ECE 511 DIGITAL ASIC DESIGN

Report of Lab 2: Finite State Machines (FSMs) and Vending Machine using VHDL

Student Name : Gangyi Li
Student Number : 1674819

November 14, 2021

Department of Electrical and Computer Engineering
University of Alberta

Abstract:

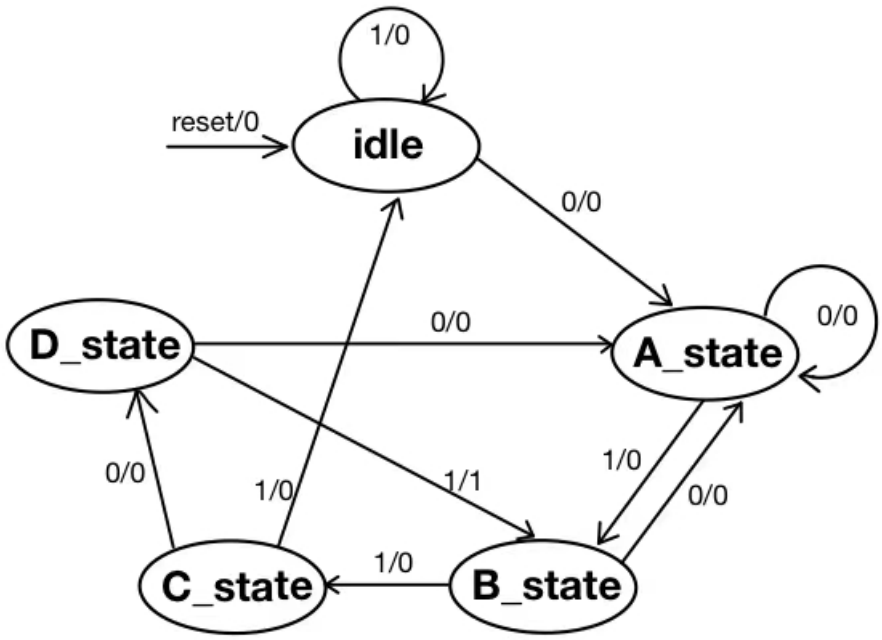
The purpose of this lab is to use the functionality of Xilinx Vivado to design and test finite state machines(FSM) on Zybo Z7 board. This lab involves two parts, which are sequence detector and the vending machine. All the design is written in VHDL and assigned the pins of each port to program.

Part #1: Sequence Detector "01101"

1.1 Design

A sequence detector can detect one specific code and the output can show ‘1’ to represent the result. In this lab, the sequential binary code is “01101” and can be detected with overlapping and based on Mealy machine model. The schematic of state transition graph based on Mealy machine are shown in figure 1.1 . In this model, there are five states in the structure, which are idle, A_state, B_state, C_state and D_state.

- For the idle state, when the input is '1', the next state remains unchanged, and when the input is '0', the next state is A_state. The output value is '0'.
- For the A_state, when the input is '1', the next state is B_state, and when the input is '0', the next state remains unchanged. The output value is '0'.
- For the B_state, when the input is '1', the next state is C_state, and when the input is '0', the next state is A_state. The output value is '0'.
- For the C_state, when the input is '1', the next state is idle, and when the input is '0', the next state is D_state. The output value is '0'.
- For the D_state, when the input is '1', the next state is B_state, and when the input is '0', the next state is A_state. The output value is '1'.

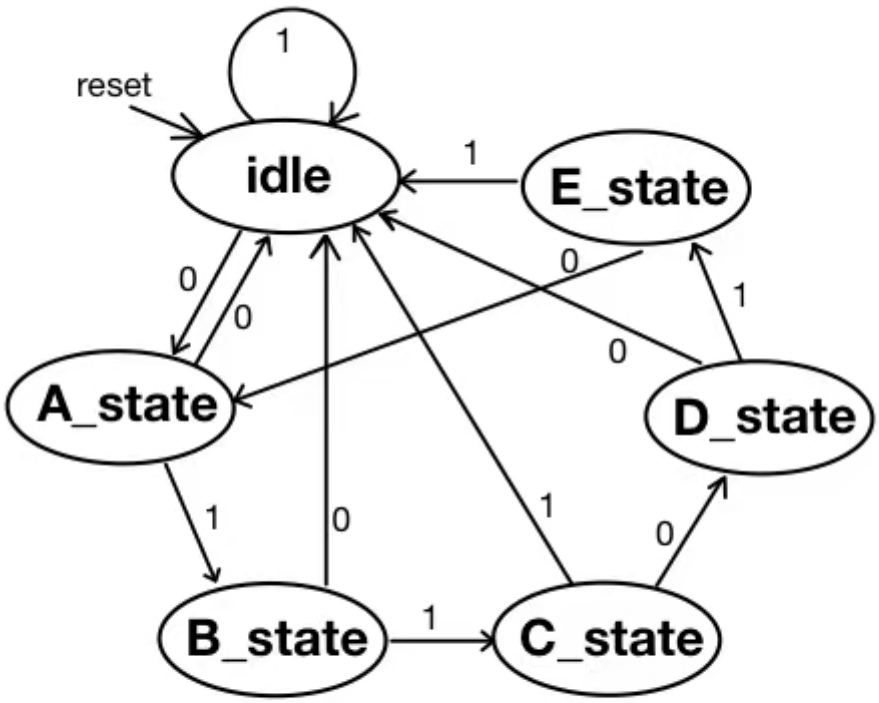


reset	din(input)	current state	next state	dout(output)
1	------	------	idle	0
0	0	idle	A_state	0
0	1	idle	idle	0
0	0	A_state	A_state	0
0	1	A_state	B_state	0
0	0	B_state	A_state	0
0	1	B_state	C_state	0
0	0	C_state	D_state	0
0	1	C_state	idle	0
0	0	D_state	A_state	0
0	1	D_state	B_state	1

Figure1.1: Schematic of State Transition Graph based on Mealy Machine

Meanwhile, Moore machine model can be used to design a detector for code without overlapping. The schematic of Moore machine and state transition graph are shown in figure 1.2 . In this model, there are states in the structure, which are idle, A_state, B_state, C_state, D_state and E_state. These states represent output as '0', '0', '0', '0', '0' and '1' respectively.

- For the idle state, when the input is '1', the next state remains unchanged, and when the input is '0', the next state is A_state.
- For the A_state, when the input is '1', the next state is B_state, and when the input is '0', the next state is idle.
- For the B_state, when the input is '1', the next state is C_state, and when the input is '0', the next state is idle.
- For the C_state, when the input is '1', the next state is idle, and when the input is '0', the next state is idle.
- For the D_state, when the input is '1', the next state is B_state, and when the input is '0', the next state is idle.
- For the E_state, when the input is '1', the next state is idle, and when the input is '0', the next state is A_state.



state	output
idle	0
A_state	0
B_state	0
C_state	0
D_state	0
E_state	1

Figure1.2: Schematic of State Transition Graph based on Moore Machine

In this lab, Mealy machine structure is chosen to realize this sequence detector with overlapping. For the design code part, two-process is better to realize this function. The clock process set to the next state will happen while there is a rising-edge of the clock signal. And the other process sets the next states for different conditions within each state, according to the state transition graph. The design code shown as below:

Design Code

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

entity sequence_detector is
    Port( clk : in std_logic;
```

```
          reset : in std_logic;
          din : in std_logic;
          dout : out std_logic);
end entity sequence_detector;

architecture behaveiour of sequence_detector is
    type stateType is (idle, A_state, B_state, C_state, D_state);
    signal current_state, next_state : stateType;

begin
    com : process(din, current_state) is
    begin
        case current_state is
            when idle =>
                if(din = '0') then
                    next_state <= A_state;
                    dout <= '0';
                else
                    next_state <= idle;
                    dout <= '0';
                end if;

            when A_state =>
                if(din = '1') then
                    next_state <= B_state;
                    dout <= '0';
                else
                    next_state <= A_state;
                    dout <= '0';
                end if;

            when B_state =>
                if(din = '1') then
                    next_state <= C_state;
                    dout <= '0';
                else
                    next_state <= A_state;
                    dout <= '0';
                end if;

            when C_state =>
                if(din = '0') then
                    next_state <= D_state;
                    dout <= '0';
                else
                    next_state <= idle;
                    dout <= '0';
                end if;

            when D_state =>
                if(din = '1') then
                    next_state <= B_state;
                    dout <= '1';
                else
                    next_state <= A_state;
                    dout <= '0';
                end if;
        end case;
```

```
    end process com;

    seq : process(reset, clk) is
    begin
        if reset = '1' then
            current_state <= idle;
        elsif rising_edge(clk) then
            current_state <= next_state;
        end if;
    end process seq;
end architecture;
```

1.2 Simulations and Tests

To test the functionality of the circuit design and design code, the test sequence : “11100101101101011001 11” can be used in the simulation process of the testbench code. When the clock signal changes from “0” to “1”, the state will transform to the next state. For this test sequence, the output value will show '1' twice. The figure 1.3 and figure 1.4 show the waveform and synthesis schematic for this simulation. The testbench code part shown as below:

Testbench Code

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

entity tb_sequence_detector is
end entity;

architecture tb_behaviour of tb_sequence_detector is
    component sequence_detector
        Port(clk : in std_logic;
             reset : in std_logic;
             din : in std_logic;
             dout : out std_logic);
    end component;
    signal din_tb : std_logic := '0';
    signal clk_tb : std_logic := '0';
    signal reset_tb : std_logic := '0';
    signal dout_tb : std_logic := '0';
    constant clock_period_tb : time := 10ns;
begin
    sd : sequence_detector port map(din=>din_tb, clk=>clk_tb, reset=>reset_tb,
dout=>dout_tb);
    clock : process
    begin
        clk_tb <= '0';
        wait for clock_period_tb/2;
        clk_tb <= '1';
        wait for clock_period_tb/2;
    end process;

    stimulus : process
    begin
        reset_tb <= '1';
        wait for 10ns;
        reset_tb <= '0';
```

```
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '0';
        wait for 10ns;
        din_tb <= '0';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '0';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '0';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '0';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '0';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '0';
        wait for 10ns;
        din_tb <= '0';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
        din_tb <= '1';
        wait for 10ns;
    end process stimulus;
end architecture tb_behaviour;
```

For the constraint file, in order to verify and test the design on board, hand pressing buttons can be used as clock and reset signal. When the buttons are pressed, the clock and reset signal will show '1'. The input data signal can be assigned to the pin of one switch, and the output signal is assigned to the led on board. When the sequence code is detected, the led will be lightened, as shown in figure 1.5. The test sequence can be entered by controlling the switch on board. The constraints shown as below:

Constraints

```
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF];
set_property -dict {PACKAGE_PIN P16 IOSTANDARD LVCMOS33} [get_ports clk]; #clk =
btn[1]

set_property -dict {PACKAGE_PIN G15 IOSTANDARD LVCMOS33} [get_ports din]; #din =
sw[0]

set_property -dict {PACKAGE_PIN Y16 IOSTANDARD LVCMOS33} [get_ports reset];
#reset=btn[3]

set_property -dict {PACKAGE_PIN M14 IOSTANDARD LVCMOS33} [get_ports dout]; #dout
= led[0]
```

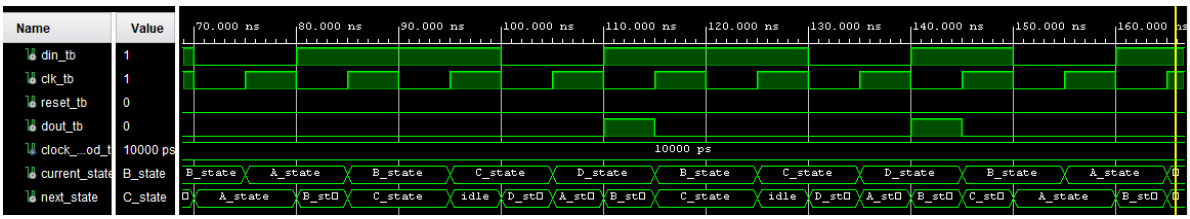


Figure1.3: Simulation Waveform for Sequence Detector

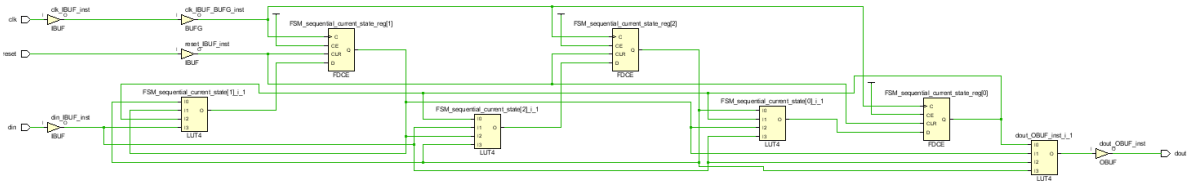


Figure1.4: Synthesis Design for Sequence Detector

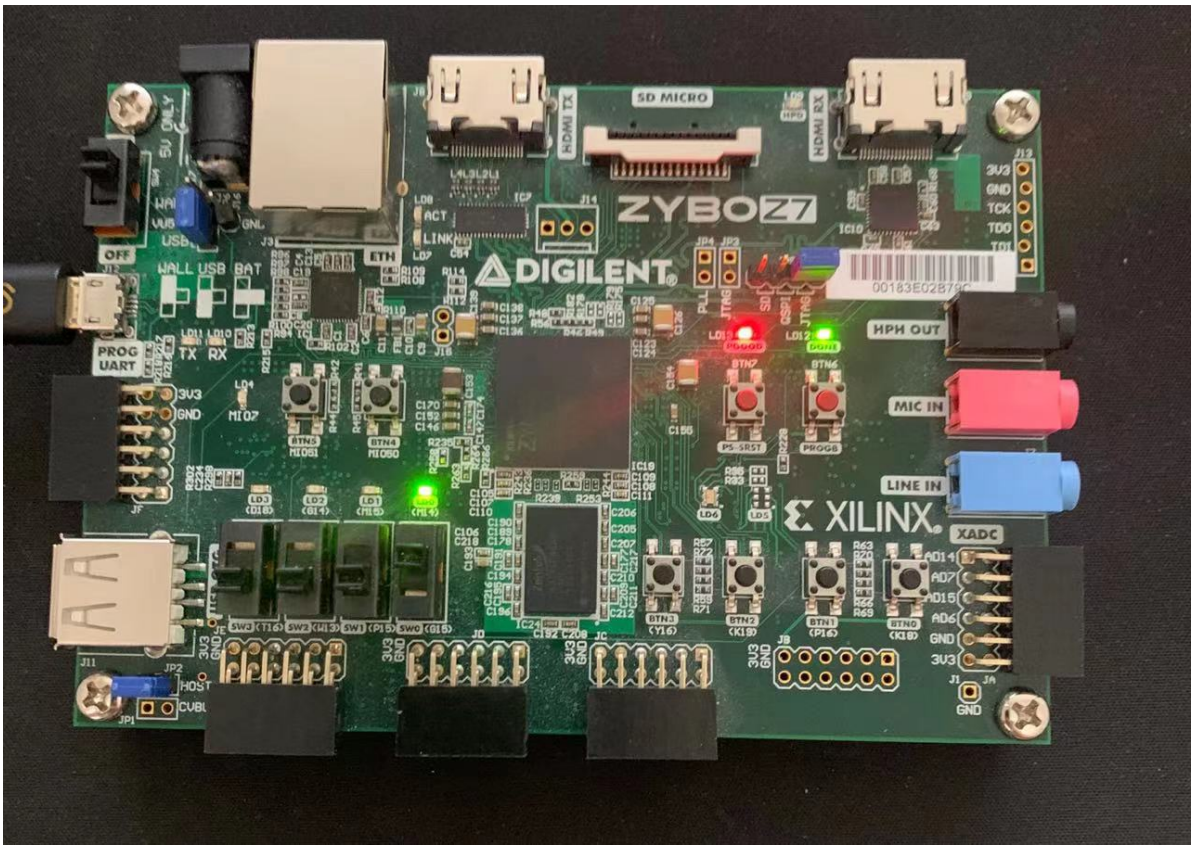


Figure1.5: LED0 of Zybo Z7-10 Board lightening when input is "01101"

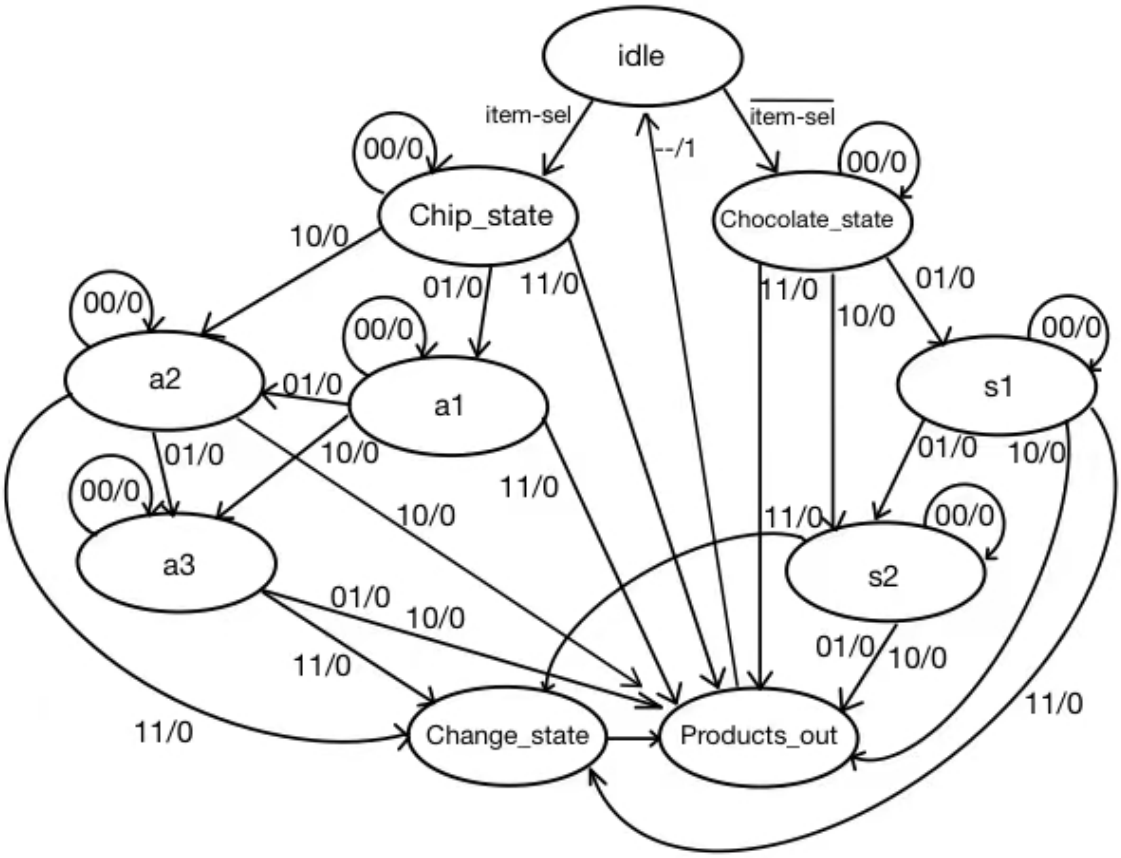
Part #2: Vending Machine

2.1 Design

In this part, a vending machine that sells chips and chocolates can be designed. Customers can select the product and then insert money. After that, the vending machine will return the change and dispense the product. The denominations of 1$, 2$ and 5$ can be used only in this machine. The input of this design is “coins_in”, “item_sel”, “clk”, “reset”, and the output is “change_out”, "sel_segment", “display_sum”, “chips” and “chocolates”. The current sum of inserted money shows on the seven segment, which can display on the left segment by assigning signal “sel_segment” as “1”. To display specific numbers, the signal “display_sum” can be assigned as below:

number	segment code(hex)	segment code(bin)
0	0x3F	0111111
1	0x06	0000110
2	0x5B	1011011
3	0x4F	1001111
4	0x66	1100110
5	0x6D	1101101
6	0x7D	1111101
7	0x07	0000111
8	0x7F	1111111
9	0x6F	1101111

In this vending machine, there are ten states in this design, which are “idle”, “Chip_state”, “a1”, “a2”, “a3”, “Chocolate_state”, “s1”, “s2”, “change_state” and “Products_out”. This machine can choose products by controlling input signal "item_sel" to step into "Chip_state" and "Chocolate_state". The a1, a2 and a3 states represent 1$, 2$ and 3$ for chip selection. The s1 and s2 states represent 1$ and 2$ for chocolate selection. If there needs two processes for giving changes (3$ = 1$ and 2$), the first change signal will be generated in the money process(a1-a3, s1-s2), and the next state will be change_state, which involves denomination of 2$. After making changes, the next state will be the "Products_out" state, which can dispense different products based on "item_sel" signal value. The state transition graph of vending machine is shown as figure2.1.


idle
item-sel
item-sel
--/1
00/0
Chip_state
00/0
Chocolate_state
10/0
01/0
11/0
11/0
10/0
01/0
00/0
a2
00/0
a1
00/0
s1
01/0
01/0
10/0
01/0
10/0
00/0
10/0
11/0
11/0
00/0
a3
s2
01/0
10/0
01/0
10/0
11/0
11/0
Change_state
Products_out

reset	item_sel	coins_in	current_state	next_state	change_out	display_sum	chips	chocolates
1	------	------	------	idle	00	0111111	0	0
0	------	------	change_state	Products_out	10	0111111	0	0
0	1	------	Products_out	idle	00	0111111	1	0
0	0	------	Products_out	idle	00	0111111	0	1
0	1	------	idle	Chip_state	00	0111111	0	0
0	0	------	idle	Chocolate_state	00	0111111	0	0
0	1	00	Chip_state	Chip_state	00	0111111	0	0
0	1	01	Chip_state	a1	00	0000110	0	0
0	1	10	Chip_state	a2	00	1011011	0	0
0	1	11	Chip_state	Products_out	01	1101101	0	0
0	1	00	a1	a1	00	0000110	0	0
0	1	01	a1	a2	00	1011011	0	0
0	1	10	a1	a3	00	1001111	0	0
0	1	11	a1	Products_out	10	1111101	0	0
0	1	00	a2	a2	00	1011011	0	0
0	1	01	a2	a3	00	1001111	0	0
0	1	10	a2	Products_out	00	1100110	0	0
0	1	11	a2	change_state	01	0000111	0	0
0	1	00	a3	a3	00	1001111	0	0
0	1	01	a3	Products_out	00	1100110	0	0
0	1	10	a3	Products_out	01	1101101	0	0
0	1	11	a3	change_state	10	1111111	0	0
0	0	00	chocolate_state	chocolate_state	00	0111111	0	0
0	0	01	chocolate_state	s1	00	0000110	0	0
0	0	10	chocolate_state	s2	00	1011011	0	0
0	0	11	chocolate_state	Products_out	10	1101101	0	0
0	0	00	s1	s1	00	0000110	0	0
0	0	01	s1	s2	00	1011011	0	0
0	0	10	s1	Products_out	00	1001111	0	0
0	0	11	s1	change_state	01	1111101	0	0
0	0	00	s2	s2	00	1011011	0	0
0	0	01	s2	Products_out	00	1001111	0	0
0	0	10	s2	Products_out	01	1100110	0	0
0	0	11	s2	change_state	10	0000111	0	0

Figure2.1: Schematic of State Transition Graph based on vending machine

To let the design be successfully tested on board, the clock divider needs to be used in the design to trigger the rising edge of the clock signal once in 2 seconds. The design code is shown below:

Design Code for Clock Divider

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

entity clk_divider is
    Port ( clk_in : in STD_LOGIC;
           clk_out : out STD_LOGIC);
```

```
end clk_divider;

architecture Behavioral of clk_divider is
signal clock_out : std_logic := '0';
signal count : integer := 1;
begin
    process(clk_in)
    begin
        if clk_in='1' and clk_in'event then
            count <= count + 1;
            if(count = 62500000) then
                clock_out <= NOT clock_out;
                count <= 1;
            end if;
        end if;
    clk_out <= clock_out;
    end process;
end Behavioral;
```

Design Code for Vending Machine

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

entity vending is
    Port( clk : in std_logic;
          reset : in std_Logic;
          item_sel : in std_logic;
          coins_in : in std_logic_vector(1 downto 0);
          change_out : out std_logic_vector(1 downto 0);
          display_sum : out std_logic_vector(6 downto 0);
          select_segment : out std_logic;
          chips : out std_logic;
          chocolates : out std_logic);
end entity vending;

architecture behaviour of vending is
    type stateType is (idle, Chip_state, a1, a2, a3, Chocolate_state, s1, s2,
change_state, Products_out);
    signal current_state, next_state : stateType;
    signal clk_out_component : std_logic;
    component clk_divider is
        Port( clk_in : in std_logic;
              clk_out : out std_logic);
    end component;
begin
    label_clk_divider : clk_divider port map(clk_in => clk, clk_out =>
clk_out_component);
    select_segment <= '1';
    com : process(item_sel, coins_in, current_state) is
    begin
        case current_state is
            when idle =>
                if(item_sel = '1') then
                    next_state <= Chip_state;
                    change_out <= "00";
                    chips <= '0'; chocolates <= '0';
```

```
                display_sum <= "0111111";
            else
                next_state <= Chocolate_state;
                change_out <= "00";
                chips <= '0'; chocolates <= '0';
                display_sum <= "0111111";
            end if;

        when Chip_state =>
            if(coins_in = "00") then
                next_state <= Chip_state;
                change_out <= "00";
                display_sum <= "0111111";
                chips <= '0'; chocolates <= '0';
            elsif (coins_in = "01") then
                next_state <= a1;
                change_out <= "00";
                display_sum <= "0000110";
                chips <= '0'; chocolates <= '0';
            elsif (coins_in = "10") then
                next_state <= a2;
                change_out <= "00";
                display_sum <= "1011011";
                chips <= '0'; chocolates <= '0';
            else
                next_state <= Products_out;
                change_out <= "01";
                display_sum <= "1101101";
            end if;

        when a1 =>
            if(coins_in = "00") then
                next_state <= a1;
                change_out <= "00";
                display_sum <= "0000110";
                chips <= '0'; chocolates <= '0';
            elsif (coins_in = "01") then
                next_state <= a2;
                change_out <= "00";
                display_sum <= "1011011";
                chips <= '0'; chocolates <= '0';
            elsif (coins_in = "10") then
                next_state <= a3;
                change_out <= "00";
                display_sum <= "1001111";
                chips <= '0'; chocolates <= '0';
            else
                next_state <= Products_out;
                change_out <= "10";
                display_sum <= "1111101";
            end if;

        when a2 =>
            if(coins_in = "00") then
                next_state <= a2;
                change_out <= "00";
                display_sum <= "1011011";
                chips <= '0'; chocolates <= '0';
```

```
            elsif (coins_in = "01") then
                next_state <= a3;
                change_out <= "00";
                display_sum <= "1001111";
                chips <= '0'; chocolates <= '0';
            elsif (coins_in = "10") then
                next_state <= Products_out;
                change_out <= "00";
                display_sum <= "1100110";
            else
                next_state <= change_state;
                change_out <= "01";
                display_sum <= "0000111";
            end if;

        when a3 =>
            if(coins_in = "00") then
                next_state <= a3;
                change_out <= "00";
                display_sum <= "1001111";
                chips <= '0'; chocolates <= '0';
            elsif (coins_in = "01") then
                next_state <= Products_out;
                change_out <= "00";
                display_sum <= "1100110";
            elsif (coins_in = "10") then
                next_state <= Products_out;
                change_out <= "01";
                display_sum <= "1101101";
            else
                next_state <= change_state;
                change_out <= "10";
                display_sum <= "1111111";
            end if;

        when Chocolate_state =>
            if(coins_in = "00") then
                next_state <= Chocolate_state;
                change_out <= "00";
                display_sum <= "0111111";
                chips <= '0'; chocolates <= '0';
            elsif (coins_in = "01") then
                next_state <= s1;
                change_out <= "00";
                display_sum <= "0000110";
                chips <= '0'; chocolates <= '0';
            elsif (coins_in = "10") then
                next_state <= s2;
                change_out <= "00";
                display_sum <= "1011011";
                chips <= '0'; chocolates <= '0';
            else
                next_state <= Products_out;
                change_out <= "10";
                display_sum <= "1101101";
            end if;

        when s1 =>
```

```
            if(coins_in = "00") then
                next_state <= s1;
                change_out <= "00";
                display_sum <= "0000110";
                chips <= '0'; chocolates <= '0';
            elsif (coins_in = "01") then
                next_state <= s2;
                change_out <= "00";
                display_sum <= "1011011";
                chips <= '0'; chocolates <= '0';
            elsif (coins_in = "10") then
                next_state <= Products_out;
                change_out <= "00";
                display_sum <= "1001111";
            else
                next_state <= change_state;
                change_out <= "01";
                display_sum <= "1111101";
            end if;

        when s2 =>
            if(coins_in = "00") then
                next_state <= s2;
                change_out <= "00";
                display_sum <= "1011011";
                chips <= '0'; chocolates <= '0';
            elsif (coins_in = "01") then
                next_state <= Products_out;
                change_out <= "00";
                display_sum <= "1001111";
            elsif (coins_in = "10") then
                next_state <= Products_out;
                change_out <= "01";
                display_sum <= "1100110";
            else
                next_state <= change_state;
                change_out <= "10";
                display_sum <= "0000111";
            end if;

        when change_state =>
                change_out <= "10";
                next_state <= Products_out;

        when Products_out =>
            change_out <= "00";
            display_sum <= "0111111";
            if(item_sel = '1') then
                chips <= '1';
                chocolates <= '0';
            else
                chips <= '0';
                chocolates <= '1';
            end if;
            next_state <= idle;
    end case;
end process com;
```

```
    seq : process(reset, clk_out_component) is
    begin
        if reset = '1' then
            current_state <= idle;
        elsif rising_edge(clk_out_component) then
            current_state <= next_state;
        end if;
    end process seq;
end architecture behaviour;
```

2.2 Simulations and Tests

In the simulation part, the expected display can not show in waveform because 125000000 pulses are counted before making an increment in the clock divider design. To solve this problem and test design simulation results, the counting number needs to be reduced to 1, which can divide the clock by 2. The waveform is shown in figure2.2. The testbench code is shown as below:

Testbench Code

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

entity tb_vending is
end entity;

architecture tb_behaviour of tb_vending is
    component vending
        Port( clk : in std_logic;
          reset : in std_Logic;
          item_sel : in std_logic;
          coins_in : in std_logic_vector(1 downto 0);
          change_out : out std_logic_vector(1 downto 0);
          display_sum : out std_logic_vector(6 downto 0);
          select_segment : out std_logic;
          chips : out std_logic;
          chocolates : out std_logic);
    end component;

    component clk_divider
           Port ( clk_in : in STD_LOGIC;
           clk_out : out STD_LOGIC);
    end component;
    signal clk_in_tb : std_logic;
    signal clk_out_tb : std_logic;
    signal reset_tb : std_logic := '0';
    signal item_sel_tb : std_logic := '1';
    signal coins_in_tb : std_logic_vector(1 downto 0) := "00";
    signal display_sum_tb : std_logic_vector(6 downto 0) := "0000000";
    signal change_out_tb : std_logic_vector(1 downto 0) := "00";
    signal select_segment_tb : std_logic;
    signal chips_tb : std_logic;
    signal chocolates_tb : std_logic;
    constant clock_period_tb : time := 20ns;
begin
```

```
    vending_tb : vending port map(clk => clk_in_tb, reset => reset_tb, item_sel
=> item_sel_tb, coins_in => coins_in_tb, change_out => change_out_tb,
display_sum => display_sum_tb, select_segment => select_segment_tb, chips =>
chips_tb, chocolates => chocolates_tb);
    divider : clk_divider port map(clk_in => clk_in_tb, clk_out => clk_out_tb);

    clock : process
    begin
        clk_in_tb <= '0';
        wait for clock_period_tb/2;
        clk_in_tb <= '1';
        wait for clock_period_tb/2;
    end process;

    stimulus : process
    begin
        reset_tb <= '1';
        wait for 40ns;
        reset_tb <= '0';
        item_sel_tb <= not item_sel_tb;
        wait for 40ns;
        coins_in_tb <= "01";
        wait for 40ns;
        coins_in_tb <= "10";
        wait for 40ns;
        coins_in_tb <= "11";
        wait for 40ns;
        coins_in_tb <= "10";
        wait for 40ns;
        coins_in_tb <= "10";
        wait for 40ns;
        coins_in_tb <= "10";
        wait for 40ns;
        coins_in_tb <= "11";
        wait for 40ns;
    end process;
end architecture;
```

In the constraint file, for the input signals, "reset" signal is assigned to button(0), "coins_in" signals are assigned to switch(0) and switch(1) and "item_sel" signal is assigned to switch(2). For the output signals, "change_out" signals are assigned to LED(3) and LED(2), and "chips" and "chocolates" signals are assigned to LED(1) and LED(0). To display the sum of money inserted, "display_sum" and "select_segment" signals are assigned to JD and JC pins. When the switches are changing, the sum will show on the seven segment. If the sum reaches at the price, the led will be lightened. And if there are changes, the led will be lightened before the product is dispensing. The results on board are shown in figure2.3.

Constraints

```
set_property -dict {PACKAGE_PIN K17 IOSTANDARD LVCMOS33} [get_ports { clk }];
create_clock -period 64.000 -name sys_clk_pin -waveform {0.000 32.000} -add
[get_ports { clk }];

set_property -dict {PACKAGE_PIN K18 IOSTANDARD LVCMOS33} [get_ports { reset }];
```

```
set_property -dict {PACKAGE_PIN P15 IOSTANDARD LVCMOS33} [ get_ports {
coins_in[0] }];
set_property -dict {PACKAGE_PIN W13 IOSTANDARD LVCMOS33} [get_ports { coins_in[1]
}];
set_property -dict {PACKAGE_PIN T16 IOSTANDARD LVCMOS33} [get_ports { item_sel
}];

set_property -dict {PACKAGE_PIN G14 IOSTANDARD LVCMOS33} [get_ports {
change_out[0] }];
set_property -dict {PACKAGE_PIN D18 IOSTANDARD LVCMOS33} [get_ports {
change_out[1] }];

set_property -dict {PACKAGE_PIN V15 IOSTANDARD LVCMOS33} [get_ports {
display_sum[0] }];
set_property -dict {PACKAGE_PIN W15 IOSTANDARD LVCMOS33} [get_ports {
display_sum[1] }];
set_property -dict {PACKAGE_PIN T11 IOSTANDARD LVCMOS33} [get_ports {
display_sum[2] }];
set_property -dict {PACKAGE_PIN T10 IOSTANDARD LVCMOS33} [get_ports {
display_sum[3] }];
set_property -dict {PACKAGE_PIN T14 IOSTANDARD LVCMOS33} [get_ports {
display_sum[4] }];
set_property -dict {PACKAGE_PIN T15 IOSTANDARD LVCMOS33} [get_ports {
display_sum[5] }];
set_property -dict {PACKAGE_PIN P14 IOSTANDARD LVCMOS33} [get_ports {
display_sum[6] }];
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports {
select_segment }];

set_property -dict {PACKAGE_PIN M14 IOSTANDARD LVCMOS33} [get_ports { chips }];
set_property -dict {PACKAGE_PIN M15 IOSTANDARD LVCMOS33} [get_ports { chocolates
}];
```

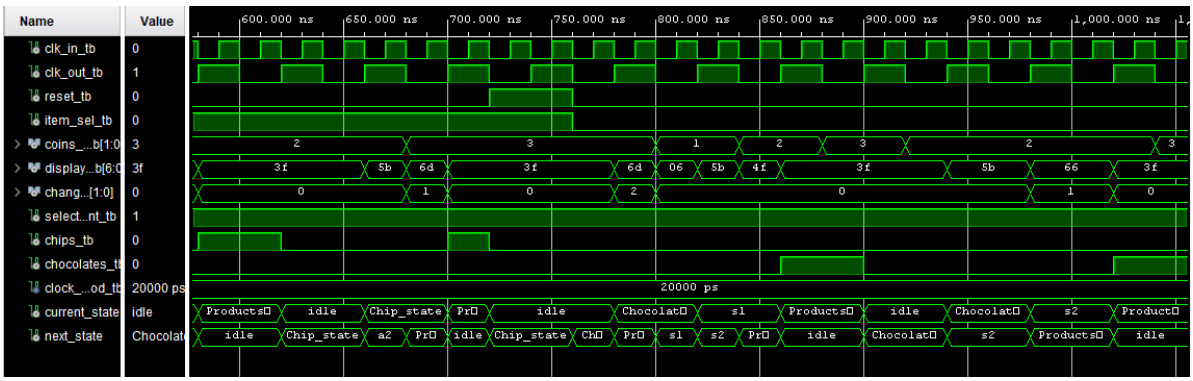
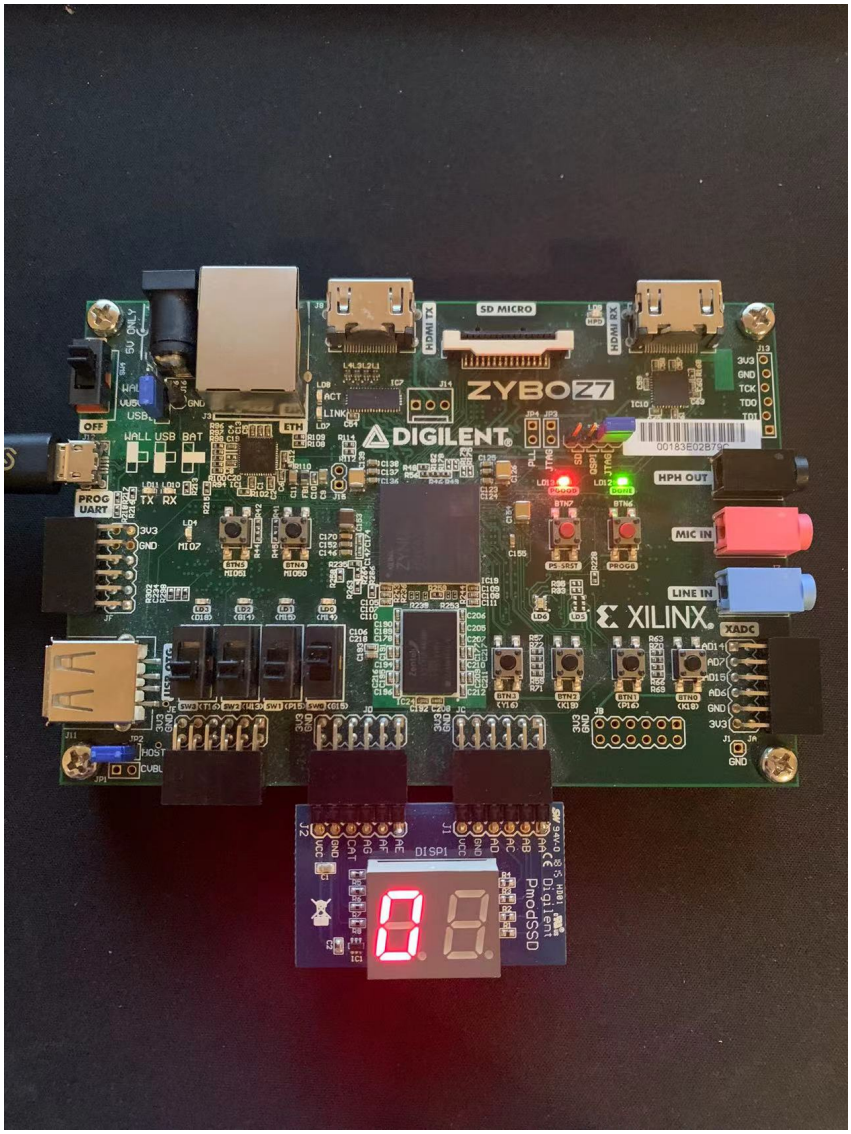
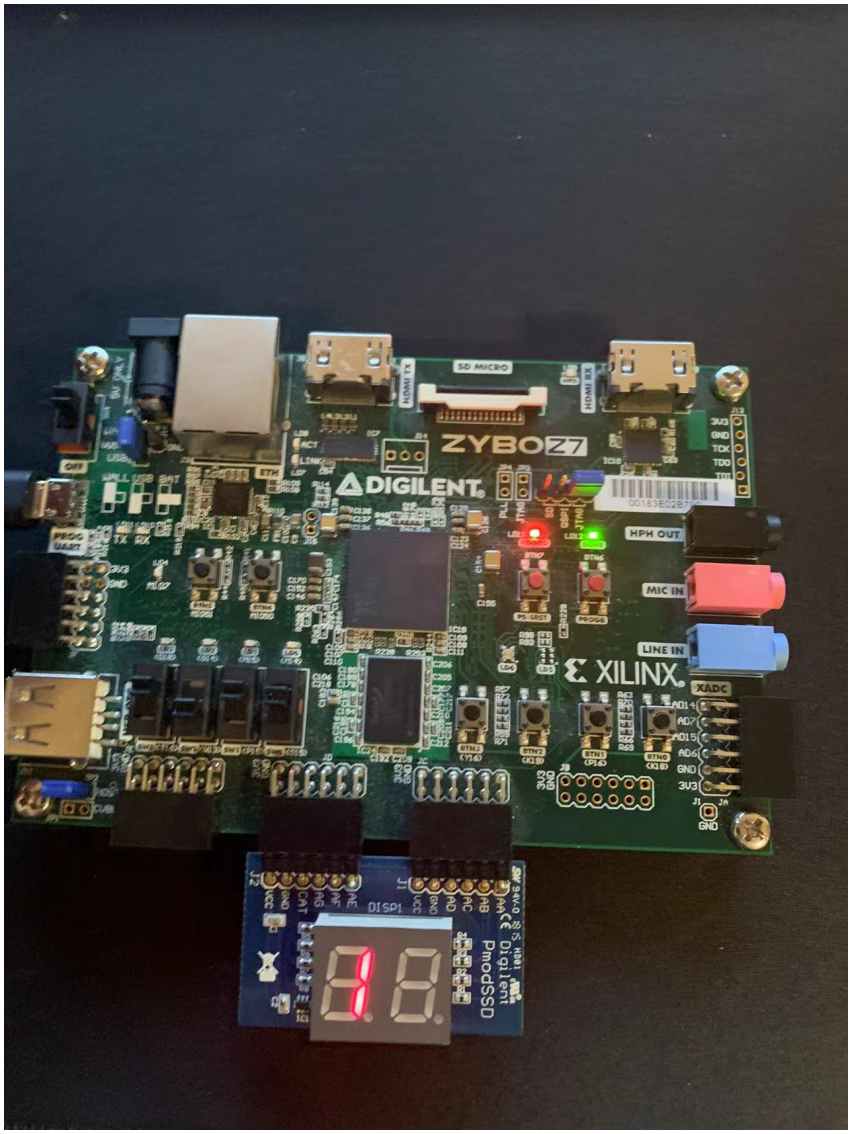
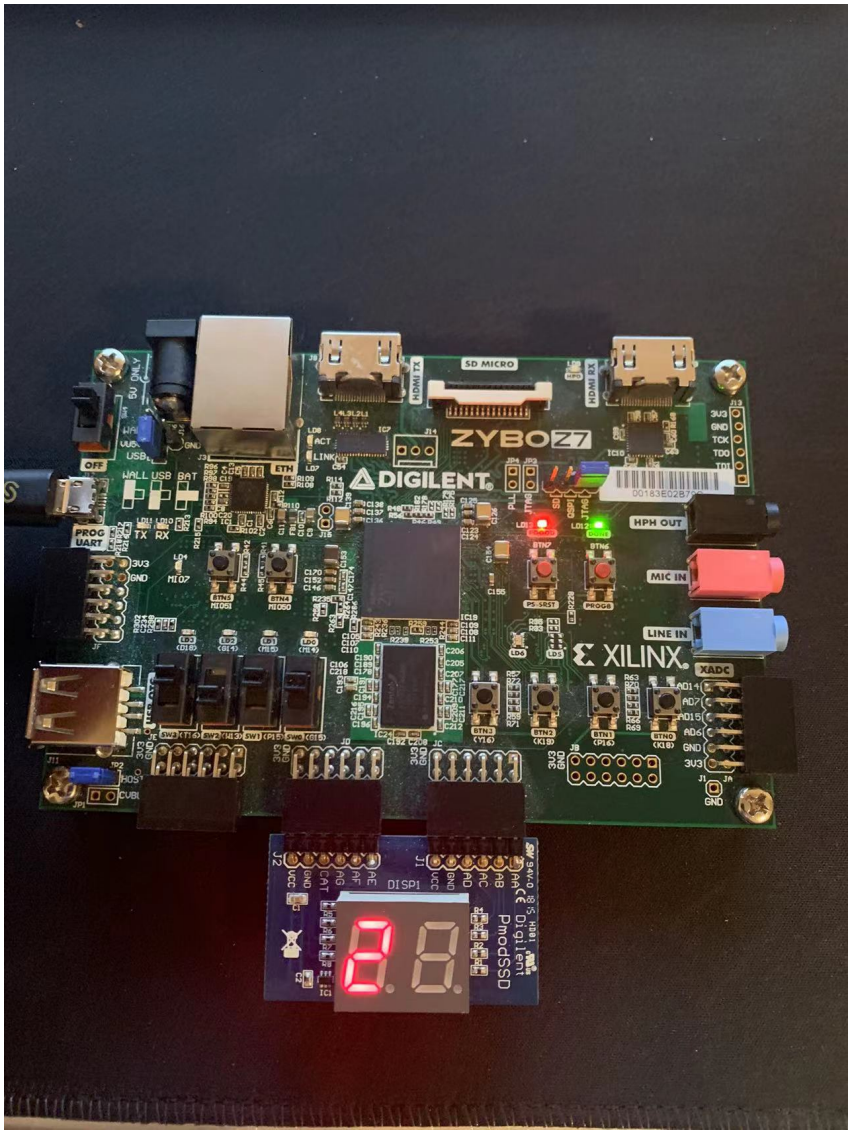


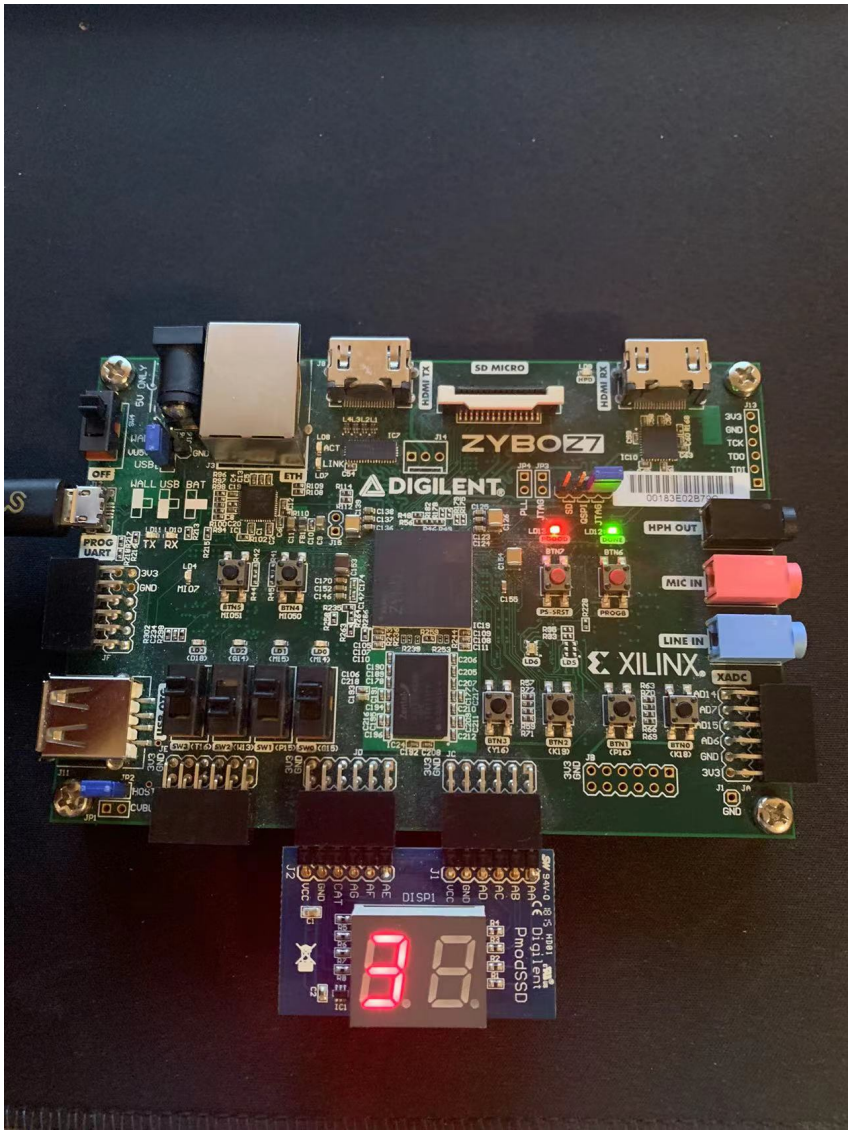
Figure2.2: Simulation Waveform for Sequence Detector

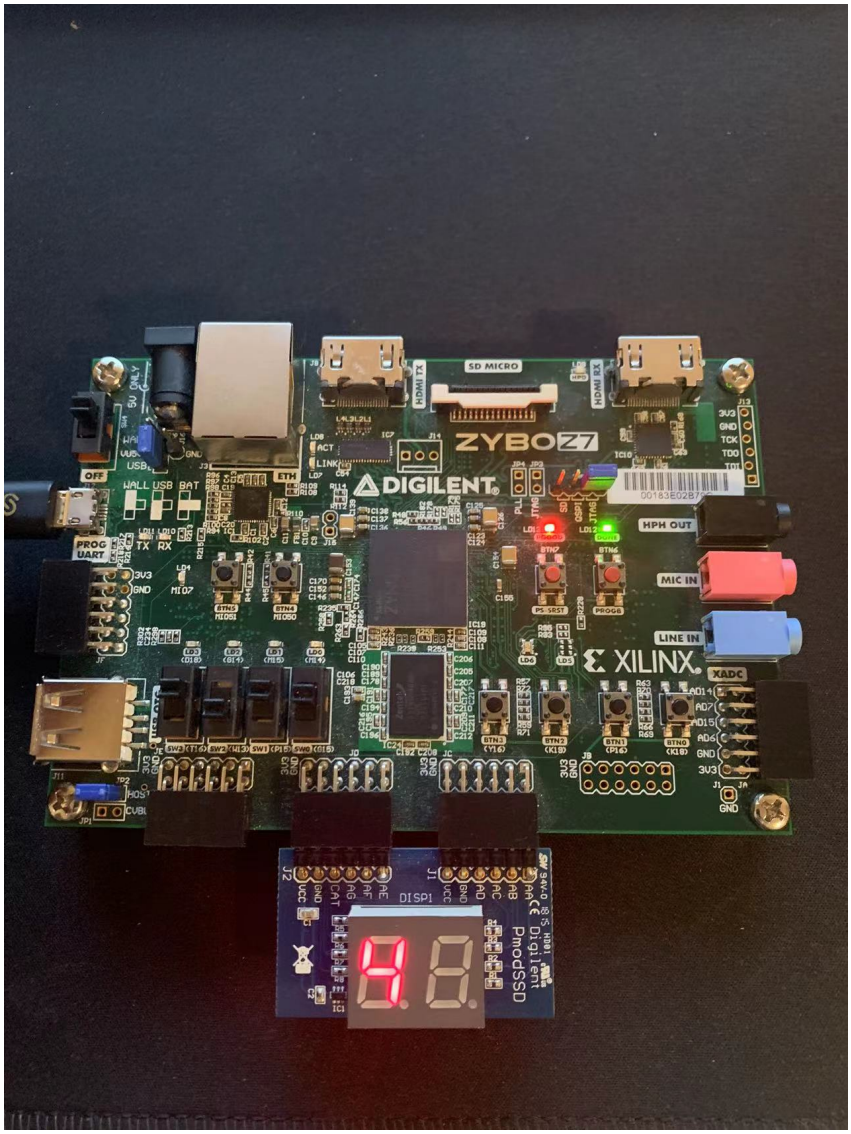
Chip selection

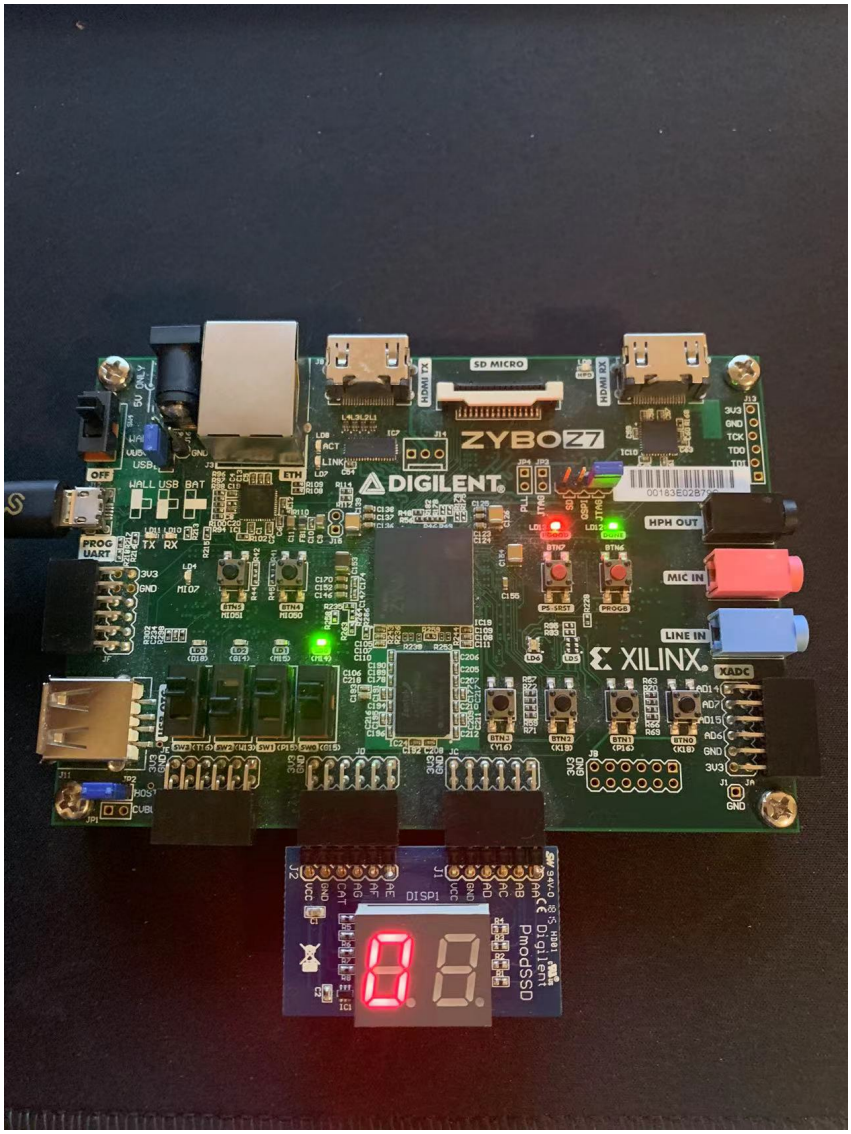

ZYBO Z7
DIGILENT
SD MICRO
HDMI TX
HDMI RX
HPH OUT
MIC IN
LINE IN
XADC
XILINX
PROG UART
ETH
OFF
WALL USB BAT
DISP1
PmodSSD
Digilent

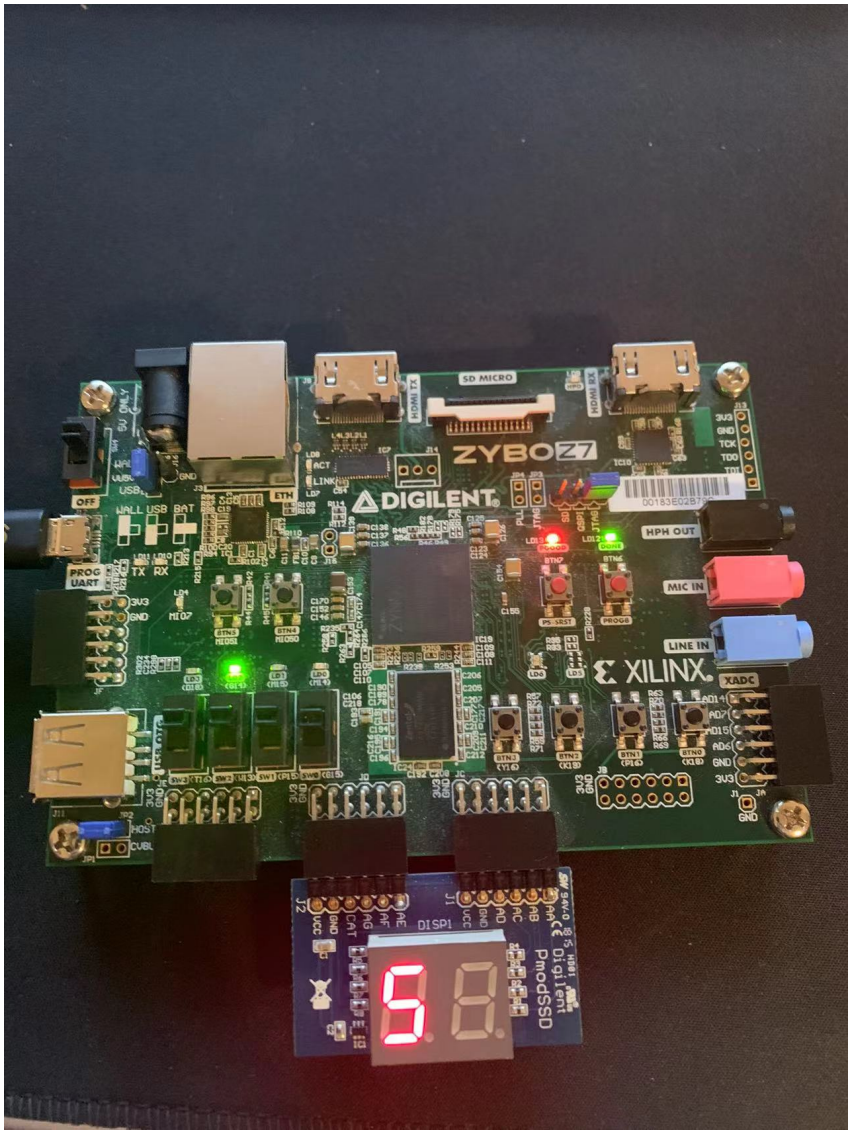

ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
HPH OUT
MIC IN
LINE IN
XADC
PmodSSD
Digilent

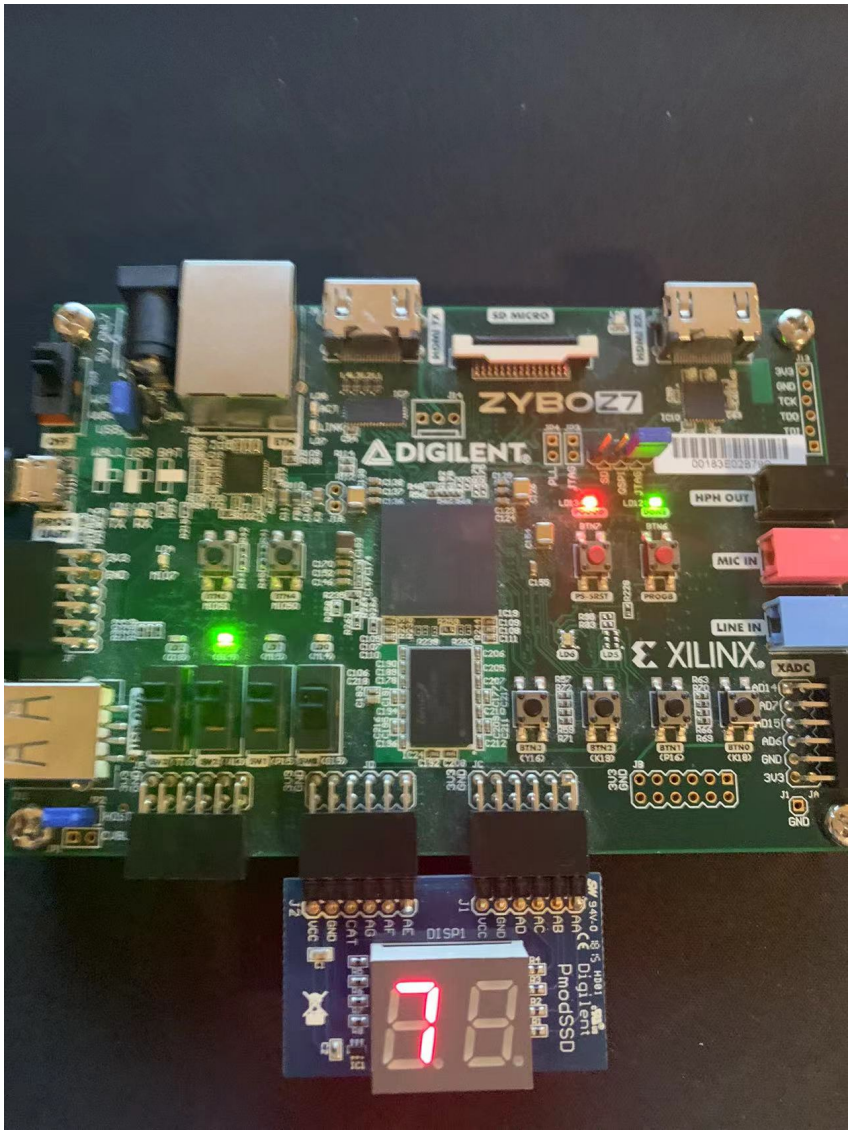

ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
HPH OUT
MIC IN
LINE IN
XADC
PROG UART
WALL USB BAT
OFF
DISP1
Digilent
PmodSSD

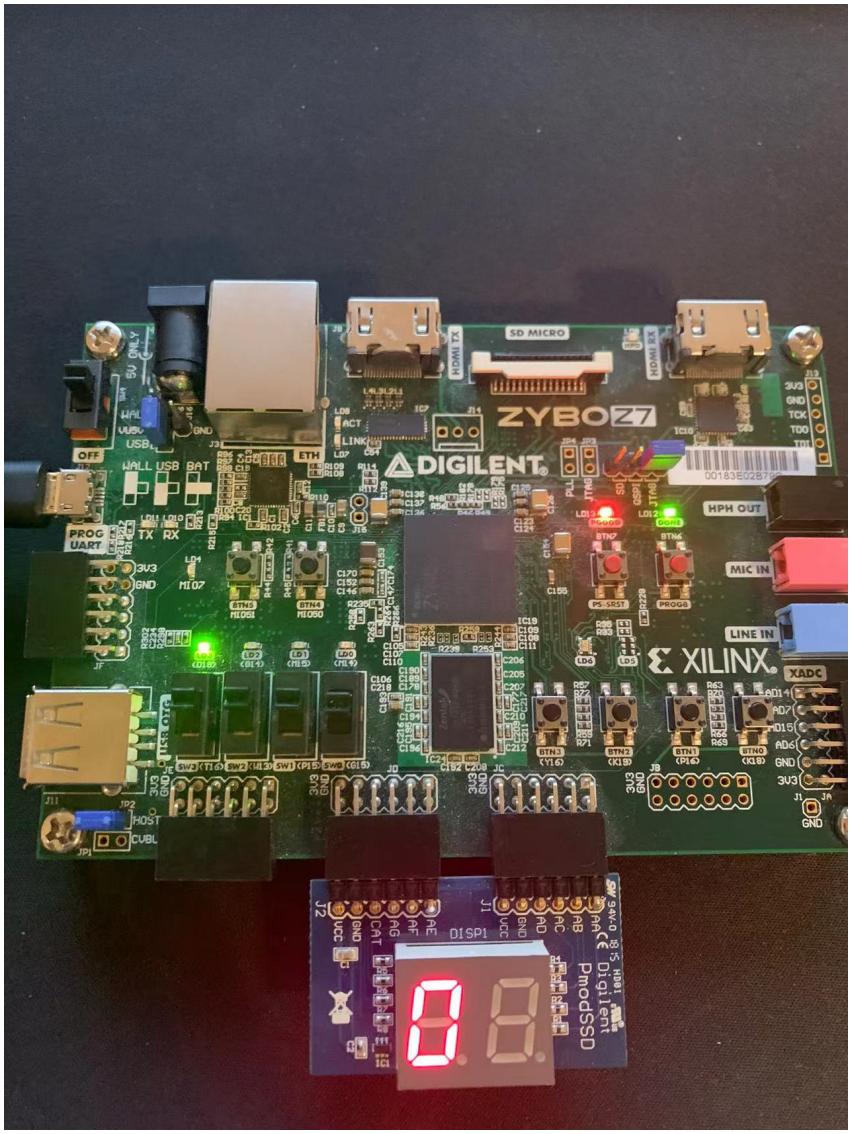

ZYBO Z7
DIGILENT
SD MICRO
HDMI TX
HDMI RX
HPH OUT
MIC IN
LINE IN
XADC
XILINX
PROG UART
ETH
USB
Digilent
PmodSSD
DISP1

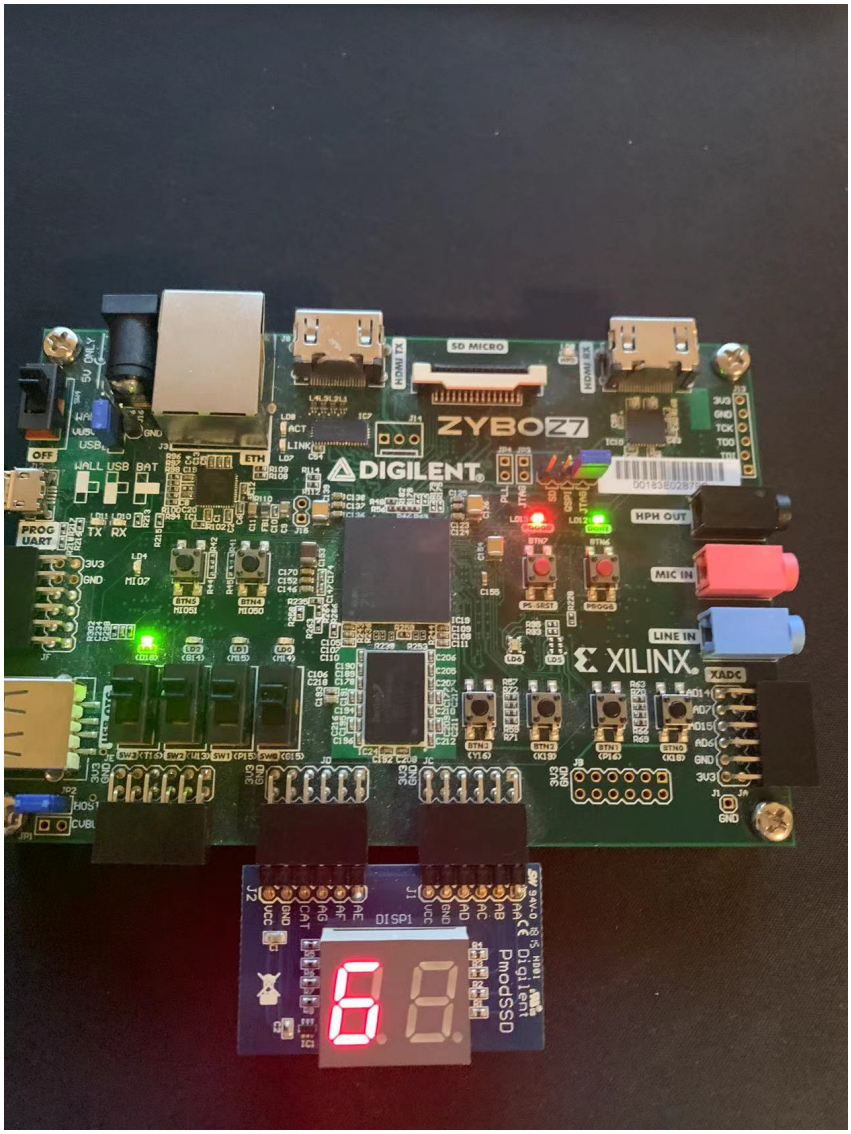

ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI TX
HPH OUT
MIC IN
LINE IN
XADC
PROG UART
ETH
DISP1
Digilent
PmodSSD


ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
HPH OUT
MIC IN
LINE IN
XADC
PROG UART
ETH
DISP1
PmodSSD

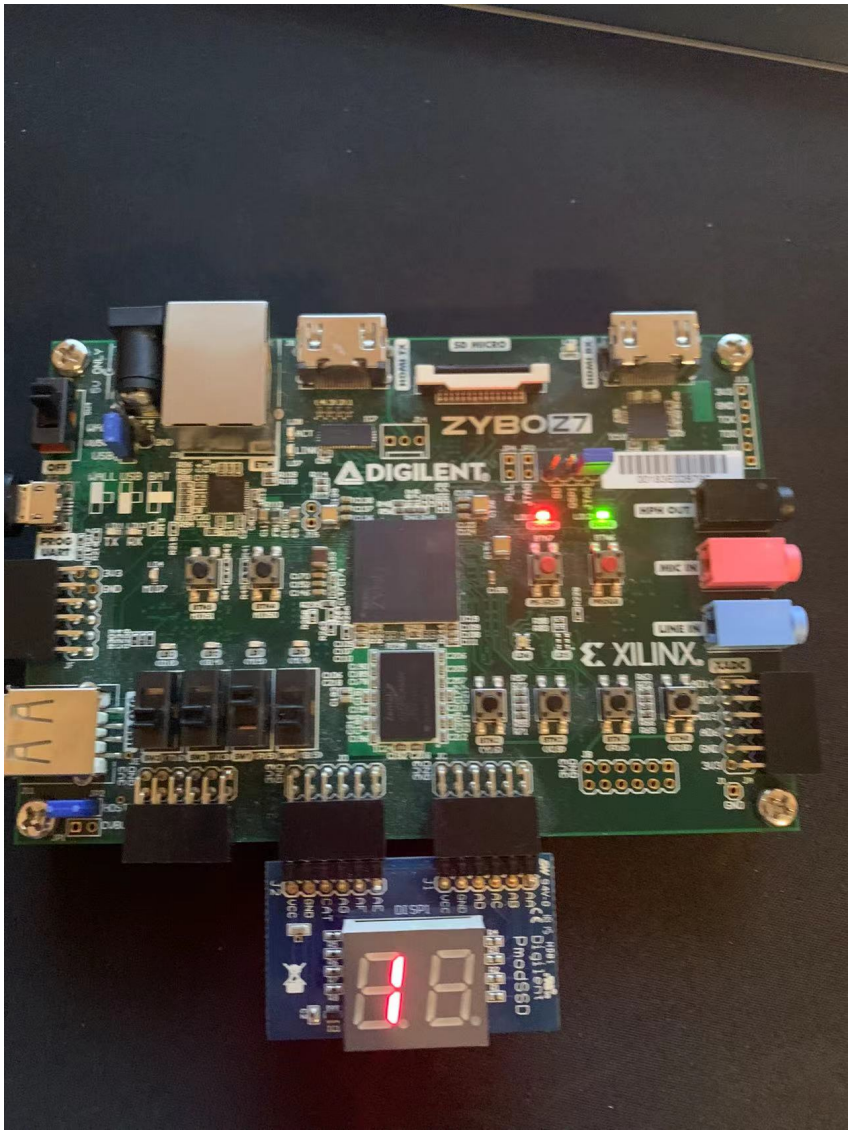

ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
HPH OUT
MIC IN
LINE IN
XADC
PROG UART
ETH
OFF
WALL USB BAT
Digilent
PmodSSD
DISP1

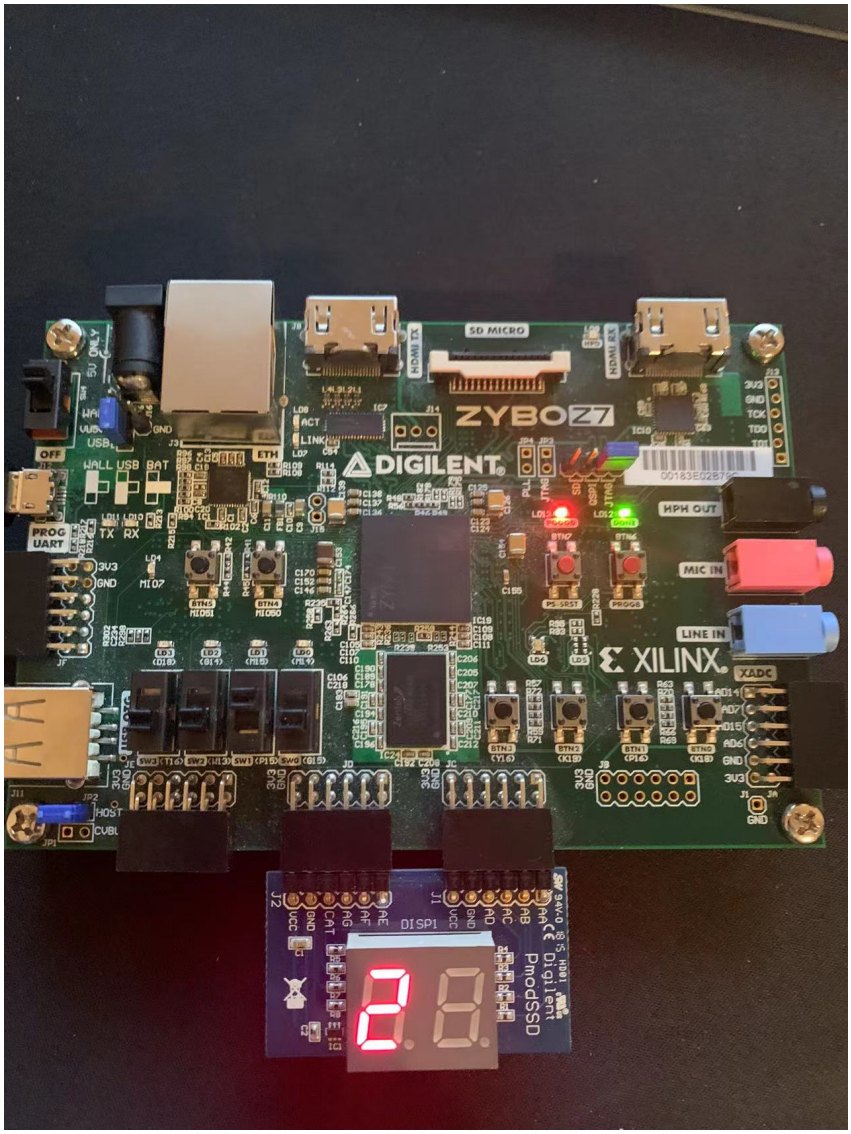

ZYBO Z7
DIGILENT
XILINX
HPH OUT
MIC IN
LINE IN
XADC
DISP1
Digilent
PmodSSD

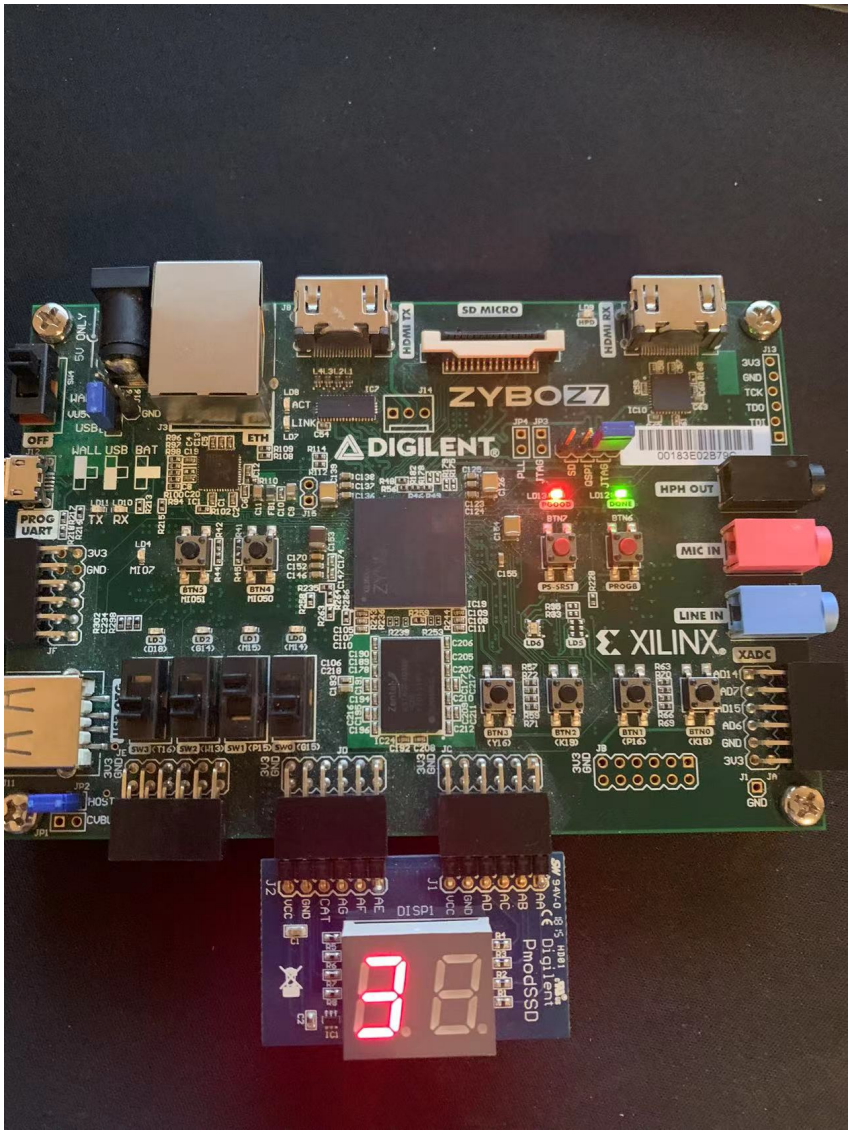

ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
ETH
HPH OUT
MIC IN
LINE IN
XADC
PROG UART
WALL USB BAT
OFF
Digilent PmodSSD
DISP1

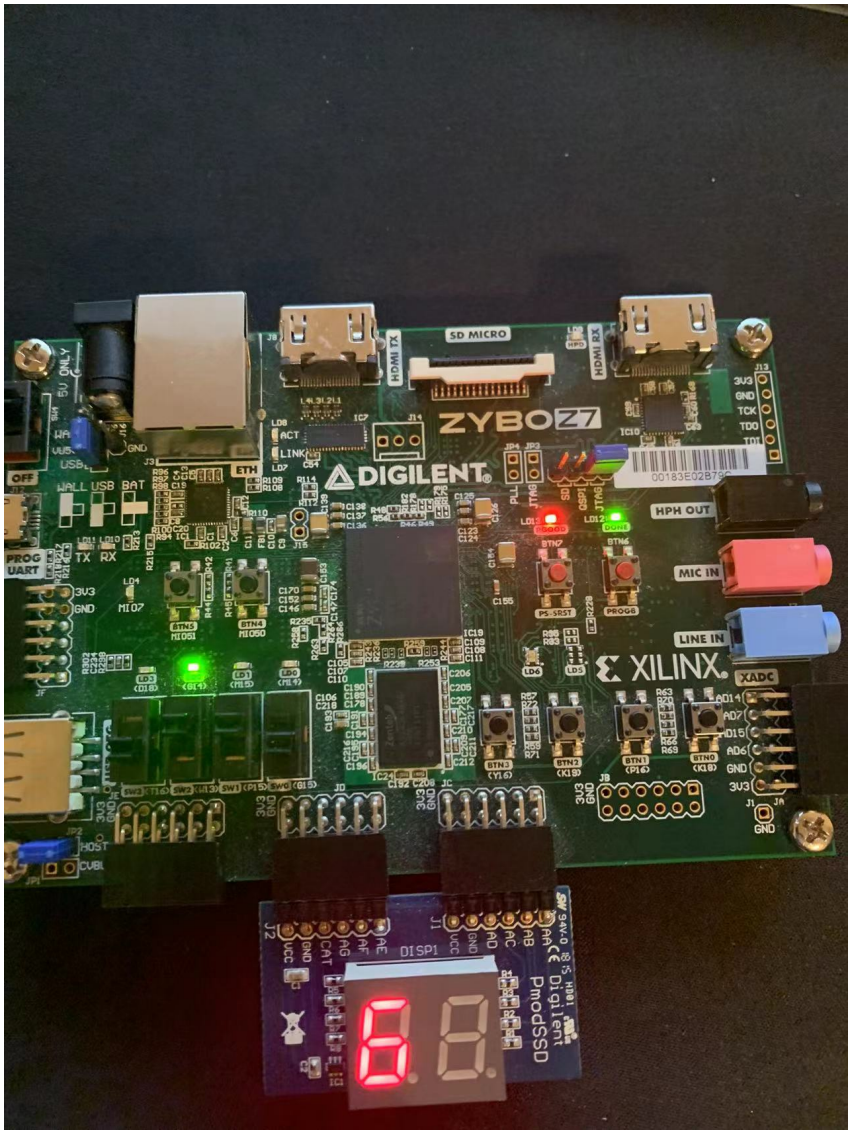


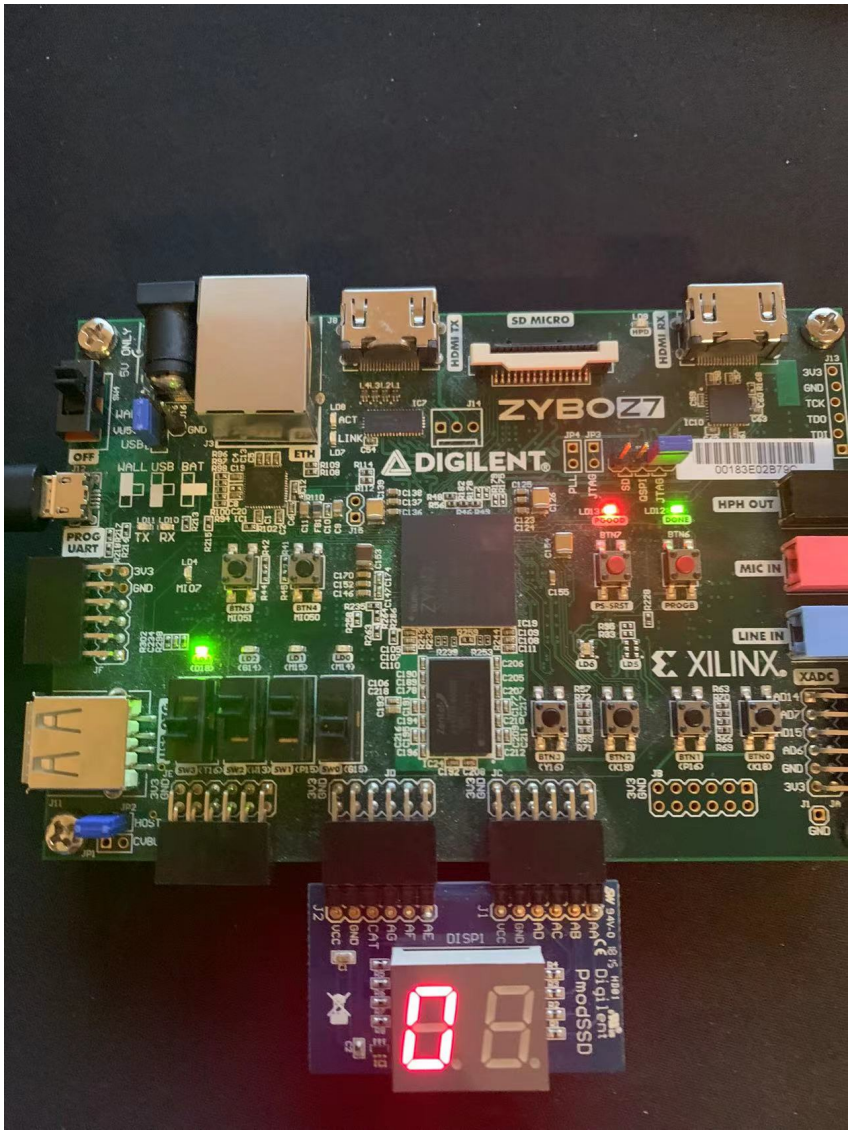
Chocolate selection

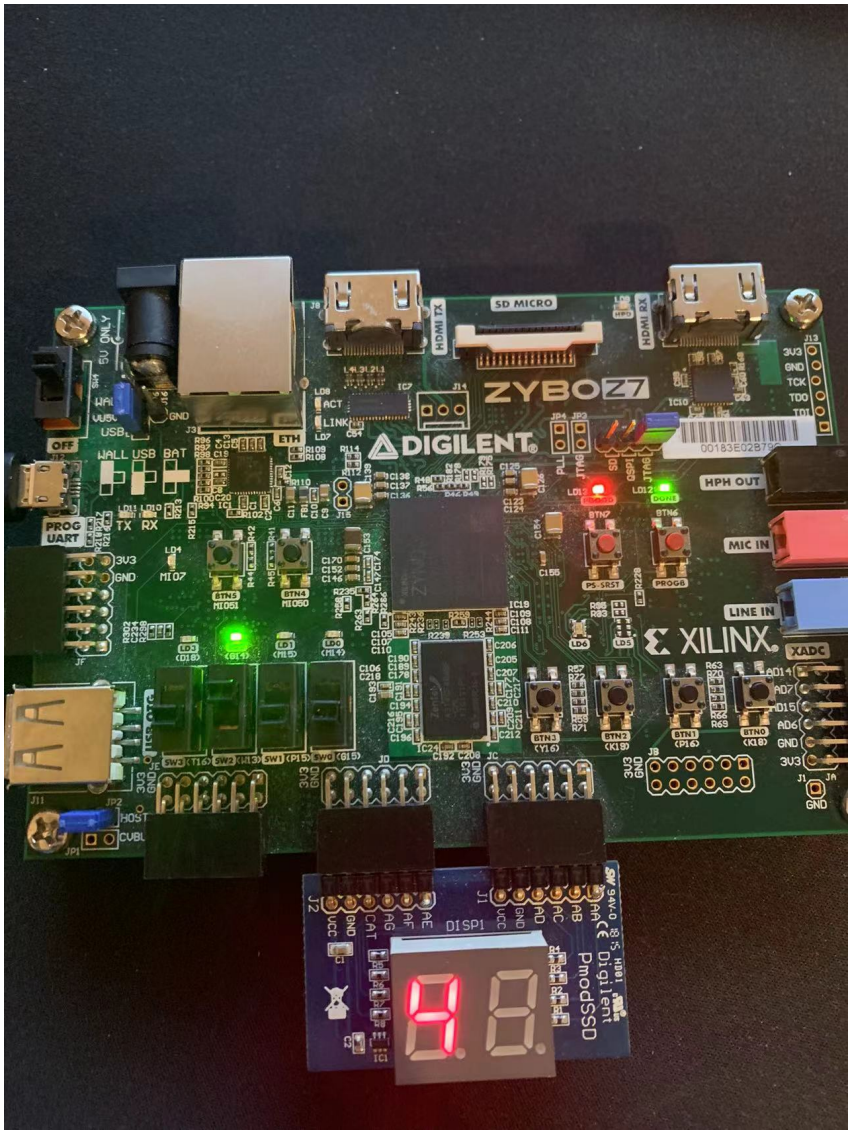

ZYBO Z7
DIGILENT
XILINX
Pmod SSD
Digilent

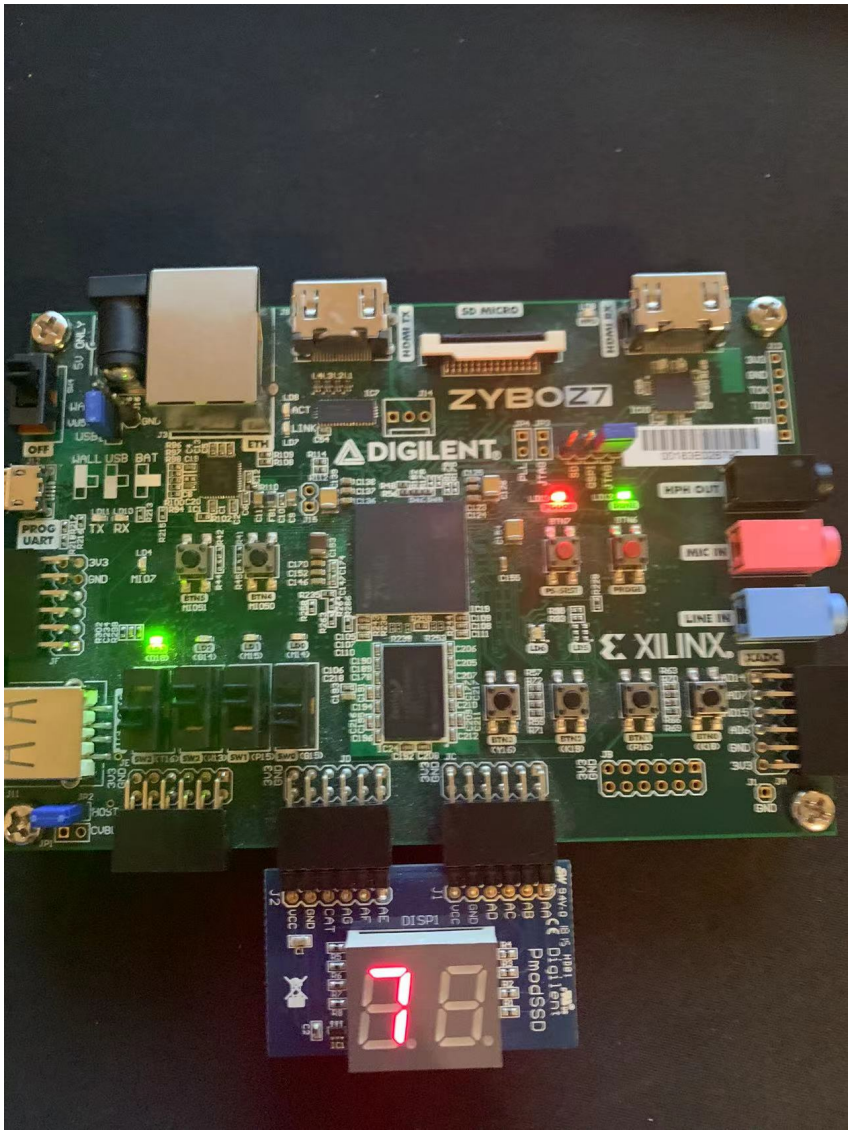

ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
HPH OUT
MIC IN
LINE IN
XADC
PROG UART
ETH
DISP1
Digilent
PmodSSD

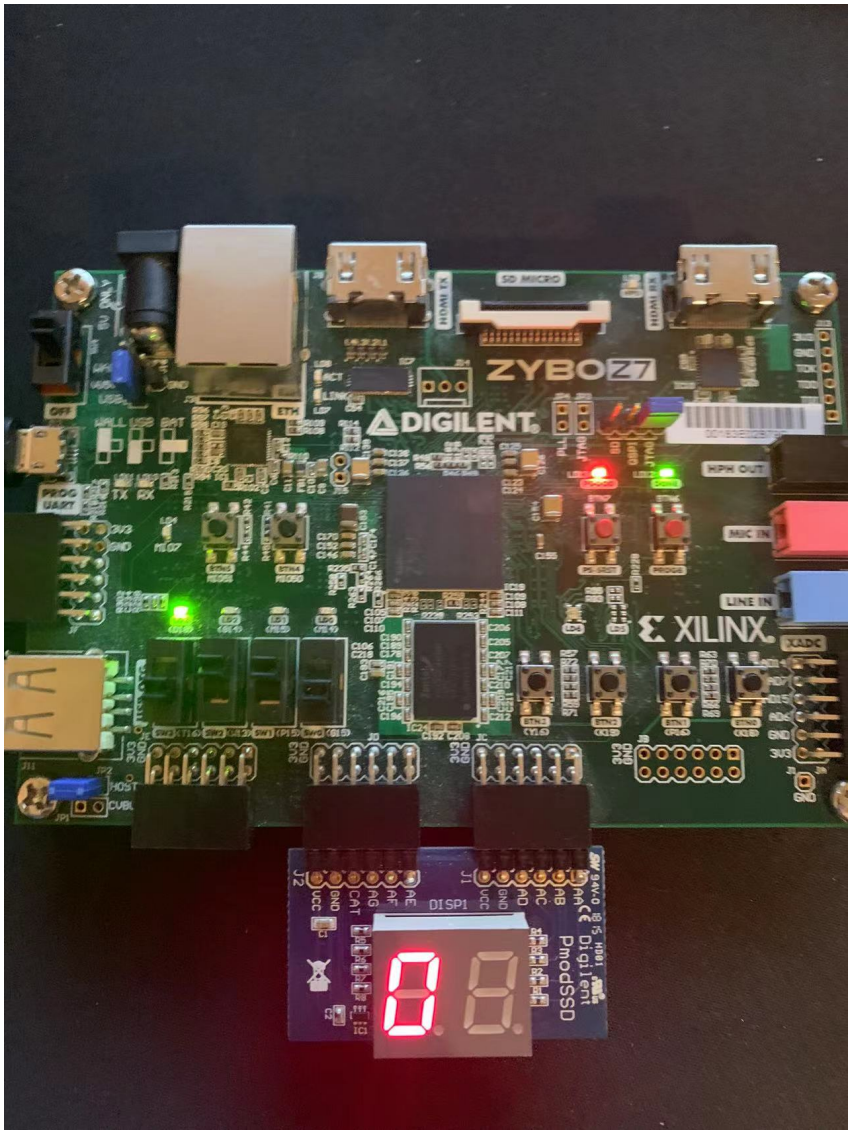

ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
ETH
HPH OUT
MIC IN
LINE IN
XADC
PROG UART
Digilent
PmodSSD
DISP1


ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
HPH OUT
MIC IN
LINE IN
XADC
ETH
USB
PROG UART
DISP1
Digilent
PmodSSD


ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
HPH OUT
MIC IN
LINE IN
XADC
PROG UART
ETH
DISP1
PmodSSD
Digilent


ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
ETH
HPH OUT
MIC IN
LINE IN
XADC
PROG UART
WALL USB BAT
DISP1
PmodSSD
Digilent


ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
HPH OUT
MIC IN
LINE IN
PROG UART
WALL USB BAT
OFF
DISP1
PmodSSD
Digilent


ZYBO Z7
DIGILENT
XILINX
SD MICRO
HDMI TX
HDMI RX
HPH OUT
MIC IN
LINE IN
XADC
PROG UART
Digilent
PmodSSD
DISP1

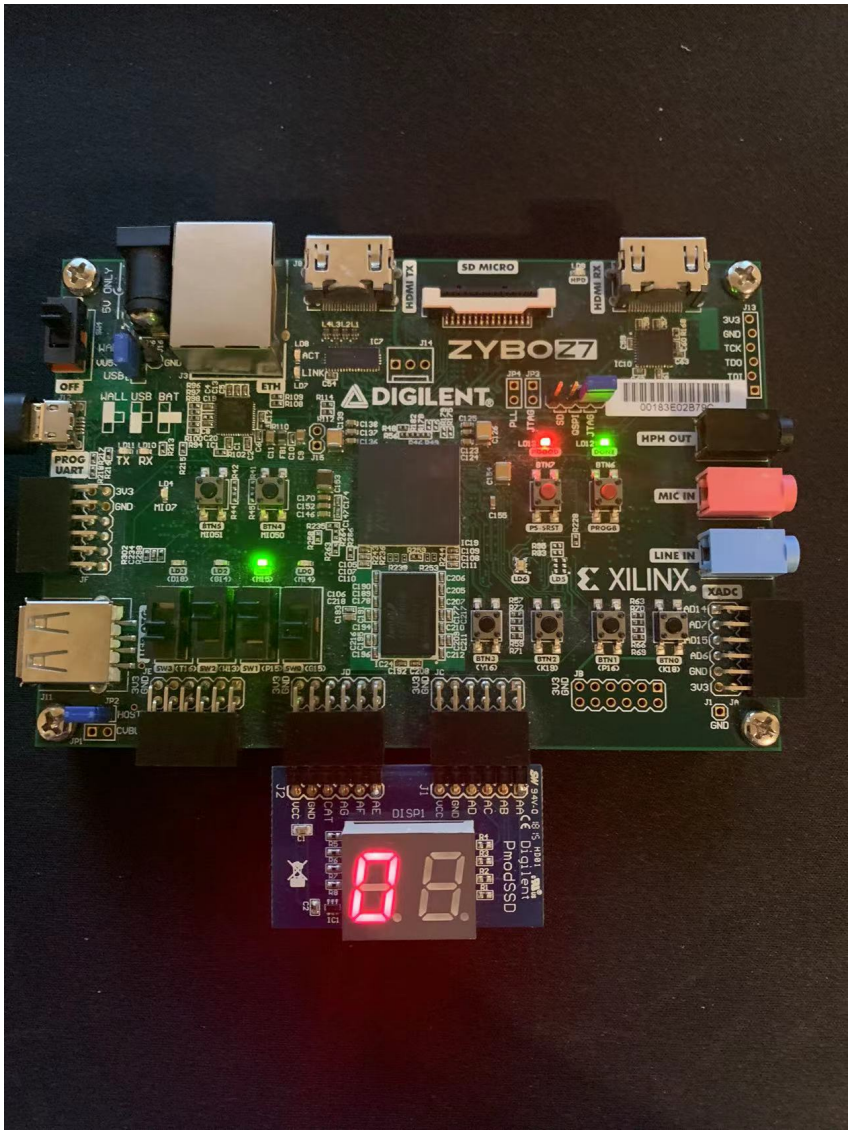


Figure2.3: Photograph of the Working Board

Conclusion

In this lab, a sequence detector and vending machine are designed, simulated and tested on Zybo Z7-10 board. This experiment can strengthen the ability to design finite state machine by using VHDL.