Full Adder and 4-bit Binary Adders

Lab 6

11/1/22

Christopher Tark

Objective: Develop a full adder and 4-bit binary adders along with decreased propagation delay. Examine full adder results to verify for accuracy using Verilog for all experiments.

Design:

In binary the only values are one or zero. Binary numbers are manipulated for the computer to calculate figures more efficiently. A full adder is a combinational electrical circuit which adds binary numbers with a carry digit if needed to “carry” the value to the next position. For example, 1 or 1 = 10, in this case the two ones create a carry digit for second slot. The following is a 1-bit full adder:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | Cin | Sum | Cout/Carry |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Truth table for 1-bit full adder

Verilog Code:

module orginal (S,Cout,A,B,Cin);

input A,B,Cin;

output S,Cout;

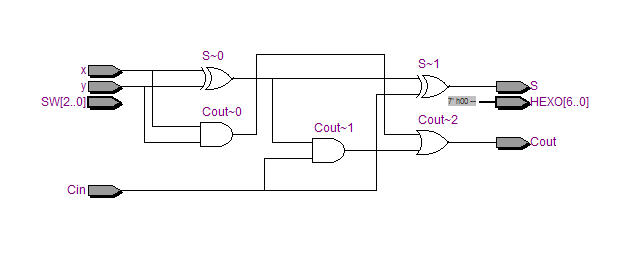
assign S = A^B^Cin;

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

endmodule

The one-bit adder is quite simple, I used three inputs and two output for and then I assign sum to equal the addition of the logical combination from the truth table. The code completely complied.

RTL Viewer for 1-bit full adder:



Augment 1-bit Full Adder:

We need inputs and outputs for the DE2-115 board to control the board. I supplemented to the port list switches or in syntax form SW and lights called HEX0. The switches are the new inputs and the lights are the outputs, therefore; commented out the previous signal and replace them with the new arrays (switches and lights). After, I assigned x, y, Cin, and Sum as wires to reconnect them to the new inputs and outputs. x is associated with SW[2], y is with SW[1], Cin is assign to SW[0], and sum is connected to the sum

module FullAdder(SW,HEX0);

input [2:0] SW; //input ports

output reg [6:0] HEX0; //output ports

wire x, y, Cin, Carry, Sum; //internal wires

assign {x, y, Cin} = SW[2:0]; //x, y, Cin values are from SW[2:0]

assign {Carry,Sum} = x+y+Cin;

always @(\*)

begin

if(Carry == 1'b0 & Sum == 1'b0)

HEX0 = 7'b1000000;

else if(Carry == 1'b0 & Sum == 1'b1)

HEX0 = 7'b1111001;

else if(Carry == 1'b1 & Sum == 1'b0)

HEX0 = 7'b0100100;

else if(Carry == 1'b1 & Sum == 1'b1)

HEX0 = 7'b0110000;

end

endmodule

4-bit Ripple Carry Adder:

A bit carry adder uses 4 full adders to calculate the sum of a binary number. The ripple carry adder calls the full adder to calculate the sum. The ripple Adders method sorts the input and output data from the arrays to create 4 linked full adders. Each full adder brings the carry/Cout digit through the adders to return the corresponding binary number.

module Full\_Adder(sum, carry,x,y,cin,);

input x,y,cin;

output sum, carry;

assign sum= x^y^cin;

assign carry=x&y| cin& (x^y);

endmodule

module rippleAdder(sum, Cout,A,B,Cin);

input [3:0] A,B;

input Cin;

output [3:0]sum;

output Cout;

wire w1, w2, w3;

Full\_Adder (sum[0], w1, A[0], B[0], Cin);

Full\_Adder (sum[1], w2, A[1], B[1], w1);

Full\_Adder (sum[2], w3, A[2], B[2], w2);

Full\_Adder (sum[3], Cout, A[3], B[3], w3);

endmodule

The theory is utilizing the basic full adder’s summation equations and placing values to produce the sum of two binary numbers. Small steps are joined to solve the greater goal.

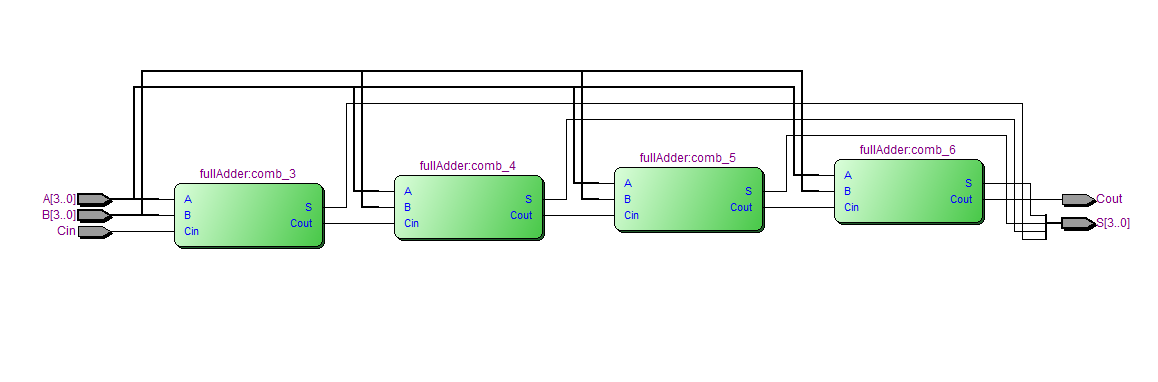
Functional Simulation of 4-bit RCA:

Graphical user interface, application, table

Description automatically generated

A file called Simulation Waveform editor needs to be created. Set the end time for the function and time intervals for each line. Nodes are added to the list or the port list. To test our outputs (Cout and Sum) we altenate the inputs (A, B, Cin) splitting them in halves for consistency. At the beginning of function, A, B, and Cin are 0. The sum of those three values is 0, we can see that Cout and Sum return just that. This continues to work through the whole timing diagram.

RTL viewer Ripple carry adder:



Augment 4-bit RCA:

Now we have a functional 4-bit ripple carry adder. I can program the logical circuit to calculate sums on a breadboard. The original port list is replaced with SW (switches) and LEDR/LEDG ( red and green lights). Subsequently, adding board commends to the program is identical to augmenting 1-bit full adder. Two groups of four switches represent the input for the calculations. Moreover, 9 red LED lights correspond to the binary input while the green lights displays the total sum.

module Full\_Adder(sum, carry,x,y,cin,);

input x,y,cin;

output sum, carry;

assign sum= x^y^cin;

assign carry=x&y| cin& (x^y);

endmodule

module RCA (SW, LEDR, LEDG);

input [8:0] SW; // 9 switches

output [8:0] LEDR; // 9 red LED lights to display all inputs

output[4:0] LEDG; // 5 green LED lights to display all outputs

wire [3:0] x, y, sum;

wire cin, carry;

assign x = SW[7:4];

assign y = SW[3:0];

assign cin = SW[8];

assign LEDR = {cin, x, y}; //Display inputs to LED green lights

assign LEDG = {carry, sum}; //Display outputs to LED red lights

Full\_Adder (sum[0], w1, x[0], y[0], cin);

Full\_Adder (sum[1], w2, x[1], y[1], w1);

Full\_Adder (sum[2], w3, x[2], y[2], w2);

Full\_Adder (sum[3], carry, x[3], y[3], w3);

Endmodule

4-bit carry look-ahead adder:

There a propagation delay for each full adder called in the ripple carry adder. The delay effects the return data so much that there is an alternative solution for solving summations in linear fashion. Inputs are calculated with the equation for C’s array. Due to not calling connected functions but efficiently solving the variables as the program retrieves data cuts out time delay.

module rippleAdder(A,B, sum,carry,cin);

// input [3:0] A,B;

// input Cin;

// output [3:0]sum;

// output Cout;

input [3:0] A,B;

input cin;

output[3:0]sum;

output carry;

wire[3:0]G, P;

wire [3:0]C;

assign G = A&B;

assign P = A|B;

assign C[0] = cin;

assign C[1] = G[0] | P[0] & C[0];

assign C[2] = G[1] | G[0] & P[1] | C[0] & P[0] & P[1];

assign C[3] = G[2] | G[1] & P[2] | G[0] & P[1] & P[2] + C[0] & P[0] & P[1] & P[2];

assign carry = G[3] | G[2] & P[3] | G[1] & P[2] & P[3] + G[0] & P[1] & P[2] & P[3] | C[0] & P[0] & P[1] & P[2] & P[3];

assign sum = A^B^C;

endmodule

Functional Simultion of 4-bit CLA:

A picture containing calendar

Description automatically generated

The window on Quartus runs a waveform editor. The timing diagram ends at 800ns, with a 50ns interval. New data is generated every 50ns for our input variables (A and C). A’s values start at 0 and increment by one for 800ns. B has random generated values. We can see that the sum of A and B return the correct corresponding values such as 0 + 8 = 8. This works with the carry as well.

Hardware:

n/a

Experiment:

RTL 1-bit FA:

Diagram

Description automatically generated

RTL viewer Ripple carry adder:

Diagram

Description automatically generated

Analysis:

The Full Adder is derived from a basic full adder truth table. Two equations form from 3 inputs and 2 puts to create an elegant tool for computers. This 1-bit Full adder is used 4 times in unison to produce a Ripple carry adder. That is what the RCA is 4 connected full adders. The ripper adder has a propagation delay due programming the full adders one step to complete for the next to continue. A CLA calculated the binary summation in linear time due to a responsive technique.

Conclusion:

In conclusion, I developed a 1-bit Full adder, a Ripple carry adder, and a 4-bit carry look-ahead adder. I tested the results with Quartus waveform editor and collected data.