# ISTANBUL TECHNICAL UNIVERSITY

# COMPUTER ENGINEERING DEPARTMENT

## BLG 351E

## MICROCOMPUTER LABORATORY EXPERIMENT REPORT

**EXPERIMENT NO** : 5

**EXPERIMENT DATE** : 18.12.2020

**LAB SESSION** : FRIDAY - 15.30

**GROUP NO** : G11

## GROUP MEMBERS:


150170063 : BURAK ŞEN

150180080 : BAŞAR DEMİR

150190719 : OBEN ÖZGÜR


**FALL 2020**

# Contents

# 1 INTRODUCTION

In this week's experiment, we have designed our Arduino circuitries given in the experiment using interrupt subroutine feature. In the experiment, we have designed some basic counters first. Then, we have designed some dynamic counters that forms a simple game between two players using our own interrupt subroutine.

# 2 MATERIALS AND METHODS

- Tinkercad
- Arduino Uno R3
- 8 Resistors
- 8 LEDs
- 2 Buttons
- 2 Switches

## 2.1 PART 1

In the first part of the experiment, we were expected to design a circular up-down counter as in the previous weeks using an interrupt subroutine as opposed to previous weeks.

The circuit has 2 buttons and 2 seven-segment displays. The first button sets counter's value to 0 and second button changes the mode.

In order to reset or change mode of our circuitry the two buttons were connected to interrupt pins. We have designed two interrupt subroutines for our buttons. In addition, to provide circular decimal counting effect to our sevent-segment displays we have designed to functions.

Since the counter normally counts in binary form, we were supposed to provide decimal effect to our counter. For that purpose, we have designed two functions called increment and decrement. Increment first increments the lower digit then checks if the lower digits has reached to 10. If the lower digit has reached 10, upper digit is incremented by 1 to and lower digit is set to 0. This provides the decimal counting effect. There is another if statement inside the if statement as show in the code below.

This statement checks if upper digit has reached 10 which means “100” in full form. If that is the case counter resets itself back to “0”. This provides the circular effect. The decrease function follows the same mentality but in a reverse way to provide the circular decimal counting to our Arduino circuitry.

For buttons we have attached two interrupts on their rising edges. Reset button is attached to PIN3 and CHANGEMODE button is attached to PIN2, so we have set our attachInterrupt functions accordingly. When reset button is clicked the resetPINS interrupt subroutine is performed. resetPINS interrupt set lower digit and upper digit to “0”. When change mode button is clicked the circuit performs changeMode interrupt subroutine. changeMode changes the mode variable and performs the required mode’s operation directly inside the function because if the operation is not performed inside the function circuit must wait 1 more second after mode change.

Our implementation is given below.

```
int prev = 0; //temp variable for time
int upper = 0; //tens digit
int lower = 0; //ones digit

byte mode = 0b0; //mode 0 = inc, mode 1 = dec

const byte modePin = 2; //change mode button
const byte resetPin = 3; //reset button

//function declarations
void increase();
void decrease();
void changeMode();
void resetPINS();



void setup() {
  DDRD = 0b11110011; //input to register D
  DDRB = 0b11111111; //output to register B
  DDRC = 0b11111111; //output to register c
```

```
  attachInterrupt(digitalPinToInterrupt(modePin), changeMode, RISING);
  attachInterrupt(digitalPinToInterrupt(resetPin), resetPINS, RISING);
}

void loop() {
  int current = millis(); //get time

  PORTB = lower;//output lower digit to B
  PORTC = upper;//output upper digit to C

  if (current - prev >= 1000) { //1s delay

    if(mode == 0b0)
    {
      increase();
    }
    else{
      decrease();
    }

    prev = current;//set prev to current time
  }
}

void increase() //increment function
{
  lower++;
  if(lower == 10) //if ones digit is supposed to be 10
  {
    upper++; //increment upper digit
        if(upper == 10) //if number becomes 99 make it zero
    {
      upper = 0;
    }
    lower = 0; //make lower digit 0
  }
}
```

```
void decrease() //same mentality with increment
{
  lower--;
  if(lower == -1)
  {
    upper--;
        if(upper == -1)
    {
      upper = 9;
    }
    lower = 9;
  }
}

void changeMode() //change mode interrupt subroutine
{
    mode = !mode; //change mod

        if(mode == 0b0)//if mode is zero
    {
      increase(); //increment
    }
    else{ //if mode is one
      decrease(); //decrement
    }
}

void resetPINS() //reset interrupt subroutine
{
  //set both digits 0
  lower = 0;
  upper = 0;
}
```

## 2.2 PART 2

In this part of the experiment we implemented a game with the 7 segment displays. In this game the goal is to reach target value with button press.

There is a counter that counts 0 to F that we use to get values from. We shift these values on the 7 segment display to show our three digit number, if this three digit number is equal to target number, the game is over.

For making this game, we used interrupts, firstly we had to display our numbers normally. We can see that our first three pins on the ddrc registers, and other four pins on the ddrb registers. Therefore we had to come up with different solution for this, we defined two individual arrays for displaying the numbers. In the first array we hold the first three digit and in the second array we hold the last four digit for the seven-segment displays. We placed the values one by one to each of the arrays, if we want to display seven we just take the seventh value of the each array and send them to PORTB and PORTD and display the values.

For the interrupt part, we used again the same attachInterrupt function, in this funciton we set interrupt pin to 2, we set interrupt type to RISING (When we press the button it triggers immediately, for FALLING it would be when we press and release the button) and we create a function which called shiftCounter for the interrupt. When we press the button interrupt calls shiftCounter function immediately.

In the shift counter function we have shift our values to right, we acquire this behaviour with an array we store each of the seven-segment (other than the counter's) displays values in this array. And when we shift the values we just have to shift all of them to the right and for the first number we just placed counter's value. In this function we also checked if the game is finished or not to stop the game playing furthermore.

In the main Loop, we firstly use the millis function to calculate the time, we take the current time to current variable. In the first "if" statement, we defined a flag variable that counts the delay statements for completeness of the delays, firstly we checked current time's and previous time's difference is greater or equal to 5ms, and if our flag variable is 1 or not, if these are true first "if" is true, inside of the first "if" statement, we activated our transistor to show up the values of the first seven-segment display (which is the counter), we set the previous value to current value, and we set our

seven-segment display's value according to counter and finally we set the flag to next if statement's flag number. In the second, third and fourth "if" statements all of the above is the same thing except we did not set the counter's seven segment we set other seven-segment display's and we set these according to array variables's values.

In the fifth "if" statement we increase the counter by 1 if counter time's and previous time's difference is 1000ms (1s) and if our game is not finished. We controlled if counter is 16 or not and if it is 16 we set the counter to 0.

And in the final "if" statement we just checked if game is finished if our value is the target value if it is we set the finished flag to 1, and set all of the array values and counter to 0.

Our implementation is given below.

```
int counter = 0; //time counter

int flag = 1;
// a flag for controlling delays for each of the if statement
const byte interruptPin = 2;// set interruptpin 2

int finished_flag = 0;  //a flag to check if game is finished or not

int prev = 0;   // previous time for the counter
int prev1 = 0;  //previous time for the other delays

int target = 321;        //game target

int array[3] = { 0, 0, 0 };
//for storing values of 3 seven degment display

byte sevensegment0[16] = {
//first three bits of the seven segment display codes
  0b000,
  0b100,
  0b001,
  0b000,
  0b100,
  0b010,
  0b010,
  0b000,
  0b000,
  0b000,
  0b000,
  0b110,
  0b011,
  0b100,
  0b011,
  0b011
};
```

```
byte sevensegment1[16] = {
//last four bits of the seven segment display codes
  0b00010000,
  0b11110000,
  0b00100000,
  0b01100000,
  0b11000000,
  0b01000000,
  0b00000000,
  0b11110000,
  0b00000000,
  0b01000000,
  0b10000000,
  0b00000000,
  0b00010000,
  0b00100000,
  0b00000000,
  0b10000000
};

void shiftCounter(); //shiftCounter prototype

void setup()
{
  DDRD = 0b11111011;
  //set second bit as an input and other pins as an output
  DDRB = 0b11111111;
  DDRC = 0b11111111;

  attachInterrupt(digitalPinToInterrupt(interruptPin),
                  shiftCounter, RISING);
    //set pin 2 as an interrupt pin, option and interrupt function
}

void loop()
{
```

```
int current = millis(); //take current time

if (current - prev1 >= 5 && flag == 1) {
//if current time - previous time is 5ms
//and flag is 1 (for the first delay)
  PORTC = 0b011111; // activate first seven segment
   prev1 = current;    // set current time to previous time
  PORTD = sevensegment1[counter];     // show the value of the counter
  PORTB = sevensegment0[counter];
   flag =2; // set second if statement's flag
 }


 if (current - prev1 >= 5 && flag == 2) {
 //if current time - previous time is 5ms
 //and flag is 2 (for the second delay)
  PORTC = 0b101111;// activate second seven segment
   prev1 = current;// set current time to previous time
   // show the value of the second seven-segment
  PORTD = sevensegment1[array[0]];
  PORTB = sevensegment0[array[0]];
      flag =3;//set third if statement's flag
 }


 if (current - prev1 >= 5 && flag == 3 ) {
 //if current time - previous time is 5ms
 //and flag is 3 (for the third delay)
  PORTC = 0b110111;// activate third seven segment
   prev1 = current;// set current time to previous time
   // show the value of the third seven-segment
  PORTD = sevensegment1[array[1]];
  PORTB = sevensegment0[array[1]];
   flag=4;//set fourth if statement's flag
 }
```

```
  if (current - prev1 >= 5 && flag == 4){
  //if current time - previous time is 5ms
  //and flag is 4 (for the fourth delay)
    PORTC = 0b111011;// activate fourth seven segment
    prev1= current;// set current time to previous time
    // show the value of the fourth seven-segment
    PORTD = sevensegment1[array[2]];
    PORTB = sevensegment0[array[2]];
    flag=1;//set first if statement's flag
  }

  if (current - prev >= 1000 && finished_flag ==0) {
  // if current time - previous time is 1000ms
  //and if our game is not finished
    counter++; // increase the counter

    if(counter == 16)
    // if counter is 16 which is if seven-segment is F+1
    {
      counter = 0; // set counter to 0
    }
    prev = current;// set current time to previous time
  }

  if(array[0]*100+array[1]*10+array[2]==target){
  // we our array is the target value
    array[0]=0; // set all of the values to 0
    array[1]=0;
    array[2]=0;
    counter =0;
    finished_flag =1;
    // and set finished flag to finish the game
  }
}
```

```
void shiftCounter() // interrupt function
{
  if(finished_flag == 0){ // if game is not finished
    array[2] = array[1]; // shift second value to third value
    array[1] = array[0]; // shift first value to the second value
    array[0] = counter; // shift counters value to the first value
  }
}
```

## 2.3 PART 3

In this part, we are expected to implement a game that is played by using two buttons and two players. Main aim of game is catching number that is selected by first player. If second player catch same number, wins; otherwise player loses.

Firstly, we have defined our necessary flags and variables. In this experiment, we have to perform many comparison and checking operations. Therefore, we have needed lots of variables. For each display, we defined variable that keeps number that have to be shown in displays. We kept the game turn and status of the game (finished or not) in variables. In previous experiment, we have already defined two arrays that keep required bit combinations for numbers. Also, for delay operations, we have defined two variables that are kept times and used for calculating time that is passed.

In the setup phase, we have performed port manipulations to determine types of the pins. For digital (0-7) pins, we have defined 2 and 3 as input and other ones as output. Digital (8-13) and analog (0-5) is was defined as output ports. In this experiment, we need to handle button push operations using interrupt subroutines. In Arduino programming, defining a interrupt subroutine is simple with using attachInterrupt() function. In this function, we can give interrupt case, function that is run after interrupt and interrupt pins. We have used this function and we defined interrupts for rising edges of buttons and we call related functions. The digital port 2 is reserved for left player and port 3 is reserved for right player.

In loop phase, we have to handle all display and counter increment operations. Normally, we use delay function while displaying required numbers in proper order. But in this experiment, usage of delay function is forbidden. Therefore, we have

implemented delay mechanism using millis function. Using our flag, we have checked display that is should be activated and passed time. If time passed 5ms, we run proper scope. In scopes, we activate display using analog outputs and we give bit combinations that shows intended number to digital outputs. In the end, we update our flag to run next scope that manages another seven segment display and out previous time variable. After manipulating seven segment displays, we have to check our counter. Every 200ms, it has to increment. To perform this operation, we check passed time and game status. If game is finished, we do not have to increment our counter. Also, we check current player to determine counter position (left-most or right-most two bits). If it provides all conditions, we increment our counter variable.

We have two interrupt functions that are left player and right player. In both of the function, we check which player is the turn and game status to prevent undesired push operations. In left player function, we read selected number by player and we assign this value to our variable. We switch player turn for right player. In right player interrupt subroutine, we read player's selected number and finish the game. If selected numbers are same, we show 2222 in displays that means second player win; otherwise we show 1111 that means first player win.

Our implementation is given below.

```
const byte interruptPin1 = 2; //2. digital pin as interrupt pin
const byte interruptPin2 = 3; //3. digital pin as interrupt pin

int finished_flag = 0; //flag to check game if finished or not
int left_right_flag=0; //flag to check which player's turn
int flag = 1; //helper flag for delay mechanism

int left_select = 0; //Keeps left player's selected number
int right_select =0; //Keeps right player's selected number

int counter_left_upper = 0; //keeps leftmost display content
int counter_left_lower = 0; //keeps second leftmost display content
int counter_right_upper = 0; //keeps second rightmost display content
int counter_right_lower = 0; //keeps rightmost display content

int prev = 0; //helper previous time variable for counter
int prev1 = 0; //helper previous time variable for display mechanism
```

```
//seven segment content for analog ports
byte sevensegment0[16] = {
  0b000,
  0b100,
  0b001,
  0b000,
  0b100,
  0b010,
  0b010,
  0b000,
  0b000,
  0b000,
  0b000,
  0b110,
  0b011,
  0b100,
  0b011,
  0b011
};

//seven segment content for digital ports
byte sevensegment1[16] = {
  0b00010000,
  0b11110000,
  0b00100000,
  0b01100000,
  0b11000000,
  0b01000000,
  0b00000000,
  0b11110000,
  0b00000000,
  0b01000000,
  0b10000000,
  0b00000000,
  0b00010000,
```

```
  0b00100000,
  0b00000000,
  0b10000000
};

//interrupt function for left button
void leftplayer(){
  //if it left player's turn and game did not finished
  if(left_right_flag ==0 && finished_flag==0){
  //calculates selected number from bits
  left_select = counter_left_lower+10* counter_left_upper;
  left_right_flag =1; //turn of play passes to the second player
  }
}

void rightplayer(){
  //if it right player's turn and game did not finished
  if(left_right_flag ==1 && finished_flag==0){
  //calculates selected number from bits
  right_select = counter_right_lower+10* counter_right_upper;
  finished_flag = 1; //game finishes

  //if selected numbers are equal
  if(right_select == left_select){
    //displays show 2222
    counter_left_lower=2;
    counter_left_upper=2;
    counter_right_lower=2;
    counter_right_upper=2;
  }else{//if selected numbers are not equal
    //displays show 1111
    counter_left_lower=1;
    counter_left_upper=1;
    counter_right_lower=1;
    counter_right_upper=1;
  }
  }
```

```
}

void setup()
{
  DDRD = 0b11110011; // sets digital 2-3 as input, others are output
  DDRB = 0b11111111; // sets digital 8-13 as output
  DDRC = 0b11111111; // sets analog 0-5 as output

  //interrupt functions defined for interrupt pins
  //for signal rise behaviour
  attachInterrupt(digitalPinToInterrupt(interruptPin1),
                  leftplayer, RISING);
  attachInterrupt(digitalPinToInterrupt(interruptPin2),
                  rightplayer, RISING);
}

void loop()
{
 //time passed since the beginning of the circuit
 int current = millis();

 //if flag is 1, it means that left-most display should be shown
 //it controls 5ms passed or not
 if (current - prev1 >= 5 && flag == 1) {
    PORTC = 0b011111; //activates left-most display
    prev1 = current; //updates time
    //gives output for required number
    PORTD = sevensegment1[counter_left_upper];
    PORTB = sevensegment0[counter_left_upper];
    flag = 2; //updates flag as 2
  }

  //if flag is 2, it means that second left-most display should be shown
  //it controls 5ms passed or not
  if (current - prev1 >= 5 && flag == 2) {
    PORTC = 0b101111; //activates second left-most display
    prev1 = current; //updates time
```

```
  //gives output for required number
  PORTD = sevensegment1[counter_left_lower];
  PORTB = sevensegment0[counter_left_lower];
  flag = 3; //updates flag as 2
}

//if flag is 3, it means that second right-most display should be shown
//it controls 5ms passed or not
if (current - prev1 >= 5 && flag == 3 ) {
  PORTC = 0b110111; //activates second right-most display
  prev1 = current; //updates time
  //gives output for required number
  PORTD = sevensegment1[counter_right_upper];
  PORTB = sevensegment0[counter_right_upper];
  flag=4; //updates flag as 4
}

//if flag is 4, it means that right-most display should be shown
//it controls 5ms passed or not
if (current - prev1 >= 5 && flag == 4){
  PORTC = 0b111011; //activates right-most display
  prev1= current; //updates time
  //gives output for required number
  PORTD = sevensegment1[counter_right_lower];
  PORTB = sevensegment0[counter_right_lower];
  flag=1; //updates flag as 1
}

//if 200ms passed, game not finished and it is turn for left player
if (current - prev >= 200 && finished_flag ==0 && left_right_flag ==0) {
  //increments lower bit
  counter_left_lower++;
  //if it reaches to 10
  if(counter_left_lower == 10)
  {
    counter_left_lower = 0; //it sets lower bit as 0
    counter_left_upper++; //increments upper bit
```

```
    }
    //if it reaches 21, it must return 0
    if(counter_left_upper == 2 && counter_left_lower ==1 )
    {
      //sets counter as 00
      counter_left_lower = 0;
      counter_left_upper = 0;
    }
    prev = current; //updates time
  }
  //if 200ms passed, game not finished and it is turn for right player
  if (current - prev >= 200 && finished_flag ==0 && left_right_flag ==1) {
    //increments lower bit
    counter_right_lower++;
    //if it reaches to 10
    if(counter_right_lower == 10)
    {
      counter_right_lower = 0; //it sets lower bit as 0
      counter_right_upper++; //increments upper bit
    }
    //if it reaches 21, it must return 0
    if(counter_right_upper == 2 && counter_right_lower ==1 )
    {
      //sets counter as 00
      counter_right_lower = 0;
      counter_right_upper = 0;
    }
    prev = current; //updates time
  }
}
```

# 3 RESULTS

## 3.1 PART 1

After our code implementation, we observed our counting and reset effect.

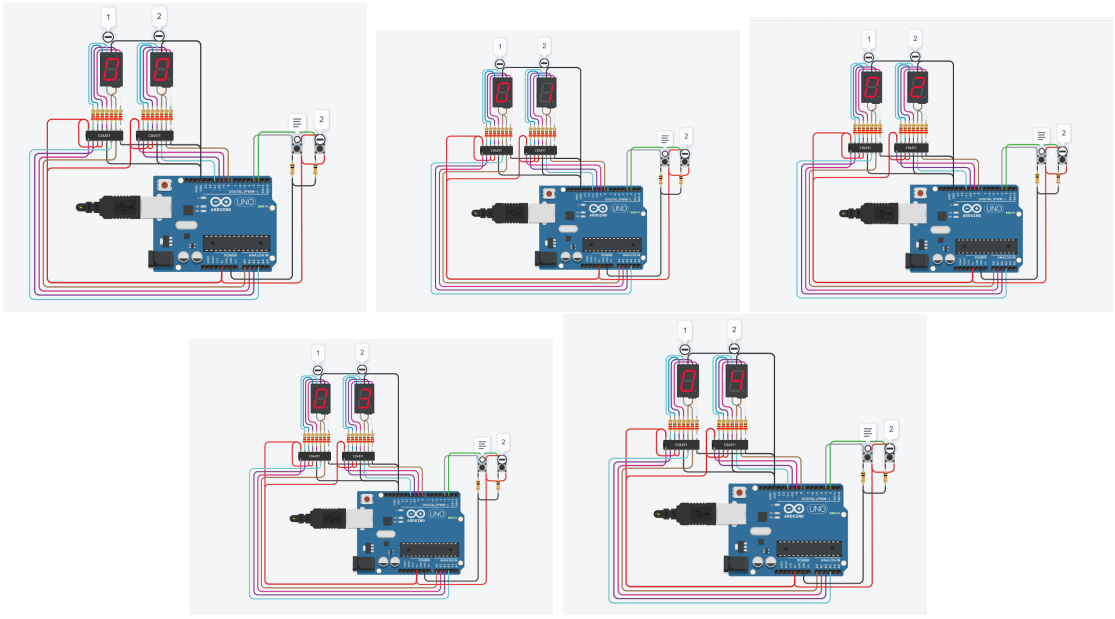

Figure 1: Up counting

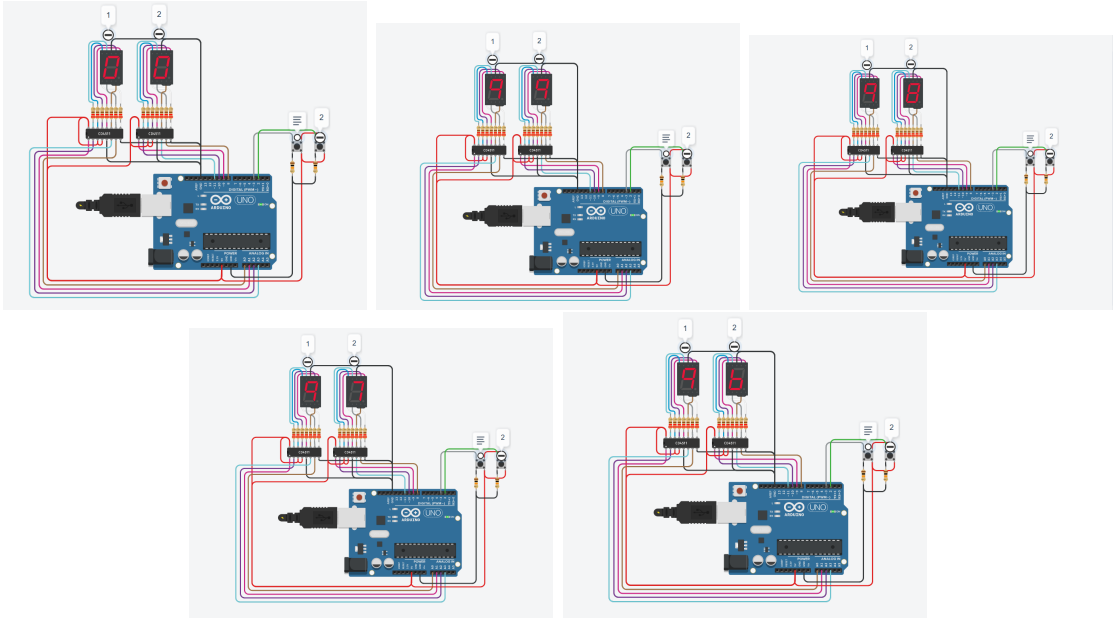

Figure 2: Down counting

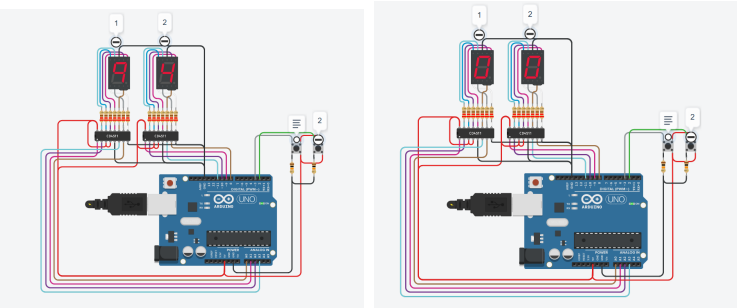

Figure 3: Reset

## 3.2 PART 2

After our code implementation, We can clearly play the game.

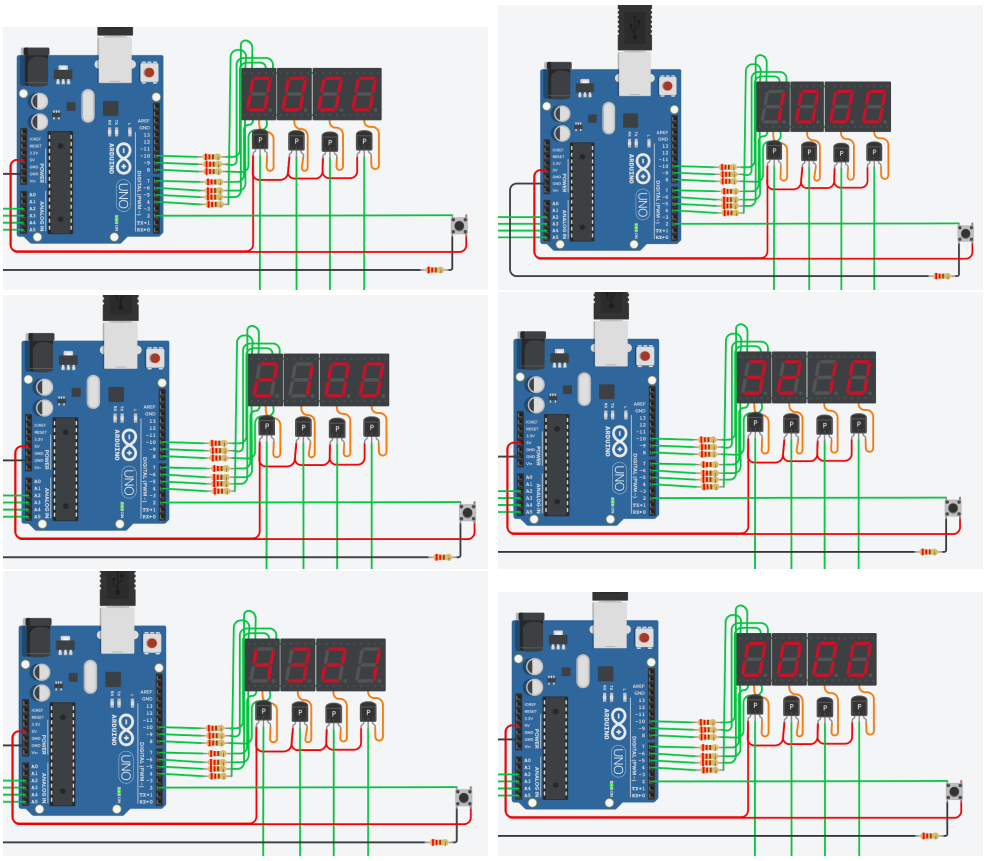

Figure 4: Shift of the counter's value [Last one is when the game is over(Target value is 321)]

## 3.3 PART 3

We have observed that our circuit works as we have expected. Game can be played easily.

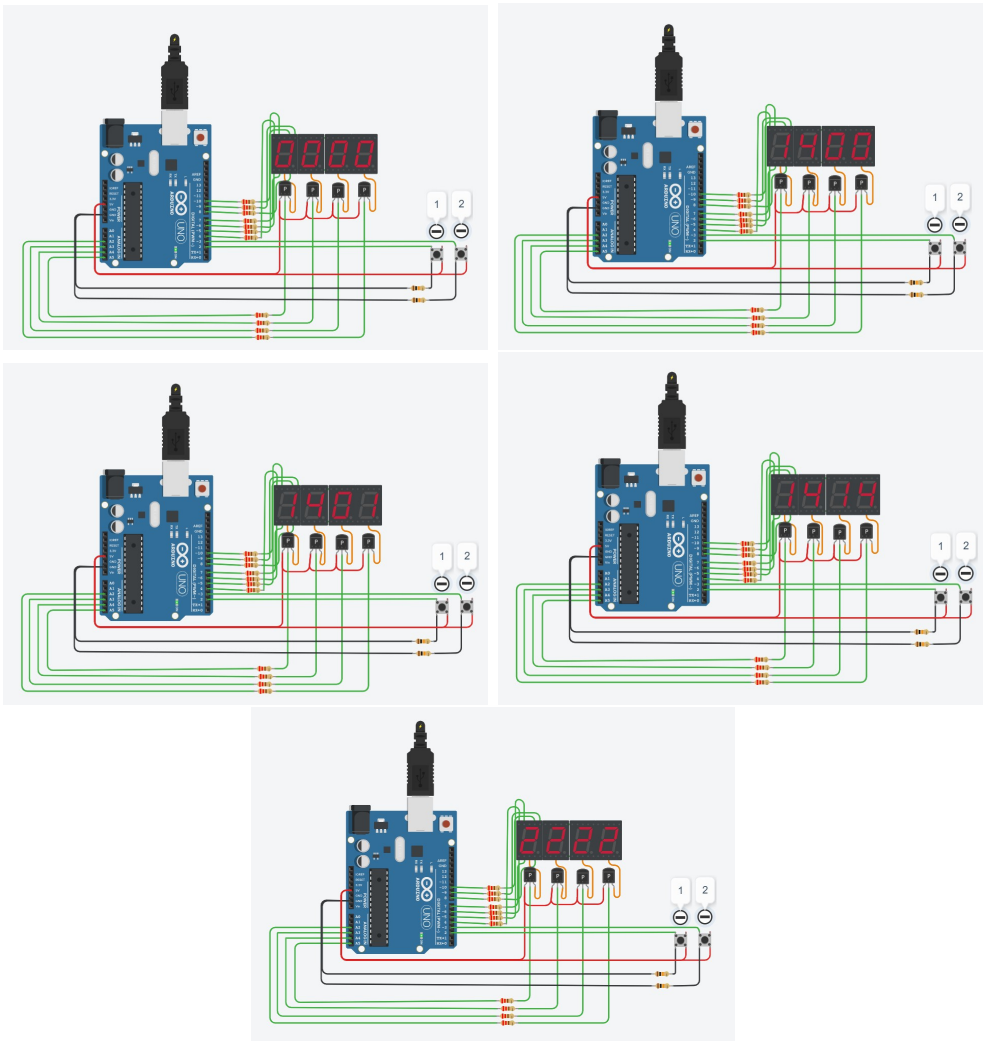

Figure 5: Simulation for right player win

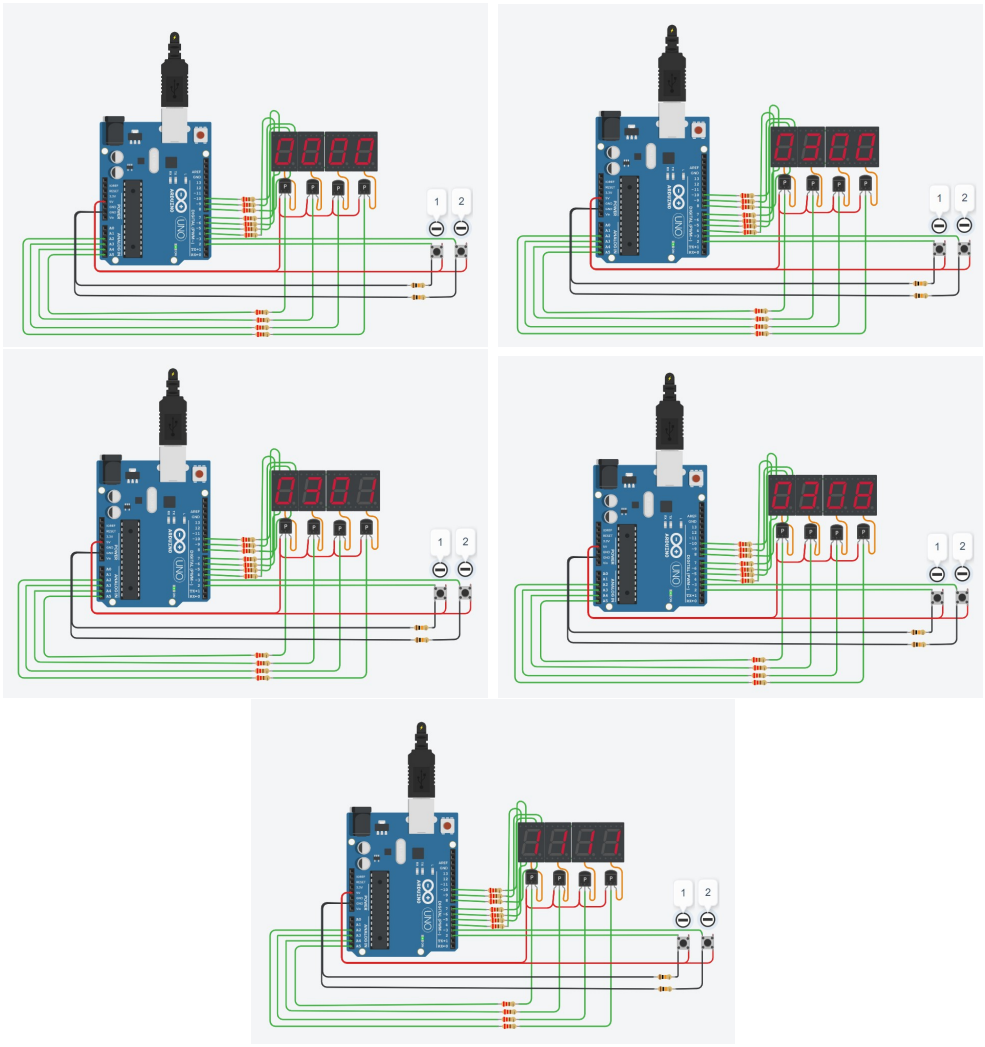

Figure 6: Simulation for left player win

# 4 DISCUSSION

## 4.1 PART 1

In this part of the experiment, we have implemented a circular, decimal up-down counter. We had reset and mode functions in our Arduino circuitry which are controlled by the buttons attached to pins 2 and 3. As opposed to previous we had to design interrupt cycles for our control inputs which was completely new to us.

The difficulty we have faced was to implement interrupt subroutine and provide delay effect because delay function was forbidden this week. After contacting with our assistants and some research we have found the required function to attach an interrupt subroutine and came up with a way to provide delay effect using millis. By checking the interrupt function's parameters we did not have a hard time designing the circuit thanks to our experience from previous weeks.

## 4.2 PART 2

In this part of the experiment, we are expected to implement a game. In this game, there is a target value and there is a counter that counts up 0 to F, when we press the button counter's value shifts to the right. With this mechanism we have to display all of the digits of the target at our seven-segment displays. We have encountered with some problems with this experiment especially at using millis function for delay. We tried some different ideas for the delay but it did not work as we expected. After some thinking we come up with a solution for this and we implemented our delay's with millis correctly.

This part of the experiment was especially challenging for us, we did everything we can to complete this part and we succeeded. It was good for us to be successfully implement this game.

## 4.3 PART 3

In this part of the experiment, we are expected to implement simple game that has a concept based on number matching. In part-2, we have faced with many problems about display operation. After solving our problems, it was not hard to implement main logic of the circuit. Actually, we have constructed our circuit on second part's code. As programming software, we have defined our needs to perform game operations and directly code them into circuit. After coding part, we thought about the edge cases of

the game such as pushing wrong button and trying to play game after it ends. We added this conditions to our code and we have obtained stable game.

It was good to see a working circuit that allows us to play a game. We think this part of experiment pleasant for both coding and testing phases. We have satisfied with our final output.

# 5 CONCLUSION

Achievement of this week's experiment is to make us learn about interrupt subroutines and reveal a better way of reading our inputs and manipulate the circuit. After learning about the interrupt subroutines we have designed some basic games with our Arduino circuitry. Thanks to this week's experiment we have widened our understanding of interrupt cycles in microcomputer systems.

# References

[1] Kadir Ozlem. *BLG 351E – Microcomputer Laboratory Slides*. 2020.

[2] Arduino. Arduino documentation. https://www.arduino.cc/reference/en/, 2018.

[3] Autodesk. Tinkercad. https://www.tinkercad.com/, 2011.