LAB: EXTI & SysTick

Date: 2023-10-13

Author: Gyeonheal An

Github: https://github.com/AnGyeonheal/Embedded_Control_GH

Demo Video: https://youtube.com/shorts/owh8RJOokzI?feature=shared

I. Introduction

In this lab, you are required to create two simple programs using interrupt:

(1) displaying the number counting from 0 to 9 with Button Press

(2) counting at a rate of 1 second

i. Requirement

Hardware

- MCU
  - NUCLEO-F411RE
- Actuator/Sensor/Others:
  - 4 LEDs and load resistance
  - 7-segment display(5101ASR)
  - Array resistor (330 ohm)
  - breadboard

Software

- Keil uVision, CMSIS, EC_HAL library
- Clion(with PlatformIO core plugin)
  Library: STM32Cube library package(Official), EC_HAL library
  Compiler: GNU Arm Embedded Toolchain
  Debugger: ST-Link

II. Problem 1: Counting numbers on 7-Segment using EXTI Button

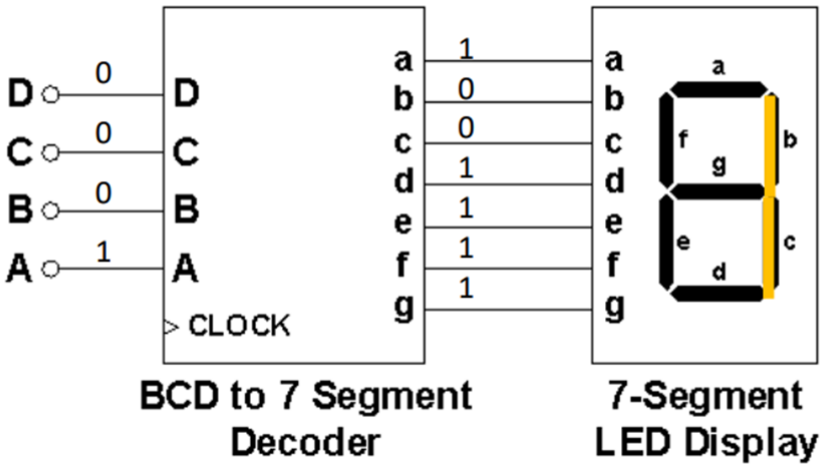
i. Create HAL library

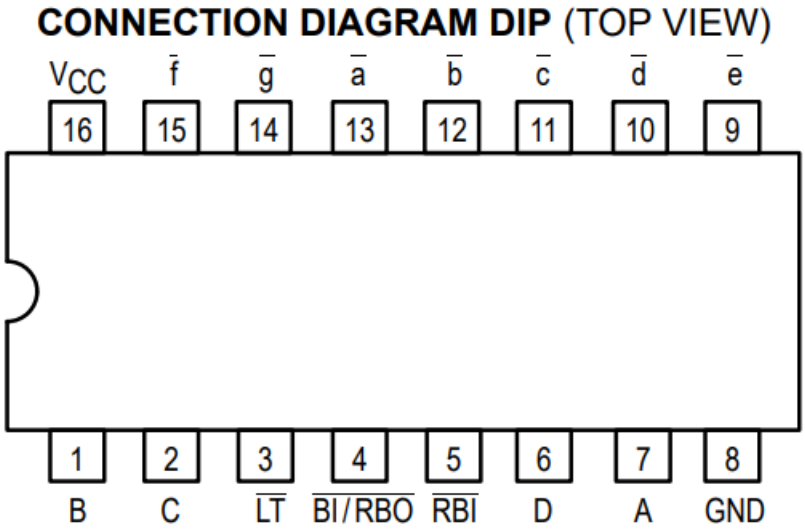
ecEXTI.h

```
void EXTI_init(GPIO_TypeDef *port, int pin, int trig_type, int priority);
void EXTI_enable(uint32_t pin);  // mask in IMR
void EXTI_disable(uint32_t pin);  // unmask in IMR
uint32_t  is_pending_EXTI(uint32_t pin);
void clear_pending_EXTI(uint32_t pin);
```

ii. Procedure

1. Use the decoder chip (**74LS47**). Connect it to the bread board and 7-segment display.



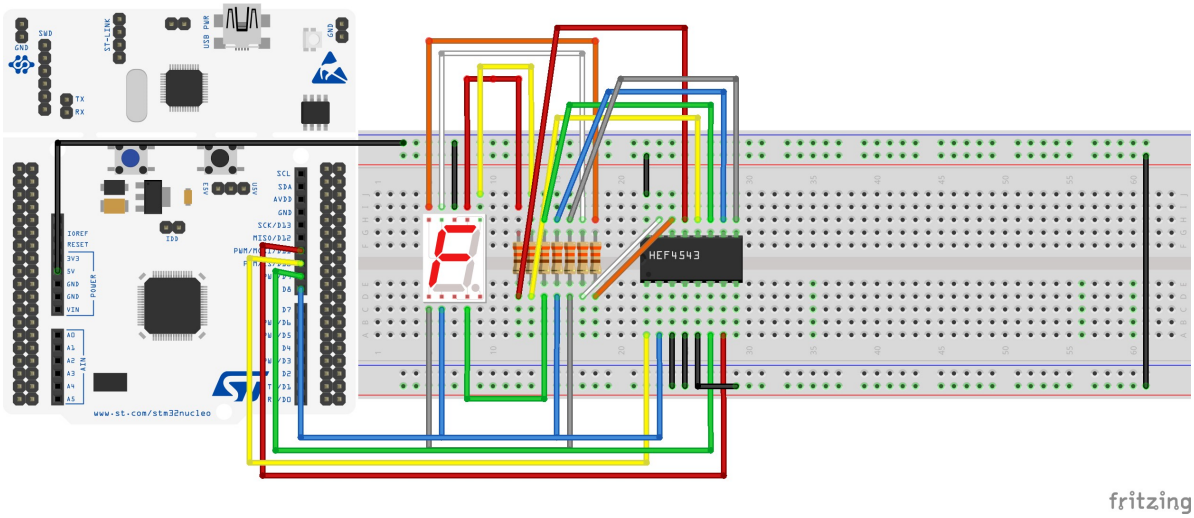


1. First, check if every number, 0 to 9, can be displayed properly on the 7-segment.
2. Then, create a code to display the number counting from 0 to 9 and repeats
    - by pressing the push button. (External Interrupt)
3. You must use your library function of EXTI.
4. Refer to an sample code

iii. Configuration

Digital In for Button (B1)	Digital Out for 7-Segment decoder
Digital In	Digital Out
PC13	PA7, PB6, PC7, PA9
PULL-UP	Push-Pull, No Pull-up-Pull-down, Medium Speed

iv. Circuit Diagram



v. Discussion

1. **We can use two different methods to detect an external signal: polling and interrupt. What are the advantages and disadvantages of each approach?**

   Polling is a technique where the system continually checks for external inputs at regular intervals. In contrast, the Interrupt mechanism triggers an event recognized by the CPU when an external input is detected, and it invokes the Interrupt Service Routine (ISR) through an Interrupt Handler.

   The Polling method is straightforward to implement, allowing for periodic event checking and processing, ensuring events are examined at fixed time intervals. However, Polling consumes CPU resources, potentially leading to slower response times as the CPU checks for events before they occur. It can also disrupt other tasks in a multitasking environment.

   The Interrupt mechanism offers faster response times compared to Polling and allows the CPU to perform other tasks while waiting for an event, optimizing resource utilization. However, it requires hardware logic and, therefore, is more complex to implement. Additionally, it involves managing priorities and processing methods, necessitating precise implementation.

2. **What would happen if the EXTI interrupt handler does not clear the interrupt pending flag? Check with your code**

   1. Repeated Interrupts: If the interrupt pending flag is not cleared within the interrupt handler, the handler will keep getting called repeatedly as long as the triggering condition is met. This can lead to excessive CPU utilization, as the handler never exits, effectively locking up the system.
   2. Unpredictable Behavior: Leaving the interrupt pending flag uncleared can lead to unpredictable behavior within the microcontroller, as multiple instances of the handler may overlap and interfere with each other. This can cause data corruption, unstable system operation, and make debugging and maintenance much more challenging.
   3. Loss of Future Interrupts: If a new interrupt event occurs while the interrupt handler is still executing, and the previous interrupt pending flag is not cleared, the new event may be missed because the system is still servicing the previous interrupt. This can result in data loss or missed important events.

vi. Code

1. **LAB_EXTI.c**

- **main()**

```
#define LED_PIN 5
#define BUTTON_PIN 13
#define MCU_CLK_PLL 84000000
#define MCU_CLK_HSI 16000000

unsigned char state = S0;
unsigned  char next_state = S0;
unsigned int input = 1;
unsigned int delay = 0;

int main(void) {
    setup();
    while (1) {
    }
}
```

This main function initializes setting and maintains loop function that execute infinitely.

- **setup()**

```
void setup(void)
{
    RCC_PLL_init();
    SysTick_init();
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);
    // Priority Highest(0) External Interrupt
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);
    sevensegment_display_init();
}
```

This function initializes CLK, EXTI and also GPIO pins (Button pin and 7-segment pins)

- **External Interrupt**

```
//EXTI for Pin 13
void EXTI15_10_IRQHandler(void) {

    if (is_pending_EXTI(BUTTON_PIN) == 1) {
        while(1){
            delay++;
            if(delay > 3000000) break;
        }
        sevensegment_switch();
        clear_pending_EXTI(BUTTON_PIN);
        delay = 0;
    }
}
```

This EXTI function is a function that recognizes signals input from the outside and executes commands set by the user. The corresponding function checks whether there is a ButtonPin in the queue at is_pending_EXTI() and executes the reservement_switch function if the return value is true. And finally, clear_pending_EXTI() removes it from the queue and enables the next input.

At this time, the delay variable plays a role in generating delay in the software to minimize noise occurring in the Button Pin.

- **Up Counting 7-Segment**

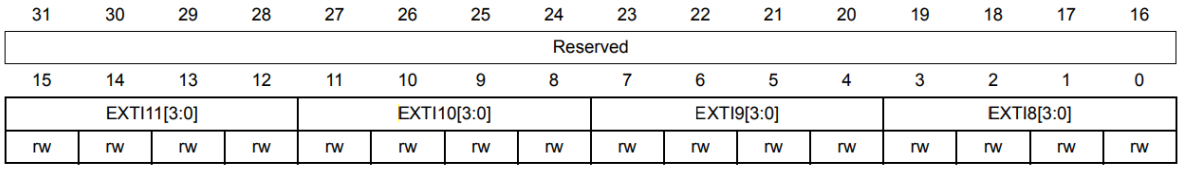
```
void sevensegment_switch(void){
    input = 0;
    next_state = FSM[state].next[input];
    state = next_state;

    sevensegment_display(state);
    input = 1;
}
```

This function is a function that runs when an input is received from the above EXTI_Handler function, and updates the next current state in the FSM method, saves the next state, and executes it.

2. **EXTI.c**

```
void EXTI_init(GPIO_TypeDef *Port, int Pin, int trig_type,int priority){

    // SYSCFG peripheral clock enable
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    // Connect External Line to the GPIO
    int EXTICR_port;
    if      (Port == GPIOA) EXTICR_port = 0;
    else if (Port == GPIOB) EXTICR_port = 1;
    else if (Port == GPIOC) EXTICR_port = 2;
    else if (Port == GPIOD) EXTICR_port = 3;
    else                    EXTICR_port = 4;

    SYSCFG->EXTICR[Pin/4] &= ~(15UL << (Pin%4*4));
// clear 4 bits
    SYSCFG->EXTICR[Pin/4] |= (EXTICR_port << (Pin%4*4));
// set 4 bits

    // Configure Trigger edge
    if (trig_type == FALL) EXTI->FTSR |= 1UL<<Pin;   // Falling trigger enable
    else if (trig_type == RISE) EXTI->RTSR |= 1<<Pin;   // Rising trigger enable
    else if (trig_type == BOTH) {           // Both falling/rising trigger
enable
        EXTI->RTSR |= 1<<Pin;
        EXTI->FTSR |= 1<<Pin;
    }

    // Configure Interrupt Mask (Interrupt enabled)
    EXTI->IMR  |= 1 << Pin;     // not masked


    // NVIC(IRQ) Setting
```

```
    int EXTI_IRQn = 0;

    if (Pin < 5)    EXTI_IRQn = 6 + Pin;
    else if (Pin < 10)  EXTI_IRQn = EXTI9_5_IRQn;
    else            EXTI_IRQn = EXTI15_10_IRQn;

    NVIC_SetPriority(EXTI_IRQn, priority);  // EXTI priority
    NVIC_EnableIRQ(EXTI_IRQn);  // EXTI IRQ enable
}
```



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI11[3:0]				EXTI10[3:0]				EXTI9[3:0]				EXTI8[3:0]			
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 **EXTIx[3:0]**: EXTI x configuration (x = 8 to 11)

These bits are written by software to select the source input for the EXTIx external interrupt.

0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: Reserved
0110: Reserved
0111: PH[x] pin

This function is responsible for overall EXTI setting. It enables SYSCFG peripheral clock, connect the corresponding external line to GPIO, configure the trigger edge and interrupt mask and enables external interrupt.

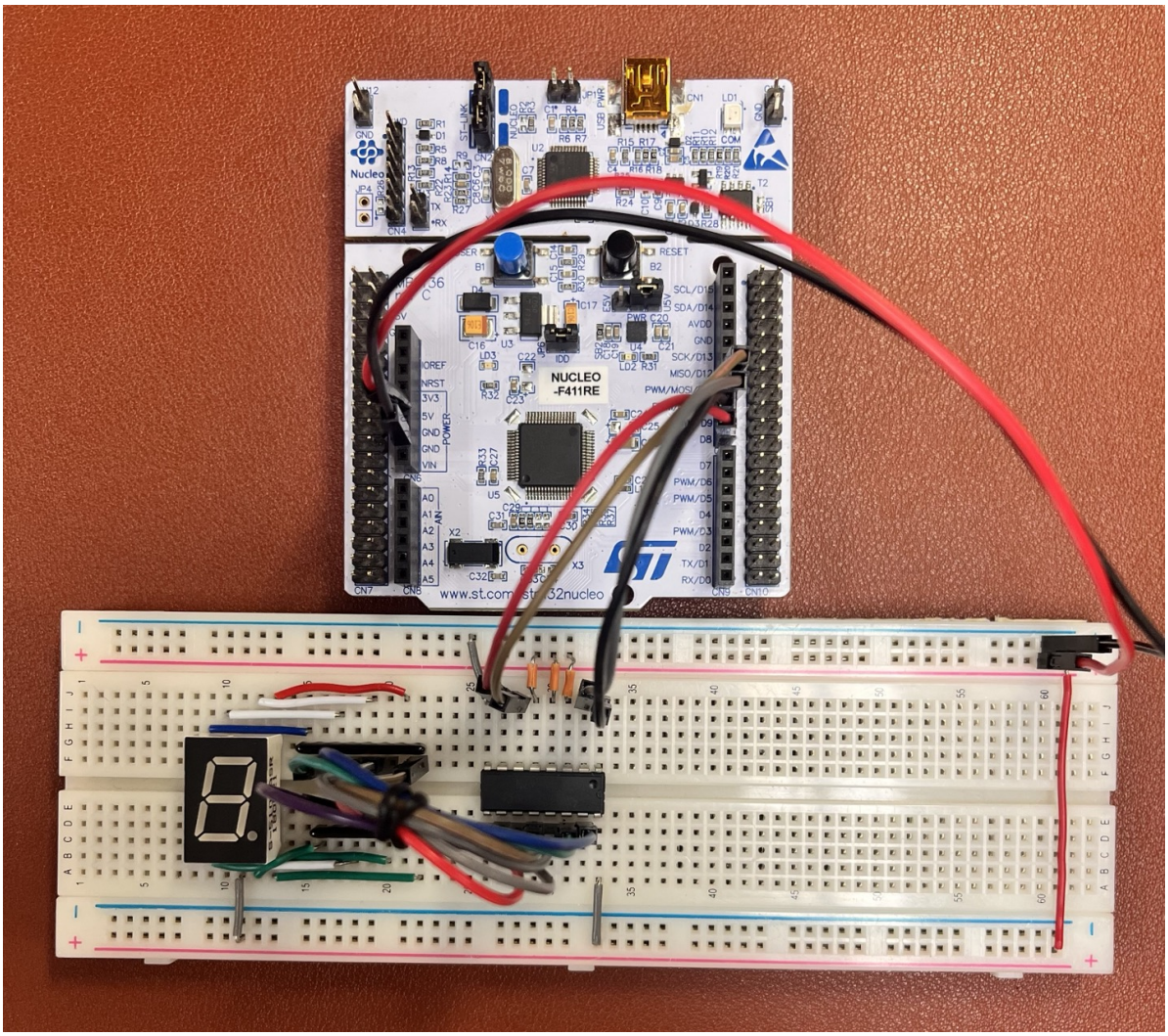
```
void EXTI_enable(uint32_t pin) {
    EXTI->IMR |= 1UL << pin;     // not masked (i.e., Interrupt enabled)
}

void EXTI_disable(uint32_t pin) {
    EXTI->IMR |= 0UL<<pin;     // masked (i.e., Interrupt disabled)
}
```

These functions enables or disables the EXTI.

```
uint32_t is_pending_EXTI(uint32_t pin){
    uint32_t EXTI_PRx = 1UL << pin;         // check  EXTI pending
    return ((EXTI->PR & EXTI_PRx) == EXTI_PRx);
}


void clear_pending_EXTI(uint32_t pin){
    EXTI->PR |= 0UL << pin;     // clear EXTI pending
}
```

Function above checks if the EXTI is on pending or not, and below function clears EXTI pending.

vii. Results



demo video link

III. Problem 2: Counting numbers on 7-Segment using SysTick

Display the number 0 to 9 on the 7-segment LED at the rate of 1 sec. After displaying up to 9, then it should display '0' and continue counting.

When the button is pressed, the number should be reset '0' and start counting again.

i. Create HAL library

ecSysTick.h

```
void SysTick_init(uint32_t msec);
void delay_ms(uint32_t msec);
uint32_t SysTick_val(void);
void SysTick_reset (void);
void SysTick_enable(void);
void SysTick_disable (void);
```

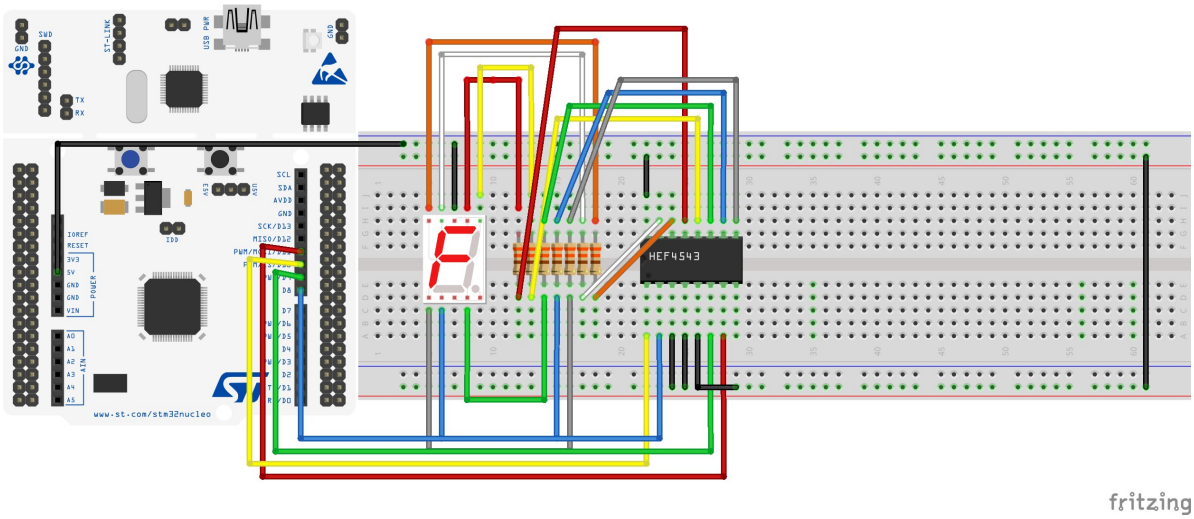
ii. Procedure

1. Use the decoder chip (**74LS47**). Connect it to the bread board and 7-segment display.
2. First, check if every number, 0 to 9, can be displayed properly on the 7-segment.
3. Then, create a code to display the number counting from 0 to 9 and repeats at the rate of 1 second.
4. When the button is pressed, it should start from '0' again.

iii. Configuration

Digital In for Button (B1)	Digital Out for 7-Segment decoder
Digital In	Digital Out
PC13	PA7, PB6, PC7, PA9
PULL-UP	Push-Pull, No Pull-up-Pull-down, Medium Speed

iv. Circuit Diagram



v. Code

```
int main(void) {
    // Initialiization ---------------------------------------------------------
    setup();

    // Inifinite Loop ----------------------------------------------------------
    while(1){
        sevensegment_display(count);
        delay_ms(1000);
        count++;
        if (count >9) count =0;
        SysTick_reset();
    }
}
```

In this main code, it uses SysTick counting 1 seconds.

Count up every second, and when the number reaches 9, it comes back to zero and repeats

```
void setup(void)
{
    RCC_PLL_init();
    SysTick_init();
    sevensegment_display_init();
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
    GPIO_pupd(GPIOC, BUTTON_PIN, EC_PU);
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);
}
```

This function initializes CLK, EXTI, SysTick and also GPIO pins (Button pin and 7-segment pins)

```
void EXTI15_10_IRQHandler(void) {
    if (is_pending_EXTI(BUTTON_PIN) == 1) {
        count = 9;
        clear_pending_EXTI(BUTTON_PIN);
    }
}
```

When the external interrupt (Button) is recognized, it comes back to zero (count = 9) and clear pending to prepare the next external interrupt input.

```
void SysTick_init(void){
    //  SysTick Control and Status Register
    SysTick->CTRL = 0;                                          // Disable
SysTick IRQ and SysTick Counter

    // Select processor clock
    // 1 = processor clock;  0 = external clock
    SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk;

    // uint32_t MCU_CLK=EC_SYSTEM_CLK
    // SysTick Reload Value Register
    SysTick->LOAD = MCU_CLK_PLL / 1000 - 1;                     // 1ms, for HSI
PLL = 84MHz.

    // SysTick Current Value Register
    SysTick->VAL = 0;

    // Enables SysTick exception request
    // 1 = counting down to zero asserts the SysTick exception request
    SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;

    // Enable SysTick IRQ and SysTick Timer
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;

    NVIC_SetPriority(SysTick_IRQn, 16);     // Set Priority to 1
    NVIC_EnableIRQ(SysTick_IRQn);           // Enable interrupt in NVIC
}
```

This function initializes SysTick setting.

Firstly, It disable SysTick IRQn and SysTick Counter.

And it selects processor clock, sets reload value, clear current value, and enables SysTick clock and IRQn.

```
void SysTick_Handler(void){
    SysTick_counter();
}

void SysTick_counter(){
    msTicks++;
}
```

This function continues to be called periodically in the delay_ms function below and counts.

```
void delay_ms (uint32_t mesc){
  uint32_t curTicks;

  curTicks = msTicks;
  while ((msTicks - curTicks) < mesc);

    msTicks = 0;
}
```

This function makes the counting up number zero again when the counting up number reaches a certain time set by the user.

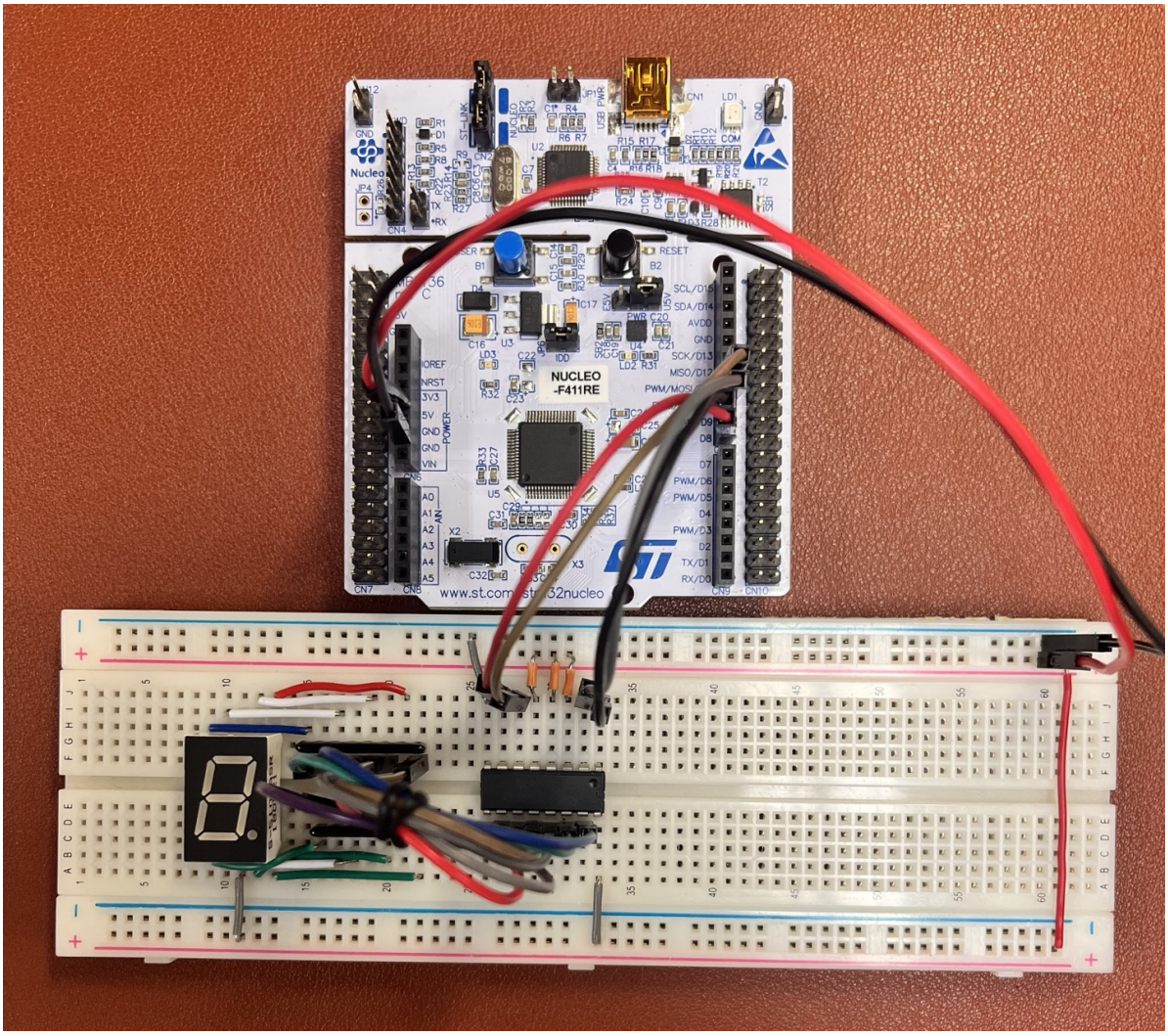
```
void SysTick_reset(void)
{
    // SysTick Current Value Register
    SysTick->VAL = 0;
}

uint32_t SysTick_val(void) {
    return SysTick->VAL;
}
```

The above function resets SysTick counter value, and the lower function returns SysTick value.

```
void SysTick_enable(void) {
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
}

void SysTick_disable(void) {
    SysTick->CTRL = 0;
}
```

This function enables or disables SysTick.

vi. Results



demo video link

IV. Reference

https://github.com/AnGyeonheal/Embedded_Control_GH

V. Troubleshooting

(Option) You can write a Troubleshooting section