UNIVERSITY OF SASKATCHEWAN

# Lab 1 : Carry Lookahead Adder

CME433-01

Addi Amaya

Caa746

11255790

October 8th, 2021

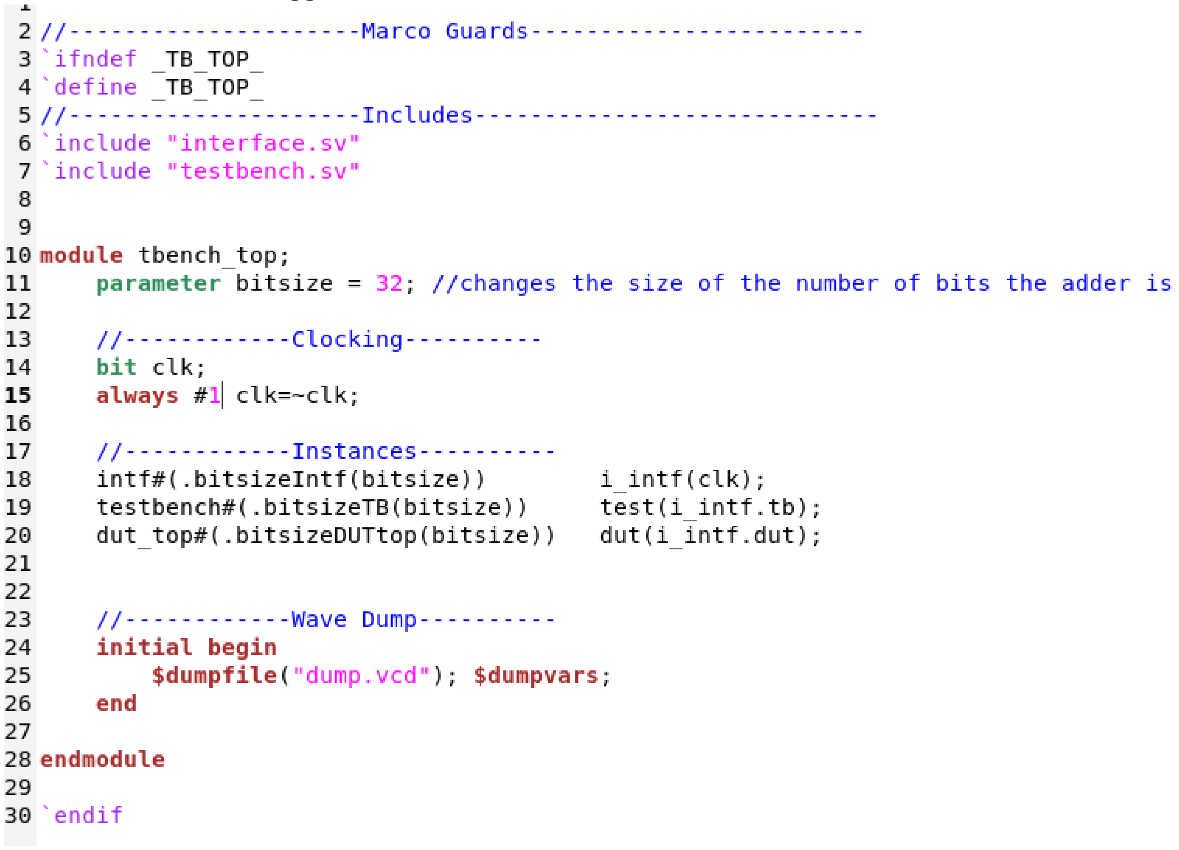
## Contents

## Ripple-Carry Adder

In this section I am going to outline the software structure I used to test the DUT and the outcome of quartus. All files are in the “RippleCarryAdder” folder within “caa746_lab2” folder

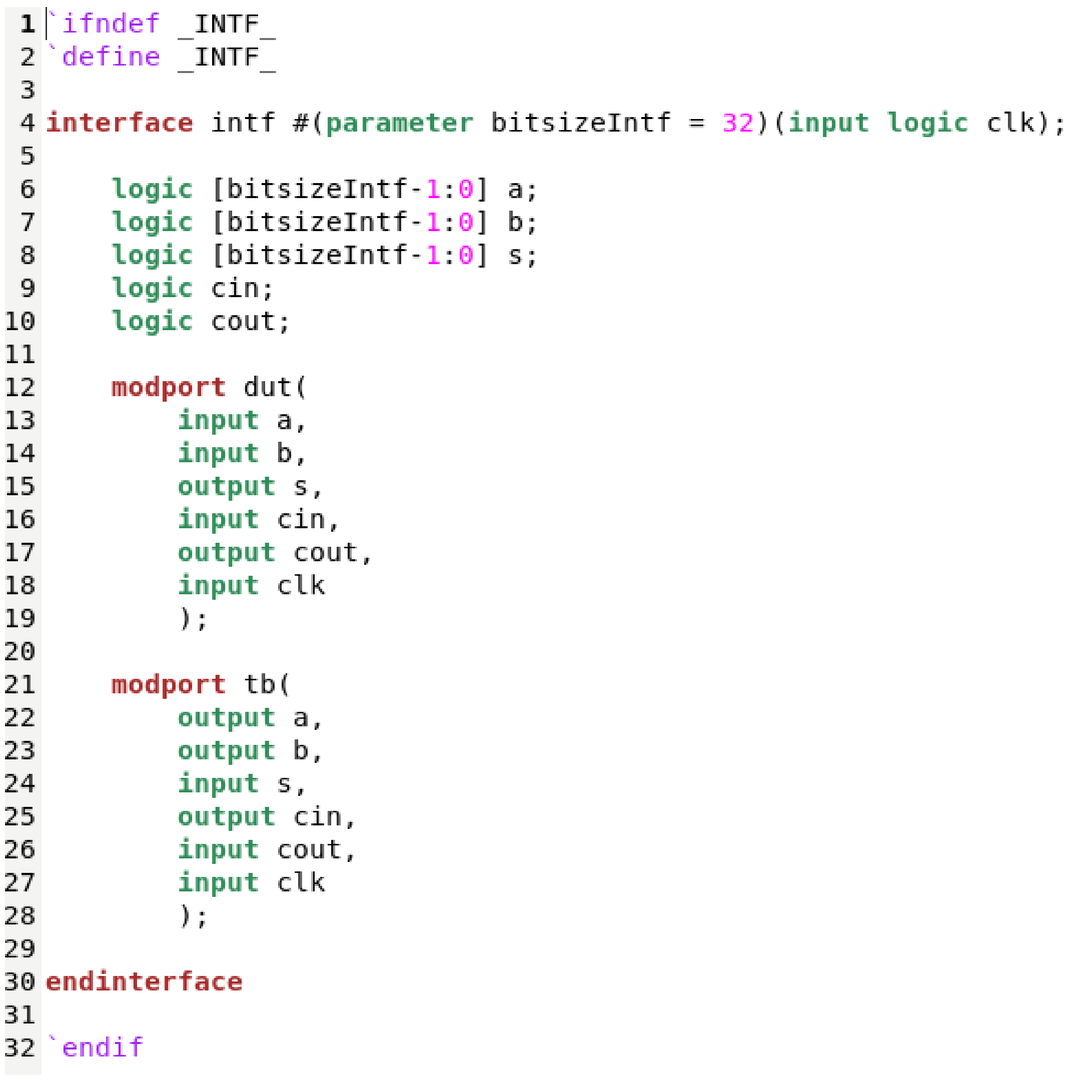
## Testbench structure

The top-level module is called tbench_top.sv. This file controls the bit size of the adder, it is where the main clk is toggles, its where all other files are instantiated.



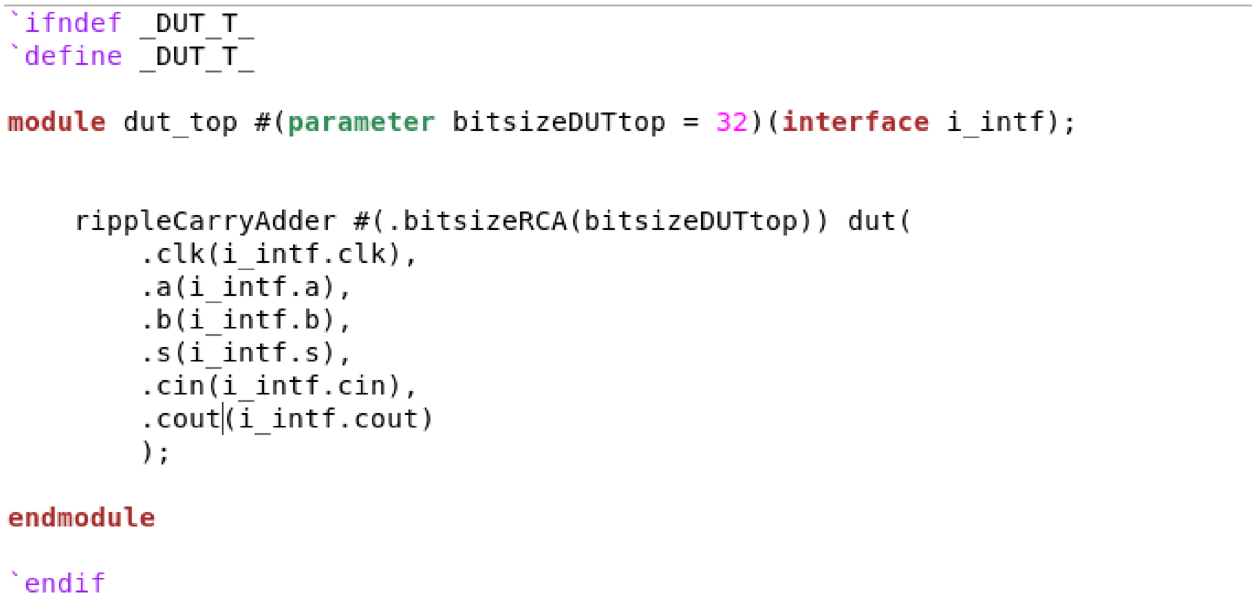
```
//--------------------Marco Guards-----------------------
`ifndef _TB_TOP_
`define _TB_TOP_
//--------------------Includes---------------------------
`include "interface.sv"
`include "testbench.sv"


module tbench_top;
    parameter bitsize = 32; //changes the size of the number of bits the adder is

    //------------Clocking----------
    bit clk;
    always #1 clk=~clk;

    //------------Instances----------
    intf#(.bitsizeIntf(bitsize))        i_intf(clk);
    testbench#(.bitsizeTB(bitsize))     test(i_intf.tb);
    dut_top#(.bitsizeDUTtop(bitsize))   dut(i_intf.dut);


    //------------Wave Dump----------
    initial begin
        $dumpfile("dump.vcd"); $dumpvars;
    end

endmodule

`endif
```

The interface takes a parameter that changes the bitsize of “a” and “b”, the two inputs for the dut.



```
`ifndef _INTF_
`define _INTF_

interface intf #(parameter bitsizeIntf = 32)(input logic clk);

    logic [bitsizeIntf-1:0] a;
    logic [bitsizeIntf-1:0] b;
    logic [bitsizeIntf-1:0] s;
    logic cin;
    logic cout;

    modport dut(
        input a,
        input b,
        output s,
        input cin,
        output cout,
        input clk
        );

    modport tb(
        output a,
        output b,
        input s,
        output cin,
        input cout,
        input clk
        );

endinterface

`endif
```

The dut_top is how the dut will integrate with the interface mentioned above

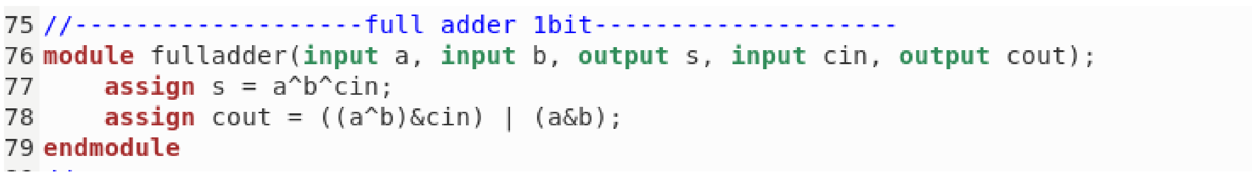

```
`ifndef _DUT_T_
`define _DUT_T_

module dut_top #(parameter bitsizeDUTtop = 32)(interface i_intf);


    rippleCarryAdder #(.bitsizeRCA(bitsizeDUTtop)) dut(
        .clk(i_intf.clk),
        .a(i_intf.a),
        .b(i_intf.b),
        .s(i_intf.s),
        .cin(i_intf.cin),
        .cout(i_intf.cout)
        );

endmodule

`endif
```

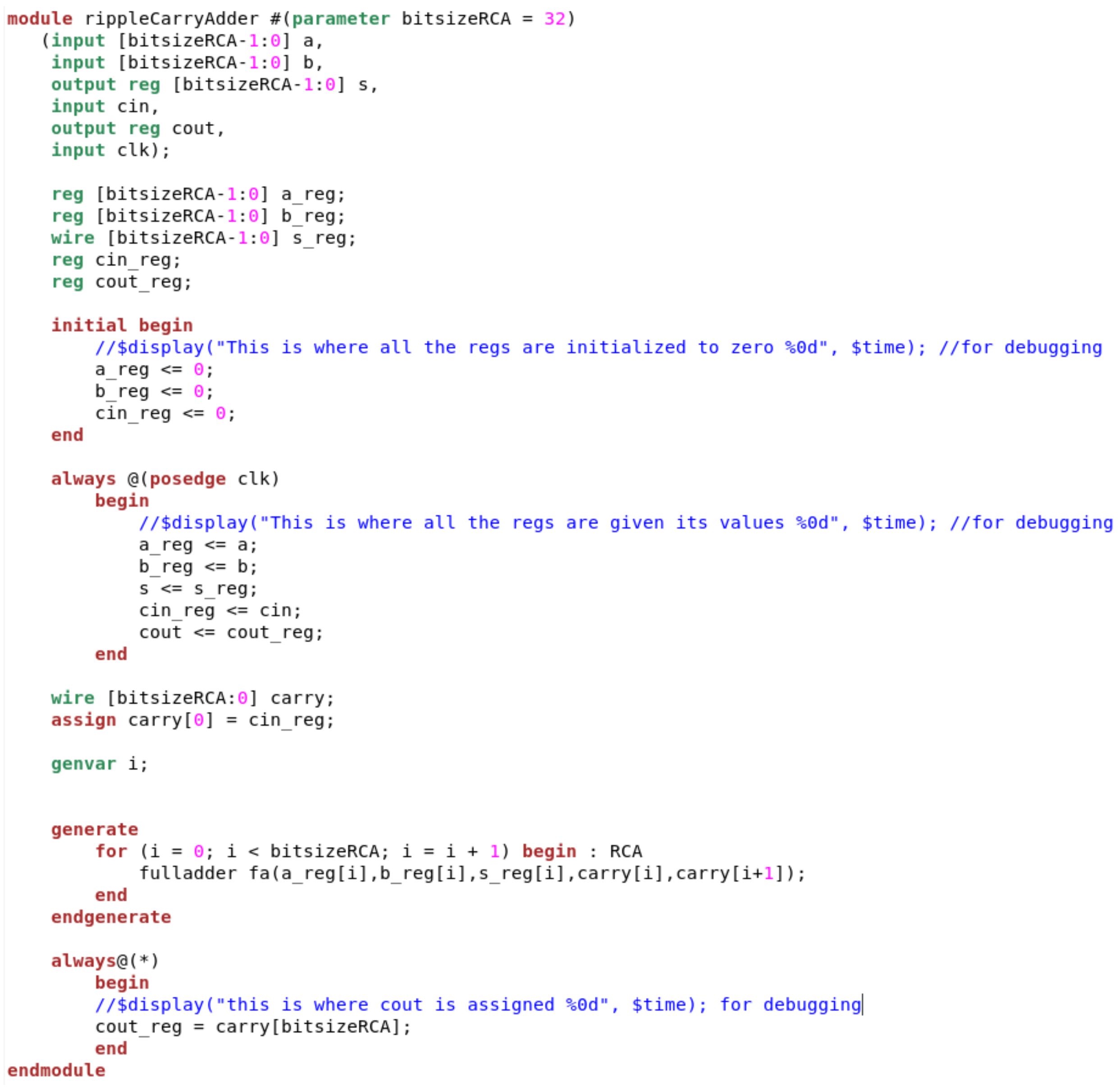
The testbench is also instantiated in the top but they will be broken down in the next section

## Ripple Carry Adder Verilog File

Since the testbench hierarchy is not expected a latched version of the dut. I needed to make two different version of the dut. One with registers and one without. The reason I had to do this is because I ran into a problem where if I got the Verilog file working on the testbench it would not produce a timing analyzing section. And when I got the timing analyzer working on quartus it broke my testbench. Although one version is latched and one isn't, the logic and how the values are handled are the exact same.

The rippleCarryadder works as the following: First I made a 1 bit full adder and in a generate wrapper I looped the instance of that full adder depending on the size of the modules parameter (which is controlled by tbench_top) as mentioned before


```
//-------------------full adder 1bit-------------------
module fulladder(input a, input b, output s, input cin, output cout);
    assign s = a^b^cin;
    assign cout = ((a^b)&cin) | (a&b);
endmodule
```



```verilog
module rippleCarryAdder #(parameter bitsizeRCA = 32)
   (input [bitsizeRCA-1:0] a,
    input [bitsizeRCA-1:0] b,
    output reg [bitsizeRCA-1:0] s,
    input cin,
    output reg cout,
    input clk);

    reg [bitsizeRCA-1:0] a_reg;
    reg [bitsizeRCA-1:0] b_reg;
    wire [bitsizeRCA-1:0] s_reg;
    reg cin_reg;
    reg cout_reg;

    initial begin
        //$display("This is where all the regs are initialized to zero %0d", $time); //for debugging
        a_reg <= 0;
        b_reg <= 0;
        cin_reg <= 0;
    end

    always @(posedge clk)
        begin
            //$display("This is where all the regs are given its values %0d", $time); //for debugging
            a_reg <= a;
            b_reg <= b;
            s <= s_reg;
            cin_reg <= cin;
            cout <= cout_reg;
        end

    wire [bitsizeRCA:0] carry;
    assign carry[0] = cin_reg;

    genvar i;


    generate
        for (i = 0; i < bitsizeRCA; i = i + 1) begin : RCA
            fulladder fa(a_reg[i],b_reg[i],s_reg[i],carry[i],carry[i+1]);
        end
    endgenerate

    always@(*)
        begin
        //$display("this is where cout is assigned %0d", $time); for debugging
        cout_reg = carry[bitsizeRCA];
        end
endmodule
```

Now for the version of the Verilog file that will be included in the testbench which uses the exact same full adder

```
module rippleCarryAdder #(parameter bitsizeRCA = 32)
   (input [bitsizeRCA-1:0] a,
    input [bitsizeRCA-1:0] b,
    output [bitsizeRCA-1:0] s,
    input cin,
    output cout,
    input clk);

    wire [bitsizeRCA:0] carry;
    assign carry[0] = cin;

    genvar i;


    generate
        for (i = 0; i < bitsizeRCA; i = i + 1) begin : RCA
            fulladder fa(a[i],b[i],s[i],carry[i],carry[i+1]);
        end
    endgenerate

    assign cout_reg = carry[bitsizeRCA];

endmodule
```

Please note that the only difference between this and the other version is the registered I/O

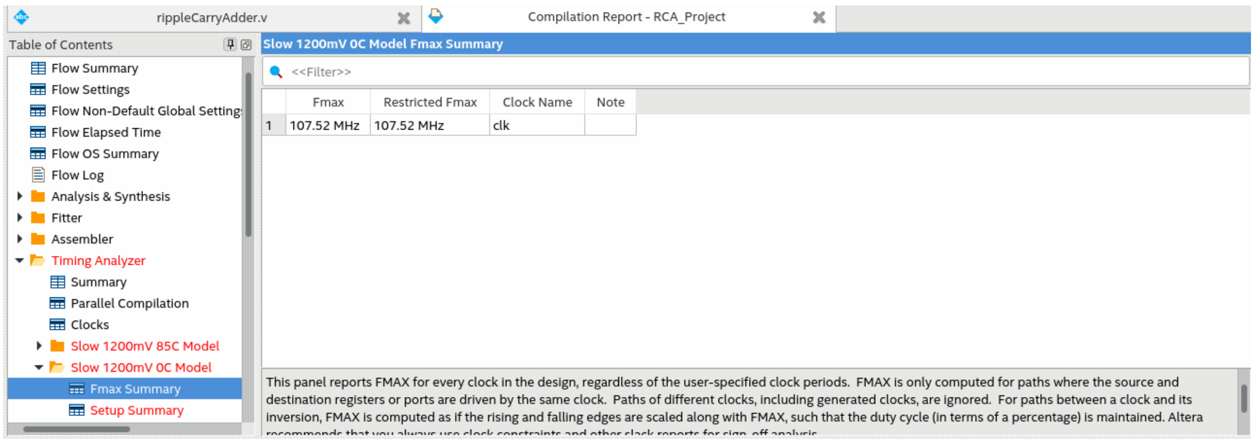
## Testbench Verilog File

The main method of testing is using random variables with a class randomizer which can be found in lines 8 to 12. The file then go on to check if the output of the dut is equal to adding the two variables (a and b) together and increments a success or failed counter

```
module testbench #(parameter bitsizeTB = 32) (intf i_intf);
    reg [31:0] failedcases, successCases;
    //----------Class for Randomizer---------
    class Packet #(int N = 32);
        rand bit [N  - 1:0] a_rand;
        rand bit [N  - 1:0] b_rand;
        constraint a_b_max {}
    endclass
    //-----------Constructor---------------
    Packet #(.N(bitsizeTB)) pkt;
    //------------Initializer---------------
    initial begin
        i_intf.a <= 32'd0;
        i_intf.b <= 32'd0;
        i_intf.cin <= 1'b0;
        failedcases <= 32'd0;
        successCases <= 32'd0;
        pkt = new();
        $display("--------------------------------------------------");
        $display("[Testbench]: Start of testcase(s) at %0d", $time);
        end
    //-------------Randomizer----------------
    always@(posedge i_intf.clk)
        begin
        pkt.randomize();
        //$display("Randomized a and b at time %0d", $time); //for debugging
        i_intf.a <= pkt.a_rand;
        i_intf.b <= pkt.b_rand;
        end
    //-----------Checking output--------------
    always@(posedge i_intf.clk)
        begin
        $display("%0d + %0d = %0d | Expected: %0d", i_intf.a, i_intf.b, i_intf.s, i_intf.a + i_intf.b); // for debugging
        if ((i_intf.a + i_intf.b) != i_intf.s)
            failedcases = failedcases + 1;
        else if ((i_intf.a + i_intf.b) == i_intf.s)
            successCases = successCases + 1;
        end
    //---------------------Termination Section----------------------
    initial begin
        #10000 $finish; //set how long you want to test to run for
    end

    final begin
        $display("[Testbench]: End of testcases(s) at %0d", $time);
        $display("Failed cases: %0d | Success cases: %0d", failedcases, successCases);
        $display("---------------------------------------------------------");

    end

endmodule
```

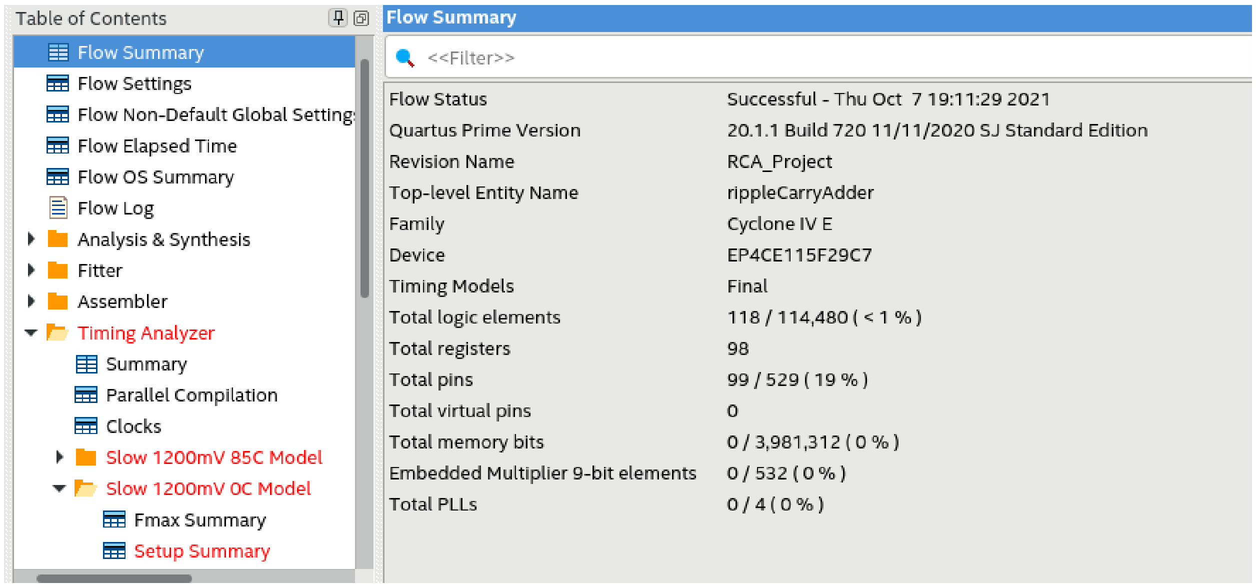
## Quartus Output for Ripple Carry Adder

After running the compatible version of the adder through Quartus and looking under the "Slow 1200mV 0C Model" the FMAX was 107.52Mhz.



|  | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 107.52 MHz | 107.52 MHz | clk |  |

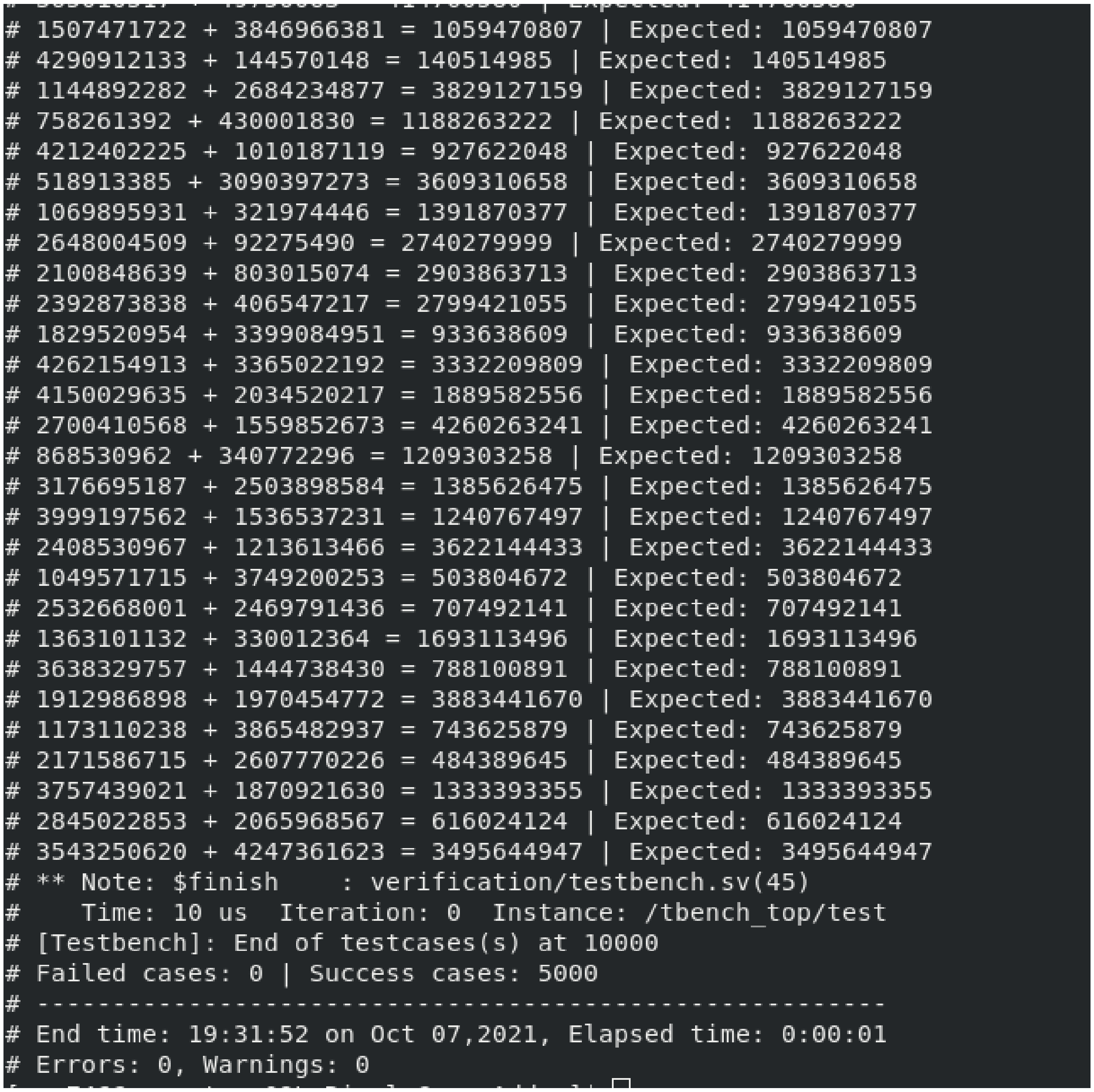
This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of a percentage) is maintained. Altera recommends that you always use clock constraints and other slack reports for sign-off analysis.



| | |
|---|---|
| Flow Status | Successful - Thu Oct 7 19:11:29 2021 |
| Quartus Prime Version | 20.1.1 Build 720 11/11/2020 SJ Standard Edition |
| Revision Name | RCA_Project |
| Top-level Entity Name | rippleCarryAdder |
| Family | Cyclone IV E |
| Device | EP4CE115F29C7 |
| Timing Models | Final |
| Total logic elements | 118 / 114,480 ( < 1 % ) |
| Total registers | 98 |
| Total pins | 99 / 529 ( 19 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 3,981,312 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 0 / 532 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

### Testbench Output for the Adder

After running >1000 cases it was determined that the adder did work.



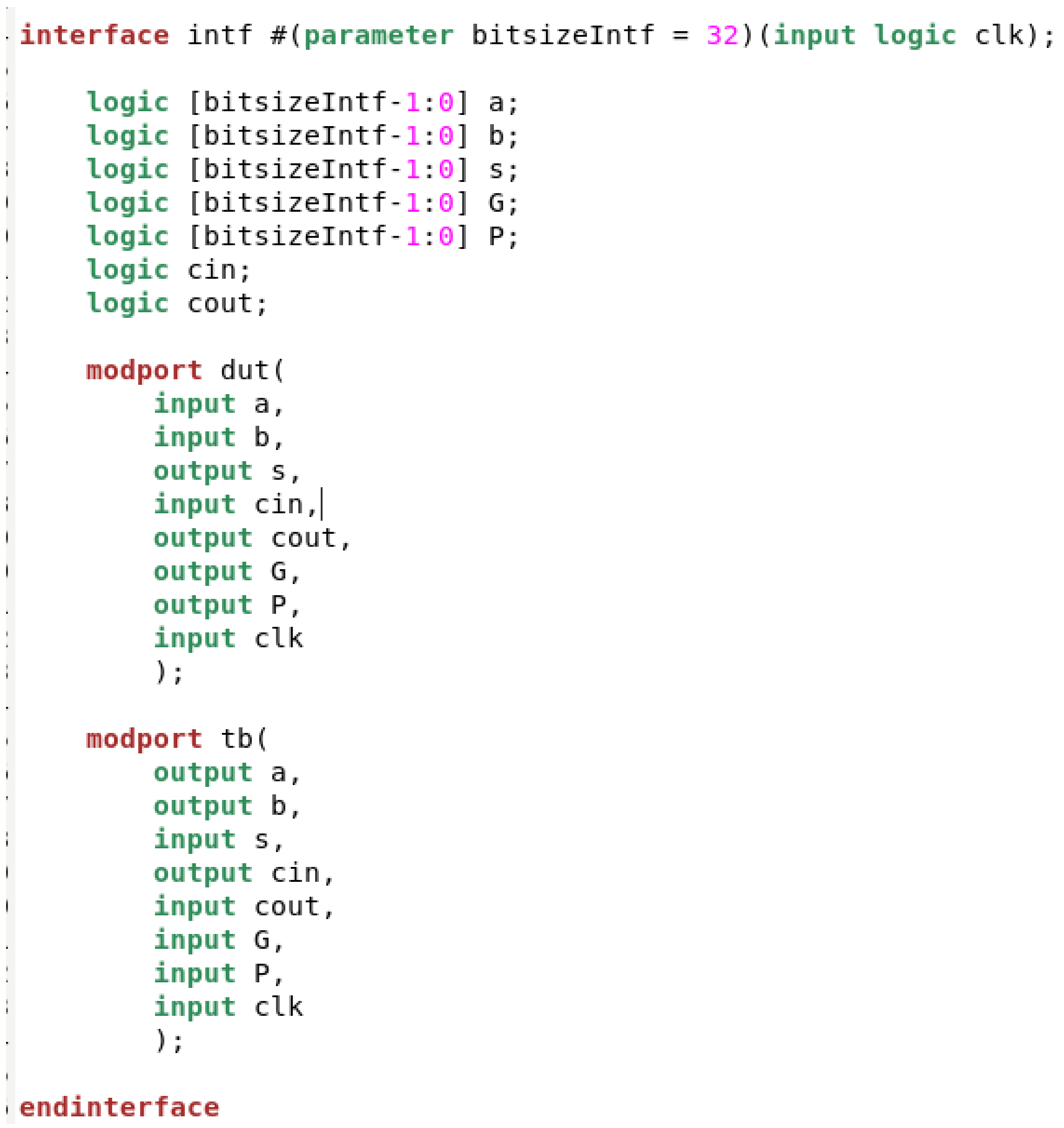
```
# 1507471722 + 3846966381 = 1059470807 | Expected: 1059470807
# 4290912133 + 144570148 = 140514985 | Expected: 140514985
# 1144892282 + 2684234877 = 3829127159 | Expected: 3829127159
# 758261392 + 430001830 = 1188263222 | Expected: 1188263222
# 4212402225 + 1010187119 = 927622048 | Expected: 927622048
# 518913385 + 3090397273 = 3609310658 | Expected: 3609310658
# 1069895931 + 321974446 = 1391870377 | Expected: 1391870377
# 2648004509 + 92275490 = 2740279999 | Expected: 2740279999
# 2100848639 + 803015074 = 2903863713 | Expected: 2903863713
# 2392873838 + 406547217 = 2799421055 | Expected: 2799421055
# 1829520954 + 3399084951 = 933638609 | Expected: 933638609
# 4262154913 + 3365022192 = 3332209809 | Expected: 3332209809
# 4150029635 + 2034520217 = 1889582556 | Expected: 1889582556
# 2700410568 + 1559852673 = 4260263241 | Expected: 4260263241
# 868530962 + 340772296 = 1209303258 | Expected: 1209303258
# 3176695187 + 2503898584 = 1385626475 | Expected: 1385626475
# 3999197562 + 1536537231 = 1240767497 | Expected: 1240767497
# 2408530967 + 1213613466 = 3622144433 | Expected: 3622144433
# 1049571715 + 3749200253 = 503804672 | Expected: 503804672
# 2532668001 + 2469791436 = 707492141 | Expected: 707492141
# 1363101132 + 330012364 = 1693113496 | Expected: 1693113496
# 3638329757 + 1444738430 = 788100891 | Expected: 788100891
# 1912986898 + 1970454772 = 3883441670 | Expected: 3883441670
# 1173110238 + 3865482937 = 743625879 | Expected: 743625879
# 2171586715 + 2607770226 = 484389645 | Expected: 484389645
# 3757439021 + 1870921630 = 1333393355 | Expected: 1333393355
# 2845022853 + 2065968567 = 616024124 | Expected: 616024124
# 3543250620 + 4247361623 = 3495644947 | Expected: 3495644947
# ** Note: $finish    : verification/testbench.sv(45)
#    Time: 10 us  Iteration: 0  Instance: /tbench_top/test
# [Testbench]: End of testcases(s) at 10000
# Failed cases: 0 | Success cases: 5000
# ------------------------------------------------------
# End time: 19:31:52 on Oct 07,2021, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
```

## Carry Lookahead Adder

This section will outline how the carry lookahead adder was done and the outcome

### Testbench Structure

The structure for this adder is setup the same as the carry ripple adder. Tbench_top.sv is the exact same. The only change to interface and dut_top is that they now incorporate generate (G) and propagate (P) and a I/O

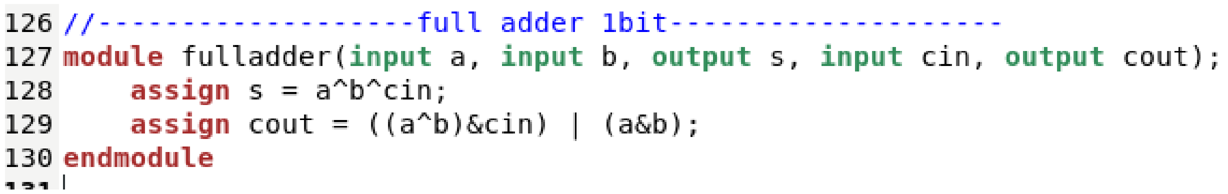


```
interface intf #(parameter bitsizeIntf = 32)(input logic clk);

    logic [bitsizeIntf-1:0] a;
    logic [bitsizeIntf-1:0] b;
    logic [bitsizeIntf-1:0] s;
    logic [bitsizeIntf-1:0] G;
    logic [bitsizeIntf-1:0] P;
    logic cin;
    logic cout;

    modport dut(
        input a,
        input b,
        output s,
        input cin,
        output cout,
        output G,
        output P,
        input clk
        );

    modport tb(
        output a,
        output b,
        input s,
        output cin,
        input cout,
        input G,
        input P,
        input clk
        );

endinterface
```

```
module dut_top #(parameter bitsizeDUTtop = 32)(interface i_intf);


    carryLookaheadAdder #(.bitsizeRCA(bitsizeDUTtop)) dut(
        .clk(i_intf.clk),
        .a(i_intf.a),
        .b(i_intf.b),
        .s(i_intf.s),
        .cin(i_intf.cin),
        .cout(i_intf.cout),
        .G(i_intf.G),
        .P(i_intf.P)
        );

endmodule
```

## Carry Lookahead Verilog File

The same problem happened for the carry lookahead as the one I encountered with the ripple adder. Where I could get a working timing analyzer or a working dut. Two different versions were made, one latched and one isn't. That being said, the exact same logic was used and it will be seen below. Both use the exact same full adder.

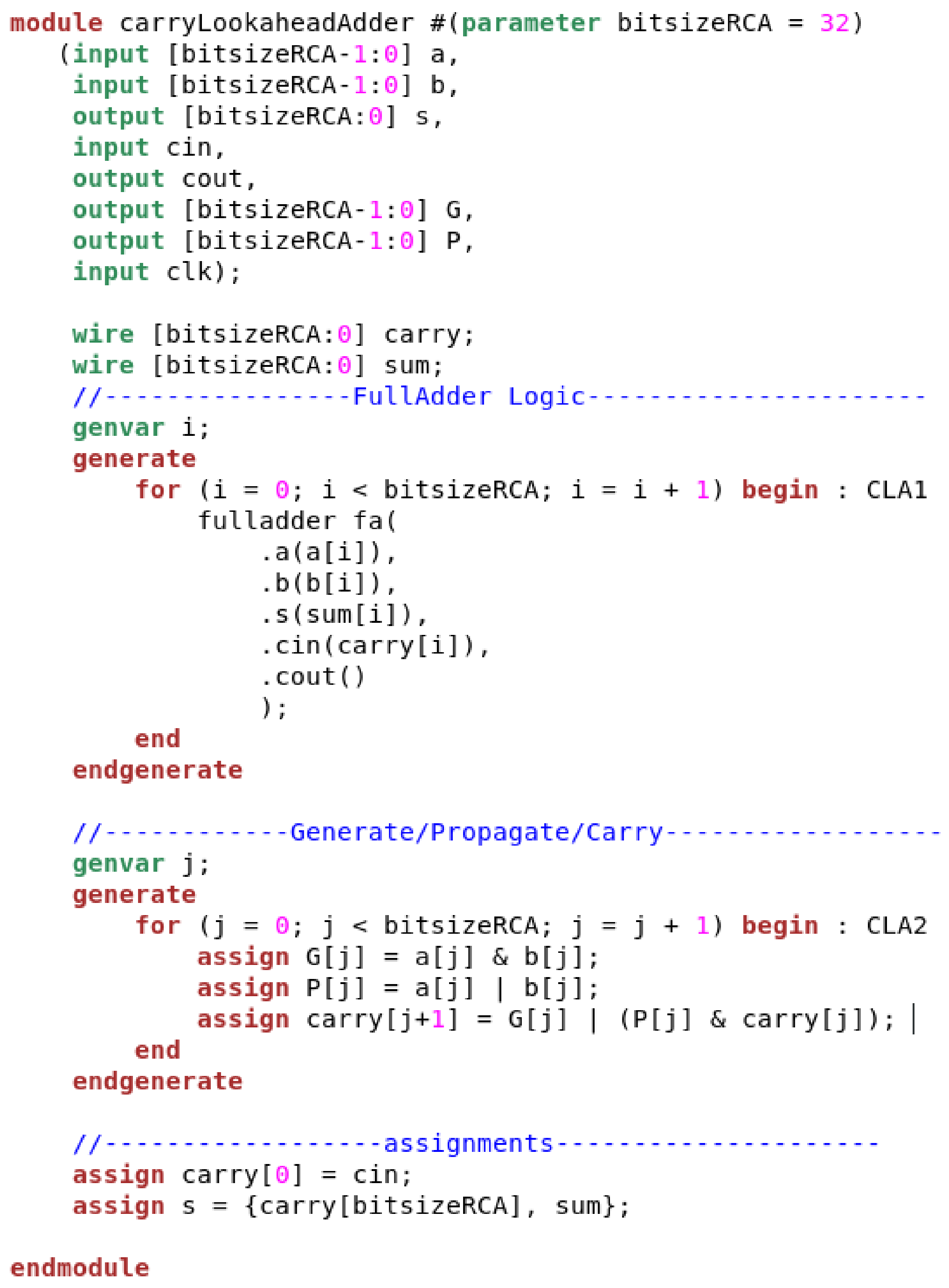


```
//-------------------full adder 1bit-------------------
module fulladder(input a, input b, output s, input cin, output cout);
    assign s = a^b^cin;
    assign cout = ((a^b)&cin) | (a&b);
endmodule
```

Below is the version that would allow the timing analyzer in quartus.

```
module carryLookaheadAdder #(parameter bitsizeRCA = 32)
   (input [bitsizeRCA-1:0] a,
    input [bitsizeRCA-1:0] b,
    output [bitsizeRCA:0] s,
    input cin,
    output reg cout,
    output reg [bitsizeRCA-1:0] G,
    output reg [bitsizeRCA-1:0] P,
    input clk);


    //----------------variable declarations---------------
    reg [bitsizeRCA-1:0] a_reg;
    reg [bitsizeRCA-1:0] b_reg;
    wire [bitsizeRCA-1:0] s_reg;
    reg cin_reg;
    reg cout_reg;
    wire [bitsizeRCA-1:0] G_reg;
    wire [bitsizeRCA-1:0] P_reg;

    wire [bitsizeRCA:0] carry;


    //-------------initialization------------------
    initial begin
        //$display("This is where all the regs are initialized to zero %0d", $time); // for debugging
        a_reg <= 0;
        b_reg <= 0;
        cin_reg <= 0;
    end
    //-------------Register Delcaration-------------
    always @(posedge clk)
        begin
            //$display("This is where all the regs are given its values %0d", $time); // for debugging
            a_reg <= a;
            b_reg <= b;
            //s <= s_reg;
            cin_reg <= cin;
            cout <= cout_reg;
            G <= G_reg;
            P <= P_reg;
        end
    //----------------FullAdder Logic---------------------
    genvar i;
    generate
        for (i = 0; i < bitsizeRCA; i = i + 1) begin : CLA1
            fulladder fa(
                .a(a_reg[i]),
                .b(b_reg[i]),
                .s(s_reg[i]),
                .cin(carry[i]),
                .cout()
                );
        end
    endgenerate
    //------------Generate/Propagate/Carry-----------------
    genvar j;
    generate
        for (j = 0; j < bitsizeRCA; j = j + 1) begin : CLA2
            assign G_reg[j] = a_reg[j] & b_reg[j];
            assign P_reg[j] = a_reg[j] | b_reg[j];
            assign carry[j+1] = G_reg[j] | (P_reg[j] & carry[j]);
        end
    endgenerate
    //-----------------assignments--------------------
    assign carry[0] = cin_reg;
    assign s = {carry[bitsizeRCA], s_reg};

endmodule
```

The version that was used for the testbench is as follows

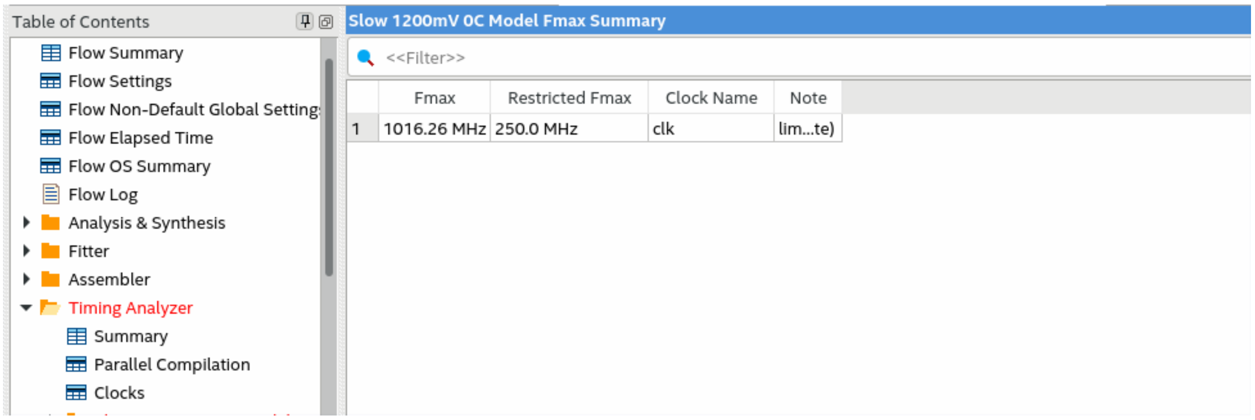


```
module carryLookaheadAdder #(parameter bitsizeRCA = 32)
   (input [bitsizeRCA-1:0] a,
    input [bitsizeRCA-1:0] b,
    output [bitsizeRCA:0] s,
    input cin,
    output cout,
    output [bitsizeRCA-1:0] G,
    output [bitsizeRCA-1:0] P,
    input clk);

    wire [bitsizeRCA:0] carry;
    wire [bitsizeRCA:0] sum;
    //----------------FullAdder Logic----------------------
    genvar i;
    generate
        for (i = 0; i < bitsizeRCA; i = i + 1) begin : CLA1
            fulladder fa(
                .a(a[i]),
                .b(b[i]),
                .s(sum[i]),
                .cin(carry[i]),
                .cout()
                );
        end
    endgenerate

    //------------Generate/Propagate/Carry------------------
    genvar j;
    generate
        for (j = 0; j < bitsizeRCA; j = j + 1) begin : CLA2
            assign G[j] = a[j] & b[j];
            assign P[j] = a[j] | b[j];
            assign carry[j+1] = G[j] | (P[j] & carry[j]);
        end
    endgenerate

    //------------------assignments---------------------
    assign carry[0] = cin;
    assign s = {carry[bitsizeRCA], sum};

endmodule
```

As you can see the logic itself is the exact same

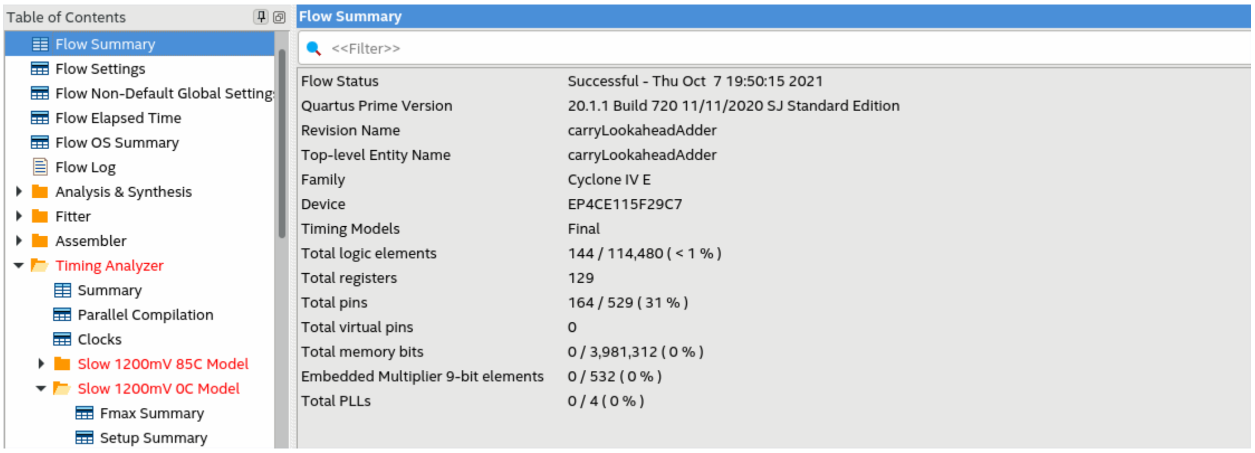
## Quartus Output for Carry Lookahead Adder

After running the file through quartus and it compiled, in the “Slow 1200mV 0C Model” it says that the Fmax was 1016.26Mhz. Which admittedly seem high, but it will be shown later that the testbench proves its functionality.



Slow 1200mV 0C Model Fmax Summary

| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 1016.26 MHz | 250.0 MHz | clk | lim...te) |

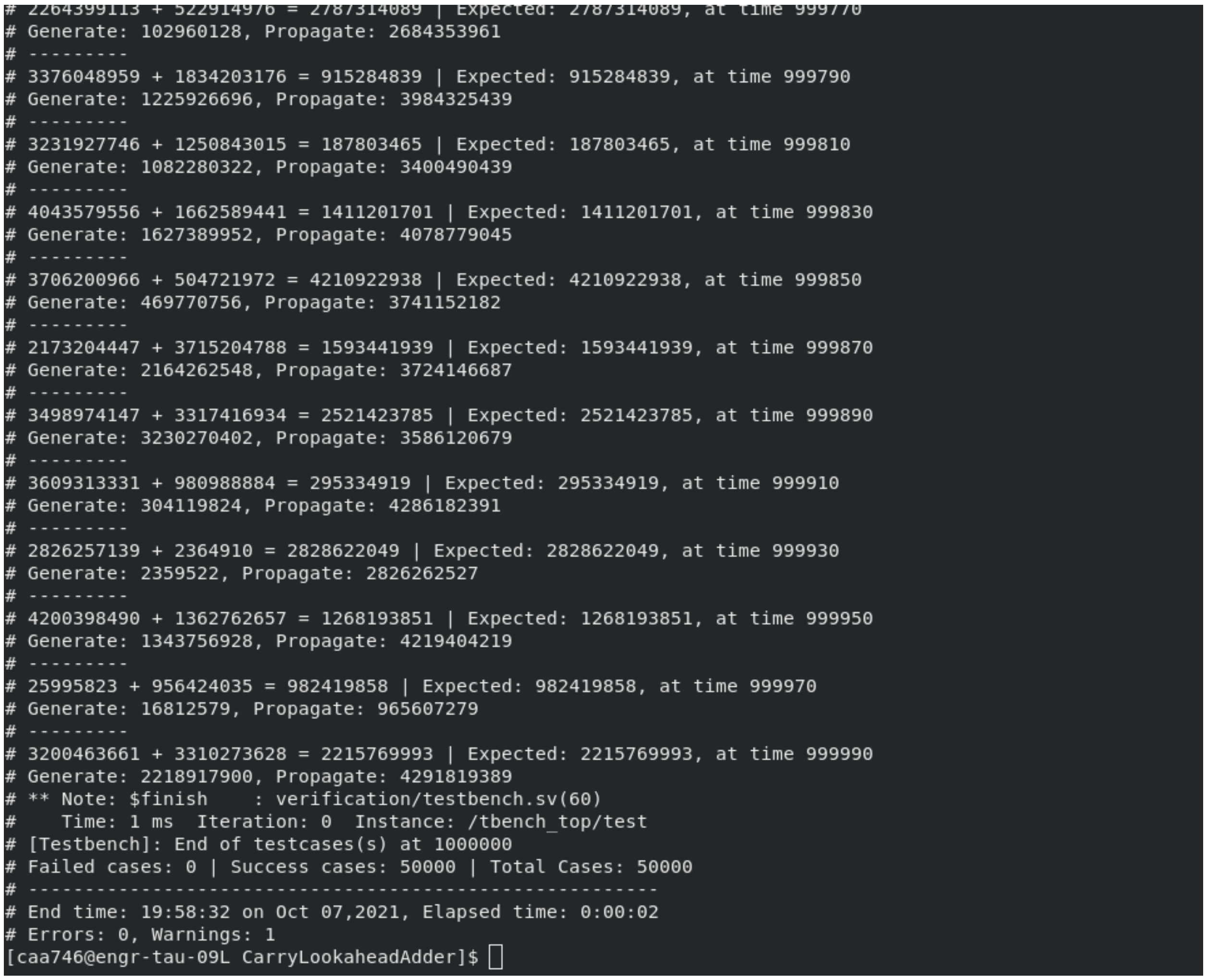


Flow Summary

| | |
|---|---|
| Flow Status | Successful - Thu Oct 7 19:50:15 2021 |
| Quartus Prime Version | 20.1.1 Build 720 11/11/2020 SJ Standard Edition |
| Revision Name | carryLookaheadAdder |
| Top-level Entity Name | carryLookaheadAdder |
| Family | Cyclone IV E |
| Device | EP4CE115F29C7 |
| Timing Models | Final |
| Total logic elements | 144 / 114,480 ( < 1 % ) |
| Total registers | 129 |
| Total pins | 164 / 529 ( 31 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 3,981,312 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 0 / 532 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

## Testbench Output for Carry Lookahead Adder

The testbench provided the same number of success cases as the ripple carry adder which makes me very confident that my design does work.

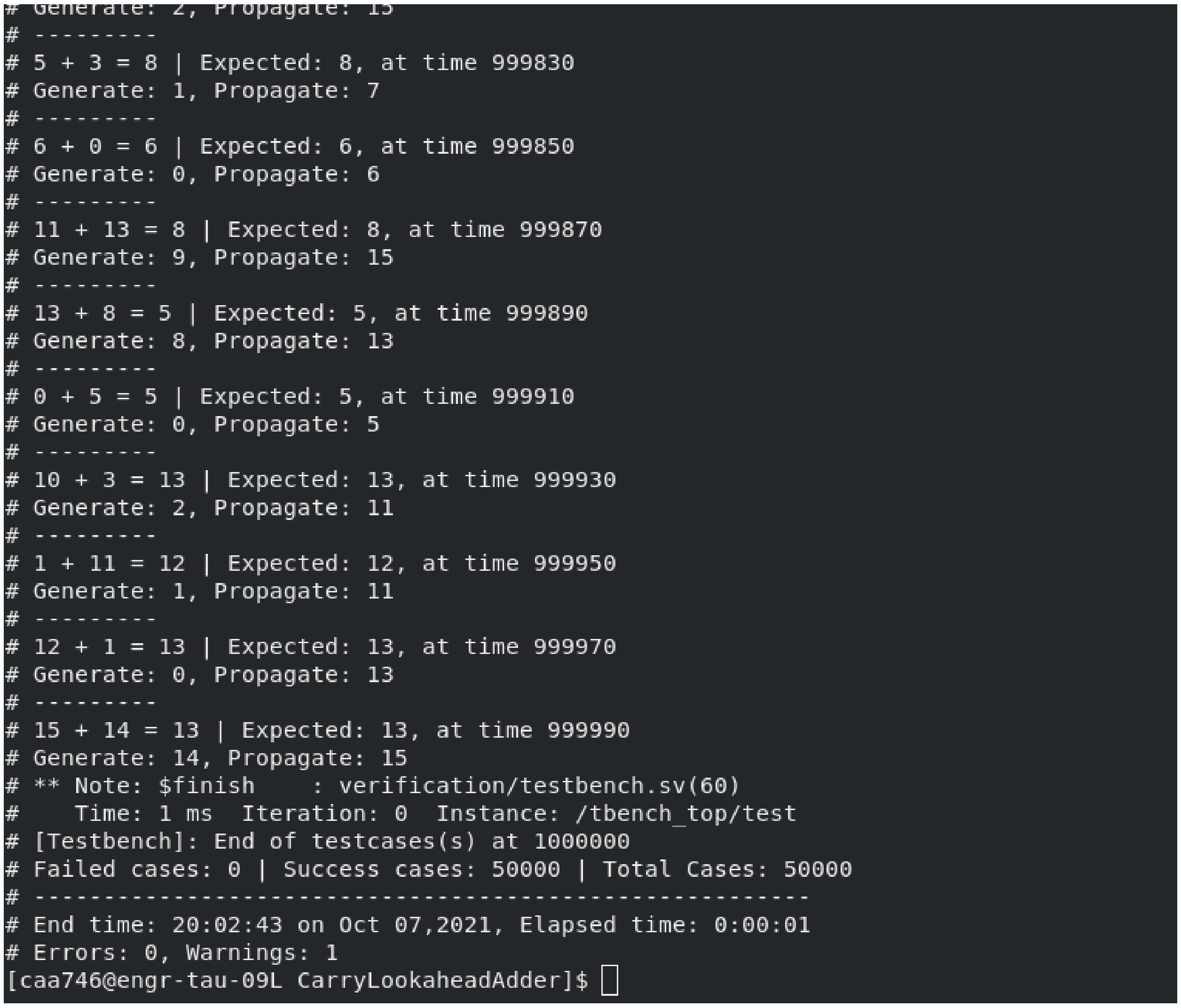


```
# 2264399113 + 522914976 = 2787314089 | Expected: 2787314089, at time 999770
# Generate: 102960128, Propagate: 2684353961
# ---------
# 3376048959 + 1834203176 = 915284839 | Expected: 915284839, at time 999790
# Generate: 1225926696, Propagate: 3984325439
# ---------
# 3231927746 + 1250843015 = 187803465 | Expected: 187803465, at time 999810
# Generate: 1082280322, Propagate: 3400490439
# ---------
# 4043579556 + 1662589441 = 1411201701 | Expected: 1411201701, at time 999830
# Generate: 1627389952, Propagate: 4078779045
# ---------
# 3706200966 + 504721972 = 4210922938 | Expected: 4210922938, at time 999850
# Generate: 469770756, Propagate: 3741152182
# ---------
# 2173204447 + 3715204788 = 1593441939 | Expected: 1593441939, at time 999870
# Generate: 2164262548, Propagate: 3724146687
# ---------
# 3498974147 + 3317416934 = 2521423785 | Expected: 2521423785, at time 999890
# Generate: 3230270402, Propagate: 3586120679
# ---------
# 3609313331 + 980988884 = 295334919 | Expected: 295334919, at time 999910
# Generate: 304119824, Propagate: 4286182391
# ---------
# 2826257139 + 2364910 = 2828622049 | Expected: 2828622049, at time 999930
# Generate: 2359522, Propagate: 2826262527
# ---------
# 4200398490 + 1362762657 = 1268193851 | Expected: 1268193851, at time 999950
# Generate: 1343756928, Propagate: 4219404219
# ---------
# 25995823 + 956424035 = 982419858 | Expected: 982419858, at time 999970
# Generate: 16812579, Propagate: 965607279
# ---------
# 3200463661 + 3310273628 = 2215769993 | Expected: 2215769993, at time 999990
# Generate: 2218917900, Propagate: 4291819389
# ** Note: $finish    : verification/testbench.sv(60)
#    Time: 1 ms  Iteration: 0  Instance: /tbench_top/test
# [Testbench]: End of testcases(s) at 1000000
# Failed cases: 0 | Success cases: 50000 | Total Cases: 50000
# --------------------------------------------------------
# End time: 19:58:32 on Oct 07,2021, Elapsed time: 0:00:02
# Errors: 0, Warnings: 1
[caa746@engr-tau-09L CarryLookaheadAdder]$ 
```

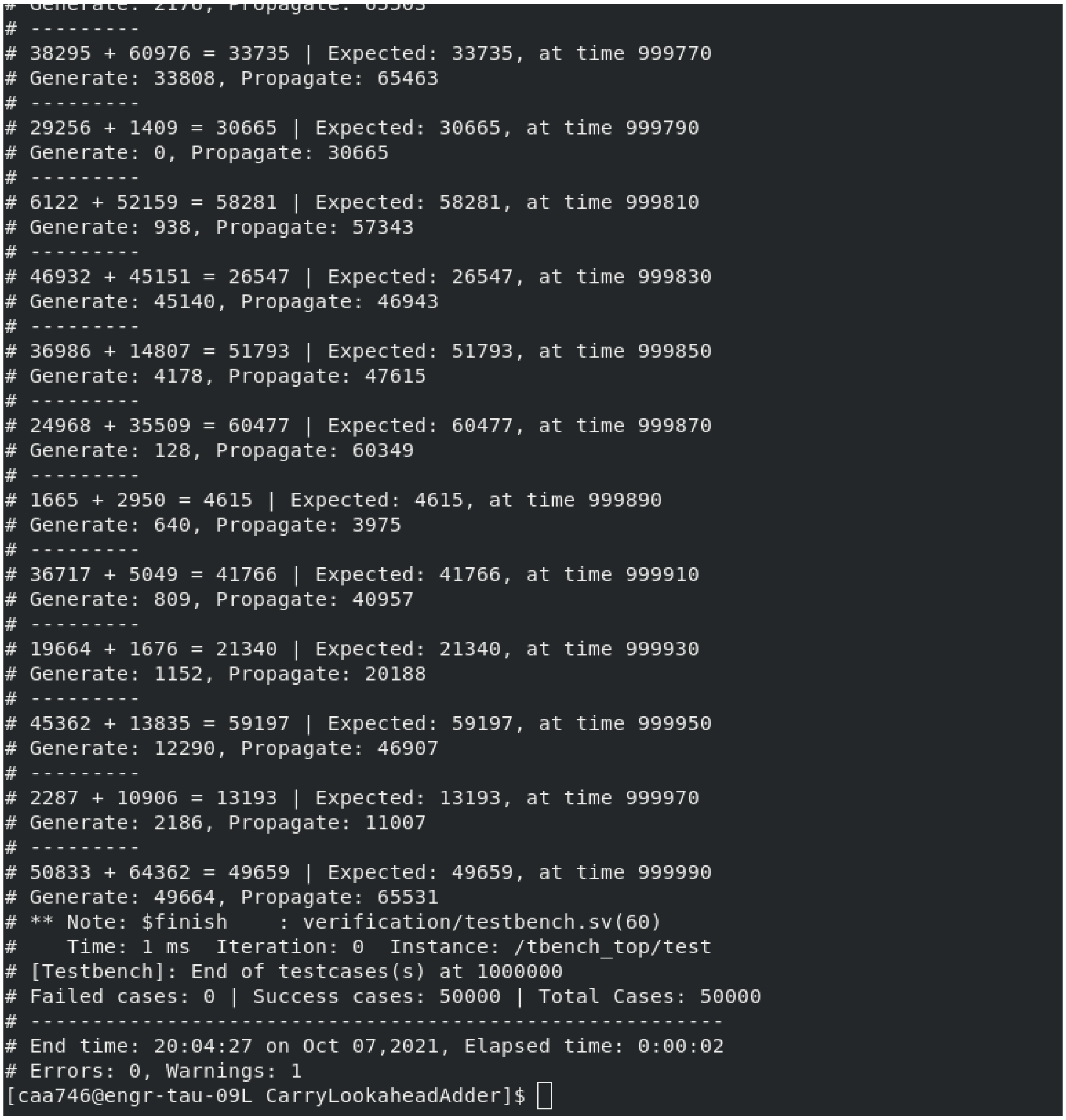
## Different Bitsize test for Carry Lookahead Adder

As mentioned before, the size of the carry lookahead adder is completely dependent on parameter variable set up on tbench_top. That parameter is on line 11. The following photos will demonstrate the testbench in different bit size

4-bit

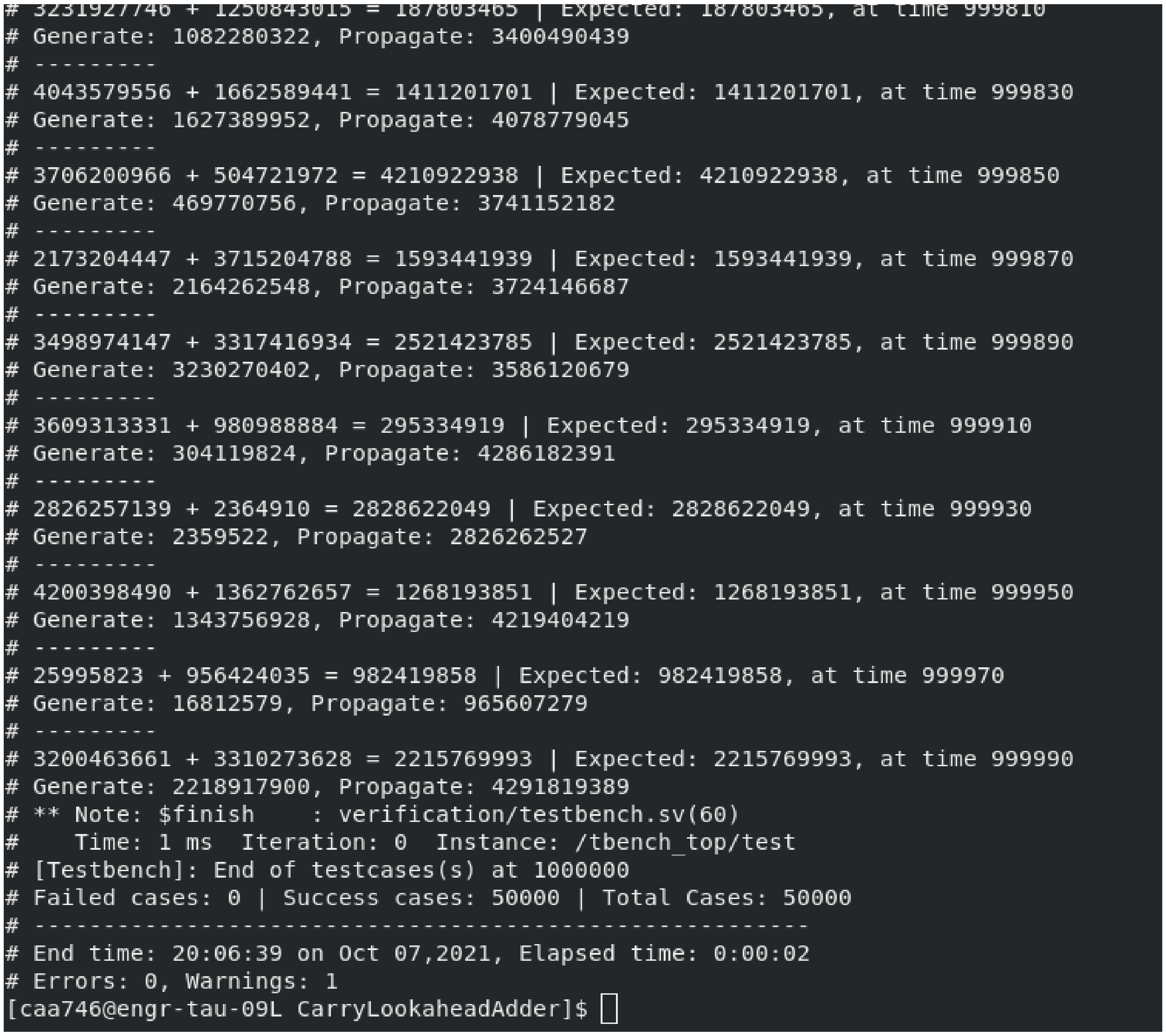

```
# Generate: 2, Propagate: 15
# ---------
# 5 + 3 = 8 | Expected: 8, at time 999830
# Generate: 1, Propagate: 7
# ---------
# 6 + 0 = 6 | Expected: 6, at time 999850
# Generate: 0, Propagate: 6
# ---------
# 11 + 13 = 8 | Expected: 8, at time 999870
# Generate: 9, Propagate: 15
# ---------
# 13 + 8 = 5 | Expected: 5, at time 999890
# Generate: 8, Propagate: 13
# ---------
# 0 + 5 = 5 | Expected: 5, at time 999910
# Generate: 0, Propagate: 5
# ---------
# 10 + 3 = 13 | Expected: 13, at time 999930
# Generate: 2, Propagate: 11
# ---------
# 1 + 11 = 12 | Expected: 12, at time 999950
# Generate: 1, Propagate: 11
# ---------
# 12 + 1 = 13 | Expected: 13, at time 999970
# Generate: 0, Propagate: 13
# ---------
# 15 + 14 = 13 | Expected: 13, at time 999990
# Generate: 14, Propagate: 15
# ** Note: $finish    : verification/testbench.sv(60)
#    Time: 1 ms  Iteration: 0  Instance: /tbench_top/test
# [Testbench]: End of testcases(s) at 1000000
# Failed cases: 0 | Success cases: 50000 | Total Cases: 50000
# ---------------------------------------------------
# End time: 20:02:43 on Oct 07,2021, Elapsed time: 0:00:01
# Errors: 0, Warnings: 1
[caa746@engr-tau-09L CarryLookaheadAdder]$ 
```

16-bit



```
# ---------
# 38295 + 60976 = 33735 | Expected: 33735, at time 999770
# Generate: 33808, Propagate: 65463
# ---------
# 29256 + 1409 = 30665 | Expected: 30665, at time 999790
# Generate: 0, Propagate: 30665
# ---------
# 6122 + 52159 = 58281 | Expected: 58281, at time 999810
# Generate: 938, Propagate: 57343
# ---------
# 46932 + 45151 = 26547 | Expected: 26547, at time 999830
# Generate: 45140, Propagate: 46943
# ---------
# 36986 + 14807 = 51793 | Expected: 51793, at time 999850
# Generate: 4178, Propagate: 47615
# ---------
# 24968 + 35509 = 60477 | Expected: 60477, at time 999870
# Generate: 128, Propagate: 60349
# ---------
# 1665 + 2950 = 4615 | Expected: 4615, at time 999890
# Generate: 640, Propagate: 3975
# ---------
# 36717 + 5049 = 41766 | Expected: 41766, at time 999910
# Generate: 809, Propagate: 40957
# ---------
# 19664 + 1676 = 21340 | Expected: 21340, at time 999930
# Generate: 1152, Propagate: 20188
# ---------
# 45362 + 13835 = 59197 | Expected: 59197, at time 999950
# Generate: 12290, Propagate: 46907
# ---------
# 2287 + 10906 = 13193 | Expected: 13193, at time 999970
# Generate: 2186, Propagate: 11007
# ---------
# 50833 + 64362 = 49659 | Expected: 49659, at time 999990
# Generate: 49664, Propagate: 65531
# ** Note: $finish    : verification/testbench.sv(60)
#    Time: 1 ms  Iteration: 0  Instance: /tbench_top/test
# [Testbench]: End of testcases(s) at 1000000
# Failed cases: 0 | Success cases: 50000 | Total Cases: 50000
# ---------------------------------------------------------
# End time: 20:04:27 on Oct 07,2021, Elapsed time: 0:00:02
# Errors: 0, Warnings: 1
[caa746@engr-tau-09L CarryLookaheadAdder]$ 
```

32-bit



```
# 3231927746 + 1250843015 = 187803465 | Expected: 187803465, at time 999810
# Generate: 1082280322, Propagate: 3400490439
# ---------
# 4043579556 + 1662589441 = 1411201701 | Expected: 1411201701, at time 999830
# Generate: 1627389952, Propagate: 4078779045
# ---------
# 3706200966 + 504721972 = 4210922938 | Expected: 4210922938, at time 999850
# Generate: 469770756, Propagate: 3741152182
# ---------
# 2173204447 + 3715204788 = 1593441939 | Expected: 1593441939, at time 999870
# Generate: 2164262548, Propagate: 3724146687
# ---------
# 3498974147 + 3317416934 = 2521423785 | Expected: 2521423785, at time 999890
# Generate: 3230270402, Propagate: 3586120679
# ---------
# 3609313331 + 980988884 = 295334919 | Expected: 295334919, at time 999910
# Generate: 304119824, Propagate: 4286182391
# ---------
# 2826257139 + 2364910 = 2828622049 | Expected: 2828622049, at time 999930
# Generate: 2359522, Propagate: 2826262527
# ---------
# 4200398490 + 1362762657 = 1268193851 | Expected: 1268193851, at time 999950
# Generate: 1343756928, Propagate: 4219404219
# ---------
# 25995823 + 956424035 = 982419858 | Expected: 982419858, at time 999970
# Generate: 16812579, Propagate: 965607279
# ---------
# 3200463661 + 3310273628 = 2215769993 | Expected: 2215769993, at time 999990
# Generate: 2218917900, Propagate: 4291819389
# ** Note: $finish    : verification/testbench.sv(60)
#    Time: 1 ms  Iteration: 0  Instance: /tbench_top/test
# [Testbench]: End of testcases(s) at 1000000
# Failed cases: 0 | Success cases: 50000 | Total Cases: 50000
# --------------------------------------------------------
# End time: 20:06:39 on Oct 07,2021, Elapsed time: 0:00:02
# Errors: 0, Warnings: 1
[caa746@engr-tau-09L CarryLookaheadAdder]$ 
```

## Conclusion

Since I made the testbench prior to the Modelsim. I could not directly import the files to Modelsim. It gave me a working saying that "(vsim-7099) Unable to checkout a verification license for the randomize() feature." So my attempt for a modelsim waveform will be unsuccessful. If you are familiar with the problem some feedback for how I could fix this in the future would be greatly appreciated. For this lab, I hope that my bash testbench is enough to convince you that my approach does in fact work.

This lab has made it clear that the carry lookahead adder is faster than the ripple carry adder. That being said, it is also clear from the flow summaries that were produced by both that the carry lookahead adder requires more space to accomplish. Both of these statements are consistent with what was taught in class. Another thing I learned was the usefulness of setting up a testbench "environment" as taught in CME435 (I am aware this structure is not taught in CME433) because of its ability to easily integrate different designs into a complex workbench