Experiment1:

Prakhar Mittal Roll Number 190070046

EE-214, WEL, IIT Bombay

February 10, 2021

## Overview of the experiment:

|  |
| --- |
| Part 1:  The purpose of the experiment was to design a logic circuit to check whether the 4-bit input is a prime number or not and realize it onto the Krypton Board. Return 1 if prime.  I first found out the equivalent function for the logic using K-map, used VHDL to implement the logic and tested it using a Testbench. I also flashed the logic onto the Krypton Board.  Part 2:  The purpose of the experiment was to design a circuit which could perform addition, subtraction, or nothing based on the number of prime inputs given to the circuit, and realize it onto the Krypton Bord.  To perform the experiment, I designed the logic using the part 1 circuit, as well as modifying the 4-bit adder to take input carry and enable as well. The carry and enable inputs were decided using truth table. Finally, the circuit was tested using the Testbench and then also implemented on the Krypton Board and used scan chain to test the outputs.  Part 3:  The purpose of the experiment was to design a logic circuit only using 4:1 MUX to check whether the 4-bit input is a prime number or not and realize it onto the Krypton Board. Return 1 if prime.  To perform the experiment, I created the truth-table of the logic design, and for every combination of bits <a3a2> I wrote the output in terms of bits <a1a0>. Hence, taking <a3a2> as the switch bits, we can design a MUX circuit. |

## Approach to the experiment:

|  |
| --- |
| Part 1:  I used K-map to find out the logical expression of the logic and then implemented the function using signals, and, not, or gates.    Part 2:  I designed the logic using the part 1 circuit, as well as modifying the 4-bit adder to take input carry and enable as well. The carry and enable inputs were decided using truth table.    Part 3:  I created the truth-table of the logic design, and for every combination of bits <a3a2> I wrote the output in terms of bits <a1a0>. Hence, taking <a3a2> as the switch bits, we can design a MUX circuit. |

## Design document and VHDL code if relevant:

|  |
| --- |
| Part 1:  prime\_check: (a = input 4-bit vector, f = output\_bit)    -- calculate inverted bits  u1 : INVERTER port map (A=> a(1), Y=> a1\_not);  u2 : INVERTER port map (A=> a(2), Y=> a2\_not);  u3 : INVERTER port map (A=> a(3), Y=> a3\_not);    -- Calculate Or outputs  or1: OR\_2 port map (A=> a(0), B=> a3\_not, Y=> o1);  or2: OR\_2 port map (A=> a3\_not, B=> a1\_not, Y=> o2);    -- Calculate And outputs  and1: AND\_2 port map(A=> a(1), B=> a2\_not, Y=> p1);  and2: AND\_2 port map(A=> a(0), B=> a(2), Y=> p2);  and3: AND\_2 port map(A=> p1, B=> o1, Y=> p3);  and4: AND\_2 port map(A=> p2, B=> o2, Y=> p4);    -- Calculate final Output  final: OR\_2 port map (A=> p3, B=> p4, Y=> f);  Part 2: (A, B = input 4-bit vectors, Final = output 4-bit vector)  -- check prime-ness  u1: prime\_check port map(a=>A, f=>led8);  u2: prime\_check port map(a=>B, f=>led7);    Final(6) <= led8;  Final(5) <= led7;    -- intermediate signals  or1: OR\_2 port map(A=>led8, B=>led7, Y=> s1);  xor1: XNOR\_2 port map(A=>led8, B=>led7, Y=> s0);    -- Take output from 4-bit adder  add: ripple\_carry port map (A=> A, B=> B, C\_in=>s0, en=> s1, S=> Final(3 downto 0), C\_out=> Final(4));  Part 3: (a = input 4-bit vector, f = output\_bit)  -- Calculate inverted bits  u1: INVERTER port map (A=> a(1), Y=> a1\_not);  -- Intermediate ANDs  a1: AND\_2 port map (A=> a(1), B=> a(0), Y=>mux2);  a2: AND\_2 port map (A=> a1\_not, B=> a(0), Y=>mux3);  -- Give values to mux\_vector  mux\_vector(0) <= a(1);  mux\_vector(1) <= a(0);  mux\_vector(2) <= mux2;  mux\_vector(3) <= mux3;    -- Apply 4:1 MUX  mux: mux\_41 port map (in\_vector=> mux\_vector, S2=> a(3), S1=> a(2), EN=> '1',  Y=> f); |

## RTL View:

|  |
| --- |
| Part 1:    Part 2:    Part 3: |

## DUT Input/Output Format:

|  |
| --- |
| Part 1:  Input: <a3a2a1a0><space><out\_bit><space><mask>  Output: <a3a2a1a0><space><out\_bit>  Testcases:  Input  0000 0 1  0001 0 1  0010 1 1  Output  0000 0  0001 0  0010 1  Part 2:  Input: <a3a2a1a0b3b2b1b0><space><l8l7l5l4l3l2l1 ><space><mask>  Output: <a3a2a1a0b3b2b1b0><space><l8l7l5l4l3l2l1 >  Testcases:  Input  00000000 0000000 1111111  00000001 0000000 1111111  00000010 0100010 1111111  Output  00000000 0000000  00000001 0000000  00000010 0100010  Part 3:  Input: <a3a2a1a0><space><out\_bit><space><mask>  Output: <a3a2a1a0><space><out\_bit>  Testcases:  Input  0000 0 1  0001 0 1  0010 1 1  Output  0000 0  0001 0  0010 1 |

## RTL Simulation:

|  |
| --- |
| Part 1:    Part 2:    Part 3: |

## Gate-level Simulation:

|  |
| --- |
| Part 1:    Part 2:    Part 3: |

## Krypton board\*:

|  |
| --- |
| Part 1:      Part 2:      Part 3: |

## Observations\*:

|  |
| --- |
| Part 1:  The led8 lights up whenever the input is prime, i.e. one of 2,3,5,7,11,13.  Part 2:  No output led lights up when both 4-bit inputs are non-prime. Led8 lights up whenever upper nibble is prime. Led7 lights up whenever lower nibble is prime. The first 5 led light up according to the sum A+B, if only one of them is prime, or difference A-B, if both of them are prime.  Part 3:  The led8 lights up whenever the input is prime, i.e. one of 2,3,5,7,11,13. |

## References:

|  |
| --- |
| Nil |

\* To be submitted after the tutorial on “Using Krypton”.