TA Andrew Gerber 10/27/2023

# Lab 06 Report

Benjamin Meads
A02323437
Nick Templeton
A02268123
Richard Snider
A02298443

## Objectives

The objective of this lab was to be introduced to the SysTick system including how to set up an interrupt timer. As well as the interrupts using the NVIC system. We were to learn how to use the GPIO interrupts. Then integrate that into a timer that 3 push buttons can control. These push buttons are green to start the timer, a yellow button to pause the timer, and finally, a red button to reset the timer. These buttons need to be called using EXTI interrupts.

## Procedure

Our process in this lab started like many others. Trying to understand in more detail the contents that we have discussed in class. Not only the logic behind interrupts, SysTick, and our EXTI NVIC system. But we also had to learn how to implement and use this information in order to set up the system in the desired way. We reviewed the slides given in class and attempted to create some code that would do it. However, this is when we received information regarding the textbook. Chapter 11 in the textbook had a plethora of information, code, and explanation that was necessary for us to move farther along in this lab.

The next segment of this lab consisted of us trying to straight copy the code and modify it into our system. This included switching and modifying the code in the textbook from having one GPIO pin like they used in the book, into the 3 GPIO pins we were trying to implement. However, we learned very quickly that just copying down the information was not the correct or smart way of moving forward. So we needed to spend some time looking at the code given to us in the book. Then we dissected it into the registers in our textbook to learn what each line of code did. Then after we learned further about what the textbook's code actually did. We were able to correctly modify it into what we were looking for. We knew what pieces of the code we needed to change and we knew what to correctly change them to.

After we understood how to set up the SysTick as well as our EXTI. We then had to implement the rest of the logic of the code. This included changing our MODER register to set up the GPIO pins we wanted to use to be Inputs. It also modified our PUPDR register to make all of our buttons be pulled up. Then once we had completed that. We went along to create the rest of the logic of the program. This was somewhat difficult including a button press while keeping in mind interrupts. But it consisted of just entering our SysTick interrupt after our button was pressed. We simply activated the interrupt when the green button was pressed. Disabled our SysTick when the yellow and red buttons were pressed. As well as reset our timer variables in the red button press. We created variables “minutes”, “seconds”, and “tenths” variables and created the logic of the clock to count up correctly. We then put all of these into a character array and had to cast them to the correct type so our LCD program would respond correctly. Then a simple function to pass our character array into the LCD program so that it could display it.

After we had this setup, we were almost done, but we just had to go through the code a few more times to fix the small mistakes we had made. However, this process lasted a lot longer than anticipated. After checking the logic of our code, again and again, all seemed lost. Until the IDR register caught my eye. It didn’t even seem to register when the button was pressed. Or when I physically touched the wires together. After hours of searching through the code to see what the issue was. It turned out to just be a typo of when I set the PUPDR, I actually had set the MODER instead. Thus turning the GPIO pins to be output. Once I checked that and fixed it. The code launched and worked correctly right away. I was then able to pass us off.

## Results

This lab started with us trying to take the concepts we’ve learned with interrupts and be able to implement them into the requirements. We were able to complete the lab after many hours of debugging. We were able to get the timer to display on the LCD. We were also able to get the buttons to respond accordingly. However, the red button was able to reset it, but it was not able to show the actual reset value. It would display the reset value after starting the timer again. This technically is within the requirements of the lab so we were able to pass it off.

## Conclusion

This lab had a bit of a rough start but as it went along, we were able to figure out the logic a bit more and more. We learned a lot more in-depth about how these interrupts work. We learned how to set up a SysTick timer as well as how to implement that interrupt with EXTI registers—definitely learned a lot about how interrupts work with

these timers and the NVIC system. Then execute all of that knowledge into a working system that uses inputs and outputs.

## Programs

Predictor.cc

```
#include "stm32l476xx.h"
#include "Interrupts.h"
#include <stdio.h>
#include "LCD.h"

// static int32_t TimeDelay;
static char tenths, seconds, minutes;

void SysTick_Initialize (uint32_t ticks) {
	SysTick->CTRL = 0;

	SysTick->LOAD = ticks - 1;
	NVIC_SetPriority (SysTick_IRQn, 4);

	SysTick->VAL = 0;

	SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk;
	SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;
	//SysTick->CTRL |= SysTick_CTRL_ENABLE;

	RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;

	//Set Set buttons to inputs
	uint32_t new_moder = GPIOC->MODER;
	//sets PC 0-2 to be 00 for input mode
	new_moder &= 0xFFFFFFC0;
	GPIOC->MODER = new_moder;

	//Set buttons to pull-up
	uint32_t new_pupdr = GPIOC->PUPDR;
	//Sets PB8-10 to be 01 for pull up
	new_pupdr &= 0xFFFFFFC0;
	new_pupdr |= 0x00000015;
	GPIOC->PUPDR = new_pupdr;

	//SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
}

//Timer Implementation
void SysTick_Handler(void) {
	//if green button pressed
```

```
	tenths++;
	//
	if(tenths >= 9){
		seconds++;
		tenths = 0;
	}
	if(seconds >= 59){
		minutes++;
		seconds = 0;
	}

	unsigned char time[8];

	snprintf((char *)time, 8, "%02d:%02d.%d", minutes, seconds, tenths);
	LCD_DisplayString(0, time);
}

void EXTI_Initialize() {
	//Enable SYSCFG clock
	RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

	//This is setting PC 0,1,2 to our source of EXTI2
	SYSCFG->EXTICR[0] |= (SYSCFG_EXTICR1_EXTI0_PC |
SYSCFG_EXTICR1_EXTI1_PC | SYSCFG_EXTICR1_EXTI2_PC);

	//enabling rising edge and disabling falling edge for EXTI 0,1,2
	EXTI->RTSR1 |= (EXTI_RTSR1_RT0 | EXTI_RTSR1_RT1 | EXTI_RTSR1_RT2);

	//EXTI->FTSR1 &= ~(EXTI_RTSR1_RT0 | EXTI_RTSR1_RT1 |
EXTI_RTSR1_RT2);
	EXTI->FTSR1 &= ~EXTI_RTSR1_RT0 & ~EXTI_RTSR1_RT1 & ~EXTI_RTSR1_RT2;

	//Enable the EXTI 0-2 Interupt for the source
	EXTI->IMR1 |= (EXTI_IMR1_IM0 | EXTI_IMR1_IM1 | EXTI_IMR1_IM2);

	//setting priority
	NVIC_SetPriority(EXTI0_IRQn, 1);
	NVIC_SetPriority(EXTI1_IRQn, 3);
	NVIC_SetPriority(EXTI2_IRQn, 2);

	//Enable the EXTI 0-2 Interupt for the dest
	NVIC_EnableIRQ(EXTI0_IRQn);
	NVIC_EnableIRQ(EXTI1_IRQn);
	NVIC_EnableIRQ(EXTI2_IRQn);

	SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
}
```

```
//Green button
void EXTI0_IRQHandler(void) {
      if ((EXTI->PR1 & EXTI_PR1_PIF0) == EXTI_PR1_PIF0){
            //Start the interrupt
            SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
            //Clear Interrut request
            EXTI->PR1 |= EXTI_PR1_PIF0;
      }
}

//Yellow button
void EXTI1_IRQHandler(void) {
      if ((EXTI->PR1 & EXTI_PR1_PIF1) == EXTI_PR1_PIF1){
            //Pause/stop the interrupt
            SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk;
            //Clear Interrupt request
            EXTI->PR1 |= EXTI_PR1_PIF1;
      }
}

//Red button
void EXTI2_IRQHandler(void) {
      if ((EXTI->PR1 & EXTI_PR1_PIF2) == EXTI_PR1_PIF2){
            //reset the values for the array
            SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk;
            minutes = 0;
            seconds = 0;
            tenths = 0;

            //SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk;
            //Clear Interrut request
            EXTI->PR1 |= EXTI_PR1_PIF2;
      }
}
```

Predictor.h

---

```
void SysTick_Initialize(uint32_t ticks);
void SysTick_Handler(void);
void EXTI_Initialize(void);
void EXTI0_IRQHandler(void);
void EXTI1_IRQHandler(void);
void EXTI2_IRQHandler(void);
```

Main.c

```
#include "stm32l476xx.h"
#include "LCD.h"
#include "Interrupts.h"

int main(void){

      LCD_Init();
      LCD_Clear();
      SysTick_Initialize(500000);
      EXTI_Initialize();

      //LCD_DisplayString(0, (unsigned char*)"ECE 3710");
      //LCD_DisplayString(1, (unsigned char*)"Utah State");

      while(1);
}
```

Pseudocode

```
SysTick Interrupt Handler:
    If timer state is Started:
        Increment the tenths
    If tenths of a second reach 10:
         Increment seconds
        Reset tenths of a second
    If seconds reach 60:
        Increment minutes
        Reset seconds
    Update the LCD display with the current time in MM:SS.T format

Button Green Press Interrupt Handler:
    If Green Btn Debouncer is not running:
        Start the debounce timer
    If timer state is Stopped:
        Set the timer state to Started

Button Yellow Press Interrupt Handler:
    If Yellow Btn Debouncer is not running:
        Start the debounce timer
    If timer state is Started:
        Set the timer state to Paused

Button Red Press Interrupt Handler:
    If Red Btn Debouncer is not running:
        Start the debounce timer
        Set timer state to Stopped
```

Reset timer variables (minutes, seconds, tenths of a second)
Update the LCD display to show 00:00.0

Debounce Timer Overflow Interrupt Handler:
Disable the debounce timer
Handle button press event (e.g., start, pause, or reset) as per the button press handler Main Loop: Run the main loop to handle other tasks and processes