

REPORT

Title : 7-Segment Display



Submission Date	2021.10.15	Major	Mechanical and control engineering
Subject	Embedded Controller	Professor	Young-Keun Kim
Name	Yeong-Won Song	Student Number	21700375
Partner Name	Jin-su Yu	Partner Number	21700469

Contents

I. Introduction	3
1.1 Purpose	3
1.2 Parts List.....	3
 II. Procedure	 4
2.1 7-Segment.....	4
2.2 Configuration	9
2.3 EC_HAL functions.....	10
2.4 Display 0 to 9 with button input.....	12
 III. Conclusion	 14

Appendix

I. Introduction

1.1 Purpose

In this lab, we intend to create a simple program to control a 7-segment display to show a decimal number (0~9).

1.2 Parts List

Parts	Specification	Quantity
7-Segment	5101ASR	1
Array Register	330[Ω]	1
breadboard	-	1
NUCLEO-F411RE	Arm®(a) Cortex®-M4 with FPU	1
M-F Jumper Wire	-	5
M-M Jumper Wire	-	5

Table 1. Part List






				
NUCLEO-F411RE	7-segment	Array Register	Breadboard	M-F/M-M jumper

Table 2. experiment equipment

II. Procedure

2.1 7-Segment

Review 7-segment Decoder and Display from Digital Logic lecture. Also, you can refer this tutorial. Popular BCD 7-segment decoder chip are 74LS47, CD4511. Here, we are going to make the 7-segment decoder by the MCU programming.

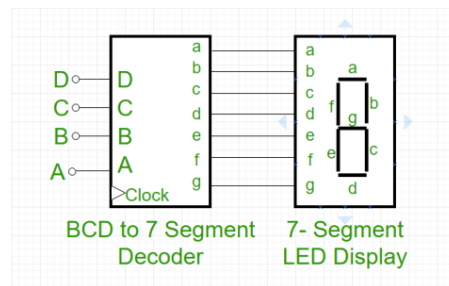


Figure 1. 7-segment Decoder and LED Display

- FLOW CHART

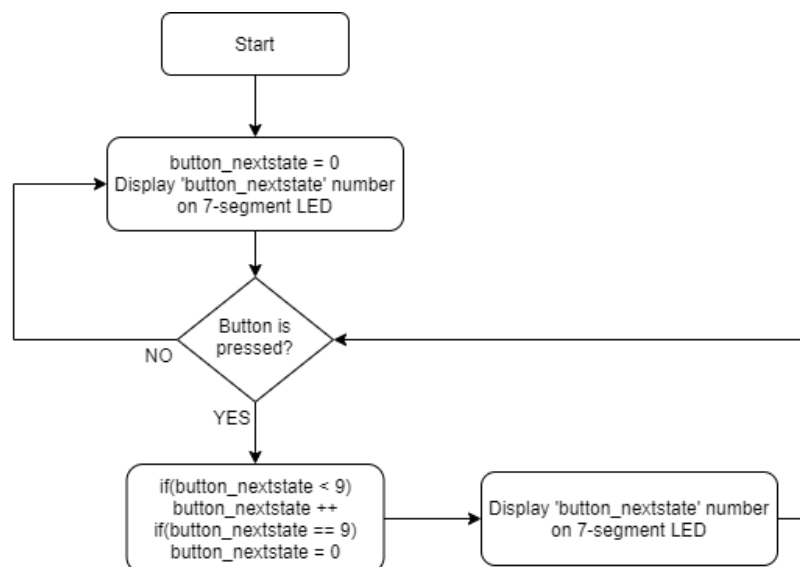


Figure 2. 7-segment Flow chart

- Source Code (main.c)

```

/**
*****
 * @author   Name : Song Yeong Won
 * @Mod      2021-10-15 by YWSONG
 * @brief    Embedded Controller: LAB Digital In/Out
 *           - 7-Segment Display
 *
*****
 */

#include "stm32f4xx.h"
#include "ecRCC.h"
#include "ecGPIO.h"

int main(void) {
    // Initialization -----
    setup();
    sevensegment_init();

    uint32_t current_state = 0; // 0=released ,1=pressed //
    uint32_t prev_state = 1;    // 0=pressed ,1=released // button
    uint32_t button_nextstate = 0; //button pressed state
    uint32_t LED_Control = 0; //button pressed state

    // Infinite Loop -----
    while(1){
        sevensegment_decoder(button_nextstate);
        if(GPIO_read(GPIOC, BUTTON_PIN) == 0){
            prev_state = 0;
            current_state = 1;
            LED_Control = 1;
        }else current_state = 0;

        if(current_state == 0 && prev_state == 0){
            if(LED_Control == 1){
                button_nextstate = ButtonST(button_nextstate); //check button is pressed and go to r
            }
            prev_state = 1;
            LED_Control = 0;
        }
    }
}

```

Figure 3. 7-segment main code

- Discussion

1) Draw the truth table for the BCD 7-Segment decoder.

Input				Output								Number
x_4	x_3	x_2	x_1	A	B	C	D	E	F	G	DP	
0	0	0	0	1	1	1	1	1	1	0	0	0
0	0	0	1	0	1	1	0	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	0	2
0	0	1	1	1	1	1	1	0	0	1	0	3
0	1	0	0	0	1	1	0	0	1	1	0	4
0	1	0	1	1	0	1	1	0	1	1	0	5
0	1	1	0	1	0	1	1	1	1	1	0	6
0	1	1	1	1	1	1	0	0	1	0	0	7
1	0	0	0	1	1	1	1	1	1	1	0	8
1	0	0	1	1	1	1	1	0	1	1	0	9
X	X	X	X	X	X	X	X	X	X	X	X	Invalid

Table 3. BCD 7-segment decoder truth table

Since it is a common-anode type, the LED is turned on when the output value is 0 and off when it is 1. Table 7 represent the common-cathode mode truth table.

2) What are the common cathode and common anode of 7-segment?

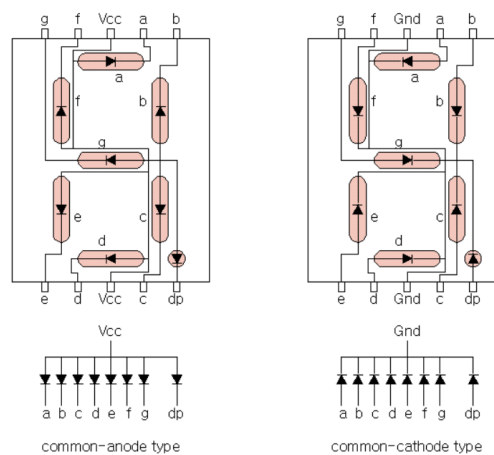


Figure 4. 7-segment types

The 7-segment can be divided into a Common-Anode type and a Common-Cathode type. 7-segment is an assembly of several LEDs. Eight LEDs must be controlled

including seven LEDs to display numbers and DP LEDs. To control eight LEDs, a total of 16 legs must be used, with eight positive legs and eight negative legs. 7-segment used common-anode type and common-cathode type to reduce the number of legs.

The Common-Anode type binds the positive legs of the LED. This method turned on or off the LED by connecting the positive legs to the Vcc and connecting the Vcc or GND to the negative legs of each LED.

The Common-cathode type binds the negative legs of the LED. This method turned on or off the LED by connecting the negative legs to the GND and connecting the Vcc or GND to the positive legs of each LED.

3) This is common anode 7-segment. Dose the LED turn on when output pin from MCU is 'High'?

The 7-segment used in this experiment is the Common-Anode type. The positive of the LED is commonly connected to the Vcc. If the HIGH value is given in software coding, there is no voltage drop between LEDs, so the LED does not turn on because there is no current flow. On the other hand, giving the LOW value causes a voltage drop between 5V and 0V, so the current flows and the LED turns on. By applying this, in coding, LOW values were given to LEDs that wanted to be turned on, and HIGH values were given to LEDs that wanted to be turned off. Therefore, LED turn off when output pin from MCU is 'HIGH'.

4) Find out how to connect a 7-segment to MCU pins with current limiting resistors

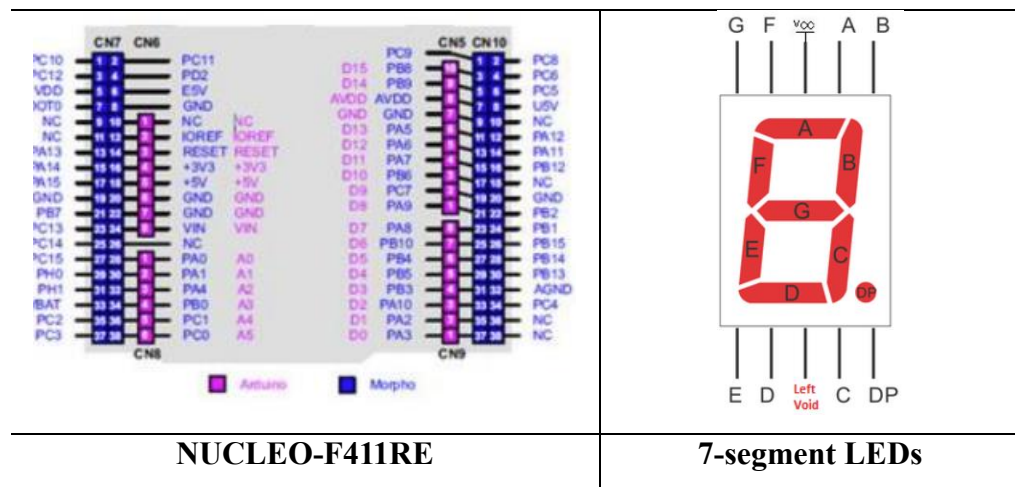


Figure 5. NUCLEO-F411RE and 7-segment LED

We must control a total of 8 LEDs of 7-segment. Digital Out pins are PA5, PA6, PA7, PB6, PC7, PA9, PB8, PB10. PA5 to PB10 pins sequentially connected from 7-segment LED A to DP. At this time, if a voltage is directly applied to the LED, overcurrent may flow and the device may be burned, so the current is limited through the array register.

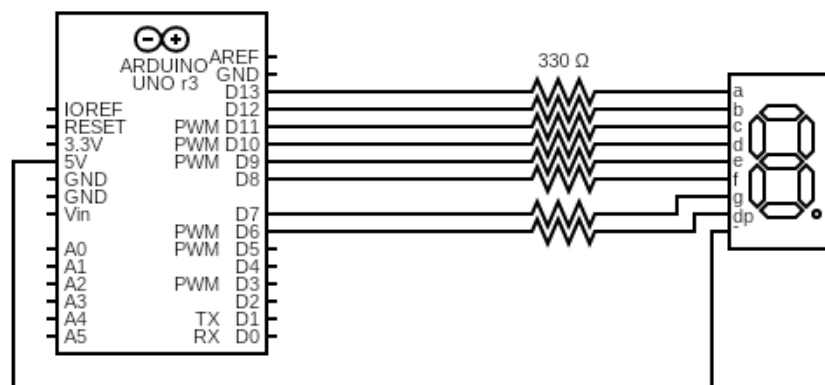


Figure 6. External Circuit Diagram

2.2 Configuration

Create a new project named as “LAB_GPIO_7segment”.

Name the source file as “LAB_GPIO_7segment.cpp”

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

Configuration Input and Output pins

Digital In: Button	Digital Out: LED
GPIOC, Pin 13	PA5, PA6, PA7, PB6, PC7, PA9, PA8, PB10
Digital Input	Digital Output
Set PULL-UP	Push-Pull
	No Pull-up Pull-down
	Fast

Table 4. Configuration Input and Output

Fill in the table

Port/Pin	Description	Register Setting
Port A Pin 5	Clear Pin5 mode	$\text{GPIOA} \rightarrow \text{MODER} \&= \sim(3 \ll (5*2))$
Port A Pin 5	Set Pin5 mode = Output	$\text{GPIOA} \rightarrow \text{MODER} = 1 \ll (5*2)$
Port A Pin 6	Clear Pin6 mode	$\text{GPIOA} \rightarrow \text{MODER} \&= \sim(3 \ll (6*2))$
Port A Pin 6	Set Pin6 mode = Output	$\text{GPIOA} \rightarrow \text{MODER} = 1 \ll (6*2)$
Port A Pin Y	Clear Pin Y mode	$\text{GPIOA} \rightarrow \text{MODER} \&= \sim(3 \ll (Y*2))$
Port A Pin Y	Set Pin Y mode = Output	$\text{GPIOA} \rightarrow \text{MODER} = 1 \ll (Y*2)$
Port A Pin 5~9	Clear Pin 5~9 mode	$\text{GPIOA} \rightarrow \text{MODER} \&= \sim(0x3FF \ll (5*2))$
	Set Pin 5~9 mode = Output	$\text{GPIOA} \rightarrow \text{MODER} = 0x155 \ll (5*2)$
Port X Pin Y	Clear Pin Y mode	$\text{GPIOX} \rightarrow \text{MODER} \&= \sim(3 \ll (Y*2))$
	Set Pin Y mode = Output	$\text{GPIOX} \rightarrow \text{MODER} = 1 \ll (Y*2)$
Port A Pin 5	Set Pin 5 otype = push pull	$\text{GPIOA} \rightarrow \text{OTYPER} \&= \sim(1 \ll 5)$
Port A Pin Y	Set Pin Y otype = push pull	$\text{GPIOA} \rightarrow \text{OTYPER} \&= \sim(1 \ll Y)$
Port A Pin 5	Set Pin 5 ospeed = Fast	$\text{GPIOA} \rightarrow \text{OSPEED} \&= \sim(3 \ll (5*2))$
		$\text{GPIOA} \rightarrow \text{OSPEED} = 2 \ll (5*2)$
Port A Pin Y	Set Pin Y ospeed = Fast	$\text{GPIOY} \rightarrow \text{OSPEED} \&= \sim(3 \ll (5*2))$
		$\text{GPIOY} \rightarrow \text{OSPEED} = 2 \ll (Y*2)$
Port A Pin 5	Set Pin 5 PUPD = no pullup/down	$\text{GPIOA} \rightarrow \text{PUPDR} \&= \sim(3 \ll (5*2))$
		$\text{GPIOA} \rightarrow \text{PUPDR} = 0 \ll (5*2)$
Port A Pin Y	Set Pin Y PUPD = no pullup/down	$\text{GPIOY} \rightarrow \text{PUPDR} \&= \sim(3 \ll (Y*2))$
		$\text{GPIOY} \rightarrow \text{PUPDR} = 0 \ll (Y*2)$

Table 5. Register Settings

2.3 EC_HAL functions

Specific for given Output Pins

Include File	Function
epGPIO.h, c	void sevensegment_init(void); void sevensegment_decoder(uint8 num);

ecGPIO.h

```
// Distributed for LAB: GPIO

#include "stm32f4llxe.h"
#include "ecRCC.h"

#ifndef __ECGPIO_H
#define __ECGPIO_H

#define LOW          0
#define HIGH         1

#define RESET 3UL

//MODER
#define INPUT        0
#define OUTPUT       1
#define ALTERNATE    2
#define ANALOG       3

//OSPEED
#define LOW_SPEED    0
#define MEDIUM_SPEED 1
#define FAST_SPEED   2
#define HIGH_SPEED   3

//PUPDR
#define NOPUPD       0
#define PU            1 //pull-up
#define PD            2 //pull-down
#define RESERVED     3

//OTYPE
#define OUTPUT_PP     0
#define OUTPUT_OD     1

//7-Segment LED PINs
#define LED_PA5       5
#define LED_PA6       6
#define LED_PA7       7
#define LED_PB6       6
#define LED_PC7       7
#define LED_PA9       9
#define LED_PA8       8
#define LED_PB10      10
#define BUTTON_PIN    13
```

```

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

void setup(void);
void GPIO_init(GPIO_TypeDef *Port, int pin, uint32_t mode);
void GPIO_mode(GPIO_TypeDef *Port, int pin, uint32_t mode);

void GPIO_write(GPIO_TypeDef *Port, int pin, uint32_t Output);
uint32_t GPIO_read(GPIO_TypeDef *Port, int pin);
void GPIO_ospeed(GPIO_TypeDef *Port, int pin, uint32_t speed);
void GPIO_otype(GPIO_TypeDef *Port, int pin, uint32_t type);
void GPIO_pudr(GPIO_TypeDef *Port, int pin, uint32_t pudr);

uint32_t ButtonST(uint32_t value);
void sevensegment_init(void);
void sevensegment_decoder(uint32_t num);

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif

```

Figure 7. ecGPIO.h code

ecGPIO.c : See Appendix

2.4 Display 0 to 9 with button input

Toggling LED by pushing button.

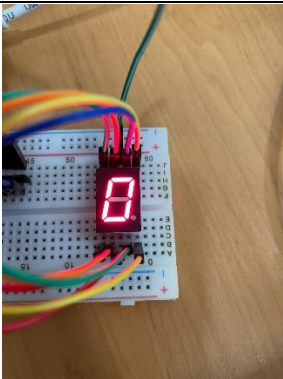
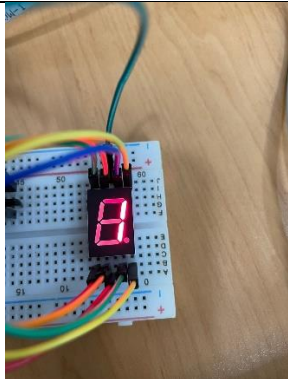
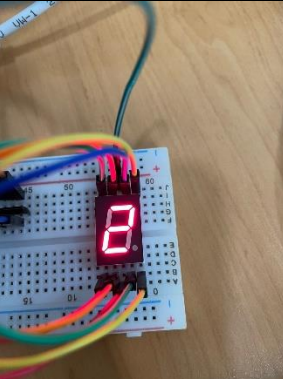
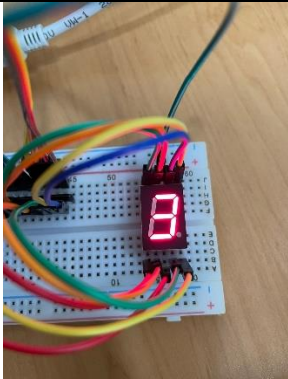
First, check each number can be displayed, properly

Then, create the code such that whenever the button is pushed, the number should display 0 to 9 and back to 0 and so on.

- Result Validity

Button_nextState	7-segment Display
0(initial condition)	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

Table 6. Result check

	
Button_nextState 0	Button_nextState 1
	
Button_nextState 2	Button_nextState 3

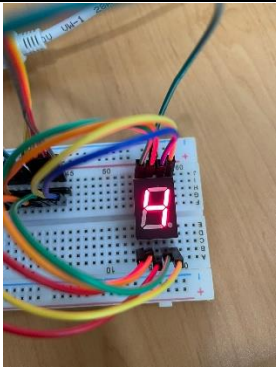
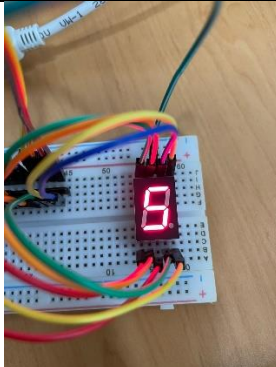
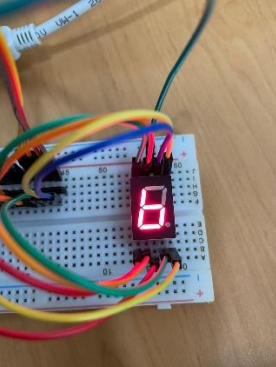
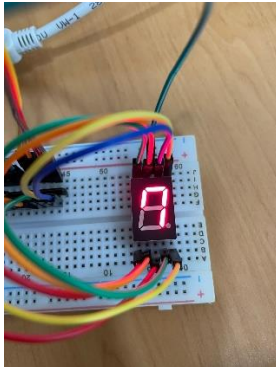
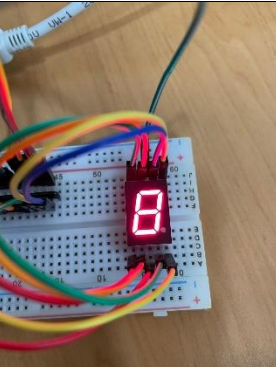
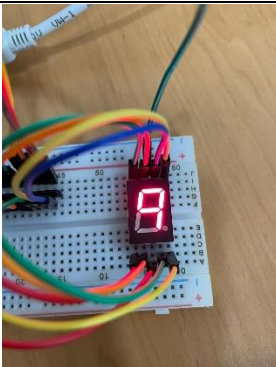
	
Button_nextState 4	Button_nextState 5
	
Button_nextState 6	Button_nextState 7
	
Button_nextState 8	Button_nextState 9

Table 7. 7-segment LED output result

III. Conclusion

Through this lab, we learned how to control the 7-segment LED and the 7-segment decoder to display numbers on 7-segment LED. Based on this, we created 7-Segment_decoder function in programming. We found two types of 7-segment which is common-anode type and common cathode type. Referring to the 7-segment data sheet used in this experiment, common-anode type was selected to work well.

- Troubleshooting

Q. How to simplify the code length of 7-segment decoder?

A. Depending on the GPIOA, B, and C, the LED is turned on differently. Since GPIO A, B, C are activated, the GPIO_write function is used only three times, one each for GPIOA, B, and C. In addition, the number of repeated sentences was reduced by making the Pin Numbers into one 1D-array.

Appendix

- **Video demo Link**

Click [here](#) watch video.

- **Source code**

<ecGPIO.c>

```
// Distributed for LAB: GPIO
#include "stm32f4xx.h"
#include "stm32f4llxe.h"
#include "ecGPIO.h"

void GPIO_init(GPIO_TypeDef *Port, int pin, uint32_t mode){
    if (Port == GPIOA)
        RCC_GPIOA_enable();

    if (Port == GPIOB)
        RCC_GPIOB_enable();

    if (Port == GPIOC)
        RCC_GPIOC_enable();

    GPIO_mode(Port, pin, mode);
}

void GPIO_mode(GPIO_TypeDef *Port, int pin, uint32_t mode){
    Port->MODER &= ~(RESET<<( pin *2));
    Port->MODER |= mode<<(pin *2);
}

void GPIO_write(GPIO_TypeDef *Port, int pin, uint32_t Output){
    if(Output == 0) //Output Low
    {
        Port->ODR &= ~(1<<pin);
    }
    if(Output == 1) //Output High
    {
        Port->ODR |= 1<<pin ;
    }
}

uint32_t GPIO_read(GPIO_TypeDef *Port, int pin){
    return (Port->IDR) & (1<<pin) ;
}

void GPIO_ospeed(GPIO_TypeDef* Port, int pin, uint32_t speed){
    Port->OSPEEDR &= ~(RESET<<( pin *2));
    Port->OSPEEDR |= speed<<(pin *2);
}

void GPIO_otype(GPIO_TypeDef* Port, int pin, uint32_t type){ // 0 : Output push-pull 1: Output open-drain
    if(type == 0) //Output push-pull
    {
        Port->OTYPER &= ~(1<<pin);
    }
    if(type == 1) //Output Open-drain
    {
        Port->OTYPER |= 1<<pin ;
    }
}

void GPIO_pudr(GPIO_TypeDef* Port, int pin, uint32_t pudr){
    Port->PUPDR &= ~(RESET<<( pin *2));
    Port->PUPDR |= pudr<<(pin *2);
}
```

```

void sevensegment_decoder(uint32_t num){

    int PIN[8]={LED_PA5,LED_PA6,LED_PA7,LED_PB6,LED_PC7,LED_PA9,LED_PA8,LED_PB10};
    //LED ON : 0 , LED OFF : 1
    uint32_t number[10][8]={
        {0,0,0,0,0,0,1,1},          //zero
        {1,0,0,1,1,1,1,1},          //one
        {0,0,1,0,0,1,0,1},          //two
        {0,0,0,0,1,1,0,1},          //three
        {1,0,0,1,1,0,0,1},          //four
        {0,1,0,0,1,0,0,1},          //five
        {1,1,0,0,0,0,0,1},          //six
        {0,0,0,1,1,0,1,1},          //seven
        {0,0,0,0,0,0,0,1},          //eight
        {0,0,0,1,1,0,0,1},          //nine
    };

    for (int i=0; i<8; i++){
        if(i==0 || i==1 || i==2 || i==5 || i==6){
            GPIO_write(GPIOA, PIN[i], number[num][i]);
        }
        if(i==3 || i==7){
            GPIO_write(GPIOB, PIN[i], number[num][i]);
        }
        if(i==4){
            GPIO_write(GPIOC, PIN[i], number[num][i]);
        }
    }
}

uint32_t ButtonSI(uint32_t value){
    uint32_t next =0;
    if(value ==0) next =1;
    if(value ==1) next =2;
    if(value ==2) next =3;
    if(value ==3) next =4;
    if(value ==4) next =5;
    if(value ==5) next =6;
    if(value ==6) next =7;
    if(value ==7) next =8;
    if(value ==8) next =9;
    if(value ==9) next =0;
    return next;
}

```

Figure 8. ecGPIO.c code

- Documentation

sevensegment_init()

Initializes 7-segment GPIO pins with default setting and Enables GPIO Clock.

```
void sevensegment_init(void);
```

Parameters

- void

Example code

```
sevensegment_init();
```


sevensegment_decoder()

Control each LED to display a number on the 7-segment LED

```
void sevensegment_decoder(uint32_t num);
```

Parameters

- **int:** unsigned integer number (7-segment can represent a number 0 to 9)

Example code

```
sevensegment_decoder(0); //Display number '0' on 7-segment
```

Figure 9. Documentation of 7-segment function

- Reference

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..E and H)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	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]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits (y = 0..15)

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

8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..E and H)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)

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

8.4.3 GPIO port output speed register (GPIOx_OSPEEDR) (x = A..E and H)

Address offset: 0x08

Reset values:

- 0x0C00 0000 for port A
- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bits 2y:2y+1 **OSPEEDRy[1:0]**: Port x configuration bits (y = 0..15)

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

- 00: Low speed
- 01: Medium speed
- 10: Fast speed
- 11: High speed

Note: Refer to the product datasheets for the values of OSPEEDRy bits versus V_{DD} range and external load.

8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..E and H)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	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]	
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bits 2y:2y+1 **PUPDRy[1:0]**: Port x configuration bits (y = 0..15)

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

Bits 2y:2y+1 **PUPDRy[1:0]**: Port x configuration bits (y = 0..15)

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

Figure 10. STM32F411xC/E Reference Manual