A PROJECT REPORT ON

Approximate Multiplier Design Using Novel Dual-Stage 4:2 Compressor

Submitted to Dept. of ECE

By

Dama Rupesh Babu (R151737)

Muppala Mahesh Babu(R151735)

In partial fulfilment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINEERING

Under the guidance of

Mr.Venkateshwarulu Naik

Rajiv Gandhi University of Knowledge Technologies-AP

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this major project titled "**Approximate Multiplier Design Using Novel Dual-Stage 4:2 Compressor"** submitted by **Dama Rupesh Babu (R151737), Muppala Mahesh Babu (R151735)** . Under the guidance of **Mr.VENKATESHWARULU NAIK** in partial fulfilment of the academic requirements for the award of degree of **Bachelor of Technology** in **Electronics and Communication Engineering** by **RGUKT, R K VALLEY** during the academic year 2020-2021.

Internal Guide	Head of the Department
Mr.Venkateshwarulu Naik,	Ms. G.Lakshmi Sireesha ,
Assistant Professor,	Assistant professor,
Dept.of ECE,RGUKT,	Dept.of ECE,RGUKT,
RK Valley.	RK Valley.

DECLARATION

We **D.Rupesh Babu** and **M.Mahesh Babu** hereby declare that this report entitled **"Approximate Multiplier Design Using Novel Dual 4:2 Compressor"** submitted by us under the guidance and supervision of **Mr.Venkateshwarulu Naik** is a bonafide work. We also declare that it has not been submitted previously in part or in full to this University or other University or Institution for the award of any degree or diploma.

Date: 04-08-2021 Dama Rupesh Babu(R151737)

Place:R K Valley M.Mahesh Babu(R151735)

ACKNOWLEDGEMENT

We would like to express my sincere gratitude to **Mr.Venkateshwarulu Naik** , our project internal guide for giving valuable suggestions and shown keen interest throughout the progress of our project.We are grateful to , HOD ECE, for providing excellent computing facilities and congenial atmosphere for progressing with my project. At the outset, I would like to thank Rajiv Gandhi University of Knowledge Technologies, R.K Valley for providing all the necessary resources for the successful completion of our project.

With sincere regards,

Dama Rupesh Babu(R151737)

Muppala Mahesh Babu(R151735)

Contents:

Abstract:

High speed error-tolerant circuits with approximate computing are required for multimedia applications. These applications have high performance at the cost of reduction in accuracy and also reduce the complexity of the system architecture, delay and power consumption. This project proposes the design of approximate 4 : 2 compressor with reduced area, delay and power with comparable accuracy when compared with the existing 4:2 compressors. The proposed compressors are utilised to implement 8 x 8 and 16 x 16 multipliers. In order to maintain the balance between delay, area and power, approximate computing has emerged as a promising solution. Approximation in arithmetic operations result in faster systems with lesser design complexity and power consumption. The trade off would be reduction in accuracy, which does not necessarily affect the normal operation. The designed Multiplier is implemented on Xilinx Vivado tool to see the Timing diagrams, Utilization and Power which comments on Area, Power and Speed.

2.Introduction:

In High Speed Multimedia Applications, higher performance is significant which may cost in accuracy in most of the compressors. So, approximate compressors are designed with reduced area, delay and power consumption with accuracy better than the existing ones.
The proposed 4:2 compressor has lesser area, power and delay constraints than the exact 4:2 compressor. This designed compressor will be used in 8 x 8 multiplier and the performance metrics are noted. For a higher performance in a processor, approximations can be used.
Speed is inversely proportional to delay of the system. To increase speed, parallel operations should be done. For that, hardware increases which may result in increasing the area and power dissipation. So, to maintain the balance between these trade-offs, approximation is nominal solution.
Generally, approximations in arithmetic operations increases the speed but, reduces the accuracy. But, in multimedia applications, it may not affect due to inability of eye to detect finer details. Therefore, Novel high speed area-efficient, low power 4 : 2 compressor architecture is proposed and dadda multiplier is implemented using this.
The proposed approximate 4:2 compressor shows 56.80% reduction in area, 57.20% reduction in power and 73.30% reduction in delay compared to

anaccurate 4:2 compressor. The proposed compressors are utilised to implement 8 x 8 and 16 x 16 multipliers. These multipliers have comparable accuracy when compared with state-of-the-art approximate multiplers.

3.Software Description:

Vivado Design Suite

Vivado includes many new tools and design flows that facilitate and enhance the latest design methods. It runs faster, allows better use of FPGA resources, and allows designers to focus their time evaluating design alternatives.

The FPGA enables the functionality of the chip to be programmed in, enabling this to be updated at any point required. This can be changed to accommodate updates or to even change the functionality a board or system when it is required to perform different functions.

The very name of the FPGA states that it is programmable. It is necessary to code to programme the FPGA. Knowing how to program an FPGA is a key skill and it forms a specialised area of electronic design.

FPGA Programming approaches

There are several ways to develop the code to program an FPGA. In the very early days of FPGAs it might just have been possible to program the simplest FPGAs manually. Today this is not an option and a software program is required. There are several options open to FPGA developers:

- **VHDL**: VHDL stands for VHSIC Hardware Description Language, where the VHSIC itself stands for Very High Speed Integrated Circuit. This FPGA programming language was developed by the US Department of Defense to document the behaviour of ASICs, or Application Specific Integrated Circuits. Base heavily on the programming language Ada, VHDL is a text language which has been very successful and popular for many years in programming FPGAs.

- **Verilog**: Verilog was the first form of hardware description language to be developed. It is standardised as IEEE 1364.

- **LabVIEW FPGA**: LabVIEW FPGA utilises the basic LabVIEW graphical interface but employs additional tools to enable it to provide the functionality required for programming FPGAs.

In this Project,Verilog HDL is used:

Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL). It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip−flop. It means, by using a HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

4.Implementation:

In multiplication operation, partial product summation has in-arguably been the major contributor to the power dissipation and the delay in the system. Research shows that compressor can reduce the delay associated with the partial product summations. Compressor estimate the count of logic 1 in the input using half adders and/or full addres. The various commonly used topologies for compressors are 7:3, 5:2, 4:2, and 3:2. A 4:2 compressor is preferred over any other topology due to the regularity achieved while cascading. Also, it has been widely used to design multipliers.

Due to the considerable reduction in delay using transmission gates when compared to traditional CMOS based logic, optimised design with transmission gates are explored in literature. But, the major disadvantage is the inconsistency in the rise and fall times for different inputs.

Exact 4 : 2 Compressor:

A exact 4:2 compressor comprises of five inputs, three outputs and two cascaded full adders. A1, A2, A3, A4 and Cin are the inputs and Cout, CARRY and SUM are the outputs of the exact 4:2 compressor. Cout,CARRY and SUM are given as

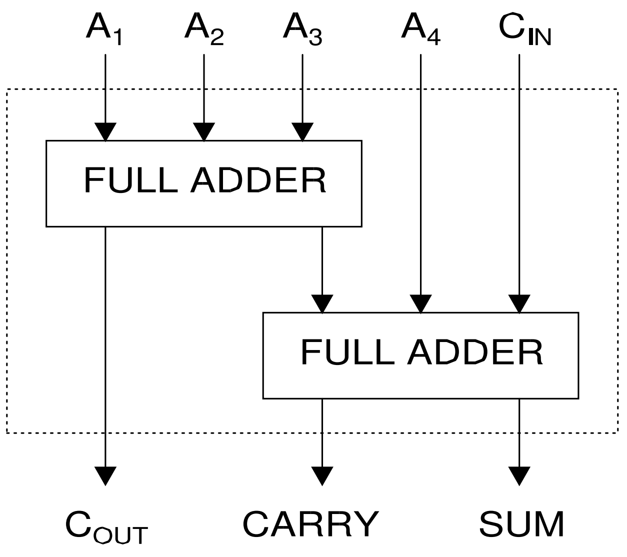


Fig. Exact 4:2 compressor

Equations for carry and sum:

$$C_{OUT} = A_3(A_1 \oplus A_2) + A_1(\overline{A_1 \oplus A_2})$$

$$\text{CARRY} = \text{CIN}\ (A1 \oplus A2 \oplus A3 \oplus A4) + A4(\overline{A1 \oplus A2 \oplus A3 \oplus A4})$$

$$\text{SUM} = \text{CIN} \oplus A1 \oplus A2 \oplus A3 \oplus A4$$

- ❑ Need more than 10 logic gates to synthesize the sum and carry.
- ❑ So,
  - i. Area is more
  - ii. Propagation delay is more
  - iii. Speed is less
  - iv. Overall performance is less.

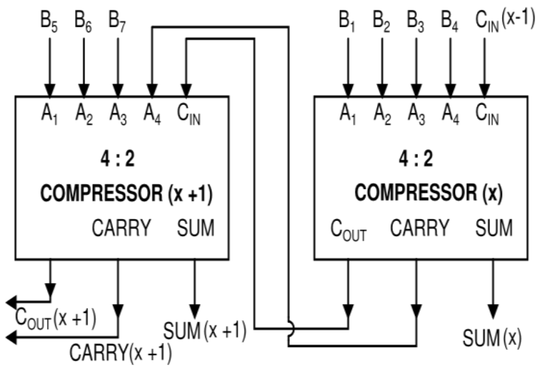


Fig. Compressor Chain

Cin represents the input carry from the preceding 4 : 2 compressor that has processed the lower significant bits. CARRY and COUT are the outputs of order ‘1’ with higher significance than the input Cin.

A_1	A_2	A_3	A_4	C_{IN}	C_{OUT}	$CARRY$	SUM
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	0
0	0	1	1	0	0	1	0
0	0	1	1	1	0	1	1
0	1	0	0	0	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0
0	1	0	1	1	0	1	1
0	1	1	0	0	1	0	0
0	1	1	0	1	1	0	1
0	1	1	1	0	1	0	1
0	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1
1	0	0	0	1	0	1	0
1	0	0	1	0	0	1	0
1	0	0	1	1	0	1	1
1	0	1	0	0	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	0	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	0	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	0	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	0	1	0	1
1	1	1	0	1	1	1	0
1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	1

Table: Truth Table for exact 4:2 Compressor

<u>Approximate Multipliers:</u>

Multiplication is unquestionably a performance determining operation in AI and DSP applications. These applications demand high speed multiplier architectures to necessitate high speed parallel operations with acceptable levels of accuracy. Introduction of approximation in multipliers leads to realisation of faster computations with reduced hardware complexity, delay and power, with accuracy in desirable levels.

Partial product summation is the speed limiting operation in multiplication due to the propagation delay in adder networks. In order to reduce the propagation delay, compressors are introduced. Compressors compute the sum and carry at each level simultaneously. The resultant carry is added with a higher significant sum bit in the next stage. This is continued until the final product is generated.

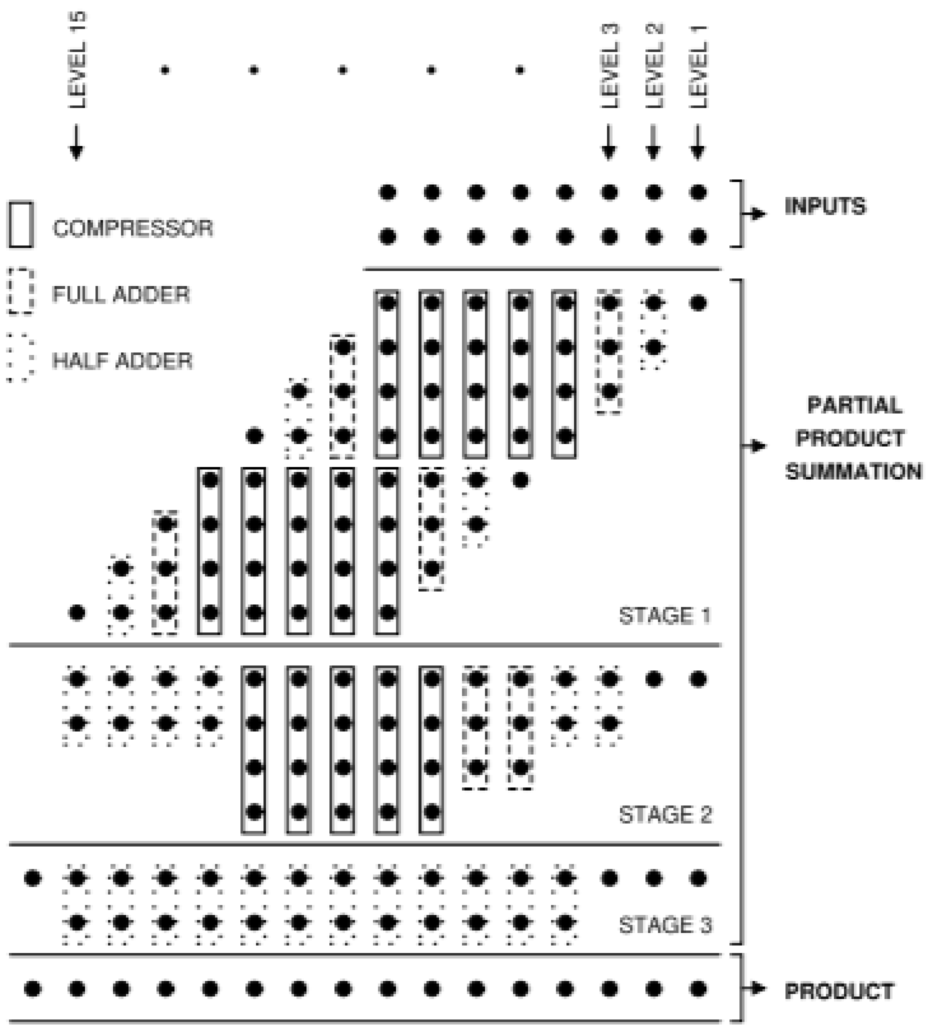


Fig: 8 x 8 approximate multiplier

Above figure shows how the multipliation is done by approximation.

Approximate Compressors:

On applying approximation to 4 : 2 compressor, output count can be reduced to 2. Approximation is done by eliminating COUT . This incurs an error only when the input combination is ‘1111’. When the input bits are ‘1111’ the CARRY and SUM are set to ‘11’ and an error of −1 is introduced. An 8×8 multiplication operation using approximate compressors.

Proposed High Speed Area Efficient Approximate 4:2 compressor:

The proposed high speed area-efficient 4 : 2 approximate compressor is shown in below figure. The compressor inputs are A1, A2, A3 and A4, outputs are CARRY and SUM.

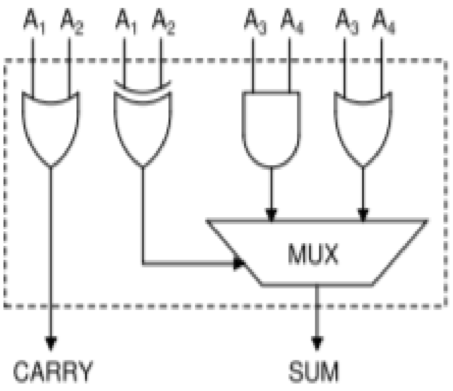
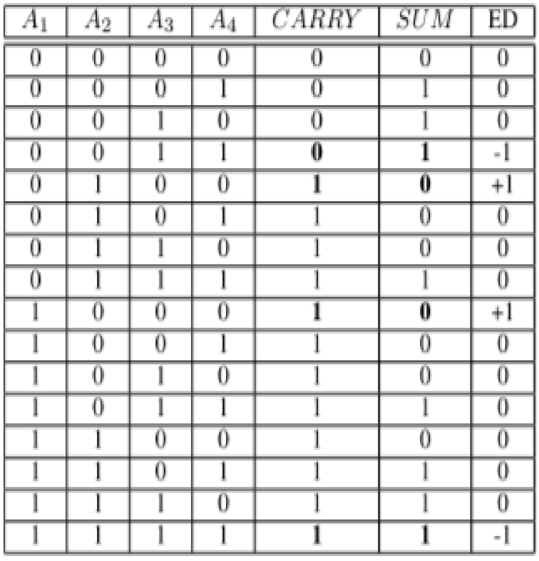


Fig:Proposed Area efficient 4:2 compressor



A_1	A_2	A_3	A_4	$CARRY$	SUM	ED
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	**0**	**1**	-1
0	1	0	0	**1**	**0**	+1
0	1	0	1	1	0	0
0	1	1	0	1	0	0
0	1	1	1	1	1	0
1	0	0	0	**1**	**0**	+1
1	0	0	1	1	0	0
1	0	1	0	1	0	0
1	0	1	1	1	1	0
1	1	0	0	1	0	0
1	1	0	1	1	1	0
1	1	1	0	1	1	0
1	1	1	1	**1**	**1**	-1

Table: Truth table for proposed Area efficient 4:2 compressor.

A multiplexer (MUX) based design approach is used to generate SUM.

Output of XOR gate acts as the select line for the MUX. When select line goes high, (A3A4) is selected and when it goes low, (A3 + A4) is selected. By introducing an error with error distance 1 in the truth table of the exact compressor, the proposed 4 : 2 compressor is able to reduce carry generation logic to an OR gate. The logical expressions for realisation of SUM and CARRY are given below

$$SUM = (A_1 \oplus A_2)A_3A_4 + \overline{(A_1 \oplus A_2)}(A_3 + A_4)$$
$$CARRY = A_1 + A_2$$

From the truth table of proposed 4 : 2 compressor, it can been observed that the error has been introduced for the input values − {0011}, {0100}, {1000} and {1111}, so as to ensure that equal positive and negative deviation with ED = 1 (minimum) is obtained.

Proposed Modified Dual-stage Approximate 4:2 compressor:

As a measure to optimise the hardware utilisation of the proposed design, this paper proposes an alternate architecture for multipliers with more than three stages of cascaded compressors. In the high speed area-efficient compressor architecture, apart from the MUX, one XOR, one AND and two OR gates are required. OR and AND gates each need 6 transistors in CMOS logic implementation. In order to reduce the transistor count, this paper proposes an architecture with NAND and NOR gates. Even though the SUM and CARRY generated by the modified architecture is not as same as that of the proposed 4 : 2 compressor architecture, with cascading of the compressor in multiples of 2, the error is nullified.

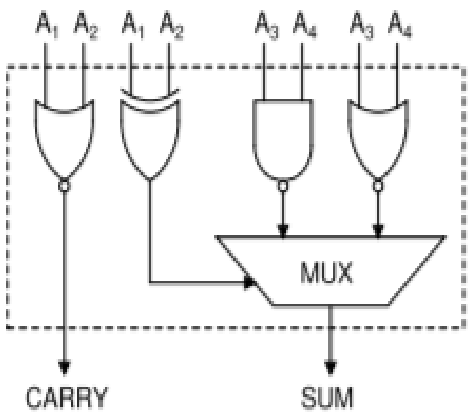


Fig: Basic building block for proposed modified dual-stage 4:2 compressor

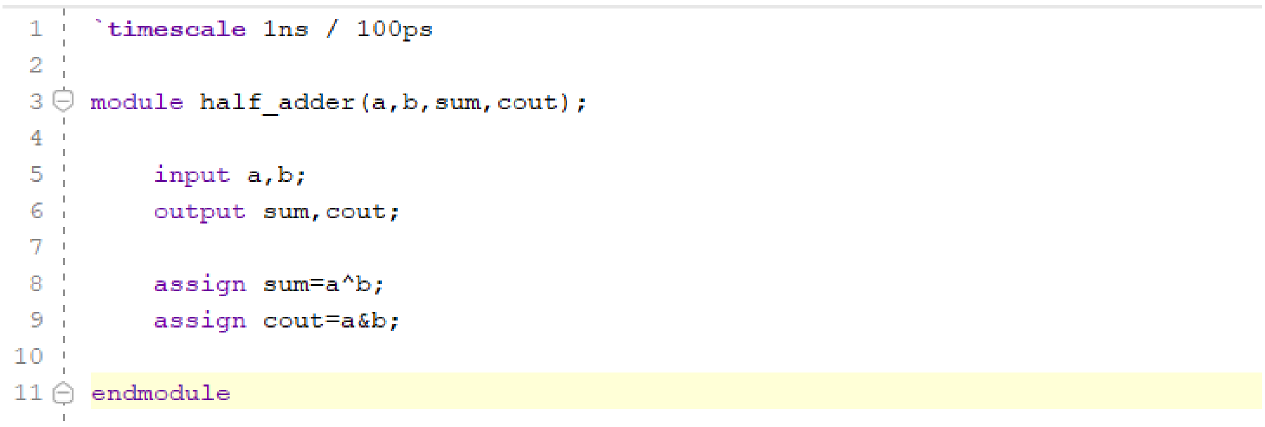
Carryout at Stage 2 output is minimised and is given by

$$\begin{aligned}&\overline{\overline{(K_3 \oplus K_4)} \cdot \overline{(K_2 + K_1)} + (K_3 \oplus K_4)\overline{(K_2K_1)}}\\&\quad = \overline{(K_3 \oplus K_4)} \cdot (K_2 + K_1) + (K_3 \oplus K_4) \cdot (K_2K_1)\\&\qquad + (K_2 + K_1)(K_2K_1)\end{aligned}$$

Here, it is seen that (K2 + K1)(K2K1) is not an essential prime implicant. Therefore, output expressions of Stage 2 for both the proposed architectures are the same. Similarly, Sum0 generated at Stage 1 differs, but the resultant logical expression at Stage 2 output remains the same. The accuracy remains unaffected with the modifications introduced.

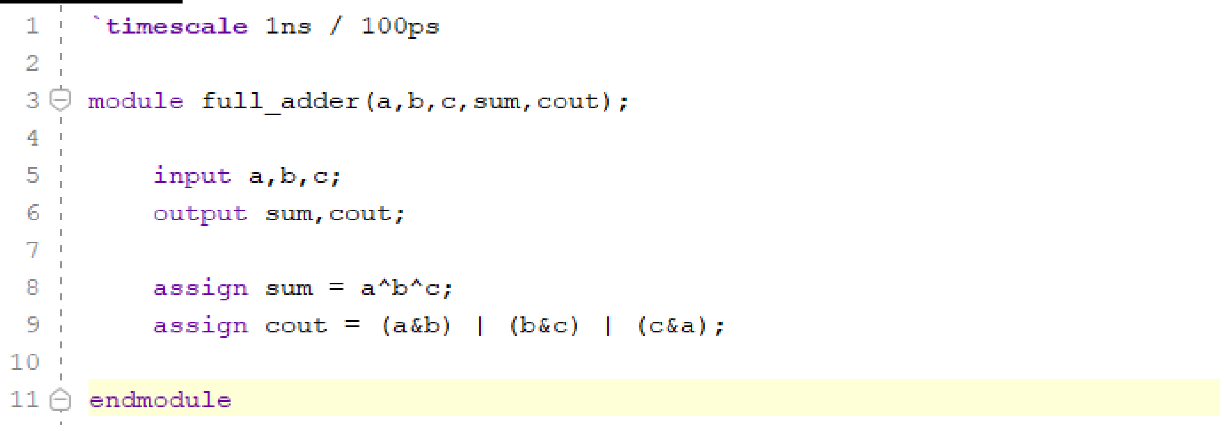
Verilog Codes:

Half Adder:



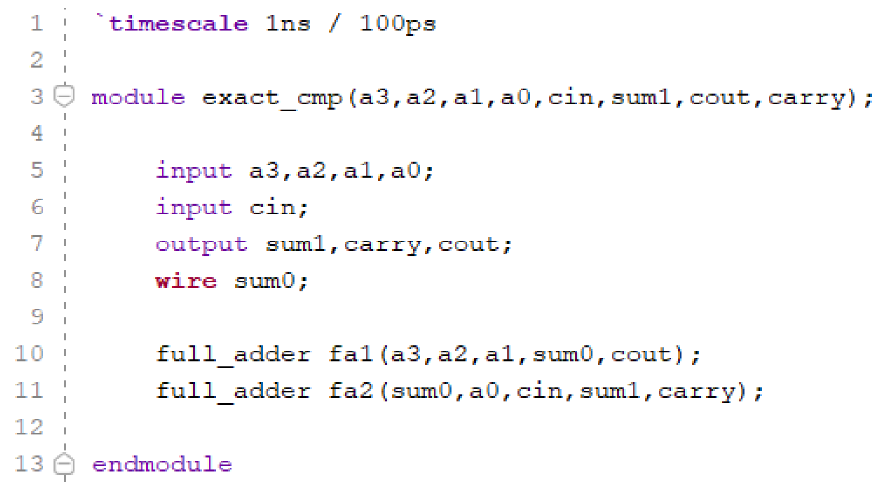
```verilog
`timescale 1ns / 100ps

module half_adder(a,b,sum,cout);

    input a,b;
    output sum,cout;

    assign sum=a^b;
    assign cout=a&b;

endmodule
```

Full Adder:



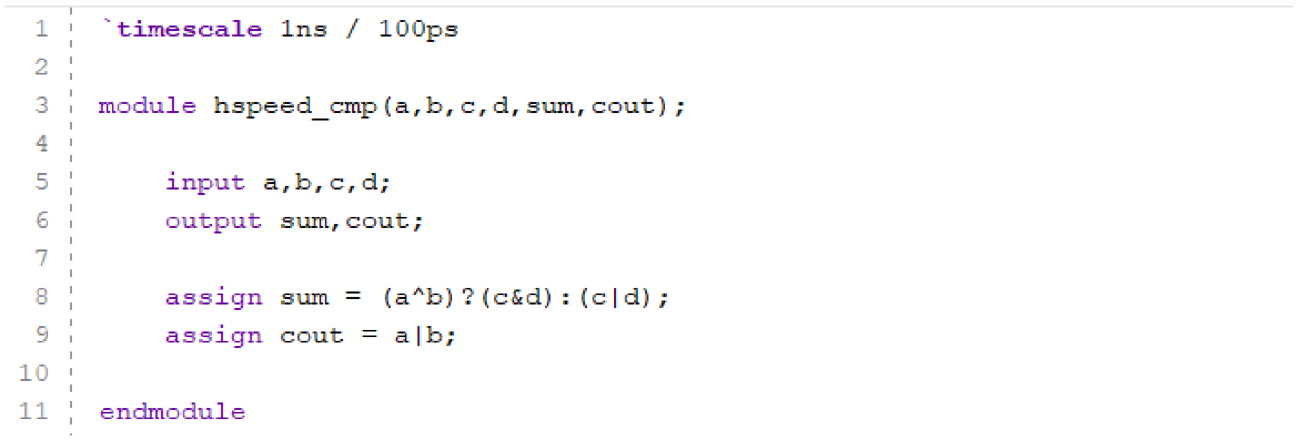
```verilog
`timescale 1ns / 100ps

module full_adder(a,b,c,sum,cout);

    input a,b,c;
    output sum,cout;

    assign sum = a^b^c;
    assign cout = (a&b) | (b&c) | (c&a);

endmodule
```

Exact 4:2 compressor:



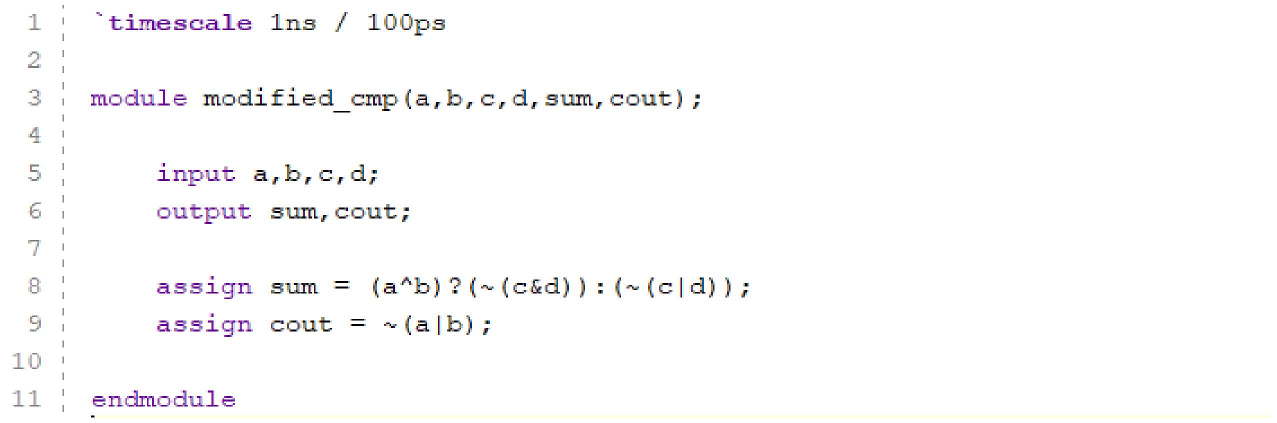
```verilog
`timescale 1ns / 100ps

module exact_cmp(a3,a2,a1,a0,cin,sum1,cout,carry);

    input a3,a2,a1,a0;
    input cin;
    output sum1,carry,cout;
    wire sum0;

    full_adder fa1(a3,a2,a1,sum0,cout);
    full_adder fa2(sum0,a0,cin,sum1,carry);

endmodule
```

Proposed area efficient approximate 4:2 compressor:



```verilog
`timescale 1ns / 100ps

module hspeed_cmp(a,b,c,d,sum,cout);

    input a,b,c,d;
    output sum,cout;

    assign sum = (a^b)?(c&d):(c|d);
    assign cout = a|b;

endmodule
```

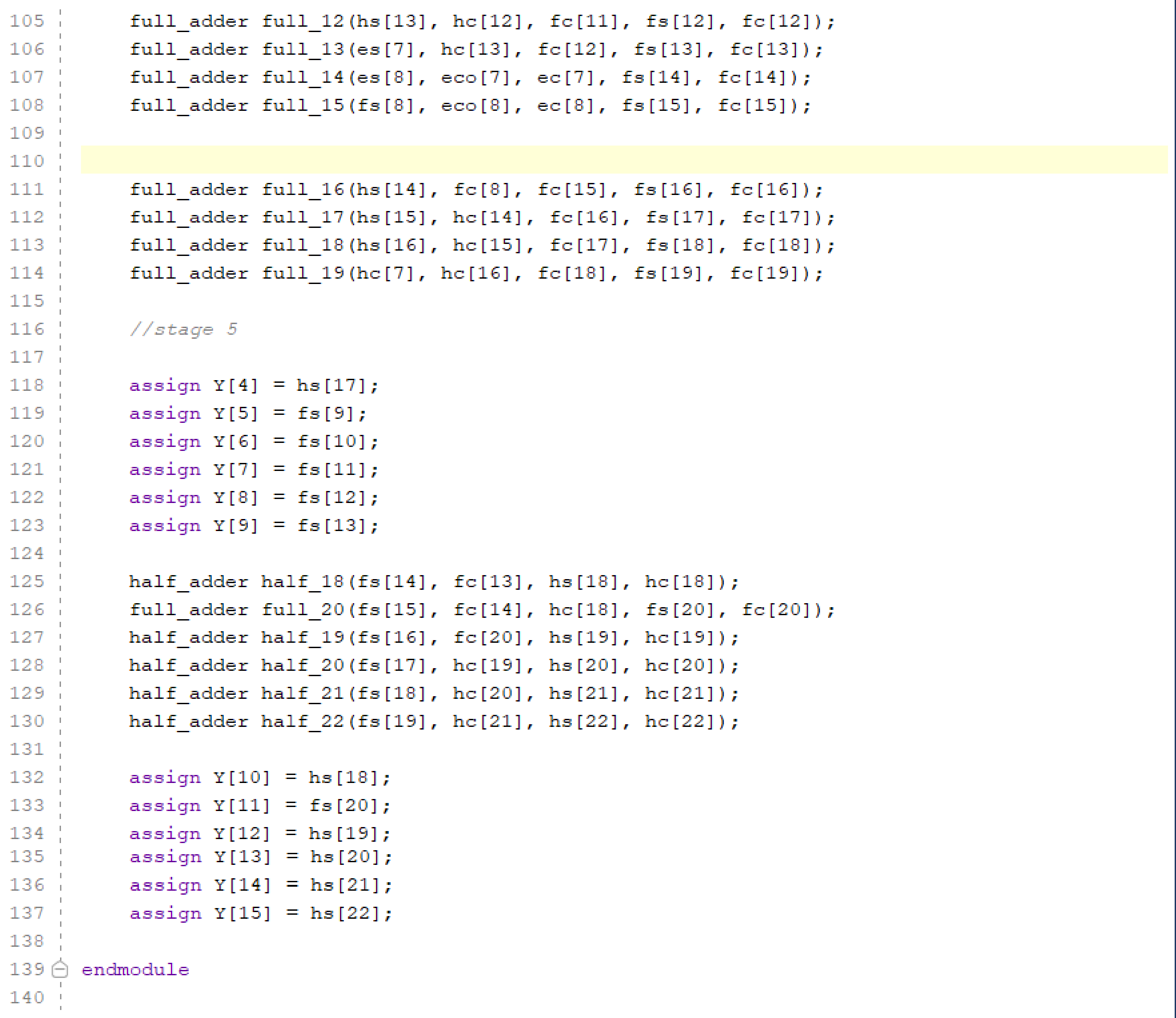
Proposed modified dual-stage approximate 4:2 compressor:



```verilog
`timescale 1ns / 100ps

module modified_cmp(a,b,c,d,sum,cout);

    input a,b,c,d;
    output sum,cout;

    assign sum = (a^b)?(~(c&d)):(~(c|d));
    assign cout = ~(a|b);

endmodule
```

8-bit Multiplier:

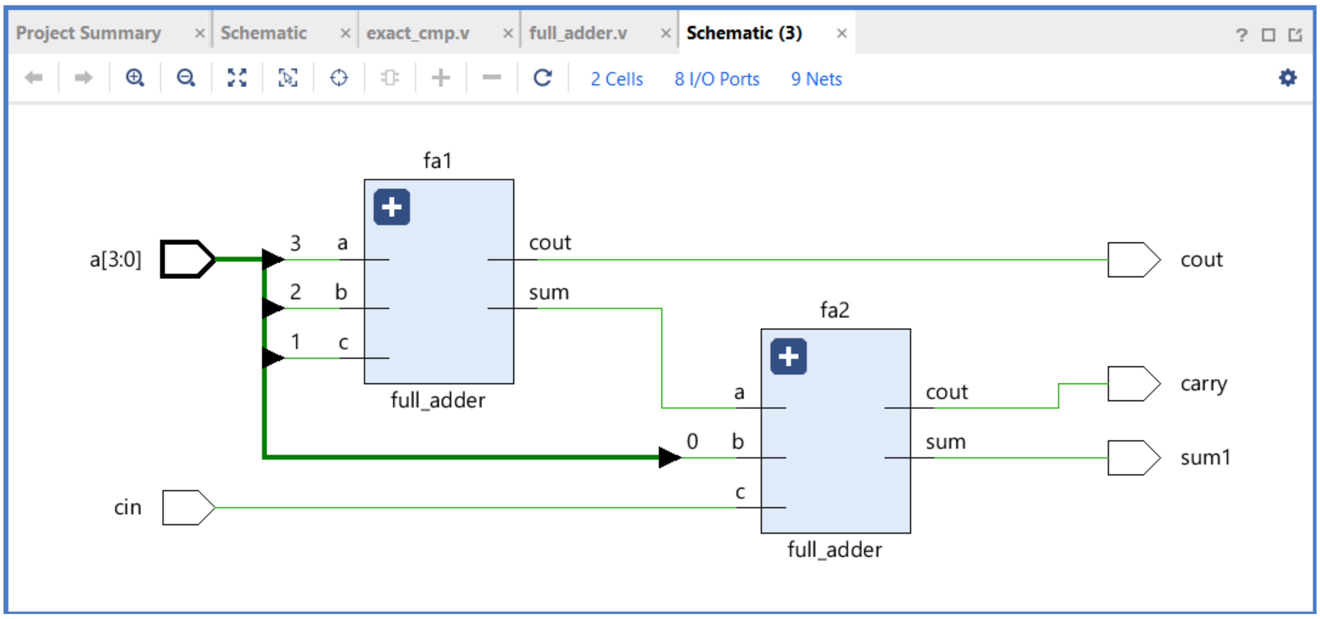
```verilog
`timescale 1ns / 100ps

module multiplier_16(a,b,Y);

    input [7:0]a,b;
    output [15:0]Y;

    //partial products of multiplication
    wire p[7:0][7:0];

    //hs, hc represent half_adder's sum and carry respectively
    wire [22:0]hs,hc;

    //fs, fc represents full_adder's sum and carry respectively
    wire [20:0]fs,fc;

    /*  ss, sc represents high speed area-efficient approximate
        4:2 compressor's sum and carry  */
    wire [4:0]ss,sc;

    /*  ms, mc represents modified dual-stage approximate
        4:2 compressor's sum and carry respectively  */
    wire [2:0]ms,mc;

    /*  es, ecou, ec represents exact compressor's sum, cout and
        carry respectively  */
    wire [8:0]es,eco,ec;
    wire dummy;

    /*  for loop is used to create the partial products of multiplication
        which are 64 in total */
    genvar i;
    genvar j;
    for(i=0;i<8;i=i+1)begin
        for(j=0;j<8;j=j+1)begin
            assign p[i][j] = a[j]*b[i];
        end
    end

    //stage1

    assign dummy = 0;
    assign Y[0] = p[0][0];

    half_adder half_0(p[0][1], p[1][0], hs[0], hc[0]);
    full_adder full_0(p[0][2], p[1][1], p[2][0], fs[0], fc[0]);
    hspeed_cmp h_cmp0(p[0][3], p[1][2], p[2][1], p[3][0], ss[0], sc[0]);
    hspeed_cmp h_cmp1(p[0][4], p[1][3], p[2][2], p[3][1], ss[1], sc[1]);
    hspeed_cmp h_cmp2(p[0][5], p[1][4], p[2][3], p[3][2], ss[2], sc[2]);
    half_adder half_1(p[4][1], p[5][0], hs[1], hc[1]);
    modified_cmp mod_cmp0(p[0][6], p[1][5], p[2][4], p[3][3], ms[0], mc[0]);
    full_adder full_1(p[4][2], p[5][1], p[6][0], fs[1], fc[1]);
    modified_cmp mod_cmp1(p[0][7], p[1][6], p[2][5], p[3][4], ms[1], mc[1]);
    modified_cmp mod_cmp2(p[4][3], p[5][2], p[6][1], p[7][0], ms[2], mc[2]);
    exact_cmp ext_cmp0(p[1][7], p[2][6], p[3][5], p[4][4],dummy, es[0], eco[0], ec[0]);
    full_adder full_2(p[5][3], p[6][2], p[7][1], fs[2], fc[2]);
    exact_cmp ext_cmp1(p[2][7], p[3][6], p[4][5], p[5][4], dummy, es[1], eco[1], ec[1]);
    half_adder half_2(p[6][3], p[7][2], hs[2], hc[2]);
    exact_cmp ext_cmp2(p[3][7], p[4][6], p[5][5], p[6][4], dummy, es[2], eco[2], ec[2]);
    exact_cmp ext_cmp3(p[4][7], p[5][6], p[6][5], p[7][4], dummy, es[3], eco[3],ec[3]);
    full_adder full_3(p[5][7], p[6][6], p[7][5], fs[3], fc[3]);
    half_adder half_3(p[6][7], p[7][6], hs[3], hc[3]);

    //stage 2

    half_adder half_4(fs[0], hc[0], hs[4], hc[4]);
    half_adder half_5(ss[0], fc[0], hs[5], hc[5]);
    full_adder full_4(ss[1], p[0][4], sc[0], fs[4], fc[4]);
    full_adder full_5(ss[2], hs[1],sc[1], fs[5], fc[5]);
    hspeed_cmp h_cmp3(ms[0], fs[1], sc[2], hc[1], ss[3], sc[3]);
    hspeed_cmp h_cmp4(ms[1], ms[2], mc[0], fc[1], ss[4], sc[4]);
    exact_cmp ext_cmp4(es[0], fs[2], mc[1], mc[2], dummy, es[4], eco[4], ec[4]);
    exact_cmp ext_cmp5(es[1], hs[2], eco[0], ec[0], dummy, es[5], eco[5],ec[5]);
    exact_cmp ext_cmp6(es[2], p[3][7], eco[1], ec[1], dummy, es[6], eco[6],ec[6]);
    full_adder full_6(es[3], eco[2], ec[2], fs[6], fc[6]);
    full_adder full_7(fs[3], eco[3], ec[3], fs[7], fc[7]);
    half_adder half_6(hs[3], fc[3], hs[6], hc[6]);
    half_adder half_7(p[7][7], hc[3], hs[7], hc[7]);

    //stage 3

    assign Y[1] = hs[0];
    assign Y[2] = hs[4];

    half_adder half_8(hs[5], hc[4], hs[8], hc[8]);
    half_adder half_9(fs[4], hc[4], hs[9], hc[9]);
    half_adder half_10(fs[5], fc[4], hs[10], hc[10]);
    half_adder half_11(ss[3], fc[5], hs[11], hc[11]);
    half_adder half_12(ss[4], sc[3], hs[12], hc[12]);
    half_adder half_13(es[4], sc[4], hs[13], hc[13]);
    exact_cmp ext_cmp7(es[5], fc[2], eco[4], ec[4], dummy, es[7], eco[7],ec[7]);
    exact_cmp ext_cmp8(es[6], hc[2], eco[5], ec[5], dummy, es[8], eco[8],ec[8]);
    full_adder full_8(fs[6], eco[6], ec[6], fs[8], fc[8]);
    half_adder half_14(fs[7], fc[6], hs[14], hc[14]);
    half_adder half_15(hs[6], fc[7], hs[15], hc[15]);
    half_adder half_16(hs[7], hc[6], hs[16], hc[16]);

    //stage 4

    assign Y[3] = hs[8];
    half_adder half_17(hs[9], hc[8], hs[17], hc[17]);
    full_adder full_9(hs[10], hc[9], hc[17], fs[9], fc[9]);
    full_adder full_10(hs[11], hc[10], fc[9], fs[10], fc[10]);
    full_adder full_11(hs[12], hc[11], fc[10], fs[11], fc[11]);
    full_adder full_12(hs[13], hc[12], fc[11], fs[12], fc[12]);
```



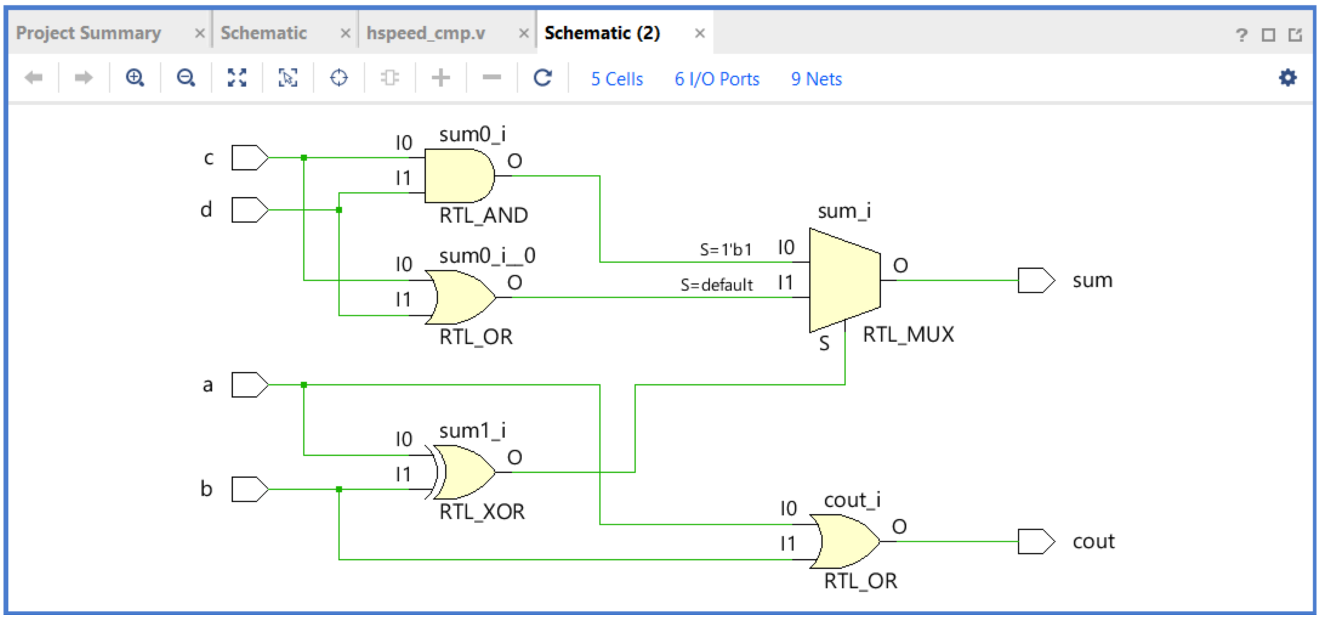
```
    full_adder full_12(hs[13], hc[12], fc[11], fs[12], fc[12]);
    full_adder full_13(es[7], hc[13], fc[12], fs[13], fc[13]);
    full_adder full_14(es[8], eco[7], ec[7], fs[14], fc[14]);
    full_adder full_15(fs[8], eco[8], ec[8], fs[15], fc[15]);


    full_adder full_16(hs[14], fc[8], fc[15], fs[16], fc[16]);
    full_adder full_17(hs[15], hc[14], fc[16], fs[17], fc[17]);
    full_adder full_18(hs[16], hc[15], fc[17], fs[18], fc[18]);
    full_adder full_19(hc[7], hc[16], fc[18], fs[19], fc[19]);

    //stage 5

    assign Y[4] = hs[17];
    assign Y[5] = fs[9];
    assign Y[6] = fs[10];
    assign Y[7] = fs[11];
    assign Y[8] = fs[12];
    assign Y[9] = fs[13];

    half_adder half_18(fs[14], fc[13], hs[18], hc[18]);
    full_adder full_20(fs[15], fc[14], hc[18], fs[20], fc[20]);
    half_adder half_19(fs[16], fc[20], hs[19], hc[19]);
    half_adder half_20(fs[17], hc[19], hs[20], hc[20]);
    half_adder half_21(fs[18], hc[20], hs[21], hc[21]);
    half_adder half_22(fs[19], hc[21], hs[22], hc[22]);

    assign Y[10] = hs[18];
    assign Y[11] = fs[20];
    assign Y[12] = hs[19];
    assign Y[13] = hs[20];
    assign Y[14] = hs[21];
    assign Y[15] = hs[22];

endmodule

```

Schematics of the Compressors used in Multiplier:

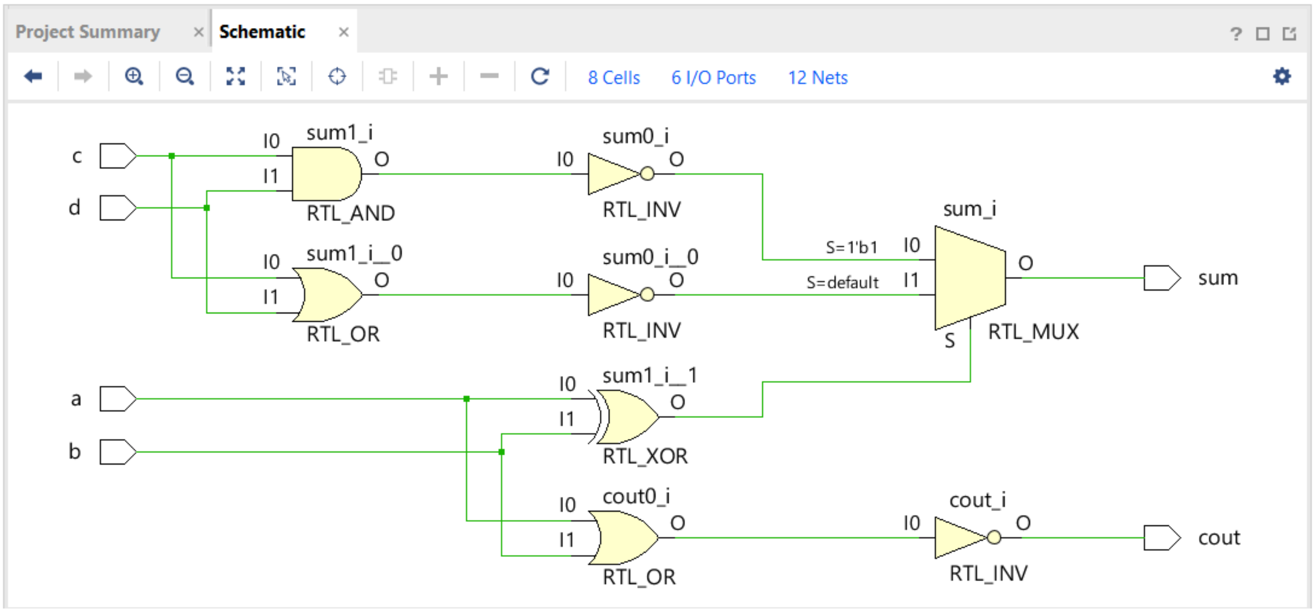
Exact 4:2 compressor:



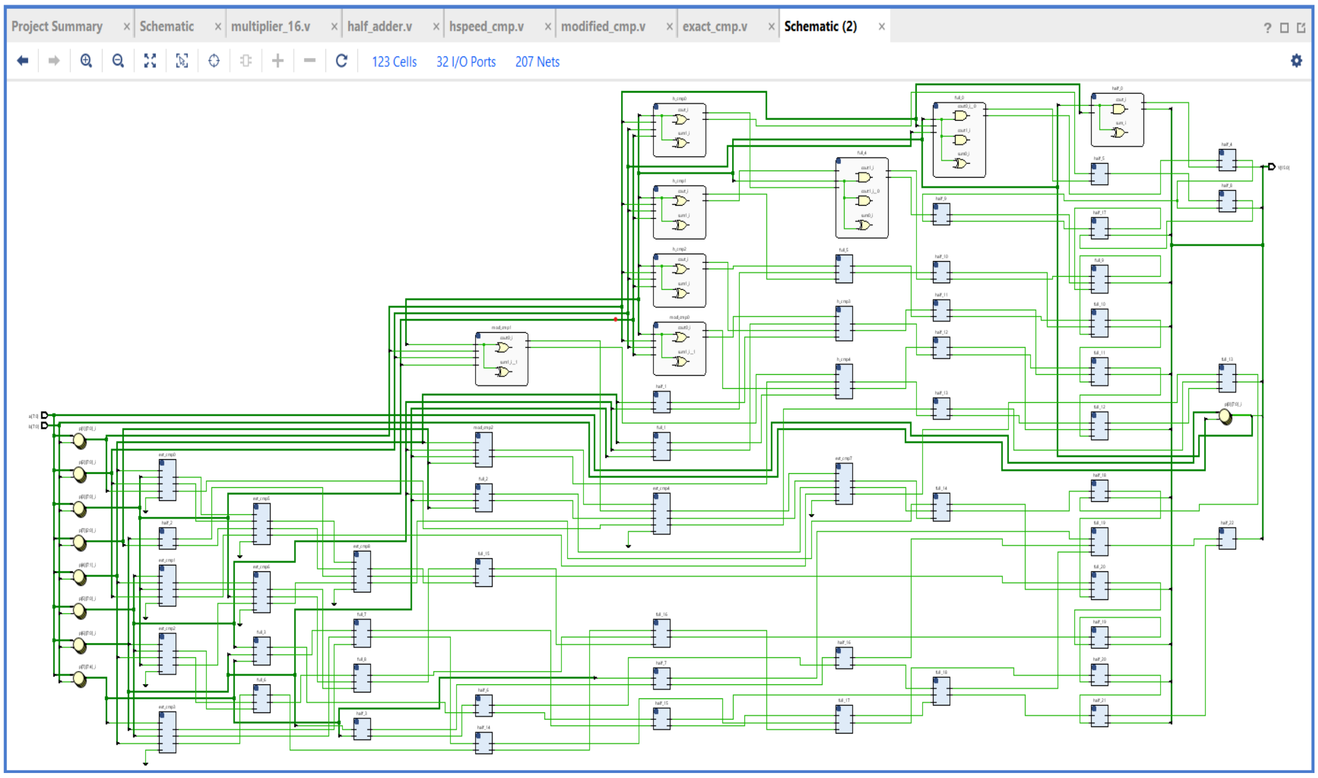
Proposed area efficient approximate 4:2 compressor:



Proposed modified dual-stage approximate 4:2 compressor:

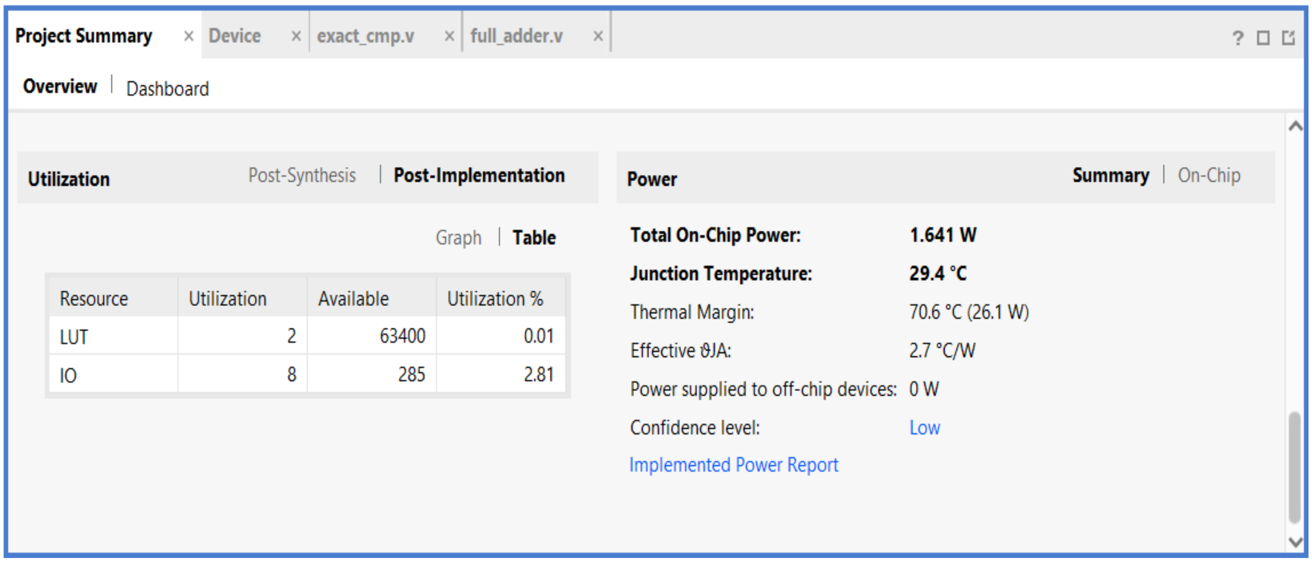


Multiplier:



Area and power observed in compressors and multiplier:

Exact 4:2 compressor:


Project Summary × | Device × | exact_cmp.v × | full_adder.v ×

Overview | Dashboard

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

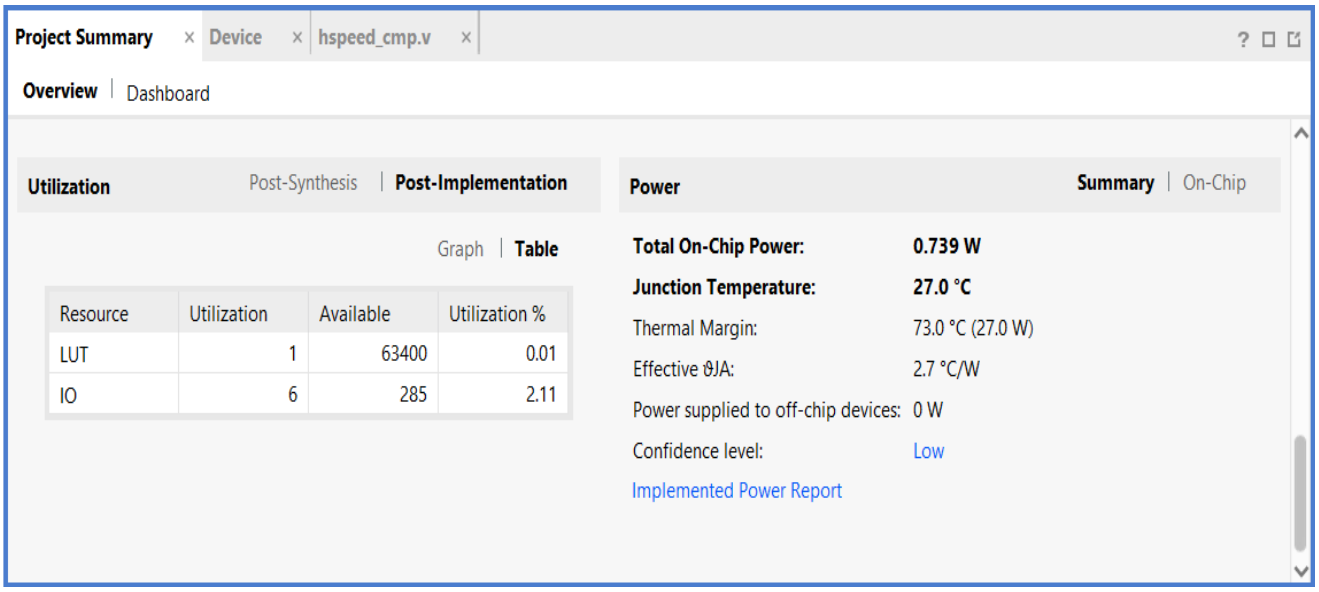
Resource	Utilization	Available	Utilization %
LUT	2	63400	0.01
IO	8	285	2.81

Power **Summary** | On-Chip

Total On-Chip Power:	**1.641 W**
Junction Temperature:	**29.4 °C**
Thermal Margin:	70.6 °C (26.1 W)
Effective ϑJA:	2.7 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

Proposed area efficient approximate 4:2 compressor:


Project Summary × | Device × | hspeed_cmp.v ×

Overview | Dashboard

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

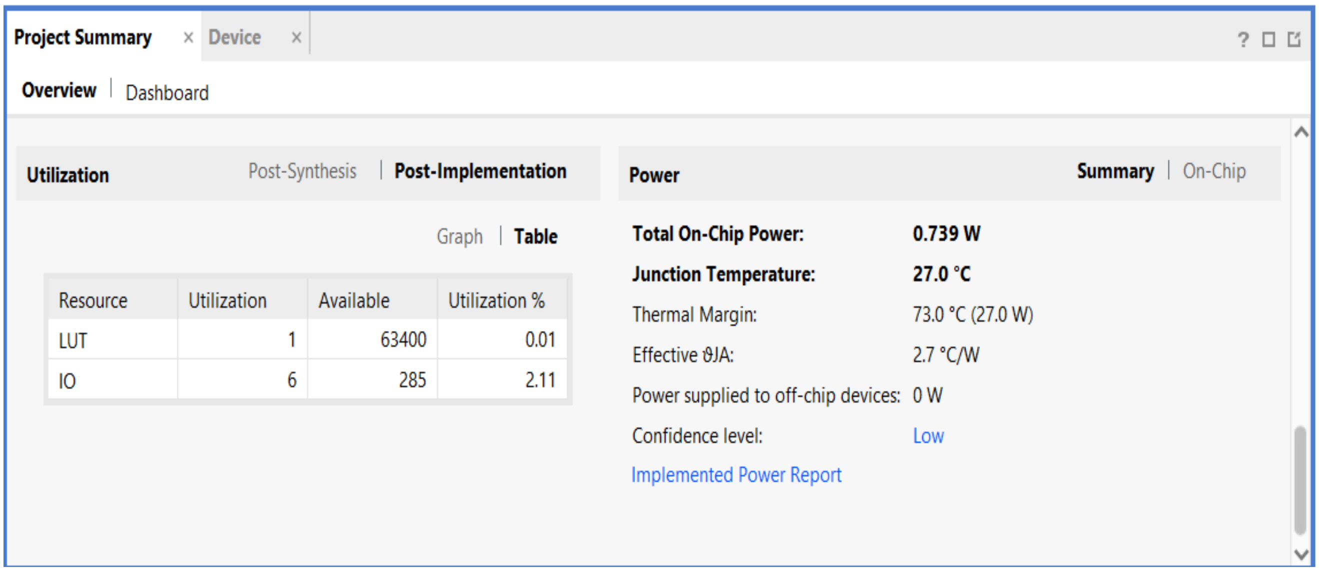
Resource	Utilization	Available	Utilization %
LUT	1	63400	0.01
IO	6	285	2.11

Power **Summary** | On-Chip

Total On-Chip Power:	**0.739 W**
Junction Temperature:	**27.0 °C**
Thermal Margin:	73.0 °C (27.0 W)
Effective ϑJA:	2.7 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

Proposed modified dual-stage approximate 4:2 compressor:



Resource	Utilization	Available	Utilization %
LUT	1	63400	0.01
IO	6	285	2.11

Total On-Chip Power:	**0.739 W**
Junction Temperature:	**27.0 °C**
Thermal Margin:	73.0 °C (27.0 W)
Effective ϑJA:	2.7 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Area Utilized:

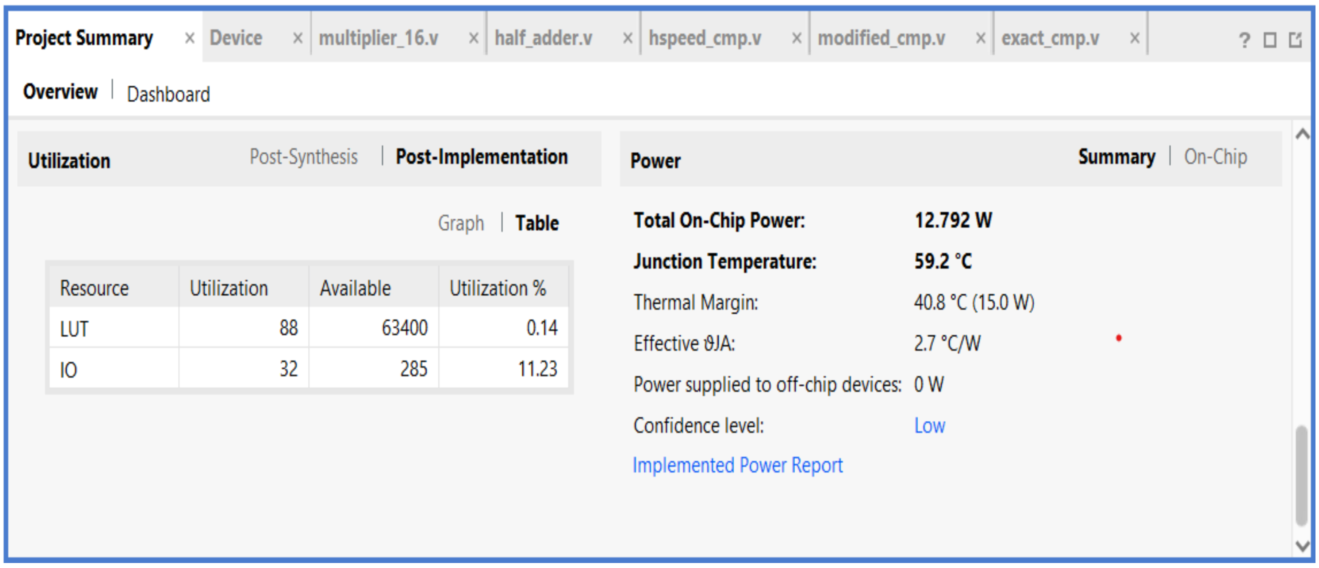
In LUT utilization, the proposed 4:2 compressors utilizes 50% less than that of the exact 4:2 compressor.

Power Consumption:

$$\left(\frac{1.641 - 0.739}{1.641}\right) \times 100 \ = \ 54.96\ \%$$

The proposed 4:2 compressors consumes 54.9% less power than the exact one.

Multiplier:


Project Summary × | Device × | multiplier_16.v × | half_adder.v × | hspeed_cmp.v × | modified_cmp.v × | exact_cmp.v ×

Overview | Dashboard

Utilization — Post-Synthesis | Post-Implementation — Graph | Table

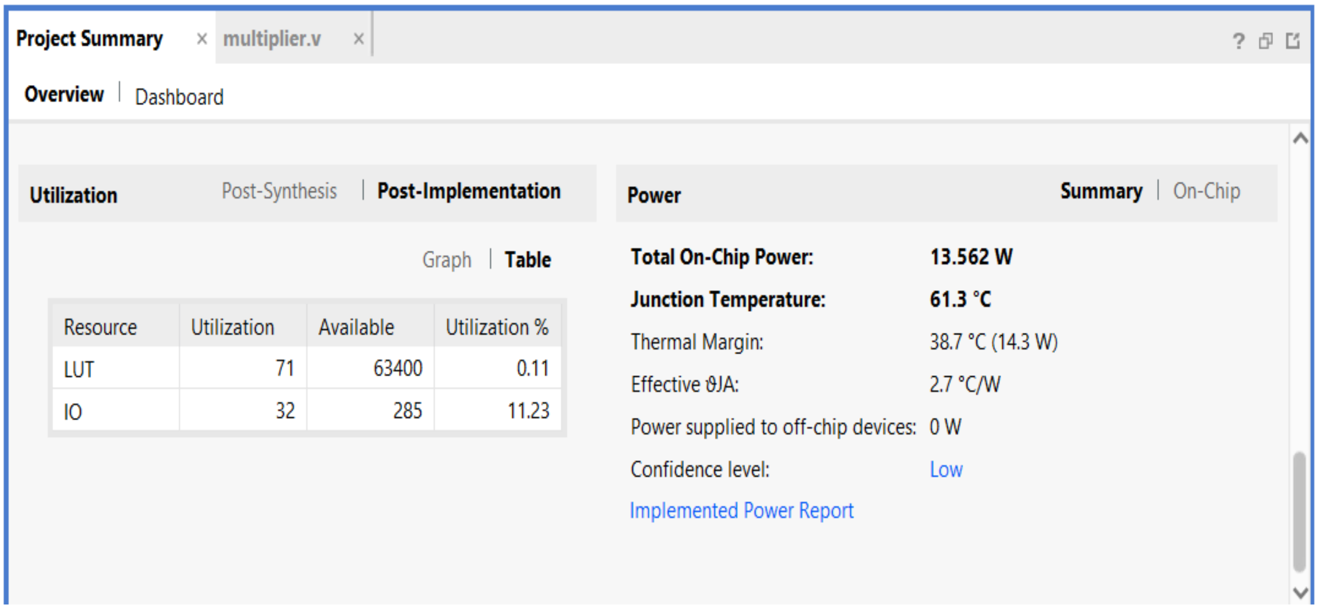
Resource	Utilization	Available	Utilization %
LUT	88	63400	0.14
IO	32	285	11.23

Power — Summary | On-Chip

Total On-Chip Power:	**12.792 W**
Junction Temperature:	**59.2 °C**
Thermal Margin:	40.8 °C (15.0 W)
Effective ϑJA:	2.7 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

By using predefined operator,


Project Summary × | multiplier.v ×

Overview | Dashboard

Utilization — Post-Synthesis | Post-Implementation — Graph | Table

Resource	Utilization	Available	Utilization %
LUT	71	63400	0.11
IO	32	285	11.23

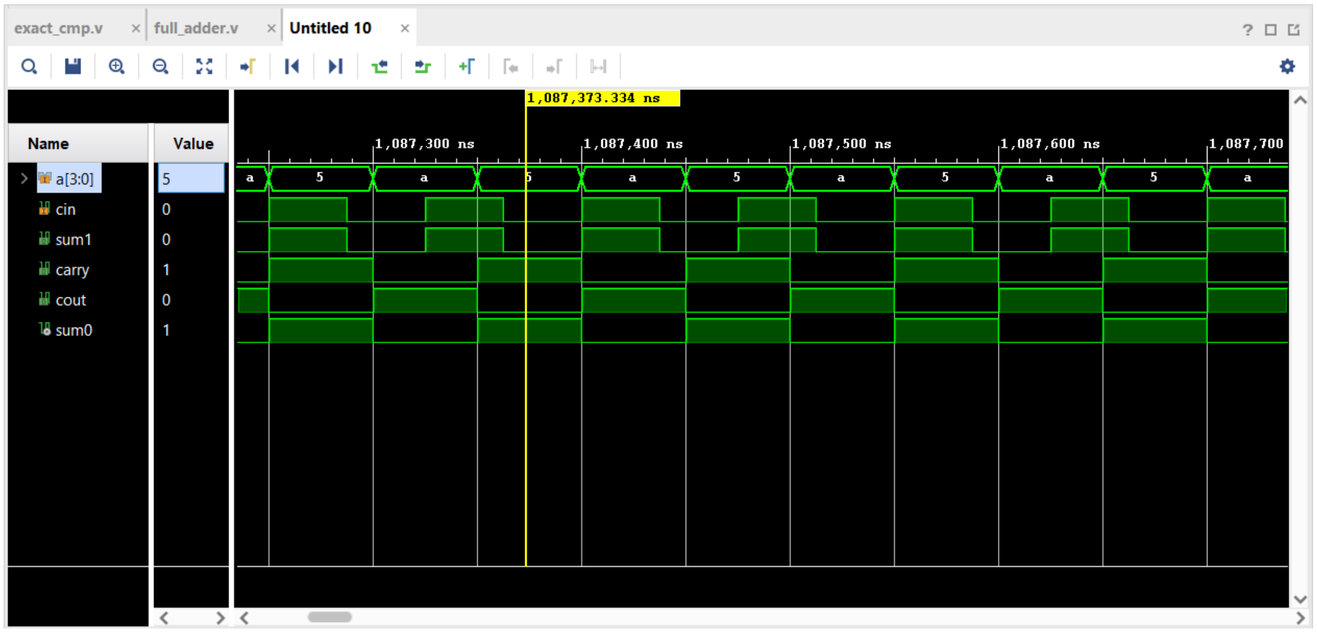
Power — Summary | On-Chip

Total On-Chip Power:	**13.562 W**
Junction Temperature:	**61.3 °C**
Thermal Margin:	38.7 °C (14.3 W)
Effective ϑJA:	2.7 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

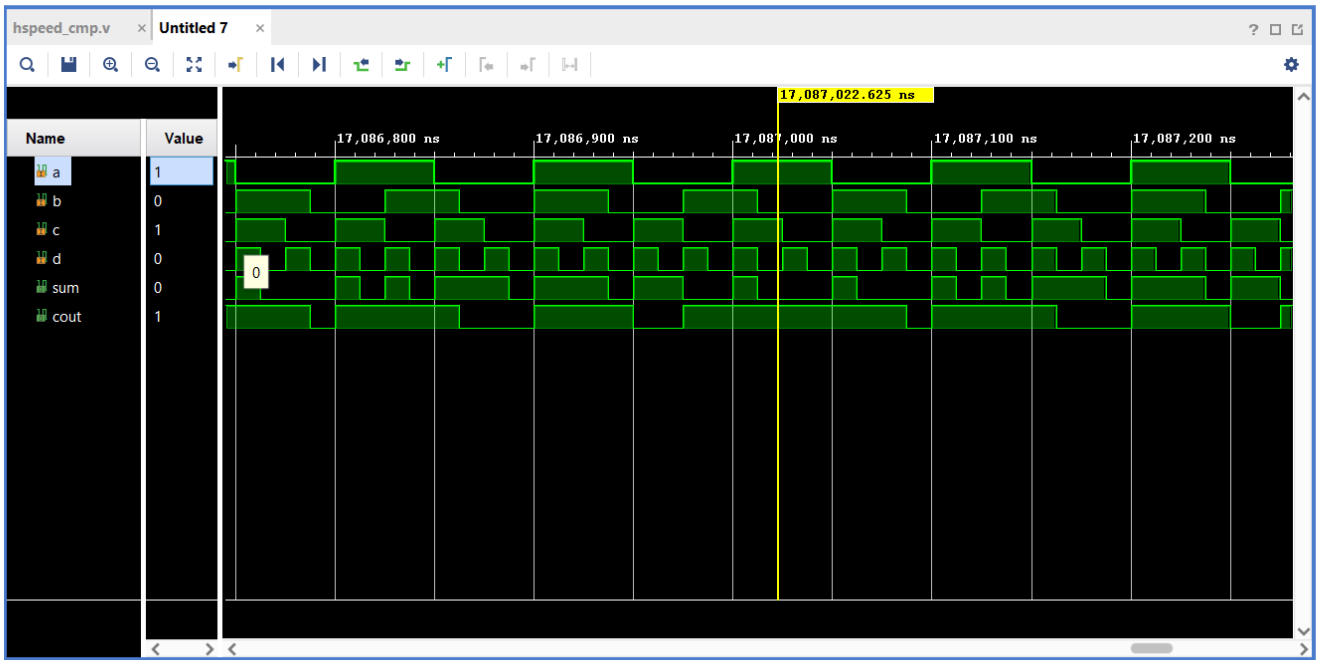
Implemented Power Report

Timing Diagrams:

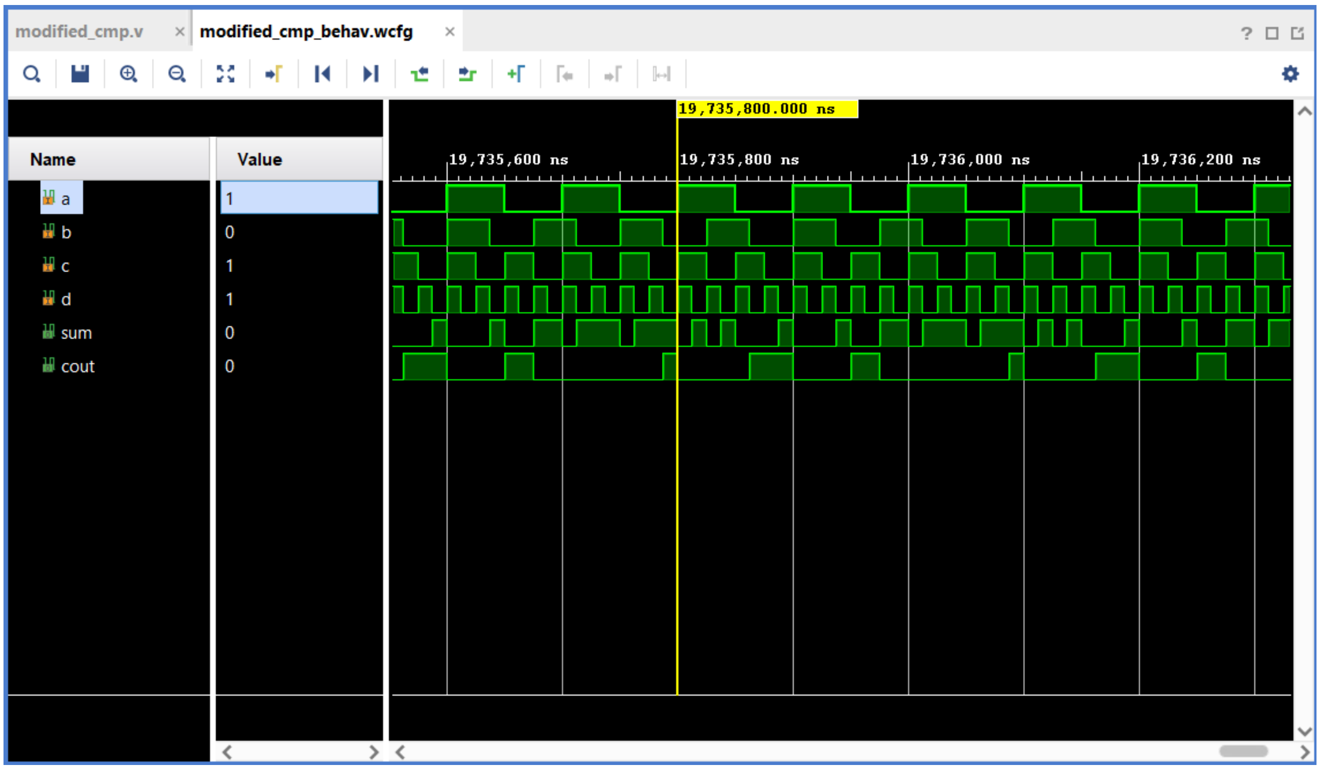
Exact 4:2 compressor:



Proposed area efficient approximate 4:2 compressor:

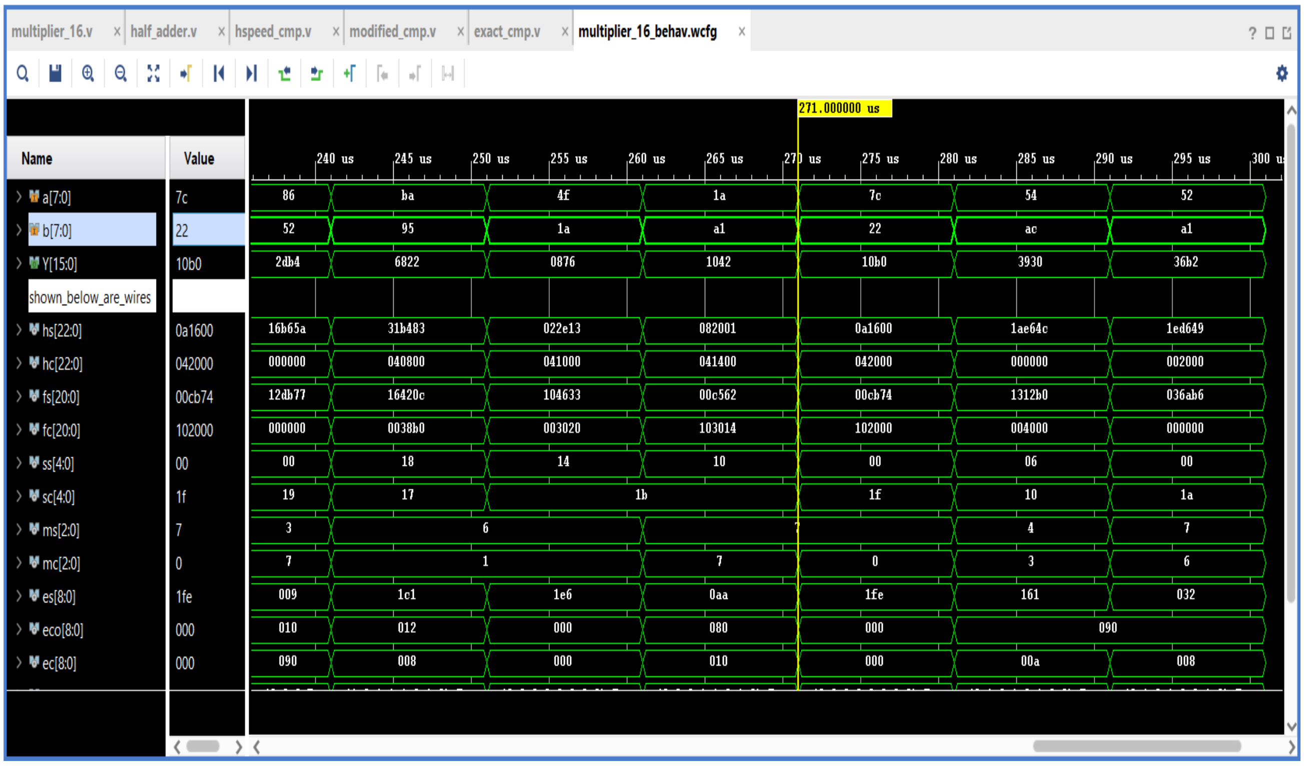


Proposed modified dual-stage approximate 4:2 compressor:



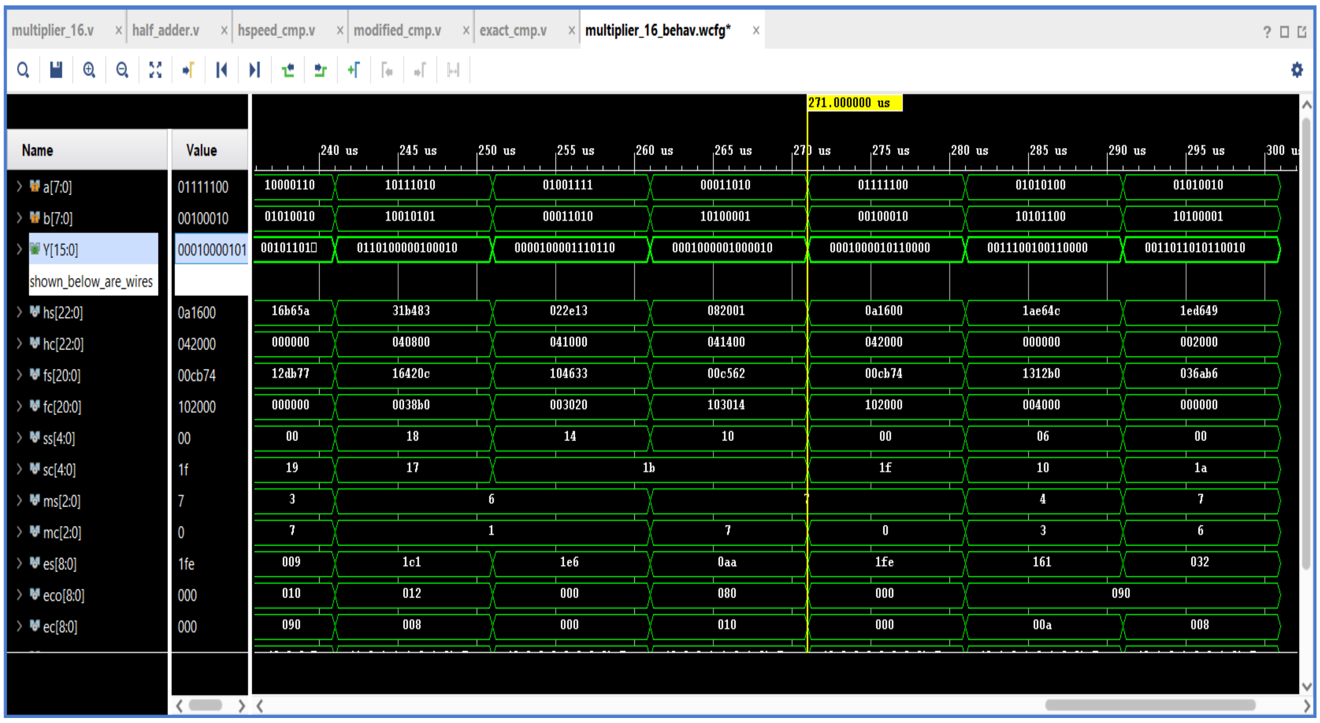
Name	Value
a	1
b	0
c	1
d	1
sum	0
cout	0

Multiplier:



```
multiplier_16.v | half_adder.v | hspeed_cmp.v | modified_cmp.v | exact_cmp.v | multiplier_16_behav.wcfg

                                              271.000000 us
Name                Value    240 us  245 us  250 us  255 us  260 us  265 us  270 us  275 us  280 us  285 us  290 us  295 us  300 us
a[7:0]              7c       86      ba      4f      1a      7c      54      52
b[7:0]              22       52      95      1a      a1      22      ac      a1
Y[15:0]             10b0     2db4    6822    0876    1042    10b0    3930    36b2
shown_below_are_wires
hs[22:0]            0a1600   16b65a  31b483  022e13  082001  0a1600  1ae64c  1ed649
hc[22:0]            042000   000000  040800  041000  041400  042000  000000  002000
fs[20:0]            00cb74   12db77  16420c  104633  00c562  00cb74  1312b0  036ab6
fc[20:0]            102000   000000  0038b0  003020  103014  102000  004000  000000
ss[4:0]             00       00      18      14      10      00      06      00
sc[4:0]             1f       19      17      1b              1f      10      1a
ms[2:0]             7        3       6               7               4       7
mc[2:0]             0        7       1               7       0       3       6
es[8:0]             1fe      009     1c1     1e6     0aa     1fe     161     032
eco[8:0]            000      010     012     000     080     000     090
ec[8:0]             000      090     008     000     010     000     00a     008
```

In binary representation,



```
multiplier_16.v | half_adder.v | hspeed_cmp.v | modified_cmp.v | exact_cmp.v | multiplier_16_behav.wcfg*

                                              271.000000 us
Name                Value          240 us  245 us  250 us  255 us  260 us  265 us  270 us  275 us  280 us  285 us  290 us  295 us  300 us
a[7:0]              01111100       10000110  10111010  01001111  00011010  01111100  01010100  01010010
b[7:0]              00100010       01010010  10010101  00011010  10100001  00100010  10101100  10100001
Y[15:0]             00010000101    00101101  0110100000100010  0000100001110110  0001000001000010  0001000010110000  0011100100110000  0011011010110010
shown_below_are_wires
hs[22:0]            0a1600         16b65a  31b483  022e13  082001  0a1600  1ae64c  1ed649
hc[22:0]            042000         000000  040800  041000  041400  042000  000000  002000
fs[20:0]            00cb74         12db77  16420c  104633  00c562  00cb74  1312b0  036ab6
fc[20:0]            102000         000000  0038b0  003020  103014  102000  004000  000000
ss[4:0]             00             00      18      14      10      00      06      00
sc[4:0]             1f             19      17      1b              1f      10      1a
ms[2:0]             7              3       6               7               4       7
mc[2:0]             0              7       1               7       0       3       6
es[8:0]             1fe            009     1c1     1e6     0aa     1fe     161     032
eco[8:0]            000            010     012     000     080     000     090
ec[8:0]             000            090     008     000     010     000     00a     008
```

Conclusion:

Approximate 8-bit multiplier is designed using proposed area efficient approximate 4:2 compressor, proposed modified dual-stage approximate 4:2 compressor and the 4:2 exact compressors. In the 8 × 8 approximate multiplier, Level 1 to Level 8 employ approximate compressors and Level 9 to Level 15 employs exact compressors. When compared to an exact compressor, the proposed compressors have reduction in area and power by 50 and 54.96% respectively. So, the multiplier with proposed design is able to achieve a reduction in area, delay, power. This will be useful for the AI and DSP applications for faster computions with a nominal accuracy.