IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-6, June 2019 E-ISSN: 2347-2693

A Brief Study and An Arduino Based Implementation of Booth's Multiplication Algorithm

Kushankur Ghosh[1*], Arghasree Banerjee[2]

[1,2] Dept. of Computer Science, University of Engineering and Management, Kolkata, India

Corresponding Author: kush1999.kg@gmail.com

DOI: https://doi.org/10.26438/ijcse/v7i6.307313 | Available online at: www.ijcseonline.org

Accepted: 10/Jun/2019, Published: 30/Jun/2019

***Abstract*—** The main objective behind this paper was to provide a brief overview about some of the well-known multiplication algorithms and design an Arduino based calculator working on the principles of the Booth's multiplication Algorithm in the simplest and cheapest possible way. Our study surrounds some of the algorithms that existed before and around the time when Booth's multiplication Algorithm came into the scene. The study found that each of the algorithms had some problems which ultimately galvanized the popularity of the Algorithm proposed by Arnold Donald Booth. The calculator we designed is capable of taking a four-bit binary multiplier and a four-bit binary multiplicand as input and will produce the product in an eight-bit sequence as the output. The calculator is basically constructed with the help of a central controlling Arduino connected to various input and output specified electronic components such as push buttons and LEDs. The circuit constructed to perform the algorithm is the simplest possible circuit. The input to the Arduino can be given from the hardware itself and the output also gets reflected through the hardware.

***Keywords*—**Arduino, Booth's Algorithm, Multiplier, Multiplicand, Internet of Things, Multiplication algorithms, Calculator, Algorithm, LSB, MSB, Partial Product, Left Shift (LS), Right Shift (RS)

I. INTRODUCTION

The manual method of multiplication of two integers suggests a method of repeated bit to bit additions and holds the commutative property for every addition. The multiplication of two numbers signifies the repetitive addition of the multiplicand. The number of times the multiplicand is added is specified by the multiplier. The process of multiplication itself satisfies:

- Commutative Property : A.B = B.A
- Associative Property : (A.B).C = A.(B.C)
- Distributive Property : A.(B + C) = A.B + A.C

Sequential Multiplication Algorithm could perform multiplication between two unsigned numbers by performing repeated additions and right shift operations. In 1950s, Arnold Donald Booth invented a special multiplication algorithm at Birkbeck College in Bloomsbury, London which was proved to be an efficient method for the multiplication for signed numbers and was called as Booth's Multiplication Algorithm.

When the operands are integers, the product in general is twice the length of operands in order to protect the information content [2]. In our experiment we have applied the algorithm for data inputs of size four bits which can produce the output in an eight-bit binary sequence. Booth's algorithm stores two inputs and deals one of them as the multiplier and the other as the multiplicand. Following the properties of multiplication, Booth's algorithm also supports the commutative property for its operands. It means that any of the given input can be used as a multiplier or a multiplicand and the result will be same for both the cases.

The model that we constructed also supports the commutative property of multiplication. Here the user of the device is free to choose the multiplier and multiplicand. The constructed model is a minimized version of a multiplying calculator based on Booth's algorithm. The model consists of an Arduino Uno board which is the central controller of the whole operation. It has a Push to On Button which provides the input data to the Arduino for calculation. Eight LED (Light Emitting Diode) are used to represent the output via the hardware circuit.

II. LITERATURE SURVEY

A. Array Multiplier

In the manual method of binary multiplication the multiplicands are repeatedly shifted left wise via Left Shift(LS) operation and then added. If we consider a four bit multiplicand say M = $(2)_{10}$ = $(0010)_2$ and a four bit multiplier say M_L = $(3)_{10}$ = $(0011)_2$ then the manual multiplication method can be implemented in the following way:

```
                    0  0  1  0
                    0  0  1  1
  ------------------------------------------
                    0  0  1  0
                 0  0  1  0
              0  0  0  0
           0  0  0  0
  ------------------------------------------
=        0  0  0  0  1  1  0
```

The obtained output is *$(0000110)_2$*. This process of multiplication can be performed by a 4x4 Array Multiplier circuit. The circuit can be constructed with n number of AND gates and Full Adders(FA). For *n-bit*. Multiplier, it requires n^2 AND gates [4]. The detailed architecture of the circuit is described in the Figure 1.

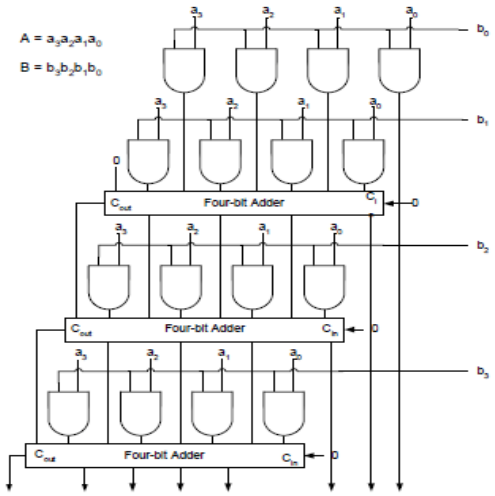


Figure 1. 4x4 Array Multiplier Circuit [4]

B. Sequential Multiplication Algorithm

The algorithm proposes a method which performs repetitive addition and Right Shift(RS) operation in order to obtain the final result. Here a register R is declared which stores values that vary from one iteration to the next. The size of the values stored by the R register which is also known as the Accumulator, are kept equal to the size of the multiplier and the multiplicand. The M register holds the multiplicand and the M_L register holds the multiplier value in binary sequence. The flip flop register F is used to store the carry generated in the addition and is used as the serial input when the register pair R and M_L is shifted right one position by RS operation. The sequential multiplication holds a copy of the partial products. If a new partial product comes then it will be added to the old partial product. The RS operation can be defined with the following example:

If we have to perform RS operation in a binary sequence $(1010)_2$, the shift will take place in the following way:

```
1  0  1  0
 ↘  ↘  ↘  ↘
0  1  0  1  X
-------------
=  (0 1 0 1)2
```

Therefore we can see that on performing the RS operation on $(1010)_2$ we obtained $(0101)_2$ as the result. This operation is used repeatedly in this algorithm. This shift and addition process will run for n number of times where n is lesser than or equal to the length of the multiplier M_L.

Algorithm 1: Sequential _Multiplication(M, M_L)
//The value of R is initially kept $(0000)_2$;
//The value of F is initially kept 0;
//SIZE = length of M_L;
//RightShift() performs the right shift operation;

```
1.  while SIZE > 0:
2.      if ML[0] = 1
3.          then perform binary addition between R and M
4.          R = R + M
5.      endif
6.      Perform Right Shift operation on the combined
        value F, R and ML placed side by side in the given order
7.      RightShift(F R ML)
8.      SIZE = SIZE – 1
9.  endwhile
10. Result = (RML)
```

The Sequential Multiplication algorithm is efficient for unsigned binary numbers. For signed numbers the sign of the result is determined by performing XOR operation between the first bit of the multiplier and the multiplicand.

C. Booth's Multiplication Algorithm

Booth's Algorithm was invented by A.D. Booth in 1950 and which could efficiently perform multiplication between two signed binary numbers using 2's complement notation. Booth multiplication is a fastest technique that allows for smaller, faster multiplication circuits, by recoding the numbers that are multiplied [5]. Booth Multiplier reduces number of iteration step to perform multiplication as compare to conventional steps thus providing a very smooth way of calculation [6]. In sequential multiplication, four additions

are required for a string of four 1s as input but these four additions can be replaced by one addition and one subtraction in booth's algorithm [3].

The algorithm examines the multiplier(M_L) or the multiplicand(M) and converts them in 2's complement notations if they are in signed binary sequences. Similar to the Sequential Multiplication algorithm, the Booth's algorithm also deals with M_L, M, R(Accumulator Register) and a flip flop Q_{-1} which is written just beside the LSB(Least Significant Bit) of M_L. The LSB of M_L is written as $M_L(0)$. The value of the R is changed by subtracting or adding it with the value of the multiplicand or kept constant depending upon the value of $M_L(0)$ and Q_{-1}. After the required addition or subtraction, Arithmetic Right Shift(ARS) is performed on the combined value of R, M_L and Q_{-1} written in the order of R M_L Q_{-1}. The diagram of Figure 2 represents the flowchart of Booth's Algorithm.

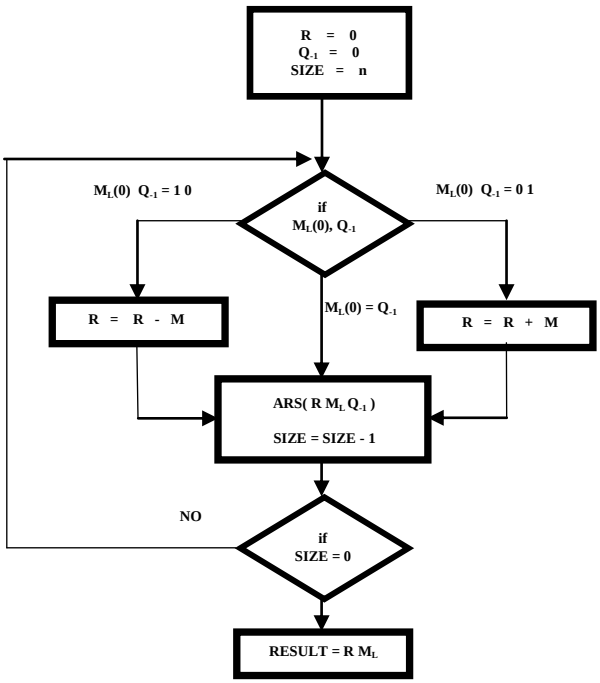
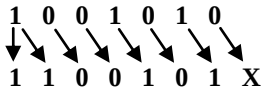


Figure 2. Booth's Algorithm Flowchart

The ARS operation on any binary expression results in the right shift of each bit in the expression and the sign bit replicates itself as the sign bit of the result. If we consider to perform the ARS operation on a binary sequence say $(1001010)_2$, it will take place in the following way:



$= (1\ 1\ 0\ 0\ 1\ 0\ 1)_2$

Therefore after performing the ARS operation on the expression we will get $(1100101)_2$ as the result. Here we see that the ARS operation is similar to that of normal RS operation except the fact that in ARS operation the sign bit or the MSB repeats itself as the MSB or the sign bit in the result also.

Algorithm 2: Booth_Multiplication(M, M_L)
//The value of R is initially kept $(0000)_2$;
//The value of Q_{-1} is initially kept 0;
//SIZE = length of M_L;
//ARS() performs the Arithmetic Right Shift operation;
//COMP() performs 2's complement;

```
1.  while SIZE > 0:
2.      if ML = 0 and Q-1 = 1
3.          then perform binary addition between R and M
4.          R = R + M
5.      elseif ML = 1 and Q-1 = 0
6.          then perform 2's complement of M and then
    perform binary addition between the obtained value and
    R
7.          MC = COMP( M )
8.          R = R + MC
9.      end if
10.     Perform Arithmetic Right Shift operation on the
    combined value of R, ML and Q-1 placed side by side in
    the given order
11.     ARS( R ML Q-1 )
12.     SIZE = SIZE – 1
13. endwhile
14. Result = (RML)
```

On average, the speed of doing multiplication with Booth's Algorithm is almost is the same as with the manual method of multiplication having very minor differences [7].

III. METHODOLOGY

This section of the paper describes our approach to construct the Arduino based calculator based on the Booth's algorithm. The model is constructed with very less number of electronic devices to make the circuit less complex and cheap. The hardware components required for the model are as follows:

- *Arduino UNO board (Quantity-1)*
- *Jumper Cables*
- *2-Pin Tactile Switch Micro – Push to On Button (Quantity-1)*
- *5 mm Multi Colour LED(Light Emitting Diode) (Quantity-8)*
- *1 K ohm Resistors (Quantity-8)*
- *Bread Board (Quantity-1)*

The model comprises of a Hardware part which is constructed with the help of the above mentioned circuit materials and a software part which is nothing but a set of instructions which chalks out the Booth's algorithm in the Arduino.

The Arduino UNO board is an open source hardware system which is used as the central computing device in this model. The board is trained with the Booth's Multiplication algorithm via Arduino IDE(Integrated Development Environment). The IDE is an application which runs on most of the Operating Systems. It is the platform where all the programming part of the model is done. The IDE supports C and C++ languages. In our model Arduino is used as the central hub for all the mathematical and logical operations required to implement the algorithm.

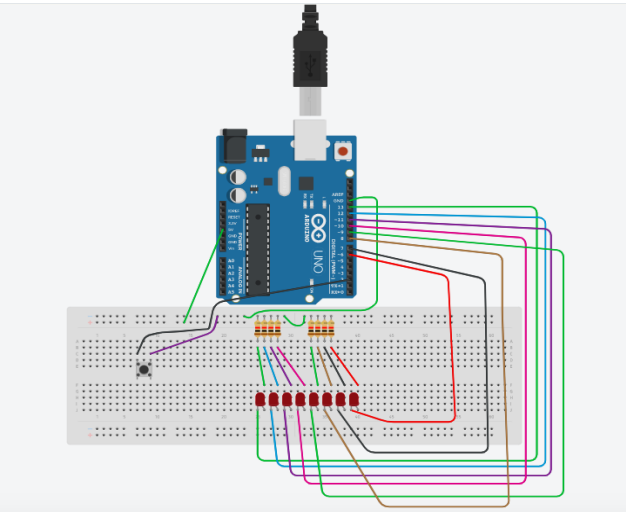


Figure 3. Proposed Circuit Diagram

The image of Figure 3 describes the circuit diagram of our model. The principle goal of our experiment was to construct an interactive device which could perform both input and output through both hardware and software. The input can be provided from the circuit itself with the help of one 2-pin Push to On Button. More than one button could be used for the implementation, but our goal was to reduce the circuit complexity. The Jumper cables are used to connect the different circuit elements with the Arduino. These are basically the connecting wires used to transfer the data from one part of the circuit to another. The Button can be called as the key element of our model. It is the main medium through which the programmer or the user can transfer data into the Arduino board. In our model this data is basically the Multiplier and the Multiplicand provided by the user. The internal *Pullup Resistors* of the Arduino has a rating around 50 K-ohm. These resistors can be used easily just by calling them while declaring the input function. The push button here is connected to the *Pullup Resistors* of the Arduino. This connection has proved a very smooth and hazardless performance of the model while giving any input. Connecting the button with the *Pullup Resistor* actually connects a 50 K-ohm resistor between the button pin and the +5 v supply of the Arduino. One leg of the button connects to the Pin 2 of the Arduino and the other leg connects to the GND pin of the Arduino. The input is provided to the Arduino through Pin 2 bitwise. When the push button circuit is open which means that it is not pressed there is no connection between its two legs as the internal pullup resistor is active and connected to +5 v supply of the Arduino. This means that when the button is not pressed the state of the button is HIGH and when it is pressed the state becomes LOW as the button gets connected to the ground. In our model when the button is pressed repeatedly to enter the binary equivalents of the Multiplicands and the Multiplier bitwise. When the button is pressed '1' is given as the input to the Arduino and else '0' is passed.

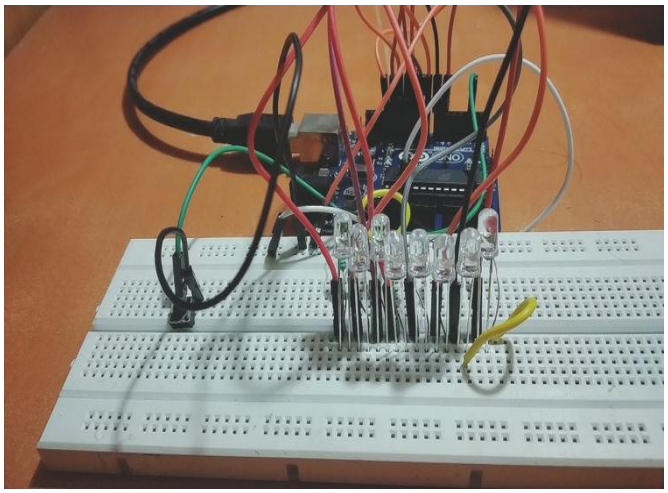


Figure 4. Actual Circuit

Eight LEDs are used to display the output via hardware. In our model eight LEDs are used to display the output of eight-bit binary sequence. The model takes 4-bit Multiplicand and 4-bit Multiplier as input and so, it will produce the result in 8 bit.

A single 1K ohm resistor is connected to the negative pin for each LED which is further connected to the GND pin of Arduino. A single Bread board is used to build the entire circuit. The bread board is the only platform where the entire circuit stands.

Proposed Algorithm: Booth_Multiplication_Implement( )
//A[ ] is an array representing the accumulator register;
//M[ ] is an array representing the Multiplicand;
//MC[ ] is an array storing the value obtained after 2's complement of M;
//Q[ ] is an array representing the Multiplier;
//Q1 represents the temporary Flip Flop bit which is initially kept 0;


```
//SIZE = length of M (SIZE = 4 for our Model);
//ARS() performs the Arithmetic Right Shift operation and returns the value;
//COMP() returns 2's complement;
//RCA() returns the sum after performing Ripple Carry Adder;
//RES[ ] array stores the result;
//Bstate returns the state of the button;
//STORE() stores the result in the RES[ ];
Button = input;
LED[ ] = output;
while (True)  (*Run the infinite Loop*)
  A = [0,0,0,0];
  Q1 = 0;
  for i = 1 to 4
    input the binary equivalent of the multiplicand bitwise
    if  Bstate = LOW
      then input '1' to M[i]
      M[ i ] = 1;
    else  M[ i ] = 0;
    endif
  endfor
  Press the button to input the sign bit for the multiplicand
  if  Bstate = LOW (*Sign bit = 1)
    then  store the value returned after 2's complementing M in M
    M = COMP(M);
  Endif
  Perform 2's complement of M and store it in MC
  MC = COMP(M);
  for i = 1 to 4
    input the binary equivalent of the multiplier bitwise
    if  Bstate = LOW
      then input '1' to Q[i]
      Q[ i ] = 1;
    else  Q[ i ] = 0;
    Endif
  endfor
  Press the button to input the sign bit for the multiplier
  if  Bstate = LOW (*Sign bit = 1)
    then  store the value returned after 2's complementing Q in Q
    Q = COMP(Q);
  endif
  for SIZE = 4 to 1
    if Q[3] = 0 and Q1 = 1
      then perform binary addition between A and M and store the value in A
      A = RCA(A, M);
    if Q[3] = 1 and  Q1 = 0
      then perform binary addition between A and MC and store the value in A
      A = RCA(A, MC);
    endif
    ARS(A, Q, Q1);
  endfor
  STORE(A, Q);
  if  RES[0] = 1
    then perform 2's complement on RES and store the result in
    RES = COMP(RES);
  endif
  Light_LED( );
endwhile

Light_LED( ) (*Displays the result via hardware by switching the corresponding LEDs*)

if  RES[ 0 ]=1
  LED[ 0 ] = HIGH;
endif
if  RES[ 1 ]=1
  LED[ 1 ] = HIGH;
endif
if  RES[ 2 ]=1
  LED[ 2 ] = HIGH;
endif
if  RES[ 3 ]=1
  LED[ 3 ] = HIGH;
endif
if  RES[ 4 ]=1
  LED[ 4 ] = HIGH;
endif
if  RES[ 4 ]=1
  LED[ 4 ] = HIGH;
endif
if  RES[ 5 ]=1
  LED[ 5 ] = HIGH;
endif
if  RES[ 6 ]=1
  LED[ 6 ] = HIGH;
endif
if  RES[ 7 ]=1
  LED[ 7 ] = HIGH;
```

IV. RESULTS AND DISCUSSION

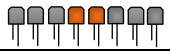
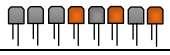
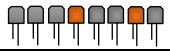
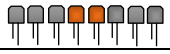
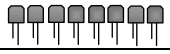
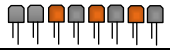
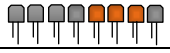
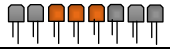
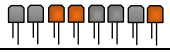
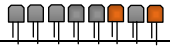
The result of the experiment can be obtained via both hardware and software. Eight LEDs either HIGH or LOW represents the eight-bit binary sequence obtained as the result. The input is supplied with the help of the push button. The experiment is conducted on a 1.8 GHz Intel Core i5 machine and the coding part is done using Arduino IDE version 1.8.9. The procedure of inputting the binary sequences of the multiplicand and the multiplier and their sign bits is the most sensitive part in the whole experiment. As the input is to be done bit by bit, there is a huge chance of overlapping of corresponding bits if the time interval is not

perfect which will result in the submission of incorrect input values. Keeping this hazard in mind, to make the input procedure easy an interval of 3000 milliseconds is kept between the input of two corresponding bits. This interval has reduced the rate of bits overlapping up to a greater extent. The following tables portraits the results of each experiment done on the model:

Table 1. Shows the result obtained from the Serial monitor of Arduino IDE:

EXP	4-BIT Multiplicand	Sign Bit	4-BIT Multiplier	Sign Bit	RESULT (Output)
1	*1000*	*0*	*0011*	*0*	*00011000*
2	*0111*	*1*	*0011*	*0*	*00010101*
3	*0110*	*0*	*0011*	*1*	*00010010*
4	*0110*	*1*	*0100*	*1*	*00011000*
5	*0000*	*0*	*0000*	*0*	*00000000*
6	*0111*	*1*	*0110*	*0*	*00101010*
7	*0010*	*0*	*0111*	*1*	*00001110*
8	*1000*	*0*	*0111*	*0*	*00111000*
9	*0111*	*1*	*0111*	*1*	*00110001*
10	*0001*	*1*	*0101*	*0*	*00000101*

Table 2. Shows the result obtained from the Hardware for the corresponding results. It shows the sequence of LEDs representing the Result:

EXP	Result Obtained from the Serial Monitor	Corresponding Hardware Output
1	*00011000*	
2	*00010101*	
3	*00010010*	
4	*00011000*	
5	*00000000*	
6	*00101010*	
7	*00001110*	
8	*00111000*	
9	*00110001*	
10	*00000101*	

In the Table 2 each orange colored LEDs represents the LEDs which are HIGH representing binary '1' in the eight-bit result and others represents binary '0'.

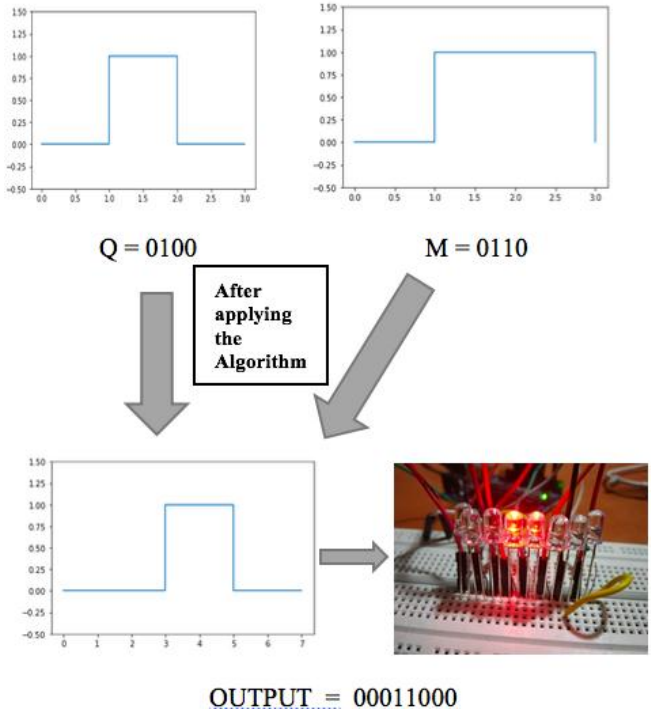


Figure 5. Timing Diagrams and Practical Output of EXP 4

In Figure 5 the Hardware based output and the timing diagrams of the input and output sequences of EXP 4 where the inputs are M = $(0110)_2$ and Q = $(0100)_2$ and the result is $(00011000)_2$ is described. Each experiment can be described in the same way like Figure 5. While constructing we encountered with some problems regarding the construction. The problems are such as loss of connection between the resistors and the GND pin due to some defects in the resistors and some defective LEDs also resulted in showing some invalid outputs.

V. CONCLUSION

In this research we have documented a study with an experiment by performing Booth's Algorithm with the help of Arduino. The study helped us to get a broader idea of the some multiplication algorithms and concluded that the Booth's Algorithm for multiplication is the most convenient way to multiply signed numbers than any other algorithm. The circuit was constructed very efficiently and performed very well for each of the experiments done on it by giving us perfect outputs every time. The work also provided us with a scope to know various electronic components. Each

component used for the model performed flawlessly during the random experiments.

REFERENCES

[1] Barun Biswas, Bidyut B Chowdhuri, "Generalization of Booth's Multiplication", International Conference on Computational Intelligence: Modeling, Techniques and Applications (CIMTA) 2013, Procedia Technology, Vol.**10**, pp.**304-310, 2013**

[2] Deepali Chandel, Gagan Kumawat, Pranay Lahoty, Vidhi Vart Chandrodaya, Shailendra Sharma, " Booth Multiplier: Ease of Multiplication", International Journal of Emerging Technology and Advanced Engineering, Vol.**3,** Issue.**3,** pp.**326-330, March 2013**

[3] T.K. Ghosh, "Computer Oganization", Second Edition, McGraw Hill Education (India) Private Limited, India, pp.**2.1-2.38, 2015**

[4] Shoba Mohan, Nakkeeran Rangaswamy "*An improved implementation of hierarchy array multiplier using Cs1A and full swing GDI logic*", ELECTRONICS, Vol.**21,** No.**1**, pp.**38-47**, **June 2017.**

[5] Ashwini K. Dhumal, Prof. S.S. Shrigan "*Comparison between Radix-2 and Radix-4 basedon Booth Algorithm*", International Journal of Advanced Research in Computer and Communication Engineering, Vol.**5**, Issue.**12**, pp.**498-500**, **December 2016**.

[6] Sandeep Shrivastava, Jaikaran Singh, Mukesh Tiwari, "*Implementation of Radix-2 Booth Multiplier and Comparison with Radix-4 Encoder Booth Multiplier*", International Journal on Emerging Technologies, Vol.**2,** pp.**14-16, 2011**

[7] Carl Hamacher, Zvonko Vranesic, Safwat Zaky, "Computer Organization", Fifth Edition, McGraw Hill Education (India) Private Limited, India, pp.**367-410, 2011**

Authors Profile

Mr. Kushankur Ghosh is currently pursuing Bachelor of Technology from University of Engineering and Management, Kolkata (UEMK) in Computer Science and Engineering. He is currently working as a Research Assistant at UEMK and working on several research projects. He is an active member of ACM Student chapter of UEMK since 2018. His research interests focuses on Machine Learning, Data Science, Data Mining, NLP, Deep Learning and IoT.

Miss Arghasree Banerjee is currently pursuing her B.Tech degree from University of Engineering and Management, Kolkata (UEMK) in Computer Science and Engineering and is also currently working as a Research Assistant at UEMK. She is a member of ACM Student chapter of UEMK since 2018. Her main research interests are Machine Learning, Data Analysis, NLP, Deep Learning and IoT.

