ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E

MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO : 8

EXPERIMENT DATE : 15.01.2021

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 created a random number generator using the **additive lagged fibonacci generator** algorithm in our Arduino circuitry. After providing the randomized generation we have displayed our random values on seven segment display and inspected the randomization.

2 MATERIALS AND METHODS

- Tinkercad
- 1 Arduino Uno R3
- 1 Seven Segment Display
- Jumper Cables

2.1 PART 1

In this part of experiment, we are expected to implement random generator circuit using Additive Lagged Fibonacci Generator (ALFG) method and we have to display our random number in seven-segment display with the help of button interrupt.

Additive Lagged Fibonacci Generator: It is one of the type of random generation methods and it is based on Fibonacci Sequence principles. Firstly, user have to define a seed value randomly and select j, k values then according to formula given below, the generation is performed.

$$S_n = S_{n-j} + S_{n-k} \ (mod\ m),\ 0 < j < k$$

By using this formula, we are trying to generate random number by summing k^{th} and j^{th} terms of our seed. The m value allows us to determine upper limit of our number. After generation, we have to shift our seed value to left and place new generated number to end of the seed. By repeating this operation, we can generate different random variables.

In the implementation phase, we have converted our theoretical knowledge into a Arduino code. Firstly, we have defined our seed, k, j and m values. We have determined our seed value as 2461378059. To obtain easier manipulation of seven-segment display, we have defined two arrays that keep bit combinations for numbers.

In the setup phase, we have defined PORTD and PORTB as output except 3^{rd} pin of PORTD. Also, we have attached interrupt to 3^{rd} pin of PORTD and it associates with generate function.

In generate function, we transform our Additive Lagged Fibonacci Generator into code. We have generate our random number using j and k. Then, we have used for loop and it shifts all elements of array to left. We have added our random number to end of the array for next generation. At the end, we display our number in seven-segment display.

Our implementation is given below.

```
//random seed array with 10 size
int seed[10] = {2, 4, 6, 1, 3, 7, 8, 0, 5, 9};

int k = 2; //k value is given as 2
int j = 1; //0<j<k
int m = 16; //random number interval

// 7 segment number binary codes for PORTB
byte sevensegment0[16] = {
  0b01111,
  0b00001,
  0b10110,
  0b10011,
  0b11001,
  0b11011,
  0b11111,
  0b00001,
  0b11111,
  0b11011,
  0b11101,
  0b11111,
  0b01110,
  0b10111,
  0b11110,
  0b11100,
};
```

```
// 7 segment number binary codes for PORTD
byte sevensegment1[16] = {
  0b11,
  0b10,
  0b11,
  0b11,
  0b10,
  0b01,
  0b01,
  0b11,
  0b11,
  0b11,
  0b11,
  0b00,
  0b01,
  0b10,
  0b01,
  0b01,
};

//random generator function initialized
void generate();
void setup()
{
    DDRD = 0b11110111; //3rd pin set as input, others are output
    DDRB = 0b111111; //all pins are set as output
    //initialize display with 0
    PORTD = sevensegment1[0] << 6;
    PORTB = sevensegment0[0];
    //attach rising interrupt to 3rd pin of PortD
    attachInterrupt(digitalPinToInterrupt(3), generate, RISING);
}

void loop()
{
        // void loop must stay empty
}
```

```
//random number generator (interrupt function)
void generate()
{
    //gets jth and kth elements of array and takes modulo with 16
    //generates number between 0–m
    int random = (seed[j - 1] + seed[k - 1]) % m;
    //shifts all elements of array to left
    for(int i = 0; i < 9; i++)
    {
        seed[i] = seed[i + 1];
    }
    //places our random number as last element of array
    seed[9] = random;
    //displays random value in seven segment display
    PORTD = sevensegment1[random] << 6;
    PORTB = sevensegment0[random];
}
```

2.2 PART 2

In this part of experiment, we are expected to implement random generator that generates numbers between 0-8 m times. We have to use Serial communication to get m value from the user. When user presses to button, we have to print frequency distribution of our randomly generated numbers.

In setup phase, we start Serial communication and we ahve attached a interrupt which associates with generate function to 3^{rd} pin of PORTD. The generate function, basically reads input from Serial Input, generates random number and prints frequency distribution of these numbers. If there is a input when the button is pressed, it starts to read input and stores it in string format. After read operation, it casts string into integer type. Then it calls randomNumberGenerator function. After returning this function it prints distribution of random numbers and delocates array.

In randomNumberGenerator function that takes m value as a argument, it allocates array for randomly generated functions and it generates m number based on Additive Lagged Fibonacci Generator method between 0-8 within the for loop. After each

generation, it stores number in a array and it updates frequency array according to new generated number.

Our implementation is given below.

```
//random seed array with 10 size
int seed[10] = {2, 4, 6, 1, 3, 7, 8, 0, 5, 9};

int k = 2; //k value is given as 2
int j = 1; //0<j<k
int m = 0; //number of random number that will be generated

int* array; //pointer for generated numbers
int freqArray[8] = {0}; //frequency distribution array

// 7 segment number binary codes for PORTB
byte sevensegment0[16] = {
  0b01111,
  0b00001,
  0b10110,
  0b10011,
  0b11001,
  0b11011,
  0b11111,
  0b00001,
  0b11111,
  0b11011,
  0b11101,
  0b11111,
  0b01110,
  0b10111,
  0b11110,
  0b11100,
};
// 7 segment number binary codes for PORTD
byte sevensegment1[16] = {
  0b11,
  0b10,
```

```
  0b11,
  0b11,
  0b10,
  0b01,
  0b01,
  0b11,
  0b11,
  0b11,
  0b11,
  0b00,
  0b01,
  0b10,
  0b01,
  0b01,
};

//random generator function initialized
void generate();

void setup()
{
    Serial.begin(9600);
        //attach rising interrupt to 3rd pin of PortD
        attachInterrupt(digitalPinToInterrupt(3), generate, RISING);
        //Print prompt message
        Serial.print("Please enter a m value:");
}

void loop()
{
        // void loop must stay empty
}


//function that generates m random numbers between 0-8
void randomNumberGenerator(int m)
{
```

```
  //allocates required space for numbers that will be generates
  array = (int*) malloc(m * sizeof(int));
  //iterates m times
  for(int j = 0; j < m; j++)
  {
    //gets jth and kth elements of array and takes modulo with 8
    int random = (seed[j - 1] + seed[k - 1]) % 8;
    //shifts all elements of array to left
    for(int i = 0; i < 9; i++)
    {
    seed[i] = seed[i + 1];
    }
    //places our random number as last element of array
    seed[9] = random;
    //push random number to array
    array[j] = random;
    //increment count value
    freqArray[random]++;
  }
}

void generate()
{
  if(Serial.available() > 0){
    String numstr = ""; //string that will be read from serial input
    byte num; //char that will be read from serial input
    int size = 0; //input size variable
    //until read all serial inputs
    while (Serial.available() > 0) {
      //reads one char from input
      num = Serial.read();
      //if it is number
      if ((num >= '0') && (num <= '9')) {
        //casts to int and add to numstr
        numstr += num - 48;
      }
      size++; //increment size
```

```
    }
    Serial.println(numstr);
    int pow = 1; //it keeps powers of 10
    //iterates over the string that is read
    while(size > 0)
        {
          m += (numstr[size - 1] - 48)*pow; //adds number to
          size--; //decrement size
          pow*=10; //multiply pow with 10
        }
        //call random number generator
        randomNumberGenerator(m);
        //it iterates over the freqArray
        for(int i = 0; i < 8; i++)
        {
          //it prints frequencies of numbers
          Serial.print(i);
          Serial.print(" -> ");
          Serial.println(freqArray[i]);
          //sets frequency to 0
          freqArray[i] = 0;
        }
        //frees space
        delete [] array;
        }
  Serial.print("Please enter a m value:");
}
```

3 RESULTS

3.1 PART 1

After our code implementation, we checked the observed values' randomization. As expected we were able to create pseudo-random number using our algorithm and display those numbers on 7 segment display.

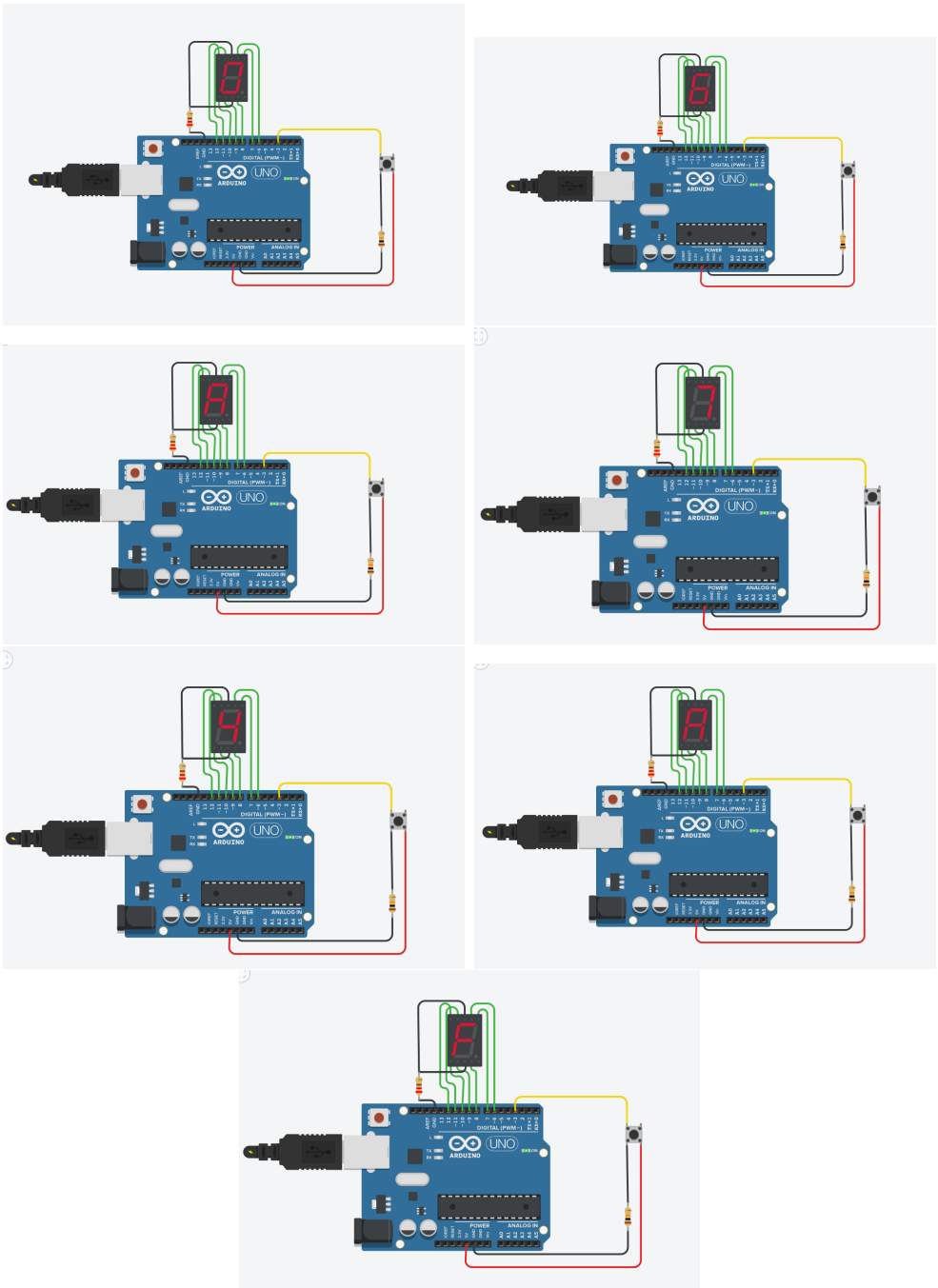

Figure 1: Randomly generated numbers in hexadecimal form.

3.2 PART 2

After our code implementation we can easily see that when our m value gets bigger our algorithm is more randomized. And also when m gets bigger our distribution is more balanced.

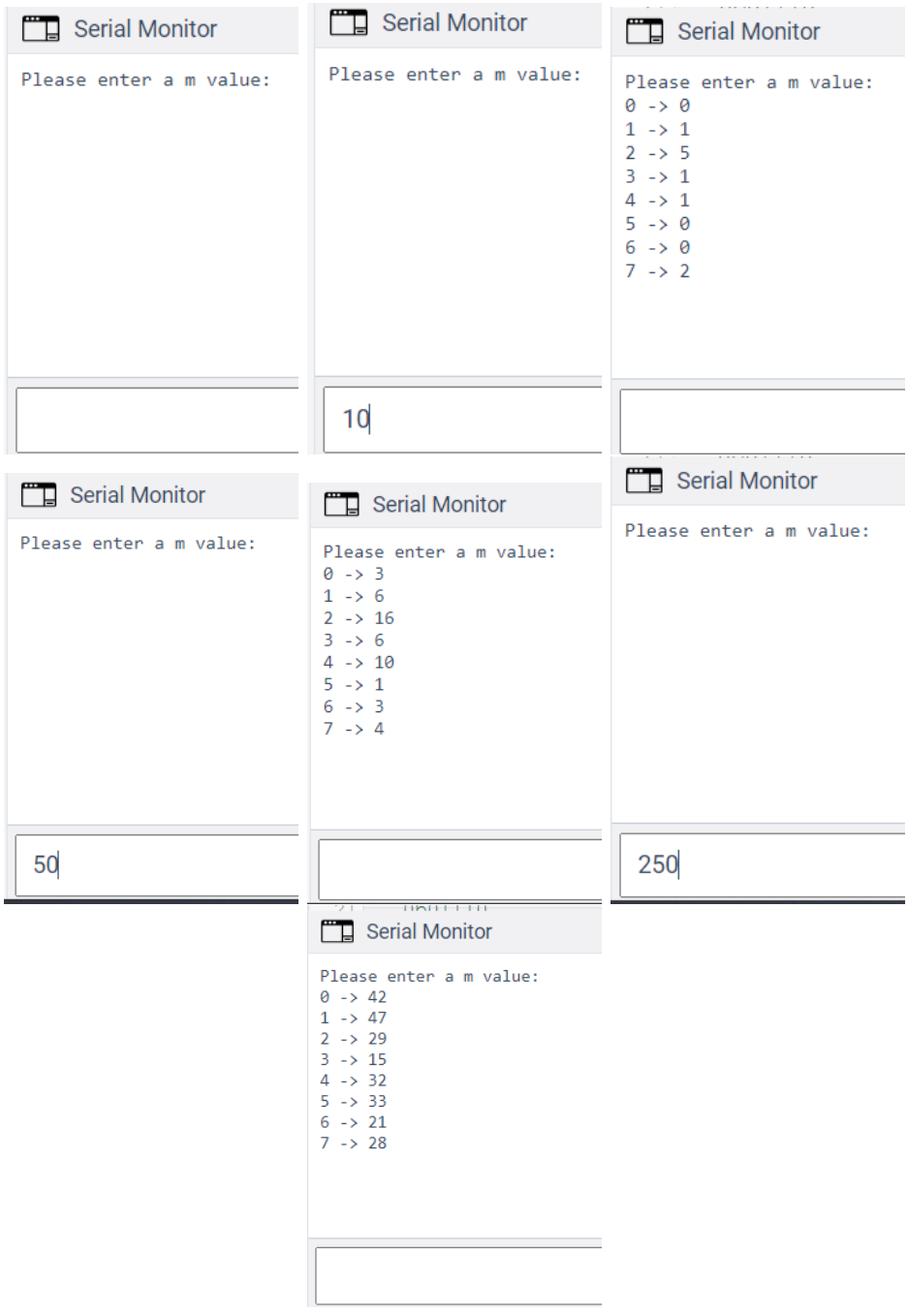

Figure 2: Number of occurrences of the numbers between 0 and 8 (8 is not included)

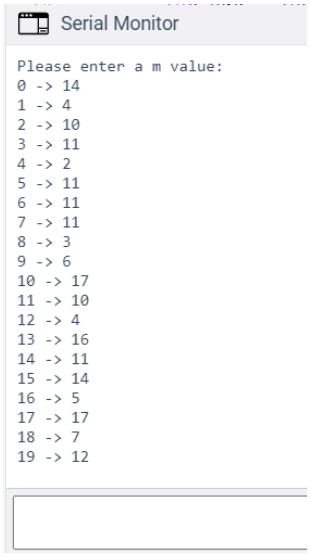

Figure 3: ALFG algorithm on different intervals i) 0-20, m=200

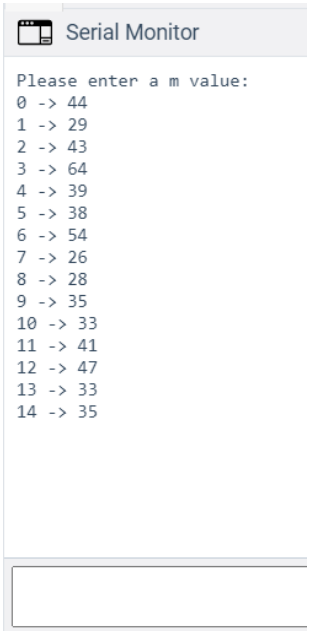

Figure 4: ALFG algorithm on different intervals i) 0-15, m=600

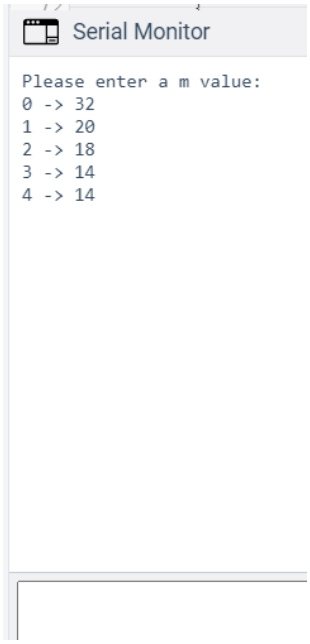

Figure 5: ALFG algorithm on different intervals i) 0-5, m=100

4 DISCUSSION

4.1 PART 1

In this part of the experiment, we had to learn about Additive Lagged Fibonacci Generator (ALFG) to implement our random number generator function. When we looked at the Wikipedia page about ALFG we didn't understand much of it but we slowly become familiar with the concept. We had to look at some examples to understand the concept. We learned that there is a seed variable that we can use to generate our random variables from. And we learned from these examples that this algorithm is not only additive but because we use "+" we call it Additive Lagged Fibonacci Generator. We easily learned that our j and k values are indexes from this seed that we can add each other and generate our numbers. With these new information about ALFG we easily created our function and it worked perfectly.

4.2 PART 2

In the second part of our experiment we have continued to use our ALFG algorithm for random number generation. As opposed to first part of the experiment, we wrote our random values to a dynamically created array and checked how random the values were by taking the number of random numbers to be generated by the user. After our implementation in the first part, we did not have much of a hard time with second part

of the experiment thanks to our Arduino skills that have been developing all these weeks with the ”Microcomputer Lab.“ course.

5 CONCLUSION

In conclusion, in this experiment we have learned Additive Lagged Fibonacci Generator algorithm that can generate random numbers in an given interval in a pseudo-randomized fashion. Learning this algorithm provided us an advantage in the software side of our course. We have gained a better understanding of random number generation. Thanks to many weeks of development on Arduino circuitry with our Microcomputer Lab. course we were able to conduct the final experiment easily.

We have reached end of a long and intense semester and our laboratory journey, both this semester and the previous semester we have developed our report writing skills and problem solving methodologies on hardware side. To be honest, laboratory experiments were really hard (We are happy we could not get a quiz.) but it was definitely educational. We encountered many problems that we saw for the first time (Assembly, Verilog, Arduino...) and solved them with a teamwork. Thank you for your efforts and thoughts. We will miss writing laboratory report...

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.