ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E

MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO : 6

EXPERIMENT DATE : 13.12.2020

LAB SESSION : WEDNESDAY - 13.30

GROUP NO : 3

GROUP MEMBERS:

150170013 : ALİ KEREM YILDIZ

150170082 : ARDA CÜCE

150170908 : MUHAMMAD RIZA FAIRUZZABADI

FALL 2020-2021

Contents

1 INTRODUCTION [10 points]

In this experiment, we were tasked to program a couple of given circuits' design that aims to give comprehensive understanding regarding the use of Arduino Timer Interrupt subroutines, implementing it for different uses such as with 7-Segment Display and LEDs. Using Timer Interrupts and other concepts we learned before such as external Attach-Intterupt, we were set to make a program that generate given patterns in LEDs, and a stopwatch.

2 MATERIALS AND METHODS [40 points]

2.1 MATERIALS

Tools Used[1]

- Autodesk Tinkercad
- Latex (overleaf.com)
- Arduino Uno

This experiment is done via Autodesk Tinkercad Design Tool as well as the previous experiments. We used the 'Microcomputer Lab Intro' Lecture Notes as a reference regarding the overall structure of the board, alongside 'Segment Digit LED Display' User Manual and sources regarding the new concepts of Timer Interrupt implemented in the experiment.

2.2 METHODS

2.3 PART 1

In this part of experiment, we will generate some given patterns on leds. The task forces us to use timer interrupt to generate patterns. In setup() function we should set some registers in order to create a timer interrupt. First, we should stop interrupts by cli(). Then we should decide which timer are we going to use (timer0 or timer1). We select timer1 which take values smaller than 65536. Then we need to set TCCR1A and TCCR1B register to 0. and our counter TCNT1 setted to 0 also. We want to create timer interrupt every 1 second. Which means our period(T) and delay frequency (f) is 1. If we set TCCR1B's CS10 and CS12 to 1 according to its documentation we evaluate prescalar = 1024. So we know prescalar and frequency, and oscillator clock signal is 16MHZ. $OCR1A = (16MHz/(prescaler * 1) - 1$. OCR1A evaluated as 15624. In order to turn on CTC mode TCCR1B's WGM12 setted to 1. And We can easily calculate OCR1A value. After the resetting flag bits, we can allow interrupts. Then, using DDR register we set led connected pins as output and button connected pin as input. Setup phase was finished. In order to generate first pattern by default, we set a mode variable and initialize it as global variable that we check it whether we are going to display first pattern or second in future. In first pattern, when leftmost 4 leds shifting left, rightmost 4's are shifting right and vice versa. We divide this pattern 2 peaces which we call them first-half-pattern1 and second-half-pattern1. After the given default values of them, our program make opposite shifting operations on first-half-pattern1 and second-half-pattern1. For example, at time=0, led pattern should = 00011000 (first-half-pattern1 = 0001 , second-half-pattern1 = 1000). first-half-pattern1 shifts left and second-half-pattern1 shifts right until pattern reaches values such (10000001). When we encounter this pattern we should going to make shifting operations opposite way for divided parts. For the second pattern, instead of 2 4-bits, we divided the pattern to two pieces of 1 and 7 bits. The leftmost bit, which is separated itself, is blinking ON and OFF according to the global counter variable that is incremented every time. This results in a single LED that is blinking on and off every second. For the other 7 bits that was divided, its value are shifted left or right according to a global variable setter, which value's changed to 0 or 1 according to the pattern. For instance in the beginning where the rightmost bit is ON, setter is set to 0. Then when that value is completely shifted to the right (1000000), setter becomes 1, resulting in right shift until it goes back to the initial value where the lowest bit is 1. The shifting functionalities are obtained with usual bitshift operators. Like in the previous pattern, the divided patterns are united back to a result variable, which first 6 bits are set to port D, and last 2 to port B. Up to this point we explained how this pattern's logic

works. When button is clicked, we change its pattern with switch-mode function which we pass an argument to attachInterrupt() in order to implement a interrupt subroutine for button press. Some pattern examples given below.

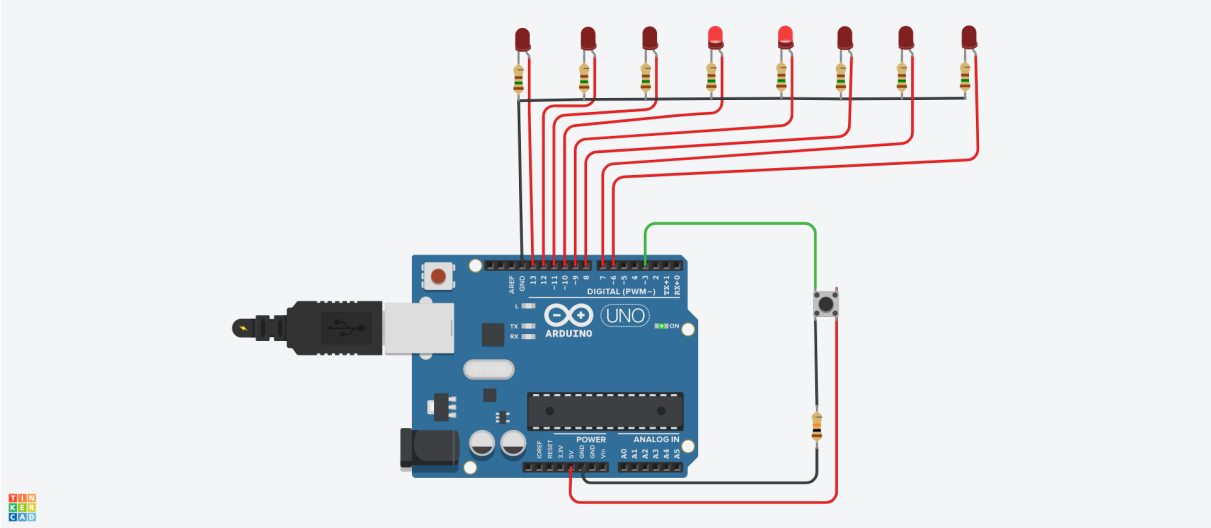

Figure 1: First pattern, initial value. Time = 0

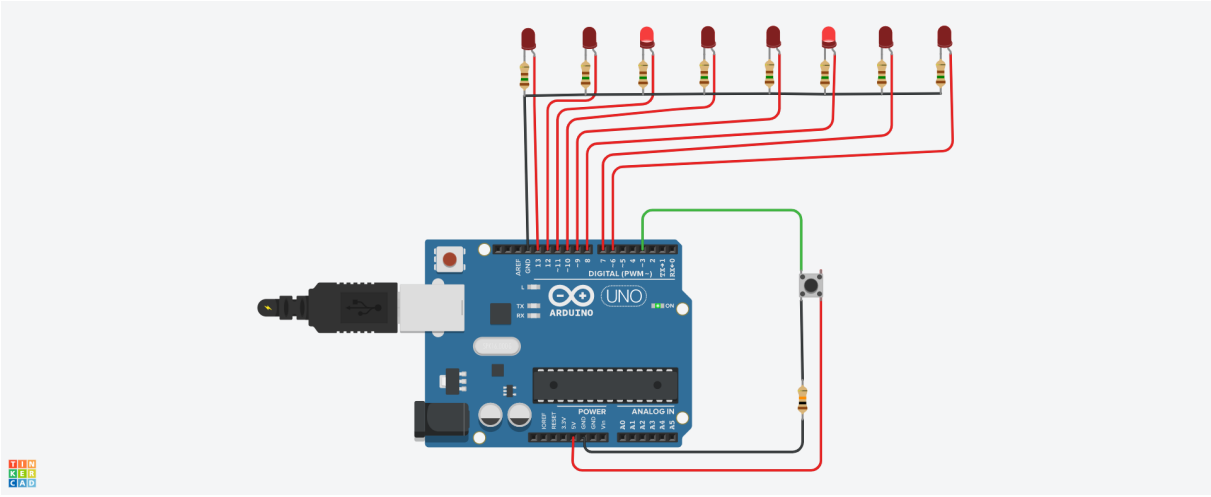

Figure 2: First pattern, second value. Time = 1

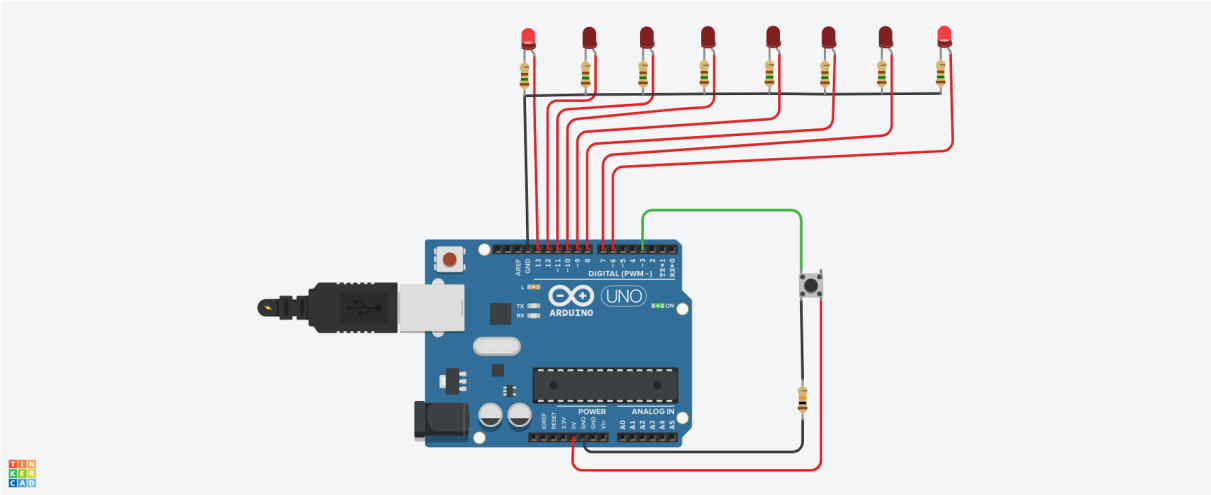

Figure 3: Second pattern, initial value. Time = 0

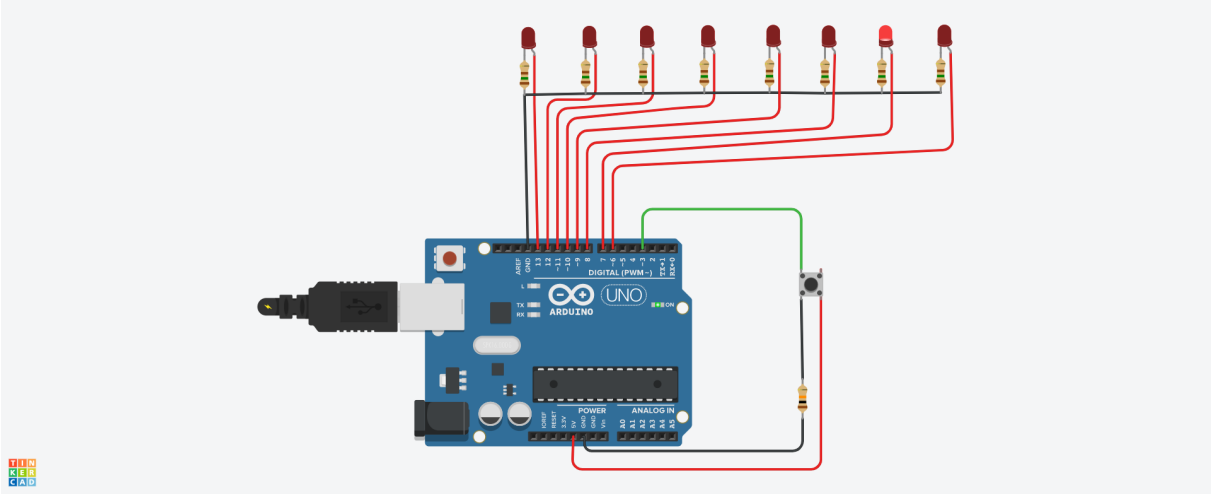

Figure 4: Second pattern, second value. Time = 1.

2.4 PART 2

In this part we implemented a stopwatch that counts down to centiseconds. To do so we used timer interrupts that interrupted every centisecond. We used the formula $OCR1A = (16MHz/(prescaler * 100) - 1)$ to find the compare value needed to call interrupt every centisecond first approximate prescaler was chosen as 256 so we turned TCCR1B's CS12 bit to high. Timer1 register can count up to 6536 and the result from our formula is 624, 624 ¡ 6536 so we don't need to increase our prescaler and. We loaded timer 1 compare register OCR1A with our result 624, then turned compare mode on by turning TCCR1B's WGM12 bit to high. As global variables counter, lap-timer, lap-count, begin and digits of the counter. We also attached an interrupt function to button one, when it is clicked begin variable is set to 1 from -1. Inside the timer interrupt, if begin

variable is 1 we increment counter variable (counter is incremented every centisecond so it respresents time in centi seconds), updates counter digits, increments lap timer, and checks if button 3 is clicked, if it is it resets the counters. We attached another interrupt function to button 2, when it is clicked print-lap function is called, this function prints lap time, lap count and total time to the serial output window, increments the lap count and resets lap time. To display the counter digits on the leds, we first deconstructed the counter variable to its decimal digits. Using our display function (from the previous experiment witch turns on a selected screen and displays the input value on it) inside the loop function, we switched the display that was on every 2 milliseconds so it created the illusion of all of them being on simultaneously. We originally weren't aware that we could use the loop function for this purpose so we had created another timer interrupt that was called every millisecond that we used in exactly the same way we use loop function.

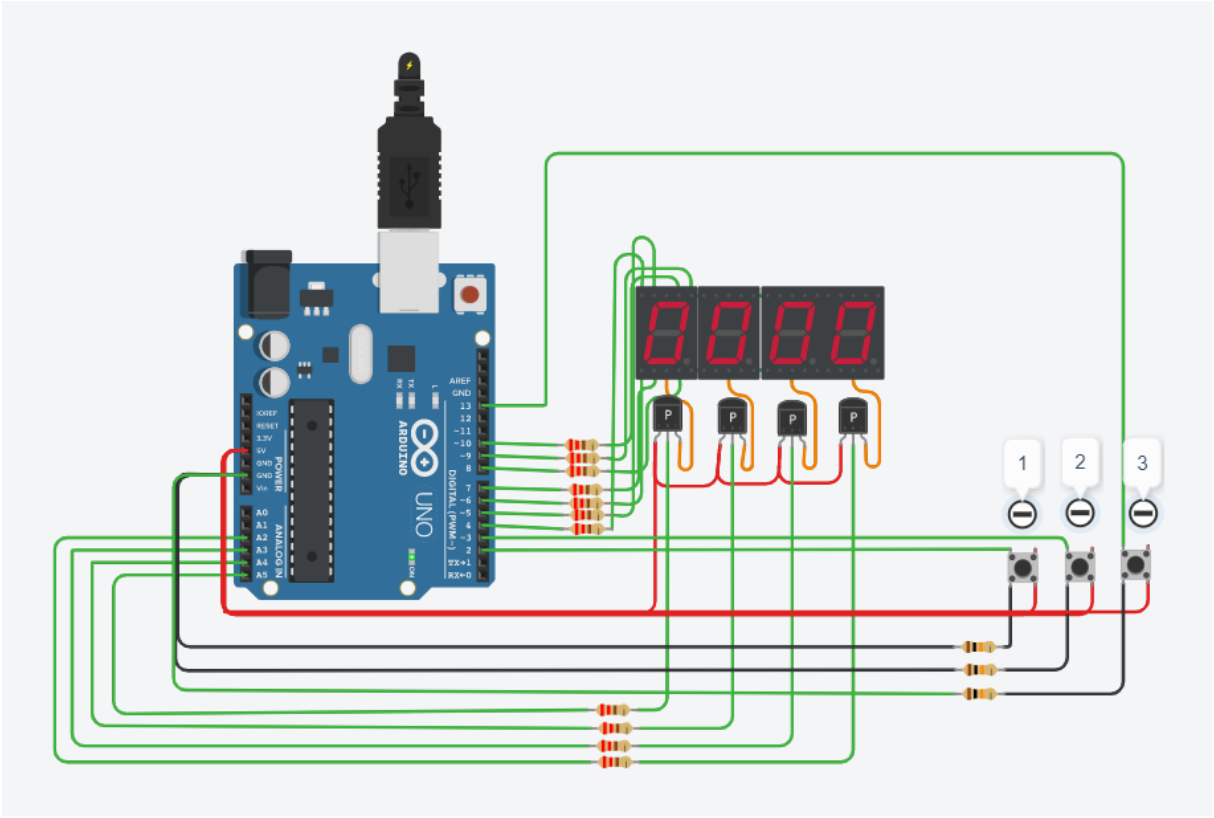

Figure 5: The condition before being activated/button pushed

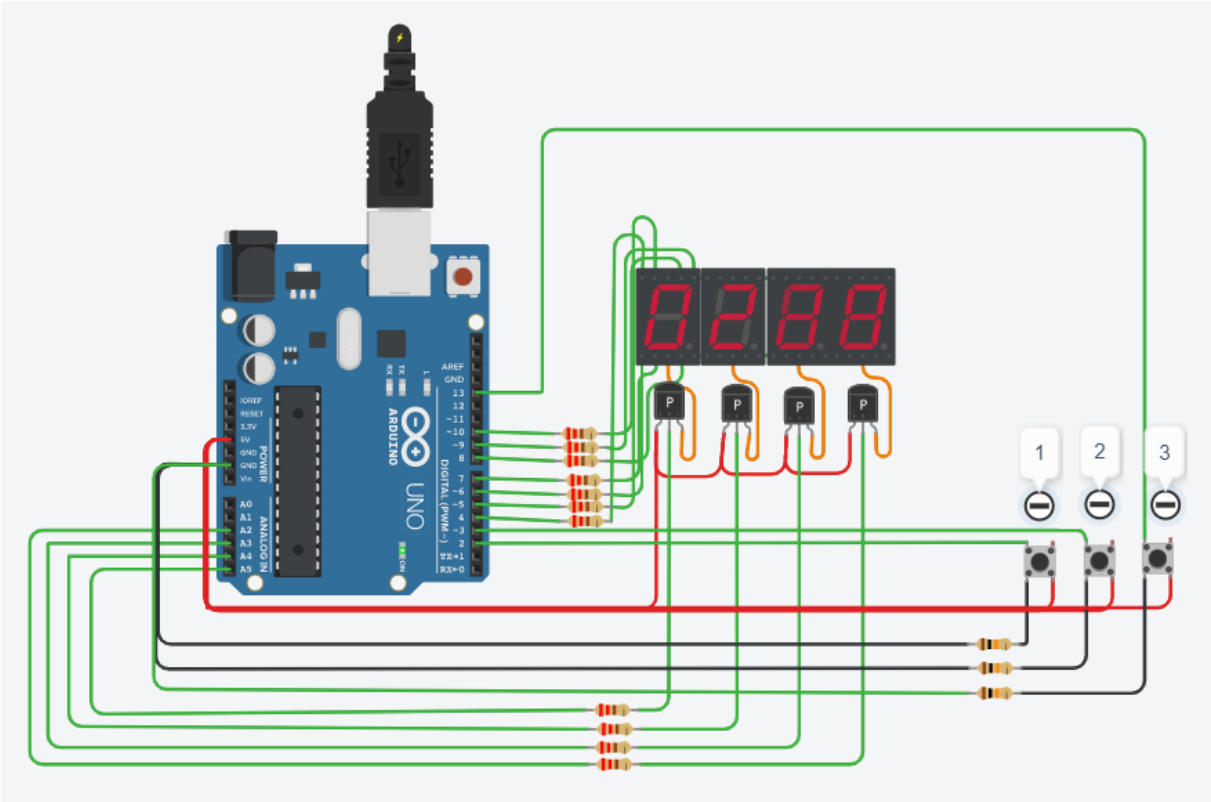

Figure 6: Active. Counting and displaying seconds alongside centiseconds

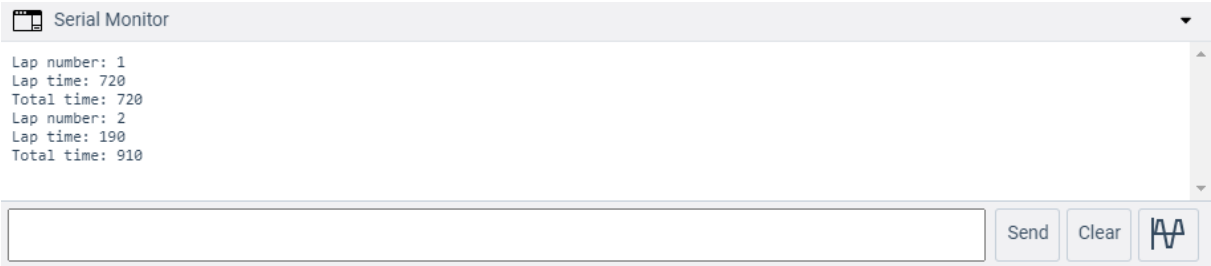

Figure 7: Lap button is clicked, lap number, lap time and total time displayed.

2.5 PART 3

3 RESULTS [15 points]

2 in 4 tasks given are accomplished. In those parts we got a chance to do further experiments with 7-Segment Display, LEDs, and pushbuttons. Some functions such as External Interrupts are not new to us, for that reason we managed to easily use, even re-use some functions we created in the previous part such as display. For some new concepts such as Timer interrupts, we managed to comprehend and implement them using the Arduino references we've used in the previous experiments as well.

4 DISCUSSION [25 points]

In this experiment, we accomplished tasks which are structurally similar with the previous parts, which are manipulating ports and displaying specific patterns or values in LEDs and 7-Segment Display. The main difference is being this time we had to use timer interrupt mechanism. This turned out particularly essential in our tasks, since in the tasks given are both time-sensitive (like having to generate outputs in exact seconds) and requiring us to get and display exact time data that we can get precisely from Arduino Uno timers.

5 CONCLUSION [10 points]

There were a couple of problems we encountered during the experiment. One is in Part 2, it wasn't clear to us if the display instruction 'can' or 'should' be put inside the loop function. For that reason we at first handled the display instruction outside the loop function, particularly within/using Arduino Uno's Timer0. However, afterwards we also wrote the display instruction inside the loop function, which also works as intended, in case if it is actually supposed/suggested to be put there. Though thanks to this, we got to use Timer0 alongside Timer1, giving us better information regarding the concept of timer interrupt subroutine.

REFERENCES

[1] Istanbul Technical University Department of Computer Engineering. Blg 351e microcomputer laboratory experiments booklet, Fall 2020-2021.