Owen Hushka Programming Embedded Systems Final Project 12/17/23

Project Description

For my project, I created a reflex tester as described in the lab instructions. I used SysTick for tracking time and interrupts for tracking the reaction of the user. The LCD was used to display shapes and animations for testing the user's reaction time. The RNG was used to find random values in order to make the shapes and events occur at random times.

Scope

I will cover the full scope of the reflex tester described in the lab document. This means using the LCD, RND, and button interrupts. I will use the SysTick for tracking time, button interrupts for tracking the reaction of the user, the LCD for displaying snapes and animations, and the RNG to find random values in order to make the events occur randomly. The final product should be able to print the reaction time to each game in milliseconds. It should provide clear instructions to the user and should run smoothly and free of errors.

Testing/Verification Strategy

My final product should be usable by somebody who does not know what the project is. Meaning, the game functions well enough and the instructions are clear enough that somebody could be handed the STM and be able to play the game with no help. I will test this by having several of my friends use the game and see if they are able to use it without my help.

The final product should be able to print the difference between the event occurring and the button being pressed in milliseconds. I will test this by trying the game extensively myself and trying to test edge cases such as pressing the button too early or waiting long to press the button. Additionally, I will press the button extra times.

Timeline

Nov 30 – Check in 1
–By check in 1, I should have the project generated and I should have a general plan for how I am going to complete the project. All filesets and the appropriate includes should be finished. The LCD demo should be runnable.

Dec 7 – Check in 2
–By check in 2 I would like to have populated the RNG filesets with the necessary macros and functions. Additionally, I should have the LCD functions created to be able to display the different parts of the game. I should also be able to accept interrupts from the button.

Dec 10 – By the end of the weekend before the final project is due, I would like to have the initial game running where the LCD runs through the simulations. This includes using the RNG to randomly display the shapes. I should be starting to accept the user input from the button and I should be trying to print this to the screen

Dec 12 – By the last class before the demo I would like to have tried to complete the project. I should use this last class to finalize the project and check it with TA's to make sure it meets the requirements.

Work Breakdown

Using the LCD

- –LCD driver files are provided
- –Create functions for the different animations
  - –Triangle
  - –Square
  - –Circle crossing over a line
  - –Grid

Writing the RNG driver files

- –Init function
- –GetRand function

ApplicationCode

- –Functions to print instructions
- –Functions to print time to
- –Button Interrupt Handler

Main

- –Using systik
- –Being able to implement the functions I wrote in order to put together my final project.
- –Being able to interpret input from the button
- –Using the scheduler to know when the button has been pressed (interrupt handler turns adds "button pressed" event

Use Cases

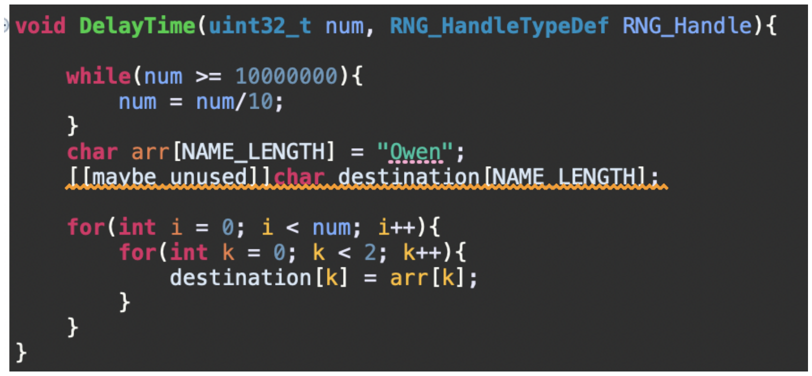
This project can be used to test the user's reflexes. The user responds to events in a couple quick games and their response times are displayed for them in milliseconds.

Project Documentation

- "How to guide"
- Coding Document

Struggles and Obstacles

Another obstacle I was initially unsure of how to overcome was delaying for a random amount of time. I was able to do this by writing my own version of the delay function provided for us in previous labs.



```c
void DelayTime(uint32_t num, RNG_HandleTypeDef RNG_Handle){

    while(num >= 10000000){
        num = num/10;
    }
    char arr[NAME_LENGTH] = "Owen";
    [[maybe_unused]]char destination[NAME_LENGTH];

    for(int i = 0; i < num; i++){
        for(int k = 0; k < 2; k++){
            destination[k] = arr[k];
        }
    }
}
```

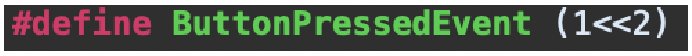
The delay function takes in a uint32_t value “num” which is a random value that has been generated from the RNG peripheral. Through trial and error, I was able to figure out the range of values “num” could be in in order to generate a delay which was not unreasonably long.

Another obstacle that I encountered was finding the user’s reaction time. Especially difficult was finding the user’s reaction time when responding to the ball crossing the line.

In order to overcome this obstacle, I created a button pressed event macro in the scheduler. In the IRQ handler for the button interrupt, I set this event.

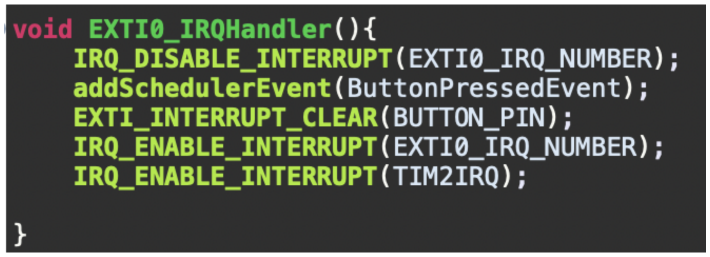
In my main program, I used the HAL’s GetTick() function in order to store the time when an event occurred. I broke the ball’s animation up so that LCD_DISPLAY_BALL1() will display the animation up until the ball finishes crossing the line. I then store the GetTick() value, clear the button pressed event, and wait until the button pressed event is set again, thus indicating that the user has pressed the button in response to the ball crossing the line. I then call GetTick() another time, save the difference between the two values, and call LCD_DISPLAY_BALL2() which will continue the animation until the ball finishes crossing the line again. I then repeat this process until the ball has crossed the line 3 times.

In the scheduler.h



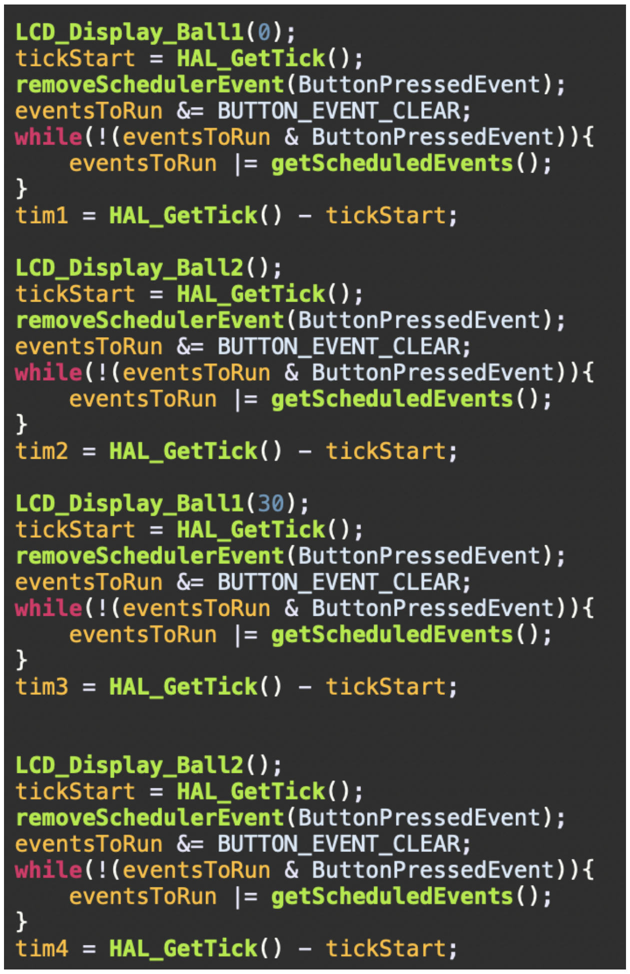
```c
#define ButtonPressedEvent (1<<2)
```

In application code



```c
void EXTI0_IRQHandler(){
    IRQ_DISABLE_INTERRUPT(EXTI0_IRQ_NUMBER);
    addSchedulerEvent(ButtonPressedEvent);
    EXTI_INTERRUPT_CLEAR(BUTTON_PIN);
    IRQ_ENABLE_INTERRUPT(EXTI0_IRQ_NUMBER);
    IRQ_ENABLE_INTERRUPT(TIM2IRQ);

}
```

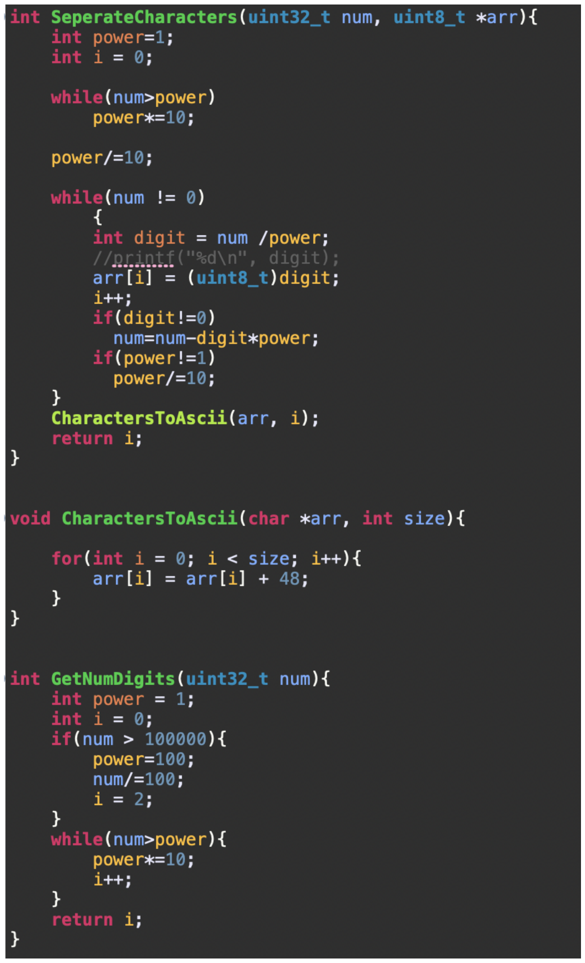
In the main:



```c
LCD_Display_Ball1(0);
tickStart = HAL_GetTick();
removeSchedulerEvent(ButtonPressedEvent);
eventsToRun &= BUTTON_EVENT_CLEAR;
while(!(eventsToRun & ButtonPressedEvent)){
    eventsToRun |= getScheduledEvents();
}
tim1 = HAL_GetTick() - tickStart;

LCD_Display_Ball2();
tickStart = HAL_GetTick();
removeSchedulerEvent(ButtonPressedEvent);
eventsToRun &= BUTTON_EVENT_CLEAR;
while(!(eventsToRun & ButtonPressedEvent)){
    eventsToRun |= getScheduledEvents();
}
tim2 = HAL_GetTick() - tickStart;

LCD_Display_Ball1(30);
tickStart = HAL_GetTick();
removeSchedulerEvent(ButtonPressedEvent);
eventsToRun &= BUTTON_EVENT_CLEAR;
while(!(eventsToRun & ButtonPressedEvent)){
    eventsToRun |= getScheduledEvents();
}
tim3 = HAL_GetTick() - tickStart;


LCD_Display_Ball2();
tickStart = HAL_GetTick();
removeSchedulerEvent(ButtonPressedEvent);
eventsToRun &= BUTTON_EVENT_CLEAR;
while(!(eventsToRun & ButtonPressedEvent)){
    eventsToRun |= getScheduledEvents();
}
tim4 = HAL_GetTick() - tickStart;
```

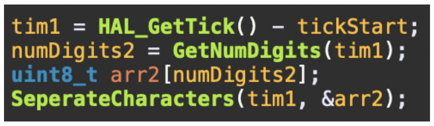
A final challenge I had to overcome was printing the results to the LCD screen. Specifically, printing the response times was difficult. In order to do this, I created several application code functions.

These functions find the number of digits which a number will require to be printed to the LCD. I then create a uint_8 array of that size. I then send this array as a pointer value as well as the uint32_t number representing the user's reaction time. I then break up the number into its individual digits and store this to the array. I then call a function which adds 48 to each of the uint8_t values in the array. This changes each digit to be their appropriate ASCII value.

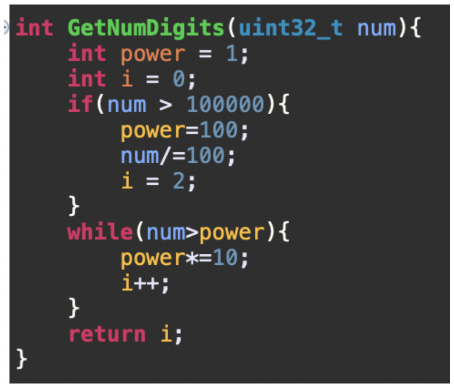


```
int SeperateCharacters(uint32_t num, uint8_t *arr){
    int power=1;
    int i = 0;

    while(num>power)
        power*=10;

    power/=10;

    while(num != 0)
        {
        int digit = num /power;
        //printf("%d\n", digit);
        arr[i] = (uint8_t)digit;
        i++;
        if(digit!=0)
          num=num-digit*power;
        if(power!=1)
          power/=10;
    }
    CharactersToAscii(arr, i);
    return i;
}


void CharactersToAscii(char *arr, int size){

    for(int i = 0; i < size; i++){
        arr[i] = arr[i] + 48;
    }
}


int GetNumDigits(uint32_t num){
    int power = 1;
    int i = 0;
    if(num > 100000){
        power=100;
        num/=100;
        i = 2;
    }
    while(num>power){
        power*=10;
        i++;
    }
    return i;
}
```

In the main:



```
tim1 = HAL_GetTick() - tickStart;
numDigits2 = GetNumDigits(tim1);
uint8_t arr2[numDigits2];
SeperateCharacters(tim1, &arr2);
```



```
int GetNumDigits(uint32_t num){
    int power = 1;
    int i = 0;
    if(num > 100000){
        power=100;
        num/=100;
        i = 2;
    }
    while(num>power){
        power*=10;
        i++;
    }
    return i;
}
```

Another sub-challenge I faced was that if “num” was too big, the power would eventually overflow when executing the while (num>power) loop. I was able to work around this by adding the if-statement.

What I Learned

- Writing driver files from scratch
    - Although we had written driver files and functions before this, it was a valuable learning experience to read through the manual and understand the RNG, read through the HAL_RNG files to understand the RNG driver functions and then implement them in my own RNG driver files.
- Planning out a project from scratch
    - Planning out a project of our choice with no directions was a great learning experience. I think I have made a lot of progress from the start of the semester when I was struggling with projects which had step to step directions.
- Planning include hierarchy
    - Planning the hierarchy of includes was another aspect of this project which was new to me but a great learning experience.
- Using new peripherals
    - I got practice with reading documentation, specifically the STM reference manual, in order to understand how to use the peripherals on the STM32.
- Implementing other people’s code
    - I employed the HAL’s driver files throughout the project. This was good practice with reading and understanding code that other people wrote and understanding how to implement it in my code.

What would you do differently

If I were to do this project differently I do not think I would change much regarding my process.

One thing I would do differently is my method for printing the time it took for the user to react to the event occurring. Instead of taking this time value, finding the number of digits it has, creating an array the size of the number of digits in number, separating this number into its individual digits, storing these digits in the array and then taking this array and incrementing the character values to be their appropriate ASCII value … I could have instead just kept the time value in a uint32_t variable until the game finishes and then used a single function to individually take the digits in the number, translate this digit to ASCII and send that digit to be printed to the LCD.

How could the project be improved

One limitation for my project is in the third test where the ball crosses the line, if the user does not press the button right when the ball crosses the line there may be a delay. Instead of using a while loop and waiting for a response, I could just use the interrupt controller to store the time of the button press.