ECE 511 Digital ASIC Design

LAB

Lab 1: Implementing Binary Adders Using VHDL

Name: Xinyue Chen

Student ID: 1664206

Lab Date: Octo, 07, 2020

Abstract:

The objective of Lab 1 is to use VHDL to design a half-adder based 2-bit circuit, a 2-bit combinational adder and a 2-bit comparator. We first use the truth table to draw a Karnaugh-map, and then use the Boolean function obtained from the K-map to implement the function.

Learning objectives:

1. Learn to design a half adder-based 2-bit circuit using the truth table, Karnaugh map and the gate logic that will be implemented in VHDL.
2. Learn how to synthesize, implement, and program a half adder-based 2-bit design for the Zybo Z7 board.
3. Learn how to design a 2-bit adder using three 16 x 1 multiplexers, two multiplexers for the sum and one for the carry output, that will be used as components in 2-bit adder.
4. Simulate, synthesize, implement, and program the 2-bit adder design on the Zybo Z7 board.
5. Learn how to build a comparator for the inputs of the 2-bit adder using conditional signal assignment statements.

Pre-Lab:

1. Review the lecture notes to understand the differences between a half and a full adder.

Difference: 1-bit half adder only has two inputs a and b, and two outputs sum and carry, while 1-bit full adder has three inputs a, b and carry(in), and two outputs sum and carry(out). The truth table of 1-bit half adder and full adder is shown in **Table 1-1** to **Table 1-2**. And we can use the truth table to obtain the Boolean function, then use it to design the circuit.

a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 1-1. Truth table of half adder.

a	b	carry(in)	sum	carry(out)
0	0	0	0	0
0	0	1	1	0

0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1-2. Truth table of full adder.

2. Review how to use a Mux to implement the Boolean function from the lecture notes.

Example: A Mux can be used to implement different Boolean function, we can simply use a 4 x 1 multiplexer to implement the function **in Table 1-2**, the schematic is shown in **Figure 1**.

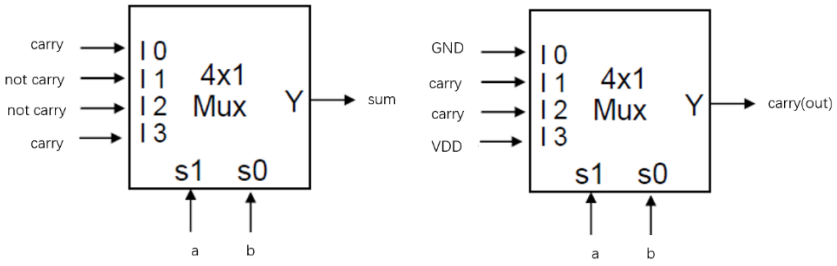

Figure 1. Schematic of 1-bit full adder using 2 multiplexers.

3. Consult the lecture slides and the VHDL textbooks as well as the given reference links, to get familiar with RTL and behavioral level design in VHDL.

The result of this work is shown in the VHDL design.

PART 1: Half adder-based 2-bit design

According to the function of 1-bit half adder, we implement a half adder-based 2-bit design. The block diagram is shown in **Figure 2**.

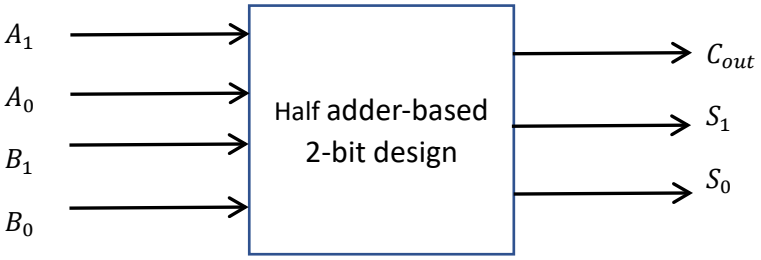

Figure 2. Block diagram of Half adder-based 2-bit design.

The half adder-based 2-bit circuit implement a function similar to the function of a 2-bit adder. The only difference is that there are only 4 inputs, we can view it as a 2-bit adder with $carry_{in} = 0$. The truth table is shown in **Table 2**.

Input				Output		
A_1	A_0	B_1	B_0	S_0	S_1	C_{out}
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	1	1	0
1	0	1	0	0	0	1
1	0	1	1	1	0	1
1	1	0	0	1	1	0
1	1	0	1	0	0	1
1	1	1	0	1	0	1
1	1	1	1	0	1	1

Table 2. Truth table of half adder-based 2-bit design.

The Karnaugh map for this design is shown in **Table 3-1** to **Table 3-3**.

$A_1A_0\backslash B_1B_0$	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	1	0	0	1
10	0	1	1	0

Table 3-1. K-map for S_0.

$A_1A_0\backslash B_1B_0$	00	01	11	10
00	0	0	1	1
01	0	1	0	1
11	1	0	1	0
10	1	1	0	0

Table 3-2. K-map for S_1.

$A_1A_0\backslash B_1B_0$	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	1	1	1
10	0	0	1	1

Table 3-3. K-map for C_{out}.

We use the Karnaugh map to obtain the Boolean function in (1)-(3), and then use the Boolean function to design the circuit.

$$\begin{aligned} S_0 &= A_1{}'A_0{}'B_0 + A_1A_0{}'B_0 + A_0B_1{}'B_0{}' + A_0B_1B_0{}' \\ &= A_0{}'B_0 + A_0B_0{}' \\ &= A_0 \; xor \; B_0 \end{aligned} \tag{1}$$

$$\begin{aligned} S_1 &= A_1{}'A_0{}'B_1 + A_1{}'B_1B_0{}' + A_1B_1{}'B_0' + A_1A_0{}'B_1' + A_1{}'A_0B_1{}'B_0 + A_1A_0B_1B_0 \\ &= (A_1 \; xor \; B_1) \cdot A_0' + \; (A_1 \; xor \; B_1) \; \cdot \; B_0' + (A_1 \; xnor \; B_1) \; \cdot A_0B_0 \end{aligned} \tag{2}$$

$$\begin{aligned} C_{out} &= A_1B_1 + A_1A_0B_1{}'B_0 + A_1{}'A_0B_1B_0 \\ &= A_1B_1 + (A_1 \; xor \; B_1) \; \cdot \; A_0B_0 \end{aligned} \tag{3}$$

The RTL schematic of this circuit is shown in **Figure 3**. The schematic matches the Boolean function we used for each output line.

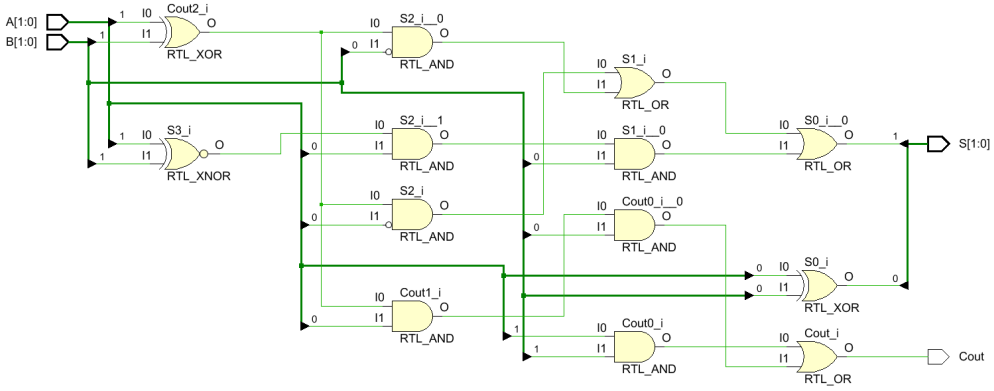

Figure 3. RTL design of half adder-based 2-bit circuit.

The VHDL code, testbench for simulation and constraint file in demonstrated in the comment section. The waveform of the simulation result is shown in **Figure 4**. The testbench is written with all the possible combinations of the input values, and the simulation result of the output values match the truth table of the half adder-based 2-bit circuit.

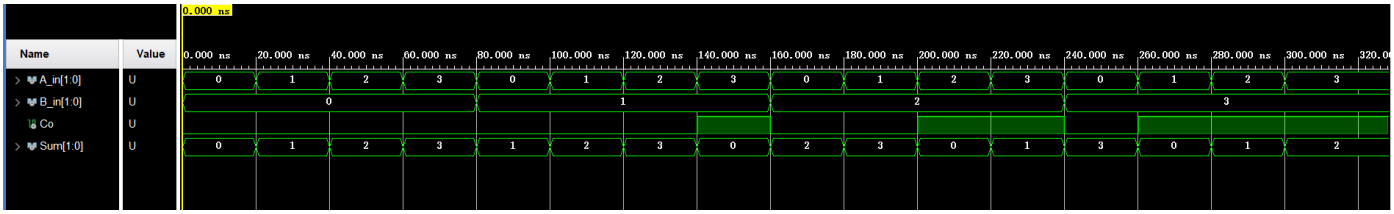

Figure 4. Waveform of half adder-based 2-bit circuit.

The synthesis result is shown in **Figure 5**.

Resource	Utilization	Available	Utilization %
LUT	2	17600	0.01
IO	7	100	7.00

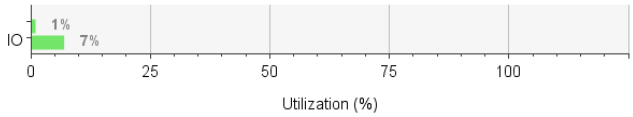

Figure 5. Synthesis result of half adder-based 2-bit adder.

After synthesis, implementation, writing bit-stream and programming the FPGA, we use the Zybo board to verify the result. We turn the corresponding switches on. When A=11, B=11, the result is sum=10, C_out=1; when A=10, B=01, the result is sum=00, C_out=1. We successfully implement the function of the half adder-based 2-bit adder.

PART 2: 2-bit adder design

According to the function of 1-bit full adder, we implement a 2-bit adder design. The block diagram is shown in **Figure 6**.

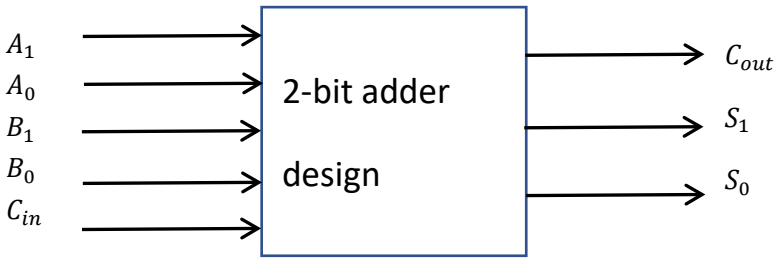

Figure 6. Block diagram of 2-bit adder design.

The truth table of this 2-bit adder is shown in **Table 4**.

Input					Output		
A_1	A_0	B_1	B_0	C_{in}	S_0	S_1	C_{out}
0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0
0	0	0	1	0	1	0	0
0	0	0	1	1	0	1	0
0	0	1	0	0	0	1	0
0	0	1	0	1	1	1	0
0	0	1	1	0	1	1	0
0	0	1	1	1	0	0	1
0	1	0	0	0	1	0	0
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0

0	1	0	1	1	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	1	0	0	1
0	1	1	1	0	0	0	1
0	1	1	1	1	1	0	1
1	0	0	0	0	0	1	0
1	0	0	0	1	1	1	0
1	0	0	1	0	1	1	0
1	0	0	1	1	0	0	1
1	0	1	0	0	0	0	1
1	0	1	0	1	1	0	1
1	0	1	1	0	1	0	1
1	0	1	1	1	0	1	1
1	1	0	0	0	1	1	0
1	1	0	0	1	0	0	1
1	1	0	1	0	0	0	1
1	1	0	1	1	1	0	1
1	1	1	0	0	1	0	1
1	1	1	0	1	0	1	1
1	1	1	1	0	0	1	1
1	1	1	1	1	1	1	1

Table 4. 2-bit adder truth table.

Since the objective is to use three 16 x 1 multiplexers, two multiplexers for the sum and one for the carry output, we simplify the truth table (shown in **Table 5**), so we can just assign values to the input lines (C_{in}, not (C_{in}) , '0' and '1') from I_0 to I_{15}.

Input				Output		
A_1	A_0	B_1	B_0	S_0	S_1	C_{out}
0	0	0	0	C_{in}	0	0
0	0	0	1	not (C_{in})	C_{in}	0
0	0	1	0	C_{in}	1	0
0	0	1	1	not (C_{in})	not (C_{in})	C_{in}
0	1	0	0	not (C_{in})	C_{in}	0
0	1	0	1	C_{in}	1	0
0	1	1	0	not (C_{in})	not (C_{in})	C_{in}
0	1	1	1	C_{in}	0	1
1	0	0	0	C_{in}	1	0
1	0	0	1	not (C_{in})	not (C_{in})	C_{in}
1	0	1	0	C_{in}	0	1
1	0	1	1	not (C_{in})	C_{in}	1
1	1	0	0	not (C_{in})	not (C_{in})	C_{in}

1	1	0	1	C_{in}	0	1
1	1	1	0	not (C_{in})	C_{in}	1
1	1	1	1	C_{in}	1	1

Table 5. 2-bit adder truth table simplified version.

In VHDL, we first design three 16 x 1 multiplexers for the sum (2-bit) output and carry output. Then we use the three 16 x 1 multiplexers as components to design the 2-bit adder. The RTL schematic of the 2-bit adder is shown in **Figure 7**. The detailed RTL schematic of three 16 x 1 multiplexers is shown in **Figure 8-1** to **Figure 8-3**. We can see from the RTL schematic that the 16 input lines of each multiplexer is assigned with different values (C_{in}, not (C_{in}), '0' and '1'), and the corresponding input value (selected according to the input value of the 4-bit selection lines) is transmitted to the output of the multiplexer. The schematic matches the truth table (simplified version for multiplexers).

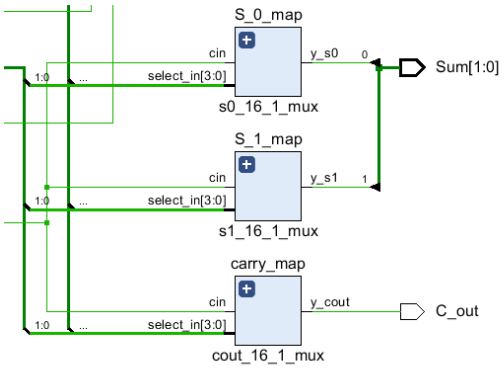

Figure 7. RTL schematic of 2-bit adder.

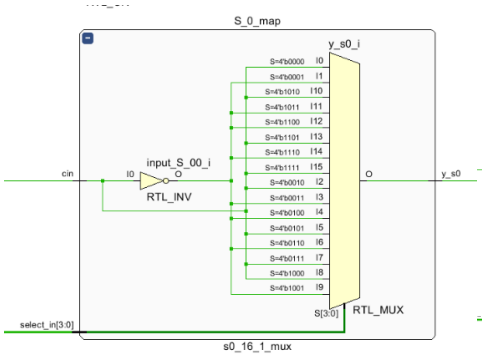

Figure 8-1. RTL schematic of Mux for S_0.

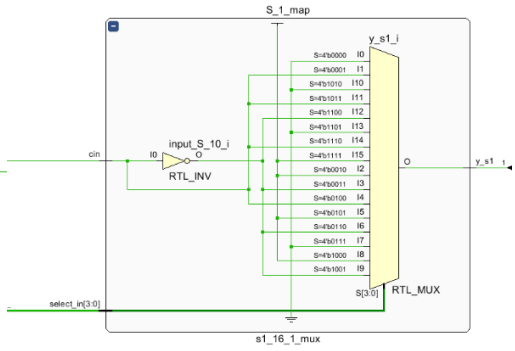

Figure 8-2. RTL schematic of Mux for S_1.

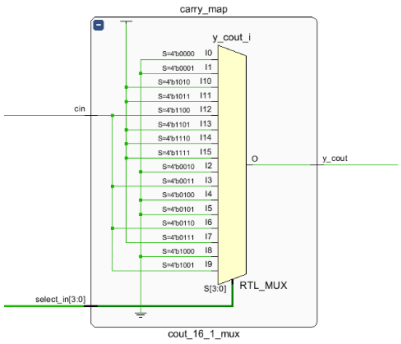

Figure 8-3. RTL schematic of Mux for C_{out}.

The VHDL code, testbench for simulation and constraint file in demonstrated in the comment section (together with that of PART 2, however, marked with bule color). The waveform of the simulation result is shown in **Figure 9**. The testbench is written with all the 32 possible combinations of the input values, and the simulation result of the output values match the desired function of the 2-bit adder.

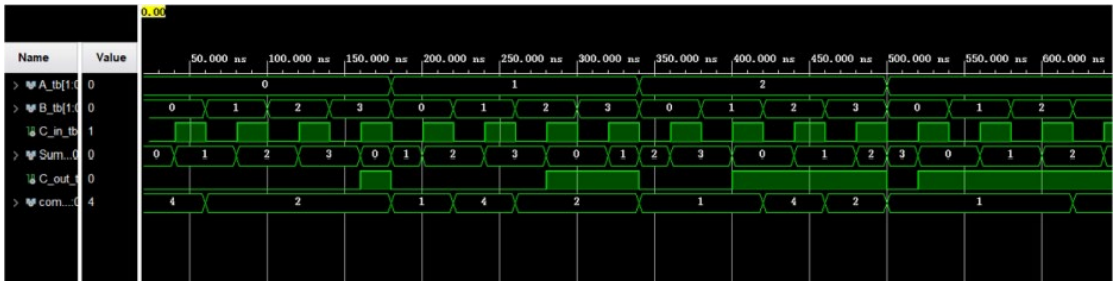

Figure 9. Waveform of 2-bit adder.

The synthesis result is shown in **Figure 10**.

Resource	Utilization	Available	Utilization %
LUT	4	17600	0.02
IO	11	100	11.00

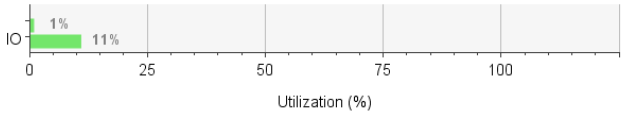

Figure 10. Synthesis result of 2-bit adder.

After synthesis, implementation, writing bit-stream and programming the FPGA, we use the Zybo board to verify the result. We turn the corresponding switches on. When A=11, B=11, we press the button of C_in, the result is sum=11, C_out=1; when A=10, B=01, we press the button of C_in, the result is sum=01, C_out=1. We successfully implement the function of the 2-bit adder.

PART 3: Comparator for the input vectors of the 2-bit adder

We implement the comparator design based on the 2-bit adder. The block diagram is shown in **Figure 11**.

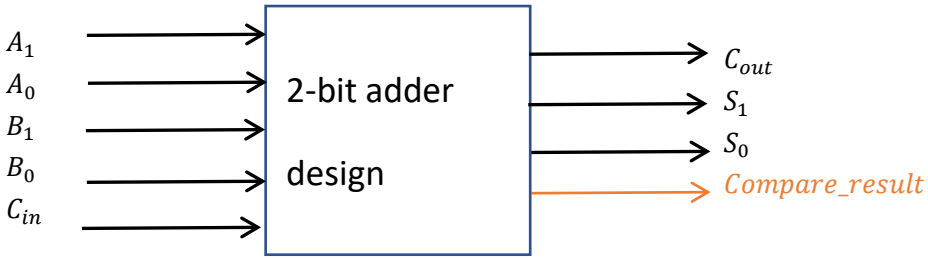

Figure 11. Block diagram of the comparator.

This comparator compares two inputs of the 2-bit adder A and B, each bit of the 3-bit output *compare result* is used to indicate different results (When A>B, compare(0)=1, When A<B, compare(1)=1, When A=B, compare(2)=1). The truth table of this comparator is shown in **Table 6**.

Input				Output		
A_1	A_0	B_1	B_0	$Comp_0$(A>B)	$Comp_1$(A<B)	$Comp_2$(A=B)
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	0	1

Table 6. Comparator truth table.

The K-map of this comparator is shown in **Table 7**. We use this to obtain the Boolean function used in the implementation.

$A_1A_0\backslash B_1B_0$	00	01	11	10
00	=	<	<	<
01	>	=	<	<
11	>	>	=	>

10	>	>	<	=

Table 7. K-map for $Comp_0$, $Comp_1$ and $Comp_2$.

$$Comp_0 = A_1B_1' + A_1'A_0B_1'B_0' + A_1A_0B_1B_0' \quad \textbf{(4)}$$

$$Comp_1 = A_1'B_1 + A_1'A_0'B_1'B_0 + A_1A_0'B_1B_0 \quad \textbf{(5)}$$

$$Comp_2 = A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0B_1B_0 + A_1A_0'B_1B_0' \quad \textbf{(6)}$$
$$= (A_1\ xnor\ B_1)\ and\ (A_0\ xnor\ B_0)$$

The detailed RTL schematic of the comparator is shown in **Figure 12**. The schematic matches the Boolean function we used.

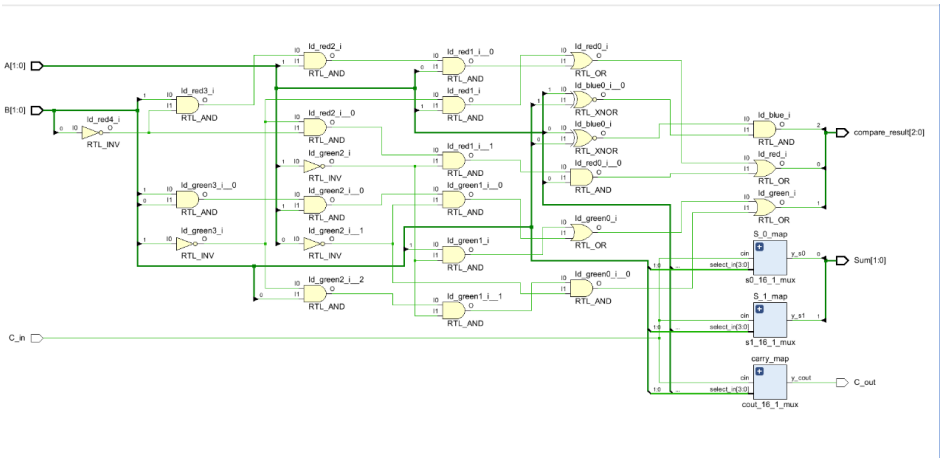

Figure 12. RTL schematic of the comparator.

The VHDL code, testbench for simulation and constraint file in demonstrated in the comment section. The waveform of the simulation result is shown in **Figure 13**. This design shares the testbench with PART 2, and the simulation result of the output compare_result matches the truth table of the comparator.

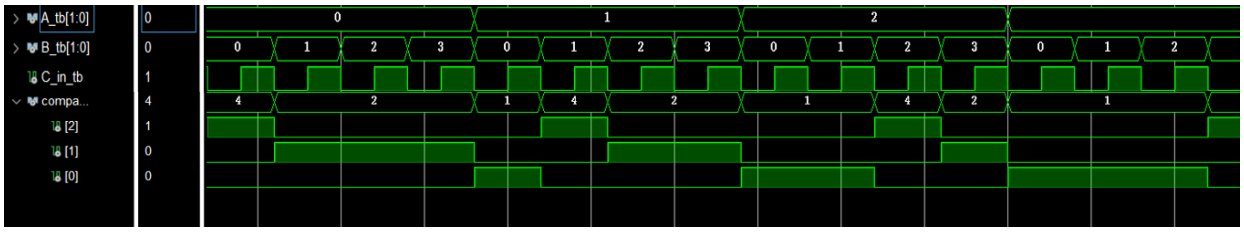

Figure 13. Waveform of the comparator.

After synthesis, implementation, writing bit-stream and programming the FPGA, we use the Zybo board to verify the result.

When A>B, RGB LDE6 turns red; when A<B, RGB LDE6 turns green; when A=B, RGB LDE6 turns green. We successfully implement the function of the comparator.

Comment Section:

PART 1:

VHDL code of the half adder-based 2- bit adder:

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/10/11 11:19:27 AM
-- Design Name: half adder-based 2-bit adder
-- Module Name: ha_2bit - Behavioral
-- Project Name: half adder-based 2-bit adder
-- Target Devices: Zybo Z-7
-- Tool Versions:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ha_2bit is
    Port ( A : in STD_LOGIC_VECTOR (1 downto 0);
           B : in STD_LOGIC_VECTOR (1 downto 0);
           Cout : out STD_LOGIC;
           S : out STD_LOGIC_VECTOR (1 downto 0));
end ha_2bit;
architecture Behavioral of ha_2bit is
begin
    --write the karnaugh map equation for S(1)
     S(1) <= ((A(1) xor B(1)) and (not A(0))) or ((A(1) xor B(1)) and (not B(0))) or ((A(1) xnor B(1)) and A(0)
     and B(0)) ;
    --write the karnaugh map equation for S(0)
    S(0) <= A(0) xor B(0);
    --write the karnaugh map equation for Cout
    Cout <= (A(1) and B(1)) or ((A(1) xor B(1)) and A(0) and B(0));
    end Behavioral;
```

VHDL testbench of the half adder-based 2- bit adder:

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 08/09/2020 07:06:23 PM
-- Design Name: testbench of half adder-based 2-bit adder
```

```
-- Module Name: ha_2bit_tb - Behavioral
-- Project Name: testbench of half adder-based 2-bit adder
-- Target Devices: Zybo Z-7
-- Tool Versions:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ha_2bit_tb is
end ha_2bit_tb;
architecture Behavioral of ha_2bit_tb is
component ha_2bit is
   Port ( A : in std_logic_vector(1 downto 0);
          B : in std_logic_vector(1 downto 0);
          Cout : out std_logic;
          S : out std_logic_vector(1 downto 0));
end component;
signal A_in, B_in : std_logic_vector(1 downto 0);
signal Co : std_logic;
signal Sum : std_logic_vector(1 downto 0);
begin
      ADDER_HA : component ha_2bit port map(A => A_in,
                                            B => B_in,
                                           Cout => Co,
                                            S => Sum);
      process
           constant time_period : time := 20ns;
           begin
           A_in <= "00"; B_in <= "00";
           assert ((Sum = "00") and (Co='0'))
           report "Error for input A_in=00 and B_in=00" severity error;
           wait for time_period;

           A_in <= "01"; B_in <= "00";
           assert ((Sum = "01") and (Co='0'))
           report "Error for input A_in=01 and B_in=00" severity error;
           wait for time_period;

           A_in <= "10"; B_in <= "00";
           assert ((Sum = "10") and (Co='0'))
           report "Error for input A_in=10 and B_in=00" severity error;
```

```
wait for time_period;

A_in <= "11"; B_in <= "00";
assert ((Sum = "11") and (Co='0'))
report "Error for input A_in=11 and B_in=00" severity error;
wait for time_period;

A_in <= "00"; B_in <= "01";
assert ((Sum = "01") and (Co='0'))
report "Error for input A_in=00 and B_in=01" severity error;
wait for time_period;

A_in <= "01"; B_in <= "01";
assert ((Sum = "00") and (Co='0'))
report "Error for input A_in=01 and B_in=01" severity error;
wait for time_period;

A_in <= "10";    B_in <= "01";
assert ((Sum = "11") and (Co='0'))
report "Error for input A_in=10 and B_in=01" severity error;
wait for time_period;

A_in <= "11"; B_in <= "01";
assert ((Sum = "10") and (Co='0'))
report "Error for input A_in=11 and B_in=01" severity error;
wait for time_period;
A_in <= "00"; B_in <= "10";
assert ((Sum = "10") and (Co='0'))
report "Error for input A_in=00 and B_in=10" severity error;
wait for time_period;

A_in <= "01"; B_in <= "10";
assert ((Sum = "11") and (Co='0'))
report "Error for input A_in=01 and B_in=10" severity error;
wait for time_period;

A_in <= "10"; B_in <= "10";
assert ((Sum = "00") and (Co='1'))
report "Error for input A_in=10 and B_in=10" severity error;
wait for time_period;

A_in <= "11"; B_in <= "10";
assert ((Sum = "01") and (Co='1'))
report "Error for input A_in=11 and B_in=10" severity error;
```

```
        wait for time_period;

        A_in <= "00"; B_in <= "11";
        assert ((Sum = "11") and (Co='0'))
        report "Error for input A_in=00 and B_in=11" severity error;
        wait for time_period;

        A_in <= "01"; B_in <= "11";
        assert ((Sum = "10") and (Co='0'))
        report "Error for input A_in=01 and B_in=11" severity error;
        wait for time_period;

        A_in <= "10"; B_in <= "11";
        assert ((Sum = "01") and (Co='1'))
        report "Error for input A_in=10 and B_in=11" severity error;
        wait for time_period;

        A_in <= "11"; B_in <= "11";
        assert ((Sum = "00") and (Co='1'))
        report "Error for input A_in=11 and B_in=11" severity error;
        wait for time_period;
        wait;
    end process;
    end Behavioral;
```

Constraint file of the half adder-based 2-bit adder:

In constraint file, each entity port is mapped to the Zybo Z7 board as shown below:

```
##Switches
set_property -dict { PACKAGE_PIN G15     IOSTANDARD LVCMOS33 } [get_ports { A[0] }];
#IO_L19N_T3_VREF_35 Sch=sw[0]
set_property -dict { PACKAGE_PIN P15     IOSTANDARD LVCMOS33 } [get_ports { A[1] }];
#IO_L24P_T3_34 Sch=sw[1]
set_property -dict { PACKAGE_PIN W13     IOSTANDARD LVCMOS33 } [get_ports { B[0] }];
#IO_L4N_T0_34 Sch=sw[2]
set_property -dict { PACKAGE_PIN T16     IOSTANDARD LVCMOS33 } [get_ports { B[1] }];
#IO_L9P_T1_DQS_34 Sch=sw[3]
##LEDs
set_property -dict { PACKAGE_PIN M14     IOSTANDARD LVCMOS33 } [get_ports { S[0] }];
#IO_L23P_T3_35 Sch=led[0]
set_property -dict { PACKAGE_PIN M15     IOSTANDARD LVCMOS33 } [get_ports { S[1] }];
#IO_L23N_T3_35 Sch=led[1]
set_property -dict { PACKAGE_PIN G14     IOSTANDARD LVCMOS33 } [get_ports { Cout }];
#IO_0_35 Sch=led[2]
```

PART 2 & PART 3:

The VHDL code, testbench and constraint file of part 3 is demonstrated together with those of part 2, marked with blue color.

VHDL code of the first multiplexer for output sum (0):

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 08/10/2020 11:28:43 AM
-- Design Name: 2-bit adder mux_1
-- Module Name: s0_16_1_mux - Behavioral
-- Project Name: 2-bit adder
-- Target Devices: Zybo Z-7
-- Tool Versions:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity s0_16_1_mux is
    Port ( cin : in STD_LOGIC;
           select_in : in STD_LOGIC_VECTOR (3 downto 0);
           y_s0: out STD_LOGIC);
end s0_16_1_mux;
architecture Behavioral of s0_16_1_mux is
signal input_S_0: std_logic_vector(15 downto 0) := (others => '0');
begin
    process(select_in,cin) is
    begin
    input_S_0 <= cin & (not (cin)) & cin & (not (cin)) & (not (cin)) & cin & (not (cin)) & cin & cin & (not
    (cin)) & cin & (not (cin)) & (not (cin)) & cin & (not (cin)) & cin;
    -- write the input line functions generated for 16:1 to derive the S(0) output.
    -- The input lines can be '0' / '1' / Cin / (not) Cin.
    -- It is possible to use if...else or case statements here.
        case select_in is
    -- write the VHDL code for all the 16 cases.
        when "0000" =>  y_s0 <= input_S_0(0);
        when "0001" =>  y_s0 <= input_S_0(1);
        when "0010" =>  y_s0 <= input_S_0(2);
        when "0011" =>  y_s0 <= input_S_0(3);
        when "0100" =>  y_s0 <= input_S_0(4);
        when "0101" =>  y_s0 <= input_S_0(5);
```

```
            when "0110" =>    y_s0 <= input_S_0(6);
            when "0111" =>    y_s0 <= input_S_0(7);
            when "1000" =>    y_s0 <= input_S_0(8);
            when "1001" =>    y_s0 <= input_S_0(9);
            when "1010" =>    y_s0 <= input_S_0(10);
            when "1011" =>    y_s0 <= input_S_0(11);
            when "1100" =>    y_s0 <= input_S_0(12);
            when "1101" =>    y_s0 <= input_S_0(13);
            when "1110" =>    y_s0 <= input_S_0(14);
            when "1111" =>    y_s0 <= input_S_0(15);
            when others =>    y_s0 <= '-';
            end case;
      end process;
end Behavioral;
```

VHDL code of the second multiplexer for output sum (1):

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/10/11 11:55:10
-- Design Name: 2-bit adder mux_2
-- Module Name: s1_16_1_mux - Behavioral
-- Project Name: 2-bit adder
-- Target Devices: Zybo Z-7
-- Tool Versions:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity s1_16_1_mux is
      Port ( cin : in STD_LOGIC;
            select_in : in STD_LOGIC_VECTOR (3 downto 0);
            y_s1 : out STD_LOGIC);
end s1_16_1_mux;
architecture Behavioral of s1_16_1_mux is
signal input_S_1: std_logic_vector(15 downto 0) := (others => '0');
begin
      process(select_in,cin) is
      begin
      input_S_1 <= '1' & cin & '0' & (not (cin)) & cin & '0' & (not (cin)) & '1' & '0' & (not (cin)) & '1' & cin &
                  (not (cin)) & '1' & cin & '0';
```

```
    -- write the input line functions generated for 16:1 to derive the S(0) output.
    -- The input lines can be '0' / '1' / Cin / (not) Cin.
    -- It is possible to use if...else or case statements here.
        case select_in is
    -- Write the VHDL codes for all the input cases.
        when "0000" =>   y_s1 <= input_S_1(0);
        when "0001" =>   y_s1 <= input_S_1(1);
        when "0010" =>   y_s1 <= input_S_1(2);
        when "0011" =>   y_s1 <= input_S_1(3);
        when "0100" =>   y_s1 <= input_S_1(4);
        when "0101" =>   y_s1 <= input_S_1(5);
        when "0110" =>   y_s1 <= input_S_1(6);
        when "0111" =>   y_s1 <= input_S_1(7);
        when "1000" =>   y_s1 <= input_S_1(8);
        when "1001" =>   y_s1 <= input_S_1(9);
        when "1010" =>   y_s1 <= input_S_1(10);
        when "1011" =>   y_s1 <= input_S_1(11);
        when "1100" =>   y_s1 <= input_S_1(12);
        when "1101" =>   y_s1 <= input_S_1(13);
        when "1110" =>   y_s1 <= input_S_1(14);
        when "1111" =>   y_s1 <= input_S_1(15);
        when others =>   y_s1 <= '-';
        end case;
    end process;
end Behavioral;
```

VHDL code of the third multiplexer for output Cout:

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/10/11 13:35:06
-- Design Name: 2-bit adder mux_3
-- Module Name: cout_16_1_mux - Behavioral
-- Project Name: 2-bit adder
-- Target Devices: Zybo Z-7
-- Tool Versions:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity cout_16_1_mux is
```

```
    Port ( cin : in STD_LOGIC;
           select_in : in STD_LOGIC_VECTOR (3 downto 0);
           y_cout : out STD_LOGIC);
end cout_16_1_mux;
architecture Behavioral of cout_16_1_mux is
signal input_count: std_logic_vector(15 downto 0) := (others => '0');
begin
    process(select_in,cin) is
    begin
     input_count <= '1' & '1' & '1' & cin & '1' & '1' & cin & '0' & '1' & cin & '0' & '0' & cin & '0' & '0' & '0';
    -- write the input line functions generated for 16:1 to derive the carry output.
    -- The input lines can be '0' / '1' / Cin / (not) Cin.
    -- It is possible to use if...else or case statements here.
        case select_in is
    -- write the vhdl code for all the input cases.
        when "0000" =>   y_cout <= input_count(0);
        when "0001" =>   y_cout <= input_count(1);
        when "0010" =>   y_cout <= input_count(2);
        when "0011" =>   y_cout <= input_count(3);
        when "0100" =>   y_cout <= input_count(4);
        when "0101" =>   y_cout <= input_count(5);
        when "0110" =>   y_cout <= input_count(6);
        when "0111" =>   y_cout <= input_count(7);
        when "1000" =>   y_cout <= input_count(8);
        when "1001" =>   y_cout <= input_count(9);
        when "1010" =>   y_cout <= input_count(10);
        when "1011" =>   y_cout <= input_count(11);
        when "1100" =>   y_cout <= input_count(12);
        when "1101" =>   y_cout <= input_count(13);
        when "1110" =>   y_cout <= input_count(14);
        when "1111" =>   y_cout <= input_count(15);
        when others =>   y_cout <= '-';
        end case;
    end process;
end Behavioral;
```

VHDL code of the 2-bit full adder & the comparator:

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/10/11 13:51:49
-- Design Name: 2-bit adder and comparator
-- Module Name: fa_2bit - Behavioral
```

```
-- Project Name: 2-bit adder and comparator
-- Target Devices: Zybo Z-7
-- Tool Versions:
-- Description:
-- Dependencies:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity fa_2bit is
      Port ( A : in STD_LOGIC_VECTOR (1 downto 0);
             B : in STD_LOGIC_VECTOR (1 downto 0);
             C_in : in STD_LOGIC;
             Sum : out STD_LOGIC_VECTOR (1 downto 0);
             C_out : out STD_LOGIC;
             compare_result : out std_logic_vector(2 downto 0));
-- compare the input vector A and B : mapped to RGB led in xdc file
end fa_2bit;
architecture Behavioral of fa_2bit is
signal ld_red: std_logic;
signal ld_green: std_logic;
signal ld_blue: std_logic;
component cout_16_1_mux is
   Port ( cin : in std_logic;
          select_in : in std_logic_vector(3 downto 0);
          y_cout : out std_logic );
end component cout_16_1_mux;
component s1_16_1_mux is
   Port ( cin : in std_logic;
          select_in : in std_logic_vector(3 downto 0);
          y_s1 : out std_logic );
end component s1_16_1_mux;
component s0_16_1_mux is
   Port ( cin : in std_logic;
          select_in : in std_logic_vector(3 downto 0);
          y_s0 : out std_logic );
end component s0_16_1_mux;

begin
-- PART 3....
-- The "LD6" - RGB led on board is used as an indication if A>B or A<B or A=B.
```

```
  ld_red <= ((not B(1)) and A(1)) or (B(1) and (not B(0)) and A(1) and A(0)) or ((not B(1)) and (not B(0)) and
            (not A(1)) and A(0));
  compare_result(0) <= ld_red;

  ld_green <= (B(1) and (not A(1))) or (B(1) and B(0) and A(1) and (not A(0))) or ((not B(1)) and B(0) and (not
              A(1)) and (not A(0)));
  compare_result(1) <= ld_green;

  ld_blue <= (A(0) xnor B(0)) and (A(1) xnor B(1));
  compare_result(2) <= ld_blue;
-- to turn the LED red when A>B, green when A<B and blue when A=B.
carry_map : component cout_16_1_mux port map( cin => C_in,
                                              select_in(3 downto 2) => A,
                                              select_in(1 downto 0) => B,
                                            y_cout => C_out);
-- port map the component for generating the C_out
S_1_map : component s1_16_1_mux port map( cin => C_in,
                                            select_in(3 downto 2) => A,
                                            select_in(1 downto 0) => B,
                                            y_s1 => Sum(1));
-- port map the component for generating the S(1)
     S_0_map : component s0_16_1_mux port map( cin => C_in,
                                            select_in(3 downto 2) => A,
                                            select_in(1 downto 0) => B,
                                            y_s0 => Sum(0));
-- port map the component for generating the S(0)
end Behavioral;
```

VHDL testbench of the 2-bit full adder & the comparator:

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/10/18 10:56:37
-- Design Name: testbench for 2-bit adder and comparator
-- Module Name: fa_2bit_tb - Behavioral
-- Project Name: testbench for 2-bit adder and comparator
-- Target Devices: Zybo Z-7
-- Tool Versions:
-- Description:
-- Dependencies:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
```

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity fa_2bit_tb is
end fa_2bit_tb;

architecture Behavioral of fa_2bit_tb is
component fa_2bit is
  Port ( A : in STD_LOGIC_VECTOR (1 downto 0);
                B : in STD_LOGIC_VECTOR (1 downto 0);
                C_in : in STD_LOGIC;
                Sum : out STD_LOGIC_VECTOR (1 downto 0);
                C_out : out STD_LOGIC;
     compare_result : out std_logic_vector(2 downto 0));
-- compare the input vector A and B : mapped to RGB led in xdc file
end component fa_2bit;
signal A_tb : std_logic_vector(1 downto 0) := "00";
signal B_tb : std_logic_vector(1 downto 0) := "00";
signal C_in_tb : std_logic := '0';
signal Sum_tb : std_logic_vector(1 downto 0);
signal C_out_tb : std_logic;
signal compare_result_tb : std_logic_vector(2 downto 0);
begin
      TEST_FA : component fa_2bit port map(A => A_tb,
                                           B => B_tb,
                                           C_in => C_in_tb,
                                           C_out => C_out_tb,
                                           Sum => Sum_tb,
                                           compare_result => compare_result_tb);
      stimu: process
            constant time_period : time := 20ns;
            begin
            wait for time_period;
               A_tb <= "00"; B_tb <= "00"; C_in_tb <= '0';
            wait for time_period;
               A_tb <= "00"; B_tb <= "00"; C_in_tb <= '1';
            wait for time_period;
               A_tb <= "00"; B_tb <= "01"; C_in_tb <= '0';
            wait for time_period;
               A_tb <= "00"; B_tb <= "01"; C_in_tb <= '1';
            wait for time_period;
               A_tb <= "00"; B_tb <= "10"; C_in_tb <= '0';
            wait for time_period;
               A_tb <= "00"; B_tb <= "10"; C_in_tb <= '1';
```

```
 wait for time_period;
    A_tb <= "00"; B_tb <= "11"; C_in_tb <= '0';
 wait for time_period;
    A_tb <= "00"; B_tb <= "11"; C_in_tb <= '1';
 wait for time_period;
    A_tb <= "01"; B_tb <= "00"; C_in_tb <= '0';
 wait for time_period;
    A_tb <= "01"; B_tb <= "00"; C_in_tb <= '1';
 wait for time_period;
    A_tb <= "01"; B_tb <= "01"; C_in_tb <= '0';
 wait for time_period;
    A_tb <= "01"; B_tb <= "01"; C_in_tb <= '1';
 wait for time_period;
    A_tb <= "01"; B_tb <= "10"; C_in_tb <= '0';
 wait for time_period;
    A_tb <= "01"; B_tb <= "10"; C_in_tb <= '1';
 wait for time_period;
    A_tb <= "01"; B_tb <= "11"; C_in_tb <= '0';
 wait for time_period;
    A_tb <= "01"; B_tb <= "11"; C_in_tb <= '1';
 wait for time_period;
    A_tb <= "10"; B_tb <= "00"; C_in_tb <= '0';
 wait for time_period;
    A_tb <= "10"; B_tb <= "00"; C_in_tb <= '1';
 wait for time_period;
    A_tb <= "10"; B_tb <= "01"; C_in_tb <= '0';
 wait for time_period;
    A_tb <= "10";    B_tb <= "01"; C_in_tb <= '1';
 wait for time_period;
    A_tb <= "10"; B_tb <= "10"; C_in_tb <= '0';
wait for time_period;
    A_tb <= "10"; B_tb <= "10"; C_in_tb <= '1';
wait for time_period;
    A_tb <= "10"; B_tb <= "11"; C_in_tb <= '0';
wait for time_period;
    A_tb <= "10"; B_tb <= "11"; C_in_tb <= '1';
wait for time_period;
    A_tb <= "11"; B_tb <= "00"; C_in_tb <= '0';
wait for time_period;
    A_tb <= "11";    B_tb <= "00"; C_in_tb <= '1';
wait for time_period;
    A_tb <= "11"; B_tb <= "01"; C_in_tb <= '0';
wait for time_period;
      A_tb <= "11"; B_tb <= "01"; C_in_tb <= '1';
```

```
        wait for time_period;
            A_tb <= "11"; B_tb <= "10"; C_in_tb <= '0';
        wait for time_period;
            A_tb <= "11"; B_tb <= "10"; C_in_tb <= '1';
        wait for time_period;
            A_tb <= "11"; B_tb <= "11"; C_in_tb <= '0';
        wait for time_period;
            A_tb <= "11"; B_tb <= "11"; C_in_tb <= '1';
         wait;
    end process stimu;
end Behavioral;
```

Constraint file of the 2-bit full adder & the comparator:

In constraint file, each entity port is mapped to the Zybo Z7 board as shown below:

```
##Switches
set_property -dict { PACKAGE_PIN G15     IOSTANDARD LVCMOS33 } [get_ports { A[0] }];
#IO_L19N_T3_VREF_35 Sch=sw[0]
set_property -dict { PACKAGE_PIN P15     IOSTANDARD LVCMOS33 } [get_ports { A[1] }];
#IO_L24P_T3_34 Sch=sw[1]
set_property -dict { PACKAGE_PIN W13     IOSTANDARD LVCMOS33 } [get_ports { B[0] }];
#IO_L4N_T0_34 Sch=sw[2]
set_property -dict { PACKAGE_PIN T16     IOSTANDARD LVCMOS33 } [get_ports { B[1] }];
#IO_L9P_T1_DQS_34 Sch=sw[3]
##Buttons
set_property -dict { PACKAGE_PIN K18     IOSTANDARD LVCMOS33 } [get_ports { C_in }];
#IO_L12N_T1_MRCC_35 Sch=btn[0]
##LEDs
set_property -dict { PACKAGE_PIN M14     IOSTANDARD LVCMOS33 } [get_ports { Sum[0] }];
#IO_L23P_T3_35 Sch=led[0]
set_property -dict { PACKAGE_PIN M15     IOSTANDARD LVCMOS33 } [get_ports { Sum[1] }];
#IO_L23N_T3_35 Sch=led[1]
set_property -dict { PACKAGE_PIN G14     IOSTANDARD LVCMOS33 } [get_ports { C_out }];
#IO_0_35 Sch=led[2]
##RGB LED 6
set_property -dict { PACKAGE_PIN V16     IOSTANDARD LVCMOS33 } [get_ports { compare_result[0] }];
#IO_L18P_T2_34 Sch=led6_r
set_property -dict { PACKAGE_PIN F17     IOSTANDARD LVCMOS33 } [get_ports { compare_result[1] }];
#IO_L6N_T0_VREF_35 Sch=led6_g
set_property -dict { PACKAGE_PIN M17     IOSTANDARD LVCMOS33 } [get_ports { compare_result[2] }];
#IO_L8P_T1_AD10P_35 Sch=led6_b
```