ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E

MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO : 5

EXPERIMENT DATE : 14.12.2020

LAB SESSION : MONDAY - 13.30

GROUP NO : G10

GROUP MEMBERS:

150180103 : EGEMEN GÜLSERLİLER

150180028 : ABDULKADİR PAZAR

150180716 : AHMET SELÇUK TUNÇER

AUTUMN 2020

Contents

1 INTRODUCTION

In this experiment, we created a counter and two number games using external interrupts on given circuit designs. One of the games is a number guessing game played by one player. The other game is a 2-player game in which player aims to catch the number selected by the other player.

2 MATERIALS AND METHODS

2.1 Part 1

```
attachInterrupt(digitalPinToInterrupt(3),reset,RISING);
attachInterrupt(digitalPinToInterrupt(2),other_way,RISING);
```

Figure 1: Functions used to run the interrupt subroutines

```
void reset()
{
    counter = 0;
    PORTB = counter;
    PORTC = counter;
}

void other_way()
{
    iterator = -iterator;
}
```

Figure 2: Interrupt subroutines

In this part we implemented a 2-digit decimal timer. The value of the timer changes every second. We used the attachInterrupt() function to run the appropriate subroutine when the correct button is pressed. When button 1 is pressed, counter is reset; when button 2 is pressed, counter changes counting mode between counting up and counting down.

2.2 Part 2

```
void pick()
{
  int guessed;
  if(!game_over)
  {
    if(lmost == -1)
    {
        lmost = counter;
    }
    else if(mid == -1)
    {
        mid = lmost;
        lmost = counter;
    }
    else
    {
        rmost = mid;
        mid = lmost;
        lmost = counter;
    }
    guessed = (lmost*256)+(mid*16)+rmost;
  }
  if (number == guessed)
  {
    game_over = true;
  }
}
```

Figure 3: Interrupt subroutine for button press

#	A	B	C	D	E	F	G	PORTB (A+B+C)	PORTD (D+E+F+G≪4)
0	0	0	0	0	0	0	1	0	16
1	1	0	0	1	1	1	1	4	240
2	0	0	1	0	0	1	0	1	32
3	0	0	0	0	1	1	0	0	96
4	1	0	0	1	1	0	0	4	192
5	0	1	0	0	1	0	0	2	64
6	0	1	0	0	0	0	0	2	0
7	0	0	0	1	1	1	1	0	240
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0	0	64
A	0	0	0	1	0	0	0	0	128
B	1	1	0	0	0	0	0	6	0
C	0	1	1	0	0	0	1	3	16
D	1	0	0	0	0	1	0	4	32
E	0	1	1	0	0	0	0	3	0
F	0	1	1	1	0	0	0	3	128

Table 1: Mapping logic for seven segment displays

In this part we built a number guessing game. We used the mapping logic explained in the table to display numbers in seven segment displays. Numbers from 0-F are displayed in leftmost seven segment display. Every time the button is clicked we enter the subroutine above. This subroutine shifts the numbers in seven segment displays to the right. In the end it compares the guess of the player with the number. If they are equal the game is won and the displays show the number.

2.3 Part 3

```
void pick1()
{
  if((player==1)&&(!game_over))
  {
    if(player_pick!=counter)
    {
      player_pick = counter;
      player = 2;
      counter = 1;
    }
    else
    {
      game_over = true;
    }
  }
}

void pick2()
{
  if((player == 2) && (!game_over))
  {
    if(player_pick!=counter)
    {
      player_pick = counter;
      player = 1;
      counter = 1;
    }
    else
    {
      game_over = true;
    }
  }
}
```

Figure 4: Interrupt subroutines for Part 3

In this part we created a 2-player game. Same mapping logic as part 2 is used to display numbers in seven segment display. At the beginning the when first player presses the button program enters the pick1() subroutine. In this subroutine the number player 1 picked is stored in player_pick variable, player changes from 1 to 2 and counter is reset to 1. After that player 2 will try to catch the player 1. When player 2 presses their button,

program enters pick2() subroutine. This subroutine is similar to pick1(). As long as one of the players don't manage to catch their opponents number, player_pick will be set to counter, player will change to their opponent, counter will reset. If one of the players manages to catch their opponent, all displays will show the player's number.

```
if(millis()%40 == 9)
{
    PORTC = 28;
    PORTB = bout[player];
    PORTD = dout[player];
}
if((millis()%40) == 19)
{
    PORTC = 44;
    PORTB = bout[player];
    PORTD = dout[player];
}
if((millis()%40) == 29)
{
    PORTC = 52;
    PORTB = bout[player];
    PORTD = dout[player];
}
if((millis()%40) == 39)
{
    PORTC = 56;
    PORTB = bout[player];
    PORTD = dout[player];
}
```

Figure 5: Code to display the winning player

3 RESULTS

3.1 Part 1

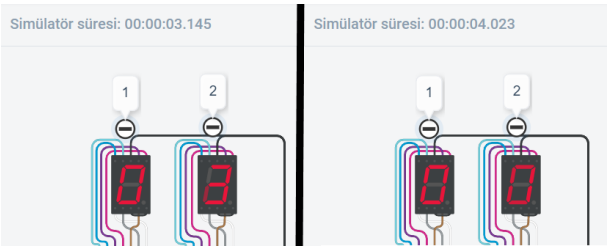

Figure 6: Counter reset functionality

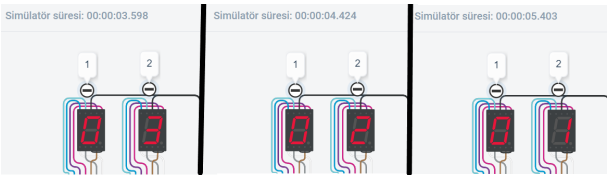

Figure 7: Counting mode change functionality

At the end of the first task, the team is succeeded in implementing a simple two-digit decimal counter using two seven-segment displays with desired functionalities. Implementation of this part is mentioned in detail at the material and methods section. Basically, program is implemented in a way that when first button is clicked the counter is reset to 0. Also, program has another functionality which is counting up and counting down modes which is controlled by second push-button.

3.2 Part 2

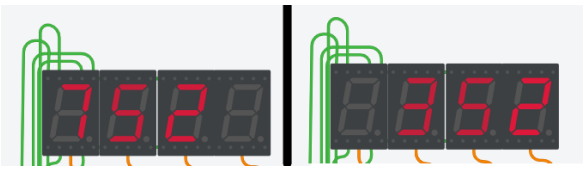

Figure 8: Displays during the game (left) Displays when the game is over (right)

In the second part, the team implemented a game in which the aim of users is to guess the target number (i.e., 3 digits long) by means of push button presses Implementation of this part is mentioned in detail at the material and methods section. Simply, 4-bit counter (0-F) is displayed on the left-most 7-segment display and takes input from user. When user make a selection, the selected number is shown on the 7-segment display on the right. When selected number matches with the target, game ends.

3.3 Part 3

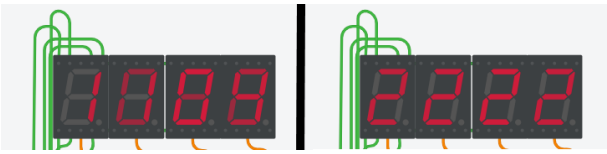

Figure 9: Displays during the game (left) Displays when the game is won by p2 (right)

In the last part of the experiment, a 2-player number game using push-buttons was developed as required. The aim of the game was to try and catch the number of the other player. At first we created a counter that counts continuously from 1 to 20 with a 200 ms delay. When the first player presses the button, the selected number is shown on two left-most 7-segment display. After that the second player will press the second button to catch the number chosen by the first player. If he catches, then player 2 wins the game. Otherwise, the first counter will start and the first player try to catch the number chosen by the second player. When the game is over, the winner of the game is indicated on all 7-segment display in the form of "1111" or "2222".

4 DISCUSSION

In the first part of the experiment, attachInterrupt() function which is provided by the Arduino environment and reset() and other_way() functions which are written by us are implemented easily and these made our task easy because the methods that are we are expected to achieve was easy to handle compared to other tasks. And also we use millis() delay function instead of delay().

In second part of the experiment, we consumed great portion of our time to understand the rules of the game. After we figured out the game, we handled the task. Rest of the experiment is a little bit straightforward and we handled successfully.

The last part of the experiment is similar with the part 2 in regards to the implementation. At first we determined the scenarios and then implemented and then we mostly use if statements and port manipulations that used in part 2. It took less time to understand but more time to handle because there was a little bit longer to code.

5 CONCLUSION

In this experiment we learned how an interrupt can stop the current process and start another one. We were mainly concerned about how could we keep our seven segment displays in a stable state so that we could observe the numbers clearly. Also especially in Part 2 we struggled a bit to understand and implement the game logic. This experiment overall was a good practice to understand the basics of I/O systems.