Assignment Three
Hardware Lab: CS224

Kavya Kumar Agrawal: 230101053
Parth Sunil Aher: 230101072
Shelke Durgesh Balkrishna: 230101093
Dhruv Pansuriya: 230101071

11th February 2025

1 Question 1: 4-bit Carry Look Ahead Adder (CLA)

1.1 Problem Statement:

Implement a ***4-bit two-input Carry Look Ahead Adder (CLA)*** in Verilog.
A carry-lookahead adder (CLA) or fast adder is a type of electronic adder used in digital logic. A carry-lookahead adder improves speed by reducing the amount of time required to determine carry bits.

1.2 Carry Look Ahead Logic:

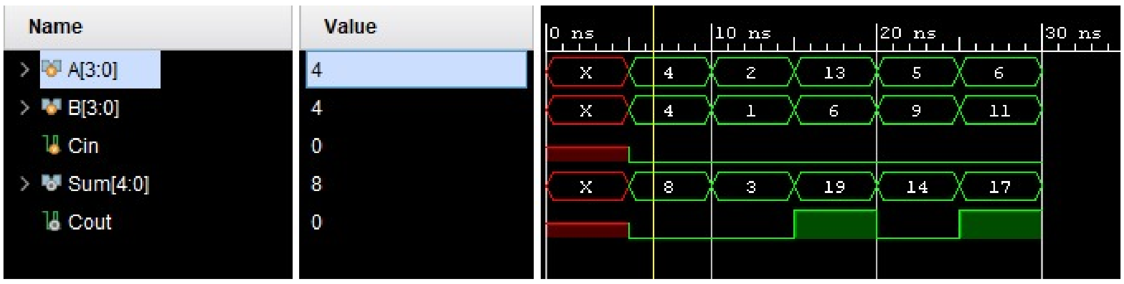
The carry generation and propagation logic are given as follows:

$$\begin{aligned}
G_i &= A_i \land B_i \\
P_i &= A_i \oplus B_i \\
C_1 &= G_0 \lor (P_0 \land C_{in}) \\
C_2 &= G_1 \lor (P_1 \land G_0) \lor (P_1 \land P_0 \land C_{in}) \\
C_3 &= G_2 \lor (P_2 \land G_1) \lor (P_2 \land P_1 \land G_0) \lor (P_2 \land P_1 \land P_0 \land C_{in}) \\
C_{out} &= G_3 \lor (P_3 \land G_2) \lor (P_3 \land P_2 \land G_1) \lor (P_3 \land P_2 \land P_1 \land G_0) \lor (P_3 \land P_2 \land P_1 \land P_0 \land C_{in})
\end{aligned}$$

1.3 Sum Calculation:

$$\begin{aligned}
S_0 &= P_0 \oplus C_{in} \\
S_1 &= P_1 \oplus C_1 \\
S_2 &= P_2 \oplus C_2 \\
S_3 &= P_3 \oplus C_3 \\
S_4 &= C_{out}
\end{aligned}$$

1.4 Simulation Results:



Name	Value
A[3:0]	4
B[3:0]	4
Cin	0
Sum[4:0]	8
Cout	0

Figure 1: 4-bit Carry Look Ahead Adder Logic Diagram

2 Question 2: 3-bit Carry Look Ahead Adder (CLA)

2.1 Problem Statement

Implement a ***3-bit two-input Carry Look Ahead Adder (CLA)*** in Verilog. **2.2 Carry Look Ahead Logic:**

Since this is a **3-bit** CLA, the carry propagation equations are similar to the **4-bit case**, with only three stages:

$$\begin{aligned}
G_i &= A_i \wedge B_i \\
P_i &= A_i \oplus B_i \\
C_1 &= G_0 \vee (P_0 \wedge C_{in}) \\
C_2 &= G_1 \vee (P_1 \wedge G_0) \vee (P_1 \wedge P_0 \wedge C_{in}) \\
C_{out} &= G_2 \vee (P_2 \wedge G_1) \vee (P_2 \wedge P_1 \wedge G_0) \vee (P_2 \wedge P_1 \wedge P_0 \wedge C_{in})
\end{aligned}$$

2.3 Calculations

The carry generation and propagation logic are given as follows:

$$\begin{aligned}
G_i &= A_i \cdot B_i \\
P_i &= A_i \oplus B_i \quad \text{(Whether previous carry affects higher bit or not)}
\end{aligned}$$

The carry propagation logic follows:

$$C_{i+1} = G_i + P_i \cdot C_i$$

For specific values:

$$\begin{aligned}
C_1 &= G_0 + P_0 \cdot C_0 \\
&= A_0 \cdot B_0 + (A_0 \oplus B_0) \cdot C_0 \\
&= A_0 \cdot B_0 \qquad ....\{as\ C_0 = 0\}
\end{aligned}$$

$$\begin{aligned}
C_2 &= A_1 \cdot B_1 + A_0 \cdot B_0 \cdot (A_1 \oplus B_1) \\
C_3 &= A_0 \cdot B_0 \cdot (A_1 \oplus B_1) \cdot (A_2 \oplus B_2) + A_2 \cdot B_2 + A_1 \cdot B_1 \cdot (A_2 \oplus B_2)
\end{aligned}$$

The Equation for Sum is:

$$S_i = A_i \oplus B_i \oplus C_i$$

For specific values:

$$\begin{aligned}
S_0 &= A_0 \oplus B_0 \oplus C_0 \\
&= A_0 \oplus B_0 \qquad ....\{as\ C_0 = 0\}
\end{aligned}$$

$$\begin{aligned}
S_1 &= A_1 \oplus B_1 \oplus C_1 \\
&= A_0 B_0 \oplus A_1 \oplus B_1
\end{aligned}$$

$$\begin{aligned}
S_2 &= A_2 \oplus B_2 \oplus C_2 \\
&= (A_1 \cdot B_1 + A_0 \cdot B_0 (A_1 \oplus B_1)) \oplus A_2 \oplus B_2
\end{aligned}$$

2.4 List of ICs used:

Quantity	Gate Type	IC Number
5	2-input XOR Gates	7486
3	2-input AND Gates	7408
3	2-input OR Gates	7432
2	3-input AND Gates	7411
1	4-input AND Gate	7421

Table 1: IC Table for Logic Gates Used

- **2.5 Circuit Diagram:**

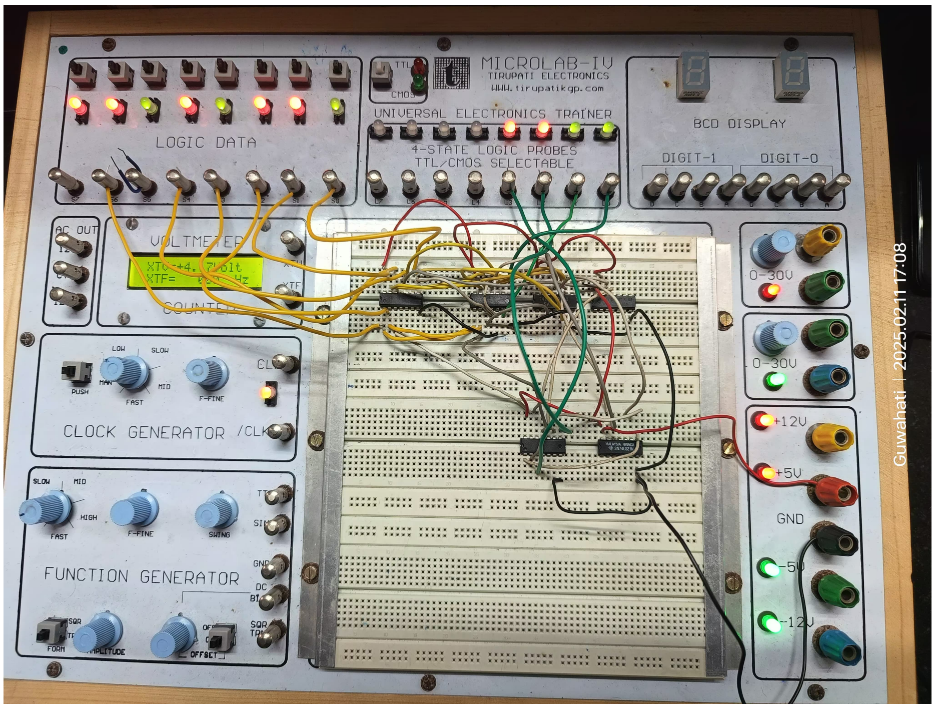


Figure 2: Implementation on Breadboard

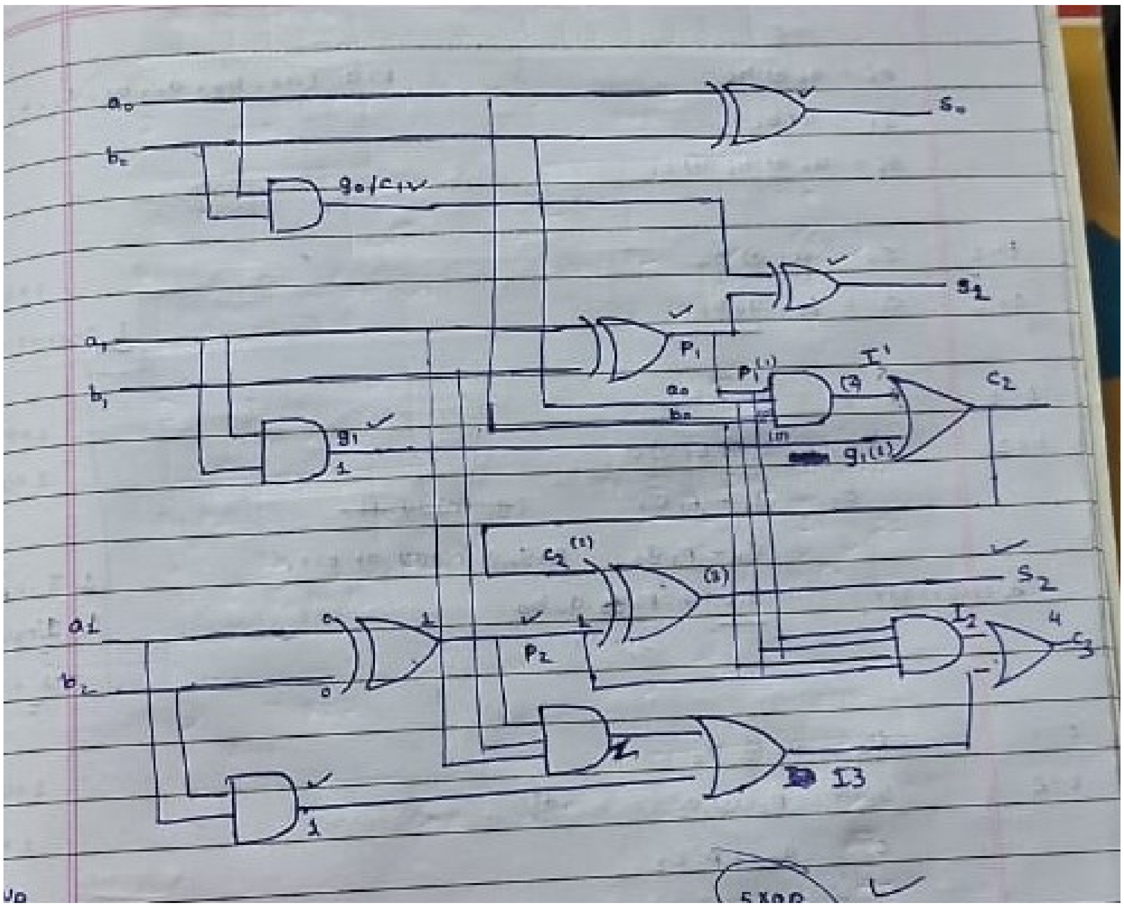


Figure 3: 3-bit Carry Look Ahead Adder Logic Diagram

3 Question 3: 32-bit CLA

3.1 Problem Statement

Implement a **32-bit three inputs Carry Save Adder (CSA)** in Verilog. You can use a ripple carry adder (RCA) for the second stage of the CSA

3.2 Block Diagrams:

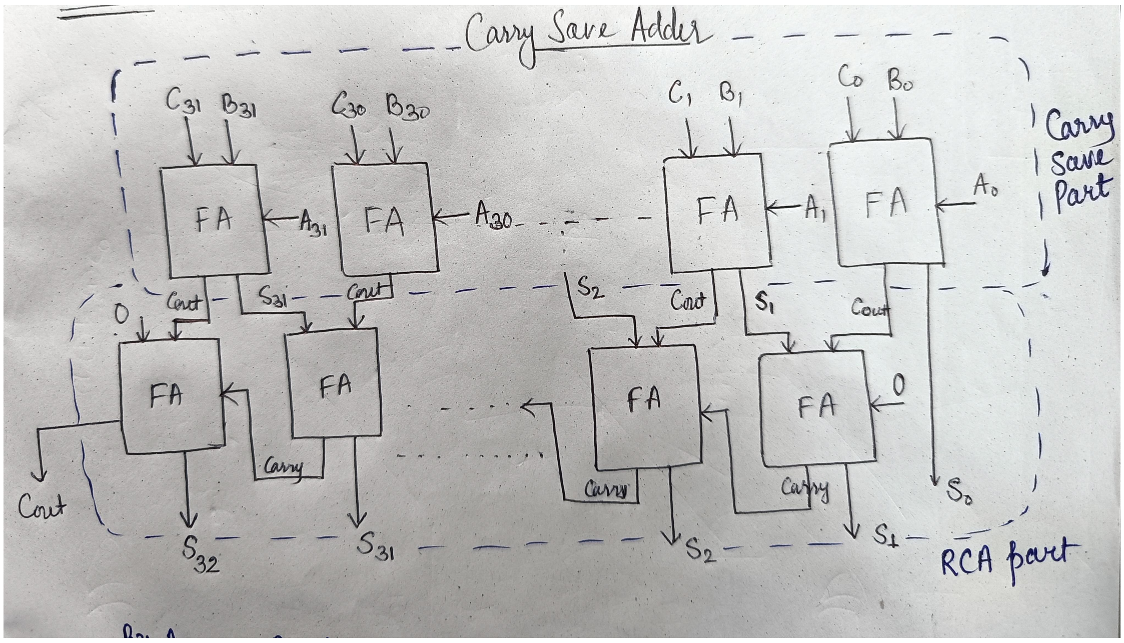


Figure 4: Block Diagram of CSA

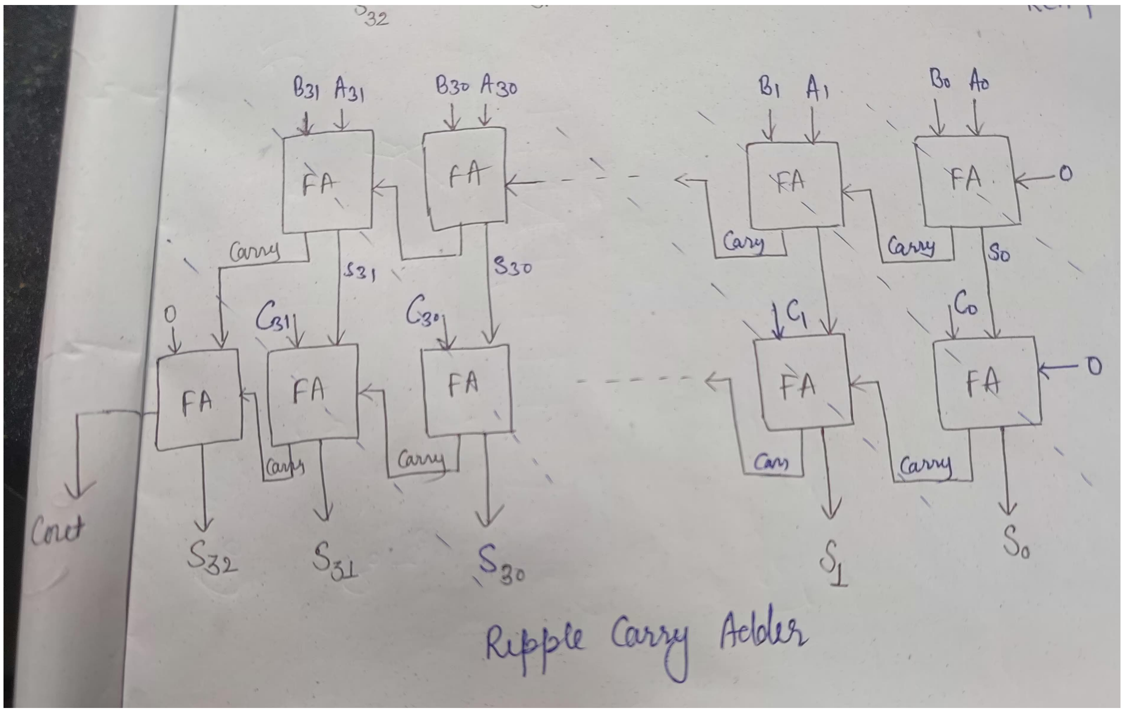
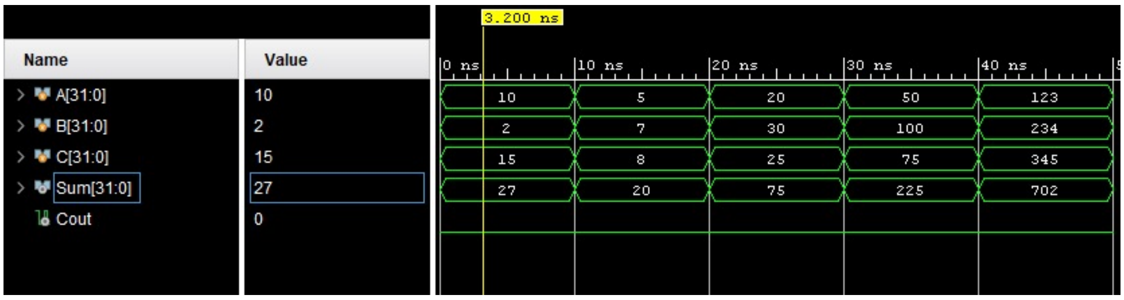


Figure 5: Block Diagram of RCA

3.3 Simulation Report



```
Name        Value   3.200 ns
                    0 ns        10 ns       20 ns       30 ns       40 ns
A[31:0]     10      10          5           20          50          123
B[31:0]     2       2           7           30          100         234
C[31:0]     15      15          8           25          75          345
Sum[31:0]   27      27          20          75          225         702
Cout        0
```

Figure 6: Delay Analysis of RCA and CSA

3.4 Comparative Analysis

Both designs are resource-light compared to the FPGA's capacity. The detailed comparison is presented below.

3.4.1 Overall Resource Utilization

- – **Slice LUTs:**
    - ∗ `rca_32_bit`: 69 LUTs (approximately 0.11% of 63,400 available).
    - ∗ `carry_save_adderr`: 77 LUTs (approximately 0.12% of 63,400 available).
  - – **Registers:** 0 registers used in both designs.
  - – **Memory and DSP:** Neither design utilizes Block RAM tiles or DSP slices.
  - – **I/O:** Both designs use 129 Bonded IOBs out of 210 available (approximately 61.43% utilization).

- **3.4.2 Delay Analysis:**

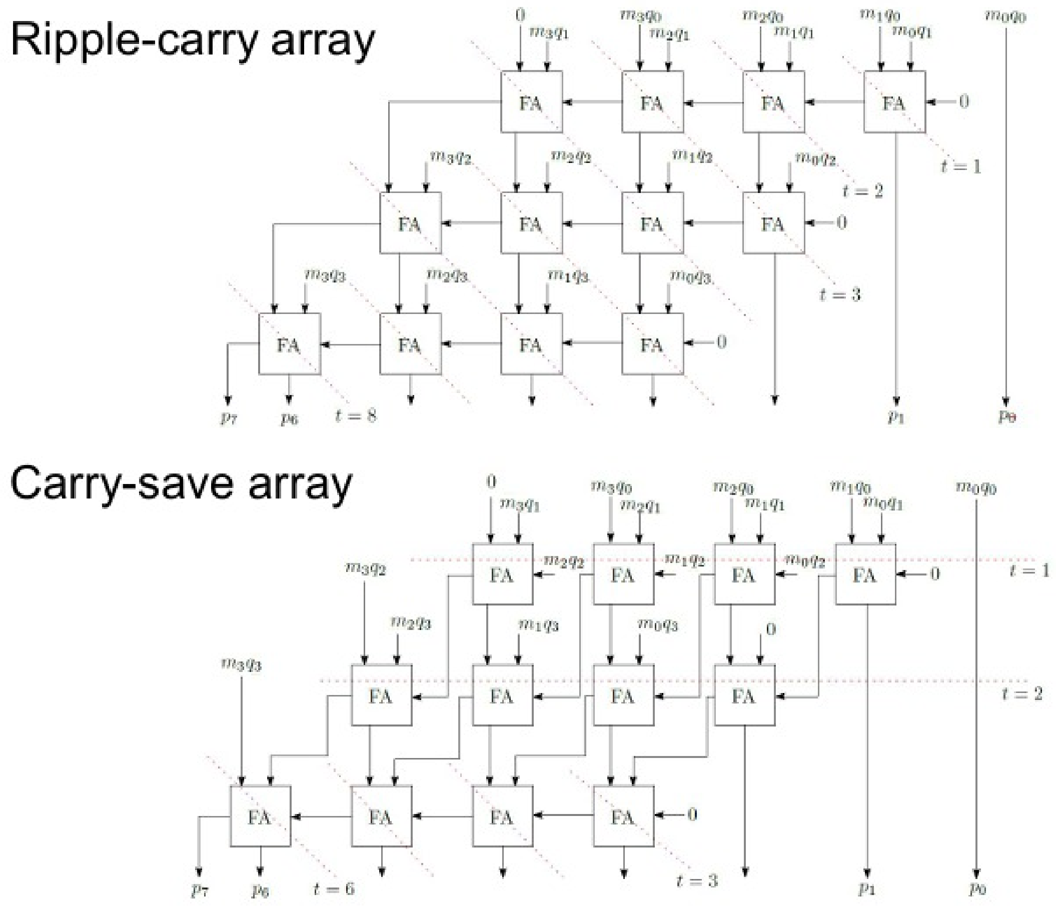


Figure 7: Delay Analysis of RCA and CSA

3.4.3 Slice Logic and LUT Usage

LUT Count and Utilization:

- - `rca_32_bit` uses 69 LUTs.
  - `carry_save_adderr` uses 77 LUTs.

Primitive Distribution:

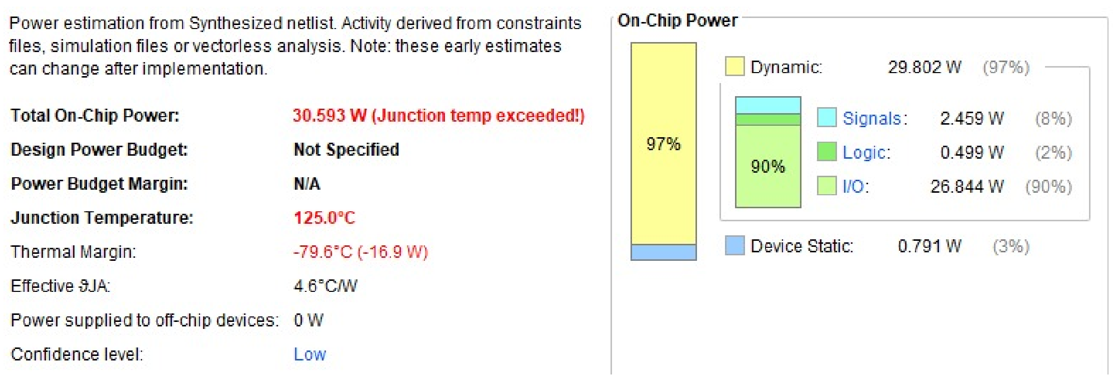
- `rca_32_bit`:
  * `LUT5`: 61 instances.
  * `LUT3`: 34 instances.
  * `LUT6`: 4 instances.
  * Additional IO primitives: 96 IBUF and 33 OBUF.
- `carry_save_adderr`:
  * `LUT3`: 46 instances.
  * `LUT6`: 31 instances.
  * `LUT5`: 16 instances.
  * Additional IO primitives: 96 IBUF and 33 OBUF.

The increased count of `LUT6` and `LUT3` in `carry_save_adderr` indicates that its logic mapping is tailored toward more complex arithmetic operations, as is common in carry-save adder architectures. In contrast, `rca_32_bit` predominantly uses `LUT5` primitives, reflecting a simpler ripple-carry adder implementation.

- ### 3.4.4 Other Resource Categories
  - **Slice Registers:** Neither design uses any registers, suggesting a fully combinatorial implementation or aggressive synthesis optimization.
  - **Memory and DSP:** No Block RAM tiles or DSP slices are used in either design.
  - **Clocking and Specific Features:** Both designs leave advanced clocking resources (e.g., BUFGCTRL, MMCME2_ADV, PLLE2_ADV) and specialized features (e.g., DNA_PORT, ICAPE2) unused.

- ### 3.4.5 Architectural Implications
  - `rca_32_bit`: Likely implements a conventional 32-bit ripple-carry adder with a straightforward logic mapping that relies heavily on `LUT5` primitives.
  - `carry_save_adderr`: Appears to implement a carry-save adder (or related arithmetic function), with a more complex logic distribution that leverages a greater number of `LUT6` and `LUT3` instances to support intricate arithmetic operations.

- ### 3.4.6 Power Analysis:



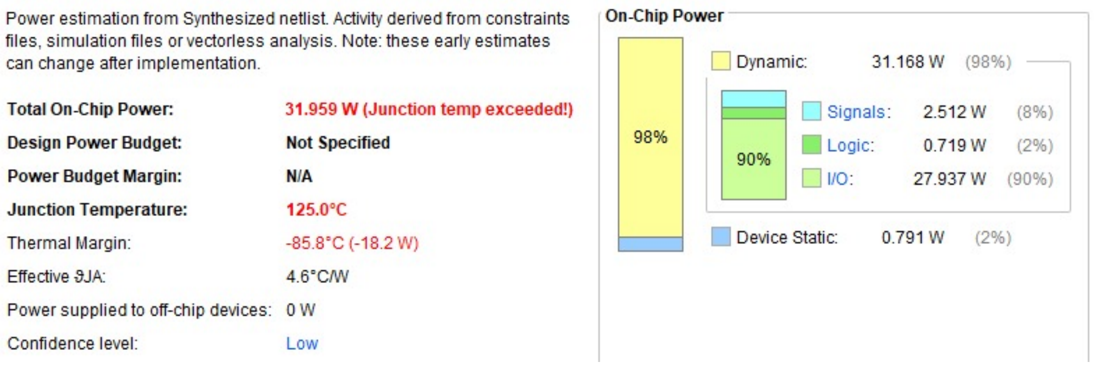
Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	30.593 W (Junction temp exceeded!)
Design Power Budget:	Not Specified
Power Budget Margin:	N/A
Junction Temperature:	125.0°C
Thermal Margin:	-79.6°C (-16.9 W)
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

On-Chip Power

🟨	Dynamic:	29.802 W	(97%)
🟦	Signals:	2.459 W	(8%)
🟩	Logic:	0.499 W	(2%)
🟨	I/O:	26.844 W	(90%)
🟦	Device Static:	0.791 W	(3%)

Figure 8: Power Analysis Report of CSA



Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**31.959 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**125.0°C**
Thermal Margin:	-85.8°C (-18.2 W)
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

On-Chip Power

Dynamic:	31.168 W	(98%)	
Signals:	2.512 W	(8%)	
Logic:	0.719 W	(2%)	
I/O:	27.937 W	(90%)	
Device Static:	0.791 W	(2%)	

Figure 9: Power Analysis Report of RCA

3.4.7 Conclusion

In summary, while both designs are highly resource-efficient, they differ in their logic mapping strategies. The `rca_32_bit` uses more area for logic gates and uses more clock-cycles while `carry_save_adderr` is an enhanced implementation design used in complex multiplier circuits taking lesser power and area.

4 Question 4: Gate Calculation

4.1 Problem Statement

Find the number of 2 inputs gates needed for a 32 bits three inputs CLA.

4.2 Calculation

In a Carry Lookahead Adder (CLA), the propagation and generation of carry signals play a crucial role in determining the sum of binary numbers efficiently. Instead of computing each carry sequentially as in a ripple carry adder, a CLA computes carry signals in parallel using logical expressions. For any given bit position i, we define the propagate signal p_i and generate signal g_i based on the input bits a_i and b_i. These signals help in the determination of the carry signals required for summation. The expressions for p_i and g_i are:

$$p_i = a_i \oplus b_i$$

$$g_i = a_i b_i$$

Here, the propagate signal p_i indicates whether a carry from a previous stage will be propagated to the next bit, while the generate signal g_i determines whether a carry will be generated at that specific bit position. Carry Computation To compute the carry signal c_i at position i, we need to consider all possible ways a carry can be generated and propagated through the previous bits. The carry expression can be written as:

$$c_i = g_{i-1}p_0p_1\cdots p_{i-1} + g_0p_1p_2\cdots p_{i-1} + g_1p_2p_3\cdots p_{i-1} + \cdots + g_{i-2}p_{i-1} + g_{i-1}$$

where the term g_{-1} is equivalent to the initial carry-in c_0. This equation essentially states that the carry at position i is generated if: 1. A carry was generated at an earlier stage and is propagated through all intermediate bits. 2. The previous bit itself generated a carry. Each product term in this summation requires $(n-1)$ AND gates with two inputs to compute the necessary conditions for carry generation. Additionally, once the product terms are obtained, an OR gate is needed to combine them and determine the final carry signal. The number of OR gates required is also $(n-1)$. Total Gate Count Derivation By summing up the number of gates required for all bit positions, we derive the total count:

$$i + \sum_{j=2}^{i+1}(j-1) = i + \frac{i(i+1)}{2} = \frac{i(i+3)}{2}$$

$$\sum_{i=1}^{n}\frac{i(i+3)}{2} = \frac{1}{6}n(n+1)(n+5)$$

Here, the first equation accounts for the number of AND gates and OR gates required for each bit position, while the second summation accumulates these counts over all n bits. Additional XOR Gates for Sum Computation To compute the final sum bits s_i, we need to consider both the propagate signal p_i and the carry signal c_i. Since the sum is determined using:

$$s_i = p_i \oplus c_i$$

each sum bit computation requires an XOR gate with two inputs. As there are n bits, we need n XOR gates. Thus, the total number of gates required is:

$$Gates = 3n + \frac{1}{6}n(n+1)(n+5)$$

Final Gate Count for a 32-bit CLA By substituting $n = 32$ into the formula:

$$Gates = 3(32) + \frac{1}{6}(32)(33)(37)$$

we find that the total number of two-input logic gates required for implementing a 32-bit Carry Lookahead Adder is:

$$2 * 6608 = 13216$$

Note: To add 3 numbers we need to use twice the number of gates

This final count represents the total logical effort required to compute sums efficiently while minimizing propagation delay compared to traditional adders.