Project 5: 16 bit Carry Save Adder

A Comprehensive Study of Advanced Digital Circuits

By: Abhishek Sharma , Gati Goyal ,Nikunj Agrawal , Ayush Jain

Contents

Project 5: 16 bit Carry Save Adder

A Comprehensive Study of Advanced Digital Circuits

By: Abhishek Sharma , Gati Goyal ,Nikunj Agrawal , Ayush Jain

1 Introduction

A Carry Save Adder (CSA) is an efficient digital circuit used to add multiple binary numbers simultaneously. Unlike a conventional adder which propagates carry bits immediately, a CSA delays the carry propagation to optimize speed, particularly useful in multi-operand addition operations.

2 Working Principle

The CSA adds three input numbers at a time and produces two outputs: a sum and a carry. These outputs are stored and then used in subsequent stages. The final stage typically involves a conventional adder to combine the intermediate results and produce the final sum.

3 Components of CSA

- **Partial Sum (S):** The sum of the individual bits without considering the carry from the previous bits.
- **Carry Out (C):** The carry resulting from the addition of individual bits.

Given three binary numbers A, B, and C, the CSA computes:

$$S_i = A_i \oplus B_i \oplus C_i \tag{1}$$

$$C_i = (A_i \wedge B_i) \vee (B_i \wedge C_i) \vee (C_i \wedge A_i) \tag{2}$$

where $\oplus$ denotes the XOR operation, and $\wedge$ denotes the AND operation.

The sum bits S_i and carry bits C_i are stored and passed to the next stage where they are added to another operand. This process continues until all operands are added.

4 Example: Adding Four Binary Numbers

Let's consider adding four binary numbers: A, B, C, and D.

1. **First Stage:** Add A, B, and C using a CSA.
2. **Second Stage:** Add the sum and carry from the first stage to D using another CSA.
3. **Final Stage:** Use a conventional adder to add the final sum and carry outputs from the second stage.

5 RTL Code

Listing 1: 16 bit Carry Save Adder RTL Code

```
module csa16a (
    output logic [15:0] Sum,
    output logic Cout,
    input  logic [15:0] A,
    input  logic [15:0] B,
    input  logic Cin
);

    logic c4, c7, c12;
    logic [3:0] p1, p2;
    logic w1, w2, w3, w4;

    // Instantiate 4-bit ripple carry adders
    rca4 cpa1 (.Sum(Sum[3:0]),   .Cout(c4),   .A(A[3:0]),
        .B(B[3:0]),   .Cin(Cin));
    rca4p cpa2 (.Sum(Sum[7:4]),  .Cout(w1),   .P(p1),
        .A(A[7:4]),   .B(B[7:4]),   .Cin(c4));
    rca4p cpa3 (.Sum(Sum[11:8]), .Cout(w3),   .P(p2),
        .A(A[11:8]),  .B(B[11:8]),  .Cin(c7));
    rca4 cpa4 (.Sum(Sum[15:12]), .Cout(Cout), .A(A[15:12]),
        .B(B[15:12]), .Cin(c12));

    // Carry skip logic
    and a1 (w2, p1[0], p1[1], p1[2], p1[3], c4);
    or  o1 (c7, w1, w2);
    and a2 (w4, p2[0], p2[1], p2[2], p2[3], c7);
    or  o2 (c12, w3, w4);

endmodule

// 4-bit Ripple Carry Adder without Propagate Logic
module rca4 (
    output logic [3:0] Sum,
    output logic Cout,
    input  logic [3:0] A,
    input  logic [3:0] B,
    input  logic Cin
);

    logic [3:0] C;

    // Full Adders for each bit
    FullAdder fa0 (.A(A[0]), .B(B[0]), .Cin(Cin),  .Sum(Sum[0]),
        .Cout(C[0]));
    FullAdder fa1 (.A(A[1]), .B(B[1]), .Cin(C[0]), .Sum(Sum[1]),
        .Cout(C[1]));
    FullAdder fa2 (.A(A[2]), .B(B[2]), .Cin(C[1]), .Sum(Sum[2]),
        .Cout(C[2]));
    FullAdder fa3 (.A(A[3]), .B(B[3]), .Cin(C[2]), .Sum(Sum[3]),
        .Cout(Cout));

endmodule

// 4-bit Ripple Carry Adder with Propagate Logic
```

```
module rca4p (
    output logic [3:0] Sum,
    output logic Cout,
    output logic [3:0] P,
    input  logic [3:0] A,
    input  logic [3:0] B,
    input  logic Cin
);

    logic [3:0] C;

    // Full Adders for each bit
    FullAdder fa0 (.A(A[0]), .B(B[0]), .Cin(Cin),  .Sum(Sum[0]), .Cout(C[0]));
    FullAdder fa1 (.A(A[1]), .B(B[1]), .Cin(C[0]), .Sum(Sum[1]), .Cout(C[1]));
    FullAdder fa2 (.A(A[2]), .B(B[2]), .Cin(C[1]), .Sum(Sum[2]), .Cout(C[2]));
    FullAdder fa3 (.A(A[3]), .B(B[3]), .Cin(C[2]), .Sum(Sum[3]), .Cout(Cout));

    // Propagate logic
    assign P = A ^ B;

endmodule

// Full Adder Module
module FullAdder (
    input  logic A,
    input  logic B,
    input  logic Cin,
    output logic Sum,
    output logic Cout
);

    // Full adder logic
    assign Sum = A ^ B ^ Cin;
    assign Cout = (A & B)  (Cin & (A ^ B));

endmodule
```

6 Testbench

Listing 2: Carry Save Adder Testbench

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Abhishek Sharma
//
// Create Date: 18.08.2024 14:34:31
// Design Name:
// Module Name: csa16tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////


module csa16tb;

    // Declare inputs as reg and outputs as wire
   logic  [15:0] A;
    logic  [15:0] B;
    logic   Cin;
   logic  [15:0] Sum;
    logic  Cout;

    // Instantiate the CarrySkipAdder_16bit module
    csa16a gg(
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout)
    );

    // Testbench procedure
    initial begin
        // Test Case 1: Add zero to zero
        A = 16'b0000000000000000;
        B = 16'b0000000000000000;
        Cin = 1'b0;
        #10;
        $display("TC1: A = %b, B = %b, Cin = %b | Sum = %b, Cout = %b", A, B, Cin, Sum, Cout);

        // Test Case 2: Add two small numbers
        A = 16'b0000000000000011;  // 3
        B = 16'b0000000000000101;  // 5
        Cin = 1'b0;
        #10;
        $display("TC2: A = %b, B = %b, Cin = %b | Sum = %b, Cout = %b", A, B, Cin, Sum, Cout);

        // Test Case 3: Add two numbers with carry in
        A = 16'b0000111100001101;  // 15 and 13
        B = 16'b0000011000000111;  // 6 and 7
        Cin = 1'b1;
        #10;
        $display("TC3: A = %b, B = %b, Cin = %b | Sum = %b, Cout = %b", A, B, Cin, Sum, Cout);

        // Test Case 4: Add random values
        A = 16'b1010101010101010;  // Some random value
        B = 16'b0101010101010101;  // Another random value
        Cin = 1'b0;
```

```
        #10;
        $display("TC4: A = %b, B = %b, Cin = %b | Sum = %b, Cout =
            %b", A, B, Cin, Sum, Cout);

        // Test Case 5: Overflow case
        A = 16'b1111111111111111;  // 65535
        B = 16'b1111111111111111;  // 65535
        Cin = 1'b0;
        #10;
        $display("TC5: A = %b, B = %b, Cin = %b | Sum = %b, Cout =
            %b", A, B, Cin, Sum, Cout);

        // Finish simulation
        $finish;
    end

endmodule
```

7 Simulation Results

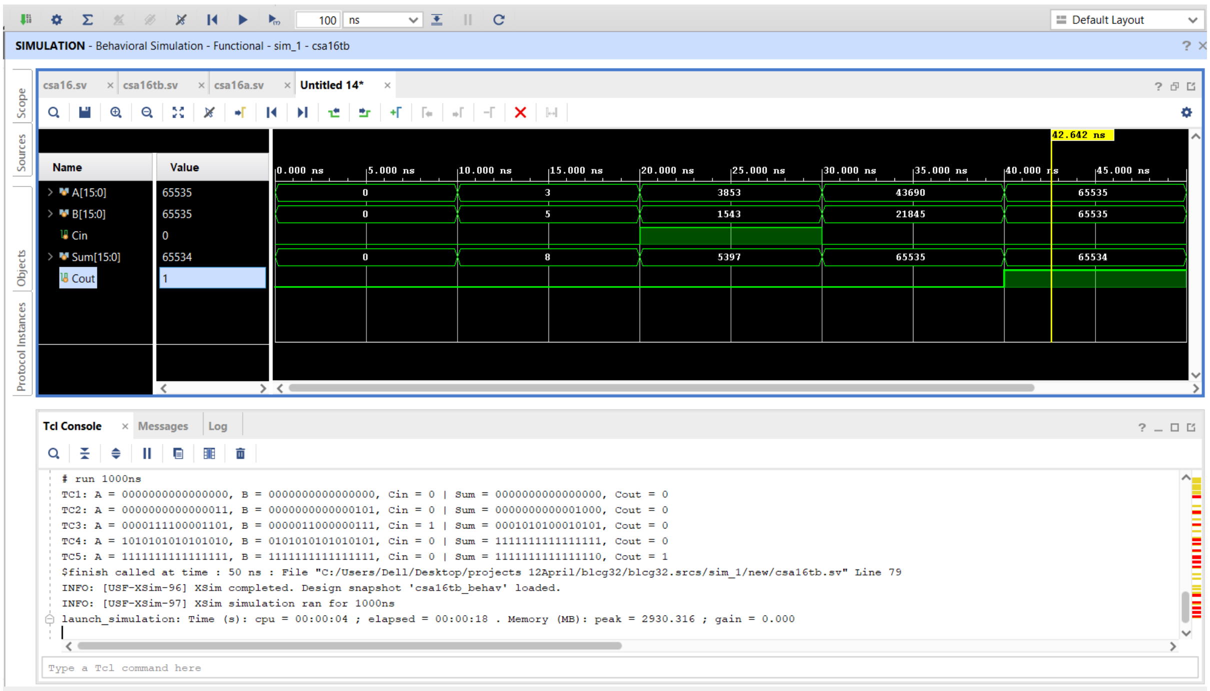

Figure 1: Simulation results of CSA

Simulation Results Explanation

In our simulation, we observed the results of adding two 16-bit binary numbers. Specifically, we tested the case where both inputs are set to their maximum value:

$$A = 16'b1111111111111111$$
$$B = 16'b1111111111111111$$
$$\text{Cin} = 1'b0$$

Here, both A and B are equal to 65535 in decimal.

Addition Process

Performing binary addition of A and B yields:

$$\begin{array}{r} 1111111111111111 \\ +1111111111111111 \\ \hline 1\ 1111111111111110 \end{array}$$

The result of this addition is 131070 in decimal, which can be represented as 1 1111111111111110 in binary.

16-bit Limitation

Since we are using a 16-bit adder, only the lower 16 bits of the result are stored. Thus:

$$\text{Lower 16 bits: } 1111111111111110$$

This binary value corresponds to 65534 in decimal.

Carry-Out

The leading bit ('1') in the binary result '1 1111111111111110' is the carry-out from the 16-bit boundary, indicating an overflow. The carry-out value is:

$$\text{Carry-out (Cout)} = 1$$

Summary

8 Carry Skip Adder (CSKA) - `cska16` Module

8.1 Description

The `cska16` module implements a 16-bit Carry Skip Adder (CSKA). This adder efficiently adds two 16-bit numbers by skipping carry propagation in certain sections, which speeds up the computation.

8.2 Inputs and Outputs

- **Inputs:**
  - `A[15:0]`: 16-bit input operand A.
  - `B[15:0]`: 16-bit input operand B.
  - `Cin`: Carry-in bit.
- **Outputs:**
  - `Sum[15:0]`: 16-bit result of the addition.
  - `Cout`: Carry-out bit.

8.3 Internal Logic

- **Instantiation of Adders:**
  - `rca4` and `rca4p` modules handle different 4-bit sections of the operands.
  - `rca4` is used for the least significant and most significant 4-bit segments.
  - `rca4p` handles intermediate 4-bit segments and includes propagate logic.
- **Carry Skip Logic:**
  - Intermediate signals `w2` and `w4` are used to determine if carry propagation should be skipped.
  - `w2` and `w4` are computed as follows:

$$w2 = p1[0] \wedge p1[1] \wedge p1[2] \wedge p1[3] \wedge c4$$

$$w4 = p2[0] \wedge p2[1] \wedge p2[2] \wedge p2[3] \wedge c7$$

  - Carry signals `c7` and `c12` are calculated using:

$$c7 = w1 \vee w2$$

$$c12 = w3 \vee w4$$

9 4-bit Ripple Carry Adder (RCA4) - `rca4` Module

9.1 Description

The `rca4` module is a 4-bit Ripple Carry Adder (RCA4) that performs addition of two 4-bit numbers with a carry-in bit.

9.2 Inputs and Outputs

- **Inputs:**
  - `A[3:0]`: 4-bit input operand A.
  - `B[3:0]`: 4-bit input operand B.
  - `Cin`: Carry-in bit.
- **Outputs:**
  - `Sum[3:0]`: 4-bit result of the addition.
  - `Cout`: Carry-out bit.

9.3 Internal Logic

- **Full Adders:**
  - Four instances of the `FullAdder` module are used to perform the addition.
  - Each `FullAdder` computes the sum for one bit and propagates the carry to the next bit.

10 4-bit Ripple Carry Adder with Propagate Logic (RCA4P) - `rca4p` Module

10.1 Description

The `rca4p` module is similar to the `rca4` module but includes additional propagate logic to support the carry skip feature of the CSKA.

10.2 Inputs and Outputs

- **Inputs:**
  - `A[3:0]`: 4-bit input operand A.
  - `B[3:0]`: 4-bit input operand B.
  - `Cin`: Carry-in bit.
- **Outputs:**
  - `Sum[3:0]`: 4-bit result of the addition.
  - `Cout`: Carry-out bit.
  - `P[3:0]`: Propagate signals for each bit.

10.3 Internal Logic

- **Full Adders:**
  - Similar to `rca4`, uses four `FullAdder` modules.
- **Propagate Logic:**
  - Computes propagate signals `P`:

$$P[i] = A[i] \oplus B[i]$$

11 Full Adder - `FullAdder` Module

11.1 Description

The `FullAdder` module performs the basic full addition operation used in both `rca4` and `rca4p` modules.

11.2 Inputs and Outputs

- **Inputs:**
  - `A`: Single-bit input operand A.
  - `B`: Single-bit input operand B.
  - `Cin`: Carry-in bit.
- **Outputs:**
  - `Sum`: Single-bit sum result.
  - `Cout`: Carry-out bit.

11.3 Internal Logic

- **Sum Calculation:**
  - Computes the sum using XOR gates:

$$\text{Sum} = A \oplus B \oplus Cin$$

- **Carry-Out Calculation:**
  - Computes the carry-out using AND and OR gates:

$$\text{Cout} = (A \wedge B) \vee (Cin \wedge (A \oplus B))$$

The simulation results are:

$$\text{Sum} = 1111111111111110 \text{ (65534 in decimal)}$$
$$\text{Carry-out} = 1$$

This behavior is consistent with the expected operation of a 16-bit adder. The sum shows the lower 16 bits of the addition result, while the carry-out signifies that the result exceeded the 16-bit limit.

12 Schematic

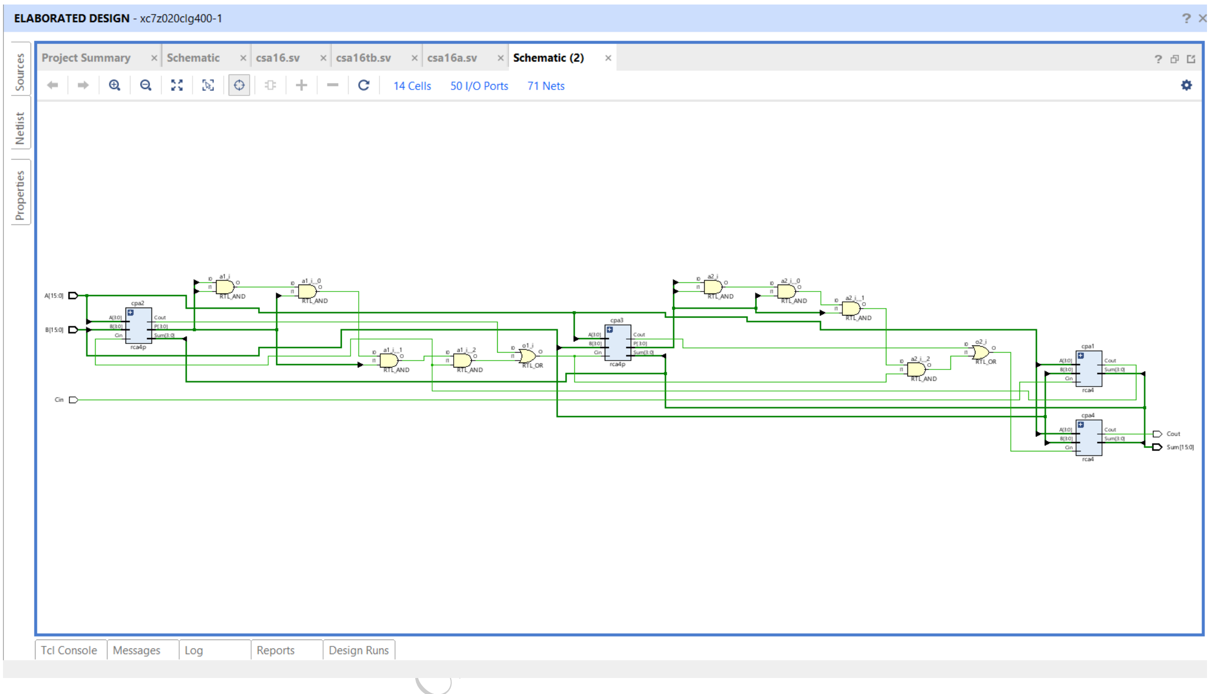

Figure 2: Schematic of RCA

13 Synthesis Design

14 Advantages

- **Speed:** CSAs significantly reduce the propagation delay since the carry is not propagated through every bit. Instead, the carry is stored and handled in the next stages.
- **Scalability:** CSAs can efficiently handle the addition of multiple operands by cascading multiple stages.

15 Applications

- **Multiplication:** CSAs are commonly used in multipliers, especially in array multipliers and Wallace tree multipliers, where multiple partial products need to be added.
- **Digital Signal Processing (DSP):** CSAs are used in DSP applications requiring fast arithmetic operations.

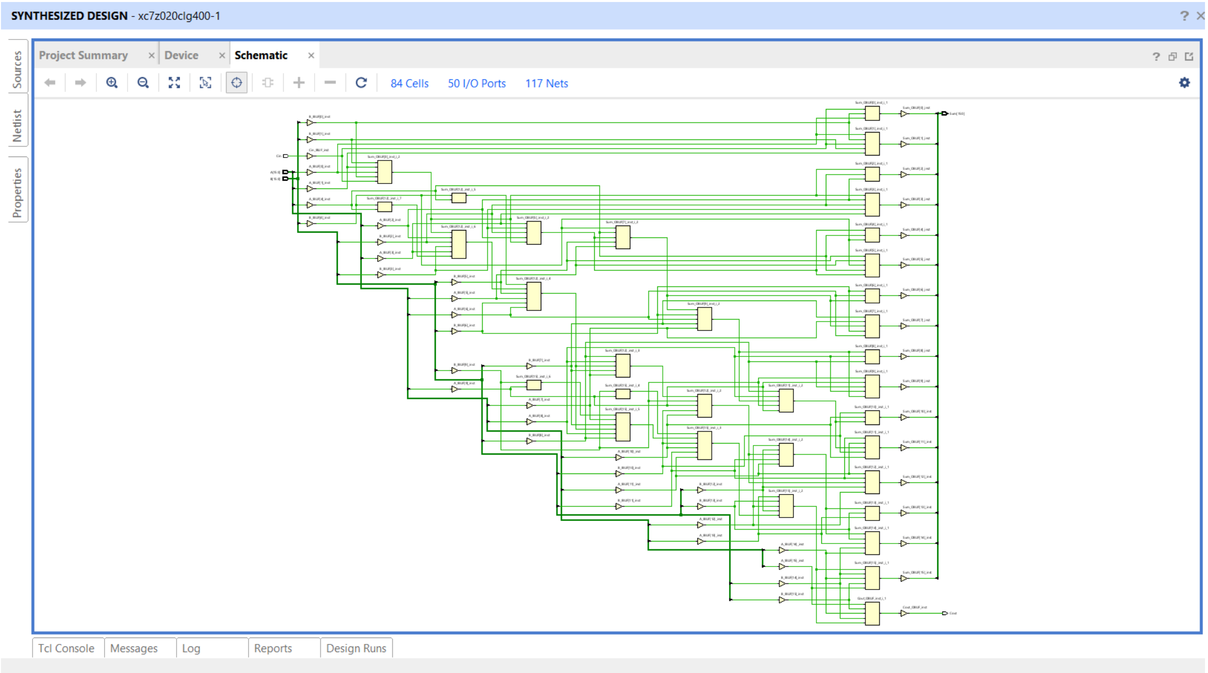

Figure 3: Synthesis Design of 16 Bit Carry Save Adder

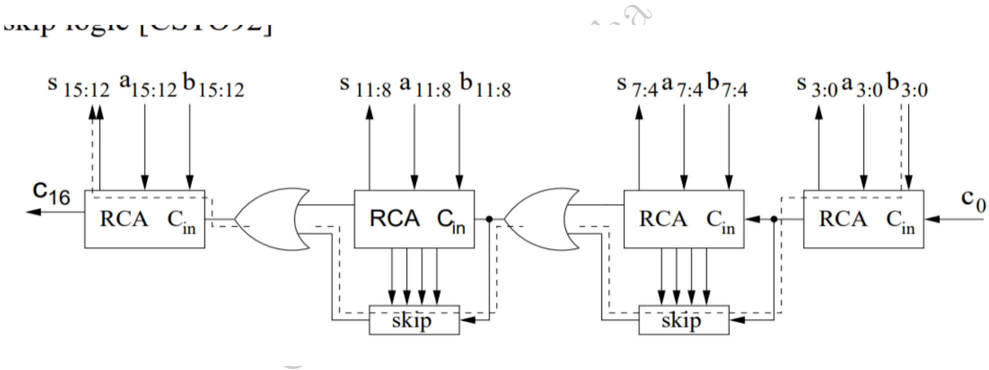

Figure 4: csa block diagram

16 Example Circuit

Here's an example circuit for a single stage of a CSA:

```
A ---------->|          |---------> S
B ---------->|  CSA     |---------> Carry
C ---------->|          |--------->
```

This process is repeated for each bit position, and the carry and sum outputs are stored and used in subsequent stages.

17 Conclusion

In summary, a Carry Save Adder (CSA) is a powerful tool for performing fast, multi-operand addition, playing a crucial role in high-speed arithmetic operations in various digital circuits.