Carry save adder

For

Submitted by:

Umatiya Bhavin

Abstract

This project designs a Carry Save Adder (CSA) in Verilog and deploys it on an FPGA (DE2 kit) for fast multi-operand addition. The design reduces carry propagation delay with Full Adders and a Ripple Carry Adder (RCA). The final sum.

Keywords

Carry Save Adder (CSA), FPGA Implementation, Verilog, Full Adder, Ripple Carry Adder (RCA), Digital Arithmetic, Multi-Operand Addition, Binary Addition, DE2 Kit, Hardware Synthesis, Low-Power Design, Parallel Computation, Carry Propagation, Logic Design, Arithmetic Logic Unit (ALU).

Contents

1. Introduction

1.1 Introduction of the Project

The Carry Save Adder (CSA) is a basic digital circuit that performs arithmetic operations quickly and is especially important in contexts where multiple numbers need to be added efficiently. Typical binary addition solutions have poor performance because carry propagation delays slow down computations. The CSA fixes this problem through parallel processing with individual full adders to calculate intermediate sums and carries simultaneously. The use of a CSA results in quicker and more efficient addition of multiple incoming values. This project will implement and deploy a Carry Save Adder using Verilog and be displayed on the FPGA (DE2 kit) to verify accuracy. The design is divided into two parts: in the first stage, input values will be changed into intermediate sums and carries in the second stage, a Ripple Carry Adder (RCA) will complete the addition and provide a fully summed value. Implementing the design in an FPGA will ensure that the design is effective and functional in real-world design and verification.

The maximum make combination is 2^6 so last decimal combination is 63.

1.2 Available Technology

1. FPGA (Field-Programmable Gate Array):

a. The CSA is implemented on an Altera DE2 FPGA Kit using Cyclone II as the target device.

b. FPGA allows for parallel processing and high-speed computation, making it ideal for arithmetic operations like CSA.

2. Verilog Hardware Description Language (HDL)

   a. The CSA and Ripple Carry Adder (RCA) are coded in Verilog, enabling efficient hardware-level implementation.

   b. The modular approach ensures reusability and scalability for future applications.

3. Quartus II Development Software:

   a. Used for design entry, synthesis, simulation, and FPGA programming**.**

   b. Provides a RTL view to verify the correct implementation of logic circuits.

4. Simulation and Testing Tools:

   a. ModelSim or Quartus II Simulator is used for functional verification before deployment.

   b. Allows waveform analysis of sum and carry outputs to ensure correctness.

5. Logic Optimization Techniques:

   a. CSA reduces carry propagation delay, improving efficiency.

   b. Future enhancements may include Carry Lookahead Adder (CLA) integration to further optimize performance.

2. Limitations

1. Final Addition Requirement – The CSA alone does not produce the final sum. Instead, it generates intermediate sum and carry values, which require an additional adder, such as a Ripple Carry Adder (RCA), to compute the final result.

2. Propagation Delay in Final Stage – Although the CSA minimizes carry propagation in the initial stages, the final addition using an RCA introduces some delay, making the overall speed dependent on the final adder's efficiency.

3. Hardware Complexity – Implementing CSA on FPGA consumes more logic elements due to multiple full adders, requiring careful resource management.

4. Scalability Constraints – For very large bit-width additions, integrating CSA with an optimized final-stage adder (e.g., Carry Lookahead Adder) is necessary to maintain high performance.

5. FPGA Resource Utilization – While efficient, the CSA's multi-bit computation can lead to increased FPGA area usage, which may be a constraint in resource-limited designs.

3.Proposed Solution & Methodology

1. Optimized Final Addition Stage**:** Instead of using a Ripple Carry Adder (RCA), integrating a Carry Lookahead Adder (CLA) can significantly reduce the final sum computation delay.
2. Parallel Processing with Pipelining: Implementing pipelining can allow multiple addition operations to be performed simultaneously, increasing throughput.
3. Resource Optimization on FPGA: Reducing logic gate usage by modifying the design architecture to balance speed and resource consumption.
4. Higher Bit-width Support: Extending the design to support higher bit-width operands for broader computational applications such as DSP and cryptography.

3.1 Methodology

3.1.1 Carry Save Adder Design

The CSA consists of multiple Full Adders (FA) operating in parallel. It takes three inputs (A, B, and C) and produces an intermediate sum and carry without immediately propagating the carry.

3.1.2 Two-Stage works

a. First Stage: A set of Full Adders computes the sum bits (S) and carry bits (C_out) for each bit position.
b. Second Stage: The carry bits are left-shifted and added to the sum bits using a Ripple Carry Adder (RCA) to generate the final sum.

4. Block Diagram

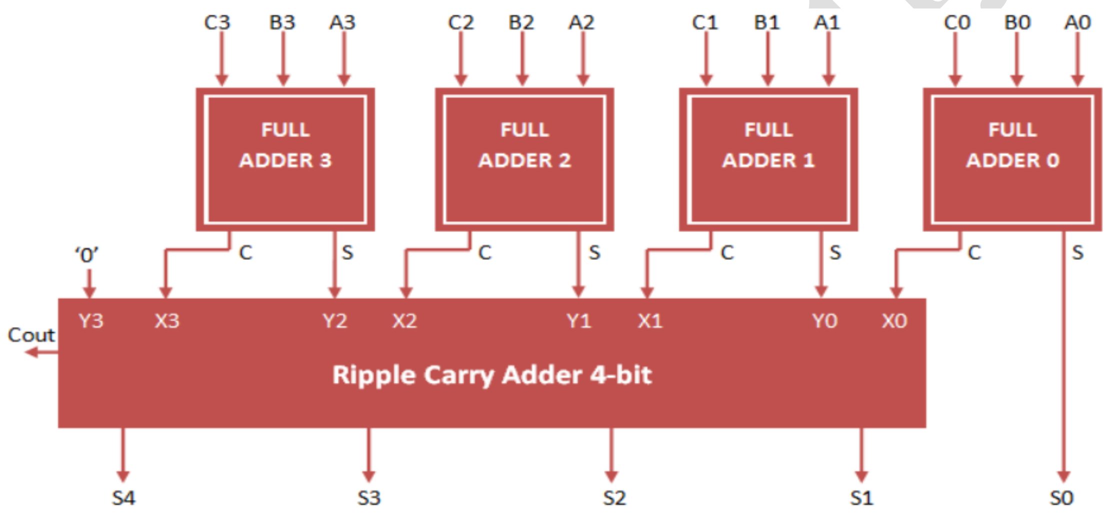
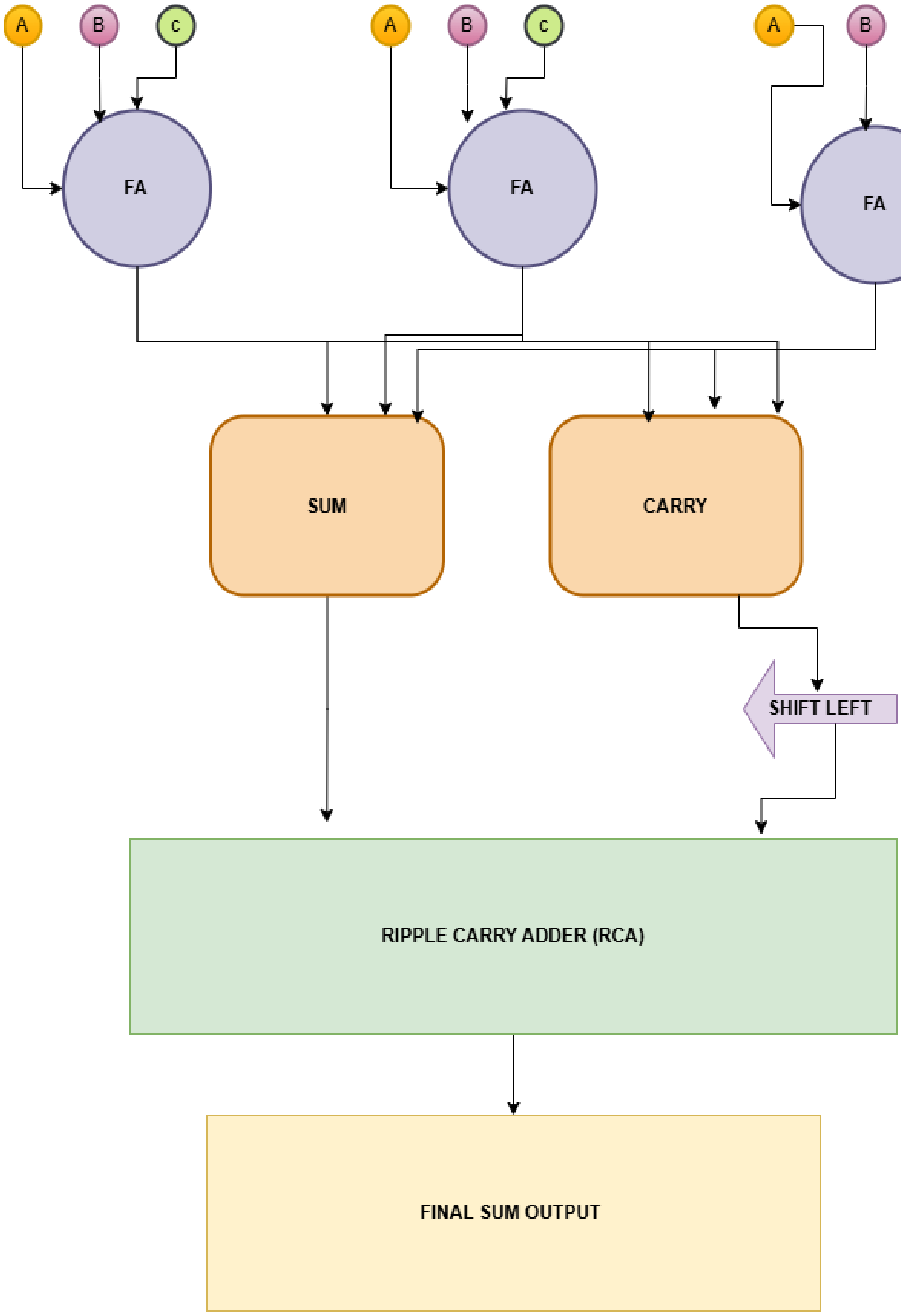


Figure1Block diagram of the system

4.1 Block Diagram

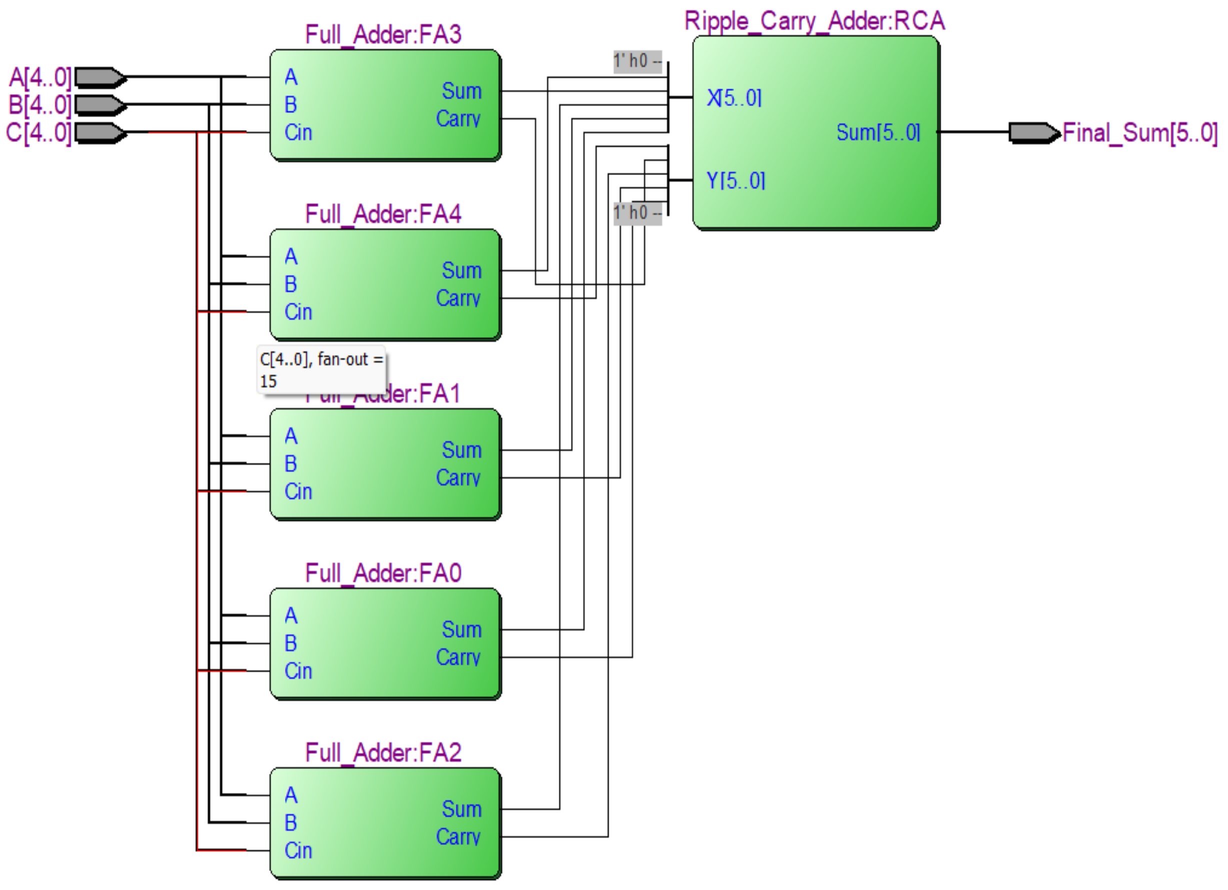
1. Input Stage: Three input values (A, B, C) are fed into the CSA.
2. First Stage - Full Adders (FAs): The input bits are passed through multiple Full Adders (FAs)
3. Second Stage - Carry Processing: The carry bits are shifted left and processed in an additional layer of Full Adders to form the final carry.
4. Final Addition - Ripple Carry Adder (RCA): This final addition produces the final sum output.

5. Flow chart

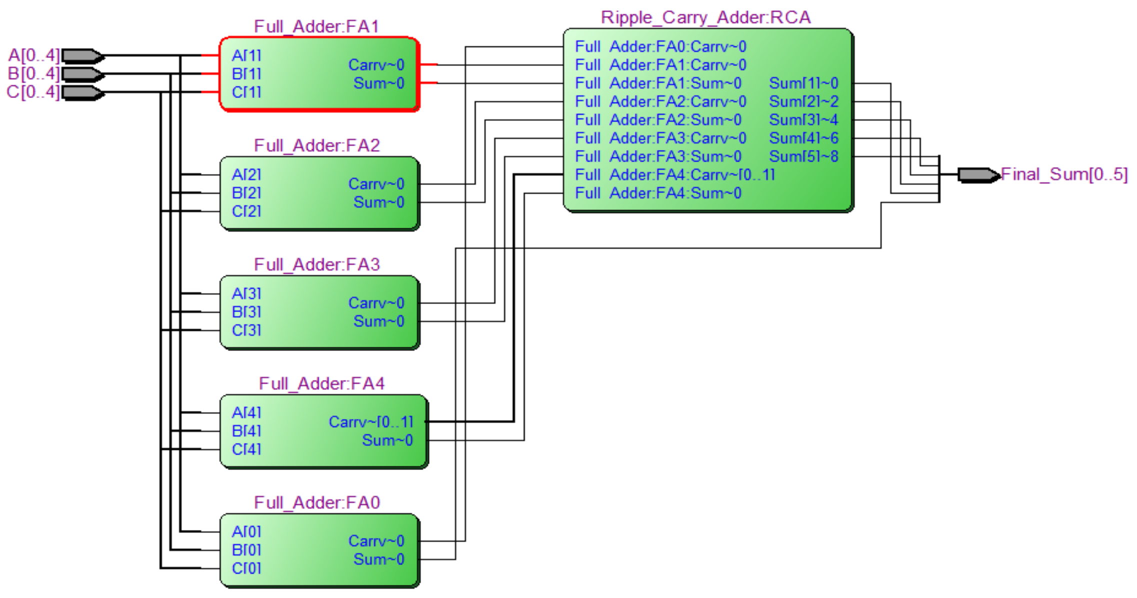


Flow chart of the code

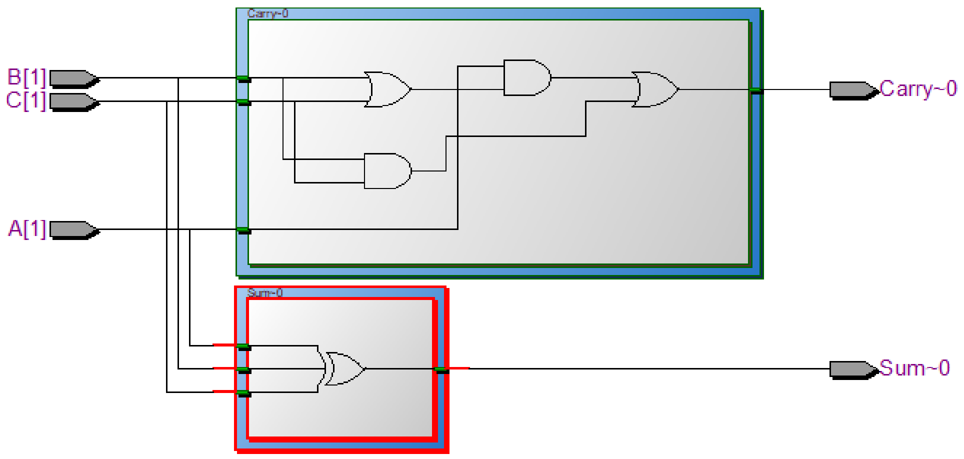
6. RTL VIEW



7. TTL VIEW



Each full adder show this:



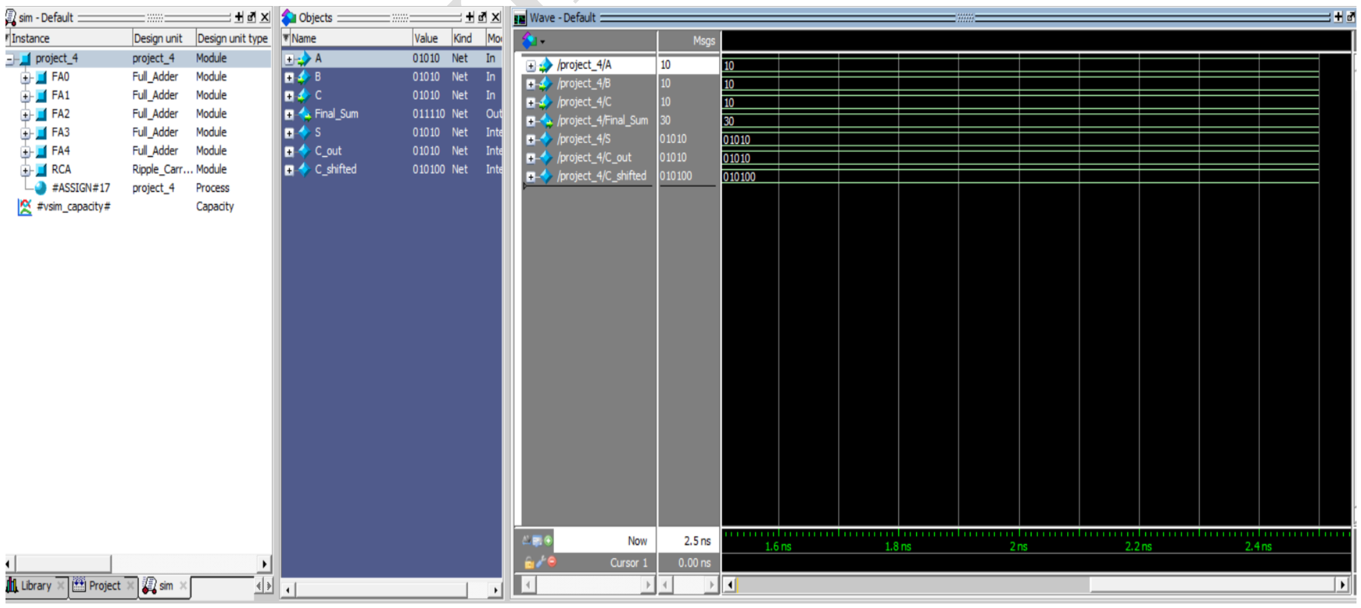
8. Wave forms

When I input a= 10

b=10

c=10

final sum is = 30


sim - Default

Instance	Design unit	Design unit type
project_4	project_4	Module
FA0	Full_Adder	Module
FA1	Full_Adder	Module
FA2	Full_Adder	Module
FA3	Full_Adder	Module
FA4	Full_Adder	Module
RCA	Ripple_Carr...	Module
#ASSIGN#17	project_4	Process
#vsim_capacity#		Capacity

Objects

Name	Value	Kind	Mo
A	01010	Net	In
B	01010	Net	In
C	01010	Net	In
Final_Sum	011110	Net	Out
S	01010	Net	Inte
C_out	01010	Net	Inte
C_shifted	010100	Net	Inte

Wave - Default

	Msgs	
/project_4/A	10	10
/project_4/B	10	10
/project_4/C	10	10
/project_4/Final_Sum	30	30
/project_4/S	01010	01010
/project_4/C_out	01010	01010
/project_4/C_shifted	010100	010100

Now 2.5 ns
Cursor 1 0.00 ns

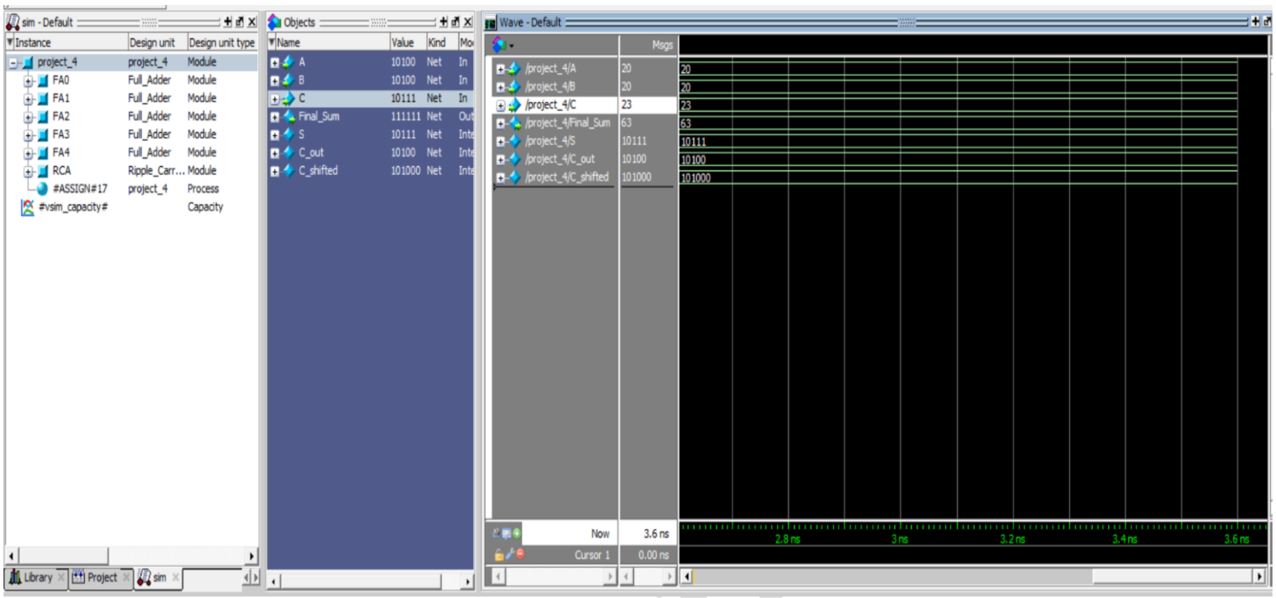
When I input a= 20

b=20

c=23

final sum is = 63

The maximum make combination is 2^6 so last decimal combination is 63.



sim - Default

Instance	Design unit	Design unit type
project_4	project_4	Module
FA0	Full_Adder	Module
FA1	Full_Adder	Module
FA2	Full_Adder	Module
FA3	Full_Adder	Module
FA4	Full_Adder	Module
RCA	Ripple_Carr...	Module
#ASSIGN#17	project_4	Process
#vsim_capacity#		Capacity

Objects

Name	Value	Kind	Mo
A	10100	Net	In
B	10100	Net	In
C	10111	Net	In
Final_Sum	111111	Net	Out
S	10111	Net	Inte
C_out	10100	Net	Inte
C_shifted	101000	Net	Inte

Wave - Default

	Msgs	
/project_4/A	20	20
/project_4/B	20	20
/project_4/C	23	23
/project_4/Final_Sum	63	63
/project_4/S	10111	10111
/project_4/C_out	10100	10100
/project_4/C_shifted	101000	101000

Now 3.6 ns
Cursor 1 0.00 ns

9. Conclusion

The use of a Carry Save Adder (CSA) and a Ripple Carry Adder (RCA) effectively accelerates multi-operand addition by minimizing propagation delay in the first computation phase. The CSA adds in parallel with high efficiency

10.References

[1] A. Kumar, *Fundamentals of Logic Design*. New Delhi, India: Prentice-Hall, 2005.

[2] M. Mano and M. Ciletti, *Digital Design with an Introduction to the Verilog HDL*, 5th ed. Boston, MA, USA: Pearson, 2013.

[3] Altera Corporation, *DE2 Development and Education Board User Manual*, [Online].

Available: https://www.altera.com.

[4] FPGA-based Arithmetic Circuits Research Papers

11. Verilog code

```
module csa (
   input [4:0] A, B, C,
             output [5:0] Final_Sum // 6-bit final sum to accommodate carry
);
   wire [4:0] S;      // Sum from first stage
   wire [4:0] C_out;   // Carry outputs from first stage

   // First stage: Carry Save Adders (Full Adders)
   Full_Adder FA0 (A[0], B[0], C[0], S[0], C_out[0]);
   Full_Adder FA1 (A[1], B[1], C[1], S[1], C_out[1]);
   Full_Adder FA2 (A[2], B[2], C[2], S[2], C_out[2]);
   Full_Adder FA3 (A[3], B[3], C[3], S[3], C_out[3]);
             Full_Adder FA4 (A[4], B[4], C[4], S[4], C_out[4]);

   // Second stage: Use a Ripple Carry Adder to add S and Carry
   wire [5:0] C_shifted;
   assign C_shifted = {C_out, 1'b0}; // Shift carry left

   // Final stage: Ripple Carry Adder to final sum
   Ripple_Carry_Adder RCA (.X({1'b0, S}), .Y(C_shifted), .Sum(Final_Sum));

endmodule

// Full Adder Module (Used in Carry Save Adder)
module Full_Adder (
   input A, B, Cin,
   output Sum, Carry
);
   assign Sum = A ^ B ^ Cin;
   assign Carry = (A & B) | (B & Cin) | (Cin & A);
endmodule
// Ripple Carry Adder for final stage
module Ripple_Carry_Adder (
   input [5:0] X, Y,
   output [5:0] Sum
);
   assign Sum = X + Y; // Simple addition
endmodule
```