March 5th 2025

Alp Efe Kılıçarslan, 22402390

EEE 102-2

LAB-04: Arithmetic Logic Unit

Purpose:

The goal of this assignment was to design and implement an arithmetic logic unit (ALU) which was capable of eight different operations that included one bitwise and one shift operation. The design was meant to be created using VHDL on Vivado and implemented modularly on a BASYS3 FPGA after the programs were simulated and analyzed in said software.

Design Specifications:

Three standard logic inputs were utilized to select the desired operation for the logic unit. Each specific state of said three pins were mapped to an operation and labeled "select". The selected operations were AND gate, OR gate, XOR gate, left logical shift, right logical shift, addition, subtraction and decrement of one. These functions were chosen to include addition, subtraction one bitwise operation and one shifting operation. For the inputs of the ALU, two sets of four standard logical were allocated and labelled A and B respectively. For the outputs of the functions, five bits were allocated to account for extra carry outputs and similar cases. Each module was implemented in a hierarchical manner in which a main module encompassed the operators, a four-bit adder, a full adder and a half adder.

After the logic design was fully written and coded, each input and output were mapped onto the BASYS3 using a constraint file. For the mode select, switches R2, T1 and U1 were allocated. For A and B, switches V17, V16, W16, W17 and V15, W14, W13, V2 were assigned respectively. For the output, LEDs U16, E19, U19, V19 and W18 were used to show the operation's results. By switching the inputs to the on positions, a Boolean "On" signal was able to be provided for the required functions.

Methodology:

Task-1) Each module for the ALU - AND gate, OR gate, XOR gate, left logical shift, right logical shift, addition, subtraction and decrement of one- were coded and checked for errors. After a working program was achieved, the necessary files were set as the hierarchical top and a testbench that iterated through every input was simulated. The waveforms were analyzed to check if they matched specific cases for each operation (Table 1). The RTL schematic was generated was generated and recorded. The program was run on the BASYS3. The predetermined inputs were put through the FPGA and their outputs were checked by comparing the results with the simulations. Different states of the BASYS3 were photographed and recorded.

Selection State	Function	Input Example	Output Example
000	AND Gate	1101, 1011	0 1001
001	OR Gate	0001, 0101	0 0101
010	XOR Gate	1110, 0111	0 1001
011	Left Shift	0111	0 1110
100	Right Shift	1111	0 0111
101	Addition	0110, 1011	1 0001
110	Subtraction	1001, 0011	0 0110
111	Decrement	1000	0 0111

Table 1: Selection modes for each operation and example inputs and outputs

Results:

Task-1) An RTL schematic was generated for each individual operation and the main program. For the AND function, the program slices each 4-bit input to four 1-bit inputs and outputs their result to the LED with the corresponding index (Figure 1.1). For the OR function, the inputs are sliced similarly and put through an OR gate whose output is connected to the related LED (Figure 1.2). For the XOR function, both inputs' equivalent indexed digits are put through the related gate and the output is also connected to the related LED (Figure 1.3). For the Left shift function, which is a shifting operation, takes a singular input instead of two and shifts the indexes of the input by positive one and outputs the result (Figure 1.4) whereas right shift, which is also a shifting operator, shifts the index by negative one (Figure 1.5). These shifting operations give the effect of visually shifting the input to the left or right respectively. The addition operation uses two other submodules -full adder and half adder- to make the code easier to read. The equal indexed inputs are put through four full adders -which consist of one OR, two AND and two XOR gates each- and the resulting outputs are represented. If the sum of each index exceeds the possible maximum for the related digit, a carry is taken and given to the following index to complete the operation (Figure 1.6). The subtractor operator works similar to the addition operator, where it takes two four-bit inputs and outputs a five-bit output. For positive outputs, only the first four are used whereas for negative outputs, the last bit denotes that the number is negative and the rest signify the magnitude of the number (Figure 1.7). For the last operator, which is the decrement, a singular input is used and subtracted by one. If the input is equal to zero, a similar system to the subtractor module represents the output as a negative number (Figure 1.8). All of these modules are taken and selected as desired using a three-bit input, similar to a multiplexer. By modifying this specific input, the function that is correspondent in Table 1 is utilized and the inputs are reformed as such (Figure 1.9).

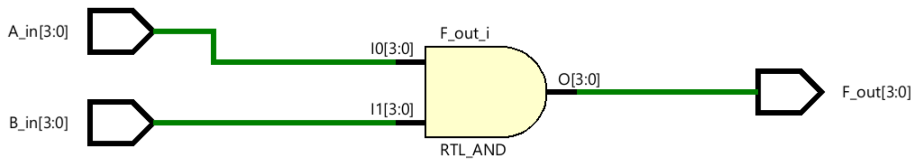

Figure 1.1: RTL Schematic for the AND Gate

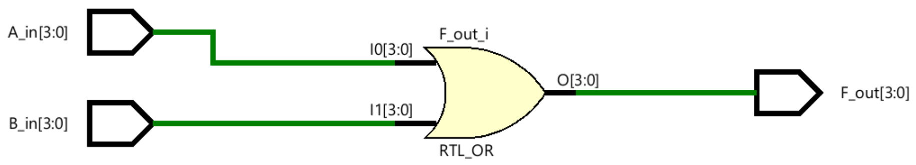

Figure 1.2: RTL Schematic for the OR Gate

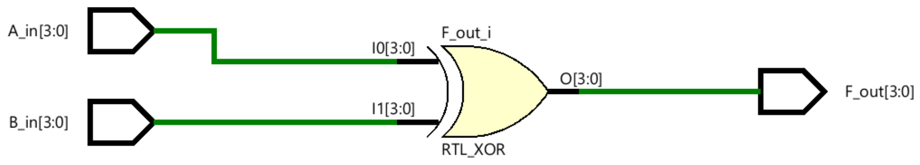

Figure 1.3: RTL Schematic for the XOR Gate

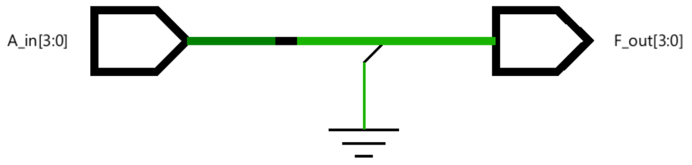

Figure 1.4: RTL Schematic for Left Shift

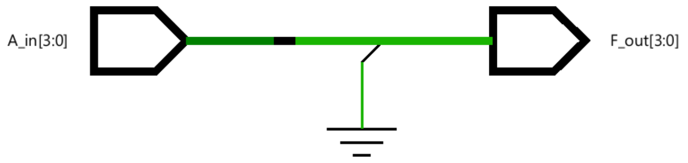

Figure 1.5: RTL Schematic for Right Shift

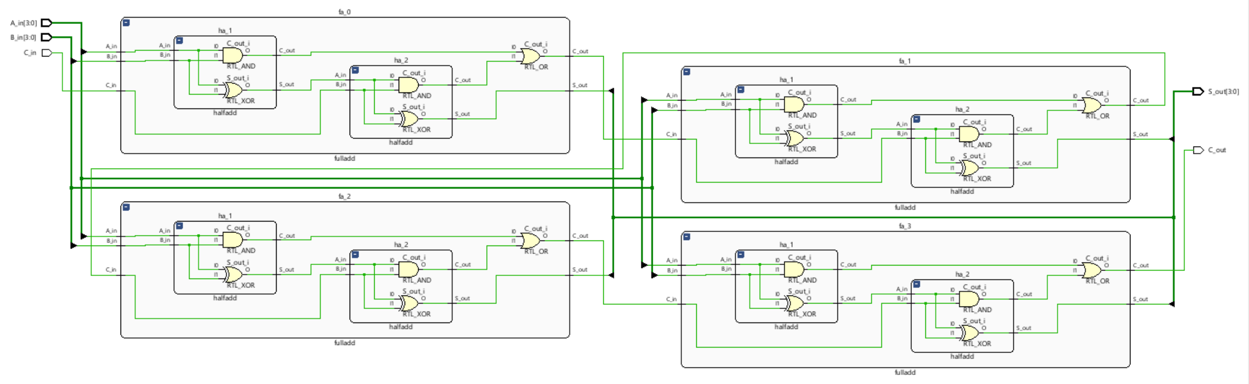

Figure 1.6: RTL Schematic for Addition

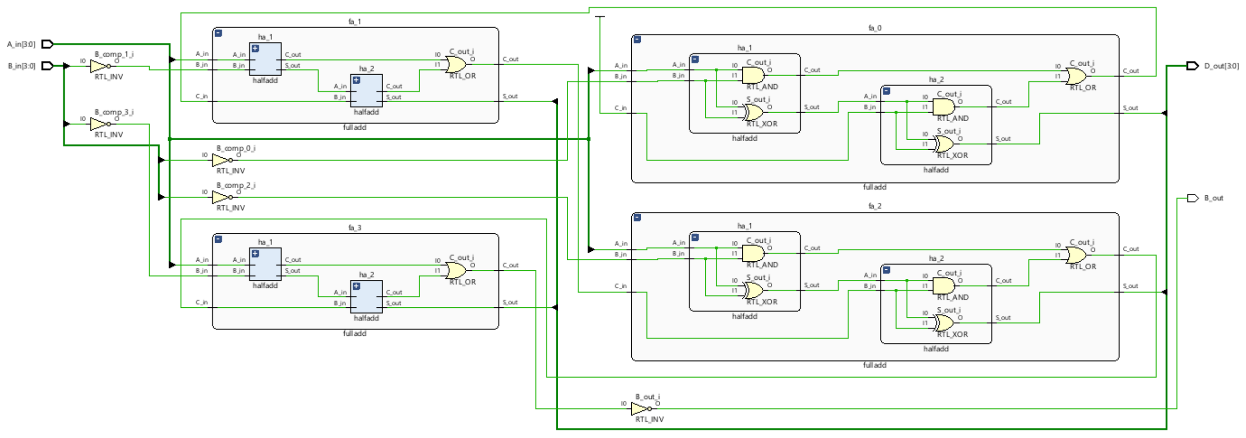

Figure 1.7: RTL Schematic for Subtraction

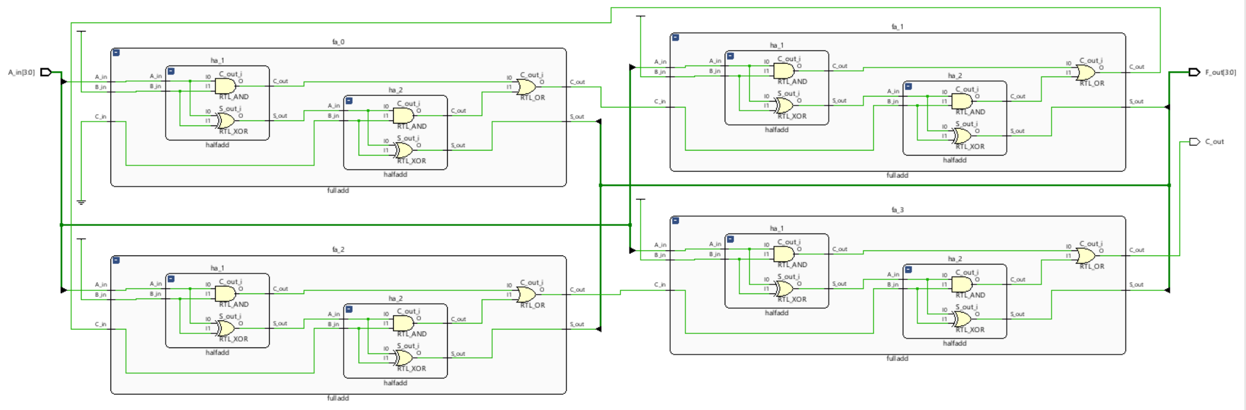

Figure 1.8: RTL Schematic for Decrement

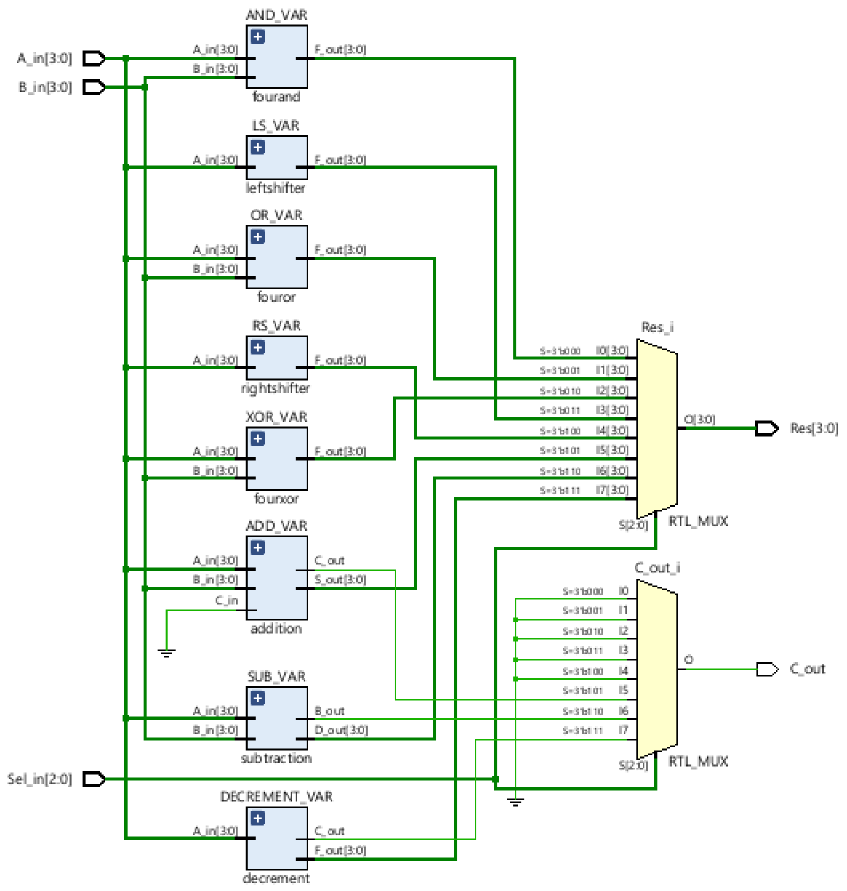

Figure 1.9: RTL Schematic for the ALU

After each schematic was generated, a testbench that iterated through every possible combination of A, B and select was simulated and ran for the required amount of time (Figure 1.10). The testbench's results were compartmentalized in octants where each octant represented a different selection. The simulation was checked at certain points of time, specifically at the given input values of Table 1 and recorded (Figure 1.11-1.18). The obtained values were observed to be identical to the expected results.

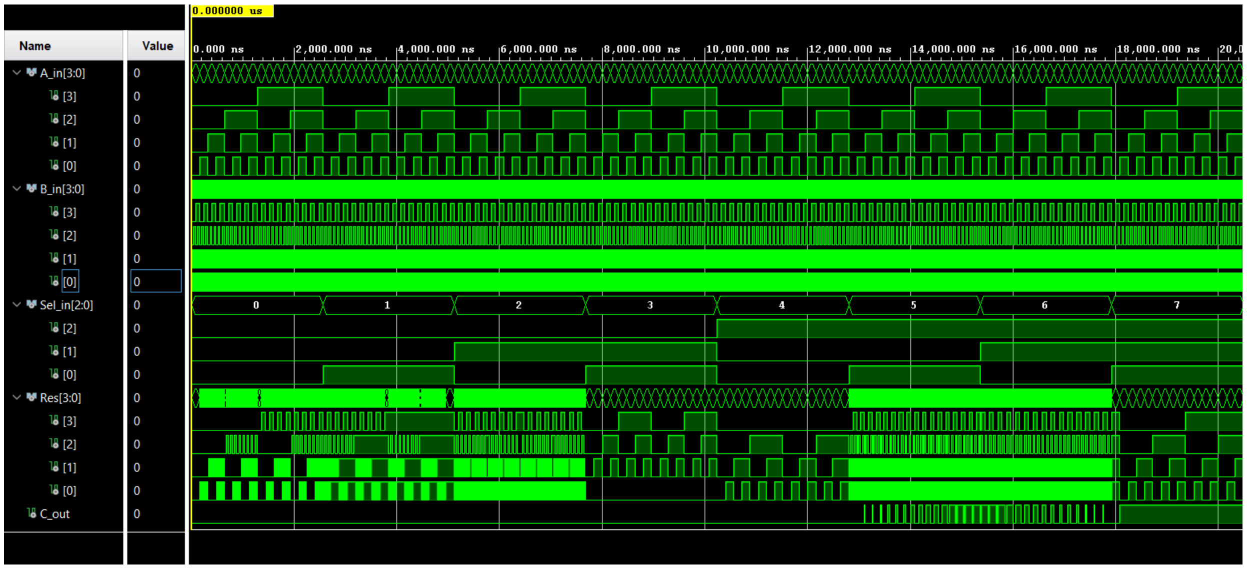

Figure 1.10: Default State of the obtained Testbench Simulation

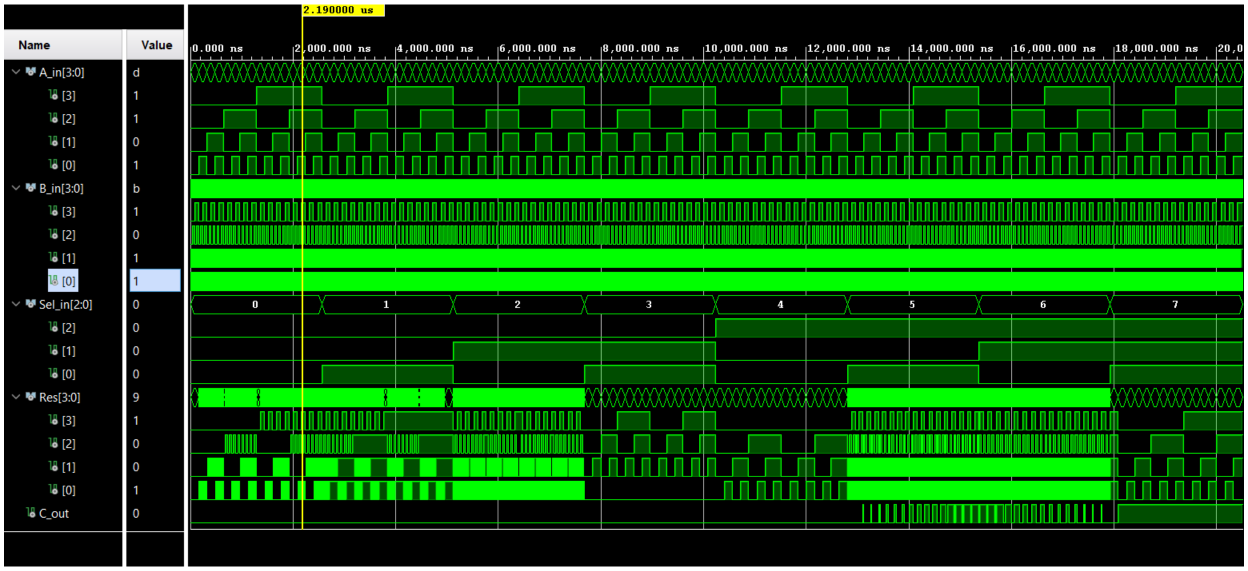

Figure 1.11: Simulation when select = “000”, A = “1101” and B = “1011”

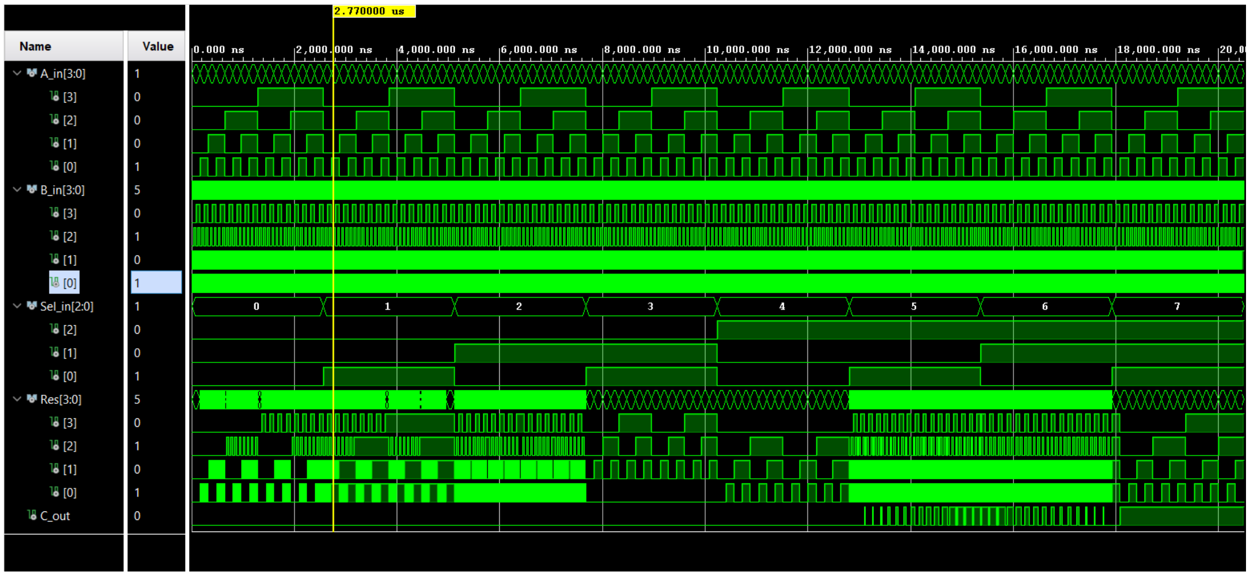

Figure 1.12: Simulation when select = “001”, A = “0001” and B = “0101”

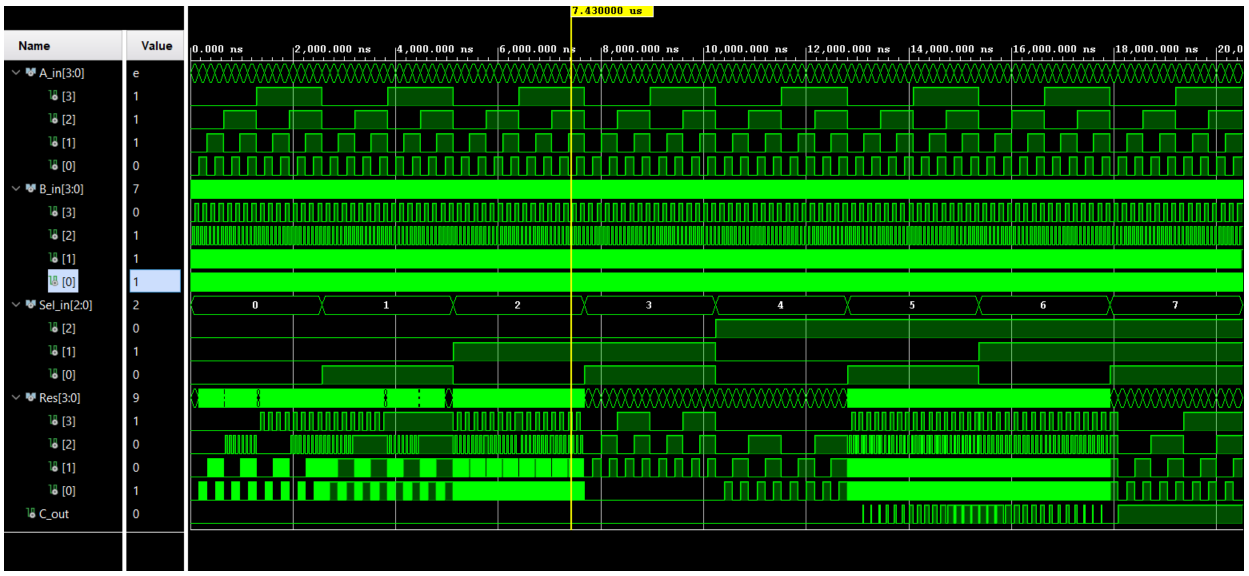

Figure 1.13: Simulation when select = “010”, A = “1101” and B = “1011”

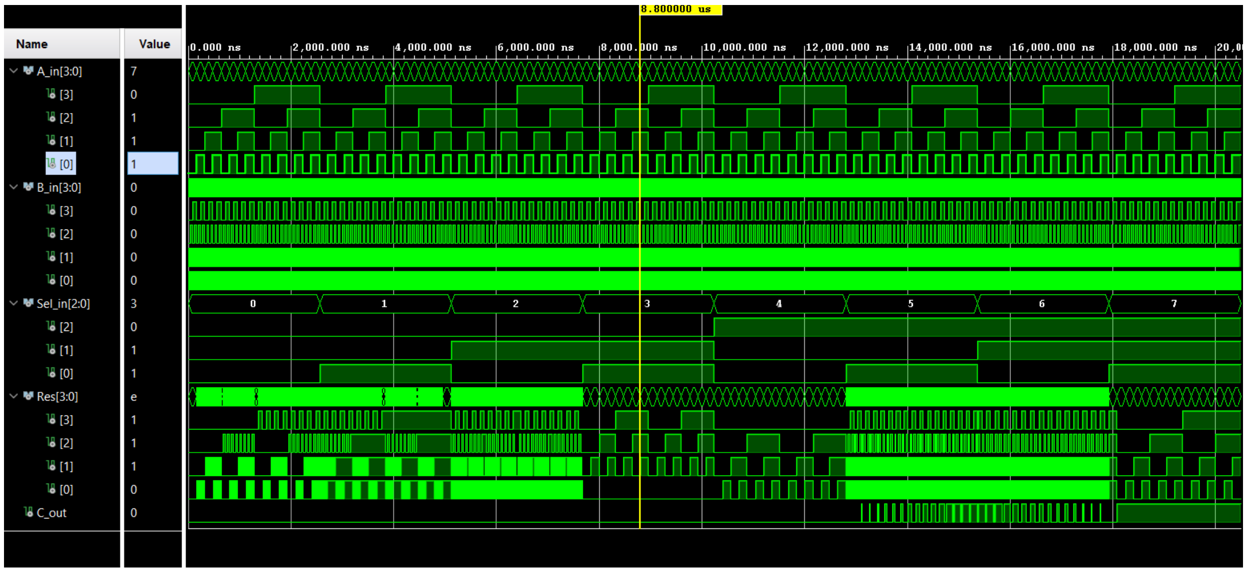

Figure 1.14: Simulation when select = “011” and A = “0111”

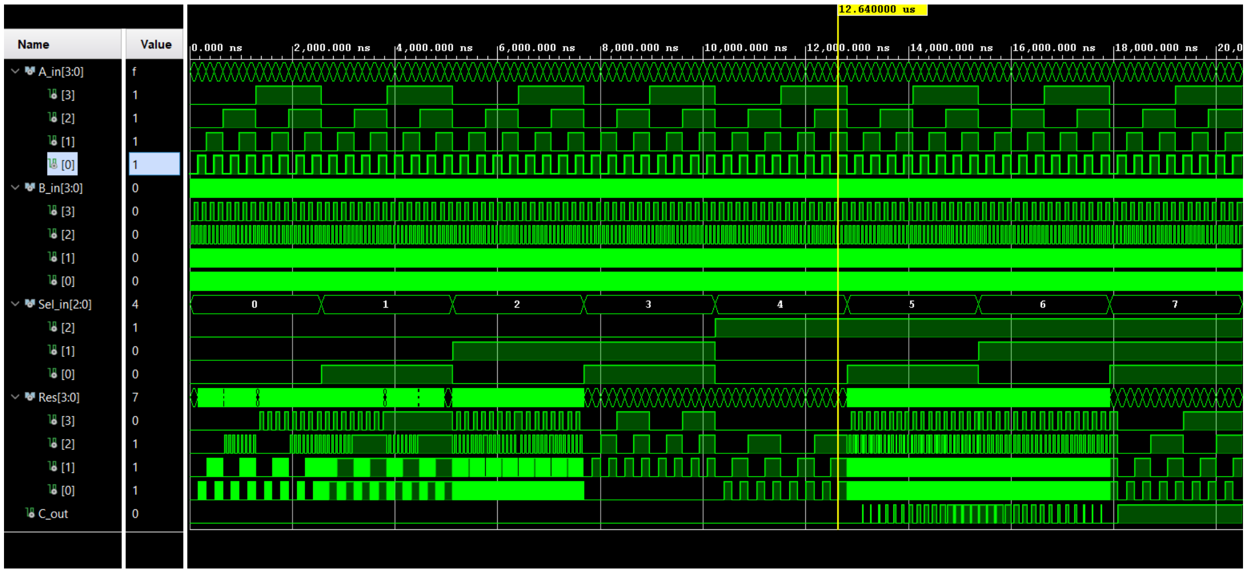

Figure 1.15: Simulation when select = “100” and A = “1111”

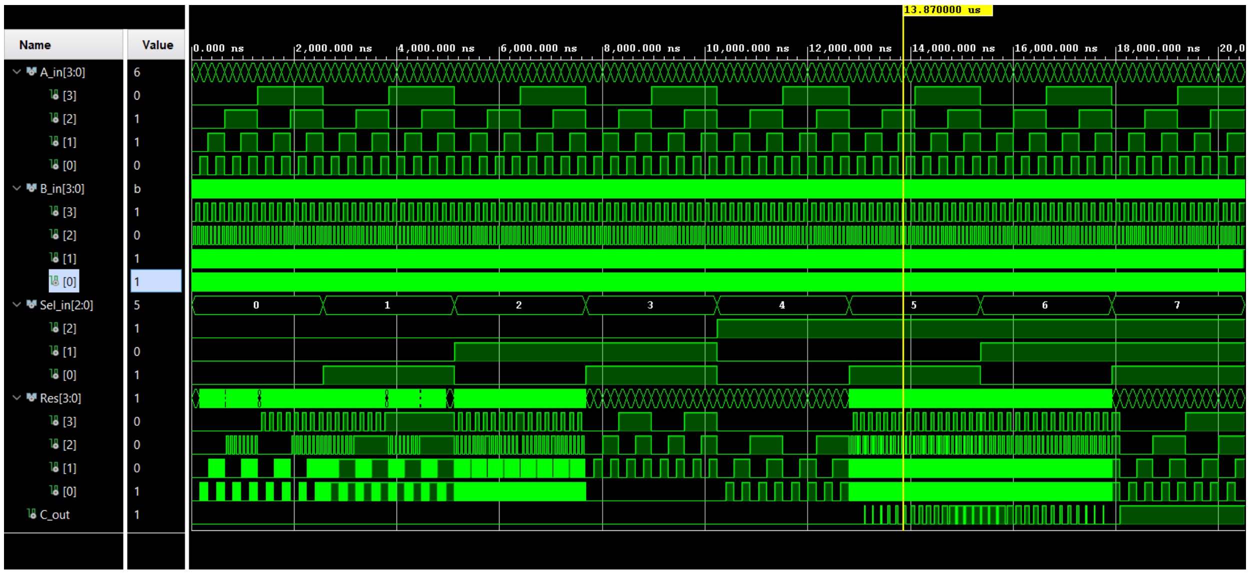

Figure 1.16: Simulation when select = “101”, A = “0110” and B = “1011”

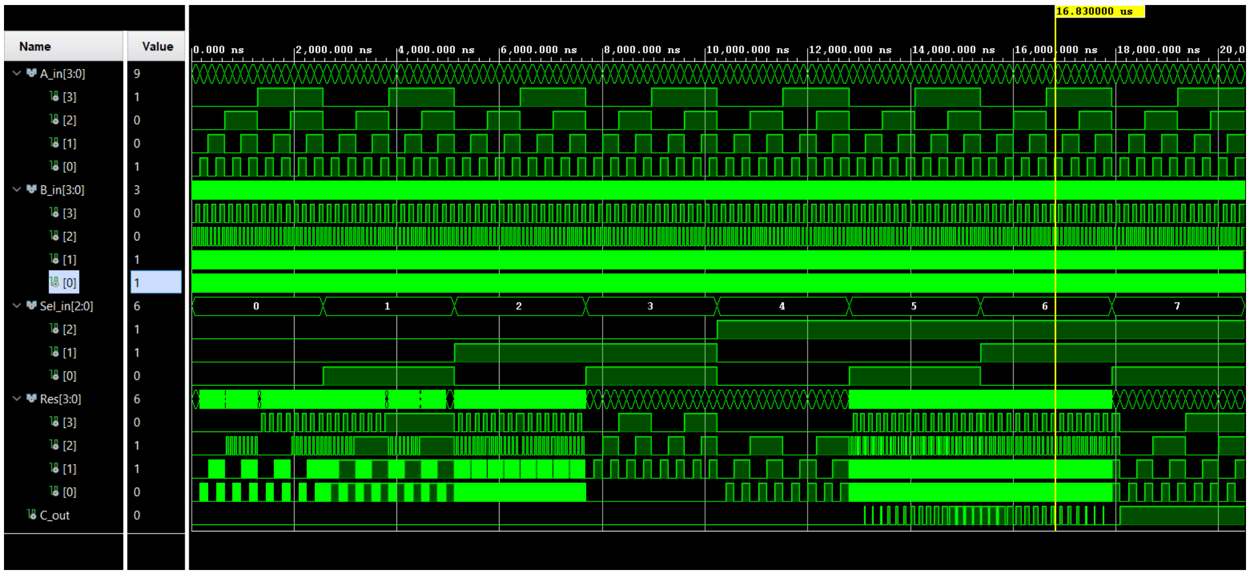

Figure 1.17: Simulation when select = “110”, A = “1001” and B = “0011”

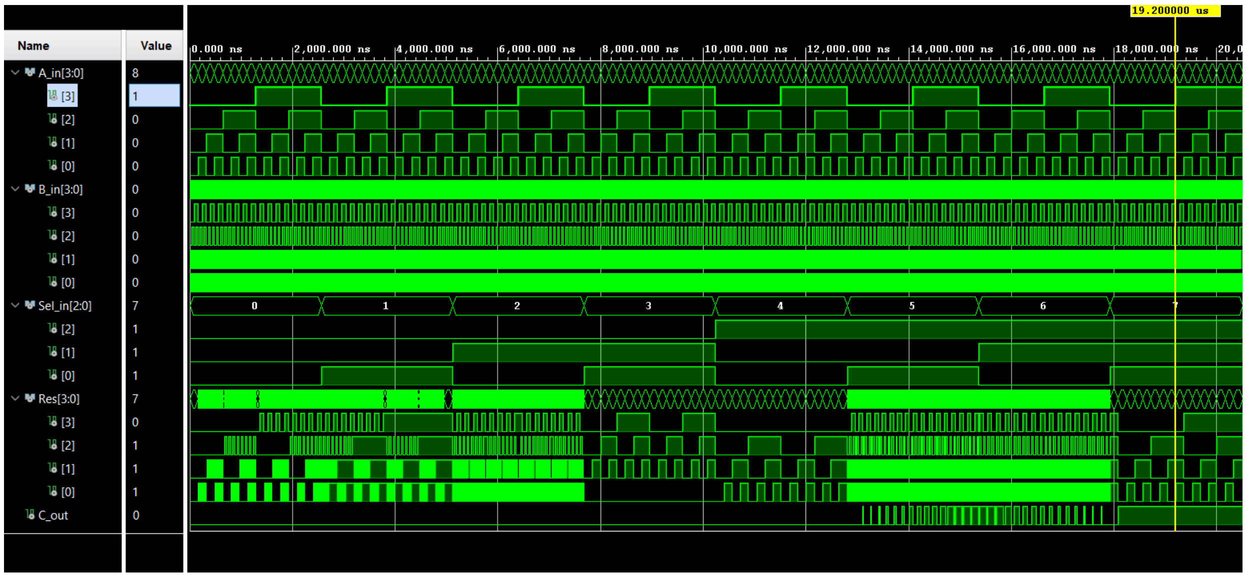

Figure 1.18: Simulation when select = "111" and A = "1000"

After each case for the simulation was analyzed, a constraint file defining every required input and output to a respective switch and LED was written. A bitstream was generated with the modules correctly ordered on the program. The BASYS3 FPGA was connected to the computer with the required code and the device was programmed. Cases representing the truth table, and indirectly the testbench simulation, were created on the board. Each state was photographed and compared to the simulation (Figure 1.19-1.27). Each representation on the BASYS3 was observed to be equivalent to the testbench simulation.

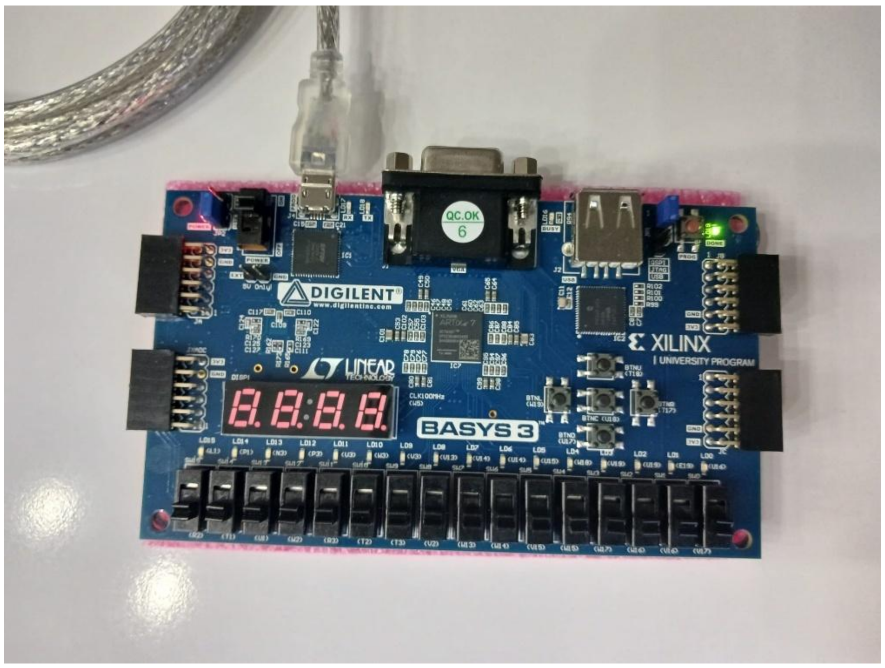

Figure 1.19: Default State of the Board

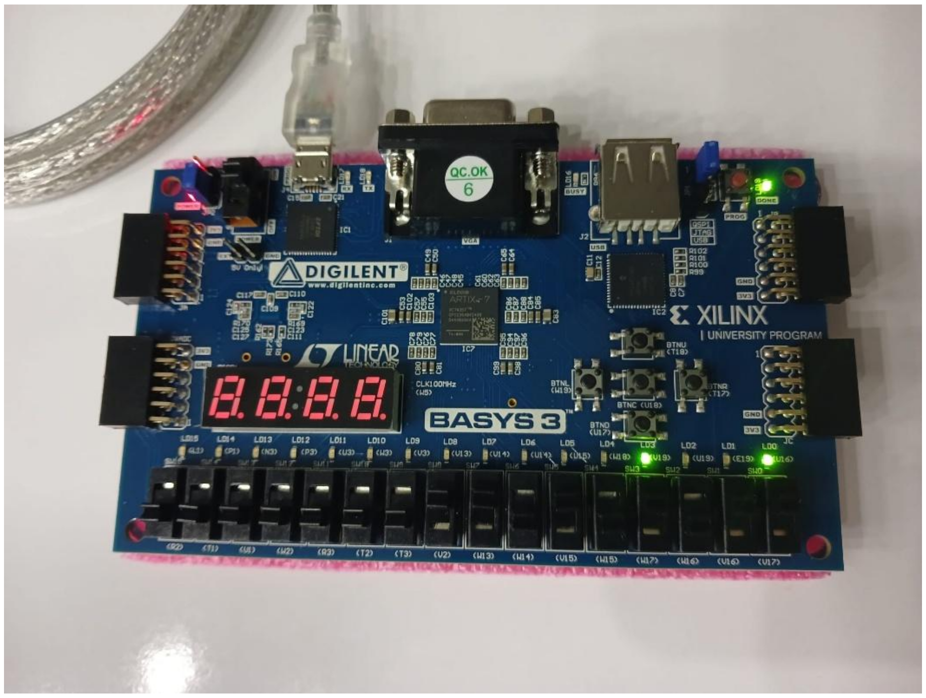
Figure 1.20: State of the board when inputs of the AND Gate operator of Table 1 were entered

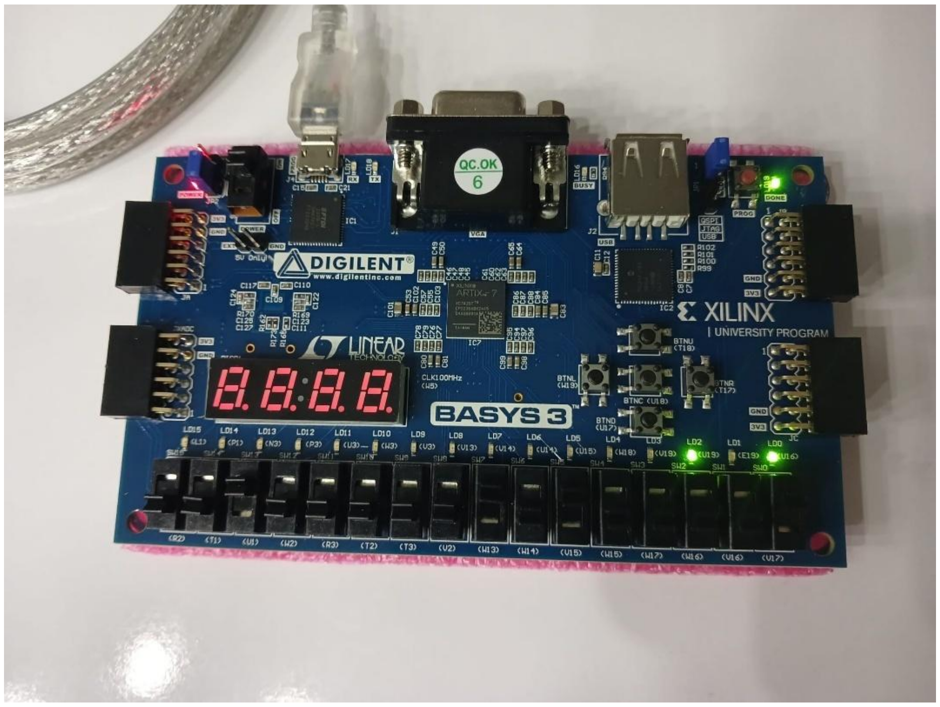
Figure 1.21: State of the board when inputs of the OR Gate operator of Table 1 were entered

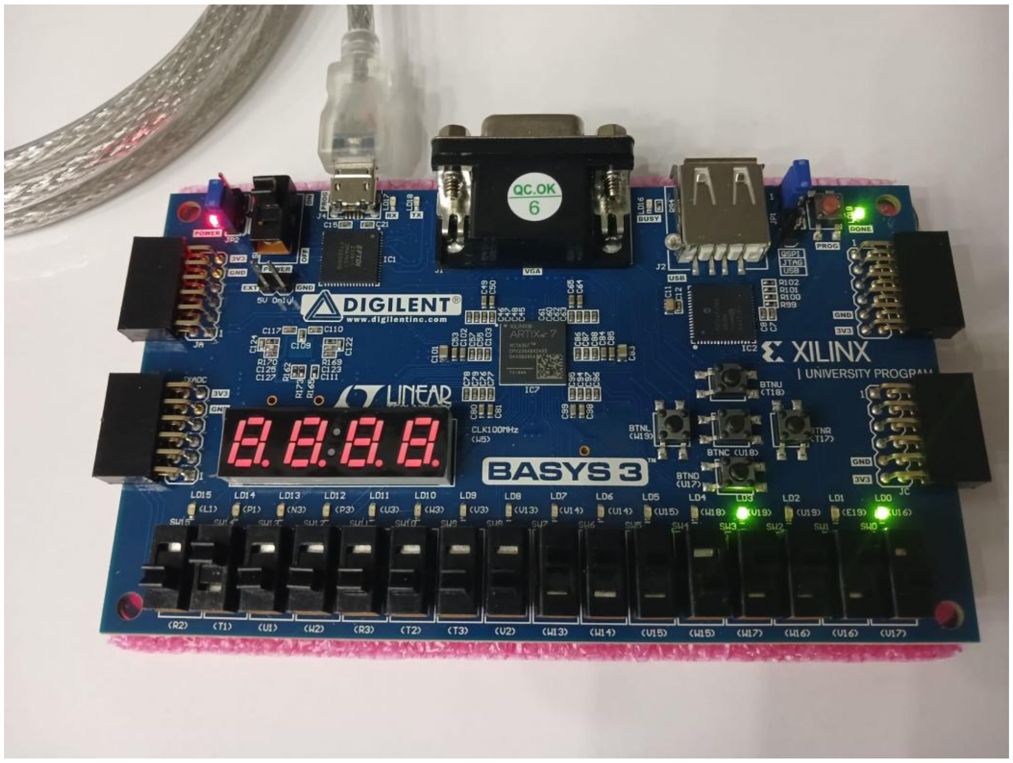
Figure 1.22: State of the board when inputs of the XOR Gate operator of Table 1 were entered

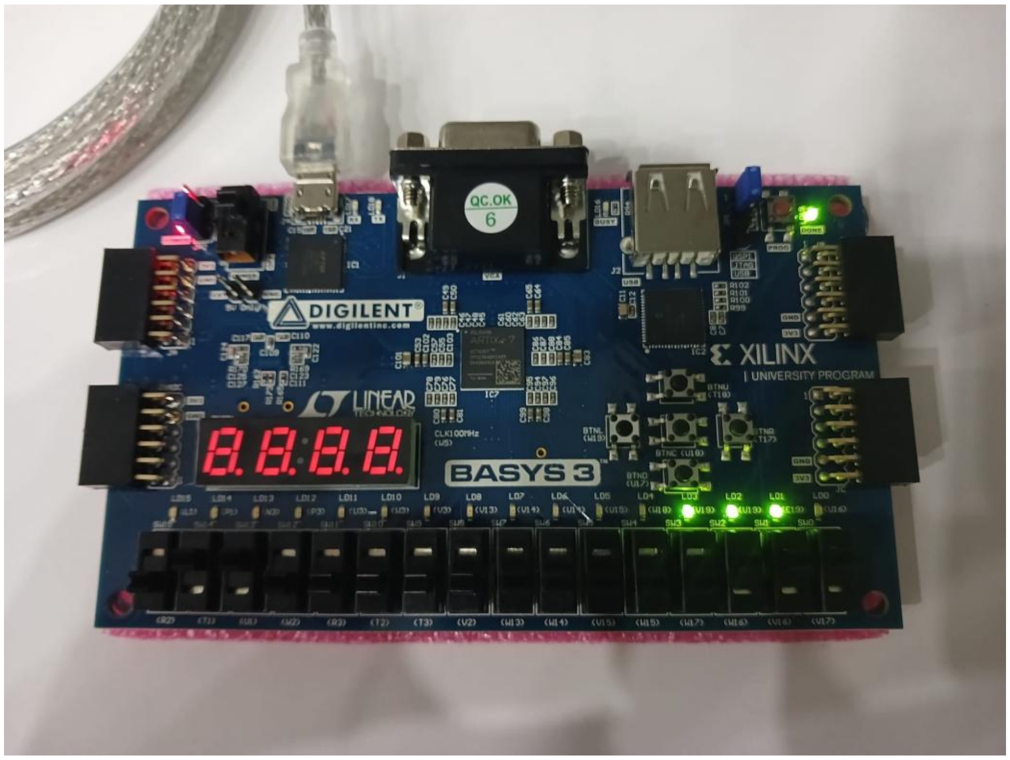
Figure 1.23: State of the board when inputs of the Left Shift operator of Table 1 were entered

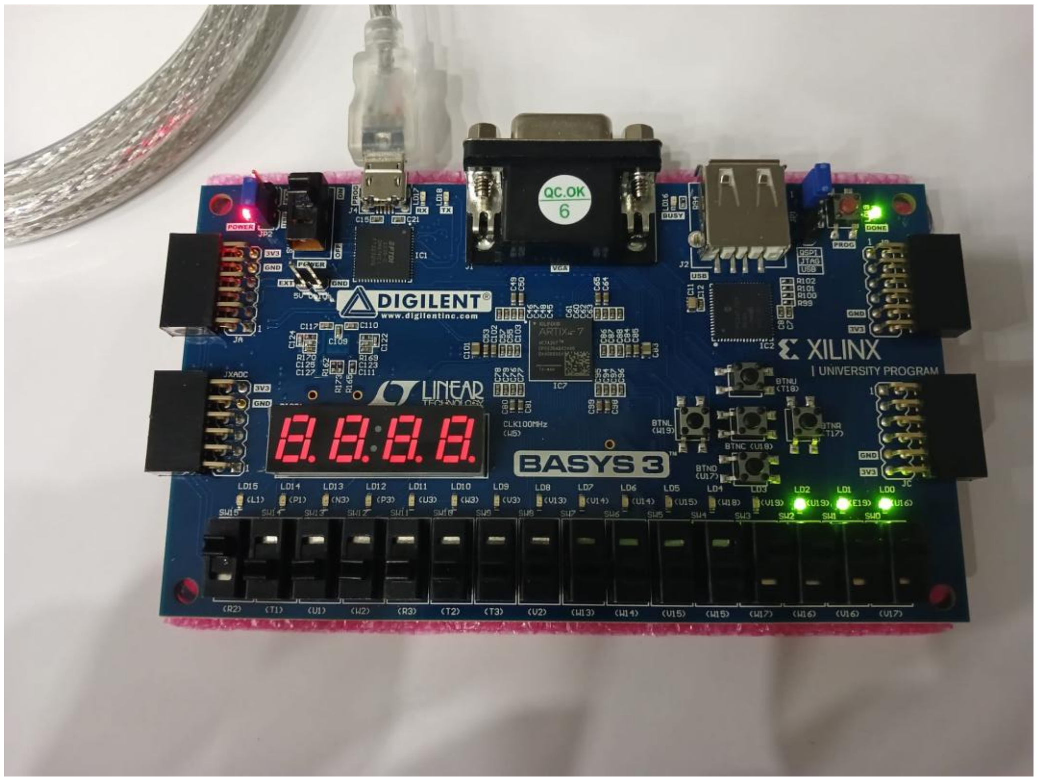

Figure 1.24: State of the board when inputs of the Right Shift operator of Table 1 were entered

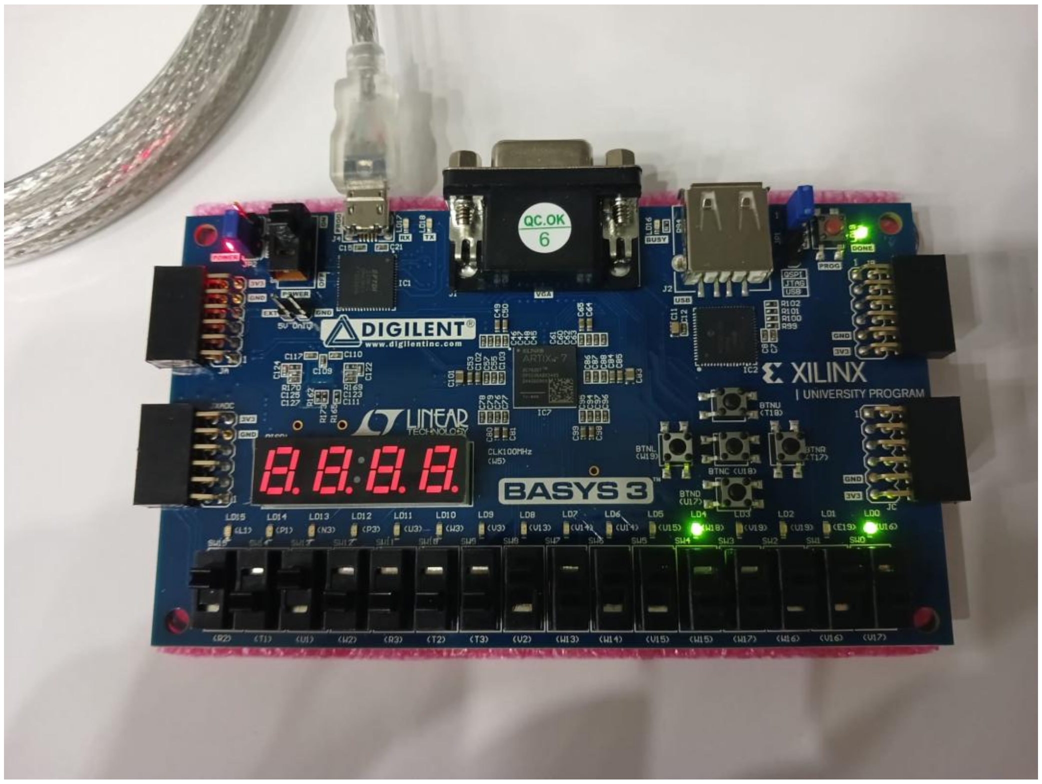

Figure 1.25: State of the board when inputs of the Addition operator of Table 1 were entered

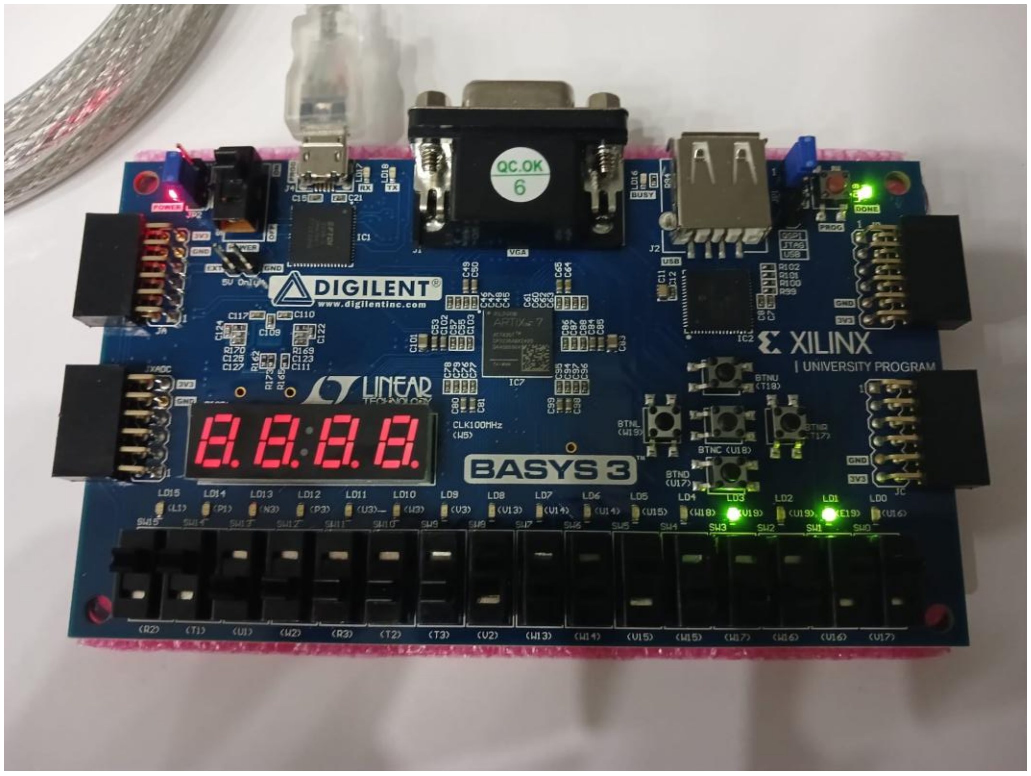
Figure 1.26: State of the board when inputs of the Subtraction operator of Table 1 were entered

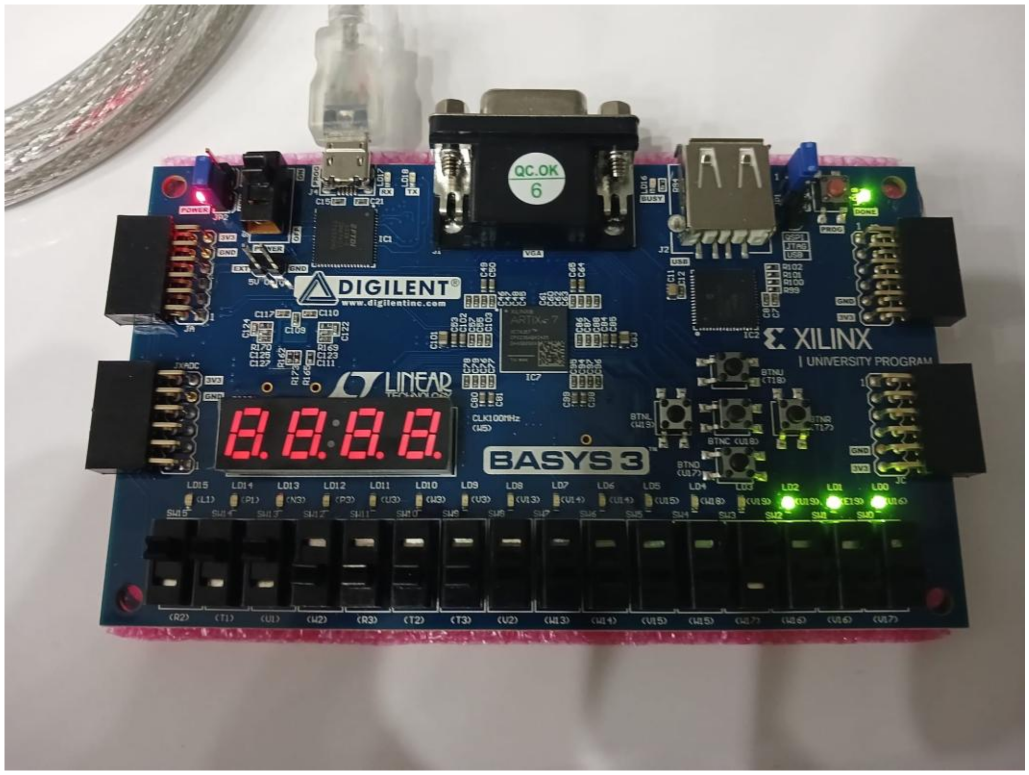
Figure 1.27: State of the board when inputs of the Decrement operator of Table 1 were entered

Conclusion:

The purpose of this lab was to implement an ALU with arithmetic, bitwise and shifting operations. A select system similar to a multiplexer was used to change between different functions. The selected functions were addition, subtraction, left shift, right shift, decrement, AND, OR and XOR. The operations were simulated in a testbench and implemented to a BASYS board afterwards. The simulation results and real-world results were found to be equivalent. A small number of basic module implementation in VHDL was learned. The mechanism of the implemented functions was also comprehended in a better manner than before for logical system designs.

References:

Horowitz, P., & Hill, W. (1989). The art of electronics (2nd ed., p. 990). Cambridge University Press.

Wikipedia contributors. (n.d.). Arithmetic logic unit. Wikipedia, The Free Encyclopedia. Retrieved March 5, 2025, from https://en.wikipedia.org/wiki/Arithmetic_logic_unit

Appendices:

alu_main.vhd:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;



entity alu_main is

  Port ( A_in : in std_logic_vector(3 downto 0);

       B_in : in std_logic_vector(3 downto 0);

       Sel_in : in std_logic_vector(2 downto 0);

       Res : out std_logic_vector(3 downto 0);

       C_out : out std_logic);

end alu_main;


architecture Structural of alu_main is


  component fourand

  Port ( A_in: in std_logic_vector(3 downto 0);
       B_in: in std_logic_vector(3 downto 0);
       F_out: out std_logic_vector(3 downto 0)
       );
end component;

component fouror
Port ( A_in: in std_logic_vector(3 downto 0);
       B_in: in std_logic_vector(3 downto 0);
       F_out: out std_logic_vector(3 downto 0)
       );
end component;

component fourxor
Port ( A_in: in std_logic_vector(3 downto 0);
       B_in: in std_logic_vector(3 downto 0);
       F_out: out std_logic_vector(3 downto 0)
       );
end component;

component leftshifter
Port ( A_in : in std_logic_vector(3 downto 0);
       F_out : out std_logic_vector(3 downto 0));
end component;

component rightshifter
Port (
    A_in : in  std_logic_vector(3 downto 0);
    F_out : out std_logic_vector(3 downto 0)
);
end component;
```

```
    component subtraction
    Port ( A_in : in std_logic_vector (3 downto 0);
           B_in : in std_logic_vector(3 downto 0);
           D_out : out std_logic_vector(3 downto 0);
           B_out : out STD_LOGIC);
    end component;

    component addition
    Port ( A_in : in std_logic_vector(3 downto 0);
           B_in : in std_logic_vector(3 downto 0);
           C_in : in STD_LOGIC;
           S_out : out std_logic_vector(3 downto 0);
           C_out : out STD_LOGIC);
    end component;

    component decrement
    Port ( A_in : in std_logic_vector(3 downto 0);
           F_out : out std_logic_vector(3 downto 0);
           C_out : out STD_LOGIC);
    end component;



    signal add_out, sub_out, and_out, or_out, xor_out : std_logic_vector(3 downto 0);
    signal shl_out, shr_out, dcr_out              : std_logic_vector(3 downto 0);

    signal add_c_out, sub_c_out, dcr_c_out : std_logic;

begin
```

```
AND_VAR: fourand port map (A_in, B_in, and_out);
OR_VAR: fouror port map (A_in, B_in, or_out);
XOR_VAR: fourxor port map (A_in, B_in, xor_out);
LS_VAR: leftshifter port map (A_in, shl_out);
RS_VAR: rightshifter port map (A_in, shr_out);
ADD_VAR: addition port map (A_in, B_in, '0',add_out, add_c_out);
SUB_VAR: subtraction port map (A_in, B_in, sub_out, sub_c_out);
DECREMENT_VAR: decrement port map(A_in, dcr_out, dcr_c_out);


process(Sel_in, add_out, sub_out, and_out, or_out, xor_out, shl_out, shr_out, dcr_out, add_c_out,
        sub_c_out, dcr_c_out)
begin
    case Sel_in is
        when "000" =>
            Res <= and_out;
            C_out   <= '0';
        when "001" =>
            Res <= or_out;
            C_out   <= '0';
        when "010" =>
            Res <= xor_out;
            C_out   <= '0';
        when "011" =>
            Res <= shl_out;
            C_out   <= '0';
        when "100" =>
            Res <= shr_out;
            C_out   <= '0';
        when "101" =>
```

```
                Res <= add_out;
                C_out   <= add_c_out;
            when "110" =>
                Res <= sub_out;
                C_out   <= sub_c_out;
            when "111" =>
                Res <= dcr_out;
                C_out   <= dcr_c_out;
            when others =>
                Res <= (others => '0');
                C_out   <= '0';
        end case;
    end process;

end Structural;
```

halfadd.vhd

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



entity halfadd is
    Port ( A_in : in STD_LOGIC;
           B_in : in STD_LOGIC;
           S_out : out STD_LOGIC;
           C_out : out STD_LOGIC);
end halfadd;

architecture Behavioral of halfadd is
```

```
begin
    -- sum of the inputs
    S_out <= A_in xor B_in;
    -- carry of the inputs
    C_out <= A_in and B_in;



end Behavioral;
```

fulladd.vhd

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



entity fulladd is
    Port ( A_in : in STD_LOGIC;
           B_in : in STD_LOGIC;
           C_in : in STD_LOGIC;
           S_out : out STD_LOGIC;
           C_out : out STD_LOGIC);
end fulladd;

architecture Structural of fulladd is

    signal sum_1, sum_2, carry_1, carry_2: std_logic;

    component halfadd is
    port
        (
           A_in : in STD_LOGIC;
           B_in : in STD_LOGIC;
        S_out : out STD_LOGIC;
        C_out : out STD_LOGIC
      );
  end component;



begin
  ha_1: halfadd
  port map (
    A_in   => A_in,
    B_in   => B_in,
    S_out => sum_1,
    C_out   => carry_1
  );

  ha_2: halfadd
  port map (
    A_in   => sum_1,
    B_in   => C_in,
    S_out => sum_2,
    C_out   => carry_2
  );

  S_out <= sum_2;

  C_out <= carry_1 or carry_2;

end Structural;
```

fourand.vhd

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



entity fourand is
   Port ( A_in: in std_logic_vector(3 downto 0);
          B_in: in std_logic_vector(3 downto 0);
          F_out: out std_logic_vector(3 downto 0)
          );
end fourand;

architecture Behavioral of fourand is

begin
   F_out <= A_in and B_in;

end Behavioral;
```

fouror.vhd

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



entity fouror is
   Port ( A_in: in std_logic_vector(3 downto 0);
          B_in: in std_logic_vector(3 downto 0);
          F_out: out std_logic_vector(3 downto 0)
          );
end fouror;
```

```
architecture Behavioral of fouror is

begin
    F_out <= A_in or B_in;

end Behavioral;
```

fourxor.vhd

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



entity fourxor is
    Port ( A_in: in std_logic_vector(3 downto 0);
           B_in: in std_logic_vector(3 downto 0);
           F_out: out std_logic_vector(3 downto 0)
           );
end fourxor;

architecture Behavioral of fourxor is

begin
    F_out <= A_in xor B_in;

end Behavioral;
```

leftshifter.vhd

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

```
entity leftshifter is
    Port ( A_in : in std_logic_vector(3 downto 0);
           F_out : out std_logic_vector(3 downto 0));
end leftshifter;

architecture Behavioral of leftshifter is

begin

    F_out <= A_in(2 downto 0) & '0';

end Behavioral;
```

rightshifter.vhd

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



entity rightshifter is
    Port ( A_in : in std_logic_vector(3 downto 0);
           F_out : out std_logic_vector(3 downto 0));
end rightshifter;

architecture Behavioral of rightshifter is

begin

F_out <= '0' & A_in(3 downto 1);
```

```
end Behavioral;
```

addition.vhd

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



entity addition is
   Port (
           A_in : in std_logic_vector(3 downto 0);
           B_in : in std_logic_vector(3 downto 0);
           C_in : in STD_LOGIC;
           S_out : out std_logic_vector(3 downto 0);
           C_out : out STD_LOGIC);
end addition;

architecture Behavioral of addition is
   signal carry: std_logic_vector(3 downto 1);
    component fulladd is
        Port (
           A_in : in STD_LOGIC;
           B_in : in STD_LOGIC;
           C_in : in STD_LOGIC;
           S_out : out STD_LOGIC;
           C_out : out STD_LOGIC
    );
```

subtraction.vhd

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

```
entity subtraction is
 Port ( A_in : in std_logic_vector (3 downto 0);
       B_in : in std_logic_vector(3 downto 0);
       D_out : out std_logic_vector(3 downto 0);
       B_out : out STD_LOGIC);
end subtraction;

architecture Behavioral of subtraction is
  signal B_comp : std_logic_vector (3 downto 0);
  signal carry : std_logic_vector(4 downto 0);
  component fulladd is
     Port ( A_in : in STD_LOGIC;
       B_in : in STD_LOGIC;
       C_in : in STD_LOGIC;
       S_out : out STD_LOGIC;
       C_out : out STD_LOGIC
  );
  end component;

begin
  B_comp(0) <= not B_in(0);
  B_comp(1) <= not B_in(1);
  B_comp(2) <= not B_in(2);
  B_comp(3) <= not B_in(3);

  carry(0) <= '1';

  fa_0: fulladd port map(A_in(0), B_comp(0), carry(0), D_out(0), carry(1));
```

```
    fa_1: fulladd port map(A_in(1), B_comp(1), carry(1), D_out(1), carry(2));
    fa_2: fulladd port map(A_in(2), B_comp(2), carry(2), D_out(2), carry(3));
    fa_3: fulladd port map(A_in(3), B_comp(3), carry(3), D_out(3), carry(4));

    B_out <= carry(4);
```

decrement.vhd

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


entity decrement is
    Port ( A_in : in std_logic_vector(3 downto 0);
           F_out : out std_logic_vector(3 downto 0);
           C_out : out STD_LOGIC);
end decrement;

architecture Structural of decrement is
component fulladd is
    Port ( A_in : in STD_LOGIC;
           B_in : in STD_LOGIC;
           C_in : in STD_LOGIC;
           S_out : out STD_LOGIC;
           C_out : out STD_LOGIC
         );
    end component;
    signal carry : std_logic_vector(4 downto 0);
begin
    carry(0) <= '0';
    fa_0: fulladd port map (A_in(0), '1', carry(0), F_out(0), carry(1));
    fa_1: fulladd port map (A_in(1), '1', carry(1), F_out(1), carry(2));
    fa_2: fulladd port map (A_in(2), '1', carry(2), F_out(2), carry(3));
    fa_3: fulladd port map (A_in(3), '1', carry(3), F_out(3), carry(4));
    C_out <= not carry(4);
```

testbench4.vhd

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

entity testbench4 is
end testbench4;

architecture Behavioral of testbench4 is
    signal A_in      : std_logic_vector(3 downto 0);
    signal B_in      : std_logic_vector(3 downto 0);
    signal Sel_in     : std_logic_vector(2 downto 0);
    signal Res : std_logic_vector(3 downto 0);
    signal C_out   : std_logic;
begin

    UUT: entity work.alu_main

        port map (
            A_in      => A_in,
            B_in      => B_in,
            Sel_in     => Sel_in,
            Res => Res,
            C_out   => C_out
        );

    stimulus: process
```

```
        variable sel_int : integer;
        variable a_int  : integer;
        variable b_int  : integer;
    begin
        for sel_int in 0 to 7 loop
            Sel_in <= std_logic_vector(to_unsigned(sel_int, 3));
            for a_int in 0 to 15 loop
                A_in <= std_logic_vector(to_unsigned(a_int, 4));
                for b_int in 0 to 15 loop
                    B_in <= std_logic_vector(to_unsigned(b_int, 4));
                    wait for 10 ns;
                end loop;
            end loop;
        end loop;
        wait;
    end process stimulus;
end Behavioral;
```

alu_constraint.xdc

```
set_property PACKAGE_PIN V17 [get_ports {A_in[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A_in[0]}]

set_property PACKAGE_PIN V16 [get_ports {A_in[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A_in[1]}]

set_property PACKAGE_PIN W16 [get_ports {A_in[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A_in[2]}]

set_property PACKAGE_PIN W17 [get_ports {A_in[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A_in[3]}]
set_property PACKAGE_PIN V15 [get_ports {B_in[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B_in[0]}]

set_property PACKAGE_PIN W14 [get_ports {B_in[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B_in[1]}]

set_property PACKAGE_PIN W13 [get_ports {B_in[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B_in[2]}]

set_property PACKAGE_PIN V2 [get_ports {B_in[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B_in[3]}]


set_property PACKAGE_PIN U1 [get_ports {Sel_in[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Sel_in[0]}]

set_property PACKAGE_PIN T1 [get_ports {Sel_in[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Sel_in[1]}]

set_property PACKAGE_PIN R2 [get_ports {Sel_in[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Sel_in[2]}]


set_property PACKAGE_PIN U16 [get_ports {Res[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Res[0]}]

set_property PACKAGE_PIN E19 [get_ports {Res[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Res[1]}]
set_property PACKAGE_PIN U19 [get_ports {Res[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Res[2]}]

set_property PACKAGE_PIN V19 [get_ports {Res[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Res[3]}]


set_property PACKAGE_PIN W18 [get_ports {C_out}]
set_property IOSTANDARD LVCMOS33 [get_ports {C_out}]
```