ECE 511 Digital ASIC Design

LAB

Lab 2: Finite State Machine (FSMs) and Vending Machine Using VHDL

Name: Xinyue Chen

Student ID: 1664206

Lab Date: Nov, 02, 2020

Abstract:

The objective of Lab 2 is to design a sequence detector and a vending machine using FSM in VHDL. The first step is to draw a state transition diagram, then we can use Mealy or Moore Machine to implement the function of the FSM.

Learning objectives:

1. Introduction to VHDL behavioral-level VHDL design using a serial input sequence detector as an example. This example is intended to prepare the students to work with FSM for the Vending Machine.
2. Learn to create the FSM and write the sequential VHDL code for the FSM.
3. Using the seven-segment display for displaying the output (only for Part – 2).

Pre-Lab:

1. Carefully read this document and follow the detailed instructions.
2. Draw a state diagram of a “01101” sequence detector using a Mealy or Moore FSM.

   The state transition diagram of “01101” sequence detector is shown in Part 1.

PART 1: Sequence detector – “01101” (overlapping)

The function of a “01101” sequence detector is to detect the occurrence of the sequence “01101” in input sequence. When “01101” appears, the output turns to ‘1’ to indicate the sequence “01101” detected.
If the sequence detector allows overlapping, here is an example:

Input: 00010110110110100011
Output: 00000000100100100000

According to the function of “01101” sequence detector with overlapping, we use a Mealy FSM to design the sequence detector. The output of Mealy FSM is directly controlled by the input and the current state of the machine, it has faster output response compare with Moore FSM, the basic structure of Mealy FSM and Moore FSM is shown in **Figure 1-1** and **Figure 1-2**.

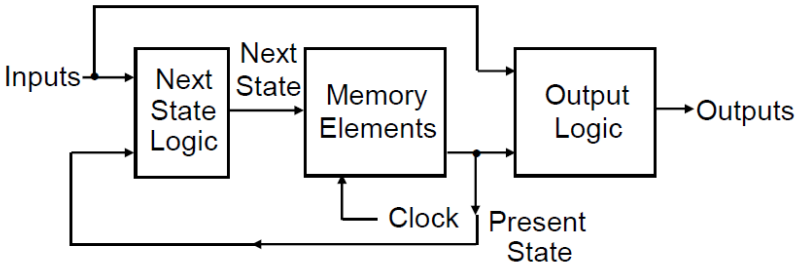

Figure 1-1. Mealy FSM diagram.

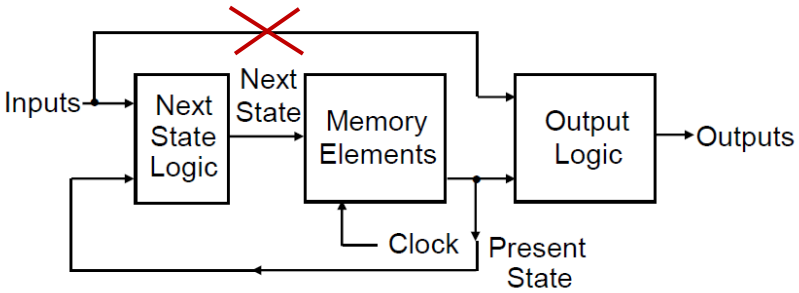

Figure 1-2. Moore FSM diagram.

Notice that we need to include a reset signal, when reset = '1', the FSM is in the initial/idle state. We first draw a truth table includes current state and next state.

reset	input	current state	next state	output
1	-	-	idle	0
0	0	idle	A	0
0	1	idle	idle	0
0	0	A	A	0
0	1	A	B	0
0	0	B	A	0
0	1	B	C	0
0	0	C	D	0
0	1	C	idle	0
0	0	D	A	0
0	1	D	E	1
0	0	E	A	0
0	1	E	C	0

Table 1. Truth table of sequence detector (including current state and next state).

The meanings of different states are indicated.

idle: Initial state (waiting for detecting the next input bit/reset state).
A: Sequence '0' detected.

B: Sequence “01” detected.
C: Sequence “011” detected.
D: Sequence “0110” detected.
E: Sequence “01101” detected.

The state transition diagram of “01101” sequence detector is shown in **Figure 2**. The output is affected by input, so when it transits from current state to next state, input/output bit is indicated in the transition line together.

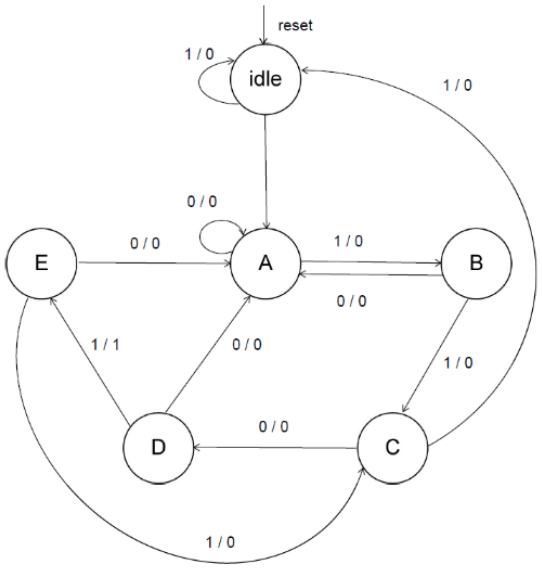

Figure 2. State transition diagram of “01101” sequence detector (Mealy FSM with overlapping)

If the sequence detector does not allow overlapping, here is an example:
Input: 00010110110110100011
Output: 00000000100000100000

We draw a state transition diagram of “01101” sequence detector without overlapping using Moore FSM in **Figure 3**.

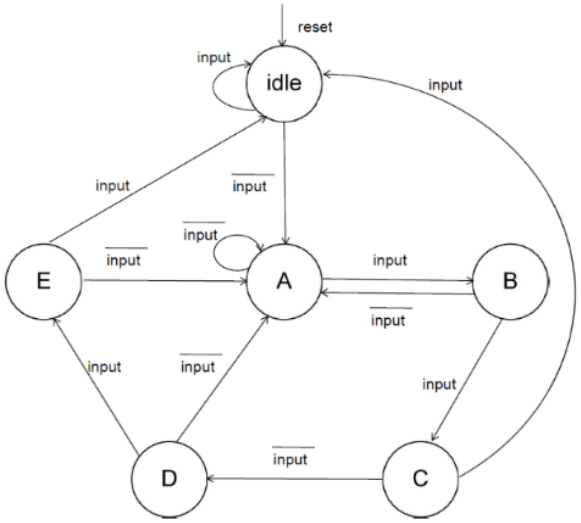

Figure 3. State transition diagram of “01101” sequence detector (Moore FSM without overlapping)

The differences of **Figure 2** and **Figure 3** are: (1) The output is not indicated in **Figure 3** since the output of Moore FSM is not directly controlled by input, while in **Figure 2**, the output bit is directly controlled by input bit. (2) In **Figure 2**, if current state is E and input = ‘1’, it transits to C since the machine allows overlapping, while in **Figure 3**, if current state is E and input = ‘1’, it transits to idle since the machine does not allow overlapping.

The VHDL code, testbench for simulation and constraint file in demonstrated in the Appendix. We use input sequence “11100101101101011001111” in test bench to test the result. The waveform of the simulation result (including current state and next state) is shown in **Figure 4**. The output sequence is “00000000001001000000000”. In conclusion, the result matches the function of “01101” sequence detector with overlapping.

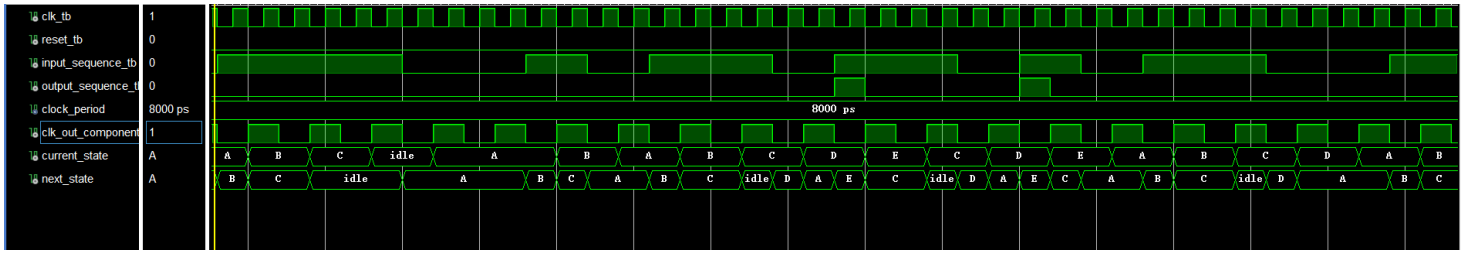

Figure 4. Waveform of sequence detector (with overlapping).

Since the system clock is 125,000,000 HZ, it is difficult to observe the implementation result on FPGA board, so a clock divider of 1 HZ is used. However, in simulation, we just rectify the count number to 1 (this will divide the system clock by 2) to get the waveform. The clk_out_component in **Figure 4** is the clock signal we use in simulation.

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

Figure 5. Part of the VHDL code of the clock divider.

The synthesis result is shown in **Figure 6**.

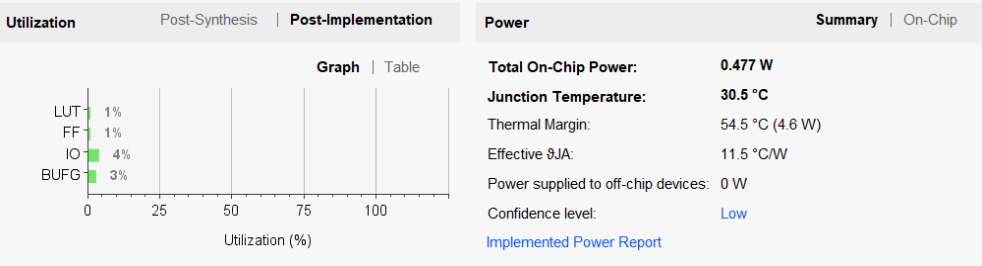

Figure 6. Synthesis result of sequence detector.

After synthesis, implementation, writing bit-stream and programming the FPGA, we use the zybo board to verify the result. We turn the input switch on and off, when the input sequence "01101" detected, the corresponding LED turns on to indicate "01101" detected. We successfully implement the function of the sequence detector.

PART 2: Vending Machine for Chocolate and Chips

In this part we design a vending machine sells chocolate and chips, here are some features about this machine:

1. Vending Machine accepts denominations of 1$, 2$ and 5$ only.
2. The price of chips is 4$ and that of chocolate is 3$.
3. Customer first select the product and then inserts the money.
4. The machine returns the change, if any (in the form of 1$, 2$ and 5$), to the customer and then dispenses the product.

The block diagram of this vending machine is shown in **Figure 7**. *Reset* button should reset the machine at any state, *coins_in* represents the denomination we inserts each time ("00" is 0$, "01" is 1$, "10" is 2$ and "11" is 5$), if *item_select* = '0' the customer wants chocolate, if *item_select* = '1' the customer wants chips, *change_out* represents the change returned to customer ("00" is 0$, "01" is 1$, "10" is 2$ and "11" is 5$).
If enough denominations are inserted, the machine will return the change to the customer and then dispense the product, when *chocolate* = '1' customer gets chocolate, when *chips* = '1' customer gets chips. The total sum of money inserted is shown on the seven_segment display.

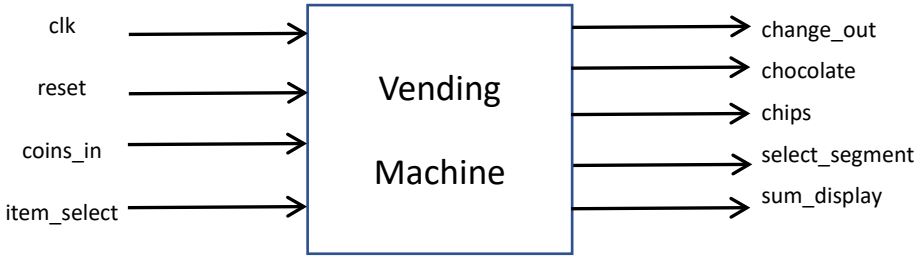

Figure 7. Block diagram of vending machine.

When design the vending machine, here are other features we need to consider:

1. Select whether you want chips or chocolate.
2. You know that chips are 4$ and chocolate is 3$. Insert the money in the vending machine. For example: Customer wants chocolate of 3$. The inserted money can be 1$, 1$ and 1$. It may be 1$ and 2$ or it can be 1$ followed by 5$. The sum of the money inserted must be visible on the seven-segment display. Pen down all the possible input combinations (for chips and chocolate) to get a clear idea.
3. However, once you have inserted the required money for the product, you cannot put any more money as it makes no sense. Take an example. Insert 1$ and then 5$. The inserted money is 6$ and you got the chocolate. Now, you do not insert any more money as it is meaningless.
4. Once, the required denominations are inserted, the Vending Machine will hand over the change if any and then it will dispense the product for you. Here, we took an example of 1$ followed by 5$. So, the machine returns the change of 3$ in the denominations of 2$ and 1$. Then it dispenses the chocolate

After considering the features mentioned, we write down all the combinations of input coins to get a better idea of the scheme. When chocolate is selected, the total sum of money could be:

1) *Sum_display* is 3$ (1$ → 2$ or 1$ → 1$ → 1$). *Change_out* is 0$.
2) *Sum_display* is 4$ (1$ → 1$ → 2$ or 2$ → 2$). *Change_out* is 1$.
3) *Sum_display* is 5$ (5$). *Change_out* is 2$.
4) *Sum_display* is 6$ (1$ → 5$). *Change_out* is 1$ + 2$.
5) *Sum_display* is 7$ (2$ → 5$ or 1$ → 1$ → 5$). *Change_out* is 2$ + 2$.

When chips are selected, the total sum of money could be:

1) *Sum_display* is 4$ (1$ → 1$ → 1$ → 1$ or 1$ → 1$ → 2$ or 2$ → 2$). *Change_out* is 0$.
2) *Sum_display* is 5$ (1$ → 1$ → 1$ → 2$ or 1$ → 2$ → 2$ or 5$). *Change_out* is 1$.
3) *Sum_display* is 6$ (1$ → 5$). *Change_out* is 2$.
4) *Sum_display* is 7$ (1$ → 1$ → 5$ or 2$ → 5$). *Change_out* is 1$ + 2$.
5) *Sum_display* is 8$ (1$ → 1$ → 1$ → 5$ or 1$ → 2$ → 5$). *Change_out* is 2$ + 2$.

According to all the combinations of coins, we design a Moore FSM. The truth table with current state and next state is in **Table 2**.

current state	next state				change	display sum	chips	chocolate
	coins in							
	00	01	10	11				
idle	item select = '0'		item select = '1'		0	0000000	0	0
	ready chocolate		ready chips					
ready chocolate	ready chocolate	A1	B1	change choc_2	0	0000000	0	0

A1	A1	B1	dispense choc	change choc_1_2	0	0000110 1$	0	0
B1	B1	dispense choc	change choc_1	change choc_2_2	0	1011011 2$	0	0
ready chip	ready chip	A2	B2	change chip_1	0	0000000	0	0
A2	A2	B2	D	change chip_2	0	0000110 1$	0	0
B2	B2	D	dispense chip	change chip_1_2	0	1011011 2$	0	0
D	D	dispense chip	change chip_1	change chip_2_2	0	1001111 3$	0	0
dispense choc	idle				0	1001111 3$	0	1
dispense chip	idle				0	1100110 4$	1	0
change choc_1	idle				01	1100110 4$	0	1
change choc_2	idle				10	1101101 5$	0	1
change chip_1	idle				01	1101101 5$	1	0
change chip_2	idle				10	1111101 6$	1	0
change choc_1_2	change choc_2				01	1111101 6$	0	0
change choc_2_2	change choc_2				10	0000111 7$	0	0
change chip_1_2	change chip_2				01	0000111 7$	0	0
change chip_2_2	change chip_2				10	1111111 8$	0	0

Table 2. Truth table of vending machine (including current state and next state).

The meanings of 18 different states are:
Idle: initial state/reset state.
Ready chocolate: chocolate is selected.
Ready chips: chips are selected.
A1: 1$ is inserted after ready chocolate.
B1: 2$ is inserted after ready chocolate.
A2: 1$ is inserted after ready chips.
B2: 2$ is inserted after ready chips.
D: 3$ is inserted after ready chips.
Dispense choc: 3$ is inserted and customer gets chocolate.
Dispense chip: 4$ is inserted and customer gets chips.
Change choc_1: 4$ is inserted and customer gets 1$ change and chocolate.
Change choc_2: 5$ is inserted and customer gets 2$ change and chocolate.
Change chip_1: 5$ is inserted and customer gets 1$ change and chips.
Change chip_2: 6$ is inserted and customer gets 2$ change and chips.
Change choc_1_2: 6$ is inserted and customer first gets 1$ change before getting chocolate.
Change choc_2_2: 7$ is inserted and customer first gets 2$ change before getting chocolate.
Change chip_1_2: 7$ is inserted and customer gets 1$ change before getting chips.

Change chip_2_2: 8$ is inserted and customer gets 2$ change before getting chips.

Notice that the four states *change choc_1_2, change choc_2_2, change chip_1_2* and *change chip_2_2* are middle states where the first change is returned (if we need to return two changes). When the first change is returned, it transits to the corresponding stages where the machine returns the second change and the customer gets the product. However, when the first change is returned, the total sum of money changes since we return a denomination to the customer. The sum_display is a 7-bit output connected to the seven segment display.

The state transition diagram is shown in **Figure 8**. There are some states (C1, C2 and E) redundant so I replace them with states that already exist in the diagram. Some lines are not drawn because the same state has the same transition line, it is easier to read if we have less transition lines.

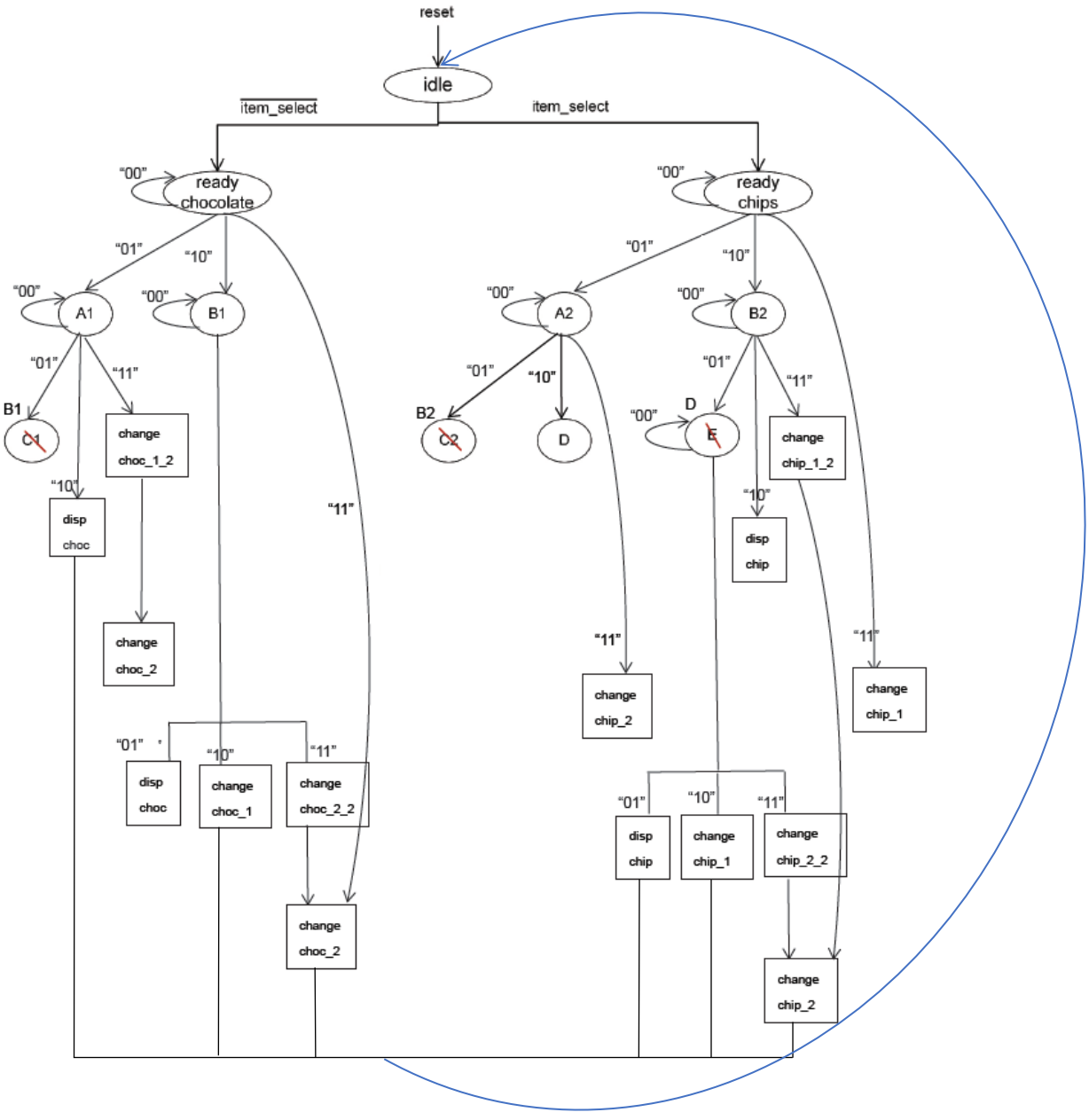

Figure 8. State transition diagram of vending machine.

The same clock divider in PART 1 is used to view the simulation result (see clock_out component in simulation waveform). As we mentioned, it is hard to see the programming result on zybo board since the system clock is 125,000,000 HZ, so we employ a cokck divider.

In the simualtion process, we wirte proper number of test cases to verify the function. In **Figure 9**, we make reset ='1' at certain time, and the result shows that it is able to reset the machine. We select chocolate, insert 1$ and 2$, it dispenses chocolate. Then we insert 2$ and 2$, the machine returns 1$ and dispenses chocolate. Then we insert 5$, the machine returns 2$ and dispenses chocolate. We select chips, insert 2$ and 5$, the machine returns 1$ and 2$ then dispenses chips. Then we insert 1$, 2$ and 5$, the machine returns 2$ and 2$ then dispenses chips. The simulation waveform indicates that the machine matches the features mentioned previously.

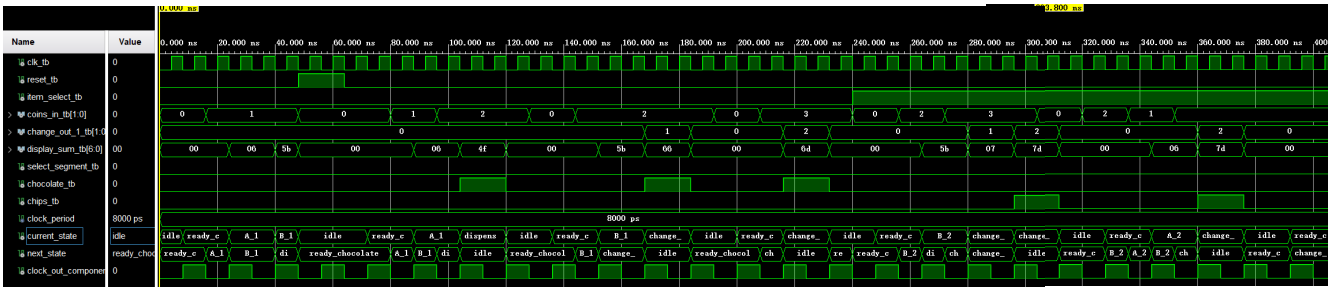

Figure 9. Waveform of vending machine.

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

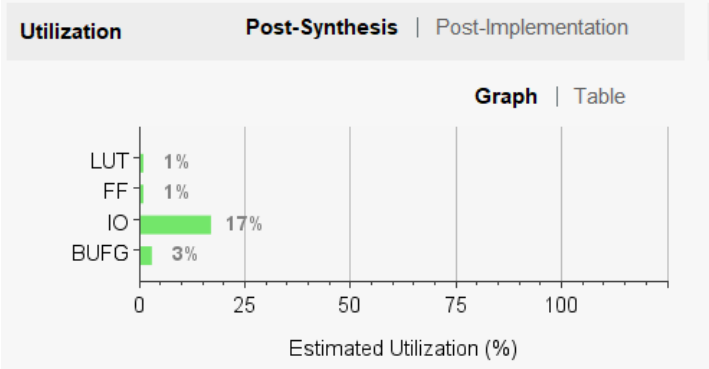

Figure 10. Synthesis result of vending machine.

After synthesis, implementation, writing bit-stream and programming the FPGA, we use the Zybo board to verify the result. When select_item = '1', coins_in = "10", then coins_in = "11", led of change_out = "01" and "10", chips = '1', the customer gets 1$+2$ changes and then chips. We successfully implement the function of the vending machine.

Appendix:

PART 1:

VHDL code of the clock divider of 1 HZ:

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/11/09 19:57:43
-- Design Name: “01101” sequence detector
-- Module Name: clock_divider - Behavioral
-- Project Name: “01101” sequence detector
-- Target Devices: ZYBO Z7-10
-- Tool Versions:
-- Description:
----------------------------------------------------------------------------------
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;
```

VHDL code of “01101” sequence detector:

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/11/08 11:34:32
-- Design Name: “01101” sequence detector
-- Module Name: sequence_detector - Behavioral
-- Project Name: “01101” sequence detector
-- Target Devices: ZYBO Z7-10
-- Tool Versions:
-- Description:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sequence_detector is
    Port ( input_sequence : in STD_LOGIC;
           clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           output_sequence : out STD_LOGIC);
end sequence_detector;

architecture finite_state_machine of sequence_detector is
component clk_divider is
  Port ( clk_in : in std_logic;
         clk_out : out std_logic);
end component clk_divider;

type state_type is (idle,A,B,C,D,E);
-- idle is the initial state
-- A: '0' detected, B: "01" detected, C: "011" detected, D: "0110" detected, E: "01101" detected.
signal clk_out_component : std_logic;
signal current_state, next_state : state_type;
begin
label_clock_divider: component clk_divider port map(clk_in => clk,
                                                    clk_out => clk_out_component);
sequ_logic: process (clk_out_component,reset) is
begin
if (reset ='1') then
current_state <= idle;
elsif (clk_out_component'event and clk_out_component='1') then
current_state <= next_state;
end if;
end process sequ_logic;
```

```
comb_logic: process (current_state,input_sequence) is
begin
output_sequence <= '0';
case current_state is
when idle =>
if (input_sequence = '0') then
next_state <= A;
output_sequence <= '0';
else
next_state <= idle;
output_sequence <= '0';
end if;

when A =>
if (input_sequence = '1') then
next_state <= B;
output_sequence <= '0';
else
next_state <= A;
output_sequence <= '0';
end if;

when B =>
if (input_sequence = '1') then
next_state <= C;
output_sequence <= '0';
else
next_state <= A;
output_sequence <= '0';
end if;

when C =>
if (input_sequence = '0') then
next_state <= D;
output_sequence <= '0';
else
next_state <= idle;
output_sequence <= '0';
end if;

when D =>
if (input_sequence = '1') then
next_state <= E;
output_sequence <= '1';
```

```
else
next_state <= A;
output_sequence <= '0';
end if;

when E =>
if (input_sequence = '1') then
next_state <= C;
output_sequence <= '0';
else
next_state <= A;
output_sequence <= '0';
end if;
end process comb_logic;
end finite_state_machine;
```

VHDL testbench of “01101” sequence detector:

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/11/09 20:18:59
-- Design Name: “01101” sequence detector
-- Module Name: sequence_detector_tb - Behavioral
-- Project Name: “01101” sequence detector
-- Target Devices: ZYBO Z7-10
-- Tool Versions:
-- Description:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sequence_detector_tb is
end sequence_detector_tb;

architecture Behavioral of sequence_detector_tb is
component sequence_detector is
      port ( input_sequence : in std_logic;
            clk : in std_logic;
            reset : in std_logic;
            output_sequence : out std_logic);
end component sequence_detector;
    signal clk_tb : std_logic := '0';
    signal reset_tb : std_logic := '0';
    signal input_sequence_tb : std_logic := '0';
```

```
    signal output_sequence_tb : std_logic;
    constant clock_period : time := 8 ns;
begin
component_sequ: component sequence_detector port map( input_sequence => input_sequence_tb,
                                                      clk => clk_tb,
                                                      reset => reset_tb,
                                                      output_sequence => output_sequence_tb);
----Clock process
clock: process
begin
        clk_tb <='0';
        wait for clock_period/2;
        clk_tb <='1';
        wait for clock_period/2;
end process clock;

stim_proc: process
   begin
   reset_tb <= '1';
   wait for 16 ns;
   reset_tb <= '0';
   wait for 16 ns;
   input_sequence_tb <= '1';
   wait for 16 ns;
   input_sequence_tb <= '1';
   wait for 16 ns;
   input_sequence_tb <= '1';
   wait for 16 ns;
   input_sequence_tb <= '0';
   wait for 16 ns;
   input_sequence_tb <= '0';
   wait for 16 ns;
   input_sequence_tb <= '1';
   wait for 16 ns;
   input_sequence_tb <= '0';
   wait for 16 ns;
   input_sequence_tb <= '1';
   wait for 16 ns;
   input_sequence_tb <= '1';
   wait for 16 ns;
   input_sequence_tb <= '0';
   wait for 16 ns;
   input_sequence_tb <= '1';
   wait for 16 ns;
```

```
    input_sequence_tb <= '1';
    wait for 16 ns;
    input_sequence_tb <= '0';
    wait for 16 ns;
    input_sequence_tb <= '1';
    wait for 16 ns;
    input_sequence_tb <= '0';
    wait for 16 ns;
    input_sequence_tb <= '1';
    wait for 16 ns;
    input_sequence_tb <= '1';
    wait for 16 ns;
    input_sequence_tb <= '0';
    wait for 16 ns;
    input_sequence_tb <= '0';
    wait for 16 ns;
    input_sequence_tb <= '1';
    wait for 16 ns;
    input_sequence_tb <= '1';
    wait for 16 ns;
    input_sequence_tb <= '1';
       wait;
     end process;
end Behavioral;
```

Constraint file of “01101” sequence detector:

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

```
##Clock signal
set_property -dict {PACKAGE_PIN K17      IOSTANDARD LVCMOS33} [get_ports { clk }];
#create_clock -period 8.000 -name sys_clk_pin -waveform {0.000 4.000} -add [get_ports { }];
##Switches
set_property -dict {PACKAGE_PIN G15      IOSTANDARD LVCMOS33} [ get_ports { input_sequence }];
#Sch=sw[0]
##Buttons
set_property -dict { PACKAGE_PIN K18      IOSTANDARD LVCMOS33 } [get_ports { reset }];
#IO_L12N_T1_MRCC_35 Sch=btn[0]
##LEDs
set_property -dict {PACKAGE_PIN M14       IOSTANDARD LVCMOS33} [get_ports { output_sequence }];
```

PART 2:

VHDL code of Vending Machine (we use the same clock divider as demonstrated in PART 1 to view the result on FPGA board):

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/11/08 16:28:53
-- Design Name: vending machine for chocolate and chips
-- Module Name: vending_machine - Behavioral
-- Project Name: vending machine for chocolate and chips
-- Target Devices: ZYBO Z7-10
-- Tool Versions:
-- Description:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity vending_machine is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           item_select : in STD_LOGIC;
           coins_in : in STD_LOGIC_VECTOR (1 downto 0);
           change_out: out STD_LOGIC_VECTOR (1 downto 0);
-- To display the output change ("01" is 1$,"10" is 2$ and "11" is 5$)
           display_sum : out STD_LOGIC_VECTOR (6 downto 0);
           select_segment : out STD_LOGIC;
           chips : out STD_LOGIC;
           chocolate : out STD_LOGIC);
end vending_machine;

architecture finite_state_machine of vending_machine is
component clock_divider is
  Port ( clk_in : in std_logic;
           clk_out : out std_logic);
end component clock_divider;

type state_type is
(idle,ready_chocolate,ready_chips,A_1,B_1,A_2,B_2,D,dispense_choc,dispense_chip,change_choc_1,change_c
hoc_2,change_choc_1_2,change_choc_2_2,change_chip_1,change_chip_2,change_chip_1_2,change_chip_2_2
);
signal current_state,next_state : state_type;
signal clock_out_component : std_logic;

begin
label_clock_divider: component clock_divider port map(clk_in => clk,
                                                      clk_out => clock_out_component);
```

```
sequ_logic: process (clock_out_component,reset) is
begin
if (reset='1') then
current_state <= idle;
elsif (clock_out_component'event and clock_out_component='1') then
current_state <= next_state;
end if;
end process sequ_logic;

comb_logic: process (current_state,item_select,coins_in) is
begin
case current_state is
when idle =>
                change_out <= "00";
                display_sum <= "0000000";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '0';
if (item_select='0') then
next_state <= ready_chocolate;
else
next_state <= ready_chips;
end if;

when ready_chocolate =>
                change_out <= "00";
                display_sum <= "0000000";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '0';
if (coins_in="01") then
next_state <= A_1;
elsif (coins_in="10") then
next_state <= B_1;
elsif (coins_in="11") then
next_state <= change_choc_2;
elsif (coins_in="00") then
next_state <= ready_chocolate;
end if;

when A_1 =>
                change_out <= "00";
                display_sum <= "0000110";
                select_segment <= '0';
```

```
                chips <= '0';
                chocolate <= '0';
if (coins_in="01") then
next_state <= B_1;
elsif (coins_in="10") then
next_state <= dispense_choc;
elsif (coins_in="11") then
next_state <= change_choc_1_2;
elsif (coins_in="00") then
next_state <= A_1;
end if;

when B_1 =>
                change_out <= "00";
                display_sum <= "1011011";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '0';
if (coins_in="01") then
next_state <= dispense_choc;
elsif(coins_in="10") then
next_state <= change_choc_1;
elsif (coins_in="11") then
next_state <= change_choc_2_2;
elsif (coins_in="00") then
next_state <= B_1;
end if;

when ready_chips =>
                change_out <= "00";
                display_sum <= "0000000";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '0';
if (coins_in="01") then
next_state <= A_2;
elsif (coins_in="10") then
next_state <= B_2;
elsif (coins_in="11") then
next_state <= change_chip_1;
elsif (coins_in="00") then
next_state <= ready_chips;
end if;
```

```
when A_2 =>
                change_out <= "00";
                display_sum <= "0000110";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '0';
if (coins_in="01") then
next_state <= B_2;
elsif (coins_in="10") then
next_state <= D;
elsif (coins_in="11") then
next_state <= change_chip_2;
elsif (coins_in="00") then
next_state <= A_2;
end if;

when B_2 =>
                change_out <= "00";
                display_sum <= "1011011";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '0';
if (coins_in="01") then
next_state <= D;
elsif (coins_in="10") then
next_state <= dispense_chip;
elsif (coins_in="11") then
next_state <= change_chip_1_2;
elsif (coins_in="00") then
next_state <= B_2;
end if;

when D =>
                change_out <= "00";
                display_sum <= "1001111";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '0';
if (coins_in="01") then
next_state <= dispense_chip;
elsif (coins_in="10") then
next_state <= change_chip_1;
elsif (coins_in="11") then
next_state <= change_chip_2_2;
```

```
elsif (coins_in="00") then
next_state <= D;
end if;

when dispense_choc =>
                change_out <= "00";
                display_sum <= "1001111";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '1';
next_state <= idle;

when dispense_chip =>
                change_out <= "00";
                display_sum <= "1100110";
                select_segment <= '0';
                chips <= '1';
                chocolate <= '0';
next_state <= idle;

when change_choc_1 =>
                change_out <= "01";
                display_sum <= "1100110";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '1';
next_state <= idle;

when change_chip_1 =>
                change_out <= "01";
                display_sum <= "1101101";
                select_segment <= '0';
                chips <= '1';
                chocolate <= '0';
next_state <= idle;

when change_choc_2 =>
                change_out <= "10";
                display_sum <= "1101101";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '1';
next_state <= idle;
```

```
when change_chip_2 =>
                change_out <= "10";
                display_sum <= "1111101";
                select_segment <= '0';
                chips <= '1';
                chocolate <= '0';
next_state <= idle;

when change_choc_1_2 =>
                change_out <= "01";
                display_sum <= "1111101";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '0';
next_state <= change_choc_2;

when change_chip_1_2 =>
                change_out <= "01";
                display_sum <= "0000111";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '0';
next_state <= change_chip_2;

when change_choc_2_2 =>
                change_out <= "10";
                display_sum <= "0000111";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '0';
next_state <= change_choc_2;

when change_chip_2_2 =>
                change_out <= "10";
                display_sum <= "1111111";
                select_segment <= '0';
                chips <= '0';
                chocolate <= '0';
next_state <= change_chip_2;
end case;
end process comb_logic;
end finite_state_machine;
```

VHDL testbench of the Vending Machine:

```
----------------------------------------------------------------------------------
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/11/12 11:05:03
-- Design Name: vending machine for chocolate and chips
-- Module Name: vending_machine_tb - Behavioral
-- Project Name: vending machine for chocolate and chips
-- Target Devices: ZYBO Z7-10
-- Tool Versions:
-- Description:
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity vending_machine_tb is
end vending_machine_tb;

architecture Behavioral of vending_machine_tb is
component vending_machine is
Port( clk : in std_logic;
        reset : in std_logic;
        item_select : 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;
        chocolate : out std_logic);
end component vending_machine;
--input
    signal clk_tb : std_logic := '0';
    signal reset_tb : std_logic := '0';
    signal item_select_tb : std_logic := '0';
    signal coins_in_tb : std_logic_vector (1 downto 0) := "00";
--output
    signal change_out_tb : std_logic_vector (1 downto 0);
    signal display_sum_tb : std_logic_vector (6 downto 0);
    signal select_segment_tb : std_logic;
    signal chocolate_tb : std_logic;
    signal chips_tb : std_logic;
    constant clock_period : time := 8 ns;
begin
component_vend: component vending_machine port map( clk => clk_tb,
                                                    reset => reset_tb,
```

```
item_select =>    item_select_tb,
coins_in => coins_in_tb,
change_out => change_out_tb,
display_sum => display_sum_tb,
select_segment => select_segment_tb,
chocolate => chocolate_tb,
chips => chips_tb);

----Clock process
clk: process
        begin
            clk_tb <='0';
            wait for clock_period/2;
            clk_tb <='1';
            wait for clock_period/2;
end process clk;
stim_proc: process
    begin
   reset_tb <= '0';
   item_select_tb <= '0';
   coins_in_tb    <= "00";
   wait for 16 ns;
   coins_in_tb    <= "01";
   wait for 16 ns;
   coins_in_tb    <= "01";
   wait for 16 ns;
   reset_tb <= '1';
   coins_in_tb    <= "00";
   wait for 16 ns;
   reset_tb <= '0';
   item_select_tb <= '0';
   coins_in_tb    <= "00";
   wait for 16 ns;
   coins_in_tb    <= "01";
   wait for 16 ns;
   coins_in_tb    <= "10";
   wait for 32 ns;
   coins_in_tb    <= "00";
   wait for 16 ns;
   coins_in_tb    <= "10";
   wait for 16 ns;
   coins_in_tb    <= "10";
   wait for 32 ns;
   coins_in_tb    <= "00";
   wait for 16 ns;
```

```
    coins_in_tb    <= "11";
    wait for 32 ns;
    item_select_tb <= '1';
    coins_in_tb    <= "00";
    wait for 16 ns;
    coins_in_tb    <= "10";
    wait for 16 ns;
    coins_in_tb    <= "11";
    wait for 48 ns;
    coins_in_tb    <= "00";
    wait for 16 ns;
    coins_in_tb    <= "10";
    wait for 16 ns;
    coins_in_tb    <= "01";
    wait for 16 ns;
    coins_in_tb    <= "11";
    wait for 48 ns;
    wait;
    end process;
end Behavioral;
```

Constraint file of the Vending Machine:

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

```
##Clock signal
set_property -dict {PACKAGE_PIN K17       IOSTANDARD LVCMOS33} [get_ports { clk }];
##create_clock -period 8.000 -name sys_clk_pin -waveform {0.000 4.000} -add [get_ports { }];
##Switches
set_property -dict {PACKAGE_PIN G15       IOSTANDARD LVCMOS33} [ get_ports { item_select }];
#Sch=sw[0]
set_property -dict {PACKAGE_PIN P15       IOSTANDARD LVCMOS33} [ get_ports { coins_in[0] }];
#Sch=sw[1]
set_property -dict {PACKAGE_PIN W13       IOSTANDARD LVCMOS33 } [get_ports { coins_in[1] }];
#IO_L4N_T0_34 Sch=sw[2]
##Buttons
set_property -dict { PACKAGE_PIN K18      IOSTANDARD LVCMOS33 } [get_ports { reset }];
#IO_L12N_T1_MRCC_35 Sch=btn[0]

##LEDs
set_property -dict {PACKAGE_PIN M14       IOSTANDARD LVCMOS33} [get_ports { chips }];
set_property -dict {PACKAGE_PIN M15       IOSTANDARD LVCMOS33} [get_ports { chocolate }];
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] }];
##Pmod Header JC
set_property -dict { PACKAGE_PIN V15      IOSTANDARD LVCMOS33} [get_ports { display_sum[0] }];
```

#IO_L10P_T1_34 Sch=jc_p[1]
set_property -dict { PACKAGE_**PIN W15** IOSTANDARD LVCMOS33} [get_ports { **display_sum[1]** }];
#IO_L10N_T1_34 Sch=jc_n[1]
set_property -dict { PACKAGE_**PIN T11** IOSTANDARD LVCMOS33} [get_ports { **display_sum[2]** }];
#IO_L1P_T0_34 Sch=jc_p[2]
set_property -dict { PACKAGE_**PIN T10** IOSTANDARD LVCMOS33} [get_ports { **display_sum[3]** }];
#IO_L1N_T0_34 Sch=jc_n[2]
##Pmod Header JD
set_property -dict { PACKAGE_**PIN T14** IOSTANDARD LVCMOS33} [get_ports { **display_sum[4]** }];
#IO_L5P_T0_34 Sch=jd_p[1]
set_property -dict { PACKAGE_**PIN T15** IOSTANDARD LVCMOS33} [get_ports { **display_sum[5]** }];
#IO_L5N_T0_34 Sch=jd_n[1]
set_property -dict { PACKAGE_**PIN P14** IOSTANDARD LVCMOS33} [get_ports { **display_sum[6]** }];
#IO_L6P_T0_34 Sch=jd_p[2]
set_property -dict { PACKAGE_**PIN R14** IOSTANDARD LVCMOS33} [get_ports { **select_segment** }];
#IO_L6N_T0_VREF_34 Sch=jd_n[2]