Project 6: Carry Skip Adders

A Comprehensive Study of Advanced Digital Circuits

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

Contents

1 Introduction 3

2 Structure and Operation 3

3 Propagate and Generate Signals 3

4 Carry Skip Logic 3

5 Example: 16-bit Carry-Skip Adder 3

6 Circuit Diagram 4

7 RTL Code 4
- 7.1 Testbench 7
- 7.2 Simulation Results 8
- 7.3 Schematic 8
- 7.4 Synthesis Design 8

8 Advantages 8

9 Applications 8

10 Carry Skip Adder (CSKA) - `csea16` Module 9
- 10.1 Description 12

11 4-bit Ripple Carry Adder (RCA4) - `rca4` Module 12
- 11.1 Description 13

12 Full Adder - `FullAdder` Module 13
- 12.1 Description 13

13 2-to-1 Multiplexer with 4-bit Inputs - `mux21x4` Module 13
- 13.1 Description 14

14 Conclusion 14

1 Introduction

A Carry-Skip Adder (CSA) is a type of adder designed to improve the speed of binary addition by allowing the carry bit to skip certain groups of bits under specific conditions, reducing the overall propagation delay. It combines elements of both Ripple Carry Adders and Carry-Lookahead Adders.

2 Structure and Operation

The Carry-Skip Adder is divided into multiple blocks or groups of bits. Each block generates its own carry-out signal, but the carry-out signal can skip over an entire block if certain conditions are met. This structure enables the adder to bypass blocks where the carry does not change, thereby reducing the delay.

- **Groups:** The adder is divided into k groups, each containing n bits.
- **Ripple Carry Adders within Groups:** Each group uses a simple Ripple Carry Adder to compute the sum and carry within the group.
- **Skip Logic:** Each group includes skip logic to determine if the carry can bypass the group. The skip condition is that all the propagate signals within the group are true.

3 Propagate and Generate Signals

- **Generate (G):** Indicates whether a carry is generated at a particular bit position.

$$G_i = A_i \land B_i \tag{1}$$

- **Propagate (P):** Indicates whether a carry will be propagated through a particular bit position.

$$P_i = A_i \oplus B_i \tag{2}$$

4 Carry Skip Logic

For a group to allow the carry to skip through it, the propagate condition for all bits within the group must be true:

$$P_{group} = P_i \land P_{i+1} \land \cdots \land P_{i+n-1} \tag{3}$$

If P_{group} is true, the carry-in for the group will be the same as the carry-out, allowing the carry to bypass the group.

5 Example: 16-bit Carry-Skip Adder

Consider a 16-bit Carry-Skip Adder divided into four groups, each containing 4 bits.

- **Group 1 (Bits 0-3):** Ripple Carry Adder for bits 0 to 3. Skip logic checks $P_0 \land P_1 \land P_2 \land P_3$.
- **Group 2 (Bits 4-7):** Ripple Carry Adder for bits 4 to 7. Skip logic checks $P_4 \land P_5 \land P_6 \land P_7$.
- **Group 3 (Bits 8-11):** Ripple Carry Adder for bits 8 to 11. Skip logic checks $P_8 \land P_9 \land P_{10} \land P_{11}$.
- **Group 4 (Bits 12-15):** Ripple Carry Adder for bits 12 to 15. Skip logic checks $P_{12} \land P_{13} \land P_{14} \land P_{15}$.

6 Circuit Diagram

The circuit diagram would involve multiple ripple carry adders connected in sequence with skip logic in place to allow carry to bypass groups.

```
Group 1:       Group 2:       Group 3:       Group 4:
Ripple Carry   Ripple Carry   Ripple Carry   Ripple Carry
Adder          Adder          Adder          Adder

Skip Logic  --> Skip Logic  --> Skip Logic  --> Skip Logic
```

7 RTL Code

Listing 1: 16 Bit Cary Skip Adder RTL Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: abhishek sharma
//
// Create Date: 18.08.2024 15:24:38
// Design Name:
// Module Name: csea16
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module csea16 (
    input  logic [15:0] A, B,
    input  logic Cin,
    output logic [15:0] Sum,
    output logic Cout
);

    logic [3:0] Sum0_0, Sum0_1, Sum1_0, Sum1_1, Sum2_0, Sum2_1;
    logic c4, g4, p4;
    logic c8, g8, p8;
    logic c12, g12, p12;
    logic w1, w2, w3;

    // Instantiate 4-bit ripple carry adders
    rca4 rca1 (
        .Sum(Sum[3:0]),
        .Cout(c4),
        .A(A[3:0]),
        .B(B[3:0]),
        .Cin(Cin)
    );

    rca4 rca2 (
        .Sum(Sum0_0),
        .Cout(g4),
        .A(A[7:4]),
        .B(B[7:4]),
        .Cin(1'b0)
    );

    rca4 rca3 (
        .Sum(Sum0_1),
        .Cout(p4),
        .A(A[7:4]),
        .B(B[7:4]),
        .Cin(1'b1)
    );

    rca4 rca4 (
        .Sum(Sum1_0),
        .Cout(g8),
        .A(A[11:8]),
        .B(B[11:8]),
        .Cin(1'b0)
    );

    rca4 rca5 (
        .Sum(Sum1_1),
        .Cout(p8),
        .A(A[11:8]),
        .B(B[11:8]),
        .Cin(1'b1)
    );

    rca4 rca6 (
        .Sum(Sum2_0),
        .Cout(g12),
        .A(A[15:12]),
        .B(B[15:12]),
        .Cin(1'b0)
    );

    rca4 rca7 (
        .Sum(Sum2_1),
        .Cout(p12),
        .A(A[15:12]),
        .B(B[15:12]),
        .Cin(1'b1)
    );

    // Carry Skip Logic
    and a1 (w1, c4, p4);
    or  o1 (c8, w1, g4);
    mux21x4 mux1 (
        .Y(Sum[7:4]),
        .A0(Sum0_0),
        .A1(Sum0_1),
        .S(c4)
    );

    and a2 (w2, c8, p8);
    or  o2 (c12, w2, g8);
    mux21x4 mux2 (
        .Y(Sum[11:8]),
        .A0(Sum1_0),
        .A1(Sum1_1),
        .S(c8)
    );

    and a3 (w3, c12, p12);
    or  o3 (Cout, w3, g12);
    mux21x4 mux3 (
        .Y(Sum[15:12]),
        .A0(Sum2_0),
        .A1(Sum2_1),
        .S(c12)
    );

endmodule // csea16
module rca4 (
    input  logic [3:0] A, B,
    input  logic Cin,
    output logic [3:0] Sum,
    output logic Cout
);

    logic [3:0] carry;

    FullAdder fa0 (
        .A(A[0]),
        .B(B[0]),
        .Cin(Cin),
        .Sum(Sum[0]),
        .Cout(carry[0])
    );

    FullAdder fa1 (
        .A(A[1]),
        .B(B[1]),
        .Cin(carry[0]),
        .Sum(Sum[1]),
        .Cout(carry[1])
    );

    FullAdder fa2 (
        .A(A[2]),
        .B(B[2]),
        .Cin(carry[1]),
        .Sum(Sum[2]),
        .Cout(carry[2])
    );

    FullAdder fa3 (
        .A(A[3]),
        .B(B[3]),
        .Cin(carry[2]),
        .Sum(Sum[3]),
        .Cout(Cout)
    );

endmodule // rca4
module FullAdder (
    input  logic A,
    input  logic B,
    input  logic Cin,
    output logic Sum,
    output logic Cout
);

    assign Sum  = A ^ B ^ Cin;
    assign Cout = (A & B)  (Cin & (A ^ B));

endmodule // FullAdder
module mux21x4 (
    input  logic [3:0] A0, A1,
    input  logic S,
    output logic [3:0] Y
);

    assign Y = S ? A1 : A0;

endmodule // mux21x4
```

7.1 Testbench

Listing 2: 16 Bit Carry Skip Adder Testbench

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: abhishek sharma
//
// Create Date: 18.08.2024 15:25:21
// Design Name:
// Module Name: csea16tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module csea16tb;

    logic [15:0] A, B;
    logic Cin;
    logic [15:0] Sum;
    logic Cout;

    // Instantiate the Carry Skip Adder
    csea16 uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout)
    );

    // Test sequence
    initial begin
        // Apply test vectors
        A = 16'h0000; B = 16'h0000; Cin = 0;
        #10; // Wait for 10 time units
        $display("A = %h, B = %h, Cin = %b, Sum = %h, Cout = %b", A,
           B, Cin, Sum, Cout);

        A = 16'hFFFF; B = 16'h0001; Cin = 0;
        #10; // Wait for 10 time units
        $display("A = %h, B = %h, Cin = %b, Sum = %h, Cout = %b", A,
           B, Cin, Sum, Cout);

        A = 16'h1234; B = 16'h5678; Cin = 1;
        #10; // Wait for 10 time units
        $display("A = %h, B = %h, Cin = %b, Sum = %h, Cout = %b", A,
           B, Cin, Sum, Cout);

        A = 16'hFFFF; B = 16'hFFFF; Cin = 1;
        #10; // Wait for 10 time units
        $display("A = %h, B = %h, Cin = %b, Sum = %h, Cout = %b", A,
           B, Cin, Sum, Cout);

        // Finish the simulation
        $finish;
    end

endmodule // tb_csea16
```

7.2 Simulation Results

7.3 Schematic

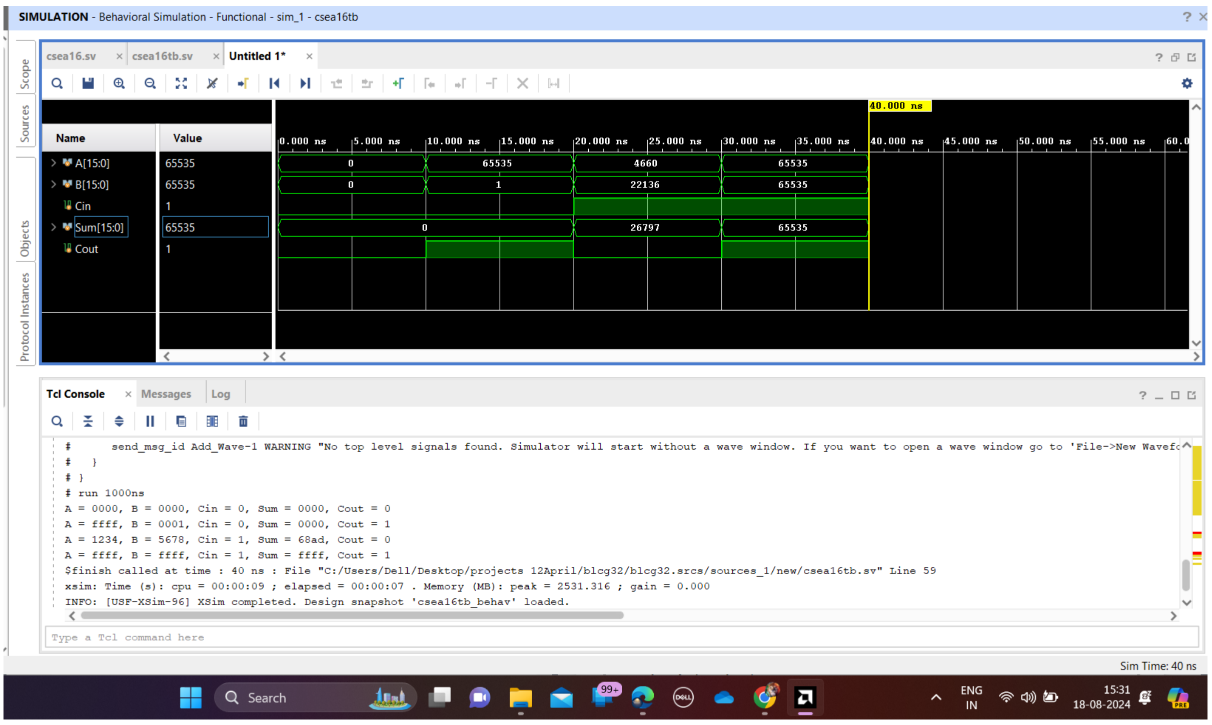
7.4 Synthesis Design

8 Advantages

- **Reduced Delay:** By allowing the carry to skip certain groups, the delay associated with carry propagation is significantly reduced.
- **Scalability:** The structure is easily scalable to larger bit-widths by adding more groups.

9 Applications

- **High-Speed Arithmetic Units:** Used in ALUs where speed is critical.
- **Digital Signal Processing (DSP):** Used in DSP applications requiring fast addition operations.



```
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window. If you want to open a wave window go to 'File->New Wavefc
#   }
# }
# run 1000ns
A = 0000, B = 0000, Cin = 0, Sum = 0000, Cout = 0
A = ffff, B = 0001, Cin = 0, Sum = 0000, Cout = 1
A = 1234, B = 5678, Cin = 1, Sum = 68ad, Cout = 0
A = ffff, B = ffff, Cin = 1, Sum = ffff, Cout = 1
$finish called at time : 40 ns : File "C:/Users/Dell/Desktop/projects 12April/blcg32/blcg32.srcs/sources_1/new/csea16tb.sv" Line 59
xsim: Time (s): cpu = 00:00:09 ; elapsed = 00:00:07 . Memory (MB): peak = 2531.316 ; gain = 0.000
INFO: [USF-XSim-96] XSim completed. Design snapshot 'csea16tb_behav' loaded.
```

Figure 1: Simulation results of 16 Bit Carry Skip Adder

10 Carry Skip Adder (CSKA) - csea16 Module

Listing 3: CSKA 16-bit Adder

```
module csea16 (
    input  logic [15:0] A, B,
    input  logic Cin,
    output logic [15:0] Sum,
    output logic Cout
);
    logic [3:0] Sum0_0, Sum0_1, Sum1_0, Sum1_1, Sum2_0, Sum2_1;
    logic c4, g4, p4;
    logic c8, g8, p8;
    logic c12, g12, p12;
    logic w1, w2, w3;

    // Instantiate 4-bit ripple carry adders
    rca4 rca1 (
        .Sum(Sum[3:0]),
        .Cout(c4),
        .A(A[3:0]),
        .B(B[3:0]),
        .Cin(Cin)
    );

    rca4 rca2 (
        .Sum(Sum0_0),
        .Cout(g4),
        .A(A[7:4]),
        .B(B[7:4]),
        .Cin(1'b0)
    );
```

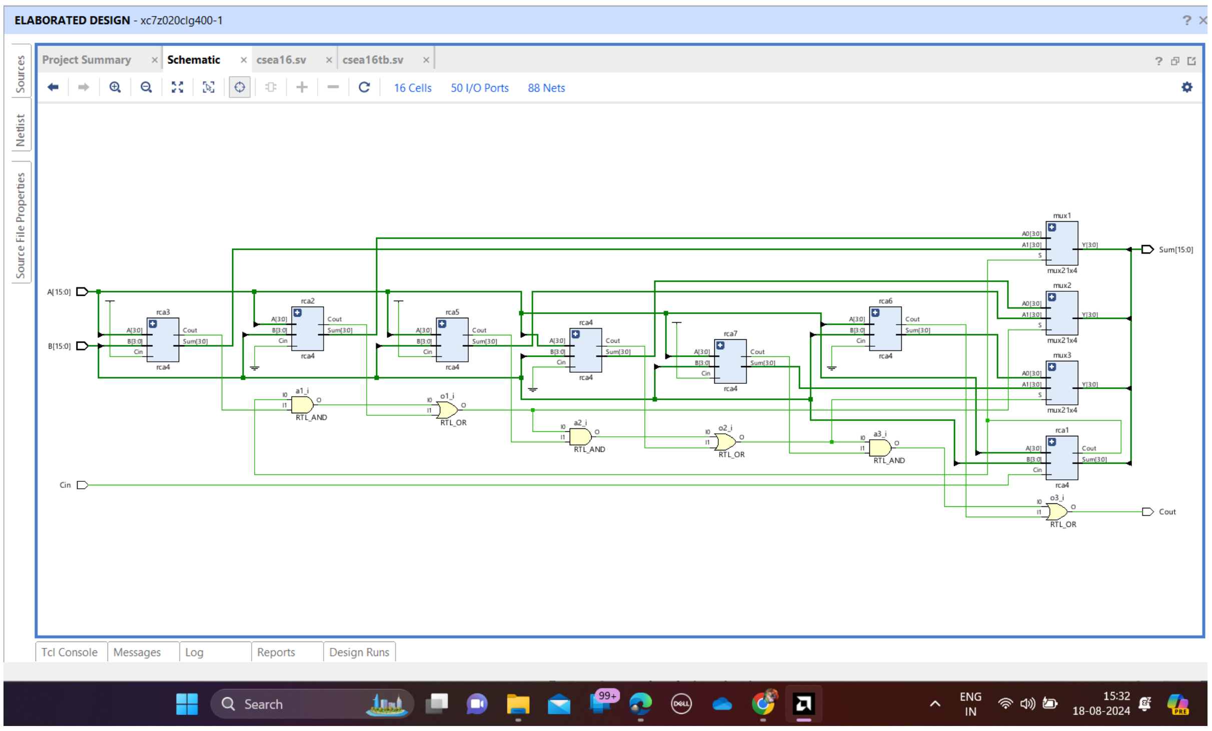


Figure 2: Schematic of 16 Bit Carry Skip Adder

```
    rca4 rca3 (
        .Sum(Sum0_1),
        .Cout(p4),
        .A(A[7:4]),
        .B(B[7:4]),
        .Cin(1'b1)
    );

    rca4 rca4 (
        .Sum(Sum1_0),
        .Cout(g8),
        .A(A[11:8]),
        .B(B[11:8]),
        .Cin(1'b0)
    );

    rca4 rca5 (
        .Sum(Sum1_1),
        .Cout(p8),
        .A(A[11:8]),
        .B(B[11:8]),
        .Cin(1'b1)
    );

    rca4 rca6 (
        .Sum(Sum2_0),
        .Cout(g12),
        .A(A[15:12]),
        .B(B[15:12]),
        .Cin(1'b0)
    );
```

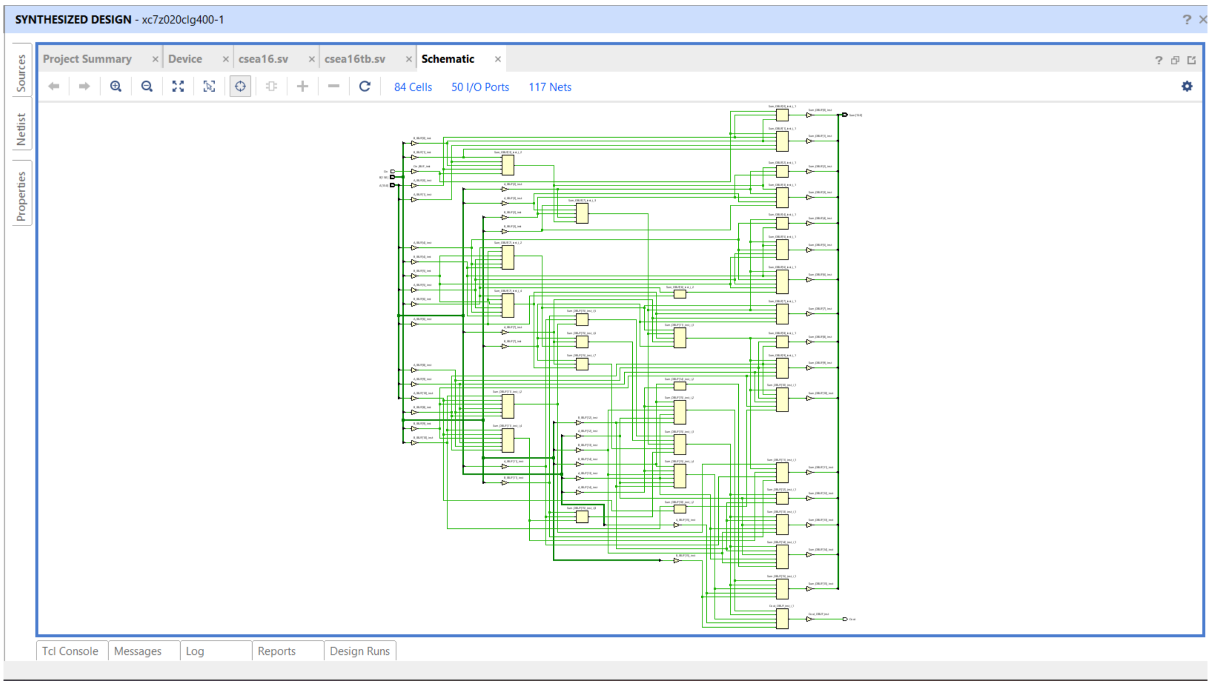


Figure 3: Synthesis Design 16 Bit Carry Skip Adder

```

    rca4 rca7 (
        .Sum(Sum2_1),
        .Cout(p12),
        .A(A[15:12]),
        .B(B[15:12]),
        .Cin(1'b1)
    );

    // Carry Skip Logic
    and a1 (w1, c4, p4);
    or  o1 (c8, w1, g4);
    mux21x4 mux1 (
        .Y(Sum[7:4]),
        .A0(Sum0_0),
        .A1(Sum0_1),
        .S(c4)
    );

    and a2 (w2, c8, p8);
    or  o2 (c12, w2, g8);
    mux21x4 mux2 (
        .Y(Sum[11:8]),
        .A0(Sum1_0),
        .A1(Sum1_1),
        .S(c8)
    );

    and a3 (w3, c12, p12);
    or  o3 (Cout, w3, g12);
    mux21x4 mux3 (
        .Y(Sum[15:12]),
        .A0(Sum2_0),
```

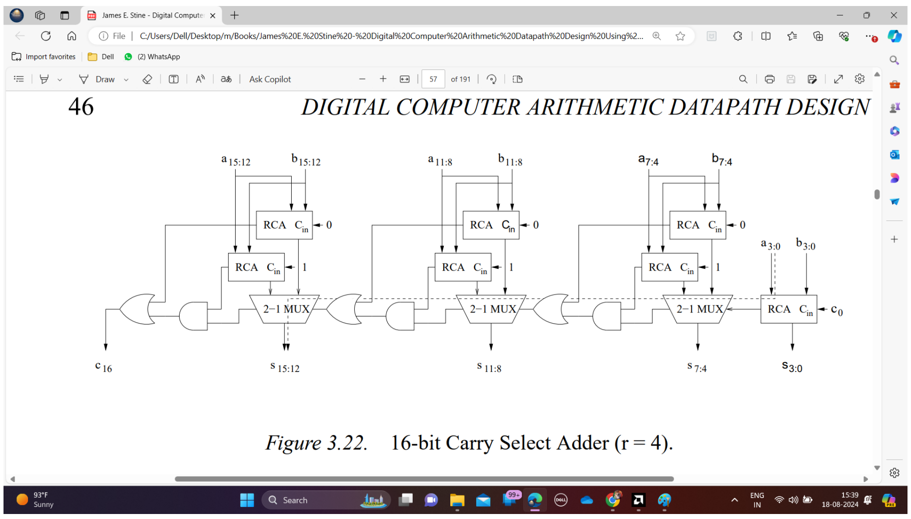

46 DIGITAL COMPUTER ARITHMETIC DATAPATH DESIGN

Figure 3.22. 16-bit Carry Select Adder (r = 4).

Figure 4: Block Diagram

```
            .A1(Sum2_1),
            .S(c12)
        );

endmodule // csea16
```

10.1 Description

The `csea16` module implements a 16-bit Carry Skip Adder. It uses several 4-bit Ripple Carry Adders (`rca4`) to compute partial sums and carry outputs for 4-bit segments of the 16-bit input numbers. The carry skip logic is applied to improve the speed of addition by skipping some carries.

11 4-bit Ripple Carry Adder (RCA4) - `rca4` Module

Listing 4: 4-bit Ripple Carry Adder

```
module rca4 (
    input  logic [3:0] A, B,
    input  logic Cin,
    output logic [3:0] Sum,
    output logic Cout
);

    logic [3:0] carry;

    FullAdder fa0 (
        .A(A[0]),
        .B(B[0]),
        .Cin(Cin),
        .Sum(Sum[0]),
        .Cout(carry[0])
    );

    FullAdder fa1 (
        .A(A[1]),
        .B(B[1]),
        .Cin(carry[0]),
        .Sum(Sum[1]),
        .Cout(carry[1])
    );

    FullAdder fa2 (
        .A(A[2]),
        .B(B[2]),
        .Cin(carry[1]),
        .Sum(Sum[2]),
        .Cout(carry[2])
    );

    FullAdder fa3 (
        .A(A[3]),
        .B(B[3]),
        .Cin(carry[2]),
        .Sum(Sum[3]),
        .Cout(Cout)
    );

endmodule // rca4
```

11.1 Description

The `rca4` module is a 4-bit Ripple Carry Adder. It uses four `FullAdder` modules to add two 4-bit numbers, taking into account a carry-in and generating a carry-out.

12 Full Adder - `FullAdder` Module

Listing 5: Full Adder

```
module FullAdder (
    input  logic A,
    input  logic B,
    input  logic Cin,
    output logic Sum,
    output logic Cout
);

    assign Sum  = A ^ B ^ Cin;
    assign Cout = (A & B)  (Cin & (A ^ B));

endmodule // FullAdder
```

12.1 Description

The `FullAdder` module performs single-bit addition. It computes the sum and carry-out from the two input bits and the carry-in.

13 2-to-1 Multiplexer with 4-bit Inputs - `mux21x4` Module

Listing 6: 2-to-1 Multiplexer

```
module mux21x4 (
    input  logic [3:0] A0, A1,
    input  logic S,
    output logic [3:0] Y
);

    assign Y = S ? A1 : A0;

endmodule // mux21x4
```

13.1 Description

The mux21x4 modul

14 Conclusion

The Carry-Skip Adder efficiently combines the simplicity of Ripple Carry Adders with the speed advantages of bypassing carry propagation in certain conditions, making it a useful design in high-speed arithmetic operations.