Assignment 2

Arnav Yadnopavit
EE24BTECH11007

April 21, 2025

1 Problem 1: Convolution of Two 8-element Vectors

Objective

Design a Verilog module to perform discrete convolution on two 8-element input vectors (4-bit each).

Code

1_convolution.v

```
module convolution(input  [31:0] x_in,input  [31:0] h_in,output reg
   [63:0] y_out);
    integer i,j;
    reg [3:0] x [7:0];
    reg [3:0] h [7:0];
    reg [7:0] temp_y [15:0];
    always @(*) begin
        for (i=0;i<8;i=i+1) begin
            x[i]=x_in[i*4 +: 4];
            h[i]=h_in[i*4 +: 4];
        end
        for (i=0;i<16;i++)
            temp_y[i]=0;
        for (i=0;i<16;i=i+1) begin
            for (j=0;j<8;j=j+1) begin
                if (i>=j&&(i-j)<8) begin
                    temp_y[i]=temp_y[i]+x[i-j]*h[j];
                end
            end
        end
        for (i=0;i<16;i=i+1)
            y_out[i*4 +: 4]=temp_y[i][3:0];
    end
endmodule
```

1_test.v

```
`timescale 1ns/1ps

module test_conv;
reg [31:0] x_in,h_in;
wire [63:0] y_out;

convolution uut(.x_in(x_in),.h_in(h_in),.y_out(y_out));
initial begin
        $dumpfile("conv.vcd");
        $dumpvars(0,test_conv);

        $display("Time\tx_in\th_in\ty_out");
        $monitor("%0dns\t%b\t%b\t%b", $time, x_in,h_in, y_out);

        x_in = 32'h00000000;h_in = 32'h00000000;#10;
        x_in = 32'hFFFFFFFF;h_in = 32'hFFFFFFFF;#10;
        x_in = 32'h12345678;h_in = 32'h87654321; #10;
        x_in = 32'h10000000;h_in = 32'h12345678; #10;
        x_in = 32'h12345678;h_in = 32'h10000000; #10;
        $finish;
    end

endmodule
```

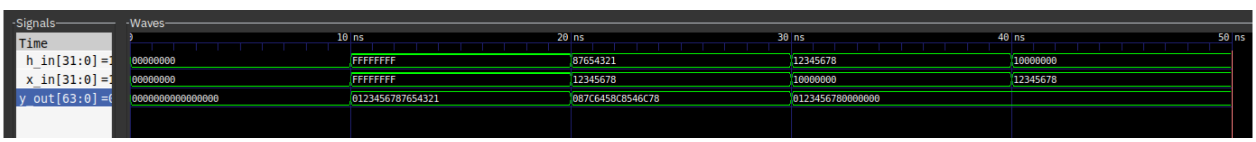


Figure 1: Timing Diagram

Explanation

This module performs a discrete convolution using nested loops over two 8-element arrays. For each output index, it computes the sum of products of overlapping elements. Overflow is ignored by truncating the result to 4 bits.

2 Problem 2: 8-bit Full Adder Using Loop

Objective

Design an 8-bit full adder using generate block (loop statement).

Code

2_adder.v

```
module EightBitAdder(input [7:0] a,input [7:0] b,input cin,output
   [7:0]sum, output cout);
genvar i;
wire [8:0] c;
assign c[0]=cin;
generate
        for(i=0;i<8;i=i+1)begin
                bitadder adders(.a(a[i]),.b(b[i]),.sum(sum[i]),.cin(
                   c[i]),.cout(c[i+1]));
        end
endgenerate
assign cout=c[8];
endmodule

module bitadder(input a,input b,input cin, output sum,output cout);
assign sum=a^b^cin;
assign cout=a&b|b&cin|cin&a;
endmodule
```

2_test.v

```
`timescale 1ns/1ps

module test_8bitadder;

reg [7:0]a,b;
reg cin;
wire [7:0] sum;
wire cout;

EightBitAdder dusra(.a(a),.b(b),.cin(cin),.sum(sum),.cout(cout));

initial begin
        $dumpfile("8bitadder.vcd");
        $dumpvars(0,test_8bitadder);

        $display("Time\ta\tb\tcin\tsum\tcout");
        $monitor("%0dns\t%b\t%b\t%b\t%b\t%b", $time, a,b,cin,sum,
           cout);
        a=8'b00000000;b=8'b00000000;cin=1'b1;#10;
        a=8'b11111111;b=8'b11111111;cin=1'b0;#10;
        a=8'b10101010;b=8'b11111111;cin=1'b1;#10;
        a=8'b10111011;b=8'b10101110;cin=1'b0;#10;
        a=8'b11010101;b=8'b10011101;cin=1'b1;#10;
```

```
        $finish;
end
endmodule
```

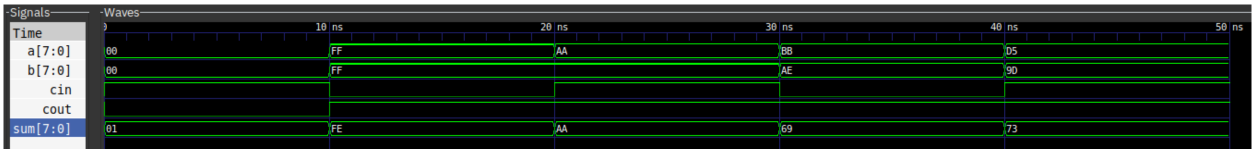


Figure 2: Timing Diagram

Explanation

The adder uses a generate block to instantiate 8 full adders for each bit. Carry propagates through a wire array. No arithmetic operators are used.

3 Problem 3: 4-bit Ripple Carry Adder using NAND Gates

Objective

Create a 4-bit ripple carry adder using only 2-input NAND gates with 1 ns delay.

Code

3_nandcounter.v

```
‘timescale 1ns/1ps
module RippleAdder(input [3:0]a,input [3:0]b,output [3:0] sum,output
    cout);
genvar i;
wire [4:0] c;
assign c[0]=0;
generate
        for(i=0;i<4;i=i+1)begin
                bitadder adders(.a(a[i]),.b(b[i]),.sum(sum[i]),.cin(
                   c[i]),.cout(c[i+1]));
        end
endgenerate
assign cout=c[4];
endmodule
module bitadder(input a,input b,input cin,output cout, output sum);
wire ws1;
xorgate w1(ws1,a,b);
xorgate w2(sum,cin,ws1);
```

```
wire wc1,wc2,wc3,wc4;
andgate ww1(wc1,a,b);
andgate ww2(wc2,b,cin);
andgate w3(wc3,cin,a);
orgate w4(wc4,wc1,wc2);
orgate w5(cout,wc4,wc3);

endmodule
module xorgate(output o,input a, input b);
wire w1,w2,w3;
nandgate ww1(w1,a,b);
nandgate ww2(w2,a,w1);
nandgate ww3(w3,b,w1);
nandgate ww4(o,w2,w3);
endmodule
module andgate(output o,input a,input b);
wire w1,w2;
nandgate ww1(w1,a,b);
nandgate ww2(w2,a,b);
nandgate ww3(o,w1,w2);
endmodule
module orgate(output o,input a,input b);
wire w1,w2;
nandgate ww1(w1,a,a);
nandgate ww2(w2,b,b);
nandgate ww3(o,w1,w2);
endmodule
module nandgate(output o,input a,input b);
assign #1 o=~(a&b);
endmodule
```

3_test.v

```
`timescale 1ns/1ps

module test_RippleAdder;

reg [3:0]a,b;
wire [3:0] sum;
wire cout;

RippleAdder tisra(.a(a),.b(b),.sum(sum),.cout(cout));

initial begin
        $dumpfile("8bitadder.vcd");
        $dumpvars(0,tisra);

        $display("Time\ta\tb\tsum\tcout");
```

```
        $monitor("%0dns\t%b\t%b\t%b\t%b", $time, a,b, sum,cout);
        a=4'b0000;b=4'b0000;#20;
        a=4'b1111;b=4'b1111;#20;
        a=4'b1010;b=4'b1111;#20;
        a=4'b1011;b=4'b1110;#20;
        a=4'b0101;b=4'b1101;#20;

        $finish;
end
endmodule
```

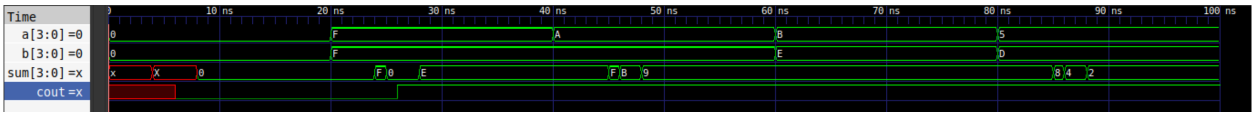


Figure 3: Timing Diagram

Explanation

The 4-bit ripple carry adder is implemented entirely using 2-input NAND gates. XOR, AND, and OR gates are constructed from NANDs. Delay per gate is set to 1 ns, and the ripple-carry effect propagates through the bitadder stages. Each bit's carry is passed to the next stage, emulating the behavior of a ripple-carry adder structurally.

Propagation Delay Analysis

Gate Delay Assumptions

Each 2-input NAND gate has a delay of 1 ns. All logic gates are built from these NANDs:

$$\begin{aligned}
\text{XOR (4 NANDs)} &\Rightarrow 4 \times 1\,\text{ns} = 4\,\text{ns}, \\
\text{AND (3 NANDs)} &\Rightarrow 3 \times 1\,\text{ns} = 3\,\text{ns}, \\
\text{OR (3 NANDs)} &\Rightarrow 3 \times 1\,\text{ns} = 3\,\text{ns}.
\end{aligned}$$

Full Adder (`bitadder`) Delay

Each 1-bit full adder has two paths:

$$\begin{aligned}
\text{Sum path:} \quad & 2\ \text{XORs} = 4\,\text{ns} + 4\,\text{ns} = 8\,\text{ns}, \\
\text{Carry path:} \quad & 3\ \text{ANDs} + 2\ \text{ORs} = 3 \times 3\,\text{ns} + 2 \times 3\,\text{ns} = 9\,\text{ns} + 6\,\text{ns} = 15\,\text{ns}.
\end{aligned}$$

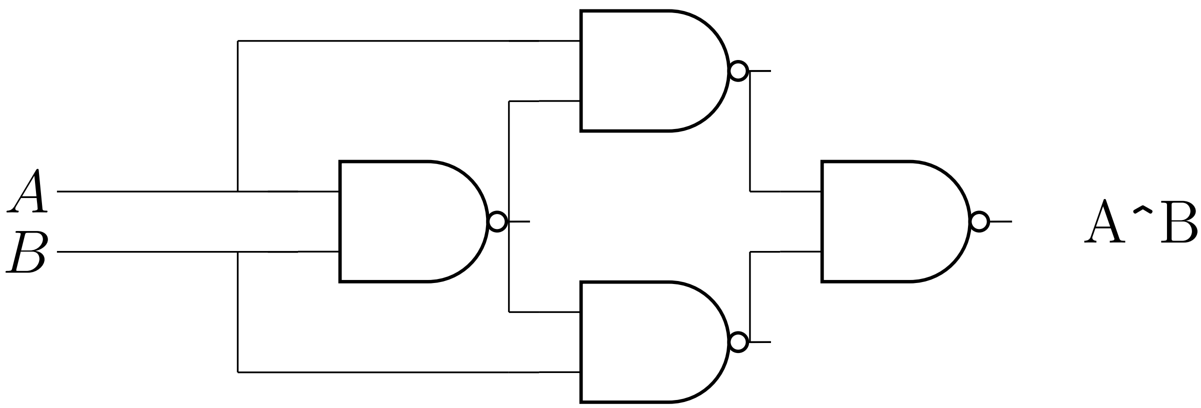
Worst-Case Delay for 4-bit Ripple Carry Adder

The carry ripples through three full-adder stages before the final sum bit:

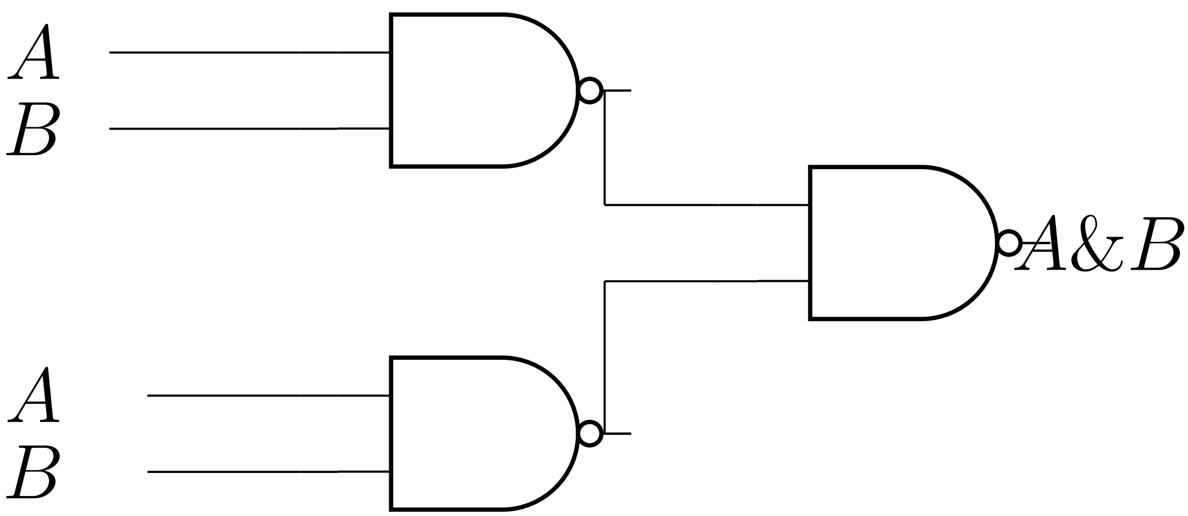
$$\text{Total worst-case delay} = 4 \times (\text{carry delay}) + (\text{sum delay}) = 4 \times 15\,\text{ns} + 4 \times 8\,\text{ns} = \boxed{82\,\text{ns}}.$$

Module Gatelevel Diagrams

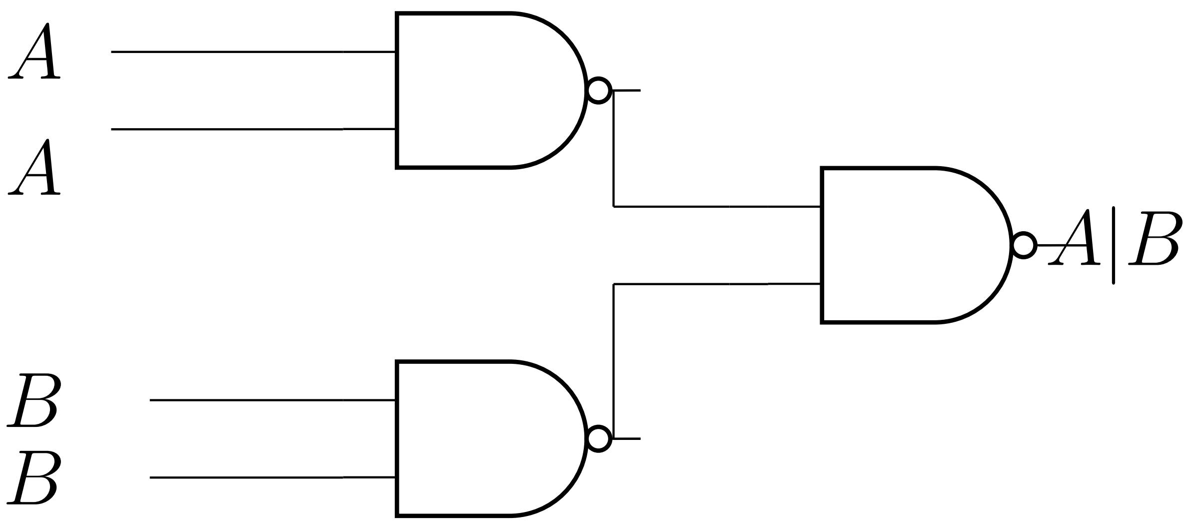
3.1 XOR



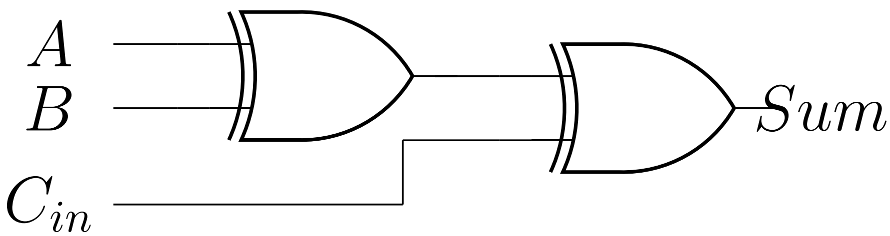
3.2 AND

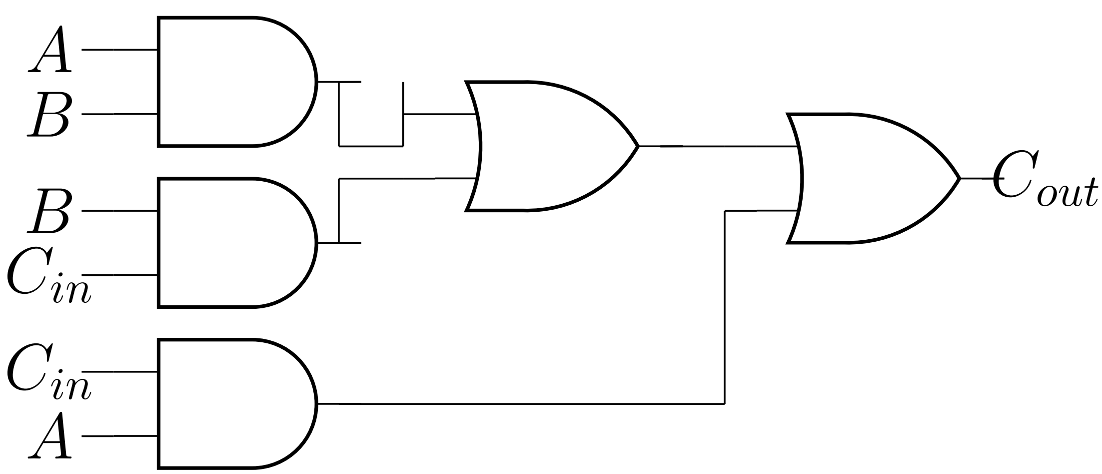


3.3 OR



3.4 Full Adder





For codes,figs etc refer to
`https://github.com/ArnavYadnopavit/DigitalSystemLabEE1501/tree/main/Assignments/Assignment2`

Thank You