeDef* Port, int pin, int mode);
void GPIO_ospeed(GPIO_TypeDef* Port, int pin,    int speed);
void GPIO_otype(GPIO_TypeDef* Port, int pin,    int type);
void GPIO_pupd(GPIO_TypeDef* Port, int pin,    int pupd);

```

- Real code in **ecGPIO2_student.c**

```
/*----------------------------------------------------------------\
@ Embedded Controller by Young-Keun Kim - Handong Global University
Author              : SSS LAB
Created             : 05-03-2021
Modified            : 08-23-2024
Language/ver        : C++ in Keil uVision
name                                : leejeayong
number                      : 22000561

Description         : Distributed to Students for LAB_GPIO
/----------------------------------------------------------------*/



#include "stm32f4xx.h"
#include "stm32f411xe.h"
#include "ecGPIO2.h"

void GPIO_init(PinName_t pinName, uint32_t mode){
	GPIO_TypeDef * Port;
	unsigned int pin;
	ecPinmap(pinName, &Port, &pin);

	// mode   : Input(0), Output(1), AlterFunc(2), Analog(3)
	if (Port == GPIOA)
		RCC_GPIOA_enable();
	if (Port == GPIOB)
		RCC_GPIOB_enable();
	if (Port == GPIOC)
		RCC_GPIOC_enable();



```

```
        //[TO-DO] YOUR CODE GOES HERE
        // Make it for GPIOB, GPIOD..GPIOH

        // You can also make a more general function of
        // void RCC_GPIO_enable(GPIO_TypeDef *Port);


        GPIO_mode(pinName, mode);
}
void GPIO_write(PinName_t pinName, int Output){
        GPIO_TypeDef * Port;
        unsigned int pin;
        ecPinmap(pinName,&Port,&pin);
        if(Output) Port->BSRR = 1UL << pin;    // Set bit
        else        Port->BSRR = 1UL << (pin+16); // Reset bit
}


// GPIO Mode          : Input(00), Output(01), AlterFunc(10), Analog(11)
void GPIO_mode(PinName_t pinName, uint32_t mode){
    GPIO_TypeDef * Port;
    unsigned int pin;
    ecPinmap(pinName,&Port,&pin);
    Port->MODER &= ~(3UL<<(2*pin));
    Port->MODER |= mode<<(2*pin);

}


// GPIO Speed          : Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
void GPIO_ospeed(PinName_t pinName, int speed){
        GPIO_TypeDef * Port;
        unsigned int pin;
        ecPinmap(pinName,&Port,&pin);
        Port->OSPEEDR &= ~(3UL<<(2*pin));
        Port->OSPEEDR |= speed<<(2*pin);
}

// GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
void GPIO_otype(PinName_t pinName, int type){
        GPIO_TypeDef * Port;
     unsigned int pin;
        ecPinmap(pinName,&Port,&pin);
        Port->OTYPER &= ~(1UL<<pin);
        Port->OTYPER |= type<<pin;
}

// GPIO Push-Pull      : No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
void GPIO_pupd(PinName_t pinName, int pupd){
        GPIO_TypeDef * Port;
        unsigned int pin;
        ecPinmap(pinName,&Port,&pin);
        Port->PUPDR &= ~(3UL<<(2*pin));
        Port->PUPDR|= pupd <<(2*pin);
}

int GPIO_read(PinName_t pinName){
        GPIO_TypeDef * Port;
        unsigned int pin;
        ecPinmap(pinName,&Port,&pin);
        return (Port->IDR & (1UL << pin));

}
*/

```

Problem 2: Toggle a single LED with Digital Sensor(Photodetector)

Procedure

1. Connect the evaluation board to the MCU.
2. Create a new project under the directory \repos\EC\lab\

- The project name is “**LAB_GPIO_DIO_LED_Photosensor”.**
- Name the source file as “**LAB_GPIO_DIO_LED_Photosensor.c**”
- Use the example code provided here.

2. If you have not done, include your library **ecGPIO2.h, ecGPIO2.c** in \repos\EC\include\.

1. You must modify the **platformio.ini ,** to add new environment. Read here for detail

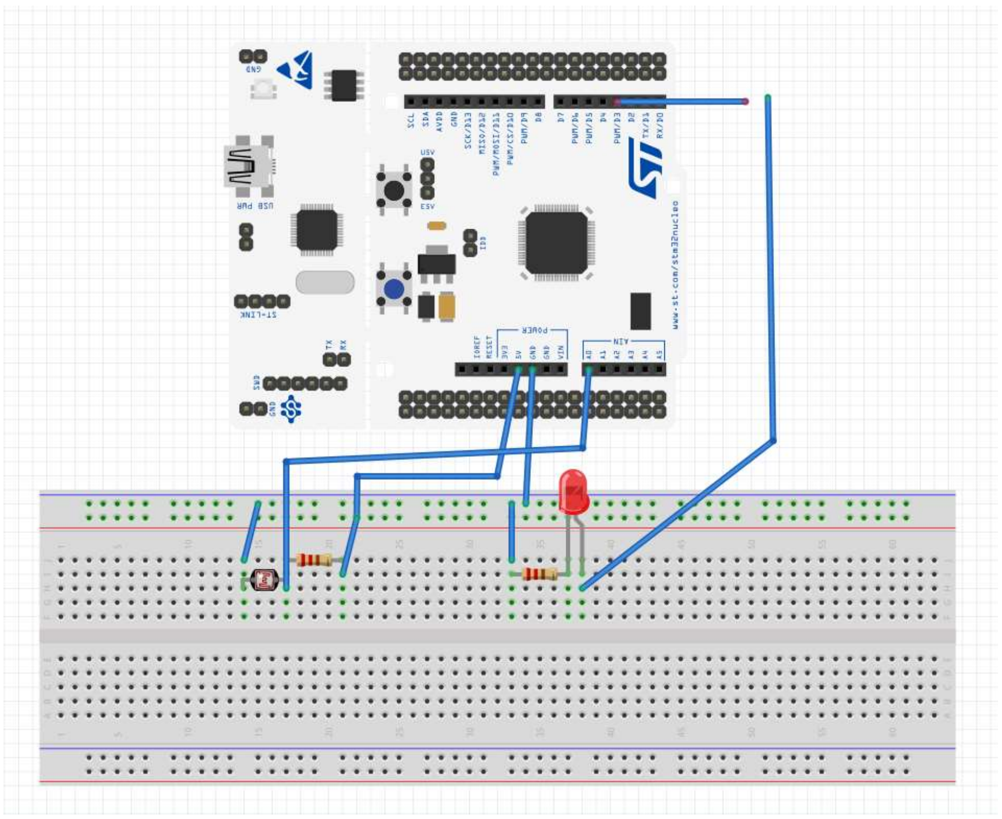
4. Toggle the LED by covering the photodetector sensor.

- Dark (LED ON), Bright (LED OFF) and repeat

Configuration

Digital Sensor (Photodetector)	LED
Digital In	Digital Out
GPIOA, Pin 0	GPIOB, Pin 12
PULL-UP	Open-Drain, Pull-up, Medium Speed

Circuit Diagram



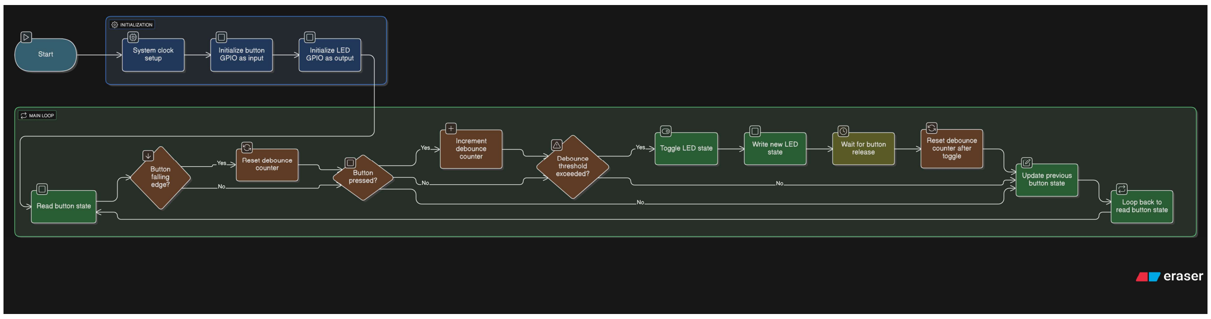
Code

```
#include "ecRCC2.h"
#include "ecGPIO2.h"

```

```
#define LED_PIN PB_12
#define BUTTON_PIN PA_0

// Initialiization
void setup(void) {
      RCC_HSI_init();
      // initialize the pushbutton pin as an input:
      GPIO_init(BUTTON_PIN, INPUT);
      // initialize the LED pin as an output:
      GPIO_init(LED_PIN, OUTPUT);
      GPIO_pupd(BUTTON_PIN, 1); // Pull-up
    GPIO_ospeed(LED_PIN, 1);  // Medium speed
}

int main(void) {
      setup();
      int buttonState=0;

      while(1){
                // check if the pushbutton is pressed. Turn LED on/off accordingly:
                buttonState = GPIO_read(BUTTON_PIN);
                if(buttonState)     GPIO_write(LED_PIN, LOW);
                else                GPIO_write(LED_PIN, HIGH);
      }
}

```

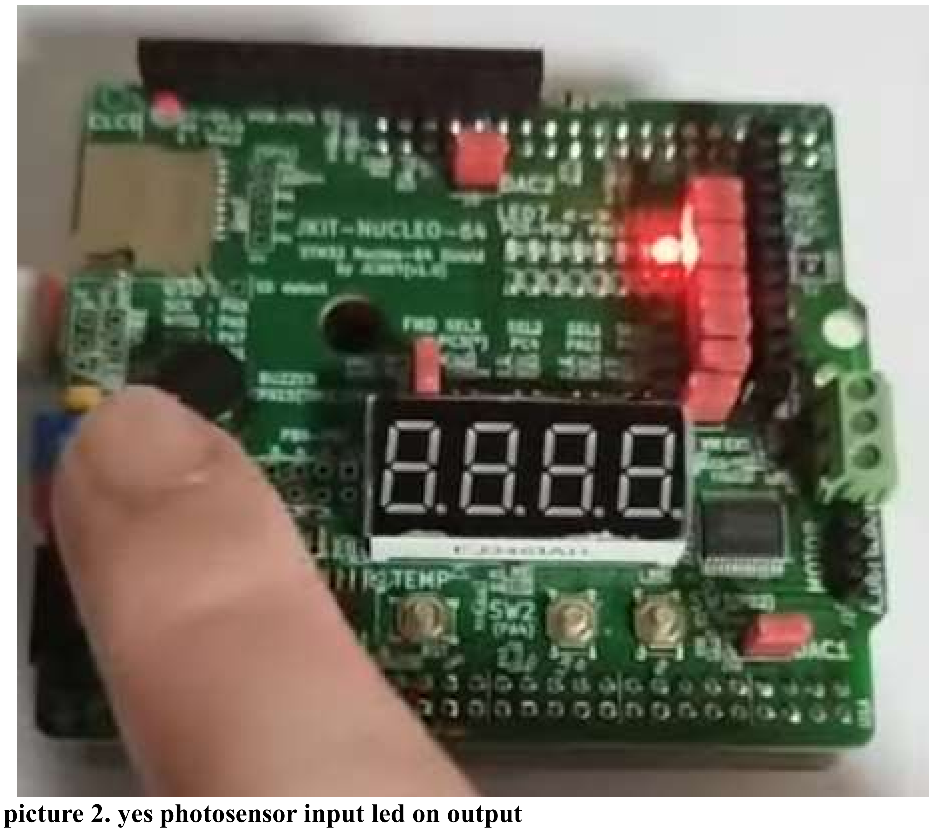
It configures PB_12 as an LED output and PA_4 as a button input, treats the button as active-low, and toggles the LED exactly once per valid press using a simple software debounce and a wait-until-release guard.

Flow chart


Start
INITIALIZATION
System clock setup
Initialize button GPIO as input
Initialize LED GPIO as output
MAIN LOOP
Read button state
Button falling edge?
Yes
No
Reset debounce counter
Button pressed?
Yes
No
No
Increment debounce counter
Debounce threshold exceeded?
Yes
No
Toggle LED state
Write new LED state
Wait for button release
Reset debounce counter after toggle
Update previous button state
Loop back to read button state
eraser

Results

picture 1. no photosensor input no output



picture 2. yes photosensor input led on output

https://youtube.com/shorts/exlSgk05boc?feature=share

Discussion

1. Find out a typical solution for software debouncing and hardware debouncing.
   Fot the software debouncing, using the a little delay after changing the input and for the hardware debouncing, there are resistor in series with the switch,and capacitor connects from the node between the resistor and switch to ground this is low pass filter.
2. What method of debouncing did this NUCLEO board use for the push-button(B1)?
   In our board doesn't include hardware diagram so I can't find method. But in another,B1 button on the NUCLEO connected to PC13, and the button signal line consists of capacitor , a resistor , and a series resistor , so that the RC lowpass fishing user registers the bounce there.

Problem 3: Toggle a single LED with a Button

Procedure

1. Create a new project under the directory \repos\EC\lab\

- The project name is "**LAB_GPIO_DIO_LED_Button".**
- Name the source file as "**LAB_GPIO_DIO_LED_Button.c"**
- Use the example code provided here.

2. Include your library **ecGPIO2.h, ecGPIO2.c** in \repos\EC\include\.

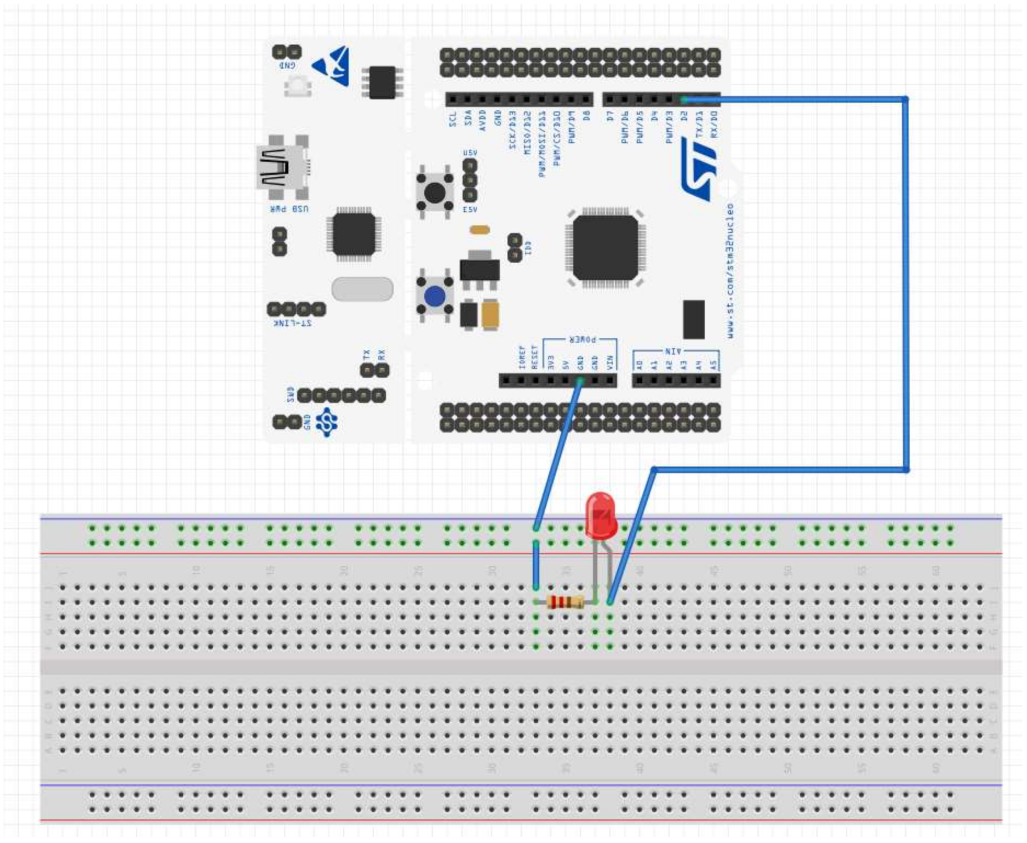
3. Toggle the LED by pushing the button.

- Push button (LED ON), Push Button (LED OFF) and repeat

Configuration

Button (B1)	LED
Digital In	Digital Out
GPIOA, Pin 4	GPIOB, Pin 12
PULL-UP	Open-Drain, Pull-up, Medium Speed

Circuit Diagram

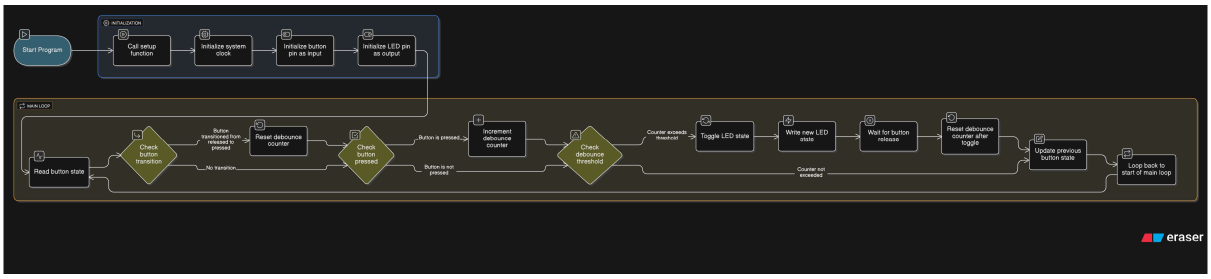


Code

```
#include "ecRCC2.h"
#include "ecGPIO2.h"

#define LED_PIN      PB_12
#define BUTTON_PIN   PA_4

void setup(void) {
    RCC_HSI_init();
    GPIO_init(BUTTON_PIN, INPUT);
    GPIO_init(LED_PIN, OUTPUT);
    GPIO_pupd(BUTTON_PIN, 1); // Pull-up
    GPIO_ospeed(LED_PIN, 1);  // Medium speed
}

int main(void) {
    setup();
    int buttonState = 0;
    int prevButtonState = GPIO_read(BUTTON_PIN);
    int ledState = 0;

    int debounceCounter = 0;
    const int DEBOUNCE_THRESHOLD = 500;

    while(1){
        buttonState = GPIO_read(BUTTON_PIN);

        if (buttonState == 0 && prevButtonState == 1) {
            debounceCounter = 0;
        }

        if (buttonState == 0) {
            debounceCounter++;
        }

        if (debounceCounter > DEBOUNCE_THRESHOLD) {
            ledState = !ledState;
            GPIO_write(LED_PIN, ledState);

            while(GPIO_read(BUTTON_PIN) == 0) {}

            debounceCounter = 0;
        }

        prevButtonState = buttonState;
    }
}

```

This program configures PB_12 as an LED output and PA_4 as an active-low pushbutton input, then toggles the LED exactly once per valid press using a counter-based software debounce and a wait-until-release guard to avoid multiple toggles per press. The internal high-speed oscillator (HSI) is selected during initialization so the MCU runs from an internal clock source before handling GPIO reads and writes

Flow chart

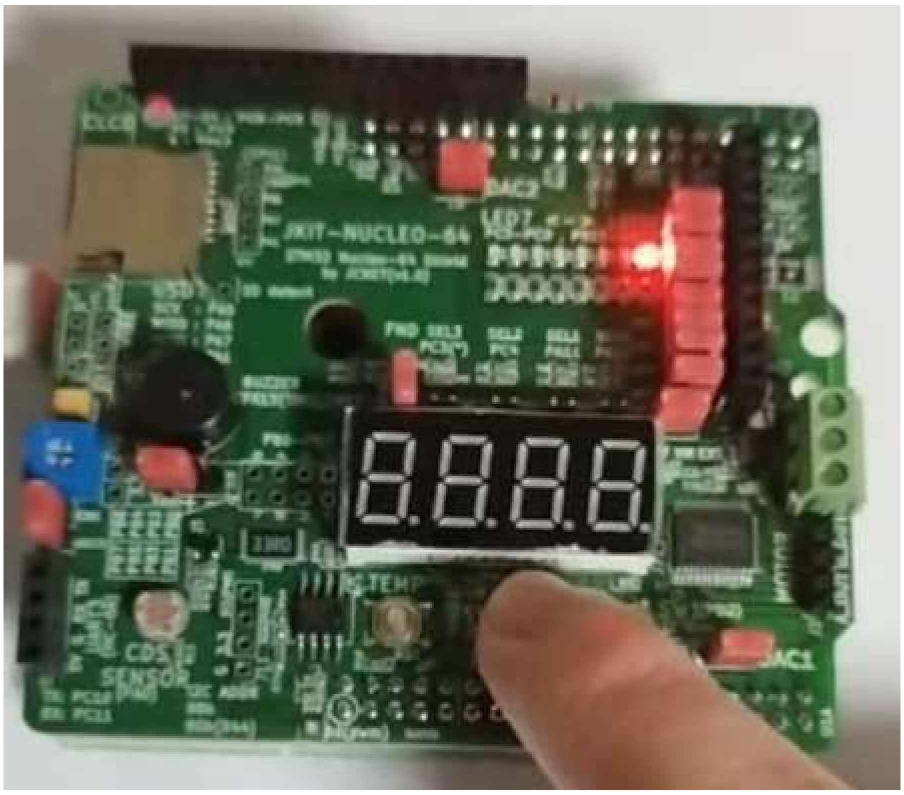


Results

picture 3. No button no output

picture 4. yes button led toggle(on) output

picture 5. yes button led toggle(off) output


picture 6. yes button led toggle(on) output

https://youtube.com/shorts/UrGJzFBJATw?feature=share

Discussion

- By applying a software debounce (counter-based) to the pushbutton, I ensured that the LED toggled exactly once per press, eliminating unwanted multiple triggers ("bouncing").
- Without proper debouncing, I observed frequent double or triple toggling due to mechanical bounce in the pushbutton.
- Through experimentation, I noticed that response delay and sensitivity depend on the MCU loop speed and debounce threshold; tuning these values was important for a good user experience.
- I gained practical insight into edge detection (tracking state transitions from released to pressed) and the importance of waiting for full release before accepting the next toggle event.
- Building the toggle switch logic deepened my appreciation for the same underlying problem and solutions used in real-world devices like keyboards and consumer electronics.

Problem 4: Toggle multiple LEDs with a button

Procedure

1. Create a new project under the directory \repos\EC\lab\

- The project name is "**LAB_GPIO_DIO_multiLED".**
- Name the source file as "**LAB_GPIO_DIO_multiLED.c"**

You MUST write your name in the top of the source file, inside the comment section.

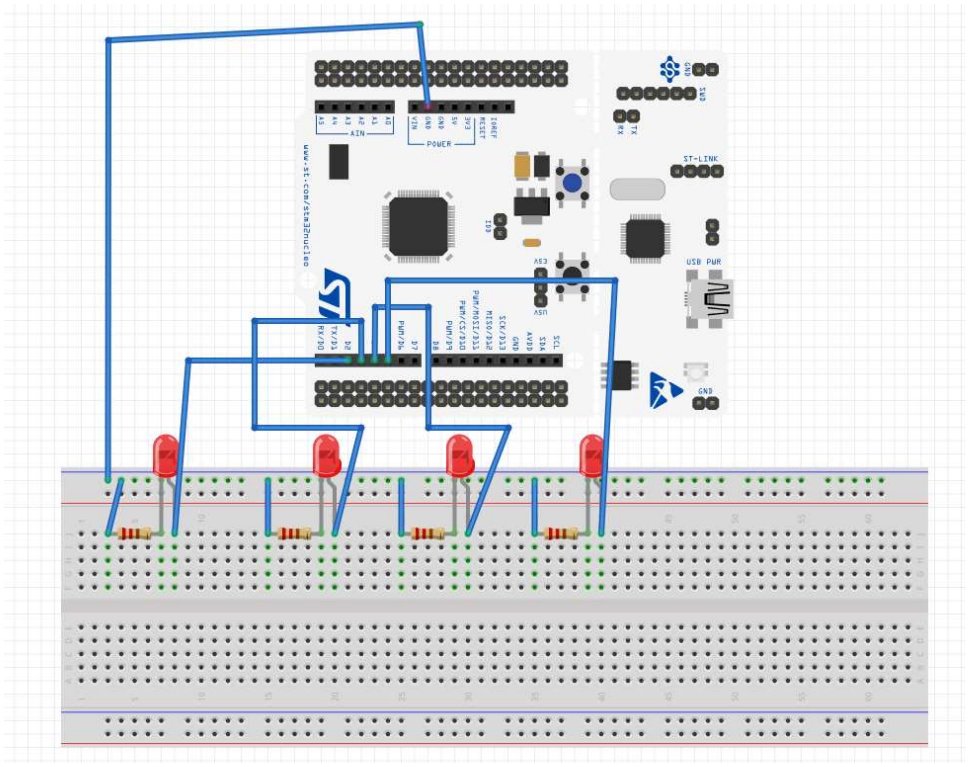
1. Include your rary **ecGPIO2.h, ecGPIO2.c** in \repos\include\.
2. Connect 4 LEDs **externally** with necessary load resistors.

- As Button B1 is Pressed, light one LED at a time, in sequence.
- Example: LED0--> LED1--> …LED3--> …LED0….

Configuration

Button	LED
Digital In	Digital Out
GPIOA, Pin 4	PB12,PB13,PB14,PB15
PULL-UP	Push-Pull, Pull-up, Medium Speed

Circuit Diagram

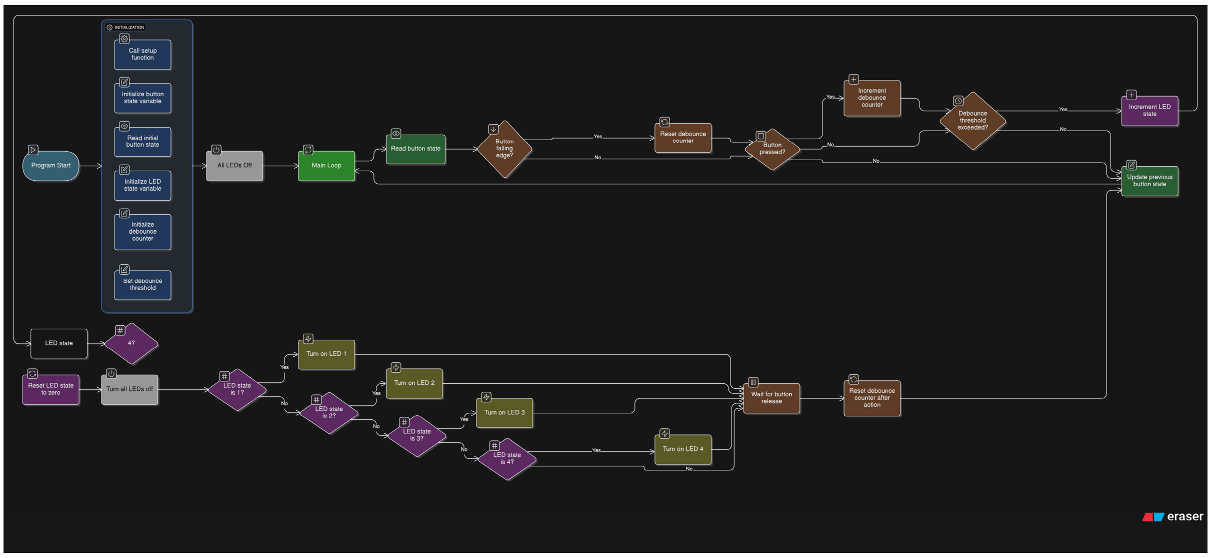


Code

```
#include "ecRCC2.h"
#include "ecGPIO2.h"

#define LED_PIN      PB_12
#define LED_PIN2     PB_13
#define LED_PIN3     PB_14
#define LED_PIN4     PB_15
#define BUTTON_PIN   PA_4

void setup(void) {
    RCC_HSI_init();
    GPIO_init(BUTTON_PIN, INPUT);
    GPIO_init(LED_PIN, OUTPUT);
    GPIO_init(LED_PIN2, OUTPUT);
    GPIO_init(LED_PIN3, OUTPUT);
    GPIO_init(LED_PIN4, OUTPUT);

    GPIO_write(LED_PIN, LOW);
    GPIO_write(LED_PIN2, LOW);
    GPIO_write(LED_PIN3, LOW);
    GPIO_write(LED_PIN4, LOW);
    GPIO_pupd(BUTTON_PIN, 1); // Pull-up
    GPIO_ospeed(LED_PIN, 1);  // Medium speed
    GPIO_ospeed(LED_PIN2, 1); // Medium speed
    GPIO_ospeed(LED_PIN3, 1); // Medium speed
    GPIO_ospeed(LED_PIN4, 1); // Medium speed
}

int main(void) {
    setup();
    int buttonState = 0;
    int prevButtonState = GPIO_read(BUTTON_PIN);
    int ledState = 0;

    int debounceCounter = 0;
    const int DEBOUNCE_THRESHOLD = 500;

    while(1){
        buttonState = GPIO_read(BUTTON_PIN);

        if (buttonState == 0 && prevButtonState == 1) {
            debounceCounter = 0;
        }

        if (buttonState == 0) {
            debounceCounter++;
        }

        if (debounceCounter > DEBOUNCE_THRESHOLD) {
            ledState++;
            if (ledState > 4) {
                ledState = 0;
            }

            GPIO_write(LED_PIN, LOW);
            GPIO_write(LED_PIN2, LOW);
            GPIO_write(LED_PIN3, LOW);
            GPIO_write(LED_PIN4, LOW);

            if (ledState == 1) {
                GPIO_write(LED_PIN, HIGH);
            } else if (ledState == 2) {
                GPIO_write(LED_PIN2, HIGH);
            } else if (ledState == 3) {
                GPIO_write(LED_PIN3, HIGH);
            } else if (ledState == 4) {
                GPIO_write(LED_PIN4, HIGH);
            }

            while(GPIO_read(BUTTON_PIN) == 0) {}

            debounceCounter = 0;
        }

        prevButtonState = buttonState;
    }
}

```

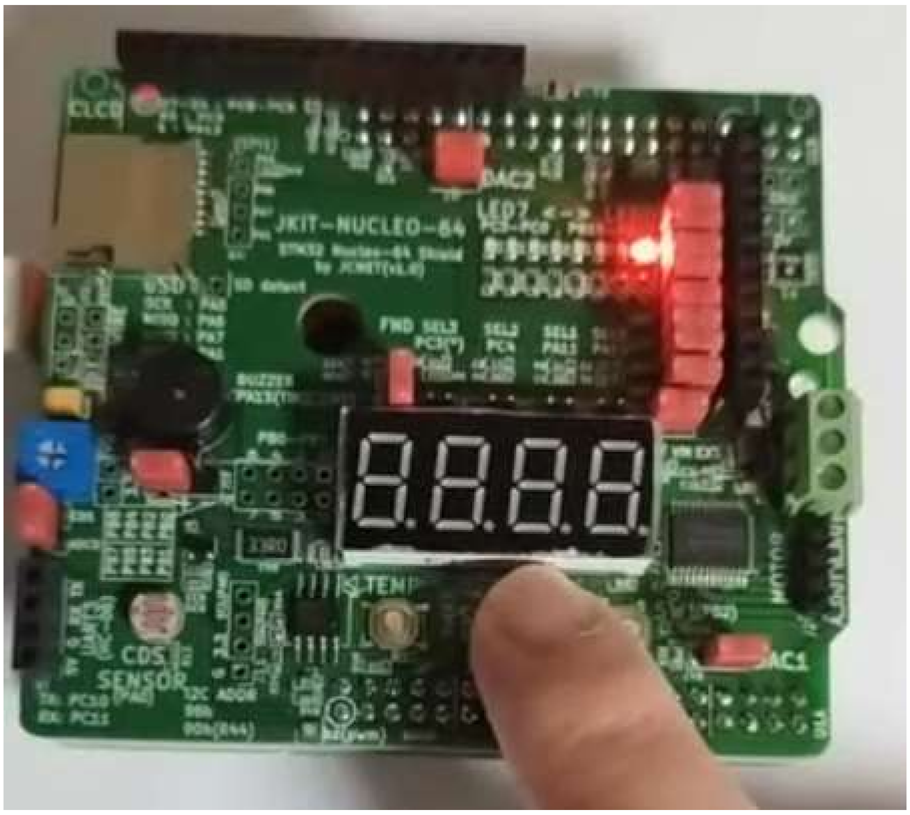
This code will cycle through the states of PB_12→PB_13→PB_14→PB_15→all off, with only one LED lit each time the active-low pushbutton on PA_4 is pressed and judged valid, and is designed to move exactly one step per click through counter-based software debouncing and button release waiting. It also initializes the internal high-speed oscillator (HSI) to the system clock, configures the GPIOs as input/output, and initially sets all LEDs to LOW and off.

Flow chart

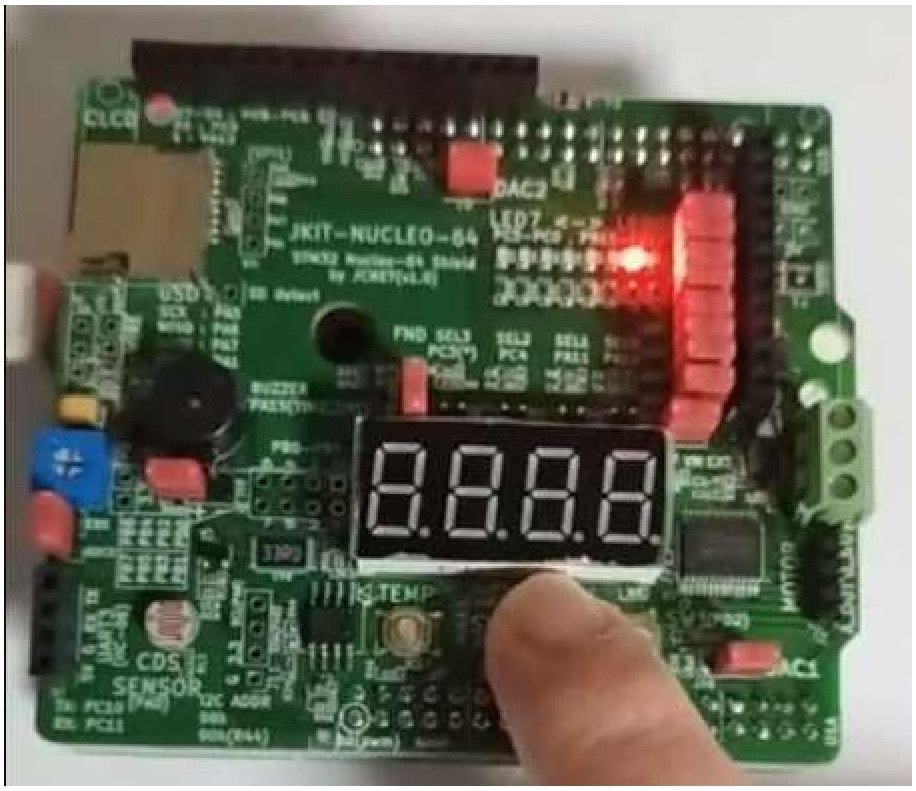


Results

picture 7. no button no output



picture 8. yes button only led1(on) output



picture 9. yes button only led2(on) output

picture 10. yes button only led3(on) output

picture 11. yes button only led4(on) output

picture 12. yes button all led(off) output

https://youtube.com/shorts/5_jPjStjiNE?feature=share

Discussion

- I implemented a simple state machine using a counter to cycle through multiple LEDs (one active at a time) with each button press.
- Potential issues such as repeated fast advances (on rapid clicking or button bounce) were solved by the same software debounce approach and by waiting for the button release.
- The "all off except one" initialization each stage kept the sequence visually clear, and wrapping the counter to zero enabled a cyclic display pattern.
- The experiment helped me realize how similar principles can be used to expand a simple user input—such as a button—into complex controls like keypads or remote controls.
- This hands-on project was effective in illustrating the relationship between physical hardware, software state, and responsive digital outputs.

Reference

https://deepbluembedded.com/stm32-button-debounce-code-examples-tutorial/

https://www.phippselectronics.com/switch-debounce-implementation/

https://deepbluembedded.com/stm32-gpio-leds-buttons-interfacing-driver/