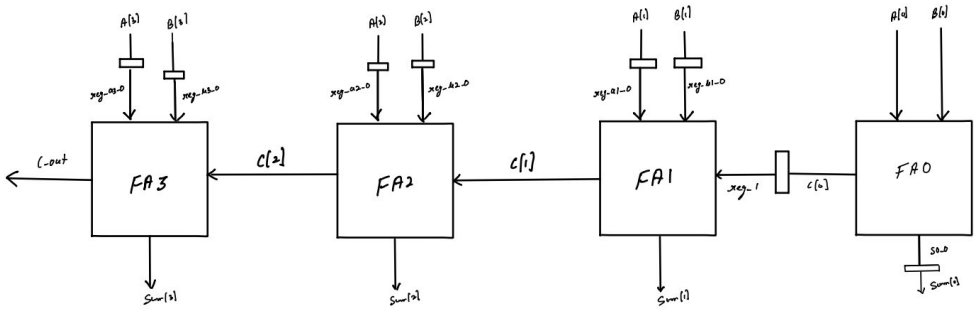
1 4-bit RCA with Single-Stage Pipelining

1.1 Introduction

A 4-bit Ripple Carry Adder (RCA) is a digital circuit used for binary addition. It consists of four full adders connected in series, where each full adder adds corresponding bits from two input numbers and the carry bit from the previous adder. The first adder takes in the least significant bits (LSB) of the two numbers along with a carry-in bit. The sum and carry-out are passed to the next full adder, which adds the next set of bits, propagating the carry until the most significant bit (MSB) is reached. The RCA works by sequentially propagating the carry, which can lead to delays when adding large numbers.

To overcome the delay introduced by the carry propagation, pipelining can be implemented. In the case of a 4-bit RCA with single-stage pipelining, the carry propagation is divided into stages, allowing for faster computation by breaking the circuit into two stages. The first stage performs the addition for two bits and the carry, while the second stage performs the addition for the remaining bits. A pipeline register is inserted between these stages to store intermediate carry values, reducing the overall delay and improving throughput. This single-stage pipelined approach enhances performance by allowing for parallel processing of subsequent additions while still maintaining the functionality of the standard RCA.



1.2 Verilog Topmodule

```
module RCA_4bit( input wire clock, reset,
```

```
                    input wire [3:0]a,b, output wire
                    [3:0]sum, output wire carry_out
                    );

  reg sum_0; wire
sum_1,sum_2,sum_3; reg reg_1; //
carry pipeplined

reg reg_a1_0; // a[1] pipelined reg
reg_b1_0; // b[1] pipelined

reg reg_a2_0; // a[2] pipelined reg
reg_b2_0; // b[2] pipelined

reg reg_a3_0; // a[3] pipelined reg
reg_b3_0; // b[3] pipelined wire s0_0;
wire [3:0]c;

assign s0_0 = a[0] ^ b[0]; // HA_0 assign c[0] = a[0]
& b[0];

assign sum_1 = reg_1 ^ reg_b1_0 ^ reg_a1_0; // FA1 assign c[1] =
(reg_a1_0 & reg_b1_0) | (reg_a1_0 & reg_1) |
                        (reg_b1_0 & reg_1);

assign sum_2 = c[1] ^ reg_b2_0 ^ reg_a2_0; // FA2 assign c[2] =
(reg_a2_0 & reg_b2_0) | (reg_a2_0 & c[1]) |
                          (reg_b2_0 & c[1]);

assign sum_3 = c[2] ^ reg_b3_0 ^ reg_a3_0; // FA3 assign carry_out =
(reg_a3_0 & reg_b3_0) | (reg_a3_0 & c[2]) |
                                (reg_b3_0 & c[2]);


always @(posedge clock, posedge reset)
  begin if(reset)
     begin
```

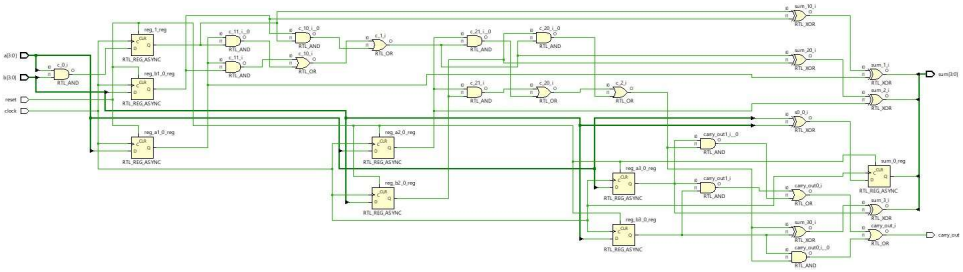
```
        reg_1 <= 1'b0;
    reg_a1_0 <= 1'b0;
    reg_b1_0 <= 1'b0;
    reg_a2_0 <= 1'b0;
    reg_b2_0 <= 1'b0;
    reg_a3_0 <= 1'b0;
    reg_b3_0 <= 1'b0;
    sum_0 <= 1'b0; end else
      begin
        reg_1 <= c[0];
      reg_a1_0 <= a[1];
      reg_b1_0 <= b[1];
      reg_a2_0 <= a[2];
      reg_b2_0 <= b[2];
      reg_a3_0 <= a[3];
      reg_b3_0 <= b[3];
      sum_0 <= s0_0; end
    end assign sum =

{sum_3,sum_2,sum_1,sum_0}; endmodule
```

1.3 Verilog Testbench

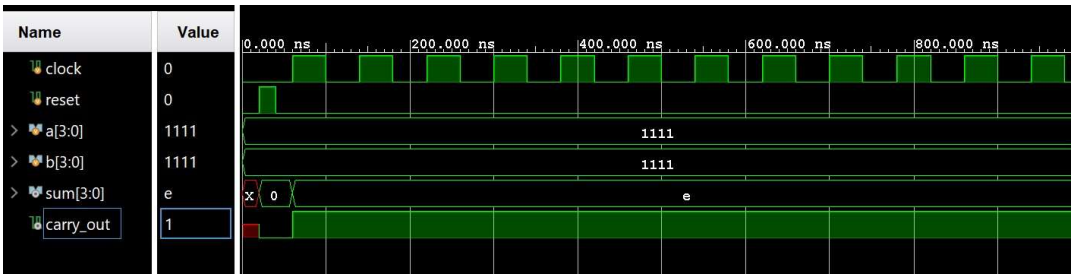
```
'timescale 1ns / 1ps module

RCA_4bit_tb();

reg clock,reset; reg
[3:0]a,b;

wire [3:0]sum; wire
carry_out; RCA_4bit
dut( .clock(clock),
                    .reset(reset),
                  .a(a),
                  .b(b),
```

```
                    .sum(sum),
                   .carry_out(carry_out) );


initial
  begin
          reset = 1'b0; #20
    reset = 1'b1;
    #20 reset = 1'b0; end

initial
  begin
          clock = 1'b0; #60 clock =
  1'b1; forever #40 clock = ~clock;
  end

initial
  begin
    a = 4'b1111; b =
  4'b1111; end

initial
  #10000 $finish; endmodule
```

Schematic



Simulations

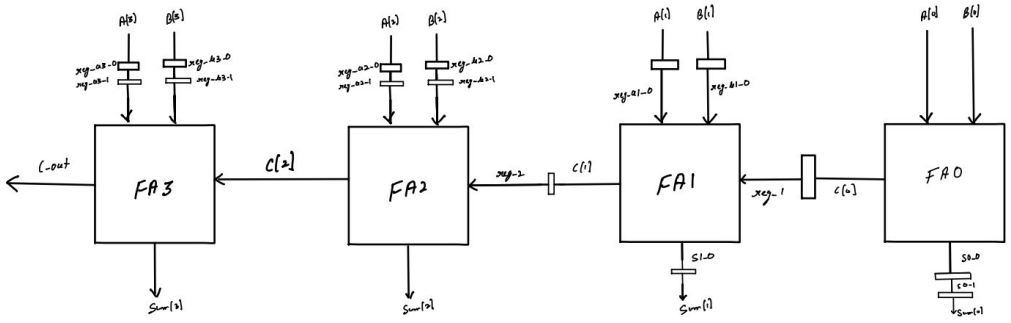


2 4-bit RCA with 2 Stages of Pipelining

2.1 Introduction

A 4-bit Ripple Carry Adder (RCA) is a simple digital circuit used to add two 4bit binary numbers. It consists of four full adders, where each full adder adds two corresponding bits from the input numbers and the carry from the previous stage. The carry output of each adder is propagated to the next, hence the term ”ripple carry.” The sum outputs of each full adder are combined to form the final sum of the two input numbers. The RCA’s main limitation is the propagation delay, as the carry must ripple through all stages sequentially.

To improve the performance of the 4-bit RCA, pipelining can be introduced. In the case of two-stage pipelining, the adder is divided into two groups: the first stage consists of two full adders, and the second stage consists of the remaining two adders. The carry from the first stage is registered and passed to the second stage, allowing the carry propagation to occur in parallel with the sum computation. This technique reduces the overall delay by allowing each stage to operate concurrently, leading to faster operation when used in a larger system where multiple adders are required in sequence.



2.2 Verilog Topmodule

```
‘timescale 1ns / 1ps

module RCA_4bit( input wire clock, reset,
                 input wire [3:0]a,b, output wire
                 [3:0]sum, output wire carry_out
                 ); reg sum_0,sum_1; wire
                 sum_2,sum_3; reg reg_1,reg_2;
                 // carry pipeplined
    reg reg_a1_0; // a[1] pipelined reg
    reg_b1_0; // b[1] pipelined

    reg reg_a2_0,reg_a2_1; // a[2] pipelined reg
    reg_b2_0,reg_b2_1; // b[2] pipelined

    reg reg_a3_0,reg_a3_1; // a[3] pipelined reg
    reg_b3_0,reg_b3_1; // b[3] pipelined

    wire  s0_0;  reg

    s0_1;       wire

    s1_0;       wire

    [3:0]c;

    assign s0_0 = a[0] ^ b[0]; // HA_0 assign c[0] = a[0]
    & b[0];

    assign s1_0 = reg_1 ^ reg_b1_0 ^ reg_a1_0; // FA1 assign c[1] =
    (reg_a1_0 & reg_b1_0) | (reg_a1_0 & reg_1) |
                    (reg_b1_0 & reg_1);

    assign sum_2 = reg_2 ^ reg_b2_1 ^ reg_a2_1; // FA2 assign c[2] =
    (reg_a2_1 & reg_b2_1) | (reg_a2_1 & reg_2) |
                    (reg_b2_1 & reg_2);

    assign sum_3 = c[2] ^ reg_b3_1 ^ reg_a3_1; // FA3 assign carry_out =
    (reg_a3_1 & reg_b3_1) | (reg_a3_1 & c[2]) |
```

```
                    (reg_b3_1 & c[2]);


always @(posedge clock, posedge reset)
  begin
    if(reset)
      begin
         reg_1 <= 1'b0; reg_2 <=
         1'b0;

  reg_a1_0 <= 1'b0; reg_b1_0 <=
                    1'b0;

  reg_a2_0 <= 1'b0; reg_b2_0 <=
       1'b0; reg_a2_1 <= 1'b0;
          reg_b2_1 <= 1'b0;

  reg_a3_0 <= 1'b0; reg_b3_0 <=
       1'b0; reg_a3_1 <= 1'b0;
          reg_b3_1 <= 1'b0;

         s0_1 <= 1'b0; sum_0
         <= 1'b0;

         sum_1 <= 1'b0;
    end else
      begin
       reg_1 <= c[0]; reg_2 <=
       c[1]; reg_a1_0 <= a[1];
       reg_b1_0 <= b[1];

       reg_a2_0 <= a[2]; reg_b2_0 <=
       b[2]; reg_a2_1 <= reg_a2_0;
       reg_b2_1 <= reg_b2_0;

       reg_a3_0 <= a[3]; reg_b3_0 <=
       b[3]; reg_a3_1 <= reg_a3_0;
       reg_b3_1 <= reg_b3_0; s0_1 <=
       s0_0; sum_0 <= s0_1;
```

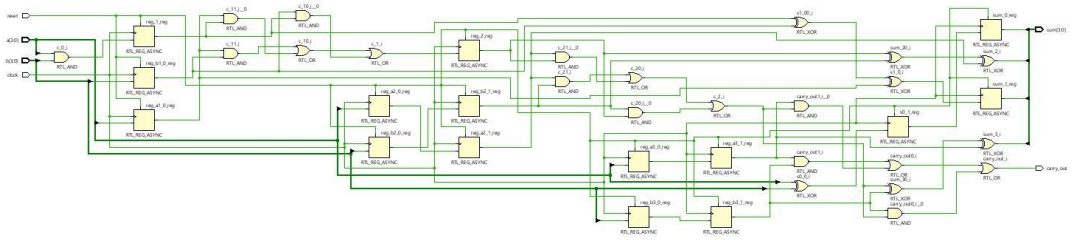
```
            sum_1 <= s1_0; end
        end assign sum =

{sum_3,sum_2,sum_1,sum_0}; endmodule
```

2.3 Verilog Testbench

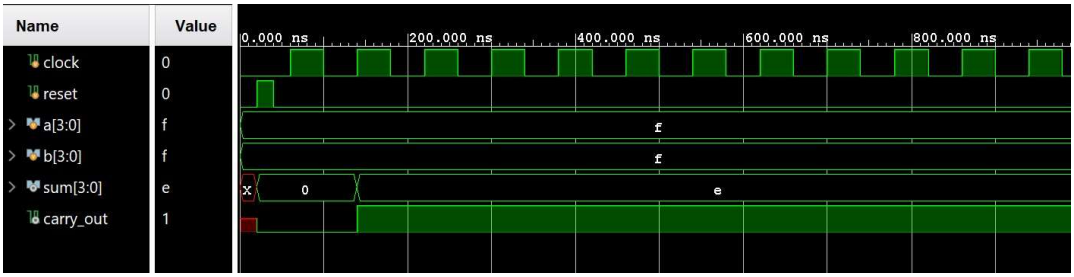
```
‘timescale 1ns / 1ps module

RCA_4bit_tb();

reg clock,reset; reg
[3:0]a,b;

wire [3:0]sum; wire
carry_out;

RCA_4bit dut( .clock(clock),
                .reset(reset),
              .a(a),
              .b(b),
               .sum(sum),
              .carry_out(carry_out) );


initial
  begin reset = 1’b0;
    #20 reset = 1’b1;
    #20 reset = 1’b0; end

initial

  begin
        clock = 1’b0; #60 clock =
  1’b1; forever #40 clock = ~clock;
  end

initial
  begin
```

```
   a = 4'b1111; b =
 4'b1111; end

initial
  #10000 $finish; endmodule
```

Schematic



Simulations



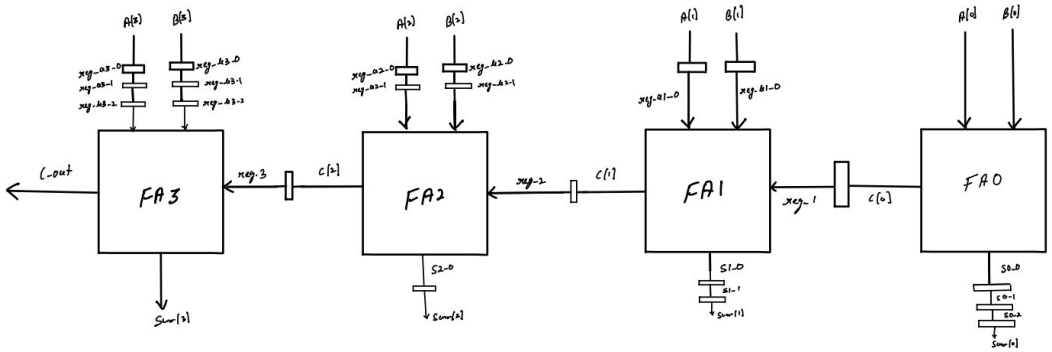
3 4-bit RCA with 4 Stages of Pipelining

3.1 Introduction

A 4-bit Ripple Carry Adder (RCA) is a simple and commonly used adder that can perform binary addition on two 4-bit numbers. The RCA consists of four full adder blocks connected in series, where each block adds one bit of the numbers, along with the carry from the previous stage. The input bits are A_3,A_2,A_1,A_0 and B_3,B_2,B_1,B_0, and the output is the sum S_3,S_2,S_1,S_0 along with the carry out C_4. The carry propagation is sequential, which means the carry from one stage has to ripple through the subsequent stages, introducing a delay proportional to the number of bits being added.

To improve the performance and reduce the propagation delay, pipelining is often employed. In this design, the 4-bit RCA is divided into 4 stages, where each stage

contains one full adder. Between each stage, registers are inserted to store intermediate values of the carry, effectively breaking the long chain of carry propagation into smaller, more manageable segments. This pipelining reduces the overall delay of the adder by allowing parallel computation of multiple stages. Each clock cycle allows the carry to propagate through one stage, resulting in faster addition and better performance for high-speed applications.



3.2 Verilog Topmodule

‘timescale 1ns / 1ps

```
module RCA_4bit( input wire clock, reset,
                 input wire [3:0]a,b,
                 output wire [3:0]sum, output
                 wire carry_out );

  reg sum_0,sum_1,sum_2; wire sum_3; reg
 reg_1,reg_2,reg_3; // carry pipeplined

 reg reg_a1_0; // a[1] pipelined reg
 reg_b1_0; // b[1] pipelined

 reg reg_a2_0,reg_a2_1; // a[2] pipelined reg
 reg_b2_0,reg_b2_1; // b[2] pipelined

 reg reg_a3_0,reg_a3_1,reg_a3_2; // a[3] pipelined reg
 reg_b3_0,reg_b3_1,reg_b3_2; // b[3] pipelined
```

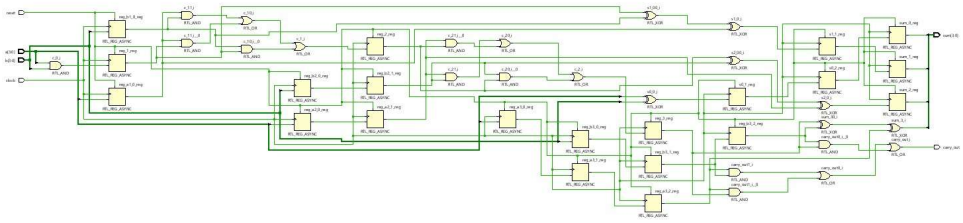
```
wire s0_0; reg
s0_1,s0_2;

wire s1_0; reg

s1_1; wire s2_0;

wire [3:0]c;

assign s0_0 = a[0] ^ b[0]; // HA_0 assign c[0] = a[0]
& b[0];

assign s1_0 = reg_1 ^ reg_b1_0 ^ reg_a1_0; // FA1 assign c[1] =
(reg_a1_0 & reg_b1_0) | (reg_a1_0 & reg_1) |
                        (reg_b1_0 & reg_1);

assign s2_0 = reg_2 ^ reg_b2_1 ^ reg_a2_1; // FA2 assign c[2] =
(reg_a2_1 & reg_b2_1) | (reg_a2_1 & reg_2) |
                        (reg_b2_1 & reg_2);

assign sum_3 = reg_3 ^ reg_b3_2 ^ reg_a3_2; // FA3 assign carry_out =
(reg_a3_2 & reg_b3_1) | (reg_a3_2 & reg_3) | (reg_b3_2 & reg_3);


always @(posedge clock, posedge reset)
  begin if(reset)
     begin
        reg_1 <= 1'b0; reg_2 <=
        1'b0; reg_3 <= 1'b0;

        reg_a1_0 <= 1'b0; reg_b1_0 <=
        1'b0;

        reg_a2_0 <= 1'b0; reg_b2_0 <=
        1'b0; reg_a2_1 <= 1'b0;
        reg_b2_1 <= 1'b0;

        reg_a3_0 <= 1'b0; reg_b3_0 <=
        1'b0; reg_a3_1 <= 1'b0;
```

```
            reg_b3_1 <= 1'b0; reg_a3_2 <=
            1'b0; reg_b3_2 <= 1'b0;

            s0_1 <= 1'b0; s0_2
            <= 1'b0; sum_0 <=
            1'b0;

            s1_1 <= 1'b0; sum_1
            <= 1'b0;

            sum_2 <= 1'b0;
        end else
          begin
            reg_1 <= c[0]; reg_2 <=
            c[1]; reg_3 <= c[2];
            reg_a1_0 <= a[1];
            reg_b1_0 <= b[1];

            reg_a2_0 <= a[2]; reg_b2_0 <=
            b[2]; reg_a2_1 <= reg_a2_0;
            reg_b2_1 <= reg_b2_0;

            reg_a3_0 <= a[3]; reg_b3_0 <=
            b[3]; reg_a3_1 <= reg_a3_0;
            reg_b3_1 <= reg_b3_0;
            reg_a3_2 <= reg_a3_1;
            reg_b3_2 <= reg_b3_1;

            s0_1 <= s0_0; s0_2
            <= s0_1; sum_0 <=
            s0_2;

            s1_1 <= s1_0; sum_1
            <= s1_1;

            sum_2 <= s2_0; end
        end assign sum =

{sum_3,sum_2,sum_1,sum_0}; endmodule
```

3.3 Verilog Testbench

```
‘timescale 1ns / 1ps module

RCA_4bit_tb();

reg clock,reset; reg
[3:0]a,b; wire [3:0]sum;
wire carry_out;

RCA_4bit dut( .clock(clock),
                .reset(reset),
               .a(a),
               .b(b),
              .sum(sum),
             .carry_out(carry_out) );


initial
  begin
        reset = 1’b0; #20
    reset = 1’b1;
    #20 reset = 1’b0; end

initial
  begin
        clock = 1’b0; #60 clock =
  1’b1; forever #40 clock = ~clock;
  end

initial
  begin
    a = 4’b1111; b =
  4’b1111; end

initial
  #10000 $finish; endmodule
```

Schematic



Simulations