CMPE-630 Digital Integrated Circuit Design
Final Project

Multiply and Accumulate (MAC) Datapath Unit Design

Brandon Key
Chris Guarini
Performed: 9 Dec 2019
Submitted: 9 Dec 2019

Instructor: Dr. Amlan Ganguly
TAs: Abhishek Vashist
Andrew Fountain
Piers Kwan

By submitting this report, you attest that you neither have given nor have received any assistance (including writing, collecting data, plotting figures, tables or graphs, or using previous student reports as a reference), and you further acknowledge that giving or receiving such assistance will result in a failing grade for this course.

Brandon Key: ____________________

Chris Guarini: ____________________

Contents

1 Abstract

2 Design Methodology and Theory

A cornerstone of IC design is the ability to create large, complex designs from smaller more manageable parts. The project outlined in this exercise calls for the design, testing and layout of a multiply and accumulate (MAC) unit, which takes two 16-bit inputs, multiplies them together, adds them to the value stored in a register, and then stores that output back into the register. The final component should contain a built in self test (BIST) that verifies the functionality of the MAC.

The MAC is composed of a carry-save multiplier, ripple carry full-adder, and parallel register. The BIST is implemented through the use of an LFSR for the inputs, an MISR for the output, and a test controller which controls the timing and sets the test passed and test complete outputs. A full diagram of the MAC with BIST can be seen in Figure 1.

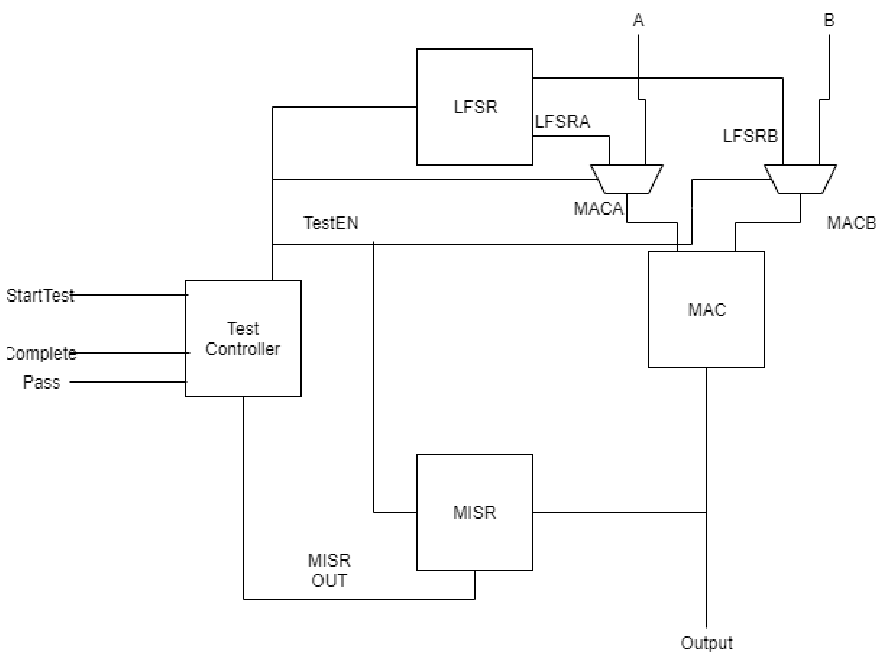


Figure 1: High Level Block Diagram of the MAC with BIST

2.1 User Operation

The full MAC and BIST design has a total of six inputs and three outputs, which are shown in Table 1 and Table 2 respectively. All inputs and outputs of the MAC are clocked, so a clk signal is required for operation. To enable the writing to register, WE must be high for both MAC and BIST functionality. This allows for initialization of input values without it writing to the registers, and for the ability to hold what value the registers contain without changing the inputs.

In order to run the MAC in its functional mode, WE must be high, reset and StartTest must be low. Once these requirements are met, the values of inputs A and B will be multiplied together and accumulated into the registers. The output RegOut will contain the current value in the register.

To run the BIST, the component must first be reset for a total of three clock cycles during initializing. This can be achieved by holding reset low during initialization. Afterwords, reset, WE and StartTest should be high. The BIST runs for a total of 1000 clock cycles, during which the outputs Pass and Complete will be low. When Complete goes high the BIST is over and Pass will contain whether the test passed or not.

Table 1: Inputs of the MAC

Input	Function	Size (Bits)
A	Input 1	N/2
B	Input 2	N/2
clk	Clock Signal	1
WE	Write Enable	1
reset	Active Low Reset	1
StartTest	Enable BIST	1

Table 2: Outputs of the MAC

Output	Function	Size (Bits)
RegOut	Output of the MAC	N
Pass	BIST Pass Flag	1
Complete	BIST Complete Flag	1

2.2 Adder

The adder used in this design is an N-bit ripple carry adder which accepts input from the outputs of the multiplier and accumulation register. A ripple carry adder is composed of a series of N full adders, where the carry-out of the previous full adder is fed into the carry-in of the next full adder, as shown in Figure 2.

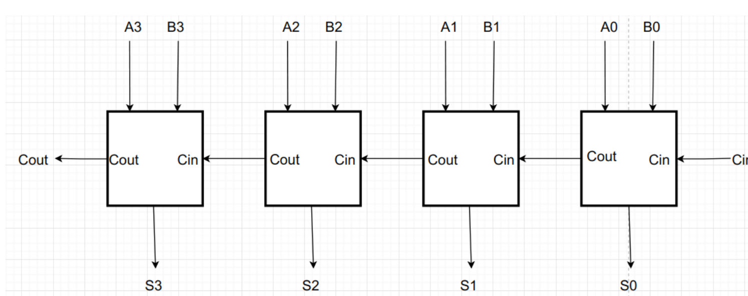


Figure 2: Ripple Carry Adder

The code in Figure 3 shows one of the generate statements used to create the nBitAdder from full adder components. The uses VHDL generate statements to generate N-2 full adders that take their carry-in bit from the previous full adder and send their carry out to the next full adder through the internal signal carray. The first and last full adder in the series are generated differently, as the first full adder's carry-in is hard coded to 0 and the last full adder's carry-out is an output. The full code of the nBitAdder is available in Listing 20 of the appendix.

```
i_mid : if (i /= 0) and (i /= (n-1)) generate
    adder : full_adder port map(
        A => A(i),
        B => B(i),
        Cin => c_array(i-1),
        Sum => Y(i),
        Cout => c_array(i)
    );
end generate i_mid;
```

Figure 3: nBitAdder Code Snippet

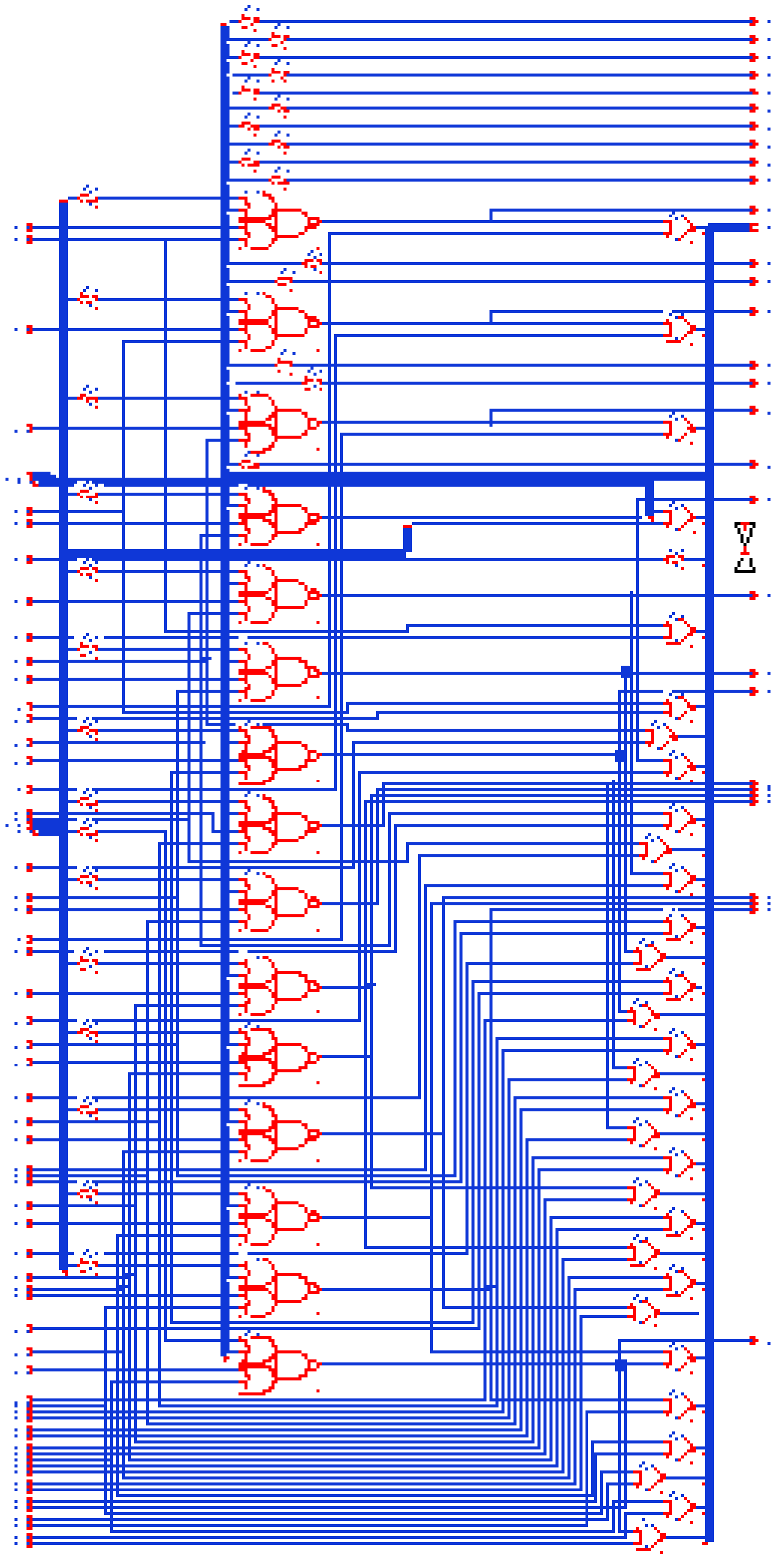


Figure 4: nBitAdder Schematic Page 1

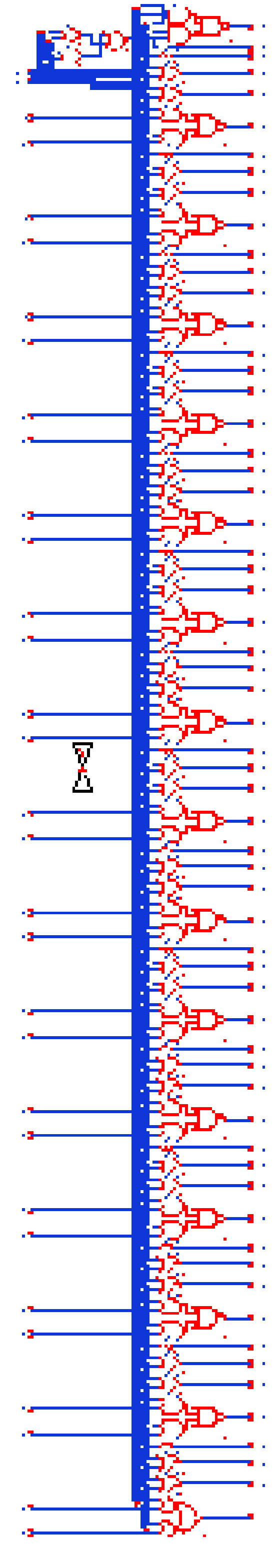


Figure 5: nBitAdder Schematic Page 2

2.3 Multiplier

The MAC includes a carry-save multiplier component in order to multiply the two inputs together. A carry-save multiplier works as shown in Figure 6, where full adders are arranged in a 2d array where each row is N/2 adders long and offset by 1.

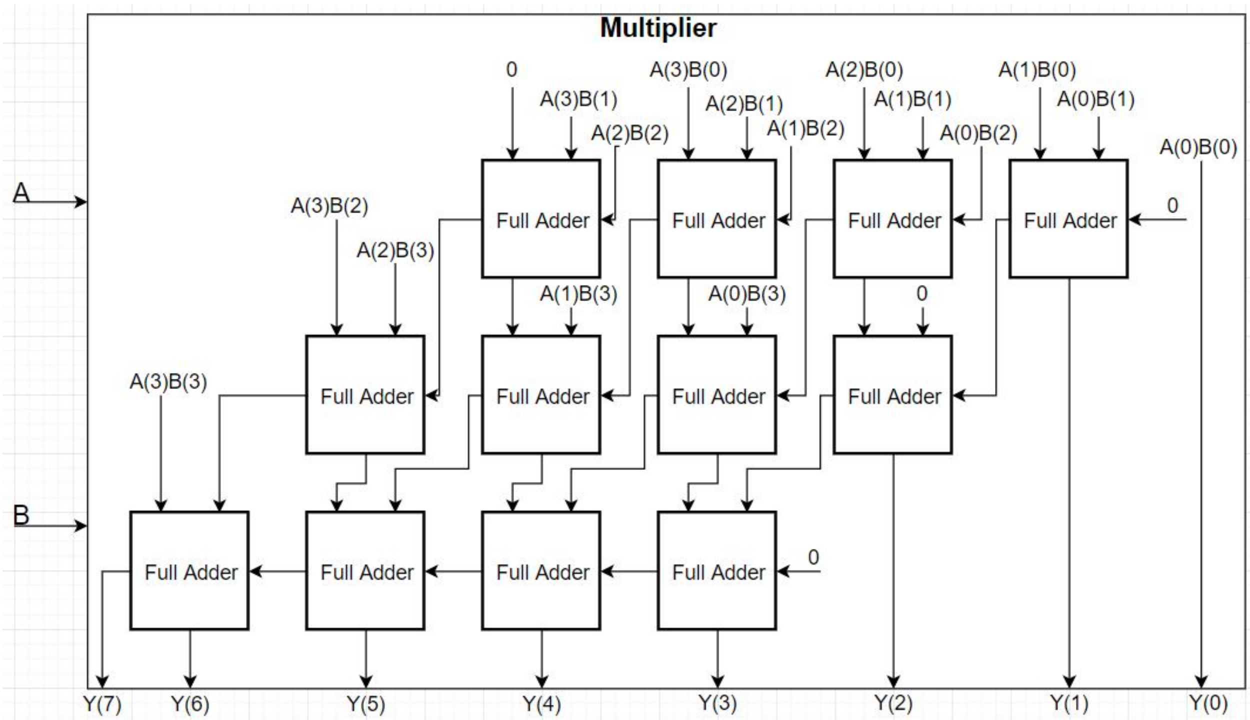


Figure 6: Carry-Save Multiplier Block Diagram

The full VHDL code for the multiplier is found in Listing 14. The code takes advantage of VHDL generate statements to generate the four different connections a full adder could have in a carry-save multiplier. The first fill adder of each row, the last full adder of the first row, the last full adder of every other row, and all other full adders.

Figures 7 and 8 show both pages of the multiplier schematic. These schematics were generated from the VHDL code in 14 using Spectrum scripts to create a Verilog file that was then imported into Pyxis.

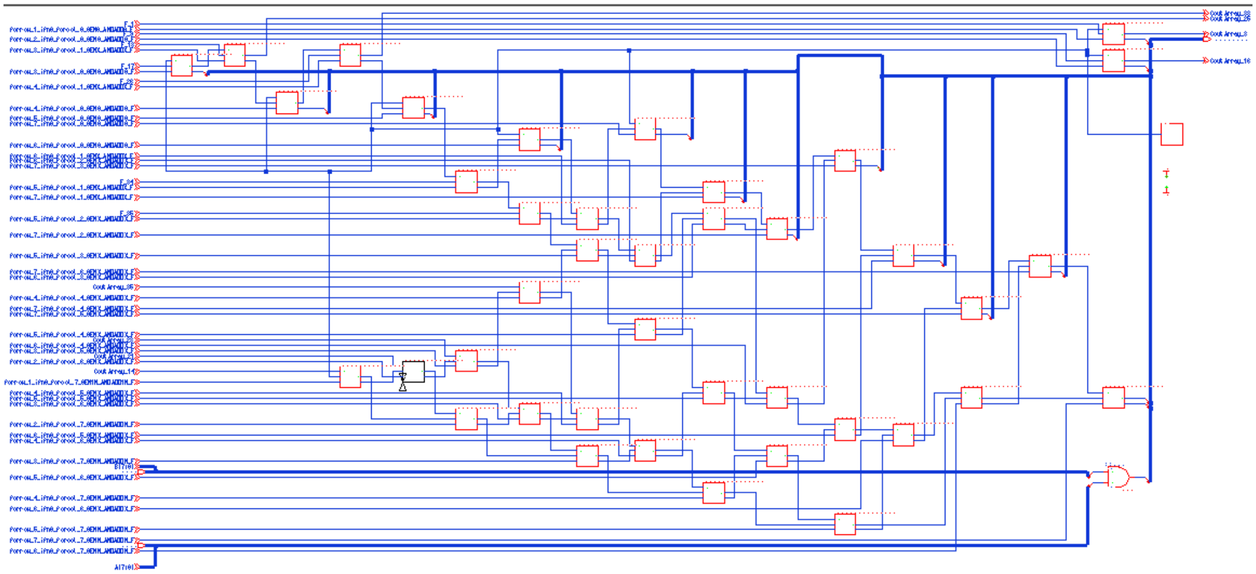


Figure 7: Multiplier Schematic Page 1

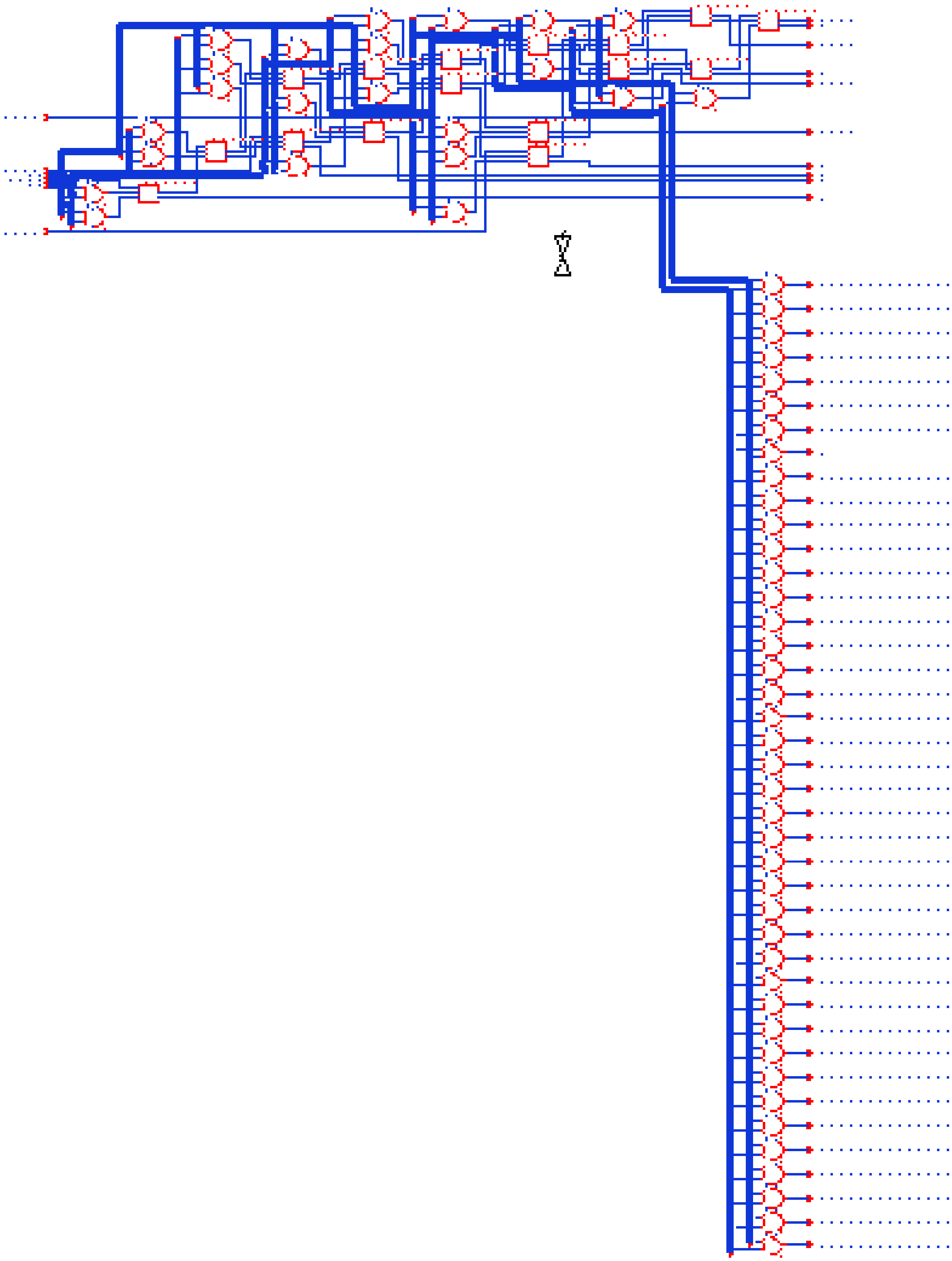


Figure 8: Multiplier Schematic Page 2

2.4 16-Bit Register

Two 16-bit parallel registers are used to clock the two inputs to the multiplier. The VHDL architecture for the register can be seen in Figure 9, which shows that the register has an active low reset, and when WE is high will take new input. Otherwise the register holds its previous value.

```
architecture behav of nBitRegister_16 is
begin
    output_proc : process (clk, Reset) begin
        if Reset = '0' then
            Y <= (others => '0');
        elsif clk'event and clk = '1' then
            if WE = '1' then
                Y <= nBitIn;
            end if;
        end if;
    end process output_proc;
end behav;
```

Figure 9: 16-Bit Register Code Snippet

Figure 10 shows the schematic for the 16-bit register, which provides some insight into how a parallel register is made. There are 16 flip-flops which are connected to their respective input bits, the inverse of the reset signal, a write enable signal, and the clock. Their outputs are connected to the respective output bit.

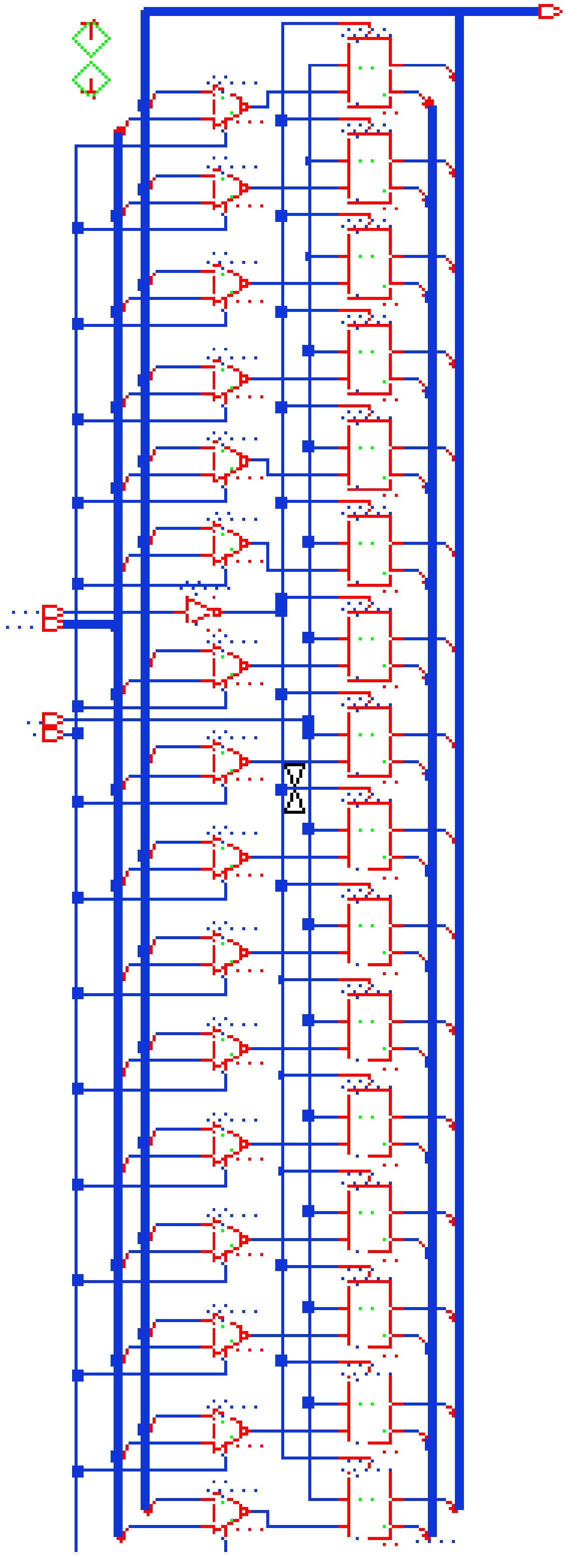


Figure 10: 16 Bit Register Schematic

2.5 32-Bit Register

A 32-bit parallel register is used as the accumulator in the MAC component. The accumulator register if functionally the same as the 16-bit register above, as they are generated from the same generic VHDL code, just with different values for N.

The code in Figure 11 shows the code used to generate the 32-bit register, which is identical to the 16-bit register in all but name. The schematic below in Figure 12 shows how the 32-bit register is functionally identical to the 16-bit register, just with double the amount of flip-flops and logic.

```
architecture behav of nBitRegister_32 is
begin
    output_proc : process (clk, Reset) begin
        if Reset = '0' then
            Y <= (others => '0');
        elsif clk'event and clk = '1' then
            if WE = '1' then
                Y <= nBitIn;
            end if;
        end if;
    end process output_proc;
end behav;
```

Figure 11: 32-Bit Register Code Snippet

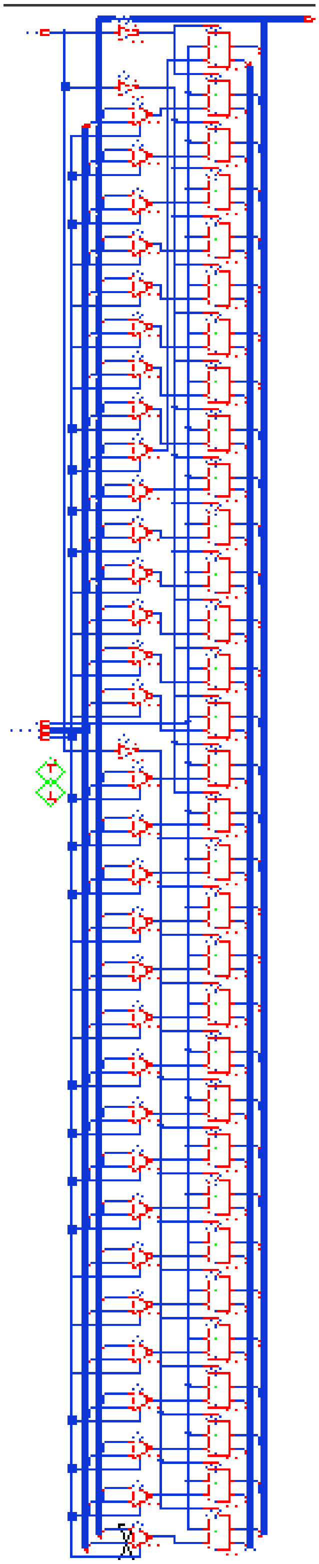


Figure 12: 32 Bit Register Schematic

2.6 MAC

The MAC component takes two N/2 bit inputs, A and B. These inputs are multiplied together using a carry-save multiplier and added together with the value in the register using a ripple carry adder. This value is then stored back into the register where it becomes the N bit output RegOut. The block diagram for this logic is in Figure 13.

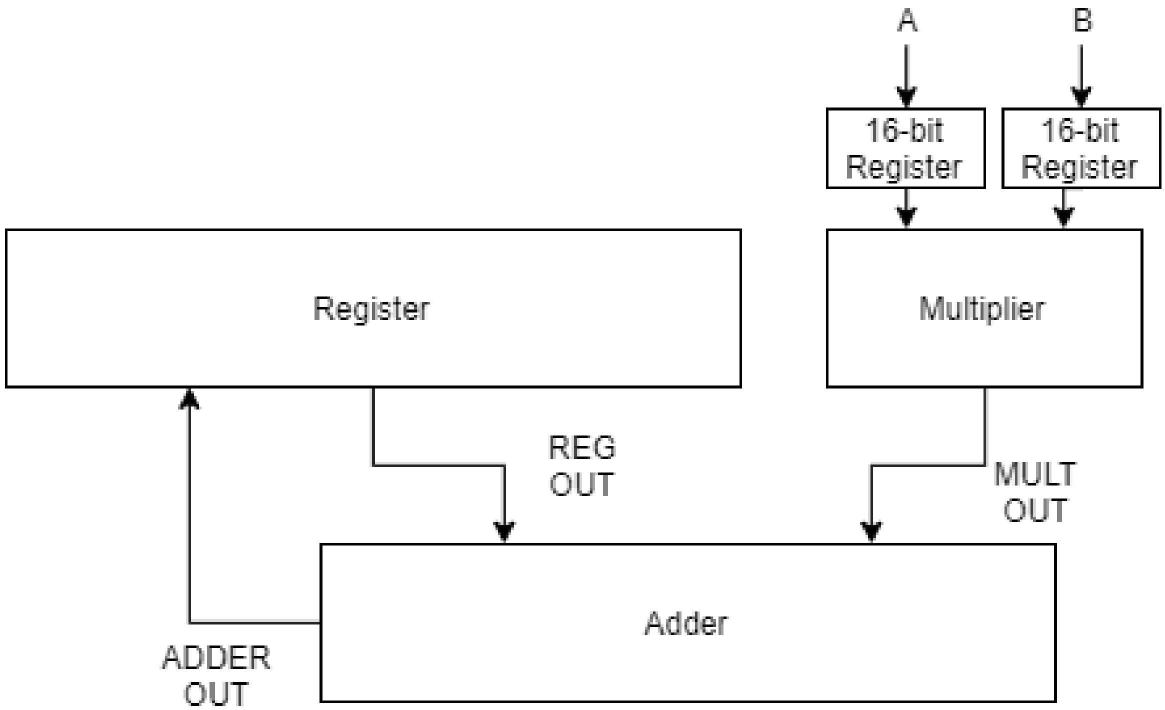


Figure 13: MAC block

A snippet of the VHDL used to structurally create the MAC is shown in Figure 14, while the full code is available in Listing 17 of the appendix. The code is a fairly straight forward structural representation of the MAC, using 7 internal signals to map the components together and to the inputs and outputs. The code creates a 32-bit MAC with two 16-bit inputs

```
signal MultA,MultB : STD_LOGIC_VECTOR((N/2)-1 downto 0);
signal Product : STD_LOGIC_VECTOR(N-1 downto 0);
signal adderA, adderB, adderOut : STD_LOGIC_VECTOR(N-1 downto 0);
signal cout : STD_LOGIC;

begin

    RegMultInA : nBitRegister_16
        generic map( N => 16)
        port map(nBitIn => A,
            WE => '1', clk => clk, Reset => reset,
            Y => MultA
        );

  RegMultInB : nBitRegister_16
   generic map( N => 16)
   port map(nBitIn => B,
 WE => '1', clk => clk, Reset => reset,
 Y=> MultB
   );

    MULT1 : Multiplier
        generic map( N => 32)
        port map(A => MultA, B => MultB, Product => Product);

    RegMultOut : nBitRegister_32
        generic map( N => 32)
        port map(nBitIn => Product, WE => '1', Reset => reset, clk => clk, Y => adderB)

    BigBoyReg : nBitRegister_32
        generic map( N => 32)
        port map(nBitIn => adderOut, WE => WE, Reset => reset, clk => clk, Y => adderA)

    ADD1 : nBitAdder
        generic map ( N => 32)
        port map( A => adderA, B => adderB, Y => adderOut, CB => cout);

 RegOut <= adderA;
end Behavioral;
```

Figure 14: MAC Code Snippet

A VHDL test bench verifying the functionality of the MAC is shown in Figure 15. ModelSim was used to simulate this waveform using the stimulus code shown in Figure 16. This code starts with an input of 2 and 2, which are multiplied and accumulated. The code then changes to inputs of 2 and 4 after 300 ns.

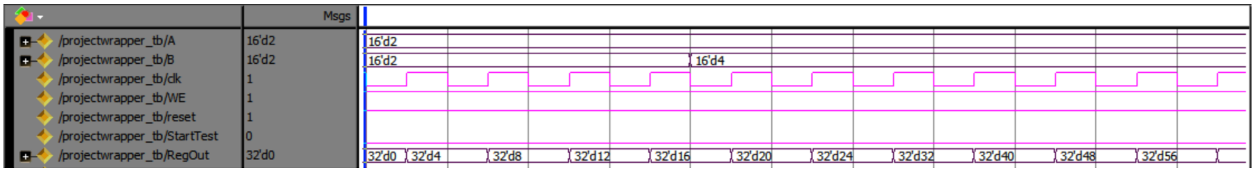


Figure 15: MAC Functional Test Bench

```
stimulus: process
begin

    WE <= '1';
    reset <= '1';
    A <= "0000000000000010";
    B <= "0000000000000010";
    wait for 300 ns;

    A <= "0000000000000010";
    B <= "0000000000000100";
    wait;

end process;
```

Figure 16: MAC Test Bench Stimulus Code Snippet

2.7 Mux

In order to implement the BIST functionality of the IC, two 16-bit multiplexers are required on each input into the multiplier's input registers. the multiplexers select whether the input should be from the LFSR or the MAC's input. Figure 17 shows the code used to create the MUX, which uses a data-flow architecture and a process to select the correct input.

```
architecture  Dataflow  of  nBitMux_2to1  is
    begin
        --Mux process
        the_proc : process (sel, A, B) begin
            case sel is
                when '0' =>
                    Y <= A;
                when others =>
                    Y <= B;
            end case;
        end process the_proc;

end  Dataflow;
```

Figure 17: MAC Test Bench Stimulus Code Snippet

Figure 18 shows the schematic of the 16-bit MUX. The schematic is 16 gdk MUX 2 to 1 gates in parallel.

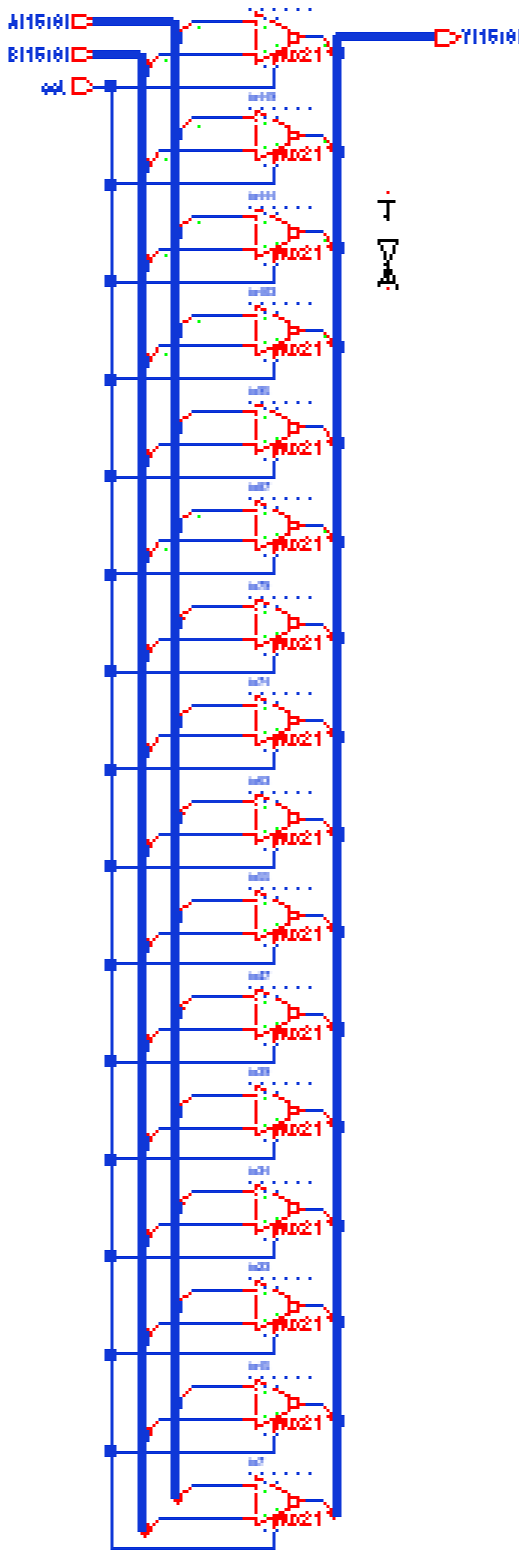


Figure 18: nBitMux 2to1 Schematic

2.8 LFSR

An LFSR is a predictably random number generator with known pattern depending on what the seed is. For the BIST functionality of the IC, a 32-bit LFSR is used to generate the two inputs, with the top 16 bits going to input A and the bottom 16 bits going to input B. The 32-bit LFSR has taps at bit 19, 24 and 25 along with an input seed of 0x00BC614E, or 12345678 in decimal. The inner workings of an 8-bit LFSR can be seen in Figure 19. The register takes the seed and continuously shifts it through the different flip-flops, with a tap on certain bits using XOR gates which randomizes the output.

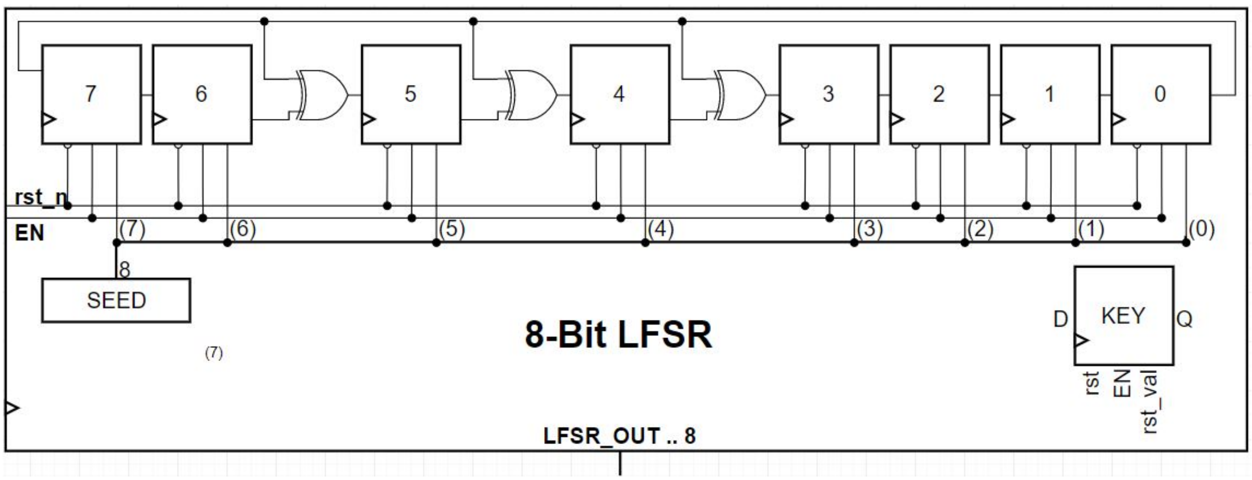


Figure 19: 8-Bit Linear Feedback Shift Register

2.9 MISR

The MISR is used by the BIST to take the output of the MAC and turns it into a 32-bit signature. The signature is then compared to the one in the test controller to see if the BIST has passed. The purpose of this rather than just comparing the actual output is that the signature takes into account when the output occurred and the previous inputs. This means that no two signatures will be the same, but they are predictably random. Figure 20 is a diagram of an 8-bit MISR as an example. The MISR used in the final design has taps that match the LFSR (19, 24 and 25).

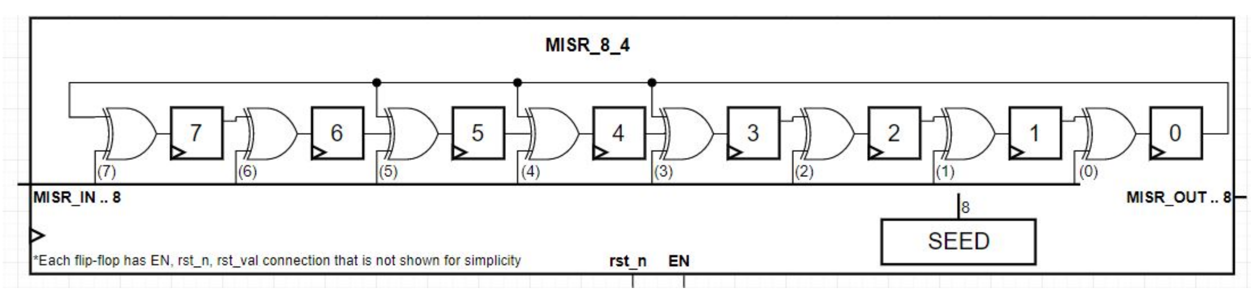


Figure 20: Multiple Input Shift Register

2.10 BIST

The test controller used to implement BIST compares the signature of the MISR to an expected signature that would occur after 1000 clock cycles. This is to make sure that the MAC is thoroughly tested with as many inputs as possible. The signature that the controller is looking for is 0x8C9781E6. The waveform in Figure 21 shows how that signature is seen on the 1000 clock cycle, which causes the Complete and Pass outputs to go high, as the BIST completed successfully.

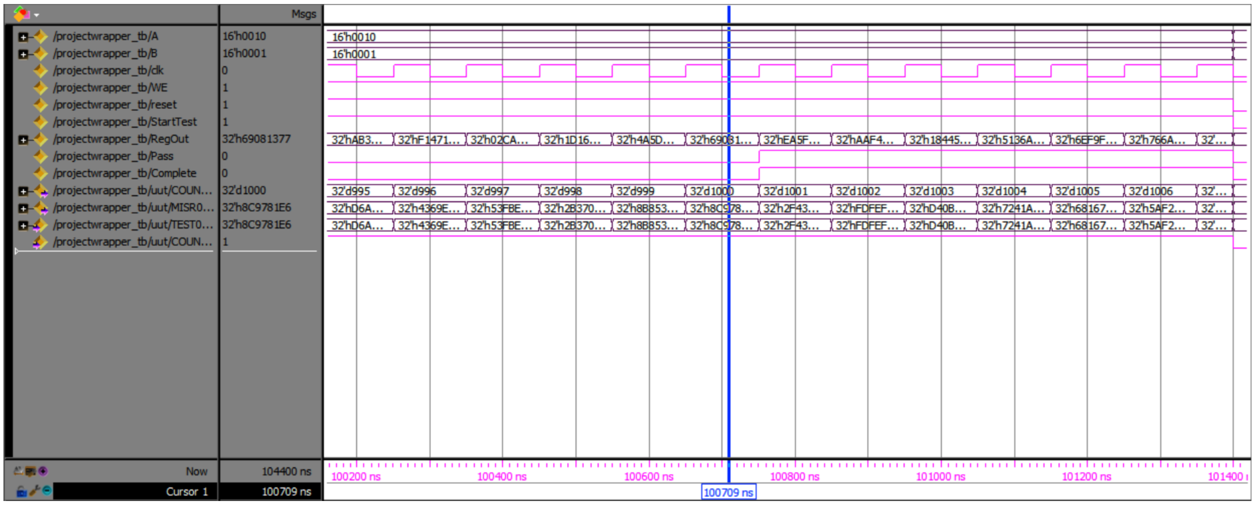


Figure 21: BIST Test Bench

2.11 Schematic

The schematics for the MAC with BIST are shown in Figures 22 through 49, which is twenty-eight sheets. These sheets were generated using the VHDL code in Listing 11 of the appendix, which defines the project structurally. Verilog code was generated from the VHDL using the Leonardo script in Listing 34, which was then import into Pyxis to generate the schematics.

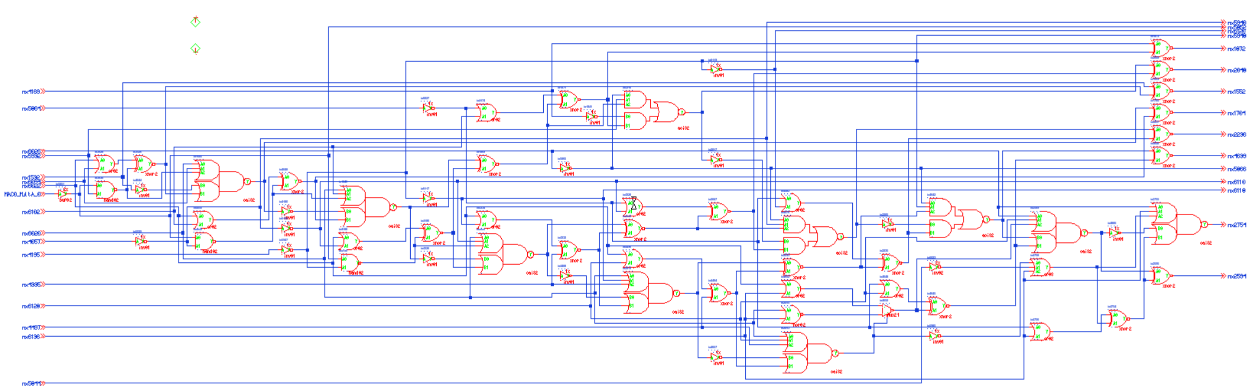


Figure 22: Full Schematic Page 1

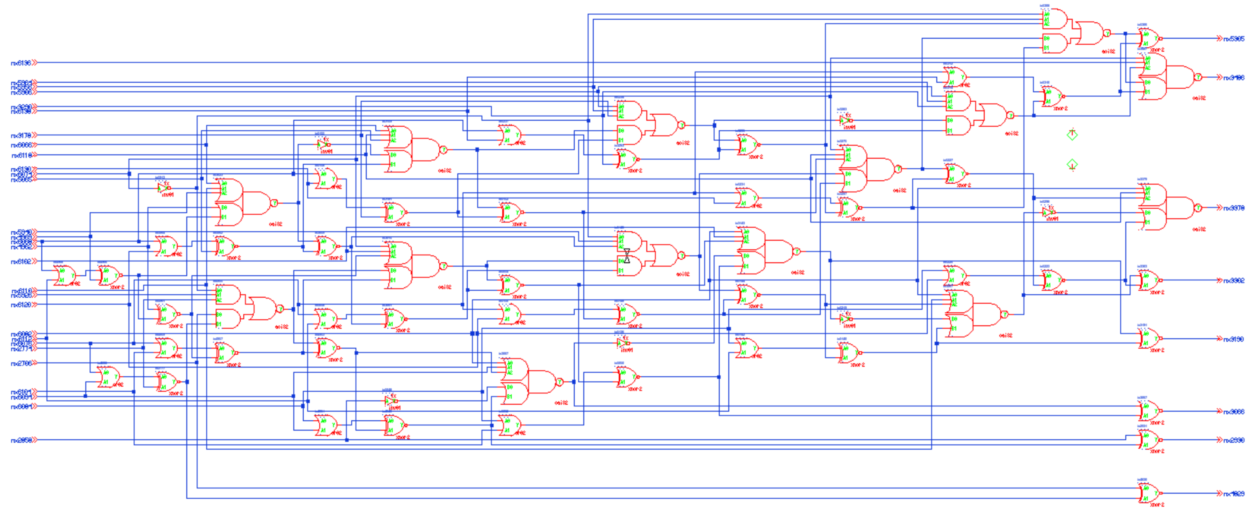


Figure 23: Full Schematic Page 2

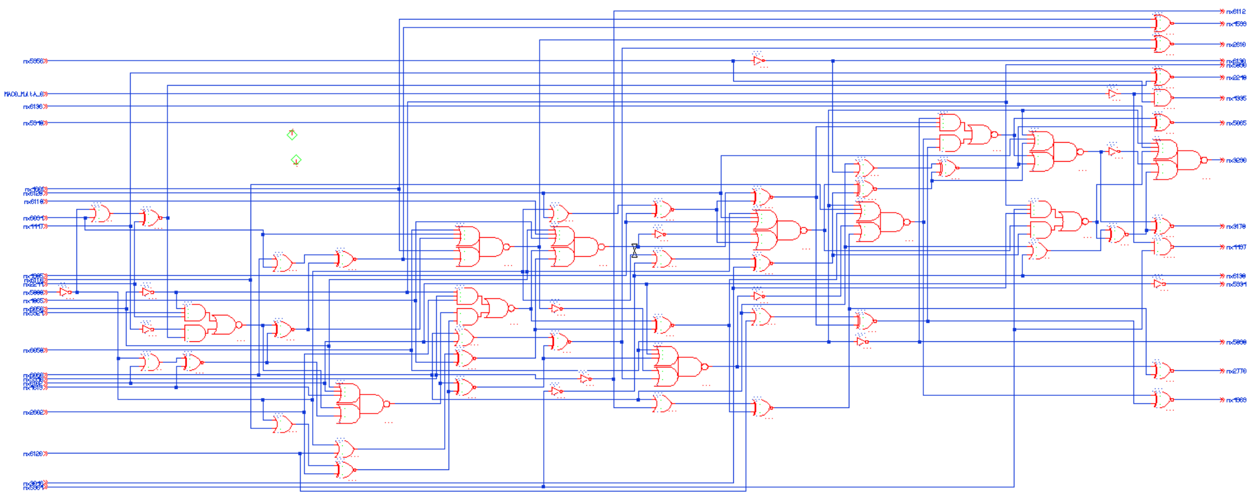


Figure 24: Full Schematic Page 3

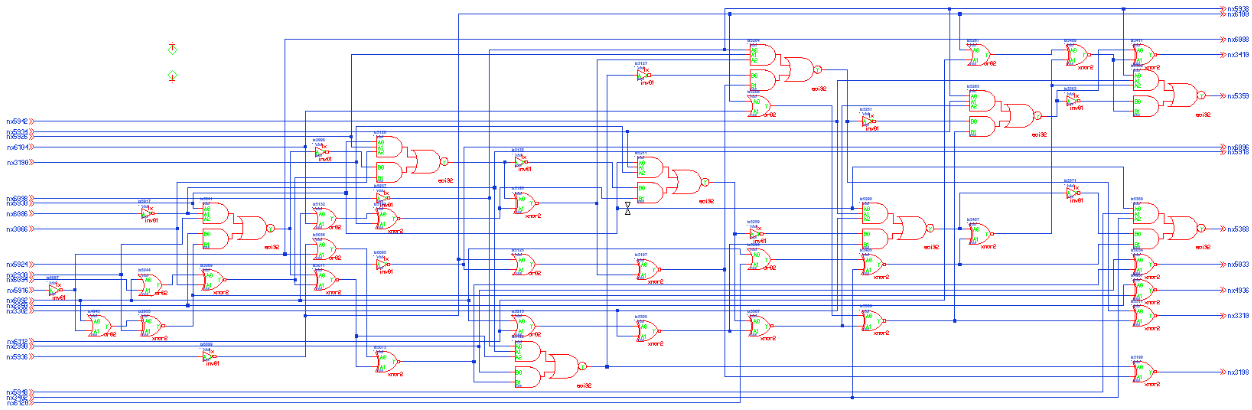


Figure 25: Full Schematic Page 4

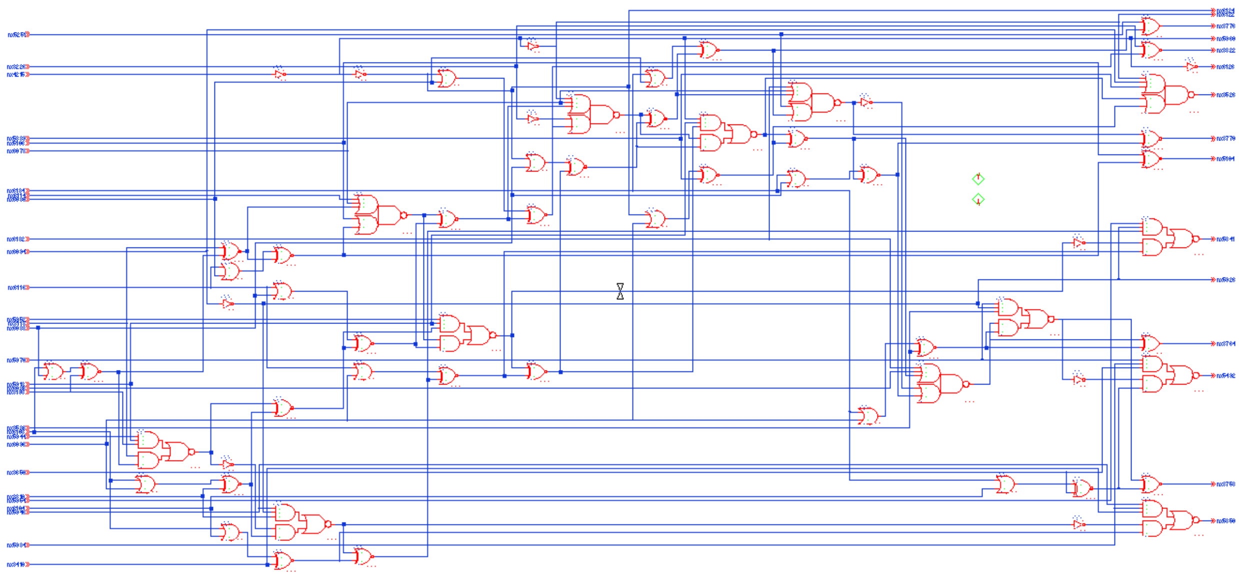


Figure 26: Full Schematic Page 5

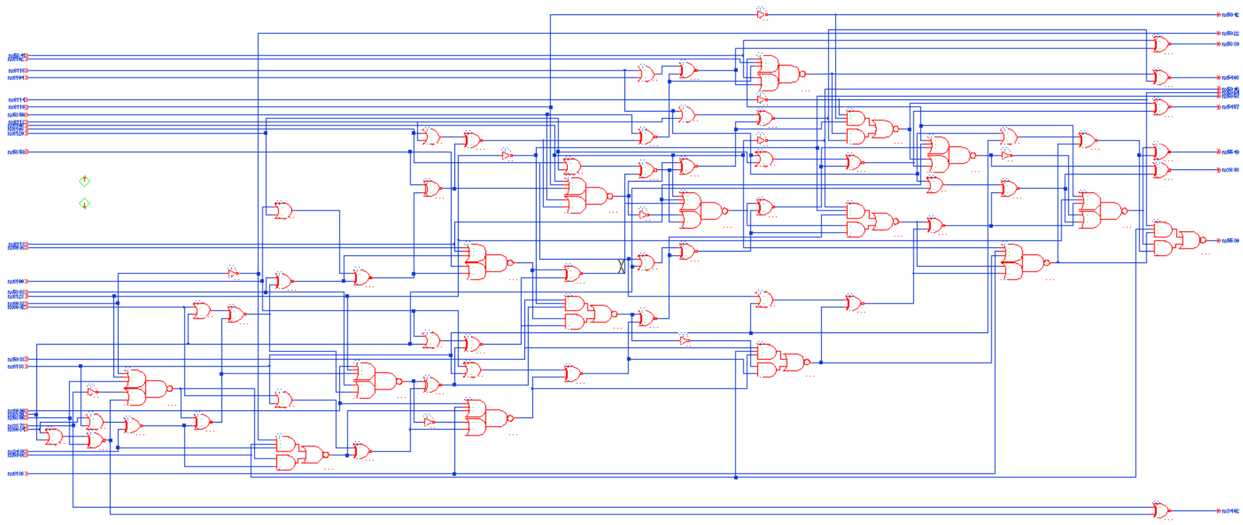


Figure 27: Full Schematic Page 6

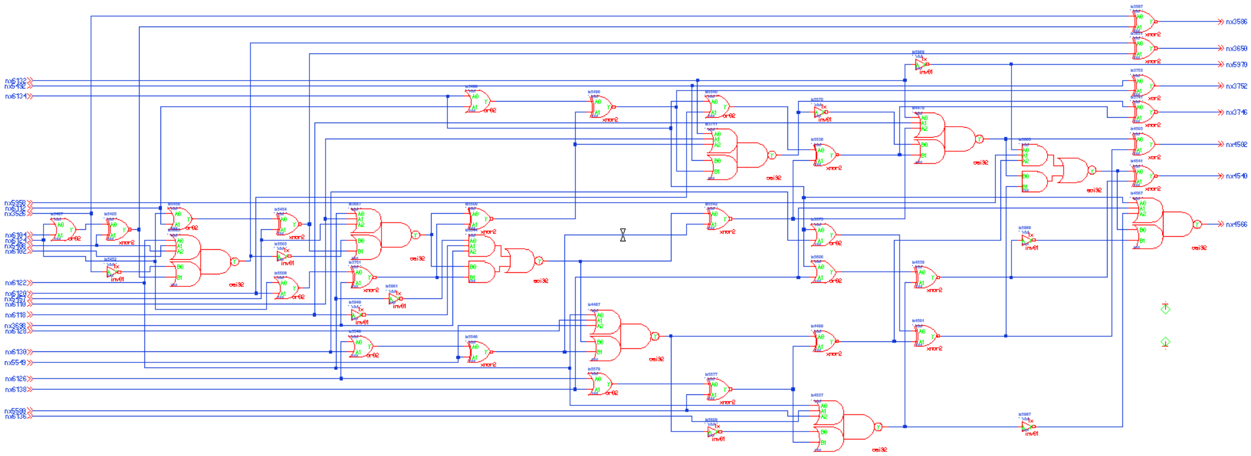


Figure 28: Full Schematic Page 7

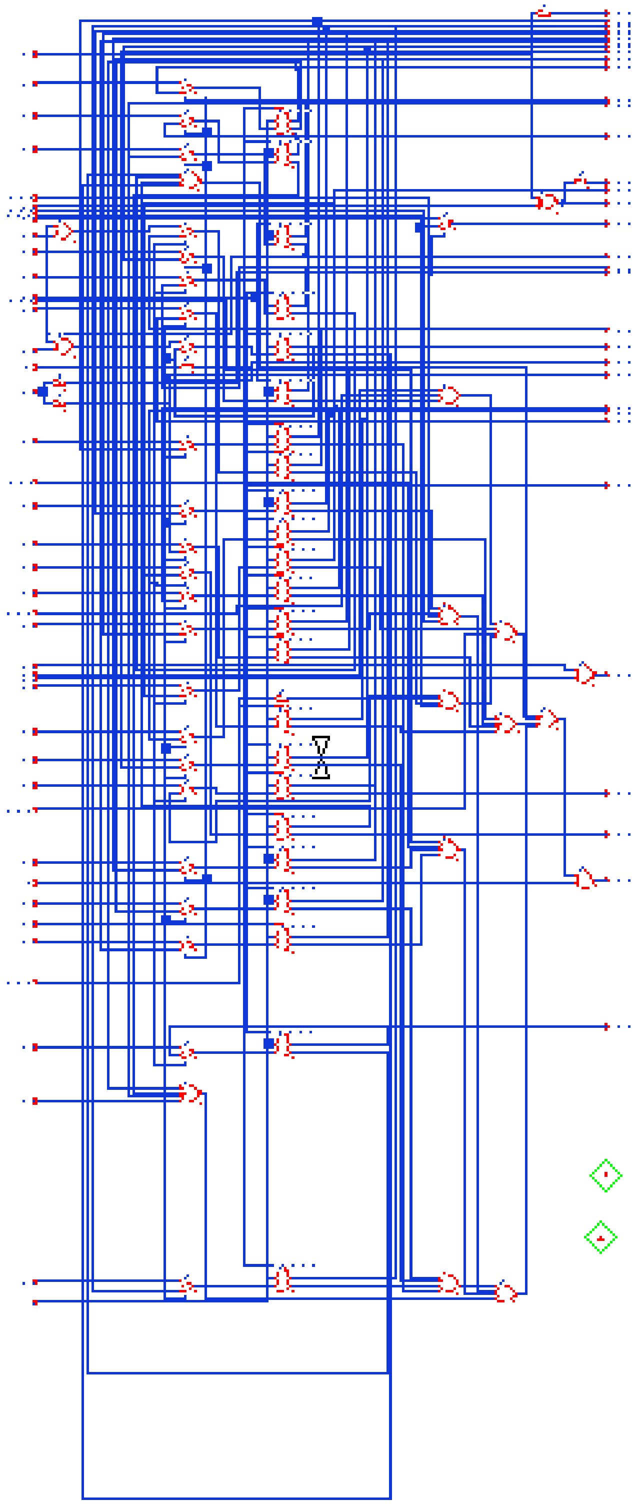


Figure 29: Full Schematic Page 8

-

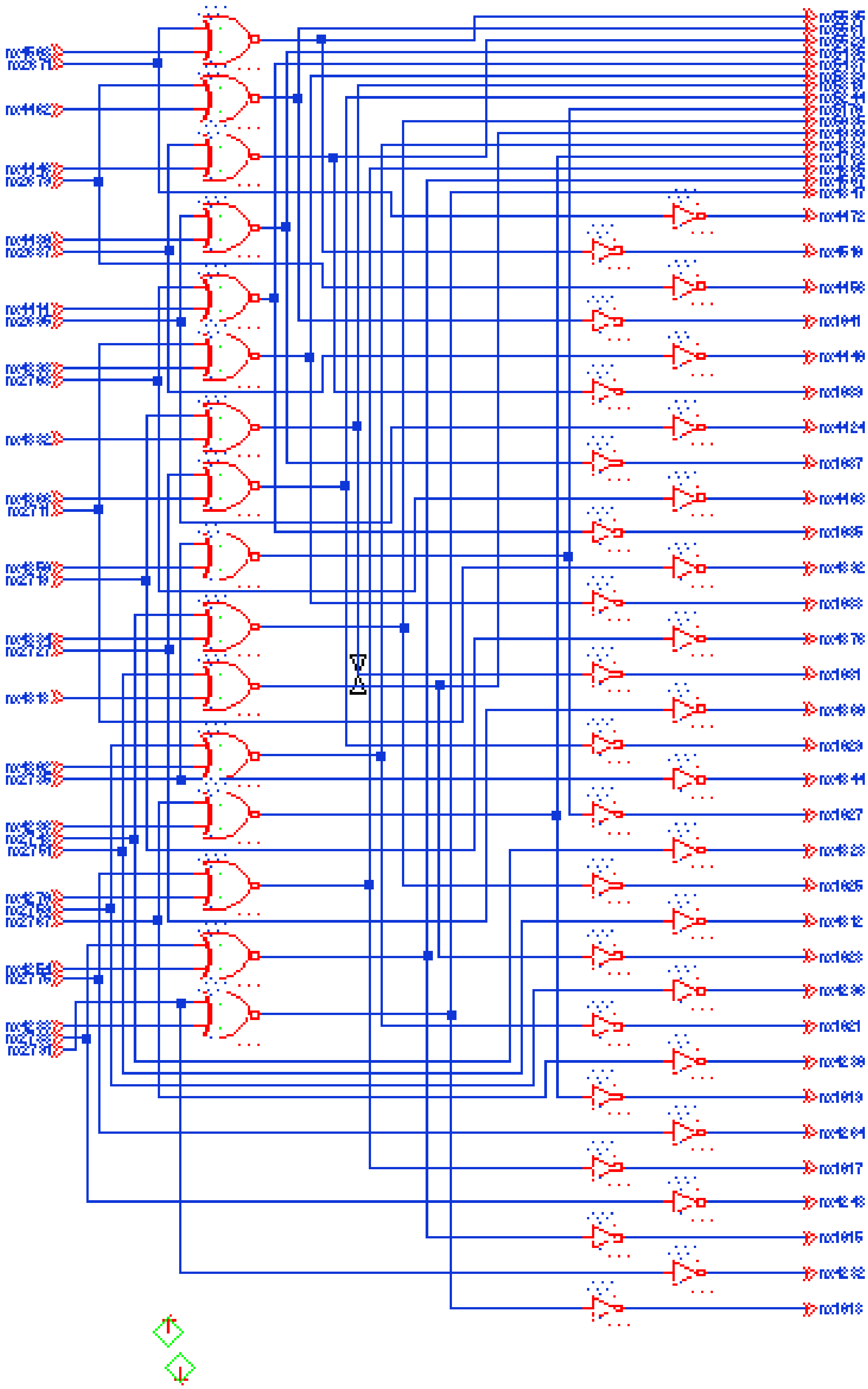


Figure 30: Full Schematic Page 9

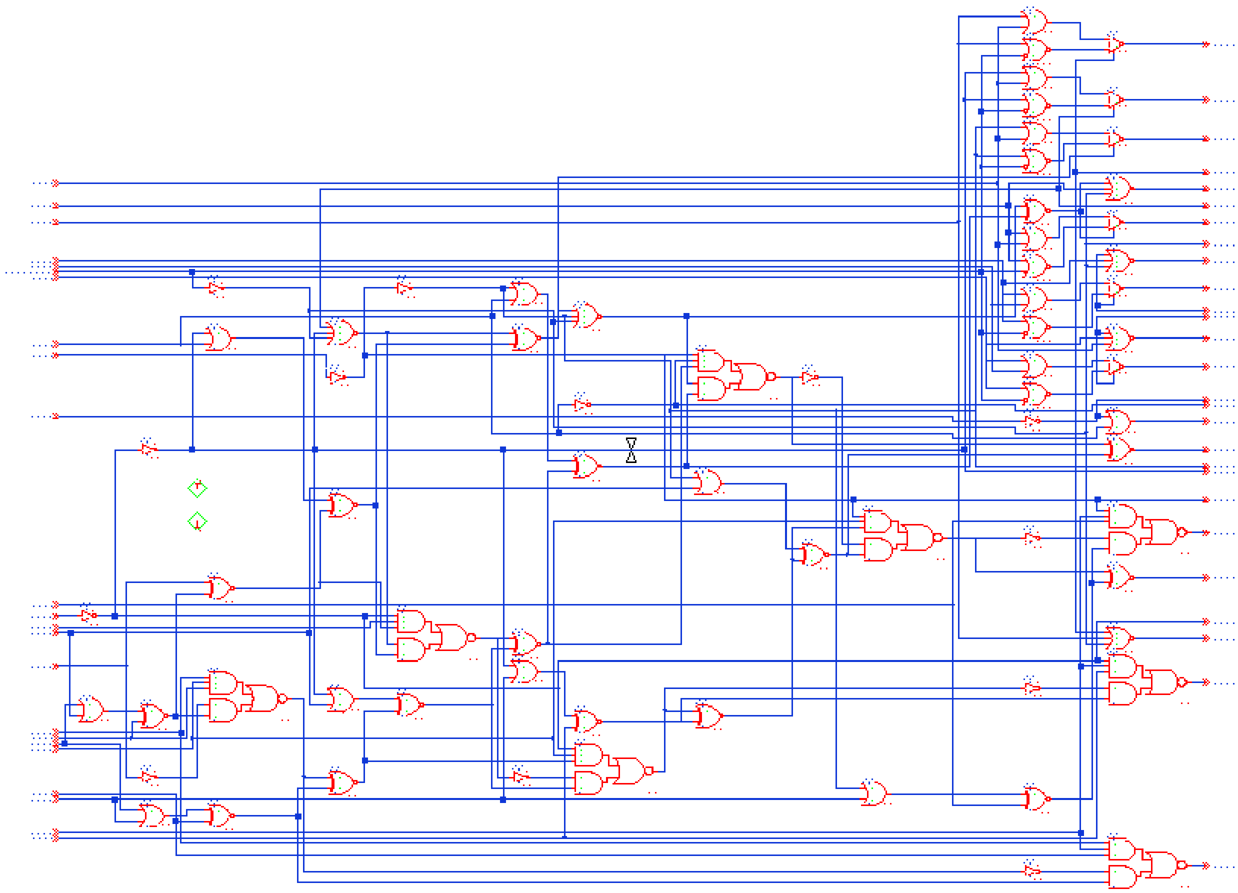


Figure 31: Full Schematic Page 10

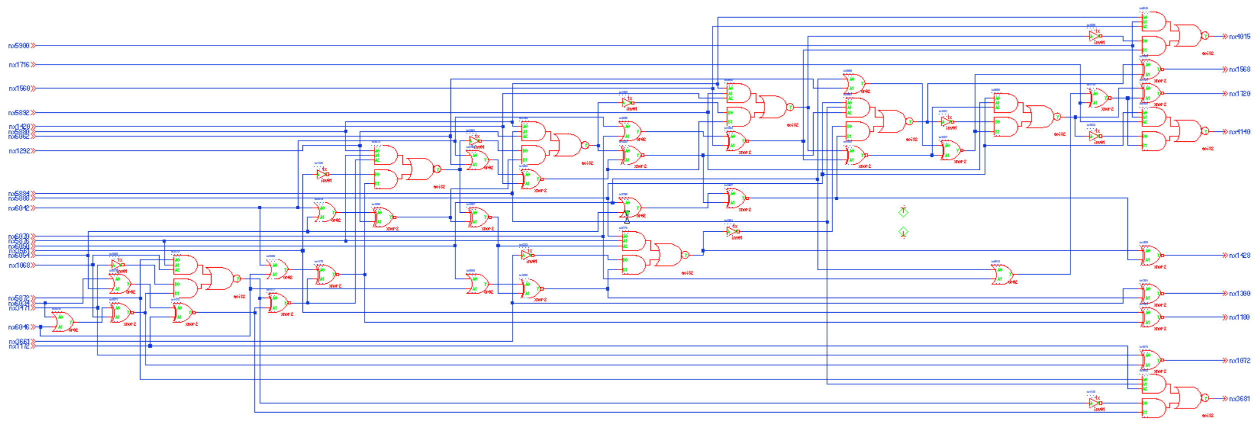


Figure 32: Full Schematic Page 11

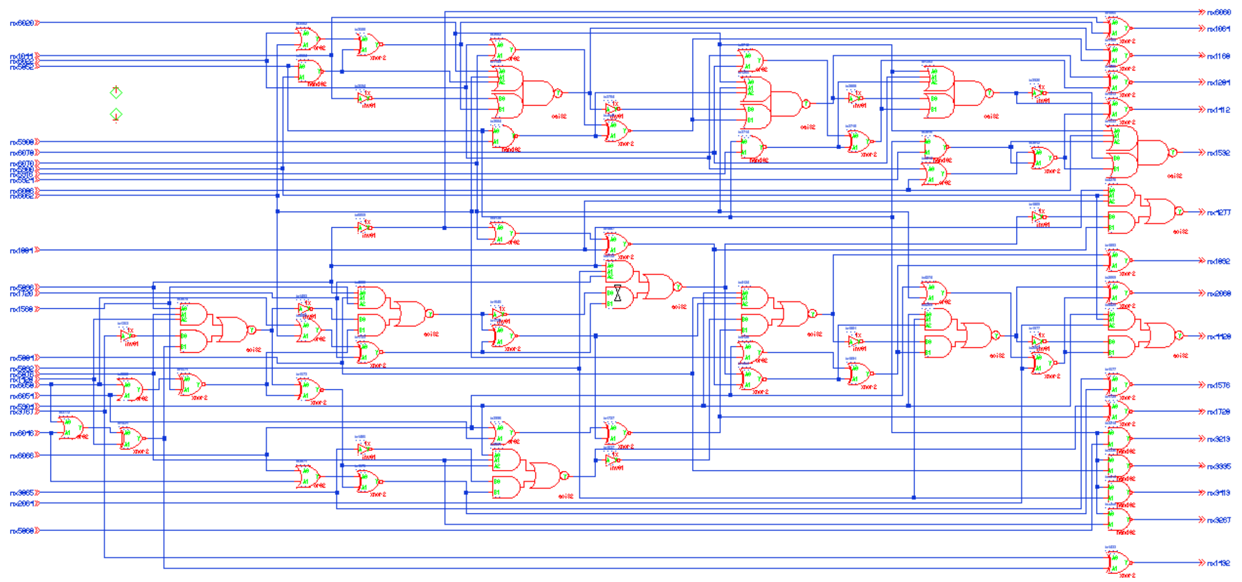


Figure 33: Full Schematic Page 12

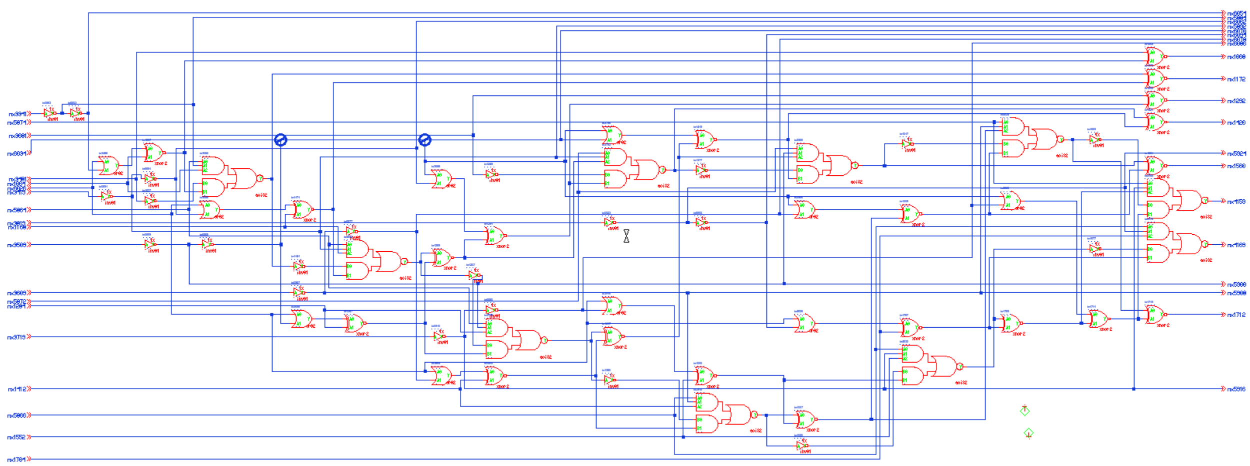


Figure 34: Full Schematic Page 13

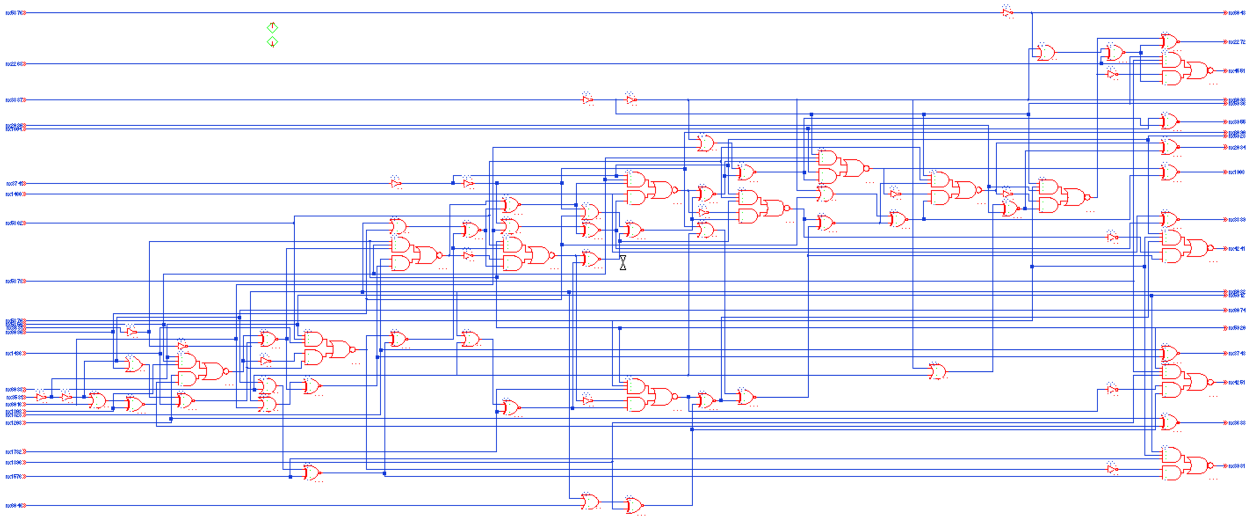


Figure 35: Full Schematic Page 14

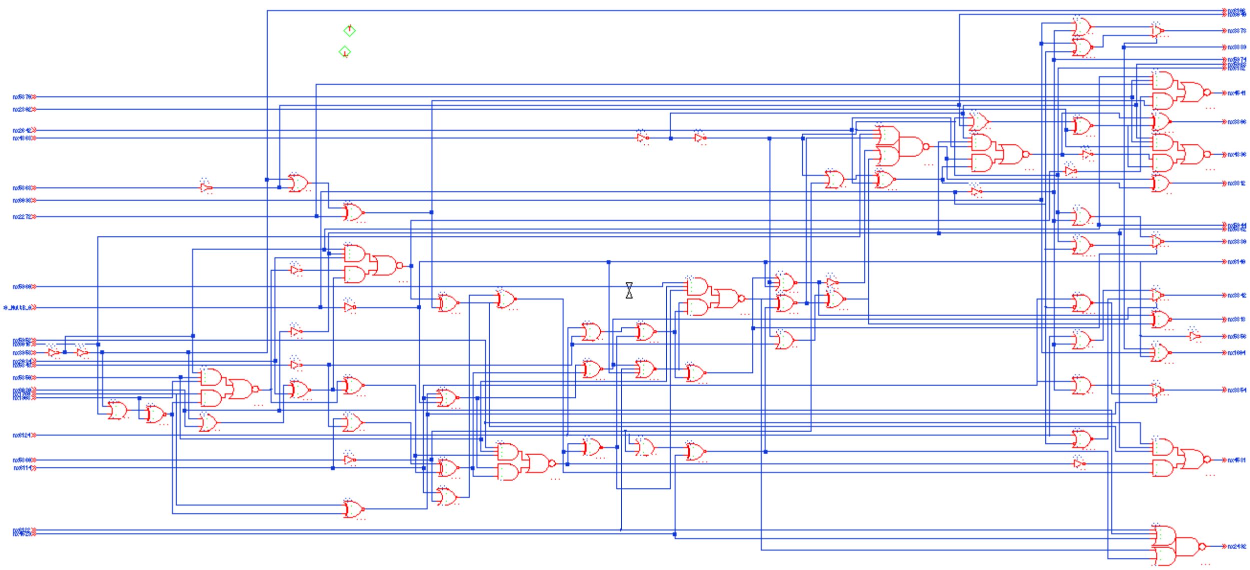


Figure 36: Full Schematic Page 15

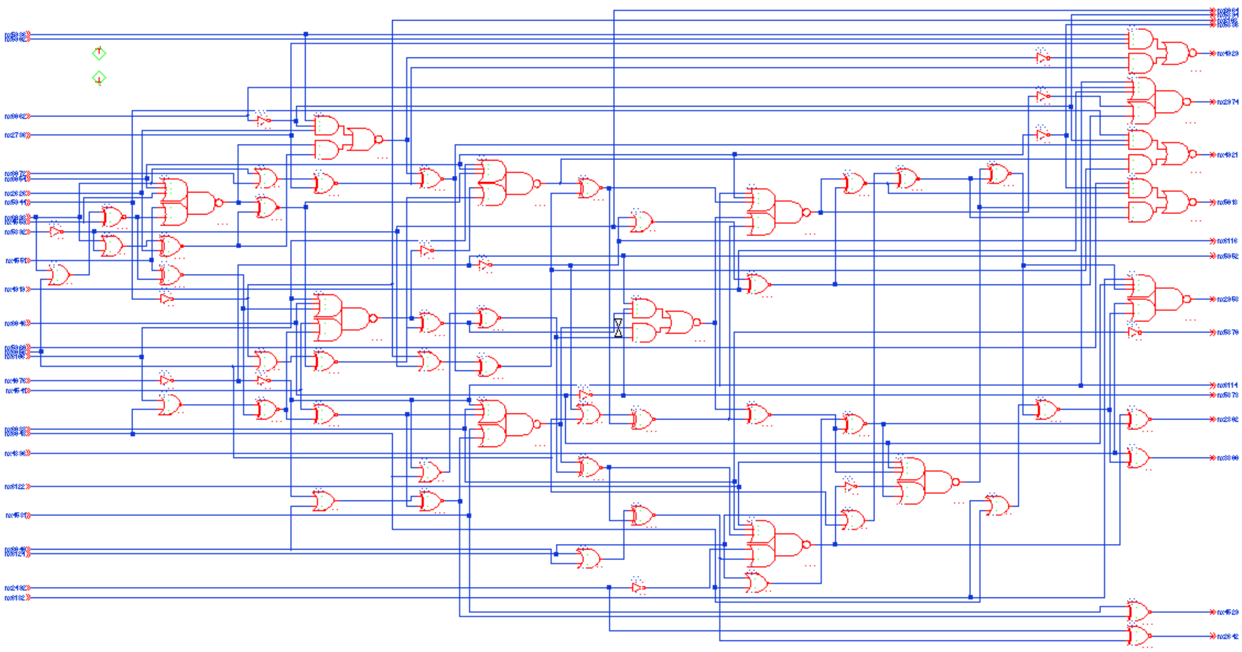


Figure 37: Full Schematic Page 16

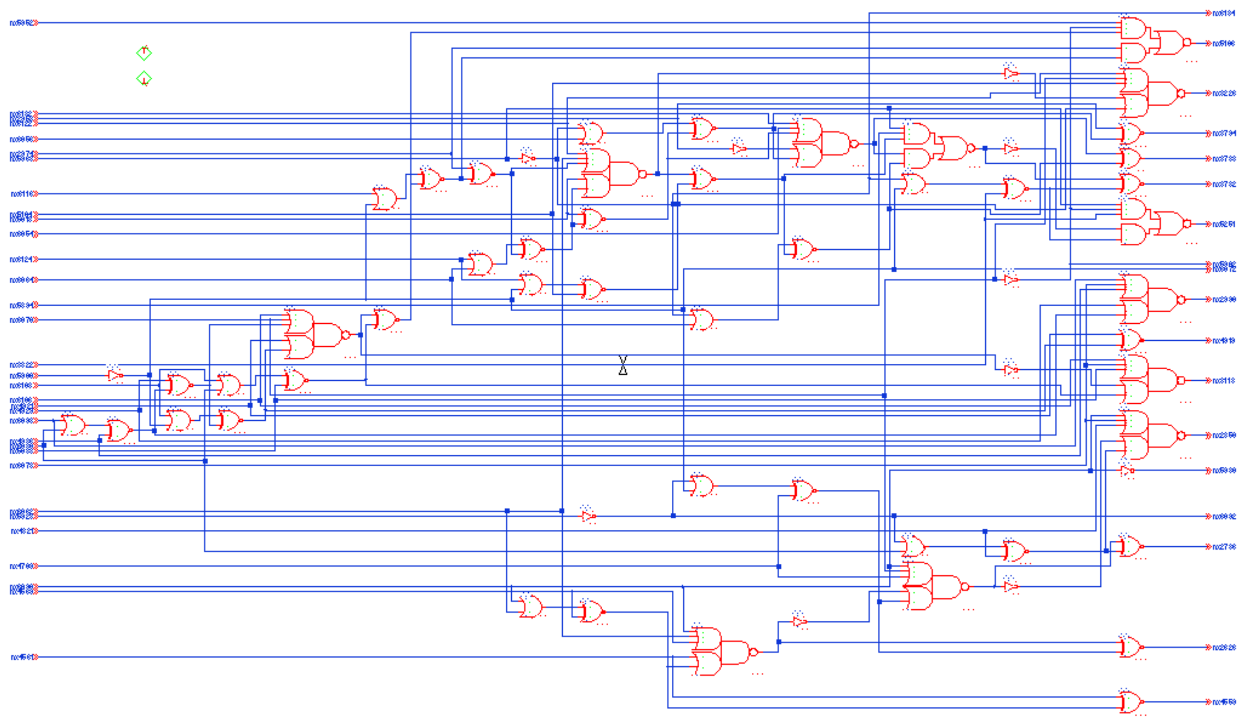


Figure 38: Full Schematic Page 17

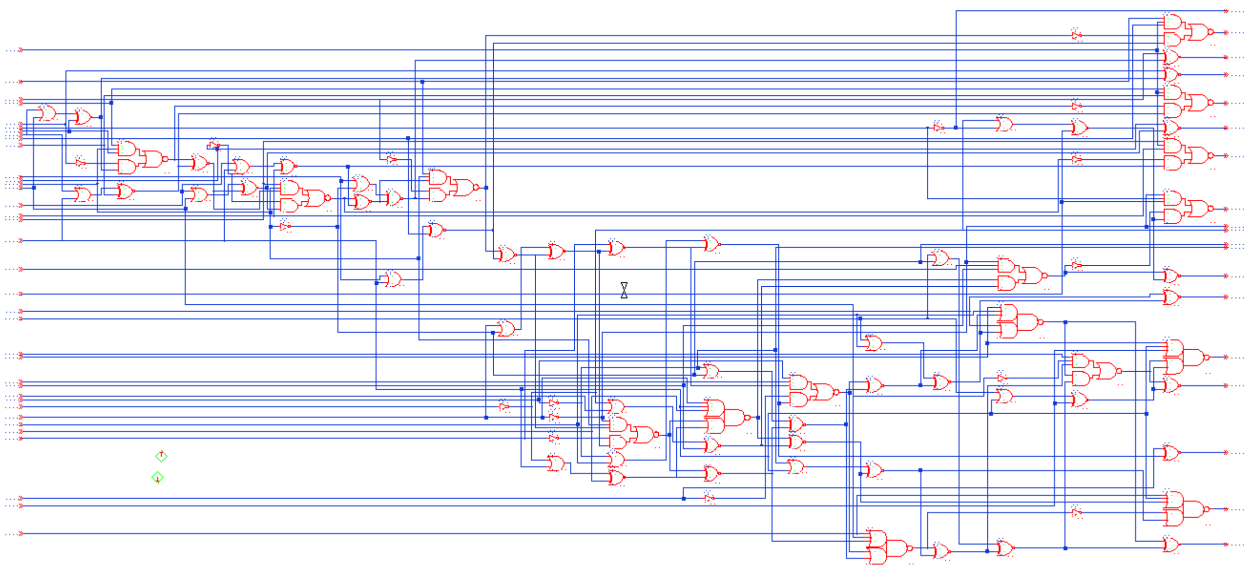


Figure 39: Full Schematic Page 18

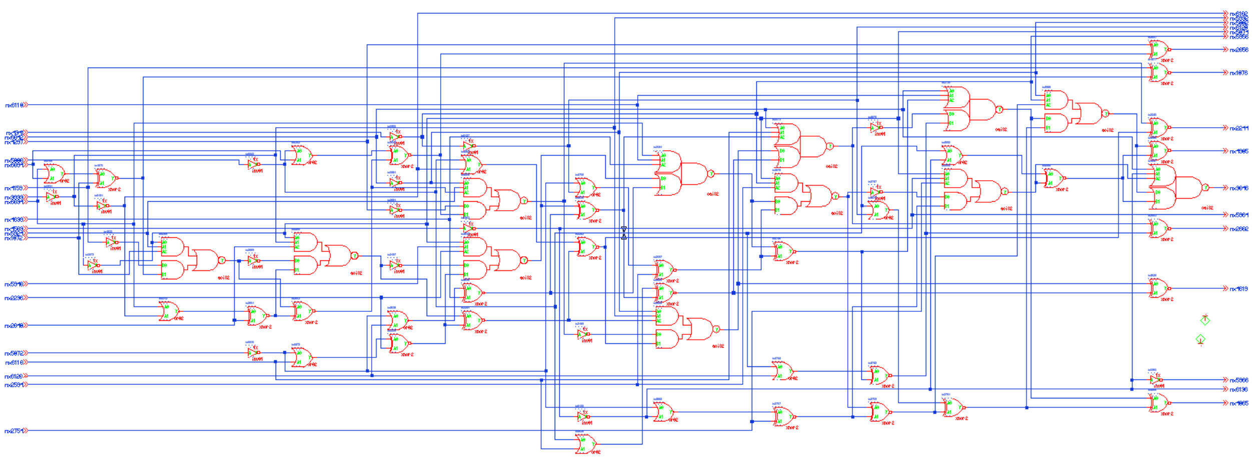


Figure 40: Full Schematic Page 19

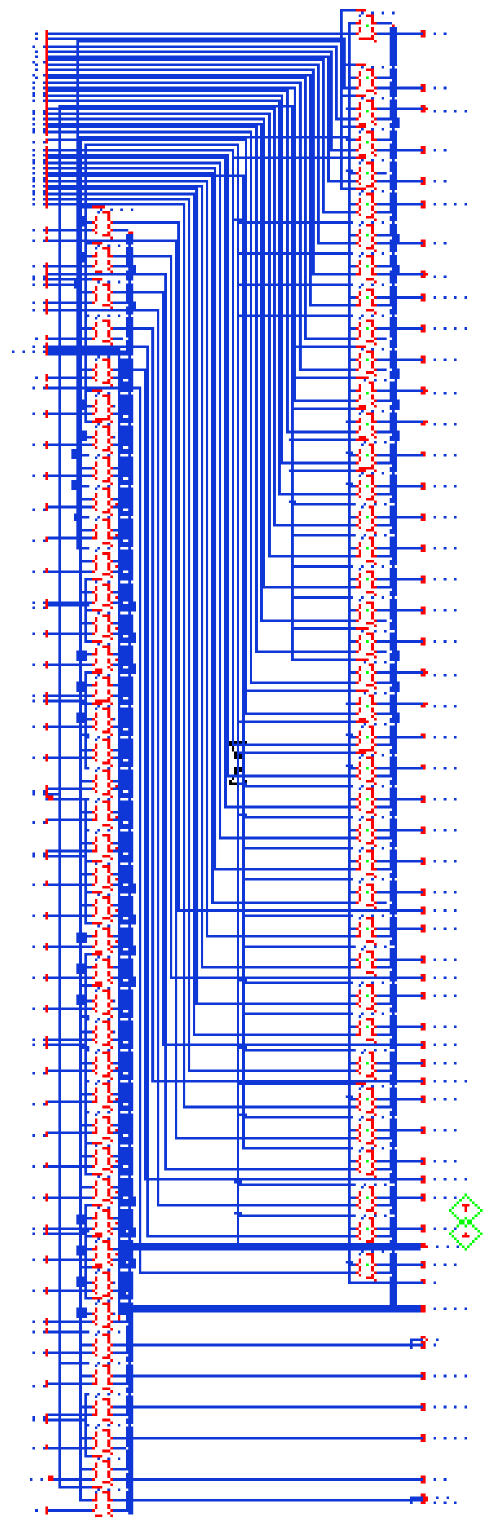


Figure 41: Full Schematic Page 20

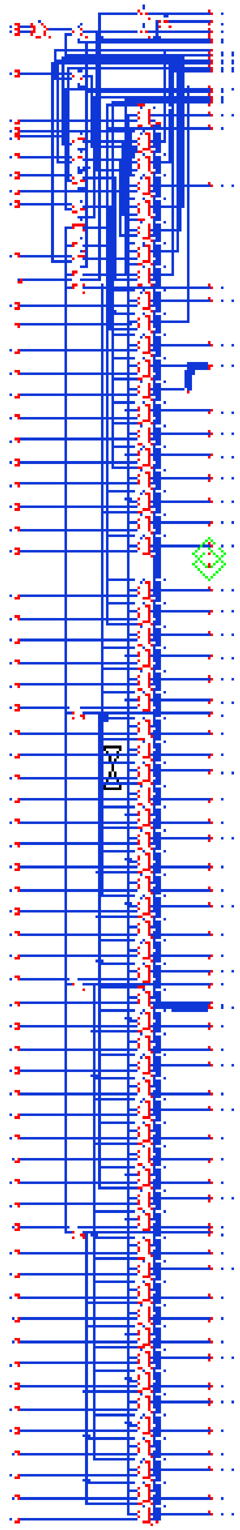


Figure 42: Full Schematic Page 21

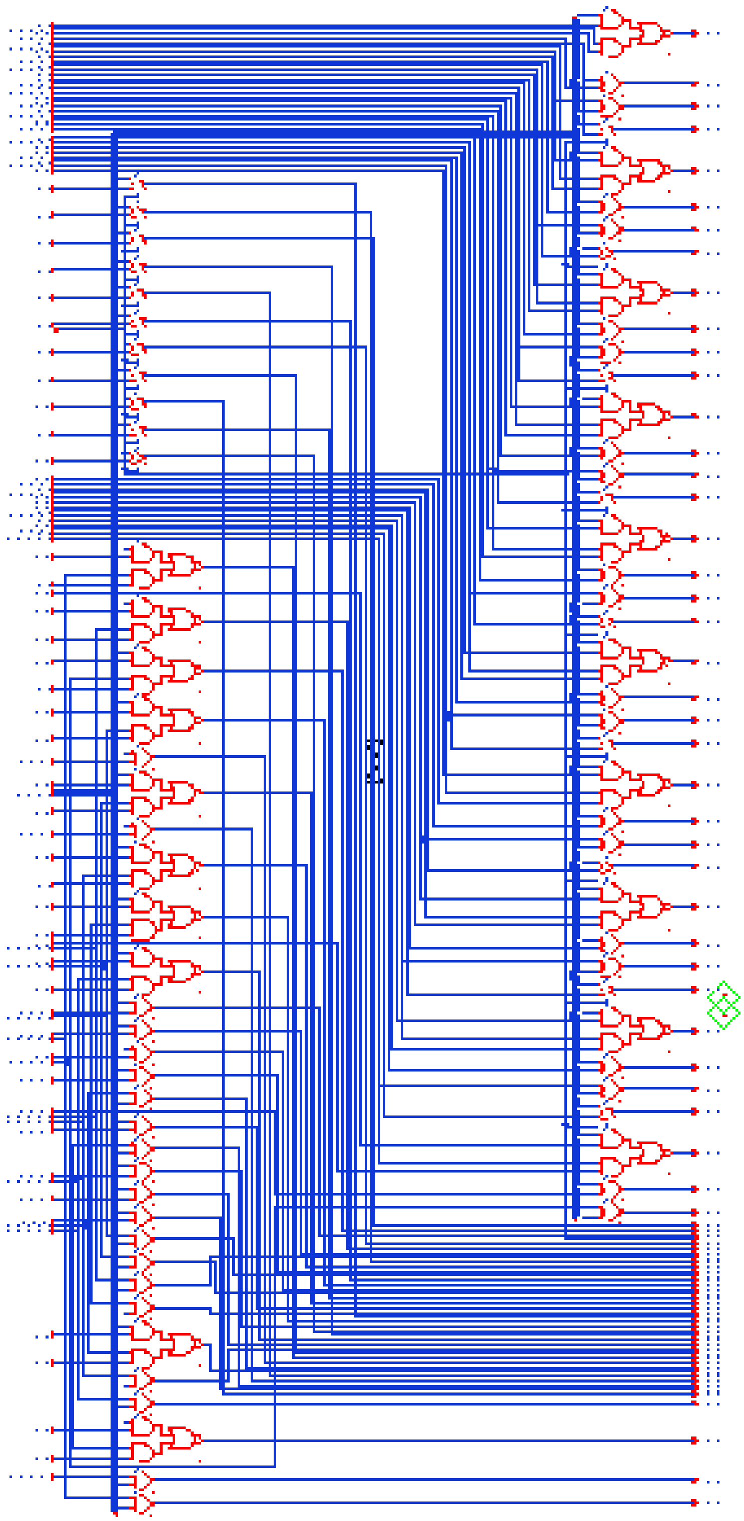


Figure 43: Full Schematic Page 22

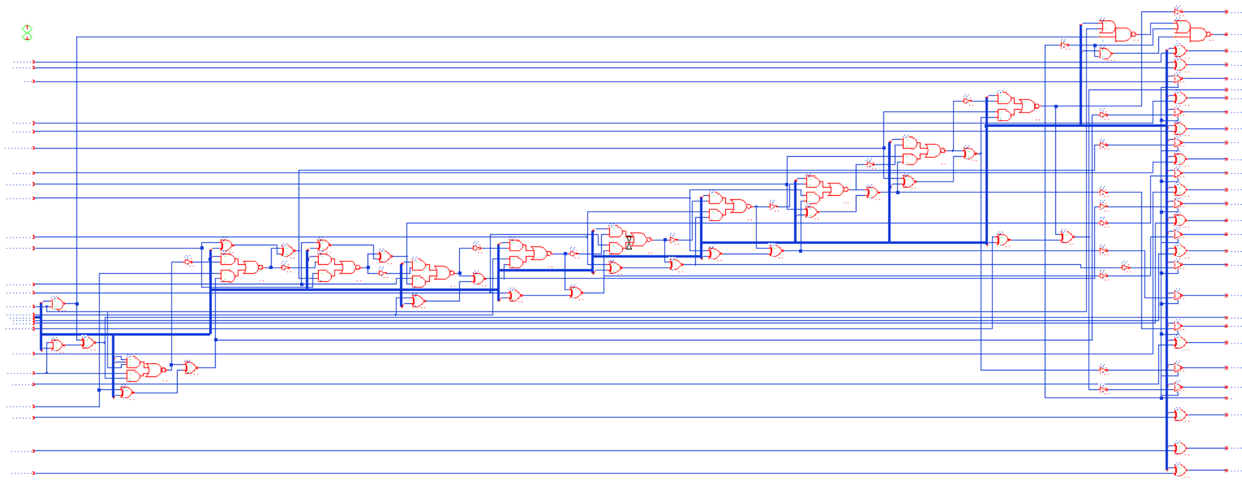


Figure 44: Full Schematic Page 23

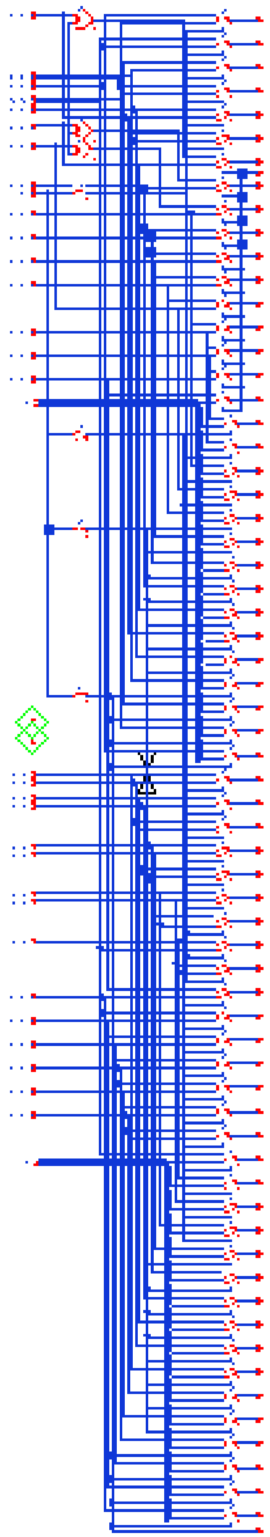


Figure 45: Full Schematic Page 24

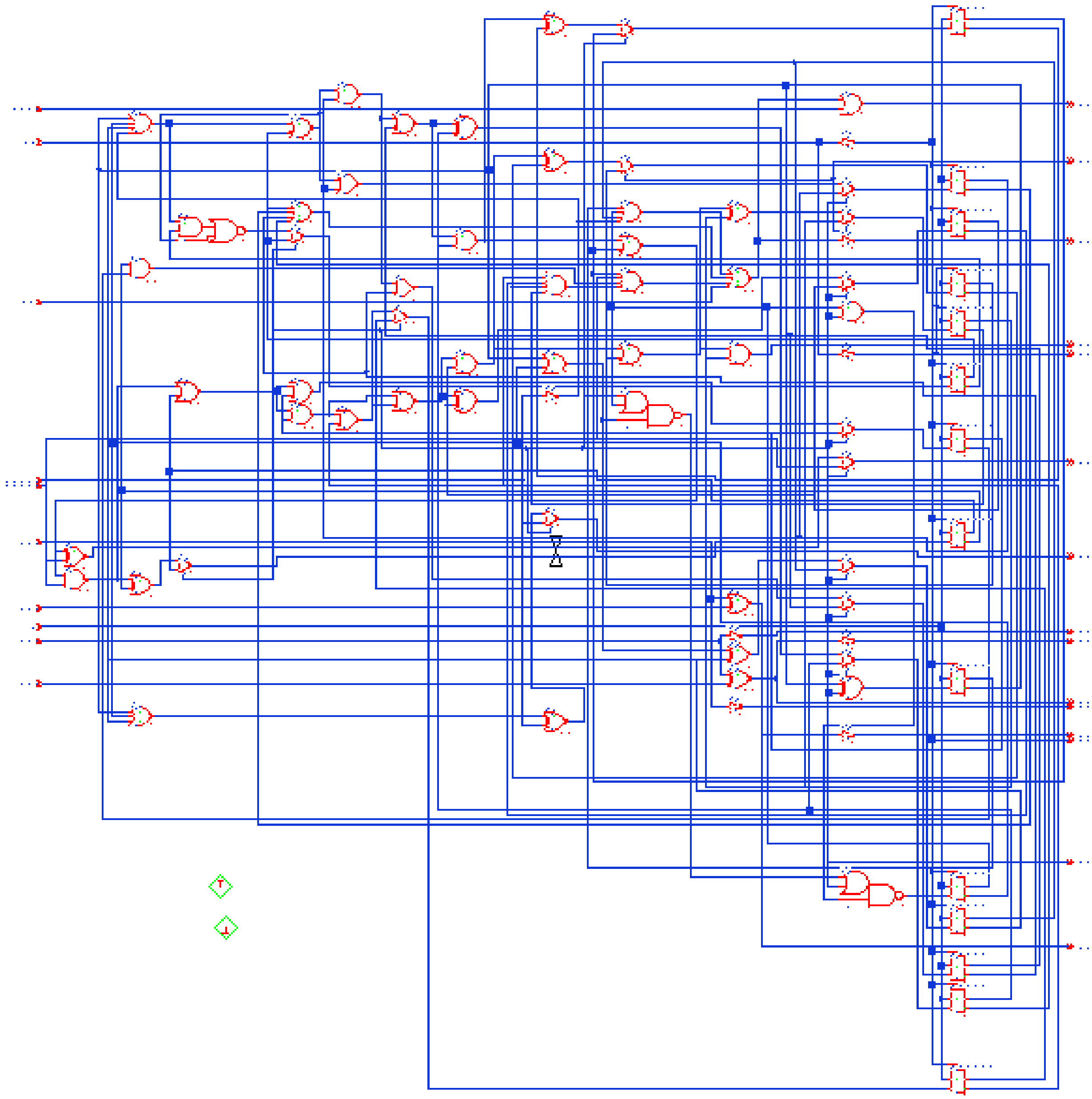


Figure 46: Full Schematic Page 25

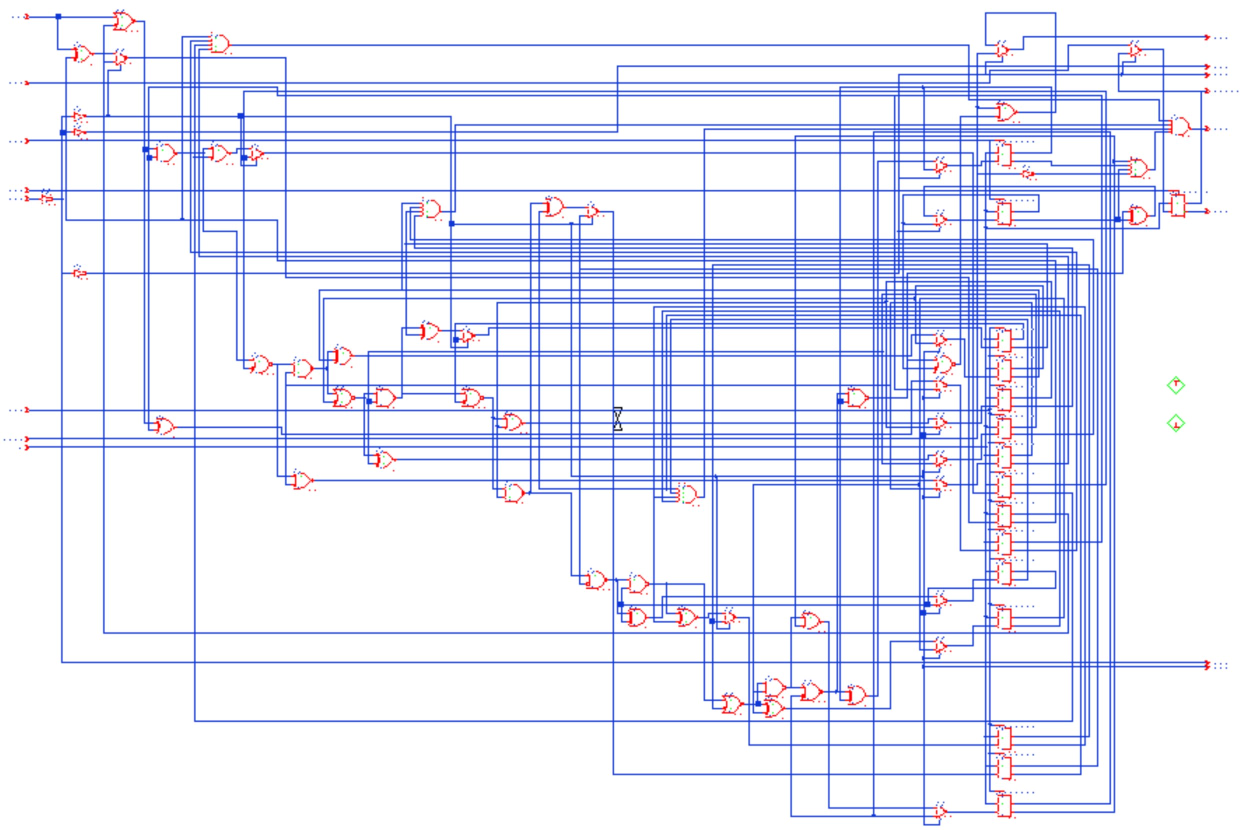


Figure 47: Full Schematic Page 26

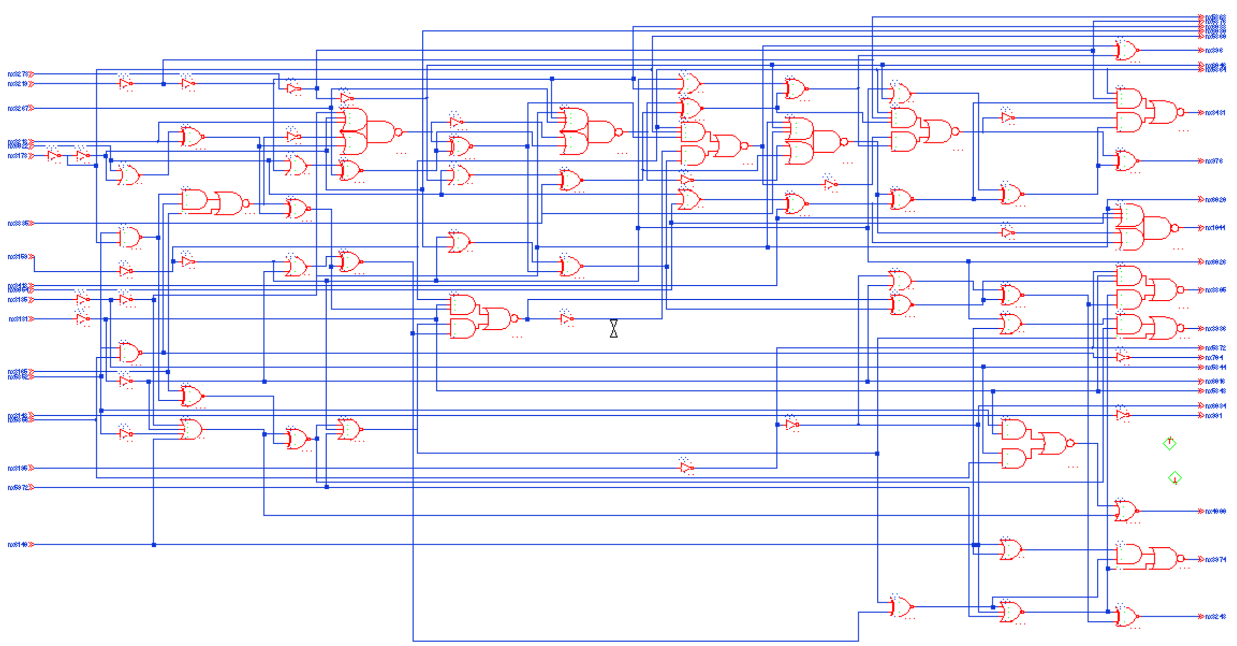


Figure 48: Full Schematic Page 27

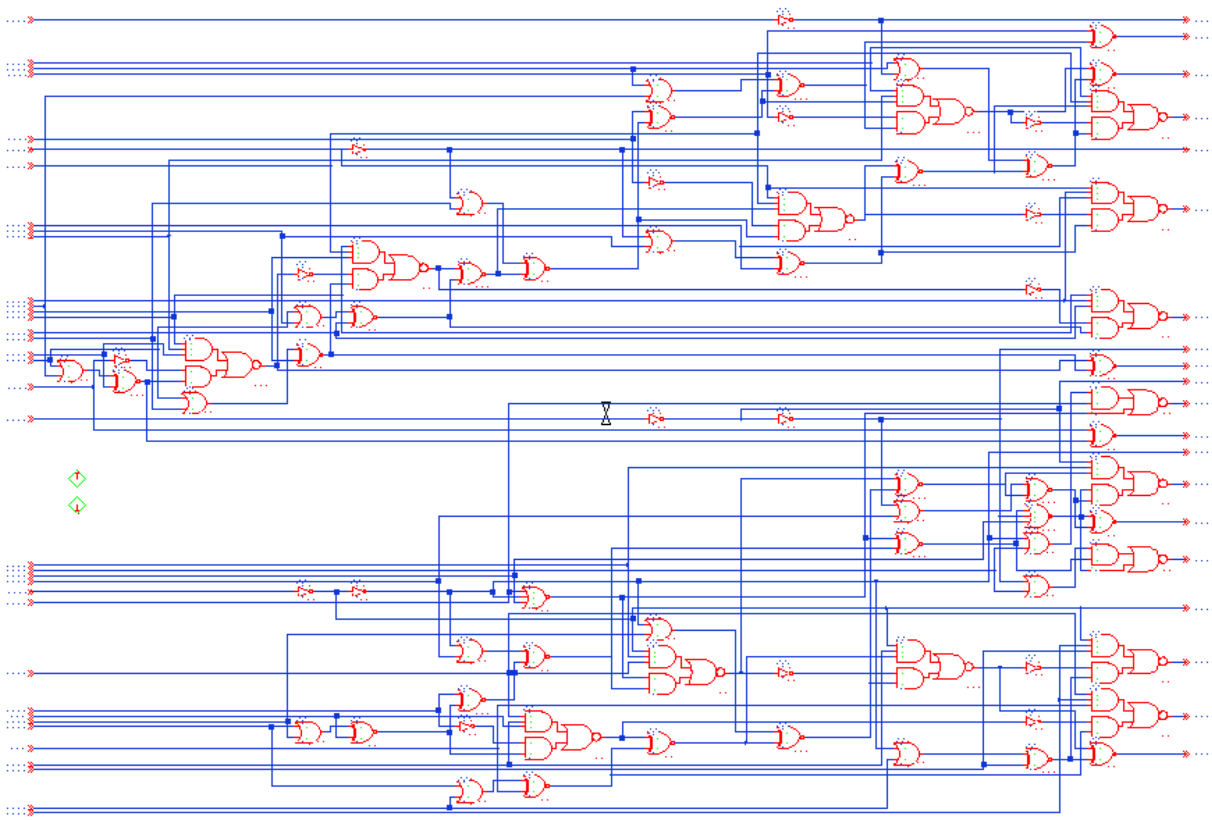


Figure 49: Full Schematic Page 28

3 Results and Analysis

The MAC was initially laid out structurally; components were laid out and turned into cells that would then be connected together. This was done to limit the complexity of the final design.

3.1 Components

The full adder was laid out first. The resulting layout can be seen in Figure 50.

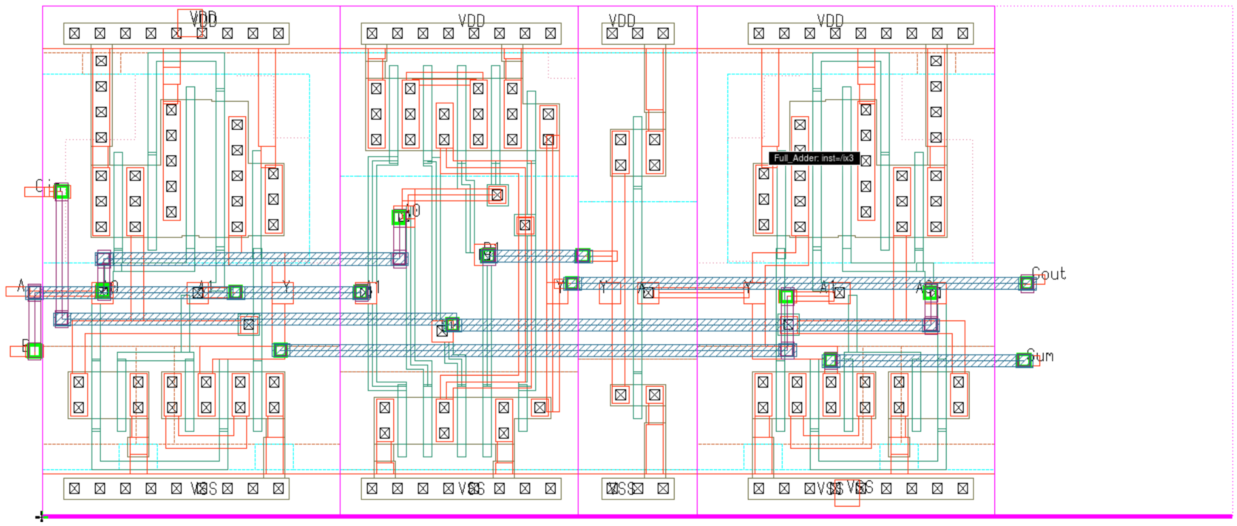


Figure 50: Full Adder Layout

The multiplier was a complex component so the feasibility of layout was questioned early. It turns out that giving appropriate area to run wires make routing the multiplier easy. The results can be seen in Figure 51.

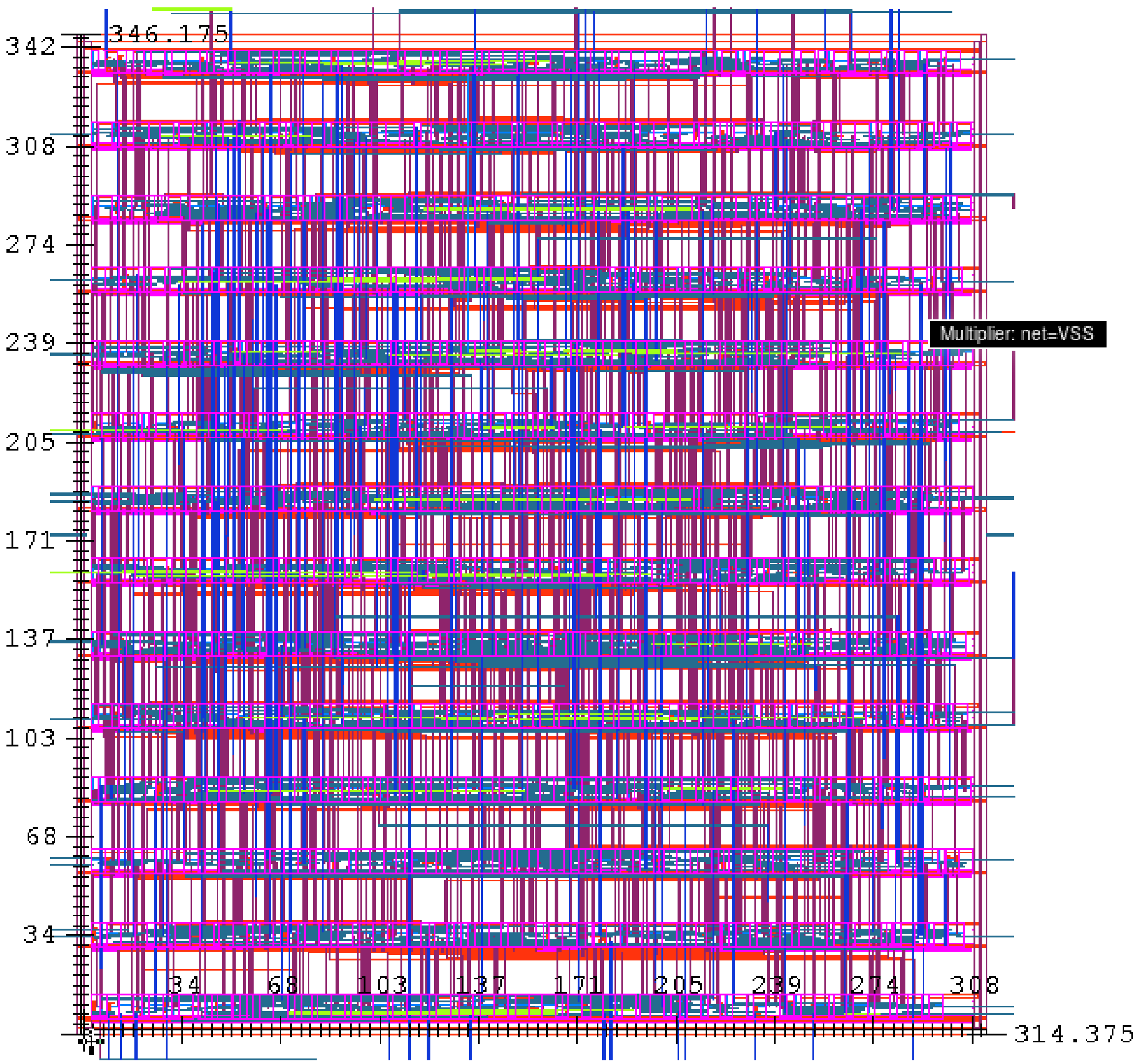


Figure 51: Multiplier Layout

The inputs of the multiplier needed to be controlled, so the 16-bit register was laid out and captured in Figure 52.

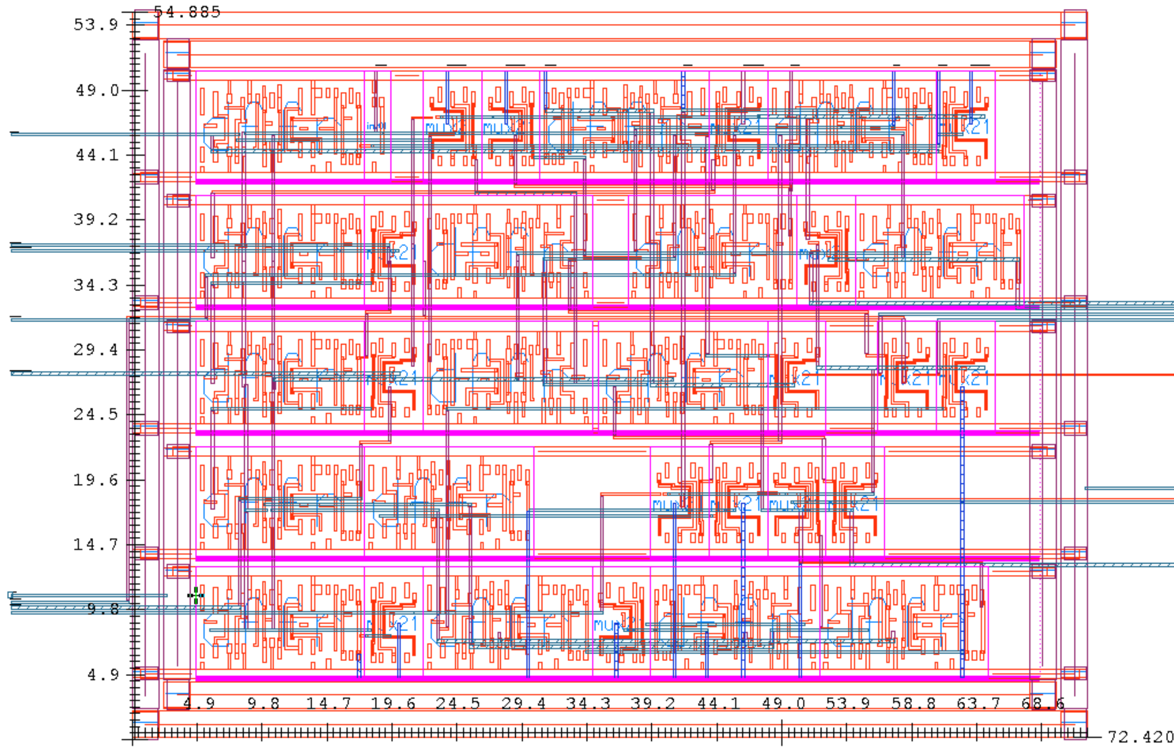


Figure 52: 16 Bit Register Layout

The accumulator register had layout performed next. The circuit is illustrated in Figure 53.

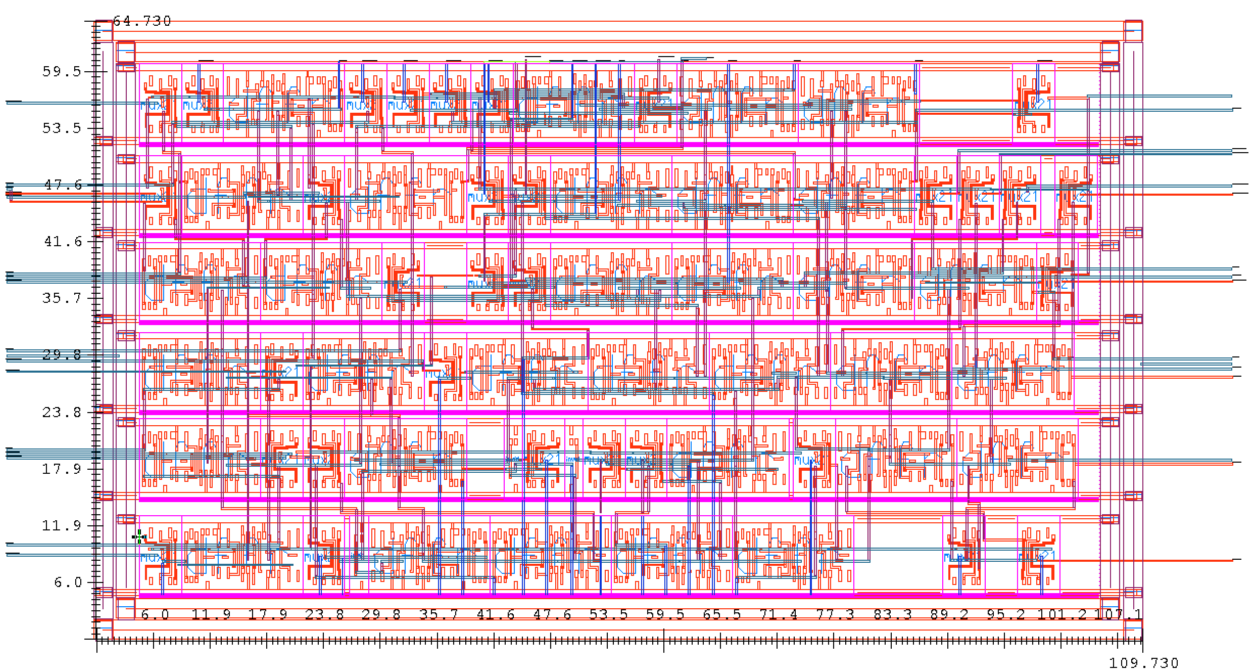


Figure 53: Accumulator Layout

A multiplexer was needed to switch between test input and user input, so it was laid out after the rest of the components. The result can be seen in Figure 54.

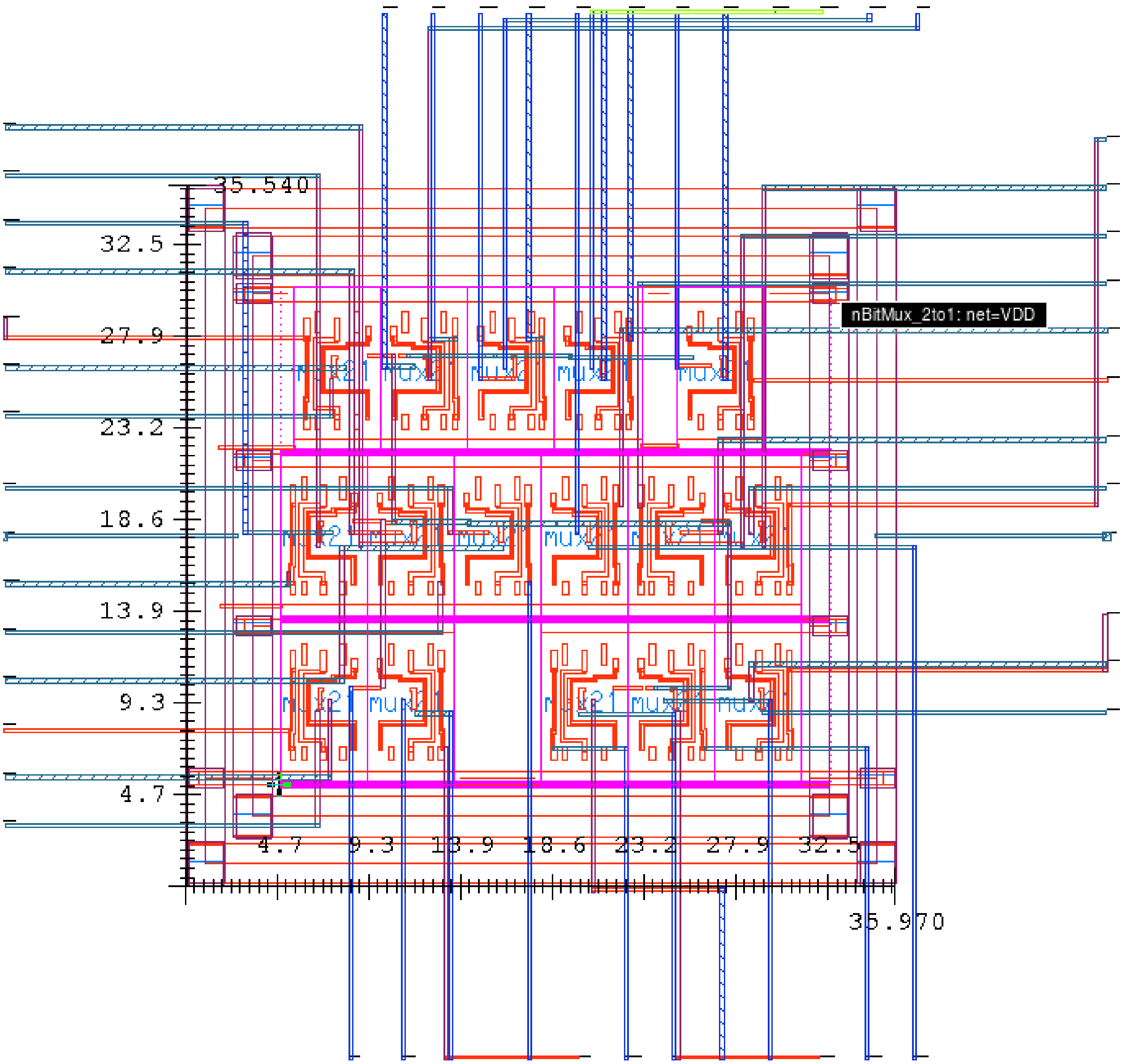


Figure 54: 32 Bit Mux 2to1 Layout

3.2 MAC with BIST Layout

It was found that the circuit could not be constructed structurally with the provided tools. Instead, the circuit was laid out in one go. In order to make the layout possible, many settings were adjusted. The circuit was first auto-instantiated(Figure 55).

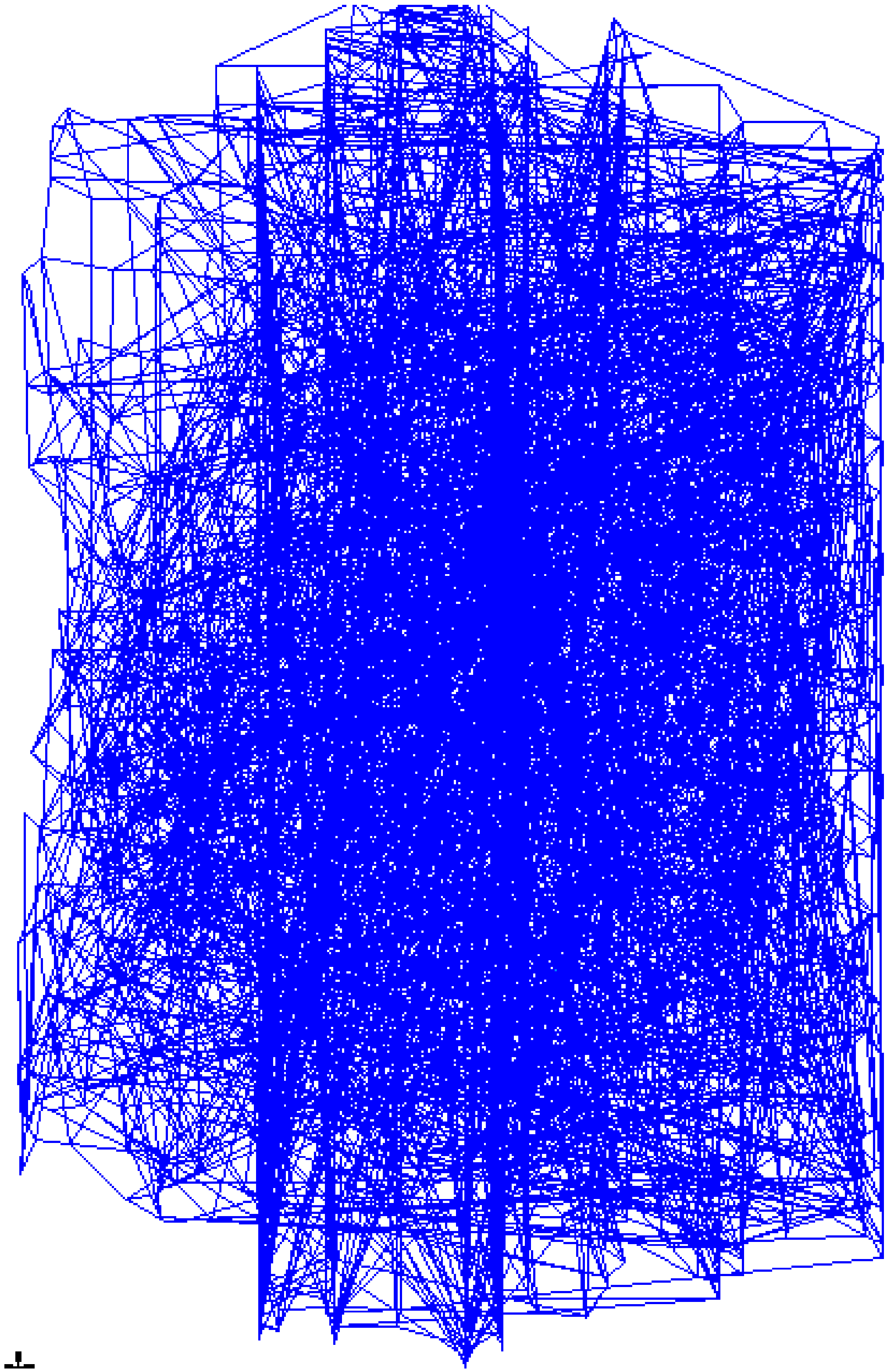


Figure 55: Pre-Layout

The instantiation could be best described as spaghetti. To organize this pasta, floor-planning was performed. To ensure enough area for routing wires, the area was defined to be 2 when performing the floor plan. Next, standard cells were placed. The cells were initially placed with "random+improve" and optimize. A second cell placement was performed with "initial+improve" and optimize. The second cell placement greatly clean up the circuit as seen in Figure 56.

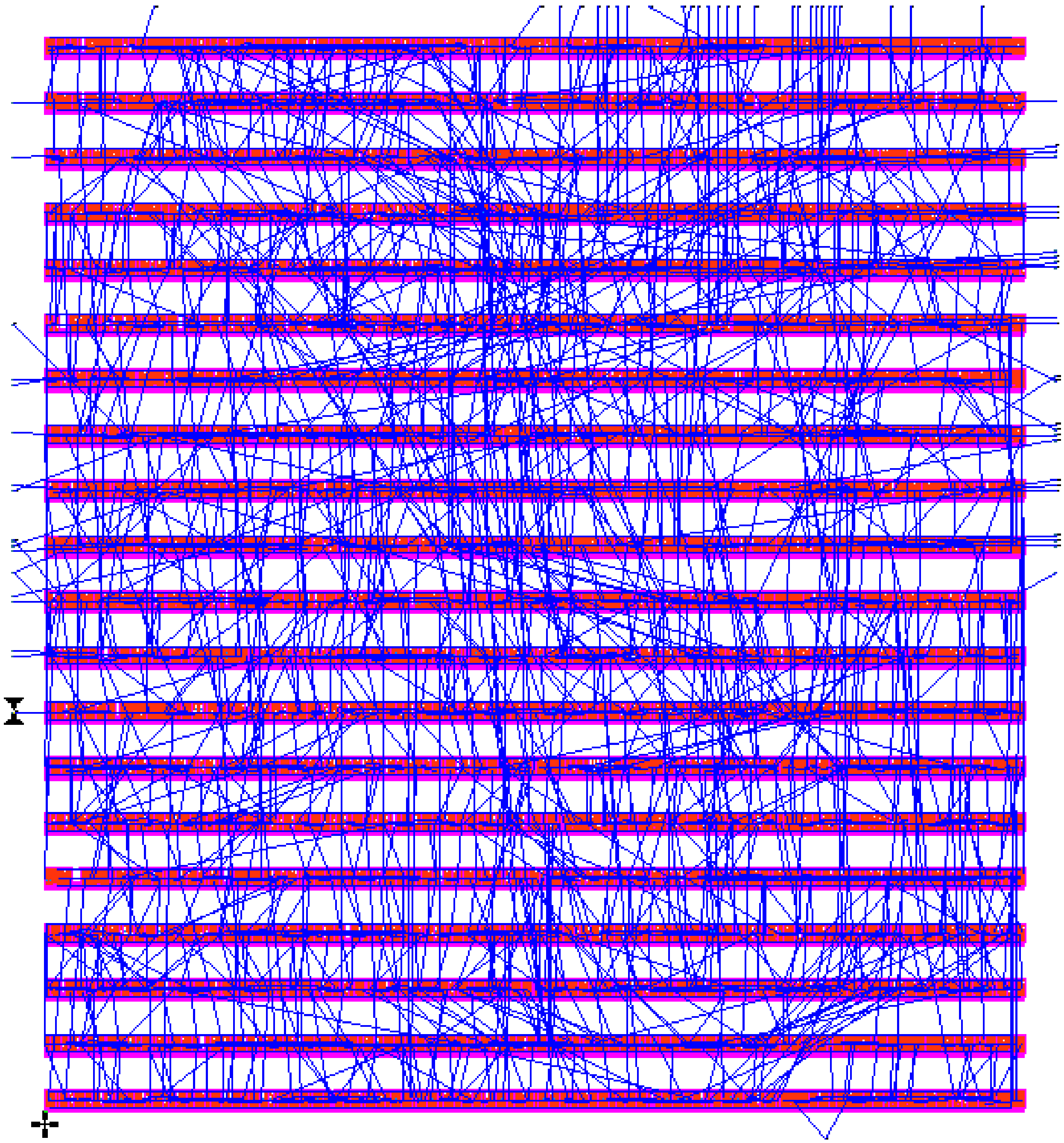


Figure 56: Standard Cells Placement

After cells placement, ports were placed as close as possible to their sources. Power routing was performed next and the result was recorded in Figure 57.

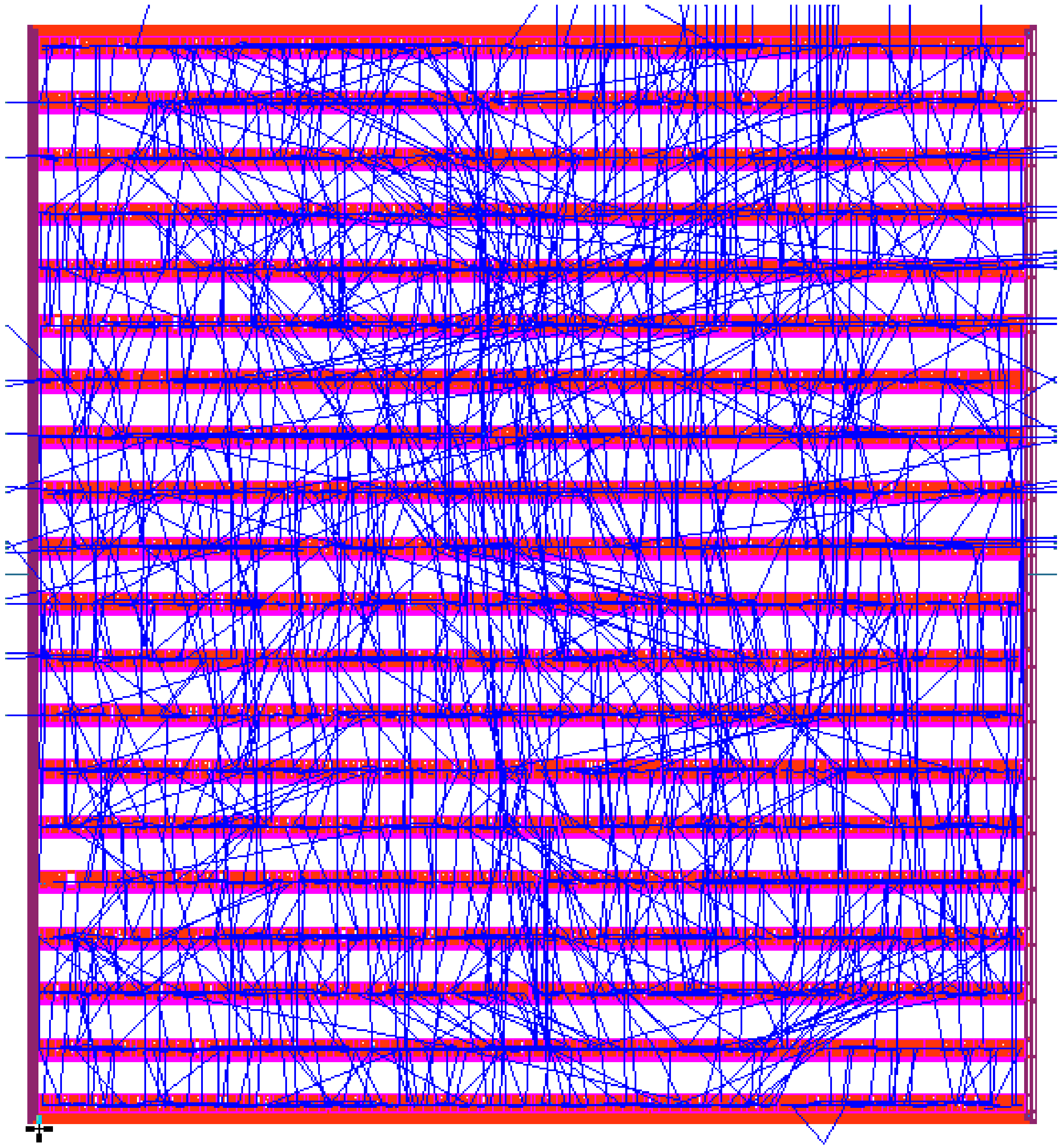


Figure 57: Power Route

Once power routing was finished, signal routing was performed. Auto-route settings were adjusted to not route with poly silicon as this tends to cause stray gates. Additionally, the following settings were applied to aid in auto routing:

- Varying levels of routing completion time
- Slight preference for jogs over via to fill the area.

- Rip
- Under rip options:

  Rips Most Aggressive

  Automatic Rip Passes

  Reroute
- Under Advanced:

  Allow all directions for stubs

  Via Options > Use via generator

Many attempts to route were performed. One of the many failures can be seen inf Figure 58.

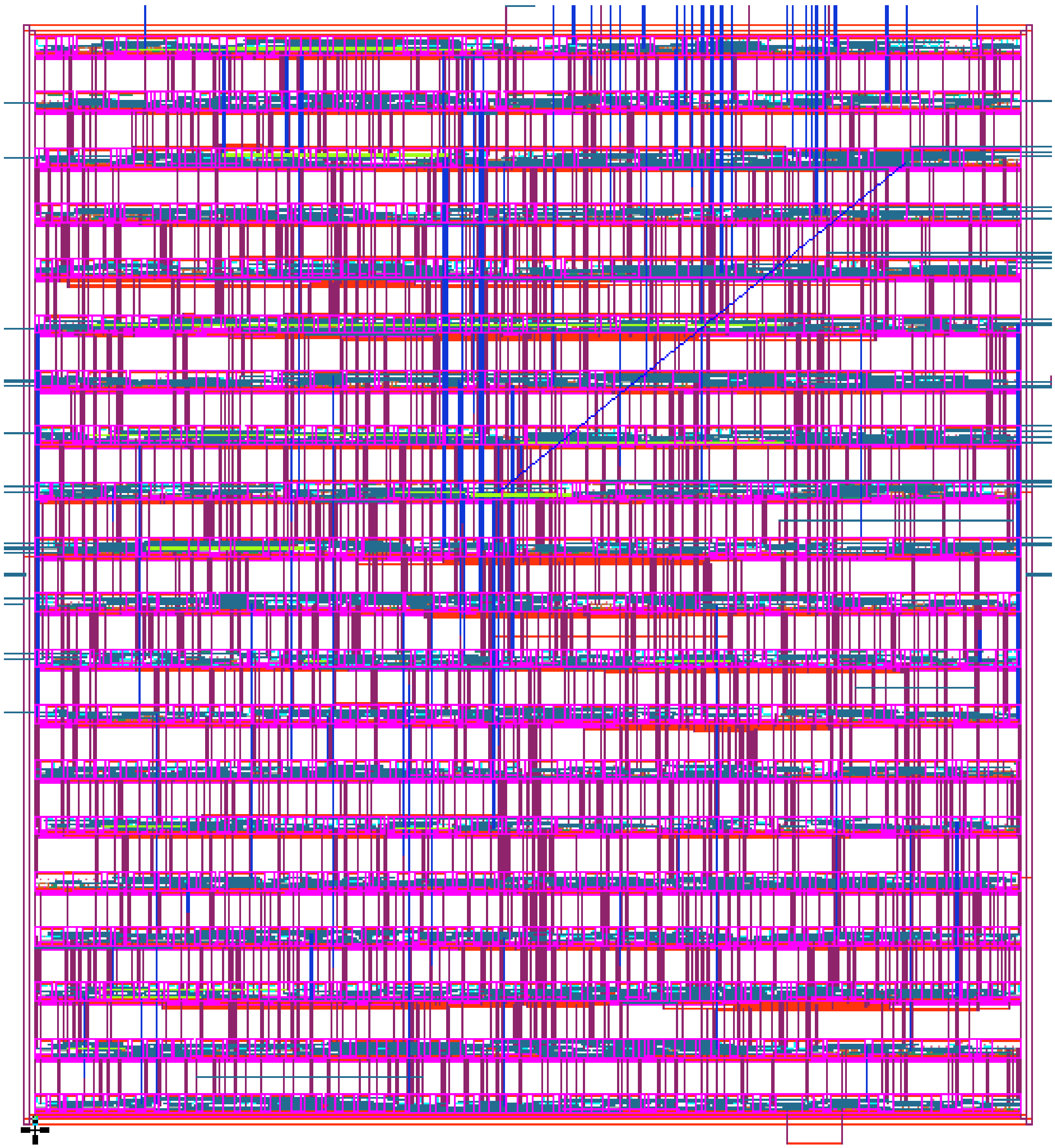


Figure 58: Failed Route

It was often the case that just a few routes caused the design to not route. After many attempts of ripping and rerouting, the routing needed to be removed and standard cells placement was run again.

The working formula consisted of 1 pass of routing with the number of routes turned to the max, and then a second pass consisted of the routes turned to a minimum and a preference for vias instead of jogs. The resulting layout can be seen in Figure 59.

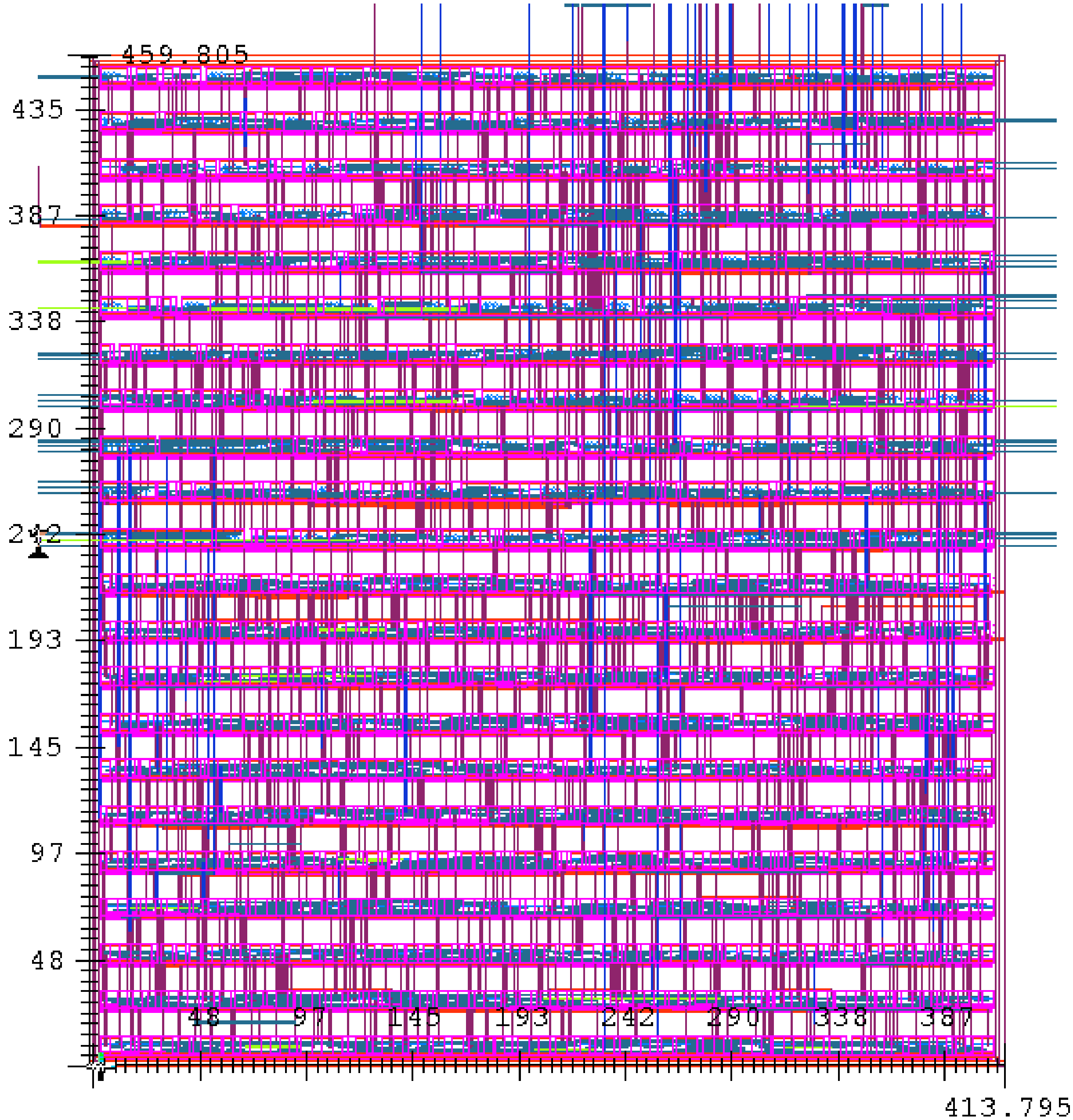


Figure 59: Full Layout

A close up view of the layout can be seen in Figure 60.

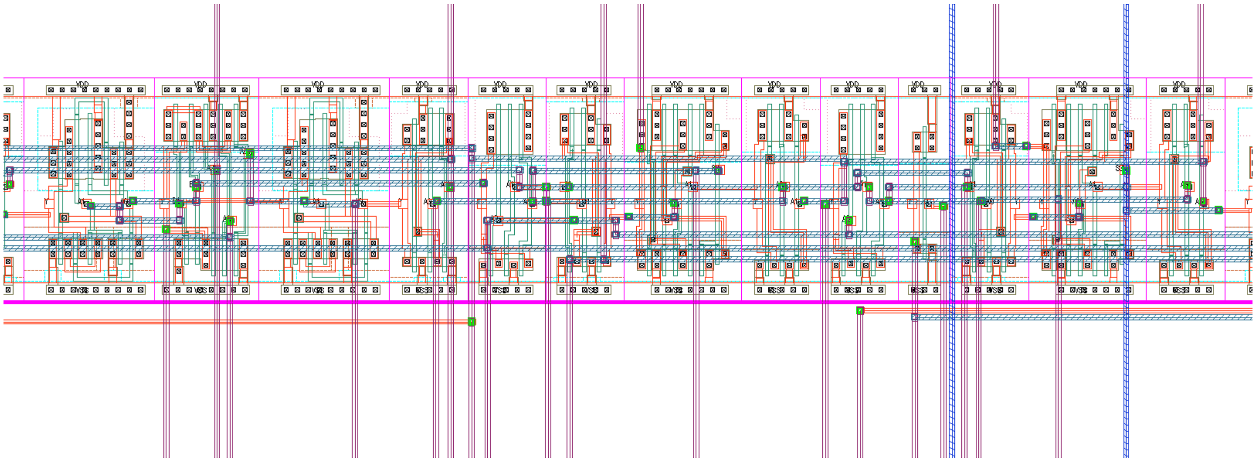
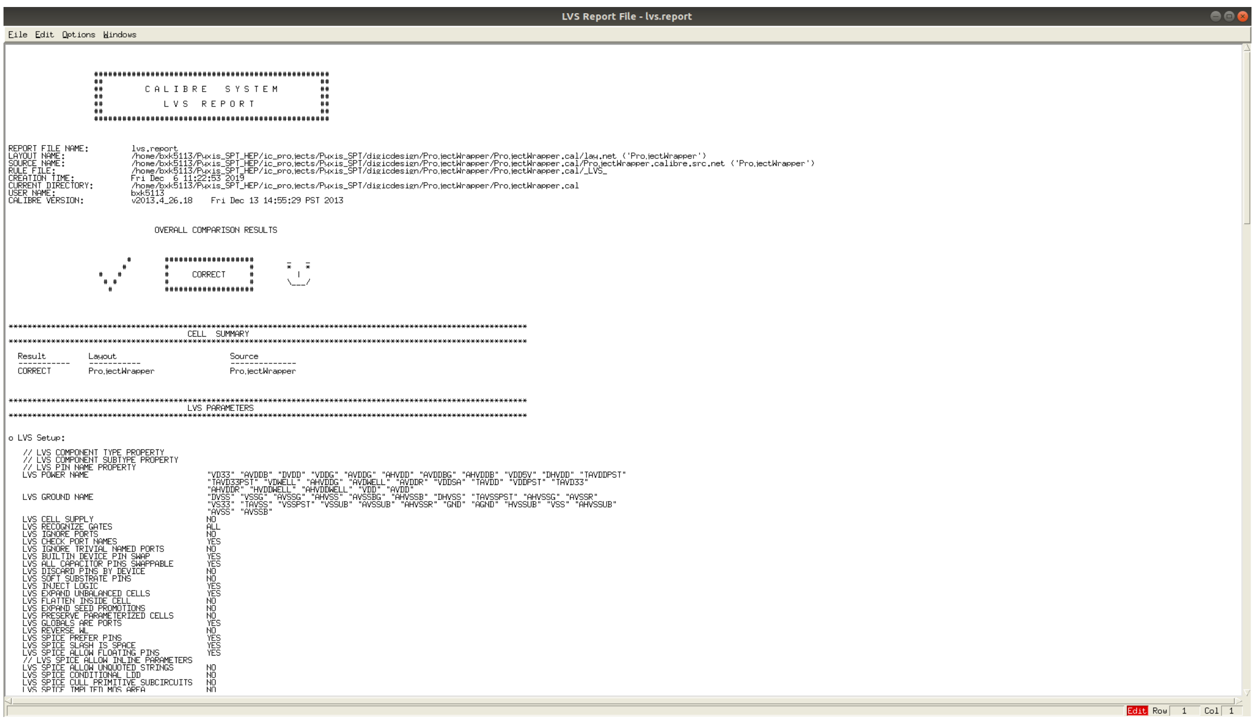


Figure 60: Full Layout Close Up View

To confirm that routing matched the schematic, an Layout Versus Schematic (LVS) test was performed. The passing test can be seen in Figure 61. The full report can be seen in Listing 43.



```
LVS Report File - lvs.report
File  Edit  Options  Windows

               ##################################################
               ##                                              ##
               ##          C A L I B R E    S Y S T E M        ##
               ##                                              ##
               ##              L V S   R E P O R T             ##
               ##                                              ##
               ##################################################

REPORT FILE NAME:         lvs.report
LAYOUT NAME:              /home/bxk5113/Pyxis_SPT_HEP/ic_projects/Pyxis_SPT/digicdesign/ProjectWrapper/ProjectWrapper.cal/lay.net ('ProjectWrapper')
SOURCE NAME:              /home/bxk5113/Pyxis_SPT_HEP/ic_projects/Pyxis_SPT/digicdesign/ProjectWrapper/ProjectWrapper.cal/ProjectWrapper.calibre.src.net ('ProjectWrapper')
RULE FILE:                /home/bxk5113/Pyxis_SPT_HEP/ic_projects/Pyxis_SPT/digicdesign/ProjectWrapper/ProjectWrapper.cal/_LVS_
CREATION TIME:            Fri Dec  6 11:22:53 2019
CURRENT DIRECTORY:        /home/bxk5113/Pyxis_SPT_HEP/ic_projects/Pyxis_SPT/digicdesign/ProjectWrapper/ProjectWrapper.cal
USER NAME:                bxk5113
CALIBRE VERSION:          v2013.4_26.18    Fri Dec 13 14:55:29 PST 2013

                               OVERALL COMPARISON RESULTS

                         #        ###################       _   _
                        #         #                 #       *   *
                   #   #          #     CORRECT     #         |
                    # #           #                 #       \___/
                     #            ###################

**************************************************************************************************************
                                            CELL  SUMMARY
**************************************************************************************************************

  Result         Layout                           Source
  -----------    -----------                      --------------
  CORRECT        ProjectWrapper                   ProjectWrapper

**************************************************************************************************************
                                            LVS PARAMETERS
**************************************************************************************************************

o LVS Setup:

   // LVS COMPONENT TYPE PROPERTY
   // LVS COMPONENT SUBTYPE PROPERTY
   // LVS PIN NAME PROPERTY
   LVS POWER NAME                            "VD33" "AVDDB" "DVDD" "VDDG" "AVDDG" "AHVDD" "AVDDBG" "AHVDDB" "VDD5V" "DHVDD" "TAVDDPST"
                                             "TAVD33PST" "VDWELL" "AHVDDG" "AVDWELL" "AVDDR" "VDDSA" "TAVDD" "VDDPST" "TAVD33"
                                             "AHVDDR" "HVDDWELL" "AHVDDWELL" "VDD" "AVDD"
   LVS GROUND NAME                           "DVSS" "VSSG" "AVSSG" "AHVSS" "AVSSBG" "AHVSSB" "DHVSS" "TAVSSPST" "AHVSSG" "AVSSR"
                                             "VS33" "TAVSS" "VSSPST" "VSSUB" "AVSSUB" "AHVSSR" "GND" "AGND" "HVSSUB" "VSS" "AHVSSUB"
                                             "AVSS" "AVSSB"
   LVS CELL SUPPLY                           NO
   LVS RECOGNIZE GATES                       ALL
   LVS IGNORE PORTS                          NO
   LVS CHECK PORT NAMES                      YES
   LVS IGNORE TRIVIAL NAMED PORTS            NO
   LVS BUILTIN DEVICE PIN SWAP               YES
   LVS ALL CAPACITOR PINS SWAPPABLE          YES
   LVS DISCARD PINS BY DEVICE                NO
   LVS SOFT SUBSTRATE PINS                   NO
   LVS INJECT LOGIC                          YES
   LVS EXPAND UNBALANCED CELLS               YES
   LVS FLATTEN INSIDE CELL                   NO
   LVS EXPAND SEED PROMOTIONS                NO
   LVS PRESERVE PARAMETERIZED CELLS          NO
   LVS GLOBALS ARE PORTS                     YES
   LVS REVERSE WL                            NO
   LVS SPICE PREFER PINS                     YES
   LVS SPICE SLASH IS SPACE                  YES
   LVS SPICE ALLOW FLOATING PINS             YES
   // LVS SPICE ALLOW INLINE PARAMETERS
   LVS SPICE ALLOW UNQUOTED STRINGS          NO
   LVS SPICE CONDITIONAL LDD                 NO
   LVS SPICE CULL PRIMITIVE SUBCIRCUITS      NO
   LVS SPICE IMPLIED MOS AREA                NO

Edit  Row  1  Col  1
```

Figure 61: Layout Versus Schematics Results

3.3 Power

Power was measured with an Eldo simulation based on the layout. To perform the simulation, a SPICE file was created which can be seen in Listing 45. To measure static power, the average power was measured while the circuit was not active but powered. The static power was measured to be 1.87nW and was recorded in Table 3.

Dynamic power was measured by recording the maximum power measured while the circuit was changing as many transistors as possible. To activate as many transistor as possible, multiple, random inputs were supplied to the circuit for a long period of time (2000us). The maximum power was found to be 16.03mW, which was recorded in Table 3.

Table 3: Simulated Power for MAC

	Measured Power (W)
Static	1.8787E-06
Dynamic	1.6029E-02

It is clear that for this circuit, dynamic power far exceeds the static power. This shows that arithmetic operations draw a lot of power. This is mostly due to their high activity factor and their high speed requirements.

4 Conclusion

5 Appendix

5.1 VHDL

Listing 1: MAC tb VHDL

```
-- Testbench created online at:
--   www.doulos.com/knowhow/perl/testbench_creation/
-- Copyright Doulos Ltd
-- SD, 03 November 2002

library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity MAC_tb is
end;

architecture bench of MAC_tb is

  constant N : integer := 32;

  component MAC
     generic( N : integer := 32);
      Port ( A : in  STD_LOGIC_VECTOR ((N/2) - 1  downto 0);
           B : in  STD_LOGIC_VECTOR ((N/2) - 1 downto 0);
           clk : in STD_LOGIC;
           WE : in STD_LOGIC;
           reset : in STD_LOGIC;
           RegOut : out  STD_LOGIC_VECTOR (N-1 downto 0));
  end component;
```

```
  signal A: STD_LOGIC_VECTOR ((N/2) - 1  downto 0);
  signal B: STD_LOGIC_VECTOR ((N/2) - 1 downto 0);
  signal clk: STD_LOGIC;
  signal WE: STD_LOGIC;
  signal reset: STD_LOGIC;
  signal RegOut: STD_LOGIC_VECTOR (N-1 downto 0);

begin

  -- Insert values for generic parameters !!
  uut: MAC generic map ( N      => 32)
            port map ( A      => A,
                       B      => B,
                       clk    => clk,
                       WE     => WE,
                       reset  => reset,
                       RegOut => RegOut );

    clk_proc : process
    begin
        if clk = '0' then
            clk <= '1';
        else
            clk <= '0';
        end if;
        wait for 50 ns;
    end process;

  stimulus: process
  begin

    -- Put initialisation code here
        WE <= '1';
        reset <= '1';
        A <= "1111111111111111";
        B <= "1111111111111111";

        wait for 300 ns;
        WE <= '0';
        A <= "0000000000010000";
        B <= "0000000000000001";
        wait for 300 ns;
        WE <= '1';

        wait for 300 ns;
        B <= "0000000000000000";

        wait for 300 ns;
        reset <= '0';


    -- Put test bench stimulus code here

    wait;
  end process;


end;
```

Listing 2: ProjectWrapper tb VHDL

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity ProjectWrapper_tb is
end;

architecture bench of ProjectWrapper_tb is

    constant N : integer := 32;

  component ProjectWrapper
      generic( N : integer := 32);
      Port ( A : in  STD_LOGIC_VECTOR ((N/2) - 1  downto 0);
      B : in  STD_LOGIC_VECTOR ((N/2) - 1 downto 0);
      clk  : in STD_LOGIC;
      WE : in STD_LOGIC;
      reset  : in STD_LOGIC;
      StartTest : in STD_LOGIC;
      RegOut : out  STD_LOGIC_VECTOR (N-1 downto 0);
      Pass : out STD_LOGIC;
      Complete : out STD_LOGIC
    );
  end component;

  signal A: STD_LOGIC_VECTOR ((N/2) - 1 downto 0);
  signal B: STD_LOGIC_VECTOR ((N/2) - 1 downto 0);
  signal clk: STD_LOGIC;
  signal WE: STD_LOGIC;
  signal reset: STD_LOGIC;
  signal StartTest: STD_LOGIC;
  signal RegOut: STD_LOGIC_VECTOR (N-1 downto 0);
  signal Pass : STD_LOGIC;
  signal Complete : STD_LOGIC;

begin

  -- Insert values for generic parameters !!
  uut: ProjectWrapper generic map ( N         =>  32)
                          port map ( A         => A,
                                     B         => B,
                                     clk       => clk,
                                     WE        => WE,
                                     reset     => reset,
                                     StartTest    => StartTest,
                                     RegOut    => RegOut,
                                     Pass => Pass,
                     Complete => Complete );

    clk_proc : process
    begin
        if clk = '0' then
            clk <= '1';
        else
            clk <= '0';
```

```
        end if;
        wait for 50 ns;
    end process;

    stimulus: process
    begin

        -- Put initialisation code here


        -- Put test bench stimulus code here


    -- Put initialisation code here
        WE <= '1';
        reset <= '0';
        StartTest <= '1';
    A <= "0000000000000010";
        B <= "0000000000000010";
        wait for 300 ns;
        WE <= '0';
        wait for 300 ns;
        reset <='1';
        A <= "0000000000010000";
        B <= "0000000000000001";
        WE <= '1';
        wait for 100800 ns;
        reset <= '0';
        StartTest <= '0';
        A <= "0000000000000010";
        B <= "0000000000000010";
        wait for 600 ns;
        reset <= '1';
        wait for 600 ns;
    B <= "0000000000000100";




        wait;
    end process;


    end;
```

Listing 3: FullAdder VHDL

```
-------------------------------------------------------------------------------
--Company        : RIT
--Author         : Brandon Key
--Created        : 02/18/2018
--
--Project Name   : Lab 3
--File           : Full_Adder.vhd
--
--Entity         : Full_Adder
--Architecture   : behav
--
--Tool Version   : VHDL '93
```

```vhdl
--Description  : Entity and behavural description of a full adder
--------------------------------------------------------------------------------

library  IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity  Full_Adder  is
    port(A,B,Cin  :  in   std_logic;
        Sum,Cout  :  out  std_logic
        );
end  Full_Adder;

architecture  behav  of  Full_Adder  is
begin
    --uses select assignment to implement the truth table of a full adder


    sum_proc: with std_logic_vector'(Cin&A&B) select
        Sum <=  '0' when "000",
                '1' when "001",
                '1' when "010",
                '0' when "011",
                '1' when "100",
                '0' when "101",
                '0' when "110",
                '1' when "111",
                '0' when others;

    Cout_proc: with std_logic_vector'(Cin&A&B) select
        Cout <=  '0' when "000",
                 '0' when "001",
                 '0' when "010",
                 '1' when "011",
                 '0' when "100",
                 '1' when "101",
                 '1' when "110",
                 '1' when "111",
                 '0' when others;

end  behav;
```

Listing 4: FA 1bit VHDL

```vhdl
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    08:18:41 03/02/2017
-- Design Name:
-- Module Name:    FA_1bit - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
```

```vhdl
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity FA_1bit is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Cin : in  STD_LOGIC;
           S : out  STD_LOGIC;
           Cout : out  STD_LOGIC);
end FA_1bit;

architecture Behavioral of FA_1bit is

begin
    S<=((A xor B) xor Cin); --Sum
    Cout<=(((A xor B) and Cin) or (A and B));--Cout


end Behavioral;
```

Listing 5: AND2 VHDL

```vhdl
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    15:02:42 03/15/2017
-- Design Name:
-- Module Name:    AND2 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
```

```
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity AND2 is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           F : out  STD_LOGIC);
end AND2;

architecture Behavioral of AND2 is

begin
    F <= A AND B;

end Behavioral;
```

Listing 6: nBitRegister VHDL

```
--------------------------------------------------------------------------------
--Company       : RIT
--Author        : Brandon Key
--Created       : 2/8/2017
--
--Project Name : Lab 2
--File          : nBitRegister.vhd
--
--Entity        : nBitRegister
--Architecture : struct
--Revision      :
--Rev 0.01      : 2/8/2017
--
--Tool Version : VHDL '93
--Description   : Entity and behavioral description of an n-bit register
--
--Notes         :
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;


entity nBitRegister is
    generic (n : integer  := 32);
    Port (
        nBitIn : in std_logic_vector(n-1 downto 0); -- n bits to store in the
    register
        WE     : in std_logic; -- Active high write enable
        Reset  : in std_logic; -- Async reset, disabled when low
        clk    : in std_logic;
        Y : out std_logic_vector(n-1 downto 0) -- 1 output , n bits wide
    );
end nBitRegister;


architecture behav of nBitRegister is
```

```
begin

    output_proc : process (clk, Reset) begin

        if Reset = '0' then
            Y <= (others => '0');
        elsif clk'event and clk = '1' then
            if WE = '1' then
                Y <= nBitIn;
            end if;
        end if;

    end process output_proc;



end behav;
```

Listing 7: Shifter VHDL

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    09:15:01 03/02/2017
-- Design Name:
-- Module Name:    Shifter - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.MATH_REAL.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Shifter is
     generic( N : integer:=16; Namnt : integer :=integer(ceil(log2(real(16)))));

    Port ( A : in  STD_LOGIC_VECTOR (N-1 downto 0);
```

```
            amnt : in  STD_LOGIC_VECTOR (Namnt-1 downto 0);
            Control : in  STD_LOGIC_VECTOR (3 downto 0);
            output : out  STD_LOGIC_VECTOR (N-1 downto 0));
end Shifter;


--1100 LSL
--1101 LSR
--1110 ASR
architecture Behavioral of Shifter is
    signal temp : integer;
begin
    proc1 : process(Control,amnt,A)
    begin
        if control ="1101" then--LSR
            for i in integer range 0 to N-1 loop
                temp<=to_integer(unsigned(amnt));--convert amnt to unsigned integer
    for indexing
                if i+temp > N-1 then--bits at leftmost
                    output(i)<='0';
                else
                    output(i)<=A(i+temp);--right Shift
                end if;
            end loop;

        elsif control ="1100" then --LSL
            for i in integer range 0 to N-1 loop
                temp<=to_integer(unsigned(amnt));--convert amnt to unsigned integer
    for indexing
                if i-temp < 0 then --rightmost bits
                    output(i)<='0';
                else
                    output(i)<=A(i-temp);--left shift
                end if;
            end loop;
        else --ASR
            for i in integer range 0 to N-1 loop
                temp<=to_integer(unsigned(amnt));--convert amnt to unsigned integer
    for indexing
                if A(N-1)='1' then--negative
                    if i+temp > N-1 then--leftmost bits
                        output(i)<='1';--preserve the negative sign
                    else
                        output(i)<=A(i+temp);-- Right Shift
                    end if;
                else--Positive
                    if i+temp > N-1 then --leftmost bits
                        output(i)<='0';
                    else
                        output(i)<=A(i+temp);--right shift
                    end if;
                end if;
            end loop;
        end if;
    end process;
end Behavioral;
```

Listing 8: TestController VHDL

```vhdl
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    14:56:40 03/15/2017
-- Design Name:
-- Module Name:    Multiplier - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library  IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity TestController is
     generic( N : integer := 32);
    Port ( clk  : in STD_LOGIC;
           StartTest  : in STD_LOGIC;
           reset_n  : in STD_LOGIC;
           Count : in  STD_LOGIC_VECTOR (N-1 downto 0);
           MISR_IN  : in STD_LOGIC_VECTOR (N-1 downto 0);
           Complete  : out STD_LOGIC;
           Pass  : out STD_LOGIC;
           TestEN: out STD_LOGIC
           );
end TestController;

architecture Datapath of TestController is

    signal complete_v, pass_v  : STD_LOGIC;

begin

    PassProc : process (clk, reset_n) begin

        if reset_n = '0' then
            Pass_v <= '0';
        elsif rising_edge(clk) then
            if (count = "00000000000000000000001111101000" and MISR_IN = "
```

```
    100011001001011111000000111100110") or Pass_v = '1' then
                Pass_v <= '1';
            else
                Pass_v <= '0';
            end if;
        end if;
    end process;

    CompleteProc : process (clk, reset_n) begin
        if reset_n = '0' then
            Complete_v <= '0';
        elsif rising_edge(clk) then
            if count = "00000000000000000000001111101000" or Complete_v = '1' then
                Complete_v <= '1';
            else
                Complete_v <= '0';
            end if;
        end if;
    end process;

    TestProc : process(clk, reset_n) begin
        if reset_n = '0' then
            TestEN <= '0';
        elsif rising_edge(clk) then
            if StartTest = '1' then
                TestEN <= '1';
            else
                TestEN <= '0';
            end if;
        end if;
    end process;

    --Assign outputs
    Complete <= Complete_v;
    Pass <= Pass_v;




end Datapath;
```

Listing 9: Subtractor VHDL

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    10:15:15 03/19/2017
-- Design Name:
-- Module Name:    Subtractor - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
```

```
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Subtractor is
     generic(N : integer :=16);
    Port ( A : in  STD_LOGIC_VECTOR (15 downto 0);
           B : in  STD_LOGIC_VECTOR (15 downto 0);
           Output : out  STD_LOGIC_VECTOR (15 downto 0));
end Subtractor;

architecture Behavioral of Subtractor is
--Component Declarations
    --Adder, to add 1
    component Ripple_Carry_FA is
        generic(N : integer :=16);--Number of bits in A and B
         Port ( A : in  STD_LOGIC_VECTOR (N-1 downto 0);
                  B : in  STD_LOGIC_VECTOR (N-1 downto 0);
                  Cin : in STD_LOGIC;
                  Sum : out  STD_LOGIC_VECTOR (N-1 downto 0);
                  Cout : out  STD_LOGIC);
    end component;

    --Logic, for bitwise not
    component Logic_Unit is
        generic( N : integer :=16);
        Port ( A : in  STD_LOGIC_VECTOR (N-1 downto 0);
                B : in  STD_LOGIC_VECTOR (N-1 downto 0);
                Control : in  STD_LOGIC_VECTOR (3 downto 0);
                output : out  STD_LOGIC_VECTOR (N-1 downto 0));
    end component;

--Signal declarations
signal logicOut, negative : STD_LOGIC_VECTOR(N-1 downto 0);
signal logicControl : STD_LOGIC_VECTOR(3 downto 0):="1001";--Set to bitwise NOT
signal one : STD_LOGIC_VECTOR(N-1 downto 0) := "0000000000000001";
signal Cout : STD_LOGIC;--Not used

begin
--Convert Input 2 to a negative number
    --Bitwise not input2
    LOGIC : Logic_Unit
        generic map(N=>N)
        port map(A=>B, B=>A, Control=>logicControl, output=>logicOut);--A=>B because it does bitwise NOT only on A

    --Add 1
    ADD1 : Ripple_Carry_FA
```

```
        generic map(N=>N)
        port map(A=>logicOut, B=>one, Cin=>'0', Sum=>negative, Cout=>Cout);

    --Add the positive A with the newly created negative B
    ADD2 : Ripple_Carry_FA
        generic map(N=>N)
        port map(A=>A, B=>negative, Cin=>'0', Sum=>Output, Cout=>Cout);
    --output now has the result of A-B

end Behavioral;
```

Listing 10: Controller VHDL

```
-----------------------------------------------------------------------
--Company      : RIT
--Author       : Brandon Key
--Created      : 03/29/2018
--
--Project Name : Lab 5
--File         : Controller.vhd
--
--Entity       : Controller
--Architecture : behav
--
--Tool Version : VHDL '93
--Description  : Contoller For BIST
-----------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Controller is
    port(
        start_BIST  : in std_logic;

        clk   : in  std_logic;
        rst_n : in  std_logic;

        is_testing     : out std_logic;
        mul_input_ctrl : out std_logic;
        EN_LFSR        : out std_logic;
        rst_n_LFSR     : out std_logic;
        EN_MISR        : out std_logic;
        rst_n_MISR     : out std_logic
        );
end Controller;

architecture struct of Controller is

    type state_type is (multiply, start_test, testing, test_hold);
    signal state, next_state : state_type := multiply;

    signal counter : integer;

    begin

       --update the state to the next_state
```

```
        state_proc : process (clk, rst_n) begin
            if rst_n = '0' then
                state <= multiply;
            elsif rising_edge(clk) then
                state <= next_state;
            end if;
        end process state_proc;

    --How to change the state
    next_state_proc : process (clk, rst_n) begin
        if rst_n = '0' then
            next_state <= multiply;
        elsif rising_edge(clk) then
            case (next_state) is
                when multiply =>
                    if start_BIST = '1' then
                        next_state <= start_test;
                    else
                        next_state <= multiply;
                    end if;

                when testing =>
                    counter <= counter + 1;
                    if counter = 255 then
                        next_state <= test_hold;
                    else
                        next_state <= testing;
                    end if;

                when start_test =>
                    next_state <= testing;
                    counter <= 0;
                when test_hold =>
                    if start_BIST = '1' then
                        next_state <= test_hold;
                    else
                        next_state <= multiply;
                    end if;
                when others =>
                    next_state <= multiply;
            end case;

        end if;
    end process next_state_proc;

    --Outputs for the states
    out_proc : process (clk) begin
        if rising_edge(clk) then
            case (state) is
                when multiply =>
                    is_testing <= '0';
                    mul_input_ctrl  <= '1';
                    EN_LFSR     <= '0';
                    rst_n_LFSR <= '0';
                    EN_MISR     <= '0';
                    rst_n_MISR <= '1';

                when start_test =>
                    is_testing <= '1';
```

```vhdl
                    mul_input_ctrl  <= '0';
                    EN_LFSR     <= '1';
                    rst_n_LFSR <= '1';
                    EN_MISR     <= '0';
                    rst_n_MISR <= '0';

                 when testing =>
                    is_testing <= '1';
                    mul_input_ctrl  <= '0';
                    EN_LFSR     <= '1';
                    rst_n_LFSR <= '1';
                    EN_MISR     <= '1';
                    rst_n_MISR <= '1';

                when test_hold  =>
                    is_testing <= '0';
                    mul_input_ctrl  <= '0';
                    EN_LFSR     <= '0';
                    rst_n_LFSR <= '1';
                    EN_MISR     <= '0';
                    rst_n_MISR <= '1';

                when others =>
                    is_testing <= '0';
                    mul_input_ctrl  <= '1';
                    EN_LFSR     <= '0';
                    rst_n_LFSR <= '0';
                    EN_MISR     <= '0';
                    rst_n_MISR <= '0';
            end case;
        end if;
    end process out_proc;

end  struct;
```

Listing 11: ProjectWrapper VHDL

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ProjectWrapper is
    generic( N : integer := 32);
    Port ( A : in  STD_LOGIC_VECTOR ((N/2) - 1  downto 0);
        B : in  STD_LOGIC_VECTOR ((N/2) - 1 downto 0);
        clk : in STD_LOGIC;
        WE : in STD_LOGIC;
        reset : in STD_LOGIC;
        StartTest : in STD_LOGIC;
        RegOut : out  STD_LOGIC_VECTOR (N-1 downto 0);
```

```
        Pass : out STD_LOGIC;
        Complete : out STD_LOGIC
    );
end ProjectWrapper;

architecture Behavioral of ProjectWrapper is
    --COMPONENT DECLARATIONS
    component MAC is
        generic( N : integer := 32);
        Port ( A : in  STD_LOGIC_VECTOR ((N/2) - 1  downto 0);
            B : in  STD_LOGIC_VECTOR ((N/2) - 1 downto 0);
            clk : in STD_LOGIC;
            WE : in STD_LOGIC;
            reset : in STD_LOGIC;
            RegOut : out  STD_LOGIC_VECTOR (N-1 downto 0));
    end component;

    component  LFSR_32_4  is
        generic (N : integer  := 32);
        port(
            clk   : in std_logic;
            rst_n : in std_logic;
            en    : in std_logic;
            bit_pattern : out std_logic_vector(N-1 downto 0)
            );
    end  component;

    component  MISR_32_4  is
        generic (N : integer  := 32);
        port(
            MISR_in : in std_logic_vector(N-1 downto 0);
            clk     : in std_logic;
            rst_n   : in std_logic;
            en      : in std_logic;
            MISR_out : out std_logic_vector(N-1 downto 0)
            );
    end  component;

    component  nBitMux_2to1  is
        generic (n : integer  := 16);
        port(
            A,B : in  std_logic_vector(n-1  downto  0);
            sel : in  std_logic;
            Y   : out std_logic_vector(n-1  downto  0)
            );
    end  component;

    component TestController is
        generic( N : integer := 32);
       Port ( clk : in STD_LOGIC;
              StartTest : in STD_LOGIC;
              reset_n : in STD_LOGIC;
              Count : in  STD_LOGIC_VECTOR (N-1 downto 0);
              MISR_IN : in STD_LOGIC_VECTOR (N-1 downto 0);
              Complete : out STD_LOGIC;
              Pass : out STD_LOGIC;
              TestEN: out STD_LOGIC
              );
    end component;
```

```
    component Counter is
        generic( N : integer := 32);
       Port ( clk : in STD_LOGIC;
              TestEN : in STD_LOGIC;
              reset : in STD_LOGIC;
              Count : out  STD_LOGIC_VECTOR (N-1 downto 0));
    end component;

    --SIGNAL DECLARATIONS
    signal MACA, MACB : STD_LOGIC_VECTOR ((N/2) - 1  downto 0);
    signal MACOUT : STD_LOGIC_VECTOR (N-1 downto 0);
    signal LFSROUT : std_logic_vector(N-1 downto 0);
    signal MISR_out, CounterOut :  std_logic_vector(N-1 downto 0);
    signal TestEN : STD_LOGIC;
begin
    --Map those ports
    MAC0 : MAC
        generic map(N => 32)
        port map ( A => MACA, B => MACB,
                   clk => clk, WE => WE, reset => reset,
                   RegOut => MACOUT);

    LFSR0 : LFSR_32_4
        generic map (N => 32)
        port map( clk => clk, rst_n => reset, en => TestEN,
                  bit_pattern => LFSROUT);

    MUXA : nBitMux_2to1
        generic map (N => 16)
        port map (A => A, B => LFSROUT(N-1 downto N/2),
                   sel => TestEN, Y => MACA);

    MUXB : nBitMux_2to1
        generic map (N => 16)
        port map (A => B, B => LFSROUT((N/2) - 1 downto 0),
              sel => TestEN, Y => MACB);

    MISR0 : MISR_32_4
        generic map(N => 32)
        port map( MISR_in => MACOUT, clk => clk,
                  rst_n => reset, en => TestEN,
                  MISR_out => MISR_out);

    TEST0 : TestController
        generic map( N => 32)
        port map(clk => clk, StartTest => StartTest,
            reset_n => reset, Count => CounterOut,
            MISR_IN => MISR_out, Complete => Complete,
            Pass => Pass, TestEN => TestEN);

    COUNT0 : Counter
        generic map( N => 32)
        port map( clk => clk, TestEN => TestEN, reset => reset,
            Count => CounterOut);

    RegOut <= MACOUT;

end Behavioral;
```

Listing 12: Counter VHDL

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    14:56:40 03/15/2017
-- Design Name:
-- Module Name:    Multiplier - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Counter is
     generic( N : integer := 32);
    Port ( clk : in STD_LOGIC;
           TestEN : in STD_LOGIC;
           reset : in STD_LOGIC;
           Count : out  STD_LOGIC_VECTOR (N-1 downto 0));
end Counter;

architecture Behavioral of Counter is
    --COMPONENT DECLARATIONS


    component  nBitAdder  is
        generic (n : integer  := 32);
        port(
            A,B : in  std_logic_vector(N-1  downto  0);
            Y   : out std_logic_vector(N-1  downto  0);
            CB  : out std_logic
            );
    end  component;

    component nBitRegister_32 is
        generic (n : integer  := 32);
```

```
        Port (
            nBitIn : in std_logic_vector(n-1 downto 0); -- n bits to store in the register
            WE     : in std_logic; -- Active high write enable
            Reset  : in std_logic; -- Async reset, disabled when low
            clk    : in std_logic;
            Y : out std_logic_vector(n-1 downto 0) -- 1 output , n bits wide
        );
    end component;

    --SIGNAL DECLARATIONS
    signal RegOut, RegIn : STD_LOGIC_VECTOR(N-1 downto 0);
    signal cout : STD_LOGIC;

begin

    --16 bit adder, always adds 1 to value in register
    ADDER0 : nBitAdder
        generic map(N => 32)
        port map(A => RegOut, B => "00000000000000000000000000000001", Y => RegIn, CB => cout);

    --Holds the current counter value
    REG0 : nBitRegister_32
        generic map(N => 32)
        port map(nBitIn => RegIn, clk => clk, WE => TestEN, Reset => reset, Y => RegOut);

    --Map output
    Count <= RegOut;

end Behavioral;
```

Listing 13: BIST tb VHDL

```
--------------------------------------------------------------------------------
--Company        : RIT
--Author         : Brandon Key
--Created        : 03/29/2018
--
--Project Name : Lab 5
--File           : BIST_tb.vhd
--
--Entity         : BIST_tb
--Architecture : behav
--
--Tool Version : VHDL '93
--Description   :  BIST
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY BIST_tb IS
END BIST_tb;
```

```vhdl
ARCHITECTURE behavior OF BIST_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Wrapper
    PORT(
         mul_input : IN  std_logic_vector(7 downto 0);
         start_BIST : IN  std_logic;
         disp_Sig : IN  std_logic;
         clk : IN  std_logic;
         rst_n : IN  std_logic;
         unused_anode : OUT  std_logic;
         hund_anode : OUT  std_logic;
         tens_anode : OUT  std_logic;
         ones_anode : OUT  std_logic;
         CAn : OUT  std_logic;
         CBn : OUT  std_logic;
         CCn : OUT  std_logic;
         CDn : OUT  std_logic;
         CEn : OUT  std_logic;
         CFn : OUT  std_logic;
         CGn : OUT  std_logic;
         is_Testing : OUT  std_logic;
         mul_disp : OUT  std_logic_vector(7 downto 0);
         sig_disp : OUT  std_logic_vector(7 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal mul_input : std_logic_vector(7 downto 0) := (others => '0');
   signal start_BIST : std_logic := '0';
   signal disp_Sig : std_logic := '0';
   signal clk : std_logic := '0';
   signal rst_n : std_logic := '0';

    --Outputs
   signal unused_anode : std_logic;
   signal hund_anode : std_logic;
   signal tens_anode : std_logic;
   signal ones_anode : std_logic;
   signal CAn : std_logic;
   signal CBn : std_logic;
   signal CCn : std_logic;
   signal CDn : std_logic;
   signal CEn : std_logic;
   signal CFn : std_logic;
   signal CGn : std_logic;
   signal is_Testing : std_logic;
   signal mul_disp : std_logic_vector(7 downto 0);
   signal sig_disp : std_logic_vector(7 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
```

```
    uut: Wrapper PORT MAP (
            mul_input => mul_input,
            start_BIST => start_BIST,
            disp_Sig => disp_Sig,
            clk => clk,
            rst_n => rst_n,
            unused_anode => unused_anode,
            hund_anode => hund_anode,
            tens_anode => tens_anode,
            ones_anode => ones_anode,
            CAn => CAn,
            CBn => CBn,
            CCn => CCn,
            CDn => CDn,
            CEn => CEn,
            CFn => CFn,
            CGn => CGn,
            is_Testing => is_Testing,
            mul_disp => mul_disp,
            sig_disp => sig_disp
        );

    -- Clock process definitions
    clk_process :process
    begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;


    -- Stimulus process
    stim_proc: process
    begin
      -- hold reset state for 100 ns.
      rst_n <= '0';
      wait for 100 ns;
      rst_n <= '1';
      wait for clk_period*3;
      -- insert stimulus here

      start_BIST <= '1';
      wait for clk_period*10;
      start_BIST <= '0';
      wait for clk_period*30;

      for i in 0 to 5 loop
        start_BIST <= '1';
        mul_input <= std_logic_vector(to_unsigned(i, mul_input'length));
        wait for clk_period*2;
        start_BIST <= '0';
        wait for clk_period*2;
      end loop;

      wait for clk_period*270;

      start_BIST <= '1';
      wait for clk_period*300;
```

```
        start_BIST <= '0';
        wait for clk_period*10;

        wait;
    end process;

END;
```

Listing 14: Multiplier VHDL

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    14:56:40 03/15/2017
-- Design Name:
-- Module Name:    Multiplier - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Multiplier is
     generic( N : integer :=32);
    Port ( A : in  STD_LOGIC_VECTOR ((N/2)-1 downto 0);
           B : in  STD_LOGIC_VECTOR ((N/2)-1 downto 0);
           Product : out  STD_LOGIC_VECTOR (N-1 downto 0));
end Multiplier;

architecture Behavioral of Multiplier is
    --COMPONENT DECLARATIONS
    component ANDADD is
        Port (  A : in  STD_LOGIC;
                B : in  STD_LOGIC;
                D : in  STD_LOGIC;
                Cin : in  STD_LOGIC;
                Sum : out  STD_LOGIC;
                Cout : out  STD_LOGIC);
    end component;
```

```
    component AND2 is
        Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           F : out  STD_LOGIC);
    end component;
    --SIGNAL DECLARATIONS
    signal Cin : STD_LOGIC :='0';
    signal Cout: STD_LOGIC :='0';
    signal F : STD_LOGIC_VECTOR(N*N downto 0);--interconnections within multiplier
    between adders and AND gates
    signal CoutArray : STD_LOGIC_VECTOR(N*N downto 0);--Signals used by the Cout
    between the 1 bit full adders
begin


        forrow : for i in integer range 0 to (N/2)-1 generate --Loop down the levels
            if0 : if i = 0 generate --top level, just AND gates
                AND0GEN : for j in integer range 0 to (N/2)-1 generate
                    AND0 : AND2
                        port map(A=>A(i), B=>B(j), F=>F(j));
                    end generate AND0GEN;
                Product(0)<=F(0);--Assign first product
            end generate if0;
            ifn0: if i>0 generate--everything else
                forcol : for j in integer range 0 to (N/2)-1 generate

                    --Generate the first full adder / and gate combo of the row
                    GEN0 : if j = 0 generate
                        ANDADD0 : ANDADD
                            port map(A=>A(i), B=>B(j),D=>F((N/2)*(i-1)+(j+1)),Cin=>
    Cin,Cout=>CoutArray((N/2)*i+j), Sum=>F((N/2)*i+j));--Tons of 2D arrays as 1D
    arrays
                            --D=>The sum of the full adder in the previous row and
    next column (i-1) (j+1).
                            --Cout=>Corresponding coordinate in CoutArray for this
    full adder [i,j].
                            --Sum=> Corresponding coordinate in F array for this
    full adder [i,j].

                        --Assign the sum of the first adder of the row to its
    respective product bit
                        Product(i)<=F((N/2)*i+j);
                    end generate GEN0;

                    --Last full adder of the first row, D has to be 0
                    GEN1N : if i=1 AND j=((N/2)-1) generate
                        ANDADD1N : ANDADD
                            port map(A=>A(i),B=>B(j),D=>Cin,Cin=>CoutArray((N/2)*i+(
    j-1)),Cout=>CoutArray((N/2)*i+j),Sum=>F((N/2)*i+j));
                                --D=>Cin, which is equal to 0
                                --Cin=>Cout of previous full adder in same row (j-1)
                                --Cout=>Corresponding coordinate in CoutArray for
    this full adder [i,j].
                                --Sum=> Corresponding coordinate in F array for this
     full adder [i,j].
                    end generate GEN1N;

```

```
                    --Generate the last full adder / and gate combo of the row
                    GENN : if i/=1 AND j= ((N/2)-1) generate
                        ANDADDN : ANDADD
                            port map(A=>A(i),B=>B(j),D=>CoutArray((N/2)*(i-1)+j),Cin=>CoutArray((N/2)*i+(j-1)),Cout=>CoutArray((N/2)*i+j),Sum=>F((N/2)*i+j));--Map the last full adder in line
                                    --D=>The Cout of the last full adder of the previous row (i-1)
                                    --Cin=>Cout of previous full adder in same row (j-1)
                                    --Cout=>Corresponding coordinate in CoutArray for this full adder [i,j].
                                    --Sum=> Corresponding coordinate in F array for this full adder [i,j].

                        --Assign Sum and Cout to Product of the last row's last full adder
                        GENNI : if i=(N/2)-1 generate
                            Product(N-1)<=CoutArray((N/2)*i+j);--Product bit from Cout
                            Product(i+j)<=F((N/2)*i+j);--Product bit from Sum
                        end generate GENNI;
                    end generate GENN;


                    --Generate all the other full adders / AND gate combos
                    GENX : if j/=((N/2)-1) AND j>0 generate
                        ANDADDX : ANDADD
                            port map(A=>A(i), B=>B(j), D=>F((N/2)*(i-1)+(j+1)), Cin=>CoutArray((N/2)*i+(j-1)), Cout=>CoutArray((N/2)*i+j),Sum=>F((N/2)*i+j));
                                    --D=>The Cout of the last full adder of the previous row (i-1)
                                    --Cin=>Cout of previous full adder in same row (j-1)
                                    --Cout=>Corresponding coordinate in CoutArray for this full adder [i,j].
                                    --Sum=> Corresponding coordinate in F array for this full adder [i,j].

                        --Assign Product bits the sum bits of the last row of full adders
                        GENXI : if i=((N/2)-1) generate
                            Product(i+j)<=F((N/2)*i+j);--Assign the sum to the respective product bit
                        end generate GENXI;

                    end generate GENX;


                end generate forcol;
            end generate ifn0;
        end generate forrow;


end Behavioral;
```

Listing 15: LFSR 32 4 VHDL

```
--------------------------------------------------------------------------------
--Company        : RIT
--Author         : Brandon Key
```

```vhdl
--Created      : 03/08/2018
--
--Project Name : Lab 5
--File         : LFSR_32_4.vhd
--
--Entity       : LFSR_32_4
--Architecture : behav
--
--Tool Version : VHDL '93
--Description  : LFSR_32_4 8 bit output, 4 tap LFSR.
------------------------------------------------------------------------------------


library  IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity  LFSR_32_4  is
    generic (N : integer  := 32);
    port(
        clk    : in std_logic;
        rst_n  : in std_logic;
        en     : in std_logic;
        bit_pattern : out std_logic_vector(N-1 downto 0)
        );
end  LFSR_32_4;

architecture  behav of  LFSR_32_4  is

    signal internal_reg : std_logic_vector(N-1 downto 0);

    constant SEED : std_logic_vector(N-1 downto 0) := x"12345678";

    begin

        bit_pattern <= internal_reg;

        --update the state to the next_state
        the_proc : process (clk, rst_n) begin
            if rst_n = '0' then
                internal_reg <= SEED;
            elsif rising_edge(clk) then
                if en = '1' then
                    --taps at 32,20,26,25
                    internal_reg(0)  <= internal_reg(1);
                    internal_reg(1)  <= internal_reg(2);
                    internal_reg(2)  <= internal_reg(3);
                    internal_reg(3)  <= internal_reg(4);
                    internal_reg(4)  <= internal_reg(5);
                    internal_reg(5)  <= internal_reg(6);
                    internal_reg(6)  <= internal_reg(7);
                    internal_reg(7)  <= internal_reg(8);
                    internal_reg(8)  <= internal_reg(9);
                    internal_reg(9)  <= internal_reg(10);
                    internal_reg(10) <= internal_reg(11);
                    internal_reg(11) <= internal_reg(12);
                    internal_reg(12) <= internal_reg(13);
                    internal_reg(13) <= internal_reg(14);
                    internal_reg(14) <= internal_reg(15);
```

```
                    internal_reg(15) <= internal_reg(16);
                    internal_reg(16) <= internal_reg(17);
                    internal_reg(17) <= internal_reg(18);
                    internal_reg(18) <= internal_reg(19);
                    internal_reg(19) <= internal_reg(20) xor internal_reg(0);
                    internal_reg(20) <= internal_reg(21);
                    internal_reg(21) <= internal_reg(22);
                    internal_reg(22) <= internal_reg(23);
                    internal_reg(23) <= internal_reg(24);
                    internal_reg(24) <= internal_reg(25) xor internal_reg(0);
                    internal_reg(25) <= internal_reg(26) xor internal_reg(0);
                    internal_reg(26) <= internal_reg(27);
                    internal_reg(27) <= internal_reg(28);
                    internal_reg(28) <= internal_reg(29);
                    internal_reg(29) <= internal_reg(30);
                    internal_reg(30) <= internal_reg(31);
                    internal_reg(31) <= internal_reg(0);

                end if;
            end if;
        end process the_proc;


end behav;
```

Listing 16: LFSR 8 4 VHDL

```
--------------------------------------------------------------------------------
--Company      : RIT
--Author       : Brandon Key
--Created      : 03/08/2018
--
--Project Name : Lab 5
--File         : LFSR_8_4.vhd
--
--Entity       : LFSR_8_4
--Architecture : behav
--
--Tool Version : VHDL '93
--Description  : LFSR_8_4 8 bit output, 4 tap LFSR.
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LFSR_8_4 is
    port(
        clk   : in std_logic;
        rst_n : in std_logic;
        en    : in std_logic;
        bit_pattern : out std_logic_vector(7 downto 0)
        );
end LFSR_8_4;

architecture behav of LFSR_8_4 is

    signal internal_reg : std_logic_vector(7 downto 0);
```

```vhdl
    constant SEED : std_logic_vector(7 downto 0) := x"6A";

    begin

        bit_pattern <= internal_reg;

        --update the state to the next_state
        the_proc : process (clk, rst_n) begin
            if rst_n = '0' then
                internal_reg <= SEED;
            elsif rising_edge(clk) then
                if en = '1' then
                    --taps at 7,5,4,3
                    internal_reg(0) <= internal_reg(1);
                    internal_reg(1) <= internal_reg(2);
                    internal_reg(2) <= internal_reg(3);
                    internal_reg(3) <= internal_reg(4) xor internal_reg(0);
                    internal_reg(4) <= internal_reg(5) xor internal_reg(0);
                    internal_reg(5) <= internal_reg(6) xor internal_reg(0);
                    internal_reg(6) <= internal_reg(7);
                    internal_reg(7) <= internal_reg(0);
                end if;
            end if;
        end process the_proc;


end  behav;
```

Listing 17: MAC VHDL

```vhdl
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    14:56:40 03/15/2017
-- Design Name:
-- Module Name:    Multiplier - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
```

```
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity MAC is
     generic( N : integer := 32);
    Port ( A : in  STD_LOGIC_VECTOR ((N/2) - 1  downto 0);
           B : in  STD_LOGIC_VECTOR ((N/2) - 1 downto 0);
           clk  : in STD_LOGIC;
           WE : in STD_LOGIC;
           reset  : in STD_LOGIC;
           RegOut : out  STD_LOGIC_VECTOR (N-1 downto 0));
end MAC;

architecture Behavioral of MAC is
    --COMPONENT DECLARATIONS
    component Multiplier is
        generic( N : integer := 32);
        Port (  A : in  STD_LOGIC_VECTOR ((N/2)-1 downto 0);
                B : in  STD_LOGIC_VECTOR ((N/2)-1 downto 0);
                Product : out  STD_LOGIC_VECTOR (N-1 downto 0));
    end component;

    component  nBitAdder  is
        generic (n : integer  := 32);
        port(
            A,B : in  std_logic_vector(N-1  downto  0);
            Y   : out std_logic_vector(N-1  downto  0);
            CB  : out std_logic
            );
    end  component;

    component nBitRegister is
        generic (n : integer  := 32);
        Port (
            nBitIn : in std_logic_vector(n-1 downto 0); -- n bits to store in the
    register
            WE     : in std_logic; -- Active high write enable
            Reset  : in std_logic; -- Async reset, disabled when low
            clk    : in std_logic;
            Y : out std_logic_vector(n-1 downto 0) -- 1 output , n bits wide
        );
    end component;

    component nBitRegister_32 is
        generic (n : integer  := 32);
        Port (
            nBitIn : in std_logic_vector(n-1 downto 0); -- n bits to store in the
    register
            WE     : in std_logic; -- Active high write enable
            Reset  : in std_logic; -- Async reset, disabled when low
            clk    : in std_logic;
            Y : out std_logic_vector(n-1 downto 0) -- 1 output , n bits wide
        );
    end component;

    component nBitRegister_16 is
        generic (n : integer  := 16);
        Port (
```

```
            nBitIn : in std_logic_vector(n-1 downto 0); -- n bits to store in the
    register
            WE     : in std_logic; -- Active high write enable
            Reset  : in std_logic; -- Async reset, disabled when low
            clk    : in std_logic;
            Y : out std_logic_vector(n-1 downto 0) -- 1 output , n bits wide
        );
    end component;

    --SIGNAL DECLARATIONS
    signal MultA,MultB : STD_LOGIC_VECTOR((N/2)-1 downto 0);
    signal Product : STD_LOGIC_VECTOR(N-1 downto 0);
    signal adderA, adderB, adderOut : STD_LOGIC_VECTOR(N-1 downto 0);
    signal cout : STD_LOGIC;

begin

    RegMultInA : nBitRegister_16
        generic map( N => 16)
        port map(nBitIn => A,
            WE => '1', clk => clk, Reset => reset,
            Y => MultA
        );

     RegMultInB : nBitRegister_16
      generic map( N => 16)
      port map(nBitIn => B,
            WE => '1', clk => clk, Reset => reset,
            Y=> MultB
      );

    MULT1 : Multiplier
        generic map( N => 32)
        port map(A => MultA, B => MultB, Product => Product);

    RegMultOut : nBitRegister_32
        generic map( N => 32)
        port map(nBitIn => Product, WE => '1', Reset => reset, clk => clk, Y =>
    adderB);

    BigBoyReg : nBitRegister_32
        generic map( N => 32)
        port map(nBitIn => adderOut, WE => WE, Reset => reset, clk => clk, Y =>
    adderA);

    ADD1 : nBitAdder
        generic map ( N => 32)
        port map( A => adderA, B => adderB, Y => adderOut, CB => cout);

    RegOut <= adderA;
end Behavioral;
```

Listing 18: MISR 32 4 VHDL

```
-------------------------------------------------------------------------------
--Company        : RIT
--Author         : Brandon Key
--Created        : 03/08/2018
--
```

```vhdl
--Project Name : Lab 5
--File         : MISR_32_4.vhd
--
--Entity       : MISR_32_4
--Architecture : behav
--
--Tool Version : VHDL '93
--Description  : MISR_32_4 32 bit output, 4 tap MISR.
-------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MISR_32_4 is
    generic (N : integer := 32);
    port(
        MISR_in : in std_logic_vector(N-1 downto 0);
        clk     : in std_logic;
        rst_n   : in std_logic;
        en      : in std_logic;
        MISR_out : out std_logic_vector(N-1 downto 0)
        );
end MISR_32_4;

architecture behav of MISR_32_4 is

    signal internal_reg : std_logic_vector(N-1 downto 0);

    constant SEED : std_logic_vector(N-1 downto 0) := x"12345678";

    begin

        MISR_out <= internal_reg;

        --update the state to the next_state
        the_proc : process (clk, rst_n) begin
            if rst_n = '0' then
                internal_reg <= SEED;
            elsif rising_edge(clk) then
                if en = '1' then
                    --taps at 32,20,26,25
                    internal_reg(0)  <= internal_reg(1)  xor MISR_in(0);
                    internal_reg(1)  <= internal_reg(2)  xor MISR_in(1);
                    internal_reg(2)  <= internal_reg(3)  xor MISR_in(2);
                    internal_reg(3)  <= internal_reg(4)  xor MISR_in(3);
                    internal_reg(4)  <= internal_reg(5)  xor MISR_in(4);
                    internal_reg(5)  <= internal_reg(6)  xor MISR_in(5);
                    internal_reg(6)  <= internal_reg(7)  xor MISR_in(6);
                    internal_reg(7)  <= internal_reg(8)  xor MISR_in(7);
                    internal_reg(8)  <= internal_reg(9)  xor MISR_in(8);
                    internal_reg(9)  <= internal_reg(10) xor MISR_in(9);
                    internal_reg(10) <= internal_reg(11) xor MISR_in(10);
                    internal_reg(11) <= internal_reg(12) xor MISR_in(11);
                    internal_reg(12) <= internal_reg(13) xor MISR_in(12);
                    internal_reg(13) <= internal_reg(14) xor MISR_in(13);
                    internal_reg(14) <= internal_reg(15) xor MISR_in(14);
                    internal_reg(15) <= internal_reg(16) xor MISR_in(15);
```

```
                        internal_reg(16) <= internal_reg(17) xor MISR_in(16);
                        internal_reg(17) <= internal_reg(18) xor MISR_in(17);
                        internal_reg(18) <= internal_reg(19) xor MISR_in(18);
                        internal_reg(19) <= internal_reg(20) xor MISR_in(19) xor
    internal_reg(0);
                        internal_reg(20) <= internal_reg(21) xor MISR_in(20);
                        internal_reg(21) <= internal_reg(22) xor MISR_in(21);
                        internal_reg(22) <= internal_reg(23) xor MISR_in(22);
                        internal_reg(23) <= internal_reg(24) xor MISR_in(23);
                        internal_reg(24) <= internal_reg(25) xor MISR_in(24) xor
    internal_reg(0);
                        internal_reg(25) <= internal_reg(26) xor MISR_in(25) xor
    internal_reg(0);
                        internal_reg(26) <= internal_reg(27) xor MISR_in(26);
                        internal_reg(27) <= internal_reg(28) xor MISR_in(27);
                        internal_reg(28) <= internal_reg(29) xor MISR_in(28);
                        internal_reg(29) <= internal_reg(30) xor MISR_in(29);
                        internal_reg(30) <= internal_reg(31) xor MISR_in(30);
                        internal_reg(31) <= internal_reg(0)  xor MISR_in(31);
                    end if;
                end if;
            end process the_proc;


end  behav;
```

Listing 19: nBitRegister tb VHDL

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   16:49:10 02/08/2018
-- Design Name:
-- Module Name:   /home/ise/DSDII/Lab/Lab2/Project/lab2/nBitRegister_tb.vhd
-- Project Name:  lab2
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: nBitRegister
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
 use ieee.numeric_std.all;

```

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY nBitRegister_tb IS
END nBitRegister_tb;

ARCHITECTURE behavior OF nBitRegister_tb IS

    constant N : integer := 4;

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT nBitRegister
    generic (n : integer  := 32);
    PORT(
          nBitIn : in std_logic_vector(n-1 downto 0); -- n bits to store in the
    register
          WE     : in std_logic; -- Active high write enable
          Reset  : in std_logic; -- Async reset, disabled when low
          clk    : in std_logic;
          Y : out std_logic_vector(n-1 downto 0) -- 1 output , n bits wide
        );
    END COMPONENT;


   --Inputs
   signal nBitIn : std_logic_vector(n-1 downto 0) := (others => '0');
   signal WE : std_logic := '0';
   signal Reset : std_logic := '0';
   signal clk : std_logic := '0';

    --Outputs
   signal Y : std_logic_vector(n-1 downto 0);

   -- Clock period definitions
   constant clk_period : time := 100 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: nBitRegister
   generic map (N => N)
   PORT MAP (
          nBitIn => nBitIn,
          WE => WE,
          Reset => Reset,
          clk => clk,
          Y => Y
        );

   -- Clock process definitions
   clk_process :process
   begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
   end process;
```

```

   -- Stimulus process
   stim_proc: process
   begin
           -- hold reset state for 100 ns.
           Reset <= '0';
      wait for (1*clk_period + 15 ns);
           Reset <= '1';
      wait for clk_period*1;
           --Setup Complete, Time to load

    --Load each register 1 by 1
      WE <= '1';
      wait for clk_period;
      nBitIn <= x"D";
      wait for clk_period;
      WE <= '0';
      nBitIn <= x"E";
      wait for clk_period;

      WE <= '1';
      wait for clk_period;
      for i in 0 to 15 loop
        nBitIn <= std_logic_vector(to_unsigned(i, nBitIn'length));
        wait for clk_period*1;
      end loop;

      wait;
   end process;

END;
```

Listing 20: nBitAdder VHDL

```
----------------------------------------------------------------------------------
--Company       : RIT
--Author        : Brandon Key
--Created       : 02/18/2018
--
--Project Name  : Lab 3
--File          : nBitAdder.vhd
--
--Entity        : nBitAdder
--Architecture  : struct
--
--Tool Version  : VHDL '93
--Description   : Entity and structural description of an adder subtractor
--              : SEL = 0 : A+B = Y
--              : SEL = 1 : A-B = Y
----------------------------------------------------------------------------------

library  IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use work.globals.all;

entity  nBitAdder  is
    generic (n : integer  := 32);
```

```
    port(
        A,B : in  std_logic_vector(n-1  downto  0);
        Y   : out std_logic_vector(n-1  downto  0);
        CB  : out std_logic
        );
end  nBitAdder;

architecture  struct of  nBitAdder  is

    component full_adder is
        port(A,B,Cin : in  std_logic;
            Sum,Cout : out  std_logic
        );
     end component full_adder;

    --Create an array to hold all of the carries
    type carry_array  is  array (n-1  downto  0) of std_logic;
    signal c_array : carry_array;

begin


    generate_adders : for i in 0 to n-1  generate
        i_first: if i = 0 generate
            --The first adder gets SEL as the Cin
            adder : full_adder port map(
                A => A(i),
                B => B(i),
                Cin => '0',
                Sum => Y(i),
                Cout => c_array(i)
            );
        end generate i_first;

        i_last  : if i = (n-1) generate
            --The last adder doesn't have a carry out
            adder : full_adder port map(
                A => A(i),
                B => B(i),
                Cin => c_array(i-1),
                Sum => Y(i),
                Cout =>c_array(i)
            );
        end generate i_last;

        --Middle adders
        i_mid : if (i /= 0) and (i /= (n-1)) generate
            adder : full_adder port map(
                A => A(i),
                B => B(i),
                Cin => c_array(i-1),
                Sum => Y(i),
                Cout => c_array(i)
            );
        end generate i_mid;

    end generate generate_adders;

    CB <= c_array(n-1);
```

```
end  struct;
```

Listing 21: nBitRegister 16 VHDL

```
----------------------------------------------------------------------------
--Company       : RIT
--Author        : Brandon Key
--Created       : 2/8/2017
--
--Project Name  : Lab 2
--File          : nBitRegister_16.vhd
--
--Entity        : nBitRegister_16
--Architecture  : struct
--Revision      :
--Rev 0.01      : 2/8/2017
--
--Tool Version  : VHDL '93
--Description   : Entity and behavioral description of an n-bit register
--
--Notes         :
----------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;


entity nBitRegister_16 is
    generic (n : integer  := 16);
    Port (
        nBitIn : in std_logic_vector(n-1 downto 0); -- n bits to store in the
    register
        WE     : in std_logic; -- Active high write enable
        Reset  : in std_logic; -- Async reset, disabled when low
        clk    : in std_logic;
        Y : out std_logic_vector(n-1 downto 0) -- 1 output , n bits wide
    );
end nBitRegister_16;


architecture behav of nBitRegister_16 is


begin

    output_proc : process (clk, Reset) begin

        if Reset = '0' then
            Y <= (others => '0');
        elsif clk'event and clk = '1' then
            if WE = '1' then
                Y <= nBitIn;
            end if;
        end if;

    end process output_proc;

```

```
end behav;
```

Listing 22: MISR 8 4 VHDL

```
--------------------------------------------------------------------------------
--Company        : RIT
--Author         : Brandon Key
--Created        : 03/08/2018
--
--Project Name : Lab 5
--File           : MISR_8_4.vhd
--
--Entity         : MISR_8_4
--Architecture : behav
--
--Tool Version : VHDL '93
--Description   : MISR_8_4 8 bit output, 4 tap MISR.
--------------------------------------------------------------------------------

library  IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity  MISR_8_4  is
    port(
        MISR_in : in std_logic_vector(7 downto 0);
        clk     : in std_logic;
        rst_n   : in std_logic;
        en      : in std_logic;
        MISR_out : out std_logic_vector(7 downto 0)
        );
end  MISR_8_4;

architecture  behav of  MISR_8_4  is

    signal internal_reg : std_logic_vector(7 downto 0);

    constant SEED : std_logic_vector(7 downto 0) := x"6A";

    begin

        MISR_out <= internal_reg;

        --update the state to the next_state
        the_proc : process (clk, rst_n) begin
            if rst_n = '0' then
                internal_reg <= SEED;
            elsif rising_edge(clk) then
                if en = '1' then
                    --taps at 7,5,4,3
                    internal_reg(0) <= internal_reg(1) xor MISR_in(0);
                    internal_reg(1) <= internal_reg(2) xor MISR_in(1);
                    internal_reg(2) <= internal_reg(3) xor MISR_in(2);
                    internal_reg(3) <= internal_reg(4) xor MISR_in(3) xor
    internal_reg(0);
                    internal_reg(4) <= internal_reg(5) xor MISR_in(4) xor
    internal_reg(0);
```

```
                    internal_reg(5) <= internal_reg(6) xor MISR_in(5) xor
    internal_reg(0);
                    internal_reg(6) <= internal_reg(7) xor MISR_in(6);
                    internal_reg(7) <= internal_reg(0) xor MISR_in(7);
                end if;
            end if;
        end process the_proc;


end behav;
```

Listing 23: nBitMux 2to1 VHDL

```
------------------------------------------------------------------------------------------
--Company        : RIT
--Author         : Brandon Key
--Created        : 3/29/2018
--
--Project Name : Lab 5
--File           : nBitMux_2to1.vhd
--
--Entity         : nBitMux_2to1
--Architecture : Dataflow
--
--Tool Version : VHDL '93
--Description   : Arbitrary width 2 to 1 mux
------------------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity nBitMux_2to1 is
    generic (n : integer := 16);
    port(
        A,B : in std_logic_vector(n-1 downto 0);
        sel : in std_logic;
        Y   : out std_logic_vector(n-1 downto 0)
        );
end nBitMux_2to1;

architecture Dataflow of nBitMux_2to1 is
    begin

        --update the state to the next_state
        the_proc : process (sel, A, B) begin
            case sel is
                when '0' =>
                    Y <= A;
                when others =>
                    Y <= B;
            end case;
        end process the_proc;

end Dataflow;
```

Listing 24: ANDADD VHDL

```
--------------------------------------------------------------------------------------------
-- Company:
```

```vhdl
-- Engineer:
--
-- Create Date:    15:25:28 03/15/2017
-- Design Name:
-- Module Name:    ANDADD - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ANDADD is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           D : in  STD_LOGIC;
           Cin : in  STD_LOGIC;
           Sum : out  STD_LOGIC;
           Cout : out  STD_LOGIC);
end ANDADD;

architecture Behavioral of ANDADD is

    --Component Declarations
    component AND2 is
        Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           F : out  STD_LOGIC);
    end component;

    component FA_1bit is
        Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Cin : in  STD_LOGIC;
           S : out  STD_LOGIC;
           Cout : out  STD_LOGIC);
    end component;

    --Signal Assignments
    signal F : STD_LOGIC;
begin
    AND0 : AND2
```

```
        port map(A=>A, B=>B,F=>F);


    FA : FA_1bit
        port map(A=>F, B=>D, Cin=>Cin, S=>Sum, Cout=>Cout);



end Behavioral;
```

Listing 25: Ripple Carry FA VHDL

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    08:27:52 03/02/2017
-- Design Name:
-- Module Name:    Ripple_Carry_FA - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Ripple_Carry_FA is
     generic(N : integer :=16);--Number of bits in A and B
    Port ( A : in  STD_LOGIC_VECTOR (N-1 downto 0);
           B : in  STD_LOGIC_VECTOR (N-1 downto 0);
              Cin : in STD_LOGIC;
           Sum : out  STD_LOGIC_VECTOR (N-1 downto 0);
           Cout : out  STD_LOGIC);
end Ripple_Carry_FA;

architecture Behavioral of Ripple_Carry_FA is
    --Interim signal used for carry ins and outs of the 1 bit full adders
    --The MSB of C is carry out
    signal C : std_logic_vector(N downto 1);

    component FA_1bit is
        Port ( A : in  STD_LOGIC;
```

```vhdl
            B : in   STD_LOGIC;
            Cin : in   STD_LOGIC;
            S : out   STD_LOGIC;
            Cout : out   STD_LOGIC);
    end component;

begin

    GEN_ADD : for i in N-1 downto 0 generate
        --Generate the first full adder, which is special because it takes Cin
        FA0_GEN : if(i=0) generate
            FA0: FA_1bit
                port map(A=>A(i),B=>B(i), Cin=>Cin, Cout=>C(1),S=>Sum(i));
        end generate FA0_GEN;

        --Generate the other adders, the last bit of C is carry out
        FAX_GEN : if(i>0) generate
            FAN : FA_1bit
                port map(A=>A(i), B=>B(i), Cin=>C(i), Cout=>C(i+1), S=>Sum(i));
        end generate FAX_GEN;


    end generate GEN_ADD;

end Behavioral;
```

Listing 26: nBitRegister 32 VHDL

```vhdl
---------------------------------------------------------------------------------
--Company        : RIT
--Author         : Brandon Key
--Created        : 2/8/2017
--
--Project Name : Lab 2
--File           : nBitRegister_32.vhd
--
--Entity         : nBitRegister_32
--Architecture : struct
--Revision       :
--Rev 0.01       : 2/8/2017
--
--Tool Version : VHDL '93
--Description   : Entity and behavioral description of an n-bit register
--
--Notes          :
---------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;


entity nBitRegister_32 is
    generic (n : integer  := 32);
    Port (
        nBitIn : in std_logic_vector(n-1 downto 0); -- n bits to store in the
    register
        WE      : in std_logic; -- Active high write enable
        Reset   : in std_logic; -- Async reset, disabled when low
        clk     : in std_logic;
```

```
        Y : out std_logic_vector(n-1 downto 0) -- 1 output , n bits wide
    );
end nBitRegister_32;


architecture behav of nBitRegister_32 is


begin

    output_proc : process (clk, Reset) begin

        if Reset = '0' then
            Y <= (others => '0');
        elsif clk'event and clk = '1' then
            if WE = '1' then
                Y <= nBitIn;
            end if;
        end if;

    end process output_proc;



end behav;
```

Listing 27: ALU Wrapper VHDL

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    13:41:10 03/18/2017
-- Design Name:
-- Module Name:    ALU_Wrapper - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.MATH_REAL.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
```

```
--use UNISIM.VComponents.all;

entity ALU_Wrapper is
    Port ( input1 : in  STD_LOGIC_VECTOR (15 downto 0);
           input2 : in  STD_LOGIC_VECTOR (15 downto 0);
           control : in  STD_LOGIC_VECTOR (3 downto 0);
           output : out  STD_LOGIC_VECTOR (15 downto 0));
end ALU_Wrapper;

architecture Behavioral of ALU_Wrapper is
--Component Declarations
    component Multiplier is
        generic( N : integer :=16);
        Port ( A : in  STD_LOGIC_VECTOR ((N/2)-1 downto 0);
               B : in  STD_LOGIC_VECTOR ((N/2)-1 downto 0);
              Product : out  STD_LOGIC_VECTOR (N-1 downto 0));
    end component;

    component Logic_Unit is
        generic( N : integer :=16);
        Port ( A : in  STD_LOGIC_VECTOR (N-1 downto 0);
                B : in  STD_LOGIC_VECTOR (N-1 downto 0);
                Control : in  STD_LOGIC_VECTOR (3 downto 0);
                output : out  STD_LOGIC_VECTOR (N-1 downto 0));
    end component;

    component Shifter is
        generic( N : integer:=16; Namnt : integer :=integer(ceil(log2(real(16)))));

        Port ( A : in  STD_LOGIC_VECTOR (N-1 downto 0);
              amnt : in  STD_LOGIC_VECTOR (Namnt-1 downto 0);
              Control : in  STD_LOGIC_VECTOR (3 downto 0);
              output : out  STD_LOGIC_VECTOR (N-1 downto 0));
    end component;

    component Ripple_Carry_FA is
        generic(N : integer :=16);--Number of bits in A and B
        Port ( A : in  STD_LOGIC_VECTOR (N-1 downto 0);
               B : in  STD_LOGIC_VECTOR (N-1 downto 0);
               Cin : in STD_LOGIC;
               Sum : out  STD_LOGIC_VECTOR (N-1 downto 0);
               Cout : out  STD_LOGIC);
    end component;

    component Subtractor is
        generic(N : integer :=16);
        Port (  A : in  STD_LOGIC_VECTOR (15 downto 0);
               B : in  STD_LOGIC_VECTOR (15 downto 0);
               Output : out  STD_LOGIC_VECTOR (15 downto 0));
    end component;

--Signal Declarations

    signal Product, Sum, ShiftOut, LogicOut, Difference : STD_LOGIC_VECTOR(15 downto
     0);
    signal Cout : STD_LOGIC;
begin
    --map multiplier
    MULT : Multiplier
```

```
        generic map(N=>16)
        port map(A=>input1(7 downto 0), B=>input2(7 downto 0), Product=>Product);

    --Map Logic Unit
    LOGIC : Logic_Unit
        generic map(N=>16)
        port map( A=>input1, B=>input2,Control=>Control,output=>LogicOut);

    --Map Shifter
    SHIFT : Shifter
        generic map(N=>16, Namnt=>4)
        port map(A=>input1, amnt=>input2(3 downto 0), Control=>Control, output=>
    ShiftOut);

    --Map Adder
    ADD : Ripple_Carry_FA
        generic map(N=>16)
        port map(A=>input1, B=>input2, Cin=>'0', Cout=>Cout, Sum=>Sum);

    --Map Subtractor
    SUB : Subtractor
        generic map(N=>16)
        port map(A=>input1, B=>input2, Output=>Difference);

    --End mapping
    ----------------------------------------

    ALU_PROC : process(input1, input2, control) begin
        C1: case control is
            when "0100"=> output<=Sum;--ADD
            when "0101"=> output<=Difference;--SUB
            when "0110"=> output<=Product;--MUL
            when "1100"=> output<=ShiftOut;--SLL
            when "1101"=> output<=ShiftOut;--SRL
            when "1110"=> output<=ShiftOut;--SRA
            when others=> output<=LogicOut;--OR,NOT,AND,XOR
        end case C1;
    end process;
end Behavioral;
```

Listing 28: nBitAdderSubtractor 16Bit VHDL

```
------------------------------------------------------------------------
--Company        : RIT
--Author         : Brandon Key
--Created        : 02/18/2018
--
--Project Name : Lab 3
--File           : nBitAdderSubtractor_16Bit.vhd
--
--Entity         : nBitAdderSubtractor_16Bit
--Architecture : struct
--
--Tool Version : VHDL '93
--Description   : Entity and structural description of an adder subtractor
--               : SEL = 0 : A+B = Y
--               : SEL = 1 : A-B = Y
------------------------------------------------------------------------

```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity nBitAdderSubtractor_16Bit is
    generic (n : integer := 16);
    port(
        A,B : in std_logic_vector(n-1 downto 0);
        SEL : in std_logic;
        Y   : out std_logic_vector(n-1 downto 0);
        CB  : out std_logic
        );
end nBitAdderSubtractor_16Bit;

architecture struct of nBitAdderSubtractor_16Bit is

    component full_adder is
        port(A,B,Cin : in std_logic;
            Sum,Cout : out std_logic
        );
     end component full_adder;

    --Create an array to hold all of the carries
    type carry_array is array (n-1 downto 0) of std_logic;
    signal c_array : carry_array;

    signal B_XOR_SEL : std_logic_vector( (n-1) downto 0);

begin

    --Generate the xor statements to be mapped to the full adders
    XORator : for i in 0 to n-1 generate
        B_XOR_SEL(i) <= B(i) xor SEL;
    end generate XORator;

    generate_adders : for i in 0 to n-1 generate
        i_first: if i = 0 generate
            --The first adder gets SEL as the Cin
            adder : full_adder port map(
                A => A(i),
                B => B_XOR_SEL(i),
                Cin => SEL,
                Sum => Y(i),
                Cout => c_array(i)
            );
        end generate i_first;

        i_last : if i = (n-1) generate
            --The last adder doesn't have a carry out
            adder : full_adder port map(
                A => A(i),
                B => B_XOR_SEL(i),
                Cin => c_array(i-1),
                Sum => Y(i),
                Cout =>c_array(i)
            );
        end generate i_last;

        --Middle adders
```

```
        i_mid : if (i /= 0) and (i /= (n-1)) generate
            adder : full_adder port map(
                A => A(i),
                B => B_XOR_SEL(i),
                Cin => c_array(i-1),
                Sum => Y(i),
                Cout => c_array(i)
            );
        end generate i_mid;

    end generate generate_adders;

    CB <= c_array(n-1) xor SEL;

end  struct;
```

Listing 29: ALU TESTBENCH VHDL

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   10:35:10 03/19/2017
-- Design Name:
-- Module Name:   G:/DSD2/Lab3/Xilinx/Lab3/ALU_TESTBENCH.vhd
-- Project Name:  Lab3
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: ALU_Wrapper
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY ALU_TESTBENCH IS
END ALU_TESTBENCH;

ARCHITECTURE behavior OF ALU_TESTBENCH IS

    -- Component Declaration for the Unit Under Test (UUT)

```

```
    COMPONENT ALU_Wrapper
    PORT(
         input1 : IN  std_logic_vector(15 downto 0);
         input2 : IN  std_logic_vector(15 downto 0);
         control : IN  std_logic_vector(3 downto 0);
         output : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal input1 : std_logic_vector(15 downto 0) := (others => '0');
   signal input2 : std_logic_vector(15 downto 0) := (others => '0');
   signal control : std_logic_vector(3 downto 0) := (others => '0');

    --Outputs
   signal output : std_logic_vector(15 downto 0);
   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name



BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: ALU_Wrapper PORT MAP (
          input1 => input1,
          input2 => input2,
          control => control,
          output => output
        );

 --0100 ADD
 --0101 SUB
 --0110 MUL
 --1000 OR
 --1001 NOT
 --1010 AND
 --1011 XOR
 --1100 SLL
 --1101 SRL
 --1110 SRA

   -- Stimulus process
   stim_proc: process
   begin
      -- insert stimulus here

        --Test ADD
        control<="0100";

        --ADD1
        input1<="1010101010101010";
        input2<="0101010101010101";
        wait for 50 ns;
        assert output="1111111111111111"
            report "ADD1 failed, expected 1111111111111111, got: " & integer'image(
    to_integer(usigned(output)));
```

```
        --ADD2
        input1<="1111111111111111";
        input2<="0000000000000000";
        wait for 50 ns;
        assert output="1111111111111111"
            report "ADD2 failed, expected 1111111111111111, got: " & integer'image(
    to_integer(usigned(output)));
        --
        --ADD3
        input1<="0000000000110011";
        input2<="0000000000010010";
                --"0000000000000000"
        wait for 50 ns;
        assert output="0000000001000101"
            report "ADD3 failed, expected 0000000001000101, got: " & integer'image(
    to_integer(usigned(output)));

        --Test SUB
        control<="0101";

        --SUB1
        input1<="1111111111111111";
        input2<="1111111111111111";
                --"0000000000000000"
        wait for 50 ns;
        assert output="0000000000000000"
            report "SUB1 failed, expected 0000000000000000, got: " & integer'image(
    to_integer(usigned(output)));

        --SUB2
        input1<="0000000001000000";--64
        input2<="0000000000010010";--18
                --"0000000000101110"--46
        wait for 50 ns;
        assert output="0000000000101110"
            report "SUB2 failed, expected 0000000000101110, got: " & integer'image(
    to_integer(usigned(output)));

        --SUB3
        input1<="0000000000000111";--7
        input2<="0000000000010000";--16
                --"1111111111110111"--(-)9
        wait for 50 ns;
        assert output="1111111111110111"
            report "SUB3 failed, expected 1111111111110111, got: " & integer'image(
    to_integer(usigned(output)));

        --TEST MUL
        control<="0110";

        --MUL1
        input1<="1111111111111111";
        input2<="0000000000000000";
                --"0000000000000000"
        wait for 50 ns;
        assert output="0000000000000000"
            report "MUL1 failed, expected 0000000000000000, got: " & integer'image(
    to_integer(usigned(output)));
```

```
        --MUL2
        input1<="0000000000000100";--4
        input2<="0000000000000101";--5
                --"0000000000010100" --20
        wait for 50 ns;
        assert output="0000000000010100"
            report "MUL2 failed, expected 0000000000010100, got: " & integer'image(
    to_integer(usigned(output)));

        --Test OR
        control<="1000";

        --OR1
        input1<="1111010101000011";
        input2<="0011100100110111";
                --"1111110101110111"
        wait for 50 ns;
        assert output="0011000100000011"
            report "OR1 failed, expected 0011000100000011, got: " & integer'image(
    to_integer(usigned(output)));

        --TEST NOT
        control<="1001";

        --NOT1
        input1<="1111010101000011";
        input2<="0000000000000000";
                --"0000101010111100"
        wait for 50 ns;
        assert output="0000101010111100"
            report "NOT1 failed, expected 0000101010111100, got: " & integer'image(
    to_integer(usigned(output)));

        --TEST AND
        control<="1010";

        --AND1
        input1<="1111010101000011";
        input2<="0011100100110111";
                --"0011000100000011"
        wait for 50 ns;
        assert output="0011000100000011"
            report "AND1 failed, expected 0011000100000011, got: " & integer'image(
    to_integer(usigned(output)));

        --TEST XOR
        control<="1011";

        --XOR1
        input1<="1111010101000011";
        input2<="0011100100110111";
                --"1100110001110100"
        wait for 50 ns;
        assert output="1100110001110100"
            report "XOR1 failed, expected 1100110001110100, got: " & integer'image(
    to_integer(usigned(output)));

        --TEST SLL
        control<="1100";
```

```
        --SLL1
        input1<="1111010101000011";
        input2<="0000000000000100";
                --"0101010000110000"
        wait for 50 ns;
        assert output="0101010000110000"
            report "SLL1 failed , expected 0101010000110000, got: " & integer'image(
    to_integer(usigned(output)));

        --TEST SRL
        control<="1101";

        --SRL1
        input1<="1111010101000011";
        input2<="0000000000000100";
                --"0000111101010100"
        wait for 50 ns;
        assert output="0000111101010100"
            report "SRL1 failed , expected 0000111101010100, got: " & integer'image(
    to_integer(usigned(output)));

        --TEST SRA
        control<="1110";

        --SRA1
        input1<="1111010101000011";
        input2<="0000000000000100";
                --"1111111101010100"
        wait for 50 ns;
        assert output="1111111101010100"
            report "SRA1 failed , expected 1111111101010100, got: " & integer'image(
    to_integer(usigned(output)));
      wait;
    end process;

END;
```

Listing 30: Logic Unit VHDL

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    08:51:09 03/02/2017
-- Design Name:
-- Module Name:    Logic_Unit - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Logic_Unit is
     generic( N : integer :=16);
    Port ( A : in  STD_LOGIC_VECTOR (N-1 downto 0);
           B : in  STD_LOGIC_VECTOR (N-1 downto 0);
           Control : in  STD_LOGIC_VECTOR (3 downto 0);
           output : out  STD_LOGIC_VECTOR (N-1 downto 0));
end Logic_Unit;
--Control:
--1000 : or
--1001 : not
--1010 : AND
--1011 : XOR
architecture Behavioral of Logic_Unit is

begin
    proc1 : process(control, A, B)
        begin
        --Depending on control, will do specific commands
        if control ="1000" then --bitwise OR
            F0 : for i in 0 to N-1 loop
                output(i)<=A(i) OR B(i);
            end loop;

        elsif control ="1001" then --bitwise NOT
            F1 : for i in 0 to N-1 loop
                output(i)<= NOT A(i);
            end loop;

        elsif control = "1010" then --bitwise AND
            F2 : for i in 0 to N-1 loop
                output(i)<= A(i) AND B(i);
            end loop;

        elsif control = "1011" then --bitwise XOR
            F3 : for i in 0 to N-1 loop
                output(i)<= A(i) XOR B(i);
            end loop;
        end if;
    end process;


end Behavioral;
```

5.2 Leonardo Scripts

Listing 31: Multiplier Spectrum Script

```
set exclude_gates {PadInC PadOut}
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {../SourceCode/FA_1bit.vhd ../SourceCode/AND2.vhd ../SourceCode/ANDADD.vhd ../
    SourceCode/Multiplier.vhd }
ungroup * -hierarchy
set sdf_write_flat_netlist TRUE
optimize
write -format verilog ./Output/Multiplier.v
write -format vhdl ./Output/Multiplier.vhdl
write -format sdf ./Output/Multiplier.sdf
```

Listing 32: LFSR 32 4 Spectrum Script

```
set exclude_gates {PadInC PadOut}
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {../SourceCode/LFSR_32_4.vhd }
set sdf_write_flat_netlist TRUE
optimize
write -format verilog ./Output/LFSR_32_4.v
write -format vhdl ./Output/LFSR_32_4.vhdl
write -format sdf ./Output/LFSR_32_4.sdf
```

Listing 33: MISR 32 4 Spectrum Script

```
set exclude_gates {PadInC PadOut}
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {../SourceCode/MISR_32_4.vhd }
set sdf_write_flat_netlist TRUE
optimize
write -format verilog ./Output/MISR_32_4.v
write -format vhdl ./Output/MISR_32_4.vhdl
write -format sdf ./Output/MISR_32_4.sdf
```

Listing 34: ProjectWrapper Spectrum Script

```
set exclude_gates {PadInC PadOut}
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {../SourceCode/AND2.vhd ../SourceCode/ANDADD.vhd ../SourceCode/FA_1bit.vhd ../
    SourceCode/FullAdder.vhd ../SourceCode/LFSR_32_4.vhd ../SourceCode/MISR_32_4.vhd
    ../SourceCode/Multiplier.vhd ../SourceCode/nBitRegister_16.vhd ../SourceCode/
    nBitRegister_32.vhd ../SourceCode/nBitAdder.vhd ../SourceCode/Counter.vhd ../
    SourceCode/TestController.vhd ../SourceCode/nBitMux_2to1.vhd ../SourceCode/MAC.
    vhd ../SourceCode/ProjectWrapper.vhd }
ungroup * -hierarchy
set sdf_write_flat_netlist TRUE
optimize
write -format verilog ./Output/ProjectWrapper.v
write -format vhdl ./Output/ProjectWrapper.vhdl
write -format sdf ./Output/ProjectWrapper.sdf
```

Listing 35: nBitAdder Spectrum Script

```
set exclude_gates {PadInC PadOut}
```

```
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {../SourceCode/FullAdder.vhd ../SourceCode/nBitAdder.vhd }
set sdf_write_flat_netlist TRUE
optimize
write -format verilog ./Output/nBitAdder.v
write -format vhdl ./Output/nBitAdder.vhdl
write -format sdf ./Output/nBitAdder.sdf
```

Listing 36: nBitAdderSubtractor Spectrum Script

```
set exclude_gates {PadInC PadOut}
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {../SourceCode/FullAdder.vhd ../SourceCode/nBitAdder.vhd }
set sdf_write_flat_netlist TRUE
optimize
write -format verilog ./Output/nBitAdder.v
write -format vhdl ./Output/nBitAdder.vhdl
write -format sdf ./Output/nBitAdder.sdf
```

Listing 37: MAC BIST Spectrum Script

```
set exclude_gates {PadInC PadOut}
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {./Output/LFSR_32_4.vhdl ./Output/MISR_32_4.vhdl ./Output/nBitAdder.vhdl ./
    Output/Multiplier.vhdl ./Output/nBitRegister_16.vhdl ./Output/nBitRegister_32.
    vhdl ./Output/nBitMux_2to1.vhdl ../SourceCode/MAC.vhd }
set sdf_write_flat_netlist TRUE
optimize
write -format verilog ./Output/MAC.v
write -format vhdl ./Output/MAC.vhdl
write -format sdf ./Output/MAC.sdf
```

Listing 38: nBitRegister 16 Spectrum Script

```
set exclude_gates {PadInC PadOut}
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {../SourceCode/nBitRegister_16.vhd }
set sdf_write_flat_netlist TRUE
optimize
write -format verilog ./Output/nBitRegister_16.v
write -format vhdl ./Output/nBitRegister_16.vhdl
write -format sdf ./Output/nBitRegister_16.sdf
```

Listing 39: MAC Spectrum Script

```
set exclude_gates {PadInC PadOut}
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {../SourceCode/FA_1bit.vhd ../SourceCode/AND2.vhd ../SourceCode/ANDADD.vhd ../
    SourceCode/Multiplier.vhd ../SourceCode/nBitAdder.vhd ../SourceCode/
    nBitRegister_16.vhd ../SourceCode/nBitRegister_32.vhd ../SourceCode/MAC.vhd }
ungroup * -hierarchy
set sdf_write_flat_netlist TRUE
optimize
```

```
write -format verilog ../SourceCode/MAC.v
write -format vhd ../SourceCode/MAC.vhd
write -format sdf ../SourceCode/MAC.sdf
```

Listing 40: nBitMux 2to1 Spectrum Script

```
set exclude_gates {PadInC PadOut}
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {../SourceCode/nBitMux_2to1.vhd }
set sdf_write_flat_netlist TRUE
optimize
write -format verilog ./Output/nBitMux_2to1.v
write -format vhdl ./Output/nBitMux_2to1.vhdl
write -format sdf ./Output/nBitMux_2to1.sdf
```

Listing 41: nBitRegister 32 Spectrum Script

```
set exclude_gates {PadInC PadOut}
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {../SourceCode/nBitRegister_32.vhd }
set sdf_write_flat_netlist TRUE
optimize
write -format verilog ./Output/nBitRegister_32.v
write -format vhdl ./Output/nBitRegister_32.vhdl
write -format sdf ./Output/nBitRegister_32.sdf
```

Listing 42: MAC (copy) Spectrum Script

```
set exclude_gates {PadInC PadOut}
load_library ~/Pyxis_SPT_HEP/ic_reflibs/external_libs/GDKgates/GDKgates_utilities/
    hdl_libs/gdk.syn
read {./Output/nBitAdder.vhdl ./Output/Multiplier.vhdl ./Output/nBitRegister_16.vhdl
     ./Output/nBitRegister_32.vhdl ../SourceCode/MAC.vhd }
set sdf_write_flat_netlist TRUE
optimize
write -format verilog ./Output/MAC.v
write -format vhdl ./Output/MAC.vhdl
write -format sdf ./Output/MAC.sdf
```

5.3 Layout Versus Schematic Results

Listing 43: MAC with BIST LVS Results

```



                    ##################################################
                    ##                                              ##
                    ##           C A L I B R E    S Y S T E M       ##
                    ##                                              ##
                    ##               L V S    R E P O R T           ##
                    ##                                              ##
                    ##################################################

```

```
REPORT FILE NAME:           lvs.report
LAYOUT NAME:                /home/bxk5113/Pyxis_SPT_HEP/ic_projects/Pyxis_SPT/
    digicdesign/ProjectWrapper/ProjectWrapper.cal/lay.net ('ProjectWrapper')
SOURCE NAME:                /home/bxk5113/Pyxis_SPT_HEP/ic_projects/Pyxis_SPT/
    digicdesign/ProjectWrapper/ProjectWrapper.cal/ProjectWrapper.calibre.src.net ('
    ProjectWrapper')
RULE FILE:                  /home/bxk5113/Pyxis_SPT_HEP/ic_projects/Pyxis_SPT/
    digicdesign/ProjectWrapper/ProjectWrapper.cal/_LVS_
CREATION TIME:              Fri Dec  6 11:22:53 2019
CURRENT DIRECTORY:          /home/bxk5113/Pyxis_SPT_HEP/ic_projects/Pyxis_SPT/
    digicdesign/ProjectWrapper/ProjectWrapper.cal
USER NAME:                  bxk5113
CALIBRE VERSION:            v2013.4_26.18     Fri Dec 13 14:55:29 PST 2013



                               OVERALL COMPARISON RESULTS



                         #         ###################       -   -
                        #          #                 #       *   *
                   #   #           #     CORRECT     #         |
                    # #            #                 #       \---/
                     #             ###################



**************************************************************************************************
                                     CELL  SUMMARY
**************************************************************************************************

  Result          Layout                         Source
  -----------     -----------                    -------------
  CORRECT         ProjectWrapper                 ProjectWrapper



**************************************************************************************************
                                     LVS PARAMETERS
**************************************************************************************************



o LVS Setup:

   // LVS COMPONENT TYPE PROPERTY
   // LVS COMPONENT SUBTYPE PROPERTY
   // LVS PIN NAME PROPERTY
   LVS POWER NAME                         "VD33" "AVDDB" "DVDD" "VDDG" "AVDDG" "
    AHVDD" "AVDDBG" "AHVDDB" "VDD5V" "DHVDD" "TAVDDPST"
                                          "TAVD33PST" "VDWELL" "AHVDDG" "AVDWELL" "
    AVDDR" "VDDSA" "TAVDD" "VDDPST" "TAVD33"
```

```
                                             "AHVDDR" "HVDDWELL" "AHVDDWELL" "VDD" "AVDD"
LVS GROUND NAME                              "DVSS" "VSSG" "AVSSG" "AHVSS" "AVSSBG" "AHVSSB" "DHVSS" "TAVSSPST" "AHVSSG" "AVSSR"
                                             "VS33" "TAVSS" "VSSPST" "VSSUB" "AVSSUB" "AHVSSR" "GND" "AGND" "HVSSUB" "VSS" "AHVSSUB"
                                             "AVSS" "AVSSB"
LVS CELL SUPPLY                              NO
LVS RECOGNIZE GATES                          ALL
LVS IGNORE PORTS                             NO
LVS CHECK PORT NAMES                         YES
LVS IGNORE TRIVIAL NAMED PORTS               NO
LVS BUILTIN DEVICE PIN SWAP                  YES
LVS ALL CAPACITOR PINS SWAPPABLE             YES
LVS DISCARD PINS BY DEVICE                   NO
LVS SOFT SUBSTRATE PINS                      NO
LVS INJECT LOGIC                             YES
LVS EXPAND UNBALANCED CELLS                  YES
LVS FLATTEN INSIDE CELL                      NO
LVS EXPAND SEED PROMOTIONS                   NO
LVS PRESERVE PARAMETERIZED CELLS             NO
LVS GLOBALS ARE PORTS                        YES
LVS REVERSE WL                               NO
LVS SPICE PREFER PINS                        YES
LVS SPICE SLASH IS SPACE                     YES
LVS SPICE ALLOW FLOATING PINS                YES
// LVS SPICE ALLOW INLINE PARAMETERS
LVS SPICE ALLOW UNQUOTED STRINGS             NO
LVS SPICE CONDITIONAL LDD                    NO
LVS SPICE CULL PRIMITIVE SUBCIRCUITS         NO
LVS SPICE IMPLIED MOS AREA                   NO
// LVS SPICE MULTIPLIER NAME
LVS SPICE OVERRIDE GLOBALS                   NO
LVS SPICE REDEFINE PARAM                     NO
LVS SPICE REPLICATE DEVICES                  NO
LVS SPICE SCALE X PARAMETERS                 NO
LVS SPICE STRICT WL                          NO
// LVS SPICE OPTION
LVS STRICT SUBTYPES                          NO
LVS EXACT SUBTYPES                           NO
LAYOUT CASE                                  NO
SOURCE CASE                                  NO
LVS COMPARE CASE                             NO
LVS DOWNCASE DEVICE                          NO
LVS REPORT MAXIMUM                           50
LVS PROPERTY RESOLUTION MAXIMUM              65536
// LVS SIGNATURE MAXIMUM
// LVS FILTER UNUSED OPTION
// LVS REPORT OPTION
LVS REPORT UNITS                             YES
// LVS NON USER NAME PORT
// LVS NON USER NAME NET
// LVS NON USER NAME INSTANCE
// LVS IGNORE DEVICE PIN

// Reduction

LVS REDUCE SERIES MOS                        NO
LVS REDUCE PARALLEL MOS                      YES
```

```
LVS REDUCE SEMI SERIES MOS              NO
LVS REDUCE SPLIT GATES                  NO
LVS REDUCE PARALLEL BIPOLAR             YES
LVS REDUCE SERIES CAPACITORS            YES
LVS REDUCE PARALLEL CAPACITORS          YES
LVS REDUCE SERIES RESISTORS             YES
LVS REDUCE PARALLEL RESISTORS           YES
LVS REDUCE PARALLEL DIODES              YES
LVS REDUCTION PRIORITY                  PARALLEL

LVS SHORT EQUIVALENT NODES              NO

// Trace Property

TRACE PROPERTY  r(rm1)  r r 0.2
TRACE PROPERTY  r(rm2)  r r 0.2
TRACE PROPERTY  r(rm3)  r r 0.2
TRACE PROPERTY  r(rm4)  r r 0.2
TRACE PROPERTY  r(rm5)  r r 0.2
TRACE PROPERTY  r(rm6)  r r 0.2
TRACE PROPERTY  r(rm7)  r r 0.2
TRACE PROPERTY  r(rm8)  r r 0.2
TRACE PROPERTY  mimcap_g13  a a 0
TRACE PROPERTY  mimcap_g13  m m 0
TRACE PROPERTY  r(rndiffs)  w w 0
TRACE PROPERTY  r(rndiffs)  l l 0
TRACE PROPERTY  r(rpdiffs)  w w 0
TRACE PROPERTY  r(rpdiffs)  l l 0
TRACE PROPERTY  r(rndiffwo)  w w 0
TRACE PROPERTY  r(rndiffwo)  l l 0
TRACE PROPERTY  r(rpdiffwo)  w w 0
TRACE PROPERTY  r(rpdiffwo)  l l 0
TRACE PROPERTY  r(rnwod)  w w 0
TRACE PROPERTY  r(rnwod)  l l 0
TRACE PROPERTY  r(rnwsti)  w w 0
TRACE PROPERTY  r(rnwsti)  l l 0
TRACE PROPERTY  r(rnpolylo)  w w 0
TRACE PROPERTY  r(rnpolylo)  l l 0
TRACE PROPERTY  r(rppolylo)  w w 0
TRACE PROPERTY  r(rppolylo)  l l 0
TRACE PROPERTY  r(rnpolyhi)  w w 0
TRACE PROPERTY  r(rnpolyhi)  l l 0
TRACE PROPERTY  r(rppolyhi)  w w 0
TRACE PROPERTY  r(rppolyhi)  l l 0
TRACE PROPERTY  mn(nmos)  l l 0
TRACE PROPERTY  mn(nmos)  w w 0
TRACE PROPERTY  mn(nmos_na)  l l 0
TRACE PROPERTY  mn(nmos_na)  w w 0
TRACE PROPERTY  mn(nmos_lvt)  l l 0
TRACE PROPERTY  mn(nmos_lvt)  w w 0
TRACE PROPERTY  mn(nmos_hvt)  l l 0
TRACE PROPERTY  mn(nmos_hvt)  w w 0
TRACE PROPERTY  mn(nmos_33)  l l 0
TRACE PROPERTY  mn(nmos_33)  w w 0
TRACE PROPERTY  mn(nmos_na33)  l l 0
TRACE PROPERTY  mn(nmos_na33)  w w 0
TRACE PROPERTY  mp(pmos)  l l 0
TRACE PROPERTY  mp(pmos)  w w 0
TRACE PROPERTY  mp(pmos_lvt)  l l 0
```

```
   TRACE PROPERTY  mp(pmos_lvt)  w w 0
   TRACE PROPERTY  mp(pmos_hvt)  l l 0
   TRACE PROPERTY  mp(pmos_hvt)  w w 0
   TRACE PROPERTY  mp(pmos_33)  l l 0
   TRACE PROPERTY  mp(pmos_33)  w w 0
   TRACE PROPERTY  d(ndiode)  a a 0.5
   TRACE PROPERTY  d(ndiode_33)  a a 0.5
   TRACE PROPERTY  d(pdiode)  a a 0.5
   TRACE PROPERTY  d(pdiode_33)  a a 0.5
   TRACE PROPERTY  d(nwdiode)  a a 0.5
   TRACE PROPERTY  c(nmosvar)  lr lr 0
   TRACE PROPERTY  c(nmosvar)  wr wr 0
   TRACE PROPERTY  mp(pmos_rf25)  rl rl 0
   TRACE PROPERTY  mp(pmos_rf25)  nr nr 0
   TRACE PROPERTY  mp(pmos_rf)  rl rl 0
   TRACE PROPERTY  mp(pmos_rf)  nr nr 0
   TRACE PROPERTY  mn(nmos_rf)  nr nr 0
   TRACE PROPERTY  mn(nmos_rf25)  nr nr 0
   TRACE PROPERTY  c(moscap_rf)  nr nr 0
   TRACE PROPERTY  c(moscap_rf25)  nr nr 0
   TRACE PROPERTY  c(mimcap)  lt lt 0
   TRACE PROPERTY  c(xjvar)  nr nr 0
   TRACE PROPERTY  spiral_inductor_lvs  nr nr 0



                   CELL COMPARISON RESULTS ( TOP LEVEL )



                         #        ###################       -   -
                        #         #                 #       *   *
                   #   #          #     CORRECT     #         |
                    # #           #                 #       \---/
                     #            ###################


LAYOUT CELL NAME:         ProjectWrapper
SOURCE CELL NAME:         ProjectWrapper

--------------------------------------------------------------------------------


INITIAL NUMBERS OF OBJECTS
--------------------------

                Layout     Source        Component Type
                ------     ------        --------------
 Ports:             72         72

 Nets:           10344      10344

 Instances:       9943       9943        MN (4 pins)
                  9943       9943        MP (4 pins)
                ------     ------
 Total Inst:     19886      19886

```

```
NUMBERS OF OBJECTS AFTER TRANSFORMATION
---------------------------------------

                     Layout      Source          Component Type
                     ------      ------          --------------
 Ports:                  72          72

 Nets:                 4859        4859

 Instances:             201         201          MN (4 pins)
                        777         777          MP (4 pins)
                        582         582          SPDW_2_1 (4 pins)
                         83          83          SPDW_3_2 (6 pins)
                          6           6          SPUP_2_1 (4 pins)
                         30          30          SPUP_2_2 (5 pins)
                        141         141          SPUP_3_2 (6 pins)
                       2001        2001          _invv (4 pins)
                        113         113          _invx2v (4 pins)
                        662         662          _nand2v (5 pins)
                          1           1          _nand3v (6 pins)
                         14          14          _nand4v (7 pins)
                        606         606          _nor2v (5 pins)
                         16          16          _nor3v (6 pins)
                          8           8          _nor4v (7 pins)
                       1128        1128          _sdw2v (4 pins)
                        141         141          _sdw3v (5 pins)
                       1586        1586          _smp2v (4 pins)
                         83          83          _smp3v (5 pins)
                     ------      ------
 Total Inst:           8179        8179


***************************************************************************************************
                                    INFORMATION AND WARNINGS
***************************************************************************************************


                     Matched    Matched    Unmatched    Unmatched    Component
                      Layout     Source       Layout       Source    Type
                     -------    -------    ---------    ---------    ---------
   Ports:                 72         72            0            0

   Nets:                4859       4859            0            0

   Instances:            201        201            0            0    MN(NMOS)
                         777        777            0            0    MP(PMOS)
                         582        582            0            0    SPDW_2_1
                          83         83            0            0    SPDW_3_2
                           6          6            0            0    SPUP_2_1
                          30         30            0            0    SPUP_2_2
                         141        141            0            0    SPUP_3_2
                        2001       2001            0            0    _invv
                         113        113            0            0    _invx2v
                         662        662            0            0    _nand2v
                           1          1            0            0    _nand3v
```

```
                         14          14           0            0      _nand4v
                        606         606           0            0      _nor2v
                         16          16           0            0      _nor3v
                          8           8           0            0      _nor4v
                       1128        1128           0            0      _sdw2v
                        141         141           0            0      _sdw3v
                       1586        1586           0            0      _smp2v
                         83          83           0            0      _smp3v
                    -------     -------     --------     --------
   Total Inst:         8179        8179           0            0


o Initial Correspondence Points:

   Ports:          VDD VSS WE REGOUT[23] CLK REGOUT[24] REGOUT[17] REGOUT[18] REGOUT[20]
                   REGOUT[19] REGOUT[26] REGOUT[25] REGOUT[21] RESET REGOUT[22] REGOUT[27]
                   REGOUT[28] REGOUT[30] REGOUT[16] REGOUT[31] REGOUT[15] REGOUT[29] REGOUT[14]
                   PASS REGOUT[13] REGOUT[9] REGOUT[12] A[12] COMPLETE REGOUT[11] REGOUT[10] A[14]
                   REGOUT[8] REGOUT[7] REGOUT[0] B[7] A[8] B[8] A[13] A[10] A[9] REGOUT[1] B[0]
                   STARTTEST A[15] A[11] REGOUT[4] A[6] B[11] B[13]


**************************************************************************************************
                                         SUMMARY
**************************************************************************************************


Total CPU Time:      0 sec
Total Elapsed Time:  0 sec
```

5.4 SPICE

Listing 44: layout test SPICE

```
* Example circuit file for simulating PEX

.OPTION DOTNODE
.HIER /

.INCLUDE "/home/bxk5113/Pyxis_SPT_HEP/ic_projects/Pyxis_SPT/digicdesign/ProjectWrapper/ProjectWrapper.cal/ProjectWrapper.pex.netlist"

.LIB /home/bxk5113/Pyxis_SPT_HEP/ic_reflibs/tech_libs/generic13/models/lib.eldo TT

* - Instantiate your parasitic netlist and add the load capacitor
** FORMAT :
* XLAYOUT [all inputs as listed by the ".subckt" line in the included netlist, in the order that they appear there] [name of the subcircuit as listed in the included netlist]
XLAYOUT COMPLETE PASS REGOUT[31] REGOUT[30] REGOUT[29] REGOUT[28] REGOUT[27] REGOUT[26] REGOUT[25] REGOUT[24] REGOUT[23] REGOUT[22] REGOUT[21] REGOUT[20] REGOUT[19]
```

```
    REGOUT[18] REGOUT[17] REGOUT[16] REGOUT[15] REGOUT[14] REGOUT[13] REGOUT[12]
    REGOUT[11] REGOUT[10] REGOUT[9] REGOUT[8] REGOUT[7] REGOUT[6] REGOUT[5] REGOUT[4]
    REGOUT[3] REGOUT[2] REGOUT[1] REGOUT[0] A[15] A[14] A[13] A[12] A[11] A[10] A[9]
    A[8] A[7] A[6] A[5] A[4] A[3] A[2] A[1] A[0] B[15] B[14] B[13] B[12] B[11] B[10]
    B[9] B[8] B[7] B[6] B[5] B[4] B[3] B[2] B[1] B[0] CLK RESET STARTTEST WE
    ProjectWrapper

* Output Capacitance
C_REGOUT[31] REGOUT[31] 0 120f
C_REGOUT[30] REGOUT[30] 0 120f
C_REGOUT[29] REGOUT[29] 0 120f
C_REGOUT[28] REGOUT[28] 0 120f
C_REGOUT[27] REGOUT[27] 0 120f
C_REGOUT[26] REGOUT[26] 0 120f
C_REGOUT[25] REGOUT[25] 0 120f
C_REGOUT[24] REGOUT[24] 0 120f
C_REGOUT[23] REGOUT[23] 0 120f
C_REGOUT[22] REGOUT[22] 0 120f
C_REGOUT[21] REGOUT[21] 0 120f
C_REGOUT[20] REGOUT[20] 0 120f
C_REGOUT[19] REGOUT[19] 0 120f
C_REGOUT[18] REGOUT[18] 0 120f
C_REGOUT[17] REGOUT[17] 0 120f
C_REGOUT[16] REGOUT[16] 0 120f
C_REGOUT[15] REGOUT[15] 0 120f
C_REGOUT[14] REGOUT[14] 0 120f
C_REGOUT[13] REGOUT[13] 0 120f
C_REGOUT[12] REGOUT[12] 0 120f
C_REGOUT[11] REGOUT[11] 0 120f
C_REGOUT[10] REGOUT[10] 0 120f
C_REGOUT[9] REGOUT[9] 0 120f
C_REGOUT[8] REGOUT[8] 0 120f
C_REGOUT[7] REGOUT[7] 0 120f
C_REGOUT[6] REGOUT[6] 0 120f
C_REGOUT[5] REGOUT[5] 0 120f
C_REGOUT[4] REGOUT[4] 0 120f
C_REGOUT[3] REGOUT[3] 0 120f
C_REGOUT[2] REGOUT[2] 0 120f
C_REGOUT[1] REGOUT[1] 0 120f
C_REGOUT[0] REGOUT[0] 0 120f


* - Analysis Setup - DC sweep
* FORMAT : .DC [name] [low] [high] [step]
*.DC VFORCE_A 0 1.2 0.01

* - Analysis Setup - Trans
* FORMAT : .TRAN [start time] [end time] [time step]
.TRAN 0 2000n 0.05n

* --- Forces
* FORMAT -- PULSE : [name] [port] [reference (0 means ground)] PULSE [low] [high] [
    delay] [fall time] [rise time] [pulse width] [period]
*
* FORMAT -- DC    : [name] [port] [reference (0 means ground)] DC [voltage]
*

VFORCE_VDD VDD 0 DC 1.08
VFORCE_VSS VSS 0 DC 0
```

```

VFORCE_CLK CLK 0 PULSE (0 1.08 25n 0.1n 0.1n 25n 50n)

VFORCE_RESET RESET 0 pwl (120n 1.08 120.1n 0 )
VFORCE_WE WE 0 DC 1.08

.SIGBUS A[15:0] VHI=1.08 VLO=0 TRISE=0.1n TFALL=0.1n TDELAY=210n THOLD=200n BASE=HEXA PATTERN 0002 0003 P
.SIGBUS B[15:0] VHI=1.08 VLO=0 TRISE=0.1n TFALL=0.1n TDELAY=210n THOLD=200n BASE=HEXA PATTERN 0003 0004 P


* --- Waveform Outputs
.PLOT TRAN V(COMPLETE)
.PLOT TRAN V(PASS)
.PLOT TRAN V(REGOUT[31])
.PLOT TRAN V(REGOUT[30])
.PLOT TRAN V(REGOUT[29])
.PLOT TRAN V(REGOUT[28])
.PLOT TRAN V(REGOUT[27])
.PLOT TRAN V(REGOUT[26])
.PLOT TRAN V(REGOUT[25])
.PLOT TRAN V(REGOUT[24])
.PLOT TRAN V(REGOUT[23])
.PLOT TRAN V(REGOUT[22])
.PLOT TRAN V(REGOUT[21])
.PLOT TRAN V(REGOUT[20])
.PLOT TRAN V(REGOUT[19])
.PLOT TRAN V(REGOUT[18])
.PLOT TRAN V(REGOUT[17])
.PLOT TRAN V(REGOUT[16])
.PLOT TRAN V(REGOUT[15])
.PLOT TRAN V(REGOUT[14])
.PLOT TRAN V(REGOUT[13])
.PLOT TRAN V(REGOUT[12])
.PLOT TRAN V(REGOUT[11])
.PLOT TRAN V(REGOUT[10])
.PLOT TRAN V(REGOUT[9])
.PLOT TRAN V(REGOUT[8])
.PLOT TRAN V(REGOUT[7])
.PLOT TRAN V(REGOUT[6])
.PLOT TRAN V(REGOUT[5])
.PLOT TRAN V(REGOUT[4])
.PLOT TRAN V(REGOUT[3])
.PLOT TRAN V(REGOUT[2])
.PLOT TRAN V(REGOUT[1])
.PLOT TRAN V(REGOUT[0])
.PLOT TRAN V(A[15])
.PLOT TRAN V(A[14])
.PLOT TRAN V(A[13])
.PLOT TRAN V(A[12])
.PLOT TRAN V(A[11])
.PLOT TRAN V(A[10])
.PLOT TRAN V(A[9])
.PLOT TRAN V(A[8])
.PLOT TRAN V(A[7])
.PLOT TRAN V(A[6])
.PLOT TRAN V(A[5])
.PLOT TRAN V(A[4])
```

```
.PLOT TRAN V(A[3])
.PLOT TRAN V(A[2])
.PLOT TRAN V(A[1])
.PLOT TRAN V(A[0])
.PLOT TRAN V(B[15])
.PLOT TRAN V(B[14])
.PLOT TRAN V(B[13])
.PLOT TRAN V(B[12])
.PLOT TRAN V(B[11])
.PLOT TRAN V(B[10])
.PLOT TRAN V(B[9])
.PLOT TRAN V(B[8])
.PLOT TRAN V(B[7])
.PLOT TRAN V(B[6])
.PLOT TRAN V(B[5])
.PLOT TRAN V(B[4])
.PLOT TRAN V(B[3])
.PLOT TRAN V(B[2])
.PLOT TRAN V(B[1])
.PLOT TRAN V(B[0])
.PLOT TRAN V(CLK)
.PLOT TRAN V(RESET)
.PLOT TRAN V(STARTTEST)
.PLOT TRAN V(WE)


* --- Params
.TEMP 125

* --- Power Measurement
.measure tran static_pwr AVG power from=90ns to=160ns
.measure tran inst_pwr MAX power from=90ns to=160ns
```

Listing 45: power test SPICE

```
* Example circuit file for simulating PEX

.OPTION DOTNODE
.HIER /

.INCLUDE "/home/bxk5113/Pyxis_SPT_HEP/ic_projects/Pyxis_SPT/digicdesign/ProjectWrapper/ProjectWrapper.cal/ProjectWrapper.pex.netlist"

.LIB /home/bxk5113/Pyxis_SPT_HEP/ic_reflibs/tech_libs/generic13/models/lib.eldo TT

* - Instantiate your parasitic netlist and add the load capacitor
** FORMAT :
* XLAYOUT [all inputs as listed by the ".subckt" line in the included netlist, in the order that they appear there] [name of the subcircuit as listed in the included netlist]
XLAYOUT COMPLETE PASS REGOUT[31] REGOUT[30] REGOUT[29] REGOUT[28] REGOUT[27] REGOUT[26] REGOUT[25] REGOUT[24] REGOUT[23] REGOUT[22] REGOUT[21] REGOUT[20] REGOUT[19] REGOUT[18] REGOUT[17] REGOUT[16] REGOUT[15] REGOUT[14] REGOUT[13] REGOUT[12] REGOUT[11] REGOUT[10] REGOUT[9] REGOUT[8] REGOUT[7] REGOUT[6] REGOUT[5] REGOUT[4] REGOUT[3] REGOUT[2] REGOUT[1] REGOUT[0] A[15] A[14] A[13] A[12] A[11] A[10] A[9] A[8] A[7] A[6] A[5] A[4] A[3] A[2] A[1] A[0] B[15] B[14] B[13] B[12] B[11] B[10] B[9] B[8] B[7] B[6] B[5] B[4] B[3] B[2] B[1] B[0] CLK RESET STARTTEST WE ProjectWrapper
```

```
* Output Capactitance
C_REGOUT[31] REGOUT[31] 0 120f
C_REGOUT[30] REGOUT[30] 0 120f
C_REGOUT[29] REGOUT[29] 0 120f
C_REGOUT[28] REGOUT[28] 0 120f
C_REGOUT[27] REGOUT[27] 0 120f
C_REGOUT[26] REGOUT[26] 0 120f
C_REGOUT[25] REGOUT[25] 0 120f
C_REGOUT[24] REGOUT[24] 0 120f
C_REGOUT[23] REGOUT[23] 0 120f
C_REGOUT[22] REGOUT[22] 0 120f
C_REGOUT[21] REGOUT[21] 0 120f
C_REGOUT[20] REGOUT[20] 0 120f
C_REGOUT[19] REGOUT[19] 0 120f
C_REGOUT[18] REGOUT[18] 0 120f
C_REGOUT[17] REGOUT[17] 0 120f
C_REGOUT[16] REGOUT[16] 0 120f
C_REGOUT[15] REGOUT[15] 0 120f
C_REGOUT[14] REGOUT[14] 0 120f
C_REGOUT[13] REGOUT[13] 0 120f
C_REGOUT[12] REGOUT[12] 0 120f
C_REGOUT[11] REGOUT[11] 0 120f
C_REGOUT[10] REGOUT[10] 0 120f
C_REGOUT[9] REGOUT[9] 0 120f
C_REGOUT[8] REGOUT[8] 0 120f
C_REGOUT[7] REGOUT[7] 0 120f
C_REGOUT[6] REGOUT[6] 0 120f
C_REGOUT[5] REGOUT[5] 0 120f
C_REGOUT[4] REGOUT[4] 0 120f
C_REGOUT[3] REGOUT[3] 0 120f
C_REGOUT[2] REGOUT[2] 0 120f
C_REGOUT[1] REGOUT[1] 0 120f
C_REGOUT[0] REGOUT[0] 0 120f


* - Analysis Setup - DC sweep
* FORMAT : .DC [name] [low] [high] [step]
*.DC VFORCE_A 0 1.2 0.01

* - Analysis Setup - Trans
* FORMAT : .TRAN [start time] [end time] [time step]
.TRAN 0 600n 0.1n

* --- Forces
* FORMAT -- PULSE : [name] [port] [reference (0 means ground)] PULSE [low] [high] [
    delay] [fall time] [rise time] [pulse width] [period]
*
* FORMAT -- DC     : [name] [port] [reference (0 means ground)] DC [voltage]
*

VFORCE_C1 CONTROL[1] 0 PULSE (0 1.08 40n 0.1n 0.1n 40n 80n)
VFORCE_C0 CONTROL[0] 0 PULSE (0 1.08 20n 0.1n 0.1n 20n 40n)

VFORCE_VDD VDD 0 DC 1.08
VFORCE_VSS VSS 0 DC 0

VFORCE_CLK CLK 0 PULSE (0 1.08 25n 0.1n 0.1n 25n 50n)

VFORCE_RESET RESET 0 pwl (120n 1.08 120.1n 0 )
```

```
.SIGBUS A[15:0] VHI=1.08 VLO=0 TRISE=0.1n TFALL=0.1n TDELAY=210n THOLD=200n BASE=
    HEXA PATTERN EFAB 3FD6 P
.SIGBUS B[15:0] VHI=1.08 VLO=0 TRISE=0.1n TFALL=0.1n TDELAY=210n THOLD=200n BASE=
    HEXA PATTERN 8C5F 0004 P


* --- Waveform Outputs
.PLOT TRAN V(COMPLETE)
.PLOT TRAN V(PASS)
.PLOT TRAN V(REGOUT[31])
.PLOT TRAN V(REGOUT[30])
.PLOT TRAN V(REGOUT[29])
.PLOT TRAN V(REGOUT[28])
.PLOT TRAN V(REGOUT[27])
.PLOT TRAN V(REGOUT[26])
.PLOT TRAN V(REGOUT[25])
.PLOT TRAN V(REGOUT[24])
.PLOT TRAN V(REGOUT[23])
.PLOT TRAN V(REGOUT[22])
.PLOT TRAN V(REGOUT[21])
.PLOT TRAN V(REGOUT[20])
.PLOT TRAN V(REGOUT[19])
.PLOT TRAN V(REGOUT[18])
.PLOT TRAN V(REGOUT[17])
.PLOT TRAN V(REGOUT[16])
.PLOT TRAN V(REGOUT[15])
.PLOT TRAN V(REGOUT[14])
.PLOT TRAN V(REGOUT[13])
.PLOT TRAN V(REGOUT[12])
.PLOT TRAN V(REGOUT[11])
.PLOT TRAN V(REGOUT[10])
.PLOT TRAN V(REGOUT[9])
.PLOT TRAN V(REGOUT[8])
.PLOT TRAN V(REGOUT[7])
.PLOT TRAN V(REGOUT[6])
.PLOT TRAN V(REGOUT[5])
.PLOT TRAN V(REGOUT[4])
.PLOT TRAN V(REGOUT[3])
.PLOT TRAN V(REGOUT[2])
.PLOT TRAN V(REGOUT[1])
.PLOT TRAN V(REGOUT[0])
.PLOT TRAN V(A[15])
.PLOT TRAN V(A[14])
.PLOT TRAN V(A[13])
.PLOT TRAN V(A[12])
.PLOT TRAN V(A[11])
.PLOT TRAN V(A[10])
.PLOT TRAN V(A[9])
.PLOT TRAN V(A[8])
.PLOT TRAN V(A[7])
.PLOT TRAN V(A[6])
.PLOT TRAN V(A[5])
.PLOT TRAN V(A[4])
.PLOT TRAN V(A[3])
.PLOT TRAN V(A[2])
.PLOT TRAN V(A[1])
.PLOT TRAN V(A[0])
```

```
.PLOT TRAN V(B[15])
.PLOT TRAN V(B[14])
.PLOT TRAN V(B[13])
.PLOT TRAN V(B[12])
.PLOT TRAN V(B[11])
.PLOT TRAN V(B[10])
.PLOT TRAN V(B[9])
.PLOT TRAN V(B[8])
.PLOT TRAN V(B[7])
.PLOT TRAN V(B[6])
.PLOT TRAN V(B[5])
.PLOT TRAN V(B[4])
.PLOT TRAN V(B[3])
.PLOT TRAN V(B[2])
.PLOT TRAN V(B[1])
.PLOT TRAN V(B[0])
.PLOT TRAN V(CLK)
.PLOT TRAN V(RESET)
.PLOT TRAN V(STARTTEST)
.PLOT TRAN V(WE)


* --- Params
.TEMP 125

* --- Power Measurement
.measure tran static_pwr AVG power from=220ns to=50ns
.measure tran inst_pwr MAX power from=10ns to=600ns
```

6 References

Key, Brandon A. *CMPE-630 Digital IC Design Laboratory Exercise 5 Full-Custom Layout Techniques*. Full-Custom Layout Techniques.

Key, Brandon A. *CMPE-630 Digital IC Design Laboratory Exercise 7 Autolayout Design Techniques (HDL-Layout)*. Autolayout Design Techniques (HDL-Layout).

Key, Brandon A. *CMPE 260 Laboratory Exercise 3 Arithmetic Logic Unit*. CMPE 260 Laboratory Exercise 3 Arithmetic Logic Unit.

Key, Brandon A. *CMPE 260 Laboratory Exercise 5 Binary Multiplier with Built in Self Test*. CMPE 260 Laboratory Exercise 5 Binary Multiplier with Built in Self Test.

//TODO Added Chris's DSD II lab