**LAB 7**

**INTERRUPT PROJECT:**

**HUMAN RESPONSE TIMER**

Contents

[1 Introduction 1](#_Toc192251269)

[1.1 Lab overview 1](#_Toc192251270)

[2 Functions Used 1](#_Toc192251271)

[3 Requirements 1](#_Toc192251272)

[4 Details 2](#_Toc192251273)

[4.1 Hardware 2](#_Toc192251274)

[4.2 Software 2](#_Toc192251275)

[4.3 Testing 2](#_Toc192251276)

[4.4 Lab Modifications 3](#_Toc192251277)

# Introduction

## Lab overview

For this project, you will create a device which measures how quickly a person can press a switch in response to an LED being lit. This will give you an idea of how much work the processor can do in the time it takes you to react to an event.

# Functions Used

* gpio\_set(PIN pin, int value) - Sets the selected pin to the specified value
* leds\_init() – Sets the 3 led pins to outputs and sets each led to 0 to start
* leds\_set(int red\_on, int green\_on, int blue\_on) – Turns on (with a 1) or off (with a 0) each led in sequence, from red to green to blue
* delay\_ms(unsigned int ms) – delays for the specified number of milliseconds. It isn’t perfectly accurate but a good enough approximation for our purposes
* rand() - Computes a pseudo-random integer in the range 0 to RAND\_MAX. We combine this with % (for modulus division) to further restrict the range from 0 to whatever you perform modulus division by minus 1. For example, the starter code with delay\_ms((rand() % 5001) + 2000) generates a random number, forces it into the range from 0 to 5000, and then adds 2000 to bring the range from 2000 ms to 7000 ms (2 seconds to 7 seconds).
* gpio\_set\_mode(PIN pin, PinMode mode) - Sets the output mode of a pin. Will primarily be either Input or Output modes for us
* gpio\_set\_trigger(PIN pin, TriggerMode trig, PinMode mode) - Sets the interrupt trigger for the specified pin. The TriggerMode can be Rising or Falling to check for either rising or falling edges respectively. You may also occasionally see None used to disable the interrupt. For PinMode, we will generally use PullUp to properly check for the interrupt.
* gpio\_set\_callback(PIN pin, void (\*callback)(int status)) – Basically just associates the pin with a particular interrupt being used

# Requirements

In this lab, we will be using the following hardware and software:

* **Keil µVision5 MDK IDE**
  + Please see the included Getting Started with Keil guide on how to download and install Keil.
* **STM32 Nucleo-L552ZE-Q**
  + For more information, click [here](https://www.st.com/en/evaluation-tools/nucleo-l552ze-q.html).
* **Logic Analyzer or Oscilloscope** 
  + Optional to monitor interrupt signals or related information that you generate

# Details

## Hardware

Use your same setup as from Lab 6!

## Software

The main code performs the following:

* Initialize peripherals
* Repeat the following
  + Turns off all LEDs
  + Clears counter
  + Waits a random amount of time (e.g. within 2-7 seconds)
    - Don’t press the button during this time!
  + Turns on one LED
  + Repeats until ISR has been triggered (from pressing the button), as indicated by the flag being set
    - increment counter and save it

The ISR performs the following:

* Sets a flag indicating the ISR has executed

You will also need some support functions:

* Use the **leds.c** module to initialize and control the RGB LEDs.
* Use the C standard library function **rand()** to generate a random integer.
* Use the **delay\_ms** function provided by **delay.c** to wait for a number of milliseconds.

## Testing

To see the number of iterations counted, set a breakpoint in your main function after the switch press has been detected and examine the counter variable using the watch window.

## Lab Modifications

Take the existing code and accomplish the following (**submitting a separate main.c file or commenting out the various lines of code needed for each situation**):

* Modify the code to collect and then average together ten response time measurements.
  + Include the average time (count) values you obtain when actually doing this (after ten total iterations that you then calculate the average of) as a comment within your code
* Modify the code so that you can determine if the user’s response time changes depending on which color the LED generates. (i.e. make the LED one color sometimes and another color at other times, keep track of the time for each via different count variables)
  + Include time (count) values you obtain while actually doing this in each case as comments within your code
* Modify the code so that you can determine if user’s response time is the same when the LED turns on vs. turns off. (i.e. make the LED turn on sometimes and turn off at other times, keep track of the time for each via different count variables)
  + Include time (count) values you obtain while actually doing this in each case as comments within your code

There is no single right approach for most of these! You should be relying on your own programming knowledge, your partner, and the Keil environment to help you. On top of the disassembled program and register values that you have come to learn about, you may find it useful to right click on the name of a programming function (such as leds\_set) and click on “Go to Definition of…” in order to see what the function is doing (this does not work sometimes/for some users – I am honestly not sure why). We also provided a description of most of the primary used functions earlier in this write up.

I would mostly encourage caution when using any internet-based resources, as the functions we are using here are not necessarily standardized and you may receive false information about how to use them!