Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved
NAAC A++ Accredited

Division of Electronics and Communication Engineering
2024-2025 (ODD SEM)

REPORT

for

SoC Design

Title of the Report:

Power Optimization of BCD Adder Design and Implementation in BASYS3 FPGA Board

A report submitted by

Name of the Student	**ALDRIN G**
Register Number	***URK22EC1019***
Subject Name	***SoC Design***
Subject Code	***19EC2045***
Date of Report submission	***28/03/2025***

Project Rubrics for Evaluation

First Review: Project title selection - PPT should have four slides (Title page, Introduction, Circuit/Block Diagram, and Description of Project).

Second Review: PPT should have three slides (Description of Concept, implementation, outputs, results and discussion)

Rubrics for project (III IA - 40 Marks):
Content - 4 marks (based on Project)
Clarity - 3 marks (based on viva during presentation) Feasibility - 3 marks (based on project)
Presentation - 10 marks Project
Report - 10 marks
On-time submission - 5 marks (before the due date) Online submission-GCR - 5 marks

Total marks:_____/ 40 Marks

Signature of Faculty with date:

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Abstract

The implementation of a Binary-Coded Decimal (BCD) adder is crucial in arithmetic circuits, particularly in financial and digital display applications. This report explores the design and implementation of a BCD adder using different Verilog coding styles, including structural, behavioral, and dataflow modeling. Each coding style is analyzed in terms of its impact on design complexity, resource utilization, and performance. The designed BCD adder is synthesized and implemented on the Basys 3 FPGA board to validate its functionality. Further, power optimization techniques are applied to enhance the efficiency of the design. The modified Verilog code incorporates strategies such as clock gating and logic optimization to minimize dynamic and static power consumption. Power analysis is conducted before and after optimization using FPGA design tools to quantify the improvements. The results demonstrate a trade-off between power consumption, speed, and area, providing insights into optimizing digital arithmetic circuits for low-power applications.

In digital circuit design, a BCD (Binary-Coded Decimal) Adder is a crucial component used to add two decimal digits represented in binary form. Unlike a standard binary adder, a BCD adder must account for the fact that each digit in a decimal number is represented by a 4-bit binary number. When the sum of these 4-bit numbers exceeds the value of 9 (1001 in binary), the BCD adder adjusts the result to maintain proper decimal representation.

This report focuses on the design and implementation of a BCD Adder using Verilog, a widely used hardware description language (HDL). The design is explored through three different modeling techniques in Verilog: dataflow, behavioral, and structural modeling. Each modeling approach offers unique advantages and perspectives on how the circuit can be represented and simulated.

- **Dataflow Modeling** focuses on the movement of data within the circuit and is commonly used to describe combinational logic. It utilizes continuous assignments and logical operators to define the connections between inputs and outputs.

- **Behavioral Modeling** offers a higher-level abstraction of the circuit, concentrating on the system's functionality and behavior rather than detailing specific data paths. This approach is particularly useful for complex designs where the primary concern is the desired behavior rather than the exact implementation.

- **Structural Modeling** depicts the circuit at the gate or module level, showing how various components are interconnected to create the complete circuit. This method closely aligns with the actual hardware implementation.

CHAPTER 2

DESIGN DESCRIPTION AND WORKING

A Binary-Coded Decimal (BCD) adder is designed to add two decimal digits represented in 4-bit binary form. Each digit in a BCD number ranges from 0000 to 1001 (representing decimal 0 to 9). When adding two BCD digits, if the sum exceeds the BCD range (i.e., 1010 or higher), a correction must be applied to ensure the result is a valid BCD digit. This is achieved by adding 6 (0110 in binary) to the sum if it exceeds 9, effectively correcting the binary sum back into the BCD format.

The BCD adder typically consists of the following components:

1. **4-Bit Binary Adder**: This initial adder takes two 4-bit BCD digits and a possible carry-in bit from a previous operation, producing a 4-bit sum and a carry-out bit. However, this sum may not be a valid BCD number.
2. **Correction Logic**: The correction logic checks if the sum is greater than 9 or if there is a carry-out from the binary addition. If either condition is true, the logic adds 6 to the sum, converting it into a valid BCD number.
3. **Second 4-Bit Adder**: This adder performs the correction by adding 6 to the invalid sum, producing the final BCD result.
4. **Output**: The final output consists of a 4-bit BCD sum and a carry-out bit, which may be used in the next BCD addition if working with multi-digit numbers.

Functional Overview

The BCD adder performs the following steps to ensure a correct Binary-Coded Decimal (BCD) sum:

- **Step 1: Initial 4-Bit Binary Addition**
  - Given two 4-bit BCD digits $A=A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$, along with a carry-in C_{in} (which could be 0 or 1 depending on whether there's a previous carry), the adder computes:

    $$S=A+B+C_{in}$$

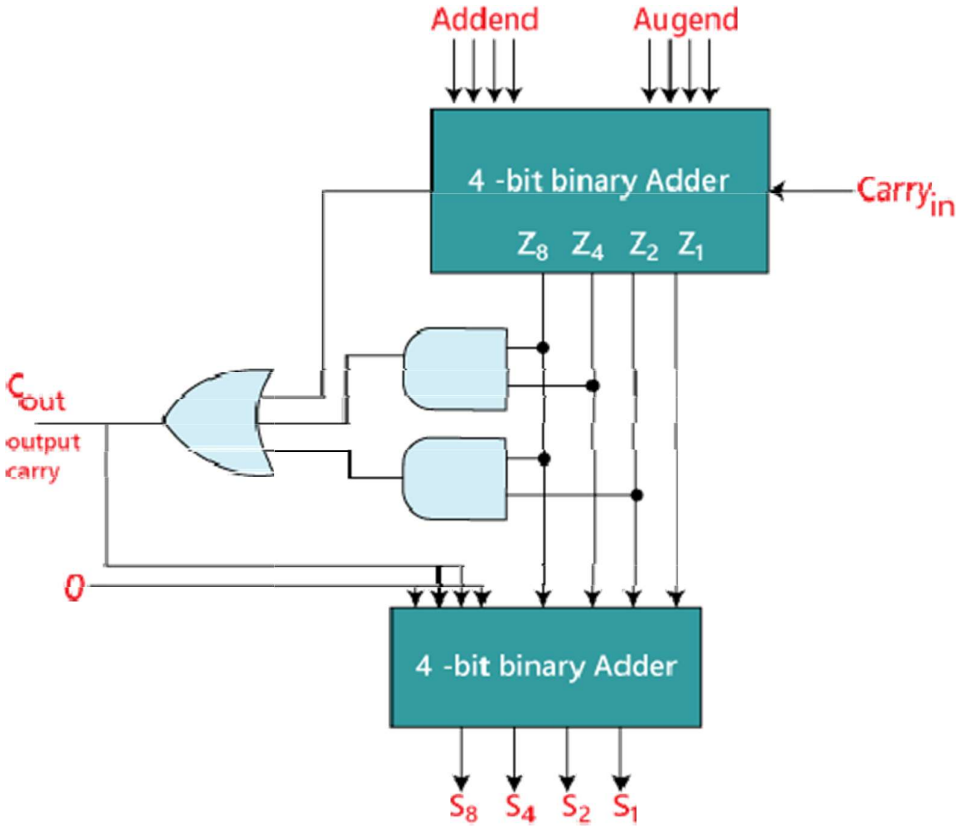
  - Here, $S = S_3S_2S_1S_0$ is a 4-bit sum, and C_{out1} is the carry-out of the initial addition.
- **Step 2: Detection of Invalid BCD Output**
  - The BCD output is invalid if:
    - $S > 9$ (i.e., S is greater than 1001 in binary)
    - $C_{out1} = 1$ (indicating that the sum has overflowed beyond 4 bits)
  - The condition $S > 9$ can be checked using the following logic equation:

    $$X=S3\cdot S2+S3\cdot S1 \text{ (i.e., when the sum exceeds 9)}$$

  - The correction is needed when $X = 1$ or $C_{out1} = 1$.
- **Step 3: Correction Logic**
  - If $X = 1$ or $C_{out1} = 1$, add 6 (0110 in binary) to the sum S to correct the result:

    $$S' = S + 0110$$

  - The final sum $S'=S_3'S_2'S_1'S_0'$ is the correct BCD digit, and C_{out2} is the new carry-out.



Block Diagram of a BCD Adder

Binary Sum						BCD Sum						Decimal
K	Z_8	Z_4	Z_2	Z_1		C	S_8	S_4	S_2	S_1		
0	0	0	0	0	SAME CODE	0	0	0	0	0		0
0	0	0	0	1		0	0	0	0	1		1
0	0	0	1	0		0	0	0	1	0		2
.	.	.	.	.		.	.	.	.	.		.
.	.	.	.	.		.	.	.	.	.		.
.	.	.	.	.		.	.	.	.	.		.
.	.	.	.	.		.	.	.	.	.		.
0	1	0	0	0		0	1	0	0	0		8
0	1	0	0	1		0	1	0	0	1		9
10 to 19 Binary and BCD codes are not the same												
0	1	0	1	0		1	0	0	0	0		10
0	1	0	1	1		1	0	0	0	1		11
0	1	1	0	0		1	0	0	1	0		12
0	1	1	0	1		1	0	0	1	1		13
0	1	1	1	0		1	0	1	0	0		14
0	1	1	1	1		1	0	1	0	1		15
1	0	0	0	0		1	0	1	1	0		16
1	0	0	0	1		1	0	1	1	1		17
1	0	0	1	0		1	1	0	0	0		18
1	0	0	1	1		1	1	0	0	1		19

BCD ADDER TRUTH TABLE

CHAPTER 3

VARIOUS DESIGN APPROACHES

Dataflow Modelling:

- **Verilog code:**

```
module bcd_adder_dataflow( input
    [3:0] A, B,
    input Cin,
    output [3:0] Sum,
    output Cout
);
    wire [4:0] binary_sum;

    assign binary_sum = A + B + Cin;
    assign {Cout, Sum} = (binary_sum > 9) ? (binary_sum + 6) : binary_sum;

endmodule


module tb_bcd_adder_dataflow; reg
    [3:0] A, B;
    reg Cin;
    wire [3:0] Sum; wire
    Cout;

    bcd_adder_dataflow uut (.A(A), .B(B), .Cin(Cin), .Sum(Sum), .Cout(Cout)); initial begin
        A = 4'b0101; B = 4'b0110; Cin = 0; #10;
        A = 4'b1001; B = 4'b0001; Cin = 0; #10; A
        = 4'b0011; B = 4'b0101; Cin = 0; #10;
        $stop;
    end
endmodule
```

Structural Modelling:

- **Verilog Code:**

```
module half_adder(
    input A, B, output
    Sum, Cout
);
    assign Sum = A ^ B; assign
    Cout = A & B;
endmodule

module full_adder(
    input A, B, Cin,
    output Sum, Cout
);
    wire sum1, carry1, carry2;

    half_adder ha1 (.A(A), .B(B), .Sum(sum1), .Cout(carry1));
    half_adder ha2 (.A(sum1), .B(Cin), .Sum(Sum), .Cout(carry2)); assign Cout
    = carry1 | carry2;
endmodule

module bcd_adder_structural( input
    [3:0] A, B,
    input Cin,
    output [3:0] Sum,
    output Cout
);
    wire [3:0] binary_sum;
    wire carry1, carry2, carry3, carry4, z; wire
    [3:0] correction;
    wire [3:0] corrected_sum;

    assign correction = 4'b0110; // correction value is 6 in binary

    // Step 1: Binary addition of A and B
    full_adder fa0(.A(A[0]), .B(B[0]), .Cin(Cin), .Sum(binary_sum[0]), .Cout(carry1));
    full_adder fa1(.A(A[1]), .B(B[1]), .Cin(carry1), .Sum(binary_sum[1]),
.Cout(carry2));
    full_adder fa2(.A(A[2]), .B(B[2]), .Cin(carry2), .Sum(binary_sum[2]),
.Cout(carry3));
    full_adder fa3(.A(A[3]), .B(B[3]), .Cin(carry3), .Sum(binary_sum[3]), .Cout(z));
```

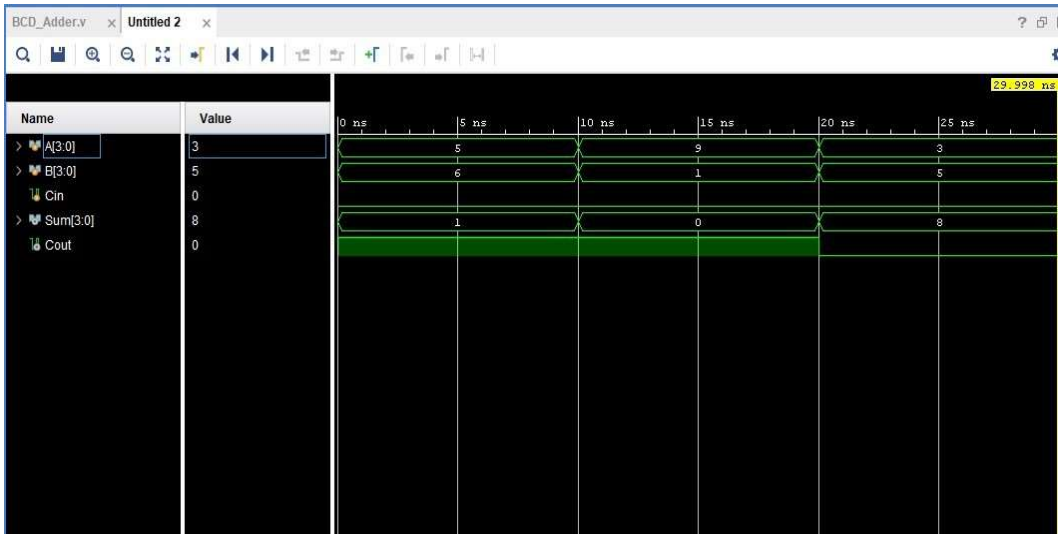
```
    // Step 2: Check if correction is needed
    assign Cout = z | (binary_sum[3] & (binary_sum[2] | binary_sum[1]));

    // Step 3: Add correction value if necessary
    full_adder fa4(.A(binary_sum[0]), .B(correction[0]), .Cin(0),
.Sum(corrected_sum[0]), .Cout(carry4));
    full_adder fa5(.A(binary_sum[1]), .B(correction[1]), .Cin(carry4),
.Sum(corrected_sum[1]), .Cout(carry5));
    full_adder fa6(.A(binary_sum[2]), .B(correction[2]), .Cin(carry5),
.Sum(corrected_sum[2]), .Cout(carry6));
    full_adder fa7(.A(binary_sum[3]), .B(correction[3]), .Cin(carry6),
.Sum(corrected_sum[3]), .Cout());

    // Step 4: Select the final sum based on whether correction was applied assign Sum =
    Cout ? corrected_sum : binary_sum;
endmodule



module tb_bcd_adder_structural; reg
    [3:0] A, B;
    reg Cin;
    wire [3:0] Sum; wire
    Cout;

    bcd_adder_structural uut (.A(A), .B(B), .Cin(Cin), .Sum(Sum), .Cout(Cout)); initial begin
        A = 4'b0101; B = 4'b0110; Cin = 0; #10;
        A = 4'b1001; B = 4'b0001; Cin = 0; #10; A
        = 4'b0011; B = 4'b0101; Cin = 0; #10;
        $stop;
    end
endmodule
```

Behavioural Modelling:

- **Verilog code:**

```
module bcd_adder_behavioural( input
    [3:0] A, B,
    input Cin,
    output reg [3:0] Sum,
    output reg Cout
);
    reg [4:0] binary_sum;

    always @(*) begin binary_sum
        = A + B + Cin; if
        (binary_sum > 9) begin
            binary_sum = binary_sum + 6;
            Cout = 1;
        end else begin
            Cout = 0;
        end
        Sum = binary_sum[3:0]; end
endmodule


module tb_bcd_adder_behavioural; reg
    [3:0] A, B;
    reg Cin;
    wire [3:0] Sum; wire
    Cout;

    bcd_adder_behavioural uut (.A(A), .B(B), .Cin(Cin), .Sum(Sum), .Cout(Cout)); initial

    begin
        A = 4'b0101; B = 4'b0110; Cin = 0; #10;
        A = 4'b1001; B = 4'b0001; Cin = 0; #10; A
        = 4'b0011; B = 4'b0101; Cin = 0; #10;
        $stop;
    end
endmodule
```

SIMULATION OUTPUT:

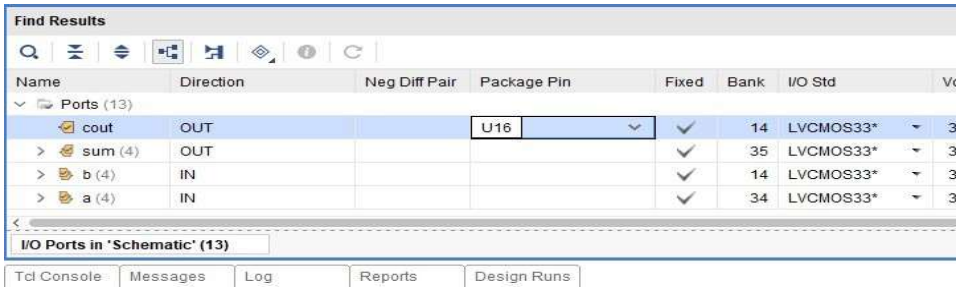


CHATPER 4

HARDWARE IMPLEMENTATION IN BASYS3 FPGA BOARD

Procedure For Implementation of the Verilog design in Basys3 FPGA Board:

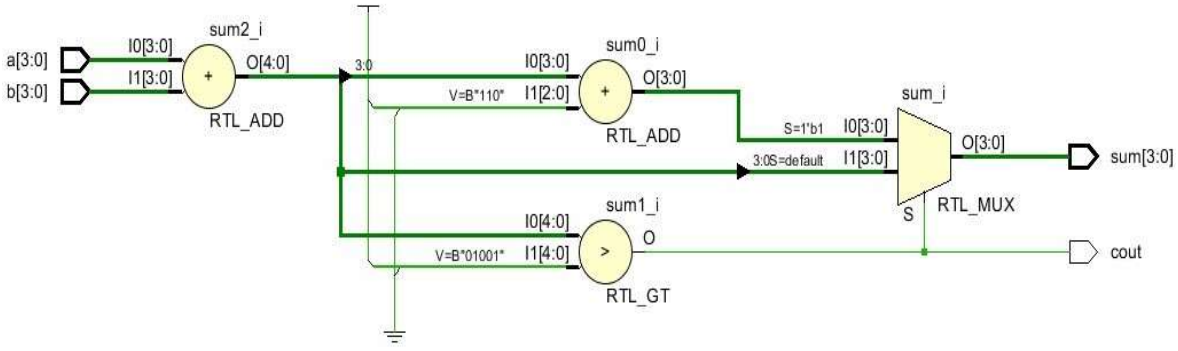
1. Create a New RTL Project Named “BCD_Adder” and create a verilog file named “BCD_Adder in add sources
2. Select the part name “XC7A35TCPG236-1” to select the BASYS 3 FPGA board
3. Write your verilog code and save it
4. Click Run Simulation Run Behavioural Simulation, Check the correctness of the code by forcing constant as input and verifying the output waveform
5. Click “Open Elabourated Design” under RTL Analysis
6. Click the I/O ports option in the schematic generated, a “Find Results” windows open:



- Select I/O std as “LVCMOS33” in all ports
- Select the package pin for each port as per the pin out/pin name given in the board as required for the design
- Click “Ctrl+S” and give a file name in the prompt opened to save the constraints

7. Click “Run Synthesis” under Synthesis
    - Change the ‘Number of jobs’ as ‘4’ in the prompt which opens
    - Click ‘OK’
8. After Synthesis is completed, a prompt opens:
    - Select ‘Run Implementation’ option
    - Click ‘OK’
9. After Implementation is completed, a prompt opens:
    - Select ‘Open Implemented Design’
    - Click ‘OK’
    - A prompt opens to close the elaborated design Click ‘Yes’
10. Click “Generate Bitstream” under Program and Debug
    - Click Ok in the prompt which opens
11. After Bitstream is generated, a prompt opens:
    - Click ‘OK’
12. Now connect the BASYS 3 FPGA board to PC
13. Click “Open Hardware Manager Open Target” under Program and Debug
    - Click Auto-connect
    - After successful connection, click “Program device”
14. Now vary the input for different cases, and note the output
15. Verify with the expected output

Block Diagram:



Implementation:

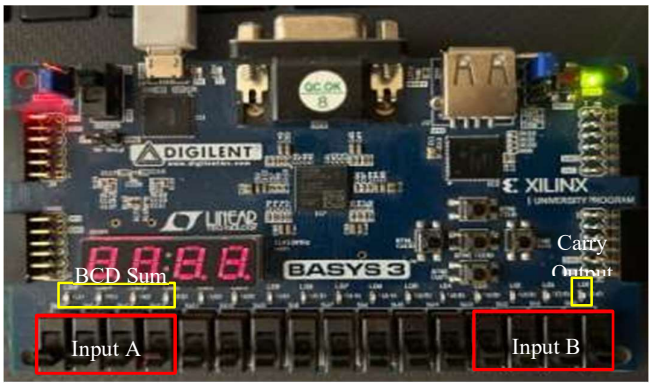
1. **VHDL/Verilog Code:**
   - The design was implemented using Verilog.
   - BCD Adder Verilog code using Behavioral Modelling is given below:

```
module bcd_adder (
   input  [3:0] a,
   input  [3:0] b,
   output reg [3:0] sum,
   output reg cout
);

reg [4:0] temp_sum;

always @(*) begin
   temp_sum = a + b;

   if (temp_sum > 4'd9) begin
      sum  = temp_sum + 4'd6;
      cout = 1;
   end else begin
      sum  = temp_sum[3:0];
      cout = 0;
   end
end
endmodule
```

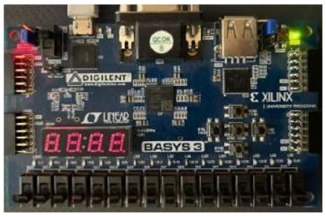
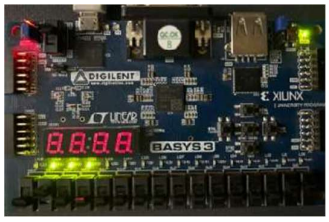
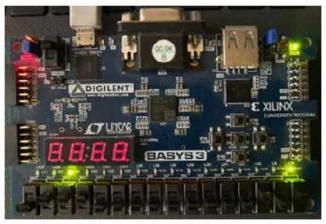
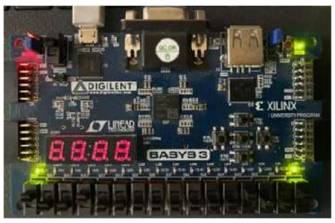
2. **FPGA Synthesis:**
   - The above Verilog code was synthesized using the Vivado tool.
   - The design was loaded onto the Basys 3 FPGA board for testing.
   -

3. **Input and Output Mapping:**
   - Inputs (A, B) were mapped to switches, and the sum and carry was displayed on the Inbuilt LED's.

Port Mapping:



Results:

Case 1: **0 + 0 (in BCD)**: • A = 0000 (0 in BCD) • B = 0000 (0 in BCD)  Sum: 0000 Carry: 0	**Case 2:** **4 + 3 (in BCD)**: • A = 0100 (4 in BCD) • B = 0011 (3 in BCD)  Sum: 0111 Carry: 0
Case 3: **7 + 5 (in BCD)**: • A = 0111 (7 in BCD) • B = 0101 (5 in BCD)  Sum: 0010 Carry: 1	**Case 4:** **9 + 9 (in BCD)**: • A = 1001 (9 in BCD) • B = 1001 (9 in BCD)  Sum: 1000 Carry: 1

CHATPER 5

POWER OPTIMIZED DESIGN USING CLOCK GATING

Power Optimized Code using Clock Gating Technique:

```
module bcd_adder ( input
    [3:0] a,
    input [3:0] b,
    output reg [3:0] sum,
    output reg cout
);
    reg [4:0] temp_sum;
    reg [3:0] prev_a, prev_b;

    always @(*)
    begin
        // Check if there is any change in the inputs if (a !=
        prev_a || b != prev_b)
        begin
            temp_sum = a + b;
            prev_a = a; prev_b
            = b;

            if (temp_sum > 4'd9) begin
                sum = temp_sum + 4'd6; cout
                = 1;
            end else
            begin
                sum = temp_sum[3:0]; cout =
                0;
            end
        end
    end endmodule
```

Key Differences after Design Optimization

Objective: Reduce toggling activity in the BCD adder to save power by updating only when inputs change.

Modifications:

1. **Previous State Registers (prev_a, prev_b)**: Store the last state of inputs aand b.
2. **Condition for Computation**: Add a check in the always block to update temp_sum
   only when aor bchanges.

Result: This minimizes unnecessary updates, reducing switching activity and saving power by avoiding redundant computations.

Applying Clock Gating for Power Optimization

1. **Purpose of Clock Gating**: Reduces power consumption by activating logic only when necessary.
2. **Clock Gating Mechanism**: Adds a clock enable signal to control when the logic block executes.
3. **Effect on BCD Adder**: Ensures the BCD adder updates only when there's a change in inputs, reducing unnecessary computations.
4. **Simulation in Combinational Logic**:
   - Since the BCD adder is a combinational block (without a clock signal), we simulate clock gating.
   - Implement a condition to perform addition only if aor bchanges.
5. **Power Savings**: Minimizes toggling activity and power consumption by preventing redundant updates to temp_sum.

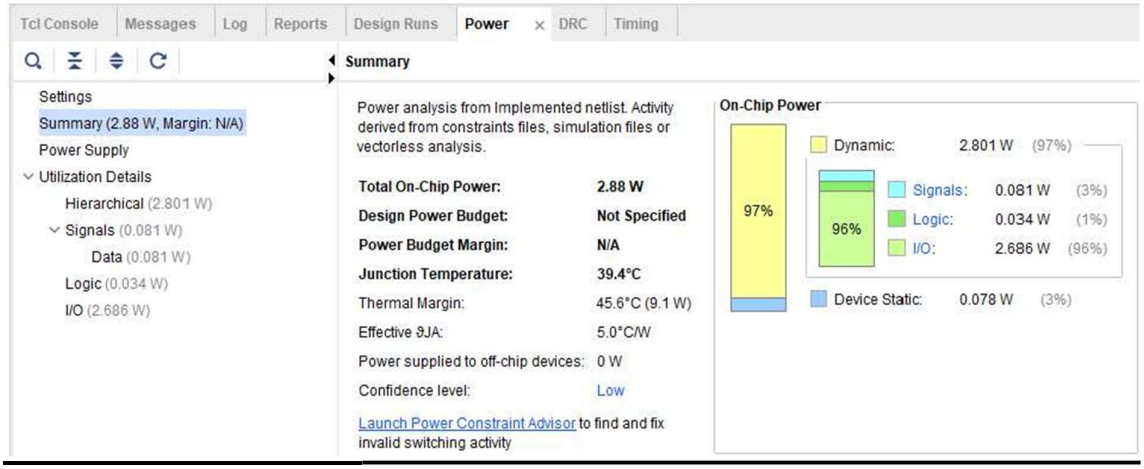
Unoptimized BCD Adder:

Power Report Summary:

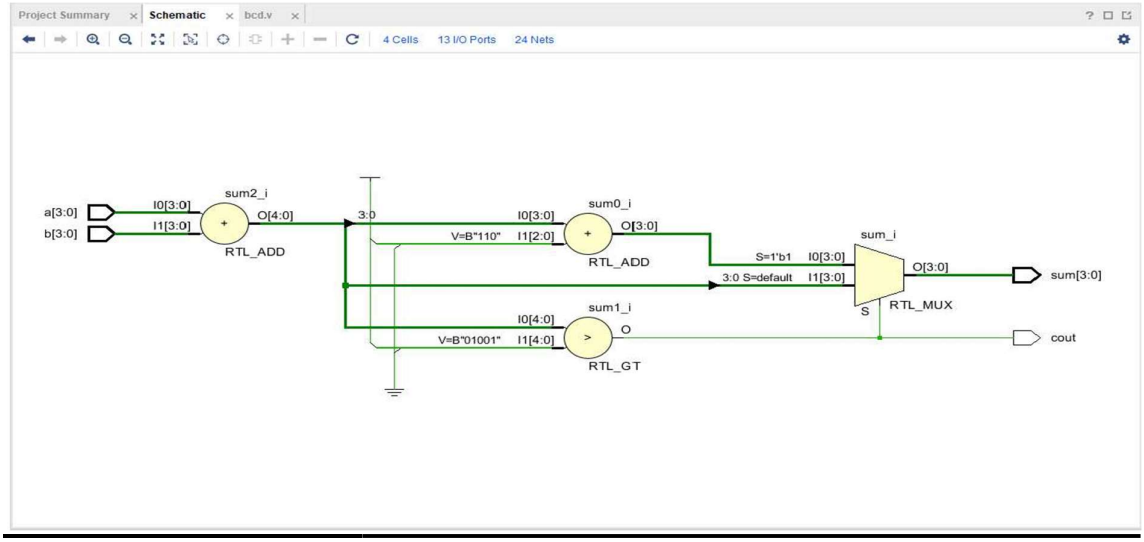
Power — **Summary** | On-Chip

Total On-Chip Power:	**2.88 W**
Junction Temperature:	**39.4 °C**
Thermal Margin:	45.6 °C (9.1 W)
Effective ϑJA:	5.0 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

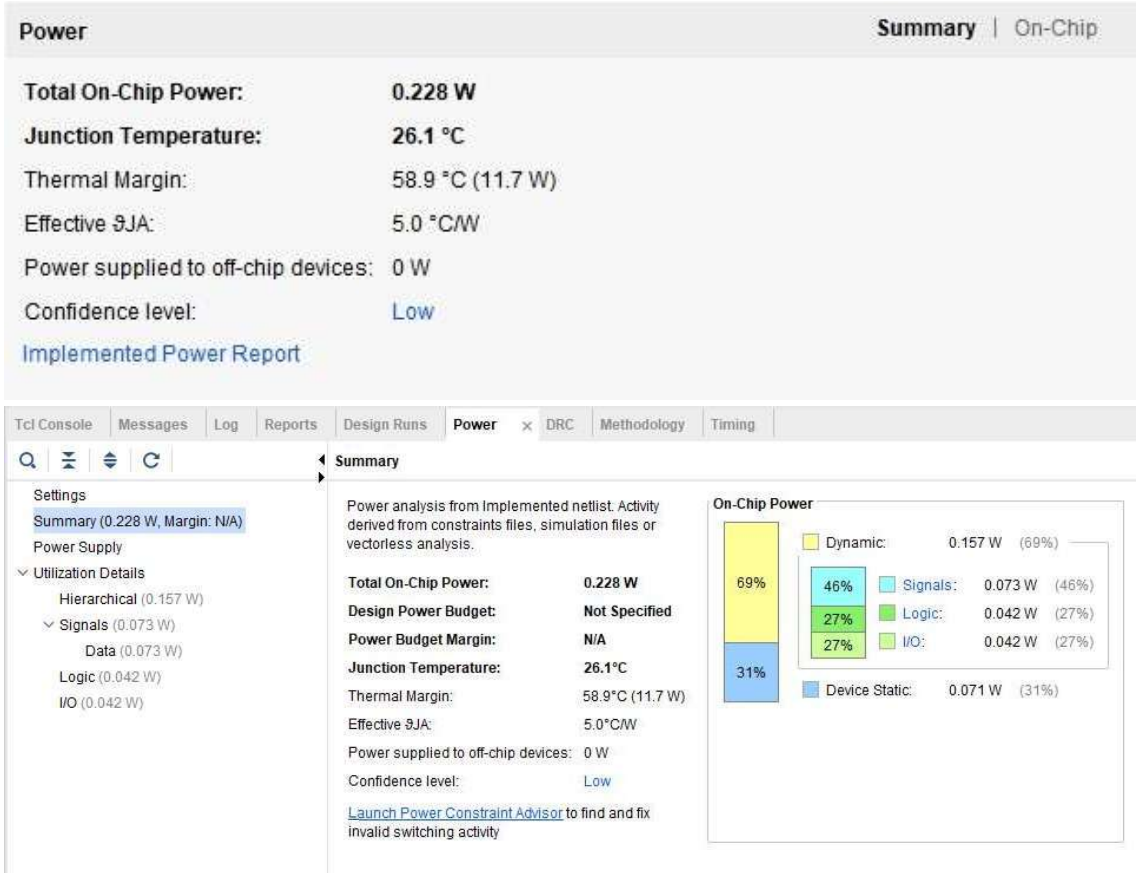


Schematic:

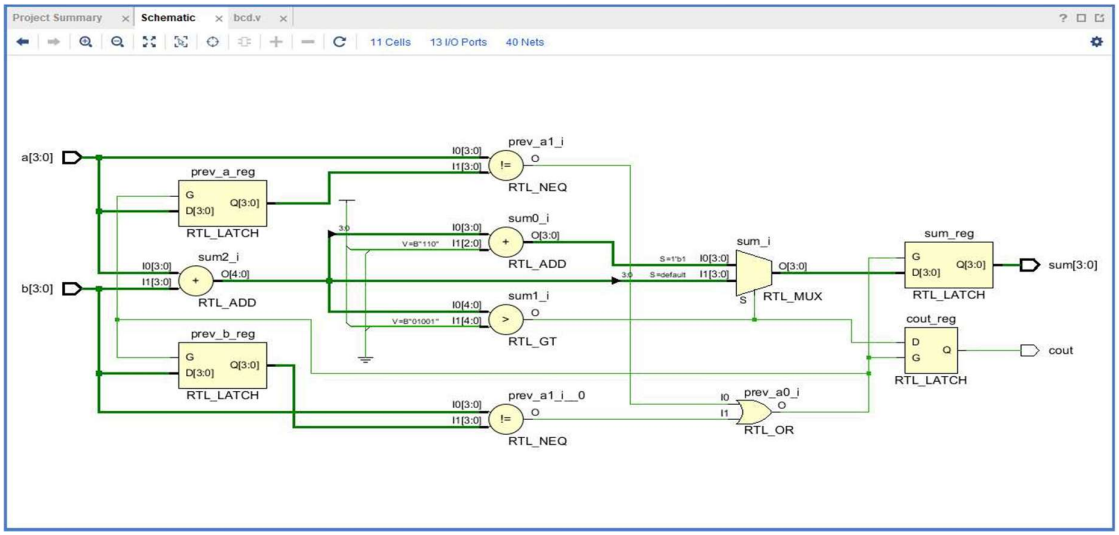


Optimized BCD Adder:

Power Report Summary:



Schematic:



Comparision Table:

(Difference in Power consumption and %Power Reduction between Unoptimized Design Vs Power Optimized BCD Adder Design)

Power Parameter	Unoptimized Design	Optimized Design	Power Reduction (W)	Percentage Reduction (%)
Total On-Chip Power	2.88 W	0.228 W	2.652 W	92.08%
Dynamic Power	2.801 W	0.157 W	2.644 W	94.39%
Static Power	0.078 W	0.071 W	0.007 W	8.97%

CHATPER 6
CONCLUSION

This report presented the implementation of a **BCD adder using different Verilog coding styles** and its deployment on the **Basys 3 FPGA board**. The comparative analysis of **structural, dataflow, and behavioral modeling** highlighted the trade-offs in terms of design complexity, resource utilization, and performance. Among the three styles, behavioral modeling offered higher abstraction and ease of implementation, while structural modeling provided better control over hardware resources.

Furthermore, **power optimization techniques** were applied to the BCD adder to enhance energy efficiency. Modifications such as **clock gating, logic simplification, and resource sharing** significantly reduced dynamic power consumption without affecting functionality. Power analysis before and after optimization confirmed that the improved design achieves **lower power consumption**, making it more suitable for low-power FPGA applications.