Experiment 1: Combinational Circuits - I

Prasann Viswanathan, Roll Number 190070047
EE-214, WEL, IIT Bombay
February 20th, 2021

Overview of the experiment:

The purpose of part 1 of the experiment was to design a logical circuit which returns an output of 1 for a prime number input and 0 for a composite number. The number input was four bits, as a result we had to check for decimal values {2, 3, 5, 7, 11, 13} only. I implemented this circuit using AND, OR, NOT gates only.

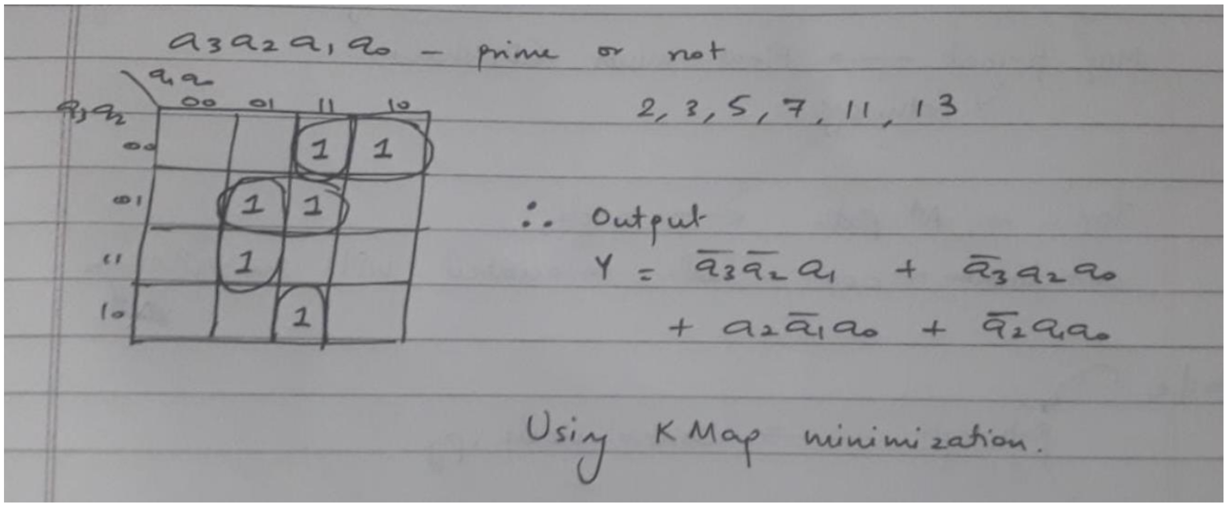
The purpose of part 2 of the experiment was to use the algorithm developed in part 1 in order to create a logical unit which performs different instructions based on whether the input binary numbers A and B were prime or composite. I implemented this circuit using the check prime logical unit from part 1, as well as four-bit adder we had created earlier and 4 input multiplexer.

For part 1 I started with the Kmap minimization for 4 bit inputs a3, …, a0. This gave a fairly easy circuit to implement with 4 product terms of size 3 each. With this in mind, I had to build a 3 input AND and a 4 input OR gate for easy implementation of the Kmap minimization. Then I finished the part 1 DUT file and made few changes to the Testbench. I generated a TRACEFILE.txt using python. Finally I ran all simulations - RTL and Gate Level to check if everything was proper. Then I implemented this on the Krypton board provided using URJtag software. On confirmation with the TA everything was proper.

For part 2 I abstracted the check prime entity as a basic logical unit returning 1 and 0 based on whether the input was prime or not. As a result this prime/not prime value became my select input for my multiplexers. The remaining was implemented via four bit adder created in a previous lab. I stored the addition and subtraction value of A and B and these different values became the 4 inputs for each of the 5 MUXes. Then I finished the DUT and Testbench and used the TRACEFILE provided to us by our TA. I checked the simulations - RTL and Gate Level and everything worked. I implemented this on Krypton board. Finally, in the subsequent lab we used scan chain along with TIVA C to implement a more efficient testing system for the krypton board and demonstrated that as well to our TA.

Approach to the experiment:

Part 1:



$a_3a_2a_1a_0$ – prime or not

$a_3a_2 \backslash a_1a_0$	00	01	11	10
00			1	1
01		1	1	
11		1		
10			1	

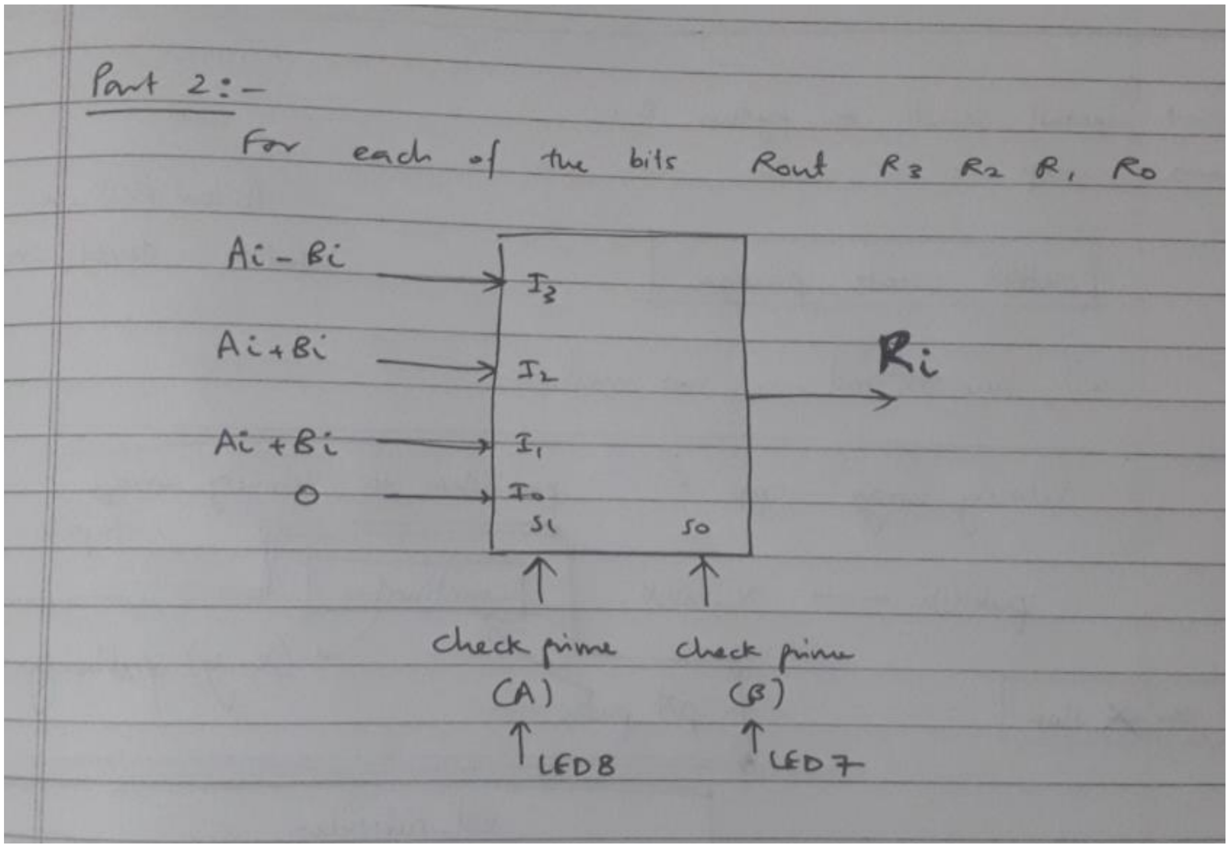
2, 3, 5, 7, 11, 13

∴ Output

$$Y = \bar{a}_3\bar{a}_2a_1 + \bar{a}_3a_2a_0 + a_2\bar{a}_1a_0 + \bar{a}_2a_1a_0$$

Using KMap minimization.

Part 2:



Part 2:-

For each of the bits Rout R_3 R_2 R_1 R_0

The logic behind generating these was previously explained in Overview.

Design document and VHDL code if relevant:

DUT: Wraps the Check_Prime entity and converts the inputs and outputs to be std logic vectors.
Testbench: Compares outputs of TRACEFILE.txt and output of implementation.

AND_3: 3 input and component made from 2 input ANDS
OR_4: 4 input or component made from 2 input ORS
Check_Prime: Main logic which implements the KMAP minimized formula in order to check if the number input is prime or not.

Architecture of Check_Prime:

```
architecture Struct of Check_Prime is

        component AND_3 is
                port (A, B, C: in std_logic; Y: out std_logic);
        end component AND_3;

        component INVERTER is
  port (A: in std_logic; Y: out std_logic);
 end component INVERTER;

 component OR_4 is
  port (A, B, C, D: in std_logic; Y: out std_logic);
 end component OR_4;

        -- Signal names are self sufficient to understand when paired with component port maps
        signal I1, I2, I3, O1, O2, O3, O4: std_logic;

begin
        -- component instances
        not1: INVERTER
                port map(A=>A1, Y=>I1);

        not2: INVERTER
                port map(A=>A2, Y=>I2);

        not3: INVERTER
                port map(A=>A3, Y=>I3);

        and1: AND_3
                port map(A=>I2, B=>A1, C=>A0, Y=>O1);

        and2: AND_3
                port map(A=>I3, B=>I2, C=>A1, Y=>O2);

        and3: AND_3
                port map(A=>I3, B=>A2, C=>A0, Y=>O3);

        and4: AND_3
                port map(A=>A2, B=>I1, C=>A0, Y=>O4);

        or_final: OR_4
                port map(A=>O1, B=>O2, C=>O3, D=>O4, Y=>Y);

end Struct;
```

End of Part 1

DUT: Wraps the Add_Subtract_Prime entity and converts the inputs and outputs to be std logic vectors.
Testbench: Compares outputs of TRACEFILE.txt and output of implementation.
Check_Prime: Previously implemented reused code.
Four_Bit_Adder: Previously implemented Four Bit Adder code.
Four_MUX: Previously implemented Four Input MUX
TwosComp: Algorithm to convert a 4 bit number to it's 2's complement.
Add_Subtract_Primes: Main logic to implement addition subtraction of numbers based on the select input for MUX.

Architecture of Add_Subtract_Primes:

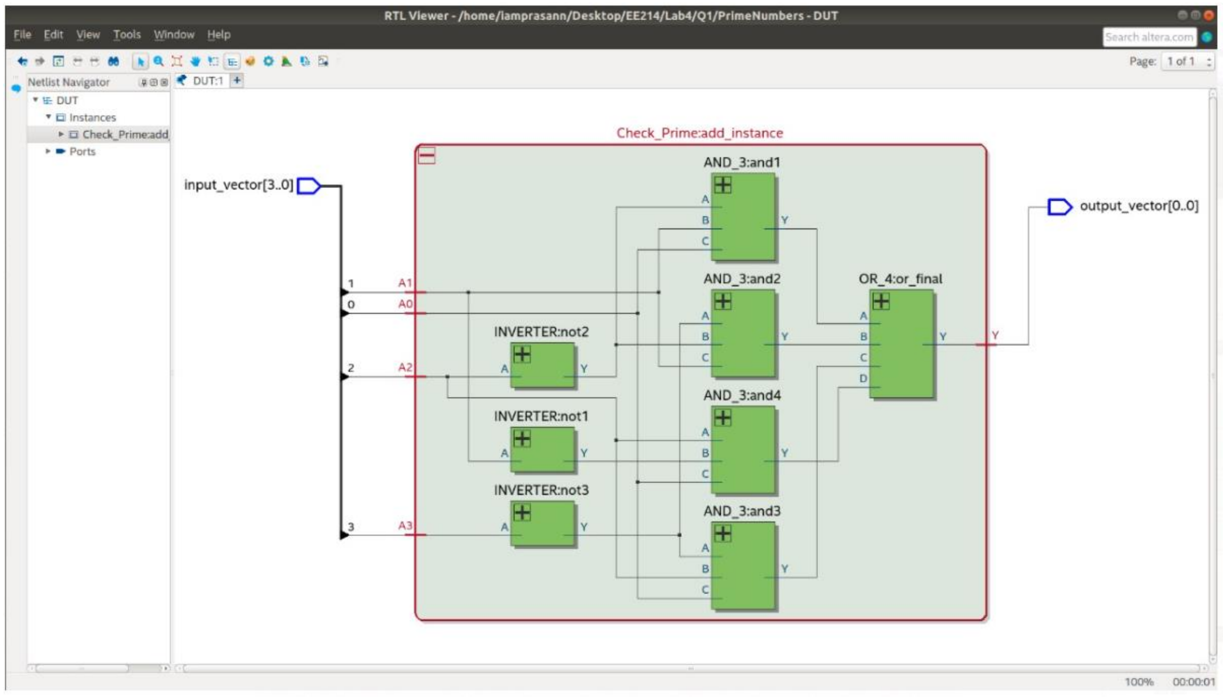
```
architecture Struct of Add_Subtract_Primes is

        component Check_Prime is
                port (A3, A2, A1, A0: in std_logic; Y: out std_logic);
        end component Check_Prime;

        component Four_Bit_Adder  is
        port (A0, A1, A2, A3, B0, B1, B2, B3: in std_logic; R0, R1, R2, R3, Cout: out std_logic);
        end component Four_Bit_Adder;

        component TwosComp is
                port (X1, X2, X3, X4: in std_logic; Y1, Y2, Y3, Y4: out std_logic);
        end component TwosComp;

        component FourIPMUX is
                port (X1, X2, X3, X4, S1, S2: in std_logic; Y: out std_logic);
        end component FourIPMUX;

        component INVERTER is
  port (A: in std_logic; Y: out std_logic);
 end component INVERTER;

        -- Signal names are self sufficient to understand when paired with component port maps
        signal Log1, Log2, I0, I1, I2, I3, Sub0, Sub1, Sub2, Sub3, CSFinal, CarrySub, Add0, Add1,
Add2, Add3, CarryAdd: std_logic;

begin
        -- component instances
        logical1: Check_Prime
                port map(A3=>A3, A2=>A2, A1=>A1, A0=>A0, Y=>Log1);

        logical2: Check_Prime
                port map(A3=>B3, A2=>B2, A1=>B1, A0=>B0, Y=>Log2);

        minus_b: TwosComp
                port map(B0, B1, B2, B3, I0, I1, I2, I3);

        sub: Four_Bit_Adder
                port map(A0, A1, A2, A3, I0, I1, I2, I3, Sub0, Sub1, Sub2, Sub3, CarrySub);

        add: Four_Bit_Adder
                port map(A0, A1, A2, A3, B0, B1, B2, B3, Add0, Add1, Add2, Add3, CarryAdd);

        mux1: FourIPMUX
```

```
                    port map('0', Add0, Add0, Sub0, Log1, Log2, R0);

          mux2: FourIPMUX
                    port map('0', Add1, Add1, Sub1, Log1, Log2, R1);

          mux3: FourIPMUX
                    port map('0', Add2, Add2, Sub2, Log1, Log2, R2);

          mux4: FourIPMUX
                    port map('0', Add3, Add3, Sub3, Log1, Log2, R3);

          inv : INVERTER
                    port map(CarrySub, CSFinal);

          mux5: FourIPMUX
                    port map('0', CarryAdd, CarryAdd, CSFinal, Log1, Log2, Cout);

          L1 <= Log1;
          L2 <= Log2;
end Struct;
```

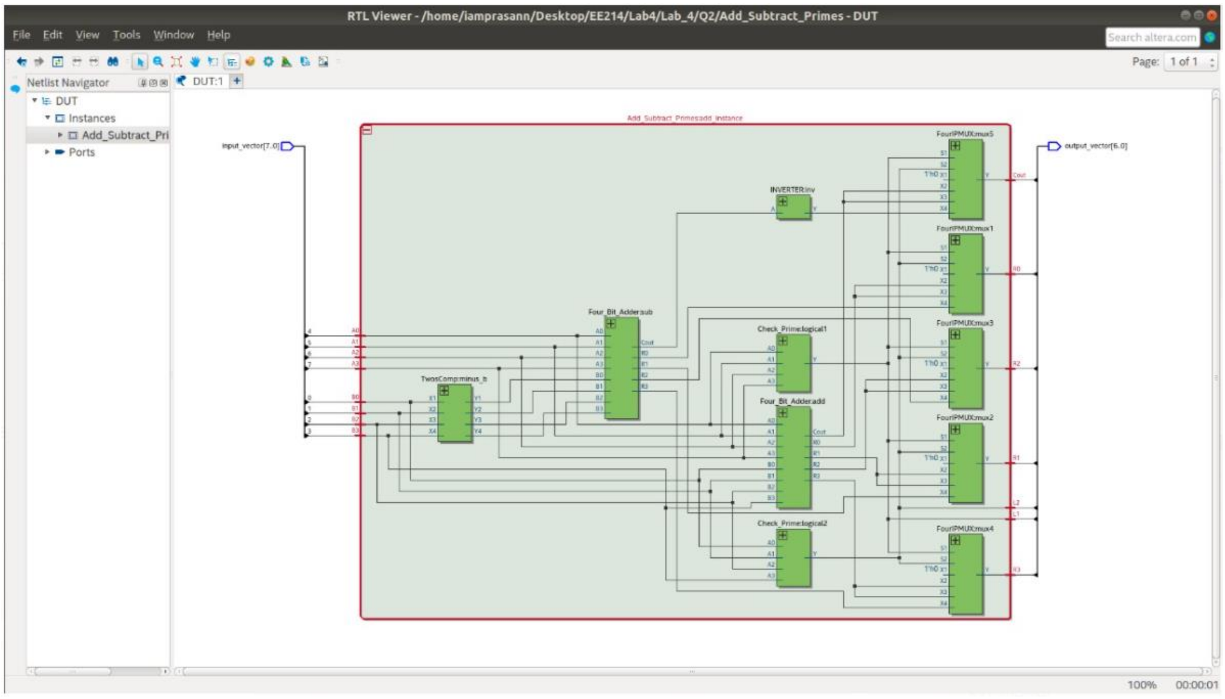
End of Part 2

RTL View:

Part 1:



Part 2:



DUT Input/Output Format:

Part 1:
Input is a std logic vector of size 4. The LSB of the input is a0 and the MSB of the input is a3.
Output is a std logic vector of size 1. It is just the truth value of whether the 4 bit number is prime.

```
0110 0 1
0111 1 1
1000 0 1
1001 0 1
1010 0 1
1011 1 1
1100 0 1
```

(Note 7 and 11 are Prime)

Part 2:
Input is a std logic vector of size 8. The lower nibble is B3-B0 and upper nibble A3-A0.
Output is a std logic vector of size 7. The MSBs are truth value of primes and lower 5 bits are the arithmetic output.

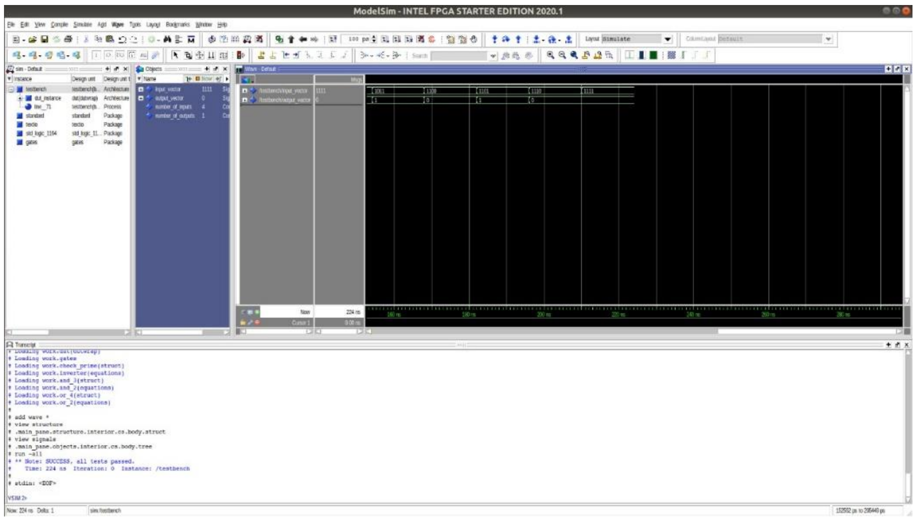
```
00001010 0000000 1111111
00001011 0101011 1111111
00001100 0000000 1111111
00001101 0101101 1111111
00001110 0000000 1111111
```

00001111 0000000 1111111
00010000 0000000 1111111
00010001 0000000 1111111
00010010 0100011 1111111
00010011 0100100 1111111
00010100 0000000 1111111

Examples of some cases: Can check that if either is prime we are adding and if both are prime we subtract. Finally if both composite, our output is 0.

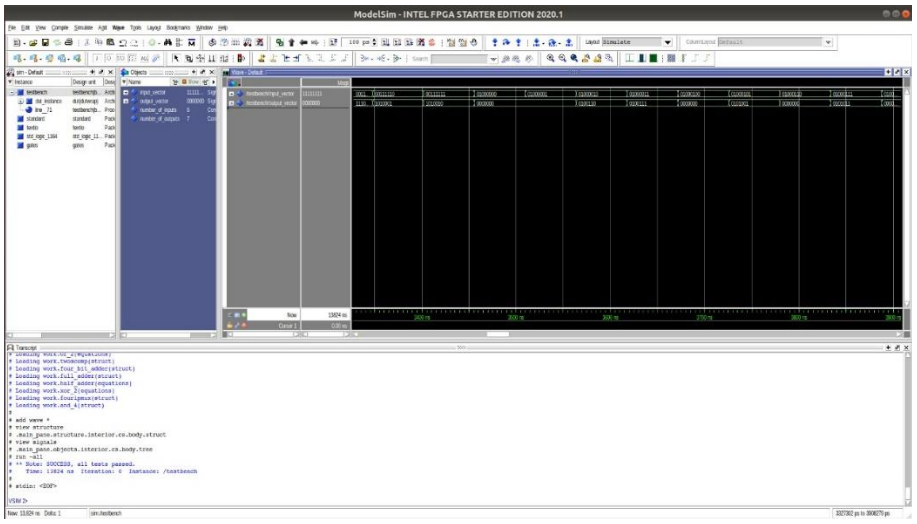
RTL Simulation:

Part 1:



```
# Loading work.gates
# Loading work.check_prime(struct)
# Loading work.inverter(equations)
# Loading work.and_3(struct)
# Loading work.and_2(equations)
# Loading work.or_4(struct)
# Loading work.or_2(equations)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 224 ns  Iteration: 0  Instance: /testbench
#
# stdin: <EOF>

VSIM 2>
```

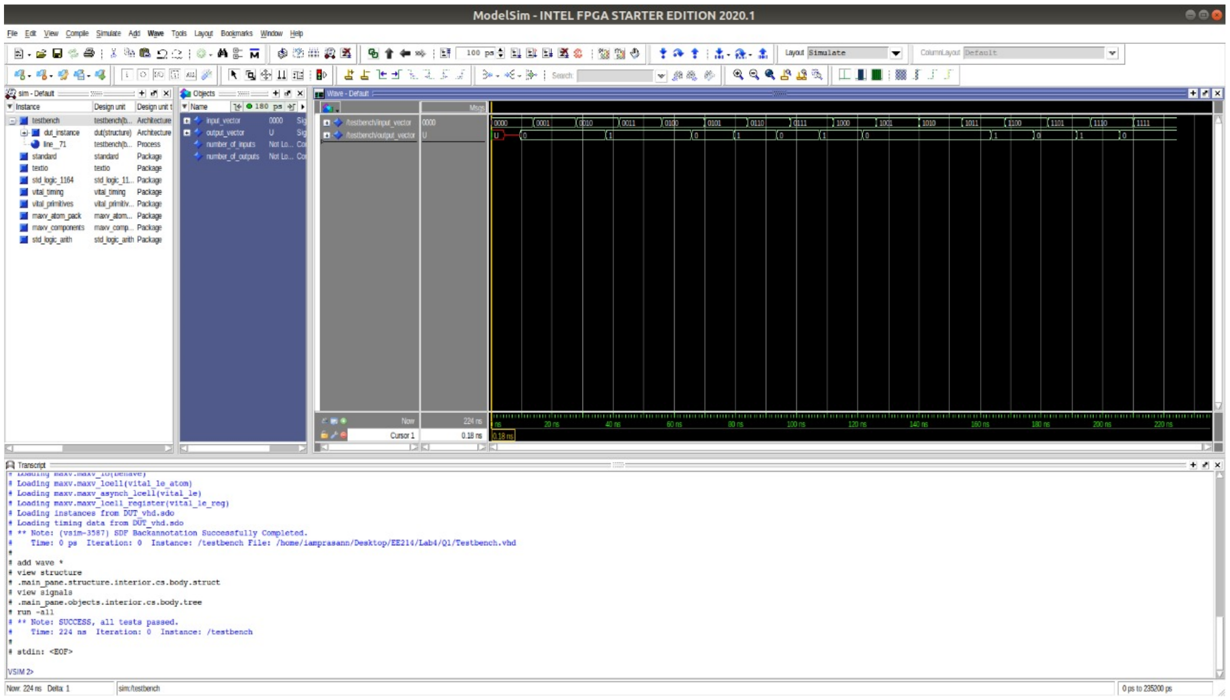
Part 2:



```
# Loading work.twoscomp(struct)
# Loading work.four_bit_adder(struct)
# Loading work.full_adder(struct)
# Loading work.half_adder(equations)
# Loading work.xor_2(equations)
# Loading work.fouripmux(struct)
# Loading work.and_4(struct)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 13824 ns  Iteration: 0  Instance: /testbench
#
# stdin: <EOF>

VSIM 2>
```

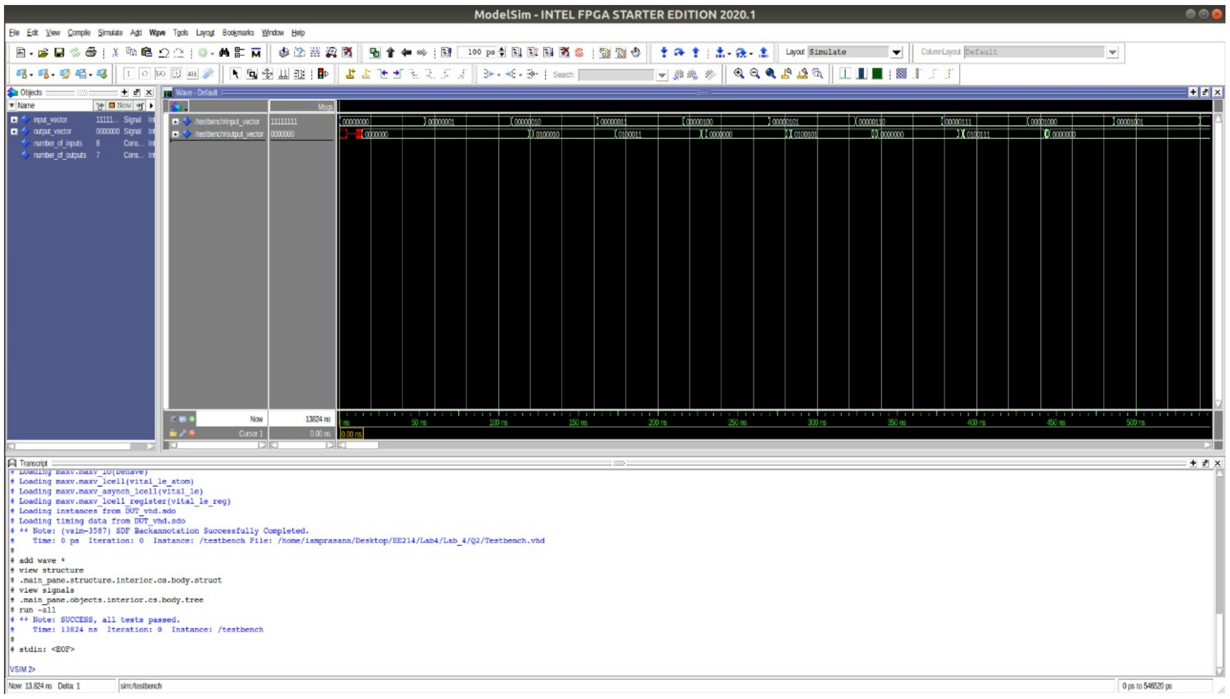
Gate-level Simulation:

Part 1:



```
# Loading maxv.maxv_io(behave)
# Loading maxv.maxv_lcell(vital_le_atom)
# Loading maxv.maxv_asynch_lcell(vital_le)
# Loading maxv.maxv_lcell_register(vital_le_reg)
# Loading instances from DUT_vhd.sdo
# Loading timing data from DUT_vhd.sdo
# ** Note: (vsim-3587) SDF Backannotation Successfully Completed.
#    Time: 0 ps  Iteration: 0  Instance: /testbench File: /home/iamprasann/Desktop/EE214/Lab4/Q1/Testbench.vhd
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 224 ns  Iteration: 0  Instance: /testbench
#
# stdin: <EOF>

VSIM 2>
```

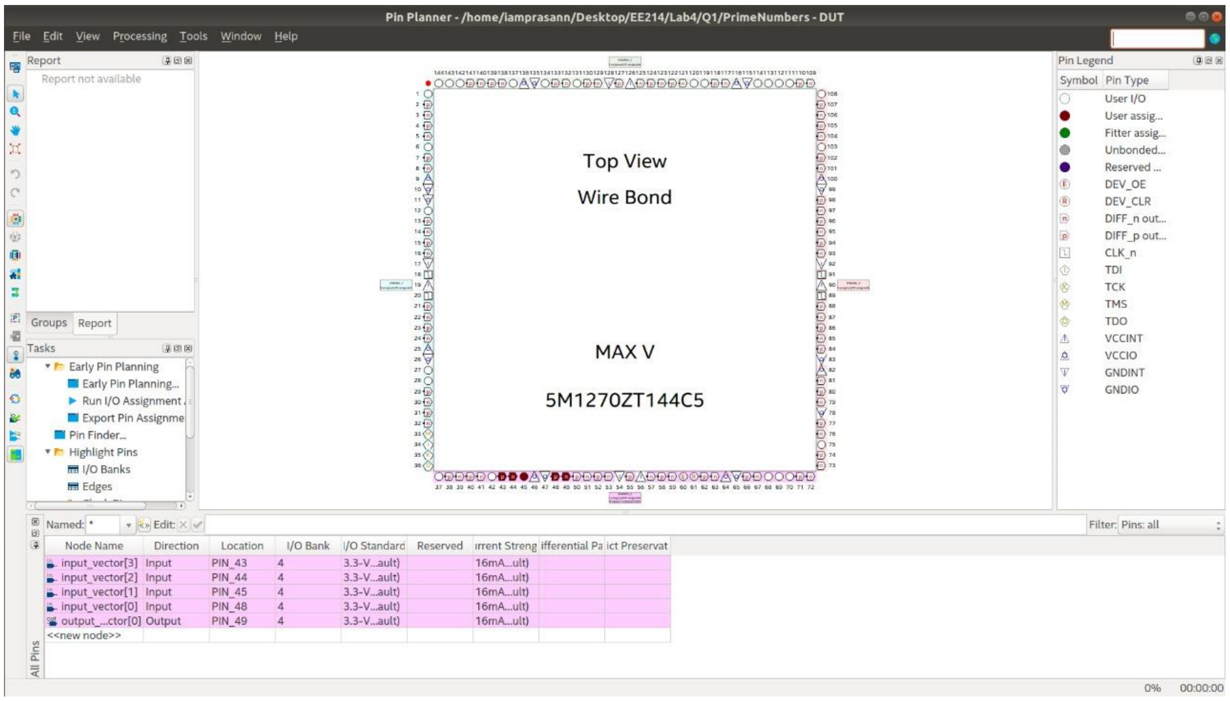
Part 2:



```
# Loading maxv.maxv_io(behave)
# Loading maxv.maxv_lcell(vital_le_atom)
# Loading maxv.maxv_asynch_lcell(vital_le)
# Loading maxv.maxv_lcell_register(vital_le_reg)
# Loading instances from DUT_vhd.sdo
# Loading timing data from DUT_vhd.sdo
# ** Note: (vsim-3587) SDF Backannotation Successfully Completed.
#    Time: 0 ps  Iteration: 0  Instance: /testbench File: /home/iamprasann/Desktop/EE214/Lab4/Lab_4/Q2/Testbench.vhd
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 13824 ns  Iteration: 0  Instance: /testbench
#
# stdin: <EOF>

VSIM 2>
```

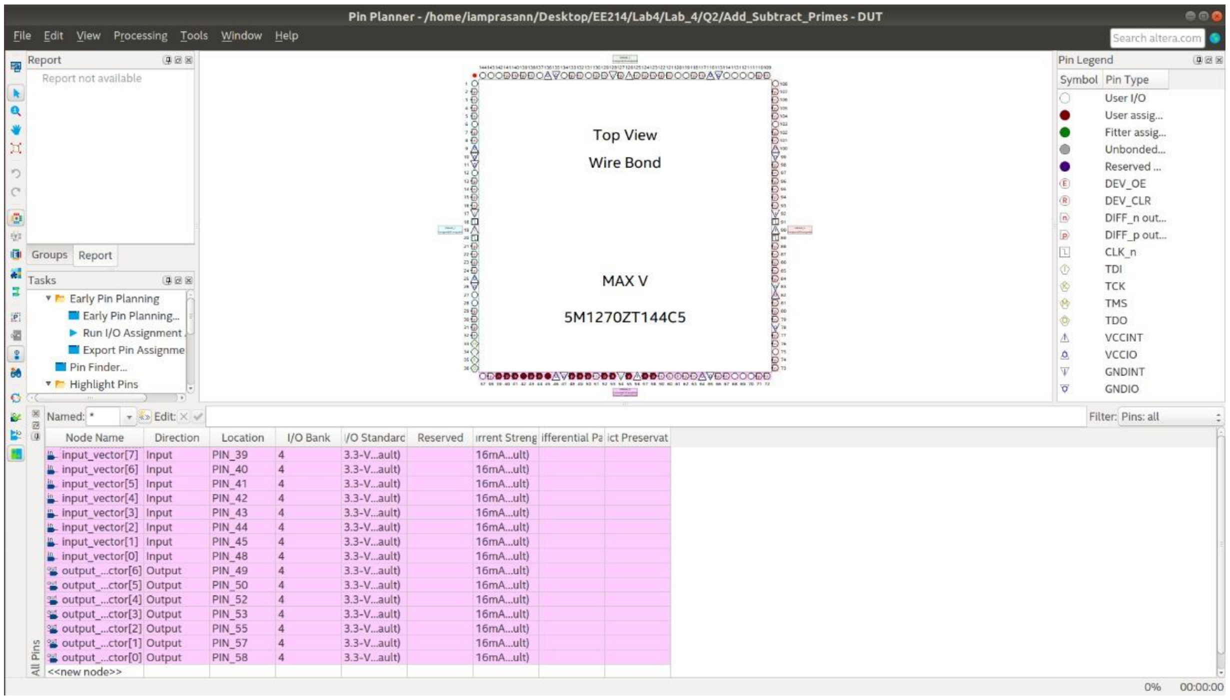
Krypton board:

Pin Planning Part 1:



Node Name	Direction	Location	I/O Bank	/O Standard	Reserved	urrent Streng	ifferential Pa	ict Preservat
input_vector[3]	Input	PIN_43	4	3.3-V...ault)		16mA...ult)		
input_vector[2]	Input	PIN_44	4	3.3-V...ault)		16mA...ult)		
input_vector[1]	Input	PIN_45	4	3.3-V...ault)		16mA...ult)		
input_vector[0]	Input	PIN_48	4	3.3-V...ault)		16mA...ult)		
output_...ctor[0]	Output	PIN_49	4	3.3-V...ault)		16mA...ult)		
<<new node>>								

Pin Planning Part 2:



Node Name	Direction	Location	I/O Bank	/O Standard	Reserved	urrent Streng	ifferential Pa	ict Preservat
input_vector[7]	Input	PIN_39	4	3.3-V...ault)		16mA...ult)		
input_vector[6]	Input	PIN_40	4	3.3-V...ault)		16mA...ult)		
input_vector[5]	Input	PIN_41	4	3.3-V...ault)		16mA...ult)		
input_vector[4]	Input	PIN_42	4	3.3-V...ault)		16mA...ult)		
input_vector[3]	Input	PIN_43	4	3.3-V...ault)		16mA...ult)		
input_vector[2]	Input	PIN_44	4	3.3-V...ault)		16mA...ult)		
input_vector[1]	Input	PIN_45	4	3.3-V...ault)		16mA...ult)		
input_vector[0]	Input	PIN_48	4	3.3-V...ault)		16mA...ult)		
output_...ctor[6]	Output	PIN_49	4	3.3-V...ault)		16mA...ult)		
output_...ctor[5]	Output	PIN_50	4	3.3-V...ault)		16mA...ult)		
output_...ctor[4]	Output	PIN_52	4	3.3-V...ault)		16mA...ult)		
output_...ctor[3]	Output	PIN_53	4	3.3-V...ault)		16mA...ult)		
output_...ctor[2]	Output	PIN_55	4	3.3-V...ault)		16mA...ult)		
output_...ctor[1]	Output	PIN_57	4	3.3-V...ault)		16mA...ult)		
output_...ctor[0]	Output	PIN_58	4	3.3-V...ault)		16mA...ult)		
<<new node>>								

Switches Part 1:

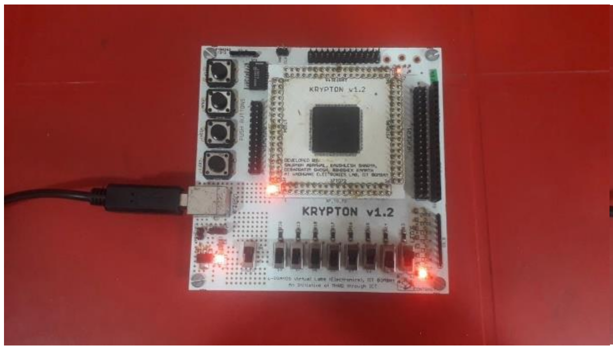
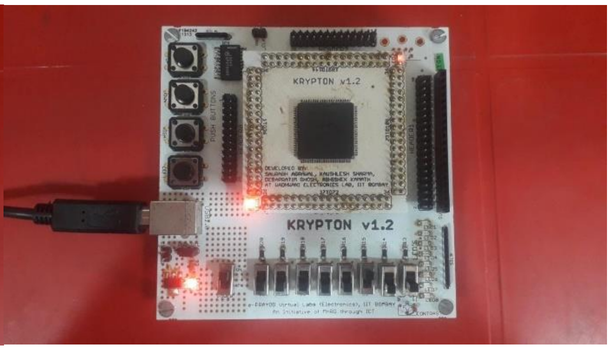



Fig 1 – IP (1101 – 13 decimal) OP Prime (LED ON)
Fig 2 – IP (0100 – 4 decimal) OP Composite (LED OFF)

Switches Part 2:

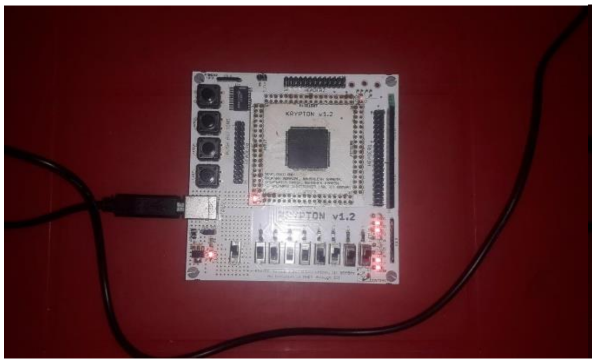
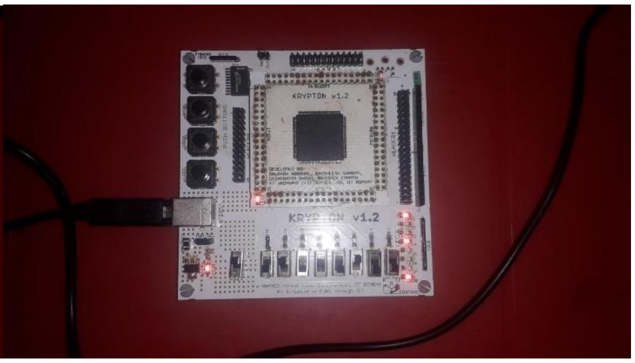



Fig 1 – IP (A=13, B=7) OP(R=6) Both LEDS 7,8 on as 13 and 7 Prime
Fig 2 – IP (A=5, B=4) OP(R=9) LEDS 8 on as only 5 Prime

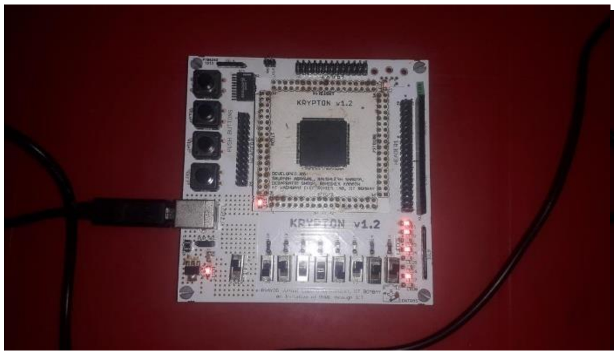
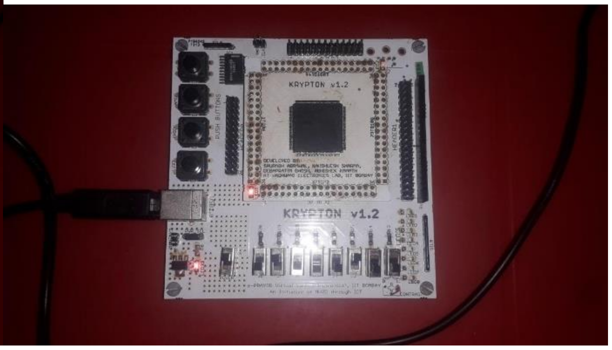
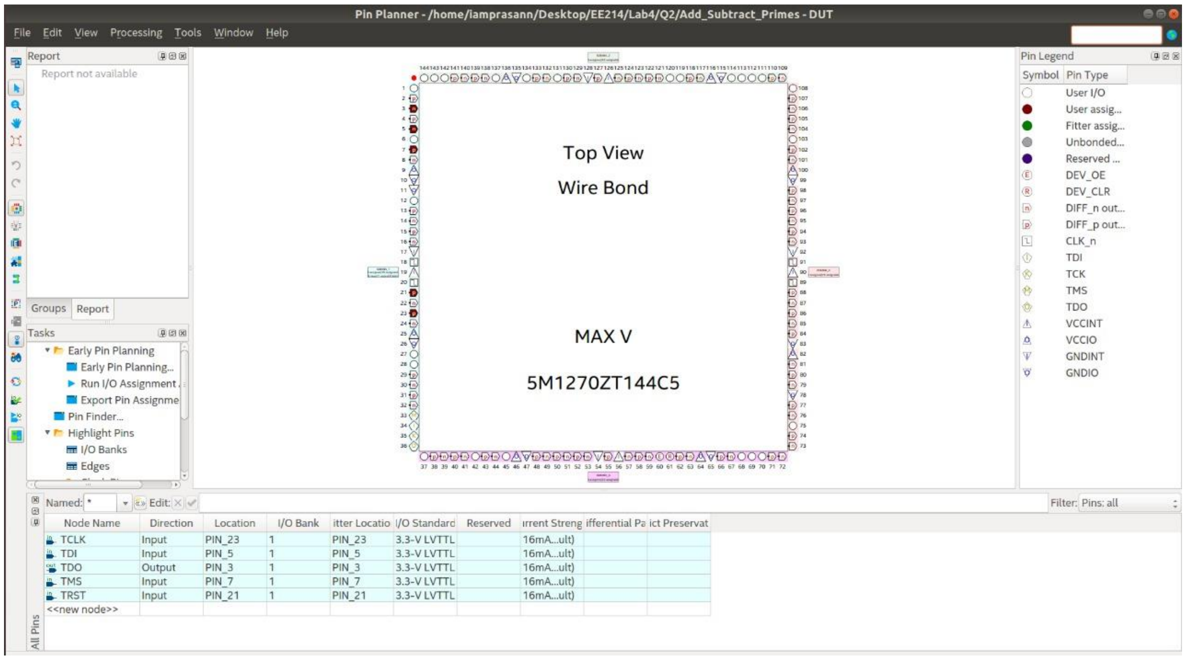



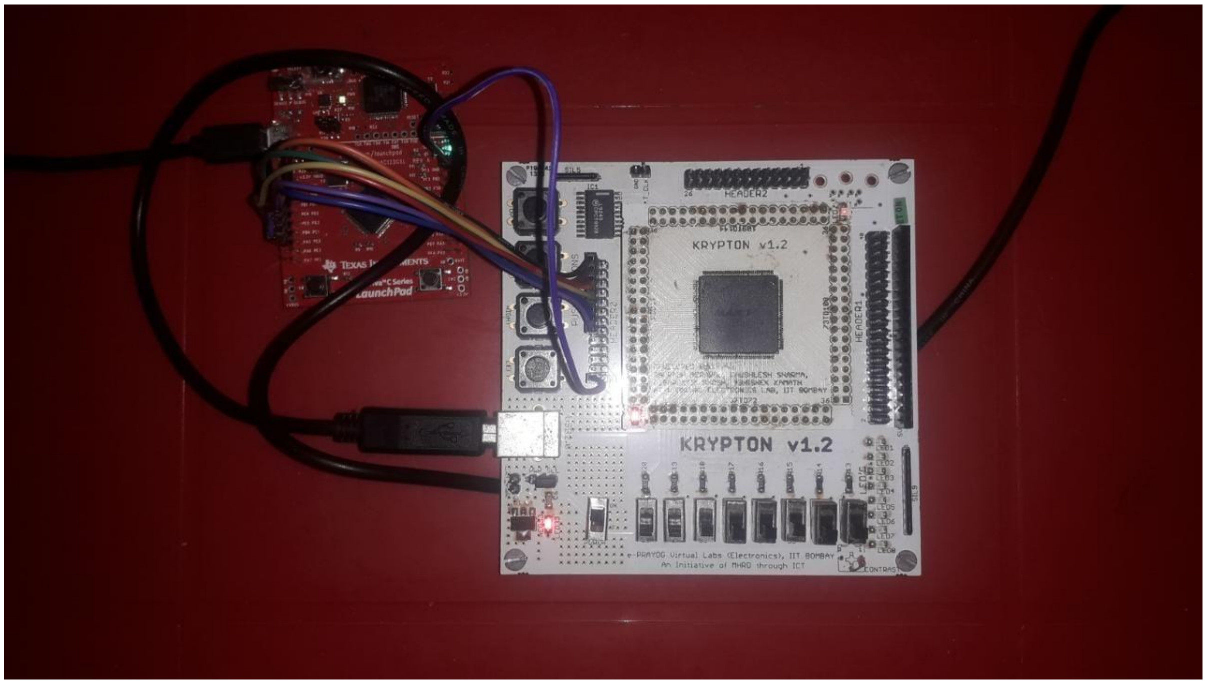
Fig 3 – IP (A=4, B=7) OP(R=11) LEDS 7 on as only 7 Prime
Fig 4 – IP (A=4, B=4) OP(R=0) All LEDS off as A and B = 4 are both composite

Scan Chain Part 2:

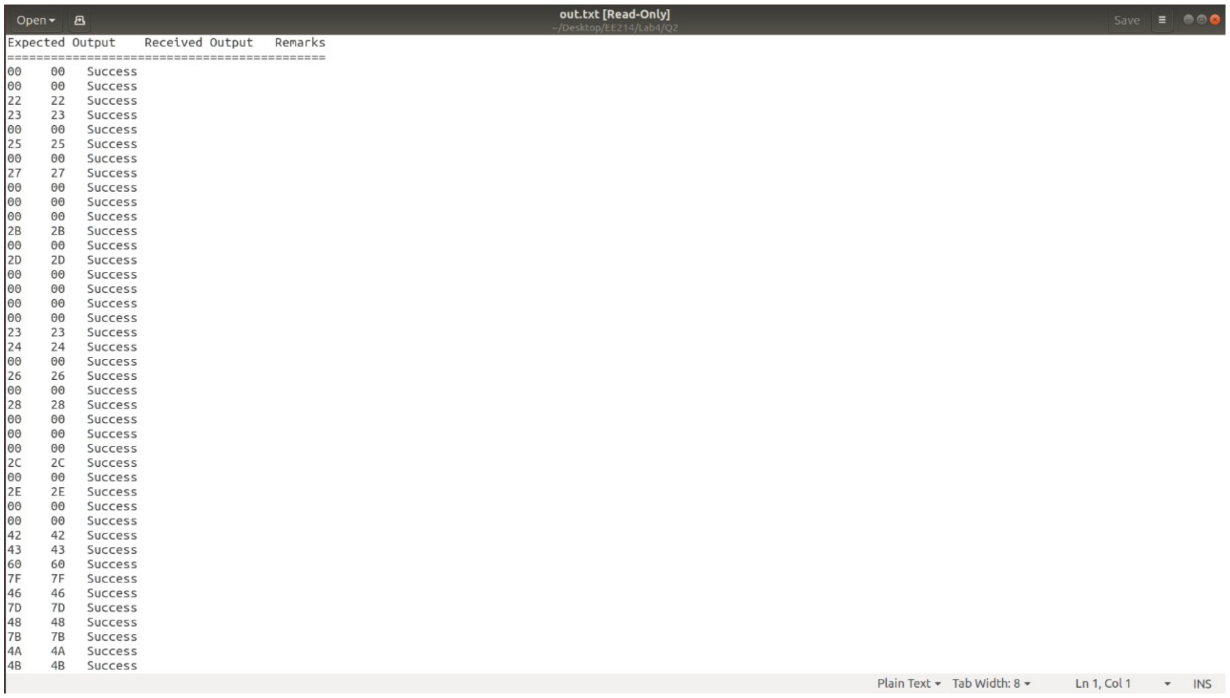

Pin Planner - /home/iamprasann/Desktop/EE214/Lab4/Q2/Add_Subtract_Primes - DUT

Node Name	Direction	Location	I/O Bank	itter Locatio	I/O Standard	Reserved	ırrent Streng	ifferential Pa	ict Preservat
TCLK	Input	PIN_23	1	PIN_23	3.3-V LVTTL		16mA...ult)		
TDI	Input	PIN_5	1	PIN_5	3.3-V LVTTL		16mA...ult)		
TDO	Output	PIN_3	1	PIN_3	3.3-V LVTTL		16mA...ult)		
TMS	Input	PIN_7	1	PIN_7	3.3-V LVTTL		16mA...ult)		
TRST	Input	PIN_21	1	PIN_21	3.3-V LVTTL		16mA...ult)		
<<new node>>									

Connections on Krypton and Tiva:


KRYPTON v1.2
KRYPTON v1.2
HEADER2
HEADER1
Texas Instruments
LaunchPad
e-PRAYOG Virtual Labs (Electronics), IIT BOMBAY
An Initiative of MHRD through ICT

Outputs Passed:



```
Expected Output    Received Output   Remarks
===========================================
00    00   Success
00    00   Success
22    22   Success
23    23   Success
00    00   Success
25    25   Success
00    00   Success
27    27   Success
00    00   Success
00    00   Success
00    00   Success
2B    2B   Success
00    00   Success
2D    2D   Success
00    00   Success
00    00   Success
00    00   Success
00    00   Success
23    23   Success
24    24   Success
00    00   Success
26    26   Success
00    00   Success
28    28   Success
00    00   Success
00    00   Success
00    00   Success
2C    2C   Success
00    00   Success
2E    2E   Success
00    00   Success
00    00   Success
42    42   Success
43    43   Success
60    60   Success
7F    7F   Success
46    46   Success
7D    7D   Success
48    48   Success
7B    7B   Success
4A    4A   Success
4B    4B   Success
```

Observations:

Part 1: The switches S1-S4 acted as input bits with S4 as MSB and S1 as LSB. If the input number was a prime number, LED 8 turns on otherwise it is off. Images attached in Krypton board part.

Part 2: Switch 5-8 acted as input bits for A. The switches 1-4 acted as input bits for B. LED 8 is turned on if A is a prime number and LED 7 is turned on if B is a prime number. The LEDs 1-5 show the output of the operation performed on A and B, represented as R and Rcarry.
Case 1: Both numbers are prime. Both LEDs 8 & 7 are turned on. LEDs 5-1 denotes the result of subtracting ‘b’ from ‘a’. LED 5 denotes the carry bit. LED 4 is the MSB while LED 1 is the LSB of the subtraction result.
Case 2: If either of the numbers is prime. Corresponding LED 8 or 7 is turned on. LEDs 5-1 denote the addition result of ‘a’ and ‘b’. LED 5 denotes the carry bit. LED 4 is the MSB while LED 1 is the LSB of the addition result.
Case 3: If both the numbers are not prime then all LEDs remain off.

References:

NA