**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.](https://ykkim.gitbook.io/ec/uvision/adding-my-api-header-in-uvision)

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.](https://ykkim.gitbook.io/ec/ec-course/tutorial/tutorial-platformio-in-vscode)

Download necessary library files: [Download library files from here](https://github.com/ykkimhgu/EC-student/tree/main/include/lib-student)

* 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](https://github.com/ykkimhgu/EC-student/tree/main/include/lib-student)
* Then, modify it as ecGPIO2.c, ecGPIO2.h

**ecRCC2.h** (provided)

1. 1. void RCC\_HSI\_init(void);

2. 2. void RCC\_GPIOA\_enable(void);

3. 3. void RCC\_GPIOB\_enable(void);

4. 4. void RCC\_GPIOC\_enable9void);

5. 5.

6.

**ecGPIO2\_student.h**

1. void GPIO\_init(PinName\_t pinName, int mode);

2. void GPIO\_write(PinName\_t pinName, int Output);

3. int GPIO\_read(PinName\_t pinName);

4. void GPIO\_mode(PinName\_t pinName, int mode);

5. void GPIO\_ospeed(PinName\_t pinName, int speed);

6. void GPIO\_otype(PinName\_t pinName, int type);

7. void GPIO\_pupd(PinName\_t pinName, int pupd);

8.

9. /\*

10. // Version 1

11. void GPIO\_init(GPIO\_TypeDef \*Port, int pin, int mode);

12. void GPIO\_write(GPIO\_TypeDef \*Port, int pin, int output);

13. int GPIO\_read(GPIO\_TypeDef \*Port, int pin);

14. void GPIO\_mode(GPIO\_TypeDef\* Port, int pin, int mode);

15. void GPIO\_ospeed(GPIO\_TypeDef\* Port, int pin, int speed);

16. void GPIO\_otype(GPIO\_TypeDef\* Port, int pin, int type);

17. void GPIO\_pupd(GPIO\_TypeDef\* Port, int pin, int pupd);

18.

* Real code in **ecGPIO2\_student.c**

1. /\*----------------------------------------------------------------\

2. @ Embedded Controller by Young-Keun Kim - Handong Global University

3. Author : SSS LAB

4. Created : 05-03-2021

5. Modified : 08-23-2024

6. Language/ver : C++ in Keil uVision

7. name : leejeayong

8. number : 22000561

9.

10. Description : Distributed to Students for LAB\_GPIO

11. /----------------------------------------------------------------\*/

12.

13.

14.

15. #include "stm32f4xx.h"

16. #include "stm32f411xe.h"

17. #include "ecGPIO2.h"

18.

19. void GPIO\_init(PinName\_t pinName, uint32\_t mode){

20. GPIO\_TypeDef \* Port;

21. unsigned int pin;

22. ecPinmap(pinName, &Port, &pin);

23.

24. // mode : Input(0), Output(1), AlterFunc(2), Analog(3)

25. if (Port == GPIOA)

26. RCC\_GPIOA\_enable();

27. if (Port == GPIOB)

28. RCC\_GPIOB\_enable();

29. if (Port == GPIOC)

30. RCC\_GPIOC\_enable();

31.

32.

33.

34. //[TO-DO] YOUR CODE GOES HERE

35. // Make it for GPIOB, GPIOD..GPIOH

36.

37. // You can also make a more general function of

38. // void RCC\_GPIO\_enable(GPIO\_TypeDef \*Port);

39.

40.

41. GPIO\_mode(pinName, mode);

42. }

43. void GPIO\_write(PinName\_t pinName, int Output){

44. GPIO\_TypeDef \* Port;

45. unsigned int pin;

46. ecPinmap(pinName,&Port,&pin);

47. if(Output) Port->BSRR = 1UL << pin; // Set bit

48. else Port->BSRR = 1UL << (pin+16); // Reset bit

49. }

50.

51.

52. // GPIO Mode : Input(00), Output(01), AlterFunc(10), Analog(11)

53. void GPIO\_mode(PinName\_t pinName, uint32\_t mode){

54. GPIO\_TypeDef \* Port;

55. unsigned int pin;

56. ecPinmap(pinName,&Port,&pin);

57. Port->MODER &= ~(3UL<<(2\*pin));

58. Port->MODER |= mode<<(2\*pin);

59.

60. }

61.

62.

63. // GPIO Speed : Low speed (00), Medium speed (01), Fast speed (10), High speed (11)

64. void GPIO\_ospeed(PinName\_t pinName, int speed){

65. GPIO\_TypeDef \* Port;

66. unsigned int pin;

67. ecPinmap(pinName,&Port,&pin);

68. Port->OSPEEDR &= ~(3UL<<(2\*pin));

69. Port->OSPEEDR |= speed<<(2\*pin);

70. }

71.

72. // GPIO Output Type: Output push-pull (0, reset), Output open drain (1)

73. void GPIO\_otype(PinName\_t pinName, int type){

74. GPIO\_TypeDef \* Port;

75. unsigned int pin;

76. ecPinmap(pinName,&Port,&pin);

77. Port->OTYPER &= ~(1UL<<pin);

78. Port->OTYPER |= type<<pin;

79. }

80.

81. // GPIO Push-Pull : No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)

82. void GPIO\_pupd(PinName\_t pinName, int pupd){

83. GPIO\_TypeDef \* Port;

84. unsigned int pin;

85. ecPinmap(pinName,&Port,&pin);

86. Port->PUPDR &= ~(3UL<<(2\*pin));

87. Port->PUPDR|= pupd <<(2\*pin);

88. }

89.

90. int GPIO\_read(PinName\_t pinName){

91. GPIO\_TypeDef \* Port;

92. unsigned int pin;

93. ecPinmap(pinName,&Port,&pin);

94. return (Port->IDR & (1UL << pin));

95.

96. }

97. \*/

98.

**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](https://github.com/ykkimhgu/EC-student/blob/main/lab/lab-student/LAB_GPIO_DIO_LED_student.c).

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**

텍스트, 스크린샷, 도표, 라인이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**Code**

1. #include "ecRCC2.h"

2. #include "ecGPIO2.h"

3.

4. #define LED\_PIN PB\_12

5. #define BUTTON\_PIN PA\_0

6.

7. // Initialiization

8. void setup(void) {

9. RCC\_HSI\_init();

10. // initialize the pushbutton pin as an input:

11. GPIO\_init(BUTTON\_PIN, INPUT);

12. // initialize the LED pin as an output:

13. GPIO\_init(LED\_PIN, OUTPUT);

14. GPIO\_pupd(BUTTON\_PIN, 1); // Pull-up

15. GPIO\_ospeed(LED\_PIN, 1); // Medium speed

16. }

17.

18. int main(void) {

19. setup();

20. int buttonState=0;

21.

22. while(1){

23. // check if the pushbutton is pressed. Turn LED on/off accordingly:

24. buttonState = GPIO\_read(BUTTON\_PIN);

25. if(buttonState) GPIO\_write(LED\_PIN, LOW);

26. else GPIO\_write(LED\_PIN, HIGH);

27. }

28. }

29.

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**

스크린샷, 회로, 전자제품, 멀티미디어 소프트웨어이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**Results**

텍스트, 전자 공학, 전자제품, 회로 구성요소이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

picture 1. no photosensor input no output

텍스트, 전자제품, 전자 공학, 회로 구성요소이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

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.

1. 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](https://github.com/ykkimhgu/EC-student/blob/main/lab/lab-student/LAB_GPIO_DIO_LED_student.c).

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**

텍스트, 스크린샷, 도표, 평행이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**Code**

1. #include "ecRCC2.h"

2. #include "ecGPIO2.h"

3.

4. #define LED\_PIN      PB\_12

5. #define BUTTON\_PIN   PA\_4

6.

7. void setup(void) {

8.     RCC\_HSI\_init();

9.     GPIO\_init(BUTTON\_PIN, INPUT);

10.     GPIO\_init(LED\_PIN, OUTPUT);

11.     GPIO\_pupd(BUTTON\_PIN, 1); // Pull-up

12.     GPIO\_ospeed(LED\_PIN, 1);  // Medium speed

13. }

14.

15. int main(void) {

16.     setup();

17.     int buttonState = 0;

18.     int prevButtonState = GPIO\_read(BUTTON\_PIN);

19.     int ledState = 0;

20.

21.     int debounceCounter = 0;

22.     const int DEBOUNCE\_THRESHOLD = 500;

23.

24.     while(1){

25.         buttonState = GPIO\_read(BUTTON\_PIN);

26.

27.         if (buttonState == 0 && prevButtonState == 1) {

28.             debounceCounter = 0;

29.         }

30.

31.         if (buttonState == 0) {

32.             debounceCounter++;

33.         }

34.

35.         if (debounceCounter > DEBOUNCE\_THRESHOLD) {

36.             ledState = !ledState;

37.             GPIO\_write(LED\_PIN, ledState);

38.

39.             while(GPIO\_read(BUTTON\_PIN) == 0) {}

40.

41.             debounceCounter = 0;

42.         }

43.

44.         prevButtonState = buttonState;

45.     }

46. }

47.

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**

스크린샷, 멀티미디어 소프트웨어, 그래픽 소프트웨어, 회로이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**Results**

텍스트, 전자 공학, 회로, 전자제품이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

picture 3. No button no output

텍스트, 전자 공학, 전자제품, 회로 구성요소이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

picture 4. yes button led toggle(on) output

텍스트, 회로, 전자 공학, 전자제품이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

picture 5. yes button led toggle(off) output

텍스트, 회로, 전자 공학, 전자제품이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

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**

텍스트, 도표, 스크린샷, 평행이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**Code**

1. #include "ecRCC2.h"

2. #include "ecGPIO2.h"

3.

4. #define LED\_PIN      PB\_12

5. #define LED\_PIN2     PB\_13

6. #define LED\_PIN3     PB\_14

7. #define LED\_PIN4     PB\_15

8. #define BUTTON\_PIN   PA\_4

9.

10. void setup(void) {

11.     RCC\_HSI\_init();

12.     GPIO\_init(BUTTON\_PIN, INPUT);

13.     GPIO\_init(LED\_PIN, OUTPUT);

14.     GPIO\_init(LED\_PIN2, OUTPUT);

15.     GPIO\_init(LED\_PIN3, OUTPUT);

16.     GPIO\_init(LED\_PIN4, OUTPUT);

17.

18.     GPIO\_write(LED\_PIN, LOW);

19.     GPIO\_write(LED\_PIN2, LOW);

20.     GPIO\_write(LED\_PIN3, LOW);

21.     GPIO\_write(LED\_PIN4, LOW);

22.     GPIO\_pupd(BUTTON\_PIN, 1); // Pull-up

23.     GPIO\_ospeed(LED\_PIN, 1);  // Medium speed

24.     GPIO\_ospeed(LED\_PIN2, 1); // Medium speed

25.     GPIO\_ospeed(LED\_PIN3, 1); // Medium speed

26.     GPIO\_ospeed(LED\_PIN4, 1); // Medium speed

27. }

28.

29. int main(void) {

30.     setup();

31.     int buttonState = 0;

32.     int prevButtonState = GPIO\_read(BUTTON\_PIN);

33.     int ledState = 0;

34.

35.     int debounceCounter = 0;

36.     const int DEBOUNCE\_THRESHOLD = 500;

37.

38.     while(1){

39.         buttonState = GPIO\_read(BUTTON\_PIN);

40.

41.         if (buttonState == 0 && prevButtonState == 1) {

42.             debounceCounter = 0;

43.         }

44.

45.         if (buttonState == 0) {

46.             debounceCounter++;

47.         }

48.

49.         if (debounceCounter > DEBOUNCE\_THRESHOLD) {

50.             ledState++;

51.             if (ledState > 4) {

52.                 ledState = 0;

53.             }

54.

55.             GPIO\_write(LED\_PIN, LOW);

56.             GPIO\_write(LED\_PIN2, LOW);

57.             GPIO\_write(LED\_PIN3, LOW);

58.             GPIO\_write(LED\_PIN4, LOW);

59.

60.             if (ledState == 1) {

61.                 GPIO\_write(LED\_PIN, HIGH);

62.             } else if (ledState == 2) {

63.                 GPIO\_write(LED\_PIN2, HIGH);

64.             } else if (ledState == 3) {

65.                 GPIO\_write(LED\_PIN3, HIGH);

66.             } else if (ledState == 4) {

67.                 GPIO\_write(LED\_PIN4, HIGH);

68.             }

69.

70.             while(GPIO\_read(BUTTON\_PIN) == 0) {}

71.

72.             debounceCounter = 0;

73.         }

74.

75.         prevButtonState = buttonState;

76.     }

77. }

78.

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**

스크린샷, 그래픽 소프트웨어, 멀티미디어 소프트웨어, 텍스트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**Results**

텍스트, 회로, 전자제품, 전자 공학이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

picture 7. no button no output

텍스트, 전자제품, 전자 공학, 회로 구성요소이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

picture 8. yes button only led1(on) output

텍스트, 전자제품, 전자 공학, 회로 구성요소이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

picture 9. yes button only led2(on) output

텍스트, 회로, 전자 공학, 전자제품이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

picture 10. yes button only led3(on) output

텍스트, 회로, 전자 공학, 전자제품이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

picture 11. yes button only led4(on) output

텍스트, 회로, 전자 공학, 전자 부품이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

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/>