***Introduction***

The lab consists of four major parts; designing a carry look adder (CLA), an inferred adder, writing test benches for both design as verification to the design, and writing the hardware validating software to be used on the Nexys board. The CLA design segments use a hierarchical approach where designs are built using smaller components and combined in the end at the highest level. The inferred adder takes on a more direct approach in which pre defined operators are used making the design less complex as Xilinx software handles all of the operations for the user. A singular test bench was written to validate the designs through simulations. The test bench can be applied to either the CLA or inferred adder. Lastly the designs were validated with a physical device which utilized LEDs, switches and the seven segment display.

The CLA design expected was to utilize a strict structure in which the design was to be broken down. The building blocks consists of half adders, glue logic, and a custom handling BCD splitter. This component was presented in lab but the design did not follow the design requirements. Missing parts such as the splitter were missing. The design presented was the same as the design expected which in result was able to achieve the proper stimuli. However the design presented used hierarchical design which utilized the one half adder instead of four half adders. The singular adder was utilized four times and summed at the end.

The parts listed below were not fully completed in lab but were finished afterword:

* Design 4-bit inferred adder
* Verify 4-bit inferred adder
* Verify both ALU and 4-bit inferred adders using switches, LEDs, and the seven segment display

***Design Methodology***

*Carry Look Ahead Adder (CLA):* CLA design is split into many modules. The first module consists of multiple half adders which took in inputs and and outputs were sent to the CLA generator. The sums were calculated using another module. The top level module was used as a binder to utilize other smaller modules. This ensures every module can be testing separately and or in large set. Figure one shows the design of the custom CLA adder implemented.

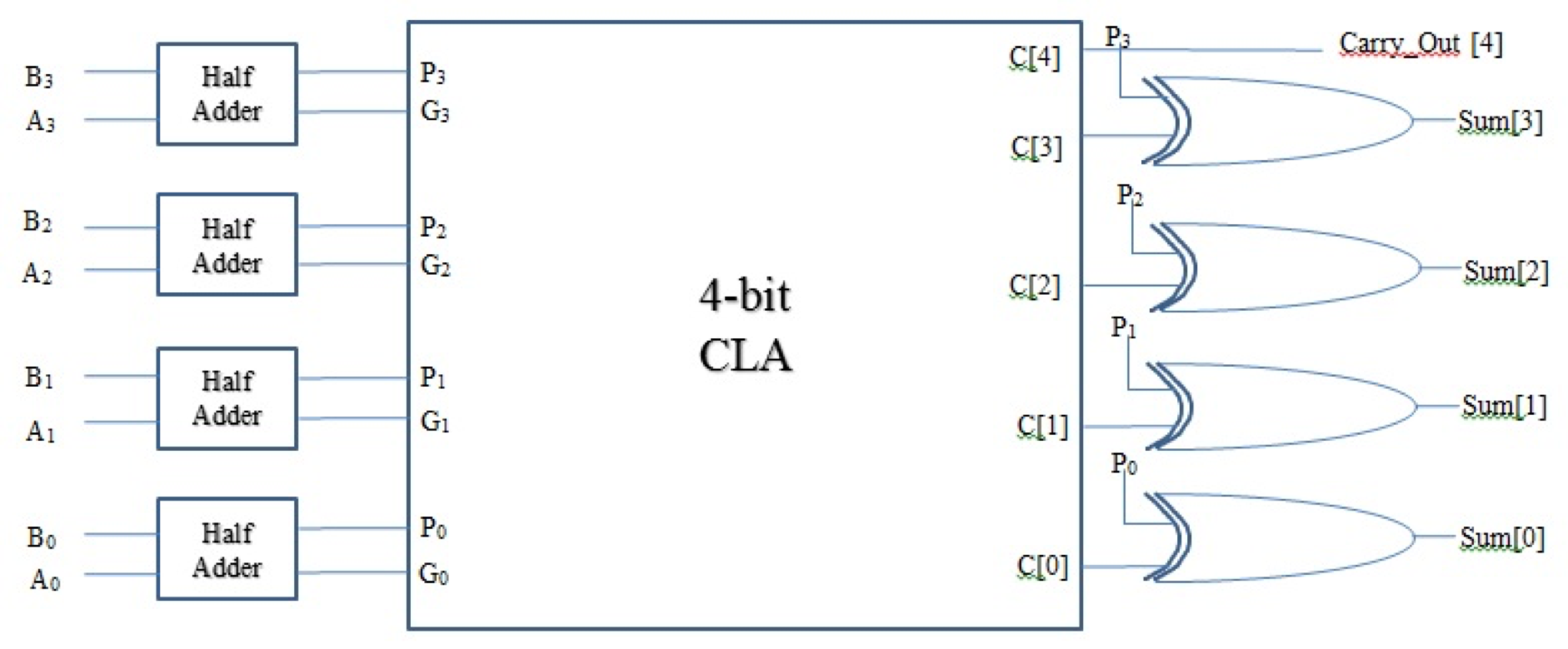


Figure 1. CLA Block Diagram

The basic starting point was a single half adder as shown below in figure two. Four of these adders created a set of quad adders which was designed in a separate submodule. All adders were used to perform a summing operation which would result in a carry out output. This resulted in a carry-in input for the other adders within the submodule. Spawning a new sub-module to handle the carry-in and carry-out bits. The finalized output sum was calculated by a multiple XORs and AND’ed carry-in input when needed. Below is a truth table which represents a full adder which is used for validation explained in the next section.

Table 1: Full Adder Truth Table

| carry-ini | ai ⊕ bi | ai & bi | si= ai+bi+ci | carry-ini+1 |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 (ai⊕bi⊕ci) | 0 ((ai⊕bi)ci+ ai & bi) |
| 0 | 1 | 0 | 1 (ai⊕bi⊕ci) | 0 ((ai⊕bi)ci+ ai & bi) |
| 0 | 1 | 0 | 1 (ai⊕bi⊕ci) | 0 ((ai⊕bi)ci+ ai & bi) |
| 0 | 0 | 1 | 0 (ai⊕bi⊕ci) | 1 ((ai⊕bi)ci+ ai & bi) |
| 1 | 0 | 0 | 1 (ai⊕bi⊕ci) | 0 ((ai⊕bi)ci+ ai & bi) |
| 1 | 1 | 0 | 0 (ai⊕bi⊕ci) | 1 ((ai⊕bi)ci+ ai & bi) |
| 1 | 1 | 0 | 0 (ai⊕bi⊕ci) | 1 ((ai⊕bi)ci+ ai & bi) |
| 1 | 0 | 1 | 1 (ai⊕bi⊕ci) | 1 ((ai⊕bi)ci+ ai & bi) |

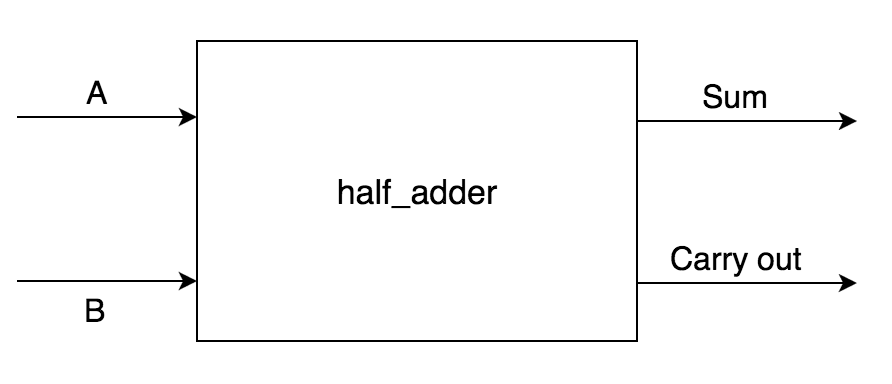


Figure 2. Singular Half Adder Block Diagram

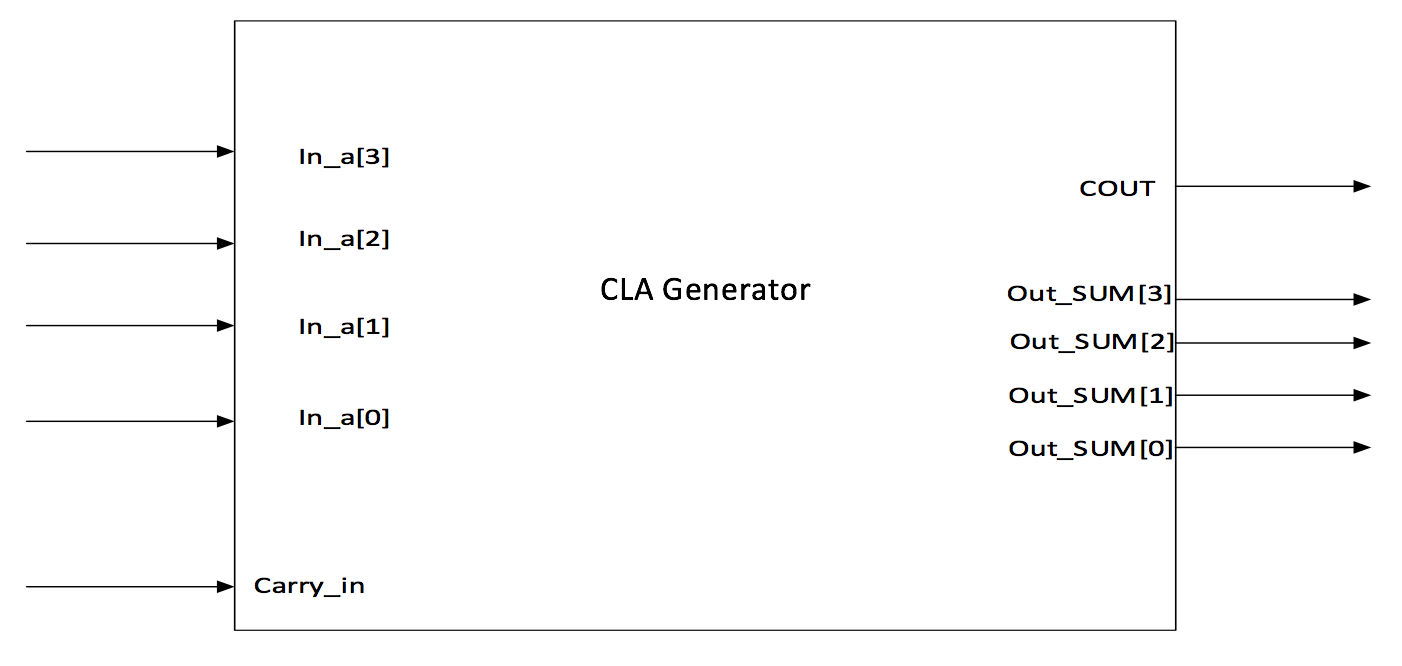


Figure 3. Carry Look-Ahead Generator block diagram

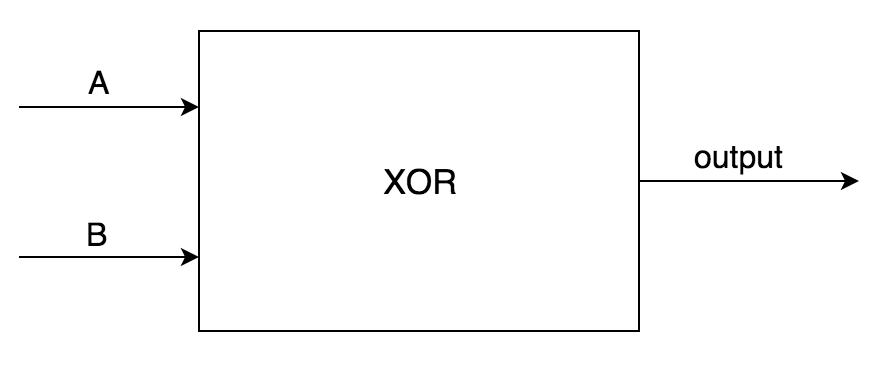


Figure 4. XOR block diagram

The inferred adder performs the same function as the custom CLA adder. However instead of defining modules to perform the summing operation the built in addition operator was utilized. This design is much more abstract and but results in the same output. The design for this adder is much less complex and easier to implement. The logical flow is coherent to the custom CLA in which the half adders are summed to create a full adder. Figure three shows the block diagram for the inferred adder.

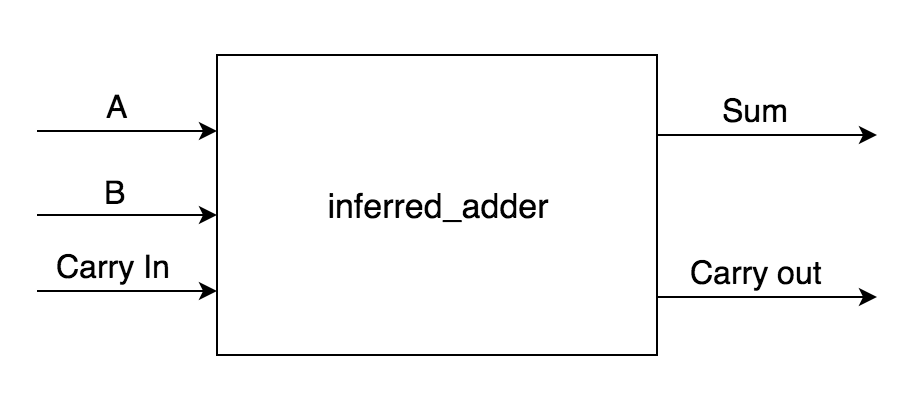


Figure 5. Inferred Adder Block Diagram

***Simulation Test Plan***

A single self checking testbench stimuli was used to validate both adders. Due to the logic and operational characteristics being the same for both adders the test bench implemented loops, if statements, and comparison logic to generate multiple stimuli. The testbench defined its 4-bit inputs plus a carry-in and expected a proper output plus a carry-out. The first section of the testbench sets the carry-in as 0 and goes through all possible inputs. The second sections sets the carry-in to 1 and loops over all inputs. Within the loops if statements check to see if the sum is equivalent to the output. For a successful testbench run for both the CLA and inferred adder the console output is shown in figure four and figure five. Additional waveforms can be found in the appendix.

Figure 4. Console output for Adder

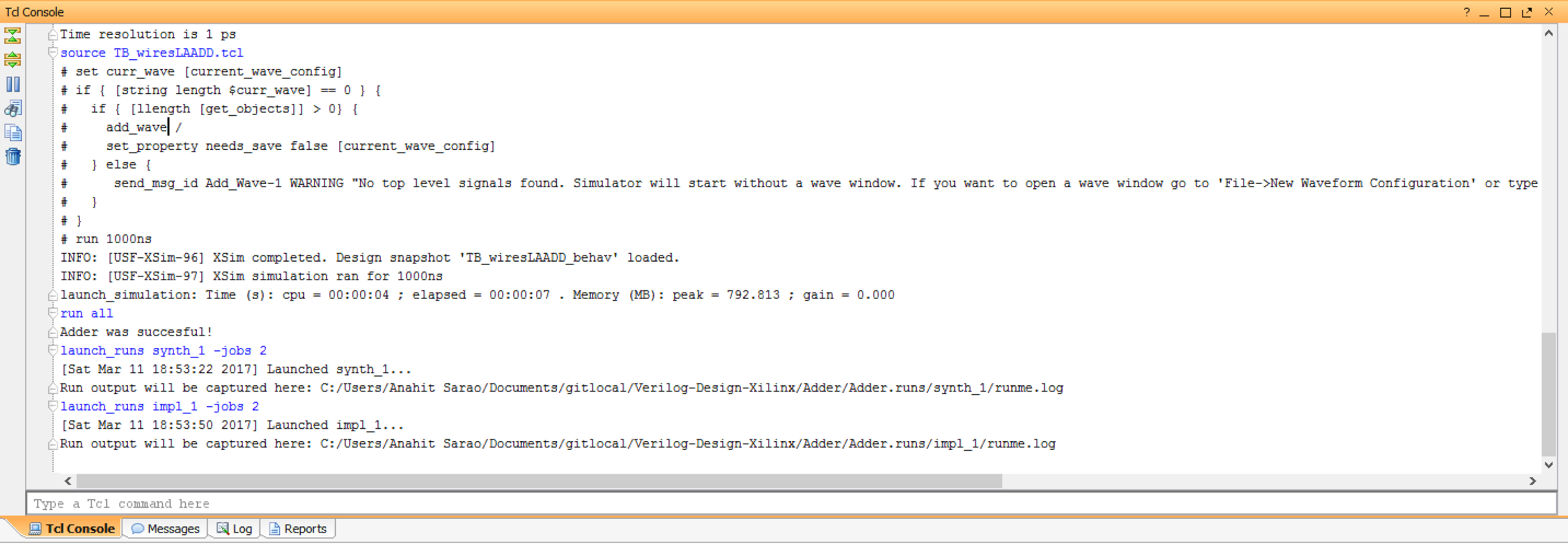
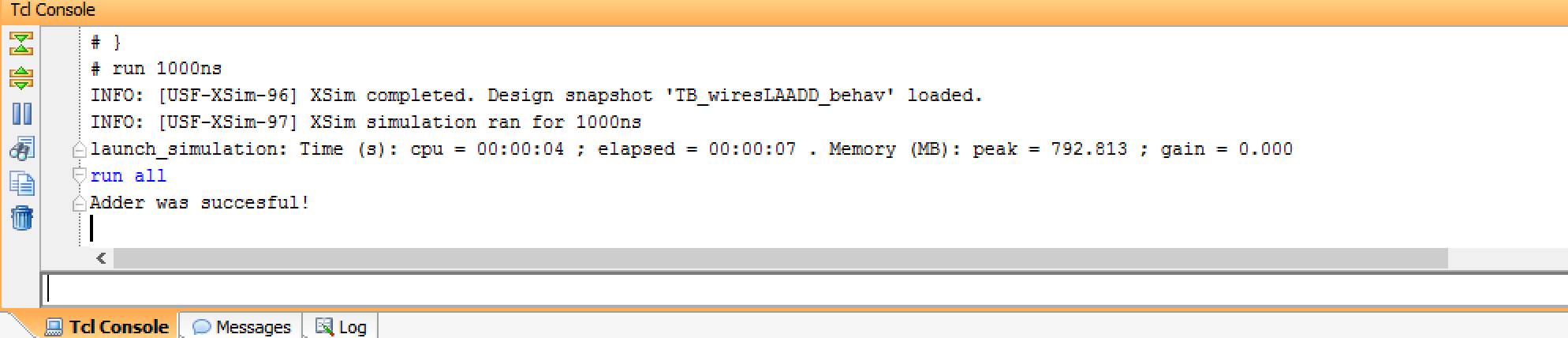


Figure 5. Console output for Adder



***FPGA Test Plan***

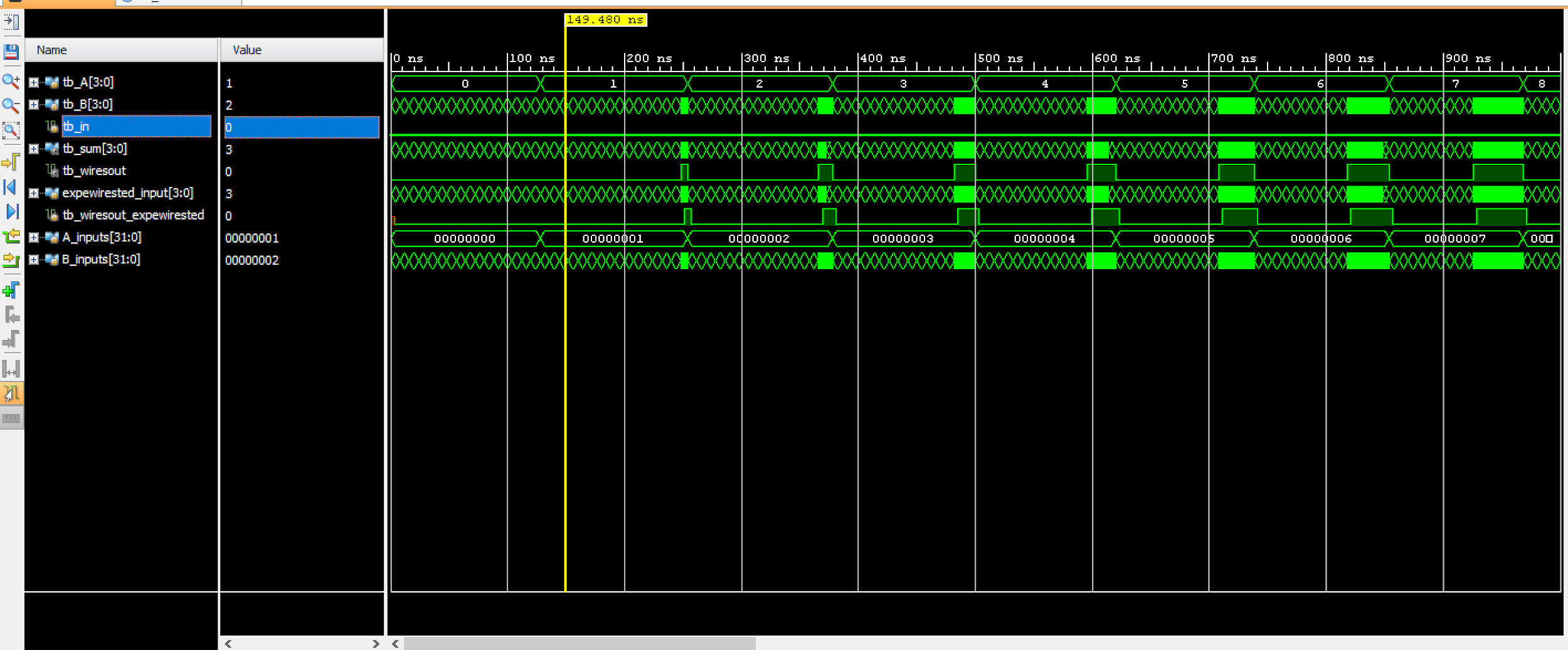
To attempt to validate the board, three new modules were added to be able to use the seven segment display. The XDC file was set to use the switches, RGB LED and seven segment display. However we were unable to get this part to work as the bitstream was failing.

***Conclusion***

In conclusion, our carry look ahead adder performed as expected, however, it did not meet the requirements for a hierarchical design approach. We have since rectified this and properly designed the lab reflecting the modular design. In addition to fixing the carry look ahead design, we designed and successfully tested the inferred adder. We were unable to successfully test the FPGA board and could not verify the complete functionality of the board.

***Appendix***

Figure x. Additional Waveforms for Stimuli



***Source Code***

Files for Presented Design

*CLA\_ADD.v*

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// wiresompany:

// Engineer:

//

// wiresreate Date: 03/08/2017 04:26:38 pM

// Design Name:

// Module Name: wiresLA\_ADD

// projewirest Name:

// Target Deviwireses:

// Tool Versions:

// Deswiresription:

//

// Dependenwiresies:

//

// Revision:

// Revision 0.01 - File wiresreated

// Additional wiresomments:

//

//////////////////////////////////////////////////////////////////////////////////

module half\_adder(A,B,p,g);

input A;

input B;

output p; //and p=AB

output g;//XOR g=A^B

assign p=A^B;

assign g=A&B;

endmodule

module half\_adder2(p,g,wiresin,Sum,wiresout);

input [3:0] p;

input [3:0] g;

input wiresin;

output [3:0] Sum;

output wiresout;

wire [3:0] wires;

assign wires[0]=wiresin;

assign wires[1] = g[0] | (p[0] & wires[0]);

assign wires[2] = g[1] | (p[1] & g[0]) | (p[1] & p[0] & wires[0]);

assign wires[3] = g[2] | (p[2] & g[1]) | (p[2] & p[1] & g[0]) | (p[2] & p[1] & p[0] & wires[0]);

assign wiresout = g[3] | (p[3] & g[2]) | (p[3] & p[2] & g[1]) | (p[3] & p[2] & p[1] & g[0]) |(p[3] & p[2] & p[1] & p[0] & wires[0]);

assign wires[4] = wiresout;

assign Sum =p^wires; //sum all wires i hope

endmodule

module top(A,B,Sum,wiresout,wiresin);

input [3:0] A;

input [3:0] B;

input wiresin;

output wiresout;

output [3:0] Sum;

wire [3:0] p;

wire [3:0] g;

half\_adder DUT1(.p(p[0]),.g(g[0]),.A(A[0]),.B(B[0]));

half\_adder DUT2(.p(p[1]),.g(g[1]),.A(A[1]),.B(B[1]));

half\_adder DUT3(.p(p[2]),.g(g[2]),.A(A[2]),.B(B[2]));

half\_adder DUT4(.p(p[3]),.g(g[3]),.A(A[3]),.B(B[3]));

// wiresall other modules to sum them up

half\_adder2 DUT5(.p(p),.g(g),.Sum(Sum),.wiresout(wiresout),.wiresin(wiresin));

endmodule

*TB\_CLAADD.v*

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// wiresompany:

// Engineer:

//

// wiresreate Date: 03/08/2017 04:34:34 pM

// Design Name:

// Module Name: TB\_wiresLAADD

// projewirest Name:

// Target Deviwireses:

// Tool Versions:

// Deswiresription:

//

// Dependenwiresies:

//

// Revision:

// Revision 0.01 - File wiresreated

// Additional wiresomments:

//

//////////////////////////////////////////////////////////////////////////////////

module TB\_wiresLAADD;

reg [3:0]tb\_A;

reg [3:0]tb\_B;

reg tb\_in;

wire [3:0] tb\_sum;

wire tb\_wiresout;

reg [3:0] expewirested\_input;

reg tb\_wiresout\_expewirested;

integer A\_inputs;

integer B\_inputs;

top DUT(.A(tb\_A), .B(tb\_B),.Sum(tb\_sum),.wiresout(tb\_wiresout),.wiresin(tb\_in));

//inferred DUT2 (.A(tb\_A), .B(tb\_B), .Sum(tb\_sum), .wiresout(tb\_wiresout),.wiresin(tb\_in));

initial

begin

tb\_in=0;

for(A\_inputs=0;A\_inputs<16;A\_inputs=A\_inputs+1)

begin

tb\_A=A\_inputs;

for(B\_inputs=0;B\_inputs<16;B\_inputs=B\_inputs+1)

begin

tb\_B=B\_inputs;

expewirested\_input=tb\_A+tb\_B;

#3;

if(tb\_A+tb\_B> 15)

begin

tb\_wiresout\_expewirested=1;

#3;

end

else

begin

tb\_wiresout\_expewirested=0;

#5;

end

if((tb\_sum!=expewirested\_input)||(tb\_wiresout\_expewirested!=tb\_wiresout))

begin

$display("Error broken");

$stop;

end

end

end

tb\_in=1;

for(A\_inputs=0;A\_inputs<16;A\_inputs=A\_inputs+1)

begin

tb\_A=A\_inputs;

for(B\_inputs=0;B\_inputs<16;B\_inputs=B\_inputs+1)

begin

tb\_B=B\_inputs;

expewirested\_input=((tb\_A+tb\_B)+ 4'b0001);

if(tb\_A+tb\_B+1>15)

begin

tb\_wiresout\_expewirested=1;

#5;

end

else

begin

tb\_wiresout\_expewirested=0;

#5;

end

if((tb\_sum!=expewirested\_input)||(tb\_wiresout\_expewirested!=tb\_wiresout))

begin

$display("Error broken");

$stop;

end

end

end

$display("Adder was succesful!");

$stop;

end

endmodule

Files added for Corrected Design

*generater.v*

module GENERATOR(

input [3:0] P, G, input Cin,

output reg [3:0] C, output reg Cout

);

//Ci = Gi | Pi & Ci;

always @(P, Q, Cin, C) begin

C[0] = Cin;

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

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

C[3] = G[2] | P[2] & C[2];

Cout = G[3] | P[3] & C[3];

end

endmodule

module SUMMER(

input [3:0] P, C,

output reg [3:0] Sum

);

xor U0(Sum[0], P[0], C[0]);

xor U1(Sum[1], P[1], C[1]);

xor U2(Sum[2], P[2], C[2]);

xor U3(Sum[3], P[3], C[3]);

endmodule

*CLA\_ADD.v*

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// wiresompany:

// Engineer:

//

// wiresreate Date: 03/08/2017 04:26:38 pM

// Design Name:

// Module Name: wiresLA\_ADD

// projewirest Name:

// Target Deviwireses:

// Tool Versions:

// Deswiresription:

//

// Dependenwiresies:

//

// Revision:

// Revision 0.01 - File wiresreated

// Additional wiresomments:

//

//////////////////////////////////////////////////////////////////////////////////

module half\_adder(A,B,p,g);

input A;

input B;

output p; //and p=AB

output g;//XOR g=A^B

assign p=A^B;

assign g=A&B;

endmodule

module half\_adder2(p,g,wiresin,Sum,wiresout);

input [3:0] p;

input [3:0] g;

input wiresin;

output [3:0] Sum;

output wiresout;

wire [3:0] wires;

assign wires[0]=wiresin;

assign wires[1] = g[0] | (p[0] & wires[0]);

assign wires[2] = g[1] | (p[1] & g[0]) | (p[1] & p[0] & wires[0]);

assign wires[3] = g[2] | (p[2] & g[1]) | (p[2] & p[1] & g[0]) | (p[2] & p[1] & p[0] & wires[0]);

assign wiresout = g[3] | (p[3] & g[2]) | (p[3] & p[2] & g[1]) | (p[3] & p[2] & p[1] & g[0]) |(p[3] & p[2] & p[1] & p[0] & wires[0]);

assign wires[4] = wiresout;

assign Sum =p^wires; //sum all wires i hope

endmodule

GENERATOR U1(

.P(P\_wire), .G(G\_wire), .Cin(Cin\_top),

.C(C\_wire), .wiresout(wiresout)

);

SUMMER U2(

.P(P\_wire), .C(C\_wire),

.Sum(Sun)

);

module top(A,B,Sum,wiresout,wiresin);

input [3:0] A;

input [3:0] B;

input wiresin;

output wiresout;

output [3:0] Sum;

wire [3:0] p;

wire [3:0] g;

half\_adder DUT1(.p(p[0]),.g(g[0]),.A(A[0]),.B(B[0]));

half\_adder DUT2(.p(p[1]),.g(g[1]),.A(A[1]),.B(B[1]));

half\_adder DUT3(.p(p[2]),.g(g[2]),.A(A[2]),.B(B[2]));

half\_adder DUT4(.p(p[3]),.g(g[3]),.A(A[3]),.B(B[3]));

// wiresall other modules to sum them up

half\_adder2 DUT5(.p(p),.g(g),.Sum(Sum),.wiresout(wiresout),.wiresin(wiresin));

*inferred.v*

module inferred(A,B,wiresin, wiresout, Sum);

input [3:0] A;

input [3:0] B;

input Carry\_in;

output wiresout;

output [3:0] Sum;

assign {wiresout,Sum}=A+B+wiresin;

endmodule