School of
Electronics and Communication Engineering

Mini Project Report

on

DESIGN AND IMPLEMENTATION OF DADDA MULTIPLIER IN 45nm TECHNOLOGY

By:

1. **Prashanth Aski** USN: 01FE21BEC268
2. **Rachanna R U** USN: 01FE21BEC273
3. **Sonal Sheth** USN: 01FE21BEC274
4. **Srushti Maharajpet** USN: 01FE21BEC301

Semester: V, 2023-2024

Under the Guidance of

Prof. Suhas Shirol

K.L.E SOCIETY'S
KLE Technological University,
HUBBALLI-580031
2023-2024

SCHOOL OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that project entitled **Design and Implementation of DADDA multiplier in 45nm Technology** is a bonafide work carried out by the student team of **Prashanth Aski** (01FE21BEC268), **Rachanna R U** (01FE21BEC273), **Sonal Sheth** (01FE21BEC274) and **Srushti Maharajpet** (01FE21BEC301). The project report has been approved as it satisfies the requirements with respect to the mini project work prescribed by the university curriculum for BE (V Semester) in School of Eletronics and Communication Engineering of KLE Technological University for the academic year 2023-2024.

Prof. Suhas Shirol	**Dr. Suneeta V Budihal**	**Dr. B S Anami**
Guide	**Head of School**	**Registrar**

External Viva:

Name of Examiners	**Signature with Date**
1.	
2.	

ACKNOWLEDGEMENT

The accomplishment of the project titled ”Design and implementation of DADDA Multiplier in 45nm technology” is imbued with a profound sense of fulfillment, which would remain incomplete without the acknowledgment of those individuals who played instrumental roles through their invaluable guidance and unwavering support. Our heartfelt gratitude extends to KLE Technological University, Hubballi, for providing us with a conducive environment and invaluable opportunities that empowered us to realize our aspirations. We express our sincere gratitude to Dr. Suneeta V Budihal, Head of the School of Electronics and Communication, for her foresight in affording us the platform to enhance our cognitive abilities. Special thanks are due to Prof. Suhas Shirol, our designated guide, whose consistent support and insightful suggestions were pivotal in navigating the complexities of the project. We extend our appreciation to the entire ADLD staff for their unwavering guidance and steadfast support throughout the project's duration.

- THE PROJECT TEAM

ABSTRACT

The initial phase of the project involved a comprehensive study of the DADDA multiplier's functionality, where the architecture and dataflow were thoroughly understood. Subsequently, a Verilog code implementing the required algorithm was meticulously written using structural description approach. The code's accuracy was validated through Xilinx simulation, covering both 8x8 and 16x16 multiplier configurations. Moving forward, hardware simulation was done on Spartan6 FPGA board, yielding the desired results. Transitioning to the physical design and synthesis phase, the Verilog source code files (.v files) underwent elaboration in the Cadence backend. Waveform simulations were executed using the nclaunch tool within the Cadence software suite. The synthesis process was then initiated using the Genus tool in Cadence, resulting in the generation of a comprehensive synthesis reports. The reports included essential parameter values such as area, power, and delay for both the 8x8 and 16x16 multipliers.

Contents

List of Figures

Chapter 1

Introduction

The 16x16 design and Implementation of a 16X16 bit multiplier with an efficient Full adder and Dadda technique is a challenging operation that calls for a comprehensive knowledge of optimization approaches and technological design ideas. The Dadda algorithm is a widely recognized technique for quick and effective multiplication of big values. Its foundation is a cyclic framework that divides the procedure of multiplication into smaller challenges that are integrated to generate the final outcome. The Dadda algorithm's energy efficiency is its main benefit, which is attained by minimizing the amount of adders and logical computations needed to complete the multiplication.

1.1 Motivation

The main Motivation to take up this project was to work in Backend. Designing a Multiplier requires writing verilog modules which requires hands-on experience on Xilinx software as a pre-requisite. To work in backend requires to have a way with software and corresponding tools. Here we get to work on cadence and explore its tools which adds to our software skills.

1.2 Objectives

1. To design an 8-bit and 16-bit Dadda multiplier in 45nm technology.
2. To generate synthesis reports of the multipliers and perform a detailed analysis.

1.3 Literature survey

Many methods, including CLA reasoning , genetic algorithms , evolutionary algorithms , duration pathway un-equalization , merging delay conversion etc., are been used to create digital components with little latency, minimal power consumption, and the highest throughput possible in a brief period of reaction time. Among the methods are bidirectional logic, Pass-transistor logic, CMOS technology, etc. With fewer transistors used and modest terminal capacitances available [4], the multiplier employing the Pass transistor logic approach increases circuitry speed and minimizes latency.Multipliers with reversible logic dissipate less energy. While transmitting charges, multiplication with adiabatic static CMOS logic generate the least amount of heat [1] .

Integrating a pair of 2-input Multiplexers for producing the bits for carry and sum, two 4-input MUXs to generate the carry bit and the sum bit, and two 2-input XOR gates to make the carry and the sum yields various complete adder designs. A 4-bit multiplier design [7] that features rapid operation and minimal power consumption was made utilizing pass transistor logic.

In the majority of embedded CPU designs, energy loss is a crucial design issue. The Arithmetic and Logic Unit is one of the processor's basic and vital components.Typically,A combinational logic circuit with many functional parts is used to implement ALUs in order to carry out various operations related to arithmetic and logic [3]. ALUs can be created using a chain structure. This is readily modelable or included into a CPU architecture to minimize total power consumption in a particular application [2].

Dadda in 1965. One among the schemes developed from the simultaneous multiplier is DADDA Multiplier [6].This is a simultaneous multiplier technique that reduces the quantity of adder steps needed to finish limited product summation. This is achieved by reducing the amount of bits at each summing stage using HA and FA in an array . Because it needs fewer gates than the Wallace tree multiplier, the DADDA multiplier consequently is less costly. The DADDA multiplication technique is sluggish because of the sequential multiplication procedure, even if it has a less complicated and easier structure. Ripple Carry full adder may be used to create a DADDA multiplication.

1.4 Problem statement

1. Design and Implementation of Dadda Multiplier in 45nm Technology

1.5 Project Planning

The project is planned in various stages:

1. Literature Survey: Understanding the previous works about how the design is made and what are the conditions for optimizing the design for lesser area, power and delay.
2. Verilog Code: Understand the flow of design and write the verilog code in structural description for Half adder, Full adder, 8X8 Dadda multiplier and 16X16 Dadda multiplier.

3. Xilinx Simulation: The verilog code is verifed by simulating in Xilinx after writing the test-bench for each module.

4. Hardware Implementation: The design is synthesized to dump on the hardware(FPGA), and verify the output for desired results.

5. Cadence Tool: After verifying the hardware results, the verilog source code files(.v files) were taken to the backend of cadence for simulating the waveforms and thus verifying the results.

6. RTL Synthesis in 45nm technology: The design was synthesized to calculate the delay, power and area in 45nm technology in cadence using Genus tool. Report files were generated along with Netlist and Synopsis Design Constraint (SDC) files.

1.6 Organization of the report

Chapter 2 System Design
This chapter consists of a block diagram of DADDA Diagram, explains the working of DADDA Multiplier. The further sections in the chapter give a brief explanation and limitations of Wallace Multipliers. Further the chapter speaks of the improvements of DADDA multiplier and the steps involved in overcoming the limitations of Wallace multiplier.

Chapter 3 Results and Discussions
This chapter has the snapshots of our simulation and hardware results. Further sections talk about the analysis of our obtained results. Reports were generated of the area, power and delay and all the data obtained in written in a concise form as a table.

Chapter 4 Conclusion and Future Scope
The insights obtained by the authors during the entire project have been written in brief as conclusion. The forthcoming potential of the project is outlined in the following section labeled future scope.

Chapter 2

System design

In this Chapter, we explain how the multiplier is designed.

2.1 Functional block diagram

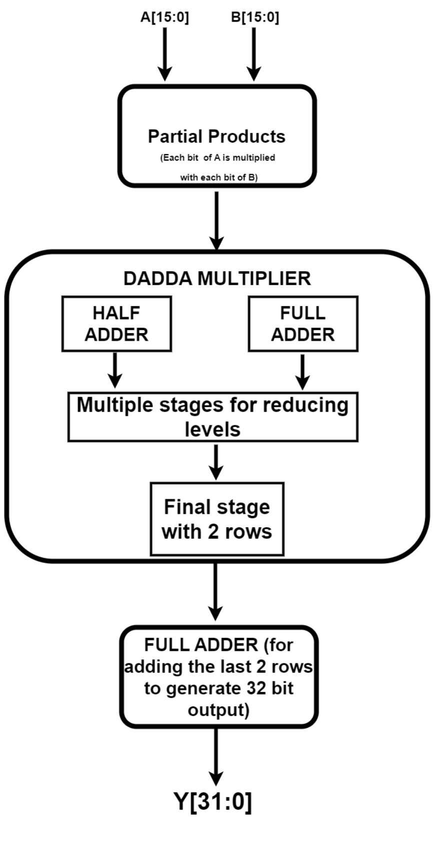

A[15:0]
B[15:0]
Partial Products
(Each bit of A is multiplied with each bit of B)
DADDA MULTIPLIER
HALF ADDER
FULL ADDER
Multiple stages for reducing levels
Final stage with 2 rows
FULL ADDER (for adding the last 2 rows to generate 32 bit output)
Y[31:0]

Figure 2.1: Block diagram of Dadda multiplier

Figure 2.1 shows the block diagram of the multiplier design. Input to the design are two 16 bit numbers which are represented as 'A' and 'B'. These numbers are are first multiplied with each bit of 'A' and 'B'. Next stage has the main reduction stage which is made of multi-stage with Full adders and Half adders. The final 2 rows are fed as the input to next stage for the final adder which give the 32-bit multiplied output 'Y'.

2.2 DADDA Multiplier

The Dadda multiplier was found in 1965 by computer scientist Luigi Dadda as a refinement of another popular multiplier design called the Wallace multiplier. Although both multipliers achieve similar tasks, the Dadda approach addressed some limitations of the Wallace design.

2.2.1 Limitations of Wallace multipliers

- High hardware complexity: They require a large number of adders and gates, especially for larger operand sizes, leading to increased area and power consumption.
- Uneven critical path: The signal propagation delay, which determines the speed of the multiplier, can vary significantly across different paths in the Wallace tree, impacting overall performance.
- Irregular layout: The placement of adders and wires in a Wallace tree can be quite intricate, making it challenging for physical design and implementation.

2.2.2 Dadda's improvements

- Reduced hardware complexity: By strategically arranging the partial product reduction stages, Dadda multipliers require fewer adders and gates compared to Wallace designs for equivalent operand sizes. This translates to lower area and power consumption.
- Balanced critical path: Dadda prioritizes placing full adders, which have longer delays, in paths with fewer additional adders. This results in a more balanced critical path and potentially faster operation.
- Regular layout: The structure of a Dadda tree is more predictable and organized compared to a Wallace tree. This simplifies physical design and makes it easier to implement in hardware.

The Dadda multiplier wasn't discovered by chance but rather arose from a deliberate effort to address the shortcomings of existing multiplier designs. Its focus on minimizing hardware complexity while maintaining reasonable speed and a regular layout made it a valuable alternative for implementation.

The steps involved in reducing the level:

- Partial Product Generation: The first step involves generating the partial products, similar to any multiplication process. Each bit of one operand multiplies with every bit of the other operand, resulting in a matrix of partial products.
- Initial Level:This matrix represents the initial level of the Dadda tree. Each column in the matrix contains partial products that need to be added together to obtain the final product.
- Level Reduction Rules:

- No reduction: If a column has only two or three partial products, no reduction is necessary.
- Half-adder (HA): If a column has four partial products, a half-adder (HA) can be used to combine the top two bits. The sum is placed at the bottom, and the carry bit is added to the next column.
- Full-adder (FA): If a column has five or more partial products, a full-adder (FA) is used to combine the top three bits. The sum and carry are placed similarly to the half-adder case.

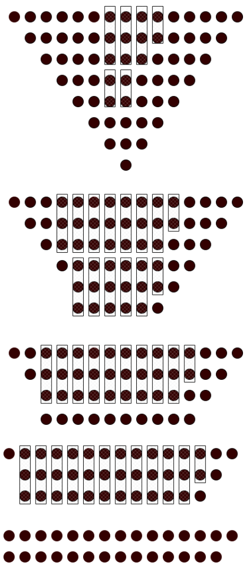


Figure 2.2: 8-bit Dadda Multiplier stage reduction algorithm

Figure 2.2 shows the dot diagram where the bits are reduced using the reduction method which involves these steps 'Partial Product Generation','Initial Level','Level Reduction Rules','Level Reduction Rules','Iterative Reduction' and 'Final Addition'.

- Iterative Reduction:These reduction rules are applied iteratively to each column, starting from the lowest level and moving upwards. Once a level is reduced, the carry bits are incorporated into the next level's calculations.
- Final Addition:These reduction rules are applied iteratively to each column, starting from the lowest level and moving upwards. Once a level is reduced, the carry bits are incorporated into the next level's calculations.

Full adder functionality
Sum : S = A$\oplus B \oplus C$
$Carry : Co = (A\&B) \mid (B\&C) \mid (C\&A)$

The above general equations can be optimised by reducing the number of transistors used in the physical design by changing the equations of sum and carry to
Wire : W1 = $\sim ((\sim (A \mid B)) \mid (A\&B))$
$Sum : S = \sim ((\sim (W1 \mid C)) \mid (W1\&C))$
$Carry : Co = ((A\&B) \mid (W1\&C))$

Initial Partial Products: At the beginning, generate 256 partial products by multiplying each pair of bits from the 16x16 input matrix.

Dadda Tree Formation: Arrange the partial products in a Dadda tree structure, where each node represents a partial product and can be combined with adjacent nodes through Dadda tree reduction rules.

3:2 Compressor Units: Implement 3:2 compressors at each level of the Dadda tree, reducing the number of partial products by combining three into two. These compressors efficiently optimize the multiplication process.

Tree Level Reduction: Continue the reduction process through multiple levels of the Dadda tree until a final set of products is obtained. This hierarchical reduction minimizes the number of partial products at each stage.

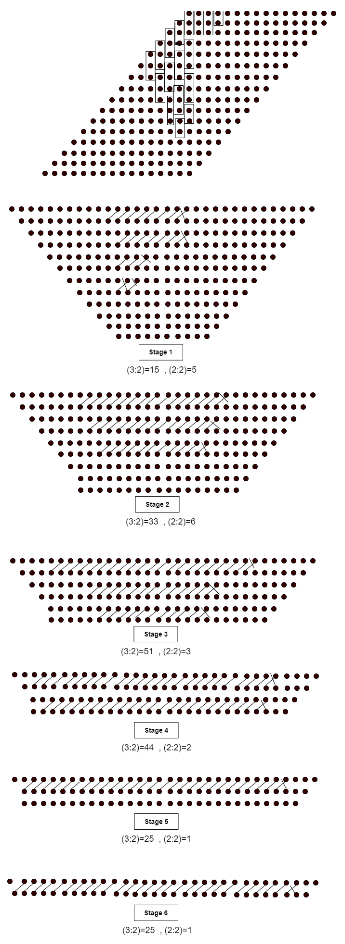


Figure 2.3: 16-bit Dadda Multiplier stage reduction algorithm

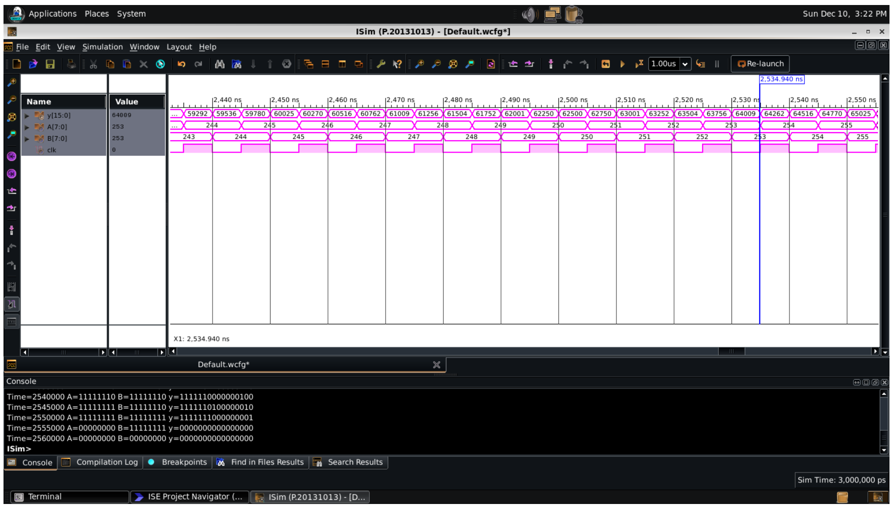
Final Summation: Figure 2.3 shows that, at last stage it collects the reduced partial products at the last level of the Dadda tree and sum them to obtain the final product of the 16x16 multiplication. The Dadda Multiplier architecture significantly reduces the number of additions compared to a simple array multiplier, leading to improved efficiency in terms of speed and area.

Chapter 3

Results and Discussions

Dadda multiplier is designed for both 8x8 and 16x16 bits. The verilog code is written in modular form so that we could use 8x8 as a function for 16x16 which would reduce considerable lines of code. The Design is implemented and simulated in Xilinx and cadence.

3.1 Result Analysis



Name	Value
y[15:0]	64009
A[7:0]	253
B[7:0]	253
clk	0

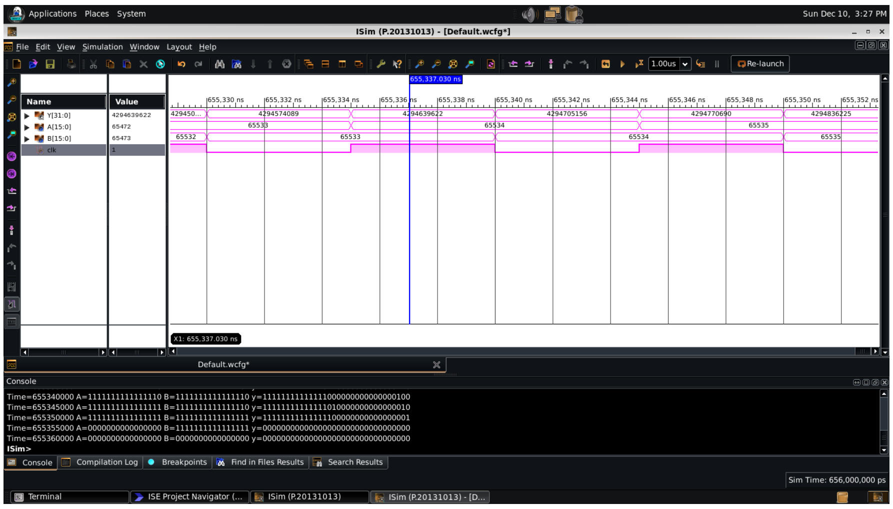
```
Time=2540000 A=11111110 B=11111110 y=1111110000000100
Time=2545000 A=11111111 B=11111110 y=1111110100000010
Time=2550000 A=11111111 B=11111111 y=1111111000000001
Time=2555000 A=00000000 B=11111111 y=0000000000000000
Time=2560000 A=00000000 B=00000000 y=0000000000000000
ISim>
```

Figure 3.1: Simulation result of 8-bit multiplier

In the above figure 3.1 we could verify the simulation results by converting the Radix from binary to unsigned decimal.

The highest value of a 8-bit binary number is 255 so in the simulation window we can see the product of 255 and 255 gives 65,025.

Further the same simulation is run for 16-bit number and the highest value of a 16-bit binary number is 65535 and the product of the highest number is 4,29,48,36,225 which is shown in Figure 3.2.



```
Time=655340000 A=1111111111111110 B=1111111111111110 y=11111111111111100000000000000100
Time=655345000 A=1111111111111111 B=1111111111111110 y=11111111111111101000000000000010
Time=655350000 A=1111111111111111 B=1111111111111111 y=11111111111111110000000000000001
Time=655355000 A=0000000000000000 B=1111111111111111 y=00000000000000000000000000000000
Time=655360000 A=0000000000000000 B=0000000000000000 y=00000000000000000000000000000000
ISim>
```

Figure 3.2: Simulation result of 16-bit multiplier

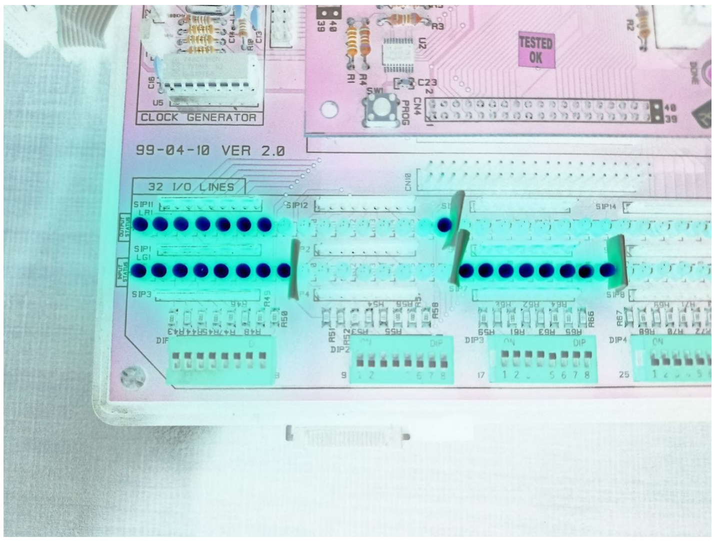
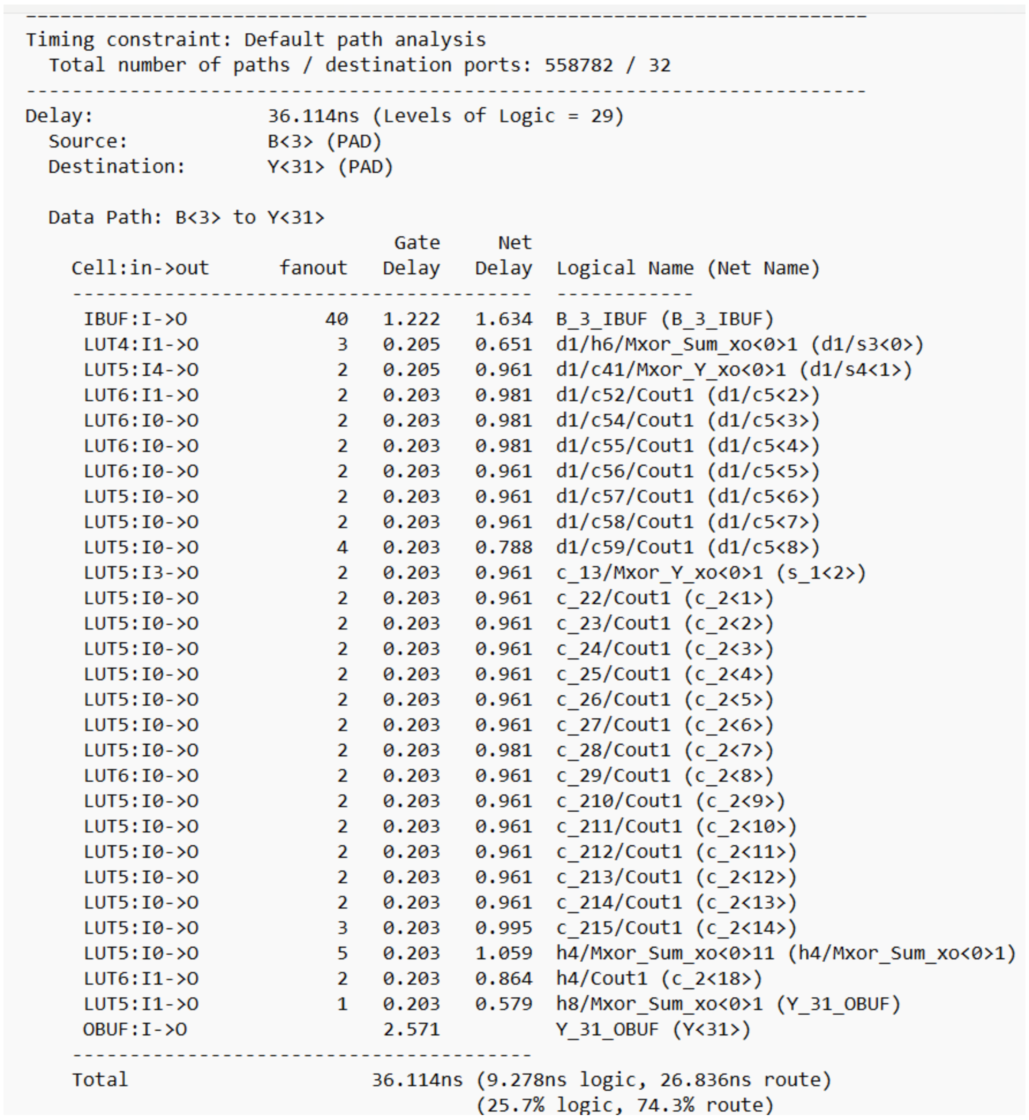


Figure 3.3: Implementation result of 8-bit multiplier on FPGA board

Figure 3.3 is the snapshot of the implementation of 8-bit miltiplier on FPGA , It is also verified with various inputs and a example is explained below. The board implementation shows the output of multiplying 8-bit numbers, The white paper represents the set of 8-bits for a number. The lower line represents the input numbers and the upper line represents the 16-bit output.
Upper line shows: Y[15:0] = 16'b1111111000000001
Decimal value for 'Y' = 65025
Lower line shows: A[7:0] = 8'b11111111
Decimal value for 'A' = 255
Lower line shows: B[7:0] = 8'b11111111
Decimal value for 'B' = 255

Y = A * B
Y = 255 * 255 = 65025



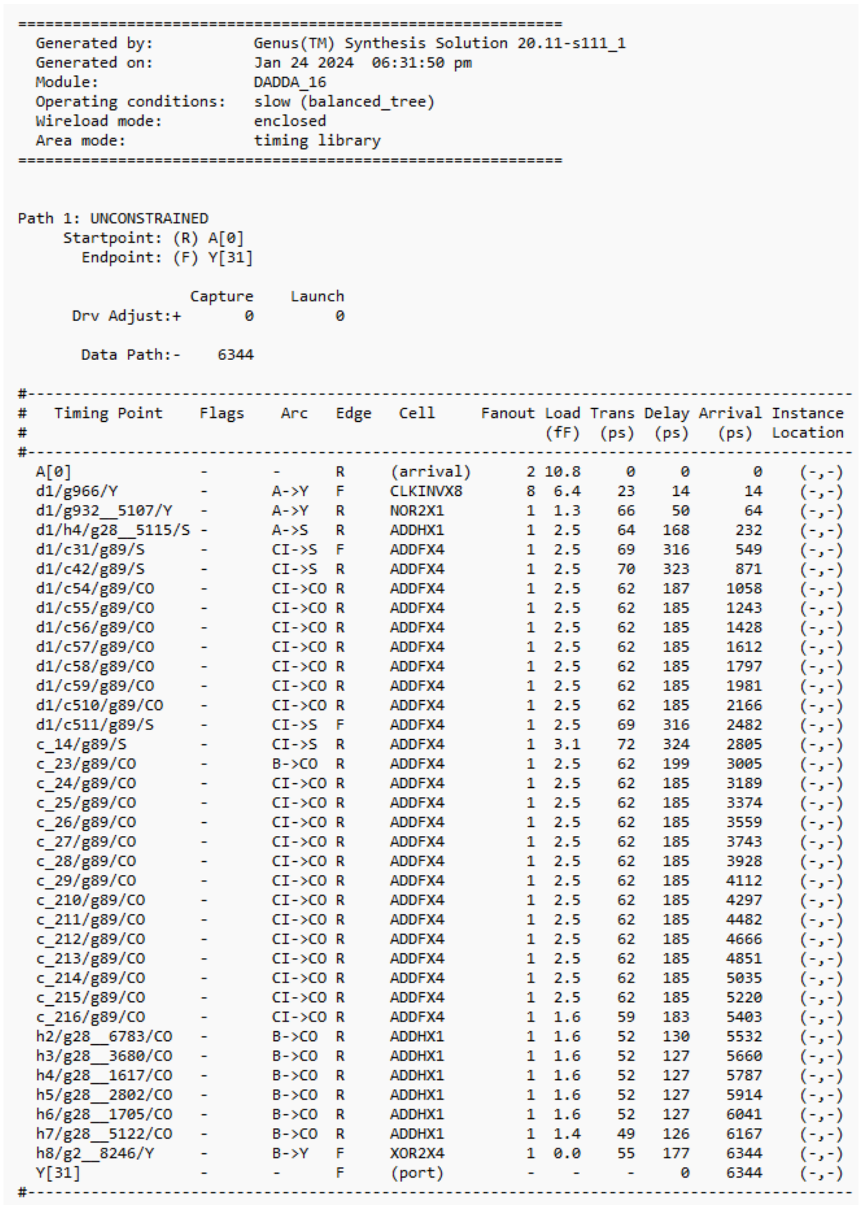
```
--------------------------------------------------------------------------
Timing constraint: Default path analysis
  Total number of paths / destination ports: 558782 / 32
--------------------------------------------------------------------------
Delay:               36.114ns (Levels of Logic = 29)
  Source:            B<3> (PAD)
  Destination:       Y<31> (PAD)

  Data Path: B<3> to Y<31>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O            40   1.222   1.634  B_3_IBUF (B_3_IBUF)
     LUT4:I1->O            3   0.205   0.651  d1/h6/Mxor_Sum_xo<0>1 (d1/s3<0>)
     LUT5:I4->O            2   0.205   0.961  d1/c41/Mxor_Y_xo<0>1 (d1/s4<1>)
     LUT6:I1->O            2   0.203   0.981  d1/c52/Cout1 (d1/c5<2>)
     LUT6:I0->O            2   0.203   0.981  d1/c54/Cout1 (d1/c5<3>)
     LUT6:I0->O            2   0.203   0.981  d1/c55/Cout1 (d1/c5<4>)
     LUT6:I0->O            2   0.203   0.961  d1/c56/Cout1 (d1/c5<5>)
     LUT5:I0->O            2   0.203   0.961  d1/c57/Cout1 (d1/c5<6>)
     LUT5:I0->O            2   0.203   0.961  d1/c58/Cout1 (d1/c5<7>)
     LUT5:I0->O            4   0.203   0.788  d1/c59/Cout1 (d1/c5<8>)
     LUT5:I3->O            2   0.203   0.961  c_13/Mxor_Y_xo<0>1 (s_1<2>)
     LUT5:I0->O            2   0.203   0.961  c_22/Cout1 (c_2<1>)
     LUT5:I0->O            2   0.203   0.961  c_23/Cout1 (c_2<2>)
     LUT5:I0->O            2   0.203   0.961  c_24/Cout1 (c_2<3>)
     LUT5:I0->O            2   0.203   0.961  c_25/Cout1 (c_2<4>)
     LUT5:I0->O            2   0.203   0.961  c_26/Cout1 (c_2<5>)
     LUT5:I0->O            2   0.203   0.961  c_27/Cout1 (c_2<6>)
     LUT5:I0->O            2   0.203   0.981  c_28/Cout1 (c_2<7>)
     LUT6:I0->O            2   0.203   0.961  c_29/Cout1 (c_2<8>)
     LUT5:I0->O            2   0.203   0.961  c_210/Cout1 (c_2<9>)
     LUT5:I0->O            2   0.203   0.961  c_211/Cout1 (c_2<10>)
     LUT5:I0->O            2   0.203   0.961  c_212/Cout1 (c_2<11>)
     LUT5:I0->O            2   0.203   0.961  c_213/Cout1 (c_2<12>)
     LUT5:I0->O            2   0.203   0.961  c_214/Cout1 (c_2<13>)
     LUT5:I0->O            3   0.203   0.995  c_215/Cout1 (c_2<14>)
     LUT5:I0->O            5   0.203   1.059  h4/Mxor_Sum_xo<0>11 (h4/Mxor_Sum_xo<0>1)
     LUT6:I1->O            2   0.203   0.864  h4/Cout1 (c_2<18>)
     LUT5:I1->O            1   0.203   0.579  h8/Mxor_Sum_xo<0>1 (Y_31_OBUF)
     OBUF:I->O                 2.571          Y_31_OBUF (Y<31>)
    ----------------------------------------
    Total                     36.114ns (9.278ns logic, 26.836ns route)
                                       (25.7% logic, 74.3% route)
```

Figure 3.4: Synthesis report on Xilinx tool on Time delay analysis

Figure 3.4 shows the sythesis report on time delay analysis on xilinx software. Analysis starting point is B[3] and the end point is Y[31]. The gate delay, and net delay at each stage is in nano-seconds.

3.2 Synthesis Report Analysis

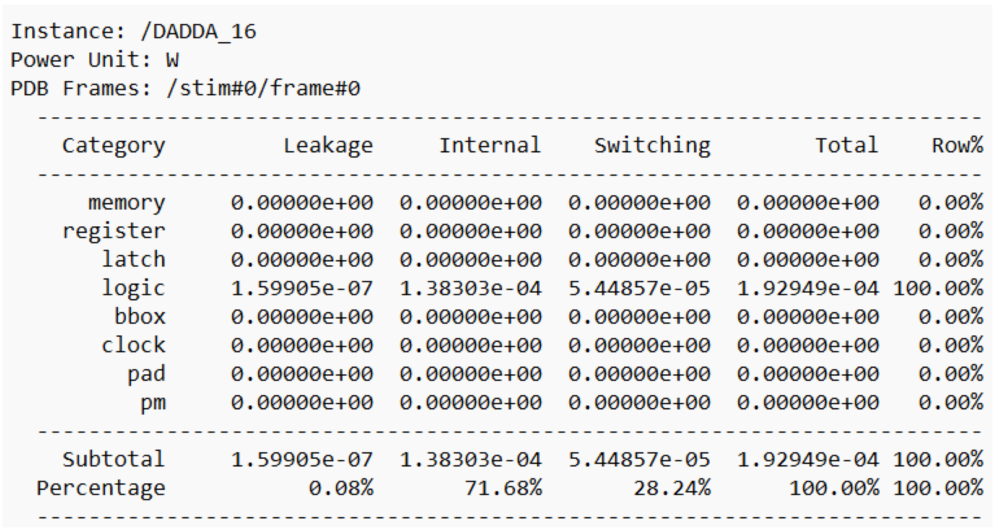
Area Analysis : The cell count for 8-bit multiplier is 136 and in 16-bit multiplier, which gives the number of instances occur during the execution of the modules.



```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
  Generated on:           Jan 24 2024  06:31:50 pm
  Module:                 DADDA_16
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================


Path 1: UNCONSTRAINED
     Startpoint: (R) A[0]
       Endpoint: (F) Y[31]

                   Capture    Launch
      Drv Adjust:+       0         0

       Data Path:-    6344

#---------------------------------------------------------------------------------------------
#   Timing Point    Flags    Arc   Edge   Cell     Fanout Load Trans Delay Arrival Instance
#                                                         (fF)  (ps)  (ps)   (ps)  Location
#---------------------------------------------------------------------------------------------
  A[0]              -       -      R     (arrival)      2 10.8     0     0       0    (-,-)
  d1/g966/Y         -       A->Y   F     CLKINVX8       8  6.4    23    14      14    (-,-)
  d1/g932__5107/Y   -       A->Y   R     NOR2X1         1  1.3    66    50      64    (-,-)
  d1/h4/g28__5115/S -       A->S   R     ADDHX1         1  2.5    64   168     232    (-,-)
  d1/c31/g89/S      -       CI->S  F     ADDFX4         1  2.5    69   316     549    (-,-)
  d1/c42/g89/S      -       CI->S  R     ADDFX4         1  2.5    70   323     871    (-,-)
  d1/c54/g89/CO     -       CI->CO R     ADDFX4         1  2.5    62   187    1058    (-,-)
  d1/c55/g89/CO     -       CI->CO R     ADDFX4         1  2.5    62   185    1243    (-,-)
  d1/c56/g89/CO     -       CI->CO R     ADDFX4         1  2.5    62   185    1428    (-,-)
  d1/c57/g89/CO     -       CI->CO R     ADDFX4         1  2.5    62   185    1612    (-,-)
  d1/c58/g89/CO     -       CI->CO R     ADDFX4         1  2.5    62   185    1797    (-,-)
  d1/c59/g89/CO     -       CI->CO R     ADDFX4         1  2.5    62   185    1981    (-,-)
  d1/c510/g89/CO    -       CI->CO R     ADDFX4         1  2.5    62   185    2166    (-,-)
  d1/c511/g89/S     -       CI->S  F     ADDFX4         1  2.5    69   316    2482    (-,-)
  c_14/g89/S        -       CI->S  R     ADDFX4         1  3.1    72   324    2805    (-,-)
  c_23/g89/CO       -       B->CO  R     ADDFX4         1  2.5    62   199    3005    (-,-)
  c_24/g89/CO       -       CI->CO R     ADDFX4         1  2.5    62   185    3189    (-,-)
  c_25/g89/CO       -       CI->CO R     ADDFX4         1  2.5    62   185    3374    (-,-)
  c_26/g89/CO       -       CI->CO R     ADDFX4         1  2.5    62   185    3559    (-,-)
  c_27/g89/CO       -       CI->CO R     ADDFX4         1  2.5    62   185    3743    (-,-)
  c_28/g89/CO       -       CI->CO R     ADDFX4         1  2.5    62   185    3928    (-,-)
  c_29/g89/CO       -       CI->CO R     ADDFX4         1  2.5    62   185    4112    (-,-)
  c_210/g89/CO      -       CI->CO R     ADDFX4         1  2.5    62   185    4297    (-,-)
  c_211/g89/CO      -       CI->CO R     ADDFX4         1  2.5    62   185    4482    (-,-)
  c_212/g89/CO      -       CI->CO R     ADDFX4         1  2.5    62   185    4666    (-,-)
  c_213/g89/CO      -       CI->CO R     ADDFX4         1  2.5    62   185    4851    (-,-)
  c_214/g89/CO      -       CI->CO R     ADDFX4         1  2.5    62   185    5035    (-,-)
  c_215/g89/CO      -       CI->CO R     ADDFX4         1  2.5    62   185    5220    (-,-)
  c_216/g89/CO      -       CI->CO R     ADDFX4         1  1.6    59   183    5403    (-,-)
  h2/g28__6783/CO   -       B->CO  R     ADDHX1         1  1.6    52   130    5532    (-,-)
  h3/g28__3680/CO   -       B->CO  R     ADDHX1         1  1.6    52   127    5660    (-,-)
  h4/g28__1617/CO   -       B->CO  R     ADDHX1         1  1.6    52   127    5787    (-,-)
  h5/g28__2802/CO   -       B->CO  R     ADDHX1         1  1.6    52   127    5914    (-,-)
  h6/g28__1705/CO   -       B->CO  R     ADDHX1         1  1.6    52   127    6041    (-,-)
  h7/g28__5122/CO   -       B->CO  R     ADDHX1         1  1.4    49   126    6167    (-,-)
  h8/g2__8246/Y     -       B->Y   F     XOR2X4         1  0.0    55   177    6344    (-,-)
  Y[31]             -       -      F     (port)         -    -     -     0    6344    (-,-)
#---------------------------------------------------------------------------------------------
```

Figure 3.5: Synthesis report on time delay

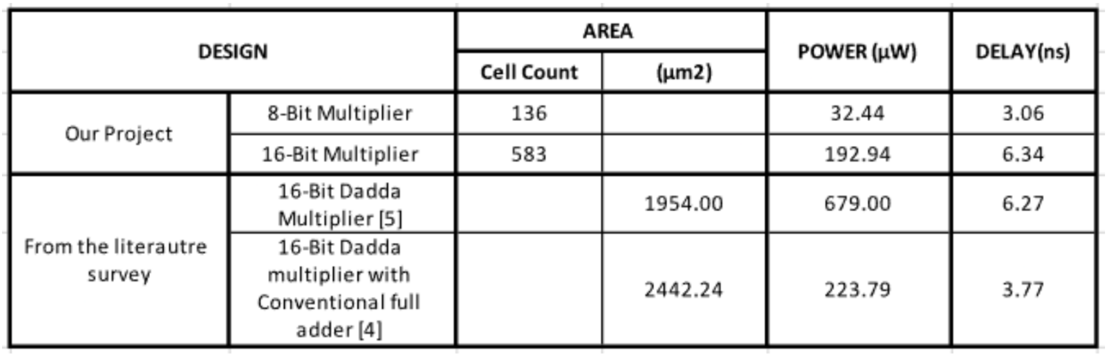
Timing Analysis : Figure 3.4 shows the synthesis report on time delay. Delay column in the table 3.1 depicts the delay of output from the starting point A[0] to Y[15] for 8-bit multiplier and A[0] to Y[31] for 16-bit multiplier. 8-bit multiplier has 3.06ns delay and 16-bit multiplier has 6.34ns delay.



```
Instance: /DADDA_16
Power Unit: W
PDB Frames: /stim#0/frame#0
  -------------------------------------------------------------------------
    Category         Leakage     Internal    Switching        Total    Row%
  -------------------------------------------------------------------------
      memory     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
    register     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
       latch     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
       logic     1.59905e-07  1.38303e-04  5.44857e-05  1.92949e-04 100.00%
        bbox     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
       clock     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
         pad     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
          pm     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
  -------------------------------------------------------------------------
    Subtotal     1.59905e-07  1.38303e-04  5.44857e-05  1.92949e-04 100.00%
  Percentage           0.08%       71.68%       28.24%      100.00% 100.00%
  -------------------------------------------------------------------------
```

Figure 3.6: Synthesis Report on power

Power Analysis : Figure 3.5 shows the power synthesis report on power analysis on 16x16 dadda multiplier. The total power is calculated by adding Leakage, Internal and Switching power. By the synthesis analysis The respective power values are shown in the Figure 3.7 also. The Figure 3.7 shows the analysis on the authors design and we can see the comparison from



DESIGN		AREA		POWER (µW)	DELAY(ns)
		Cell Count	(µm2)		
Our Project	8-Bit Multiplier	136		32.44	3.06
	16-Bit Multiplier	583		192.94	6.34
From the literautre survey	16-Bit Dadda Multiplier [5]		1954.00	679.00	6.27
	16-Bit Dadda multiplier with Conventional full adder [4]		2442.24	223.79	3.77

Figure 3.7: Analysis and comparison of reports

the paper [5] and paper [4]. It shows the power, area and delay analysis. The authors design shows comparatively more delay than mentioned papers, But The power is significantly reduced for the proposed design of full adder. The previous researchers [4] and [5] have worked on 45nm technology and has the power of 679 W and 223.79 W respectively. The authors design has the total power of 192.94 W. The main highlight of the comparison focuses on the power efficiency, where the proposed design performed better in reducing the power.

Chapter 4

Conclusions and Future Scope

4.1 Conclusion

To lower the overall power dissipation in VLSI architecture, a number of methods have been proposed, each with varying degrees of power saving. In order to create new efficient data path designs that lowers total electrical consumption, mapping methods are used.

The novel framework for the entire adder design is realized in the proposed research using 45nm technology . The Full adder is used for generating 16-bit dadda multiplier, and the synthesized results are produced.

Multipliers are used in processors to execute multiplication operations in a range of different software and hardware configurations. Decrease in multiplying power . The functioning of DSP is essential. Because of this, the low power DADDA multiplier is created utilizing complete adders' circuits that incorporate various logic designs.

The XILINX tool is used to simulate the DADDA multiplier. Next, the multiplier's characteristics of performance are ascertained and examined. The parameter study confirms that the DADDA multiplier, which employs both logical types, outperforms the Wallace tree multiplication with regard to energy. Nonetheless, the DADDA multiplier helps to reach the ideal delay.

4.2 Future scope

The Dadda multiplier can be implemented on lesser technology node to increase the power efficiency, and reduce the Area and delay for faster performance. The currently used Full adder is written using XOR gates which consumes more area i.e more transistors to form XOR gates. This logic can be improved by writing basic gates using AND and OR for Sum and carry , which uses less number of gates and eventually less transistors are used.

Bibliography

[1] Anurag Chauhan, Amit Kumar Meena, and Amit Kumar. Performance analysis of 4-bit multiplier using 90nm technology. In *2022 2nd International Conference on Intelligent Technologies (CONIT)*, pages 1–5. IEEE, 2022.

[2] Priti Gangwar, Ritik Gupta, and Gurjit Kaur. A design technique for delay and power efficient dadda-multiplier. In *2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 66–71. IEEE, 2021.

[3] V Manu, AM Vijaya Prakash, and Mohan U Chandra. Design and implementation of sixteen-bit low power and area efficient dadda multiplier. In *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, pages 631–636. IEEE, 2019.

[4] Muteen Munawar, Talha Khan, Muhammad Rehman, Zain Shabbir, Kamran Daniel, Ahmed Sheraz, and Muhammad Omer. Low power and high speed dadda multiplier using carry select adder with binary to excess-1 converter. In *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, pages 1–4. IEEE, 2020.

[5] S Ravi, Govind Shaji Nair, Rajeev Narayan, and Harish M Kittur. Low power and efficient dadda multiplier. *Research Journal of Applied Sciences, Engineering and Technology*, 9(1):53–57, 2015.

[6] Muhammad Hussnain Riaz, Syed Adrees Ahmed, Qasim Javaid, and Tariq Kamal. Low power 4$\times$ 4 bit multiplier design using dadda algorithm and optimized full adder. In *2018 15th international Bhurban conference on applied sciences and technology (IBCAST)*, pages 392–396. IEEE, 2018.

[7] P Samundiswary and K Anitha. Design and analysis of cmos based dadda multiplier. *IJCEM International Journal of Computational Engineering & Management*, 16(6):12–17, 2013.