VENDING MACHINE TECHNICAL DOCUMENTATION

Team H

15.06.23

1 Team members

Maxim Emile Speczyk

Muhammad Umer Bin Yaqoob

Stephanie Chinenye Okosa

2 Concept description

In this project, we aim to design a hardware system for a vending machine using Field-Programmable Gate Arrays (FPGA), as well as designing our own PCB to its specific needs.

The vending machine's main applications are to dispense an item, accept coins, as well as calculate and dispense the change. For this we use switches to select the item and coin input. Also, the 7-segment display shows the cost of an item, refund price and the insufficient amount.

Figure 1 is the state machine of the vending machine. In the state "Idle", the system waits for input of the item that is requested. After the item is requested, the money needs to be accepted, which happens in the state "Accepting". After enough money is detected, the item gets dispensed and the state changes to "Dispensing". Then the system changes to the state "Refund" which includes calculating and giving out the change back to the customer.

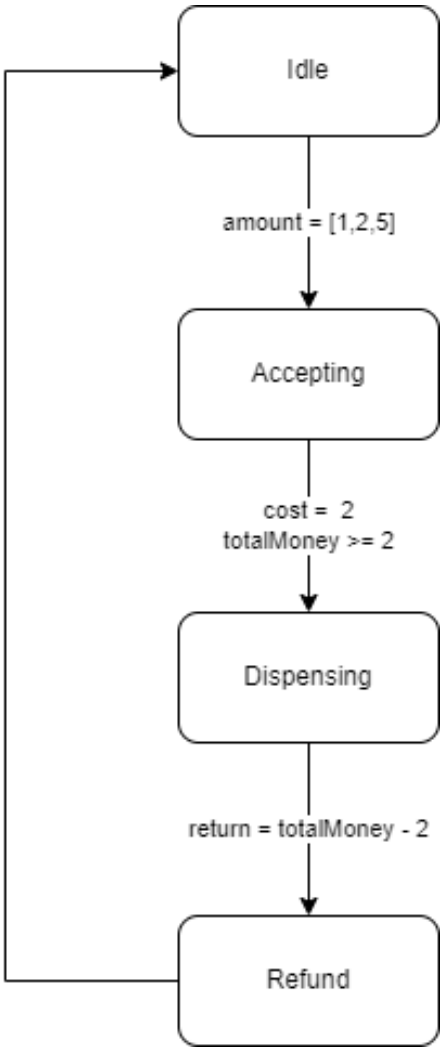

Figure 1: Vending Machine FSM

Below is the block diagram which illustrates the components of the vending machine.

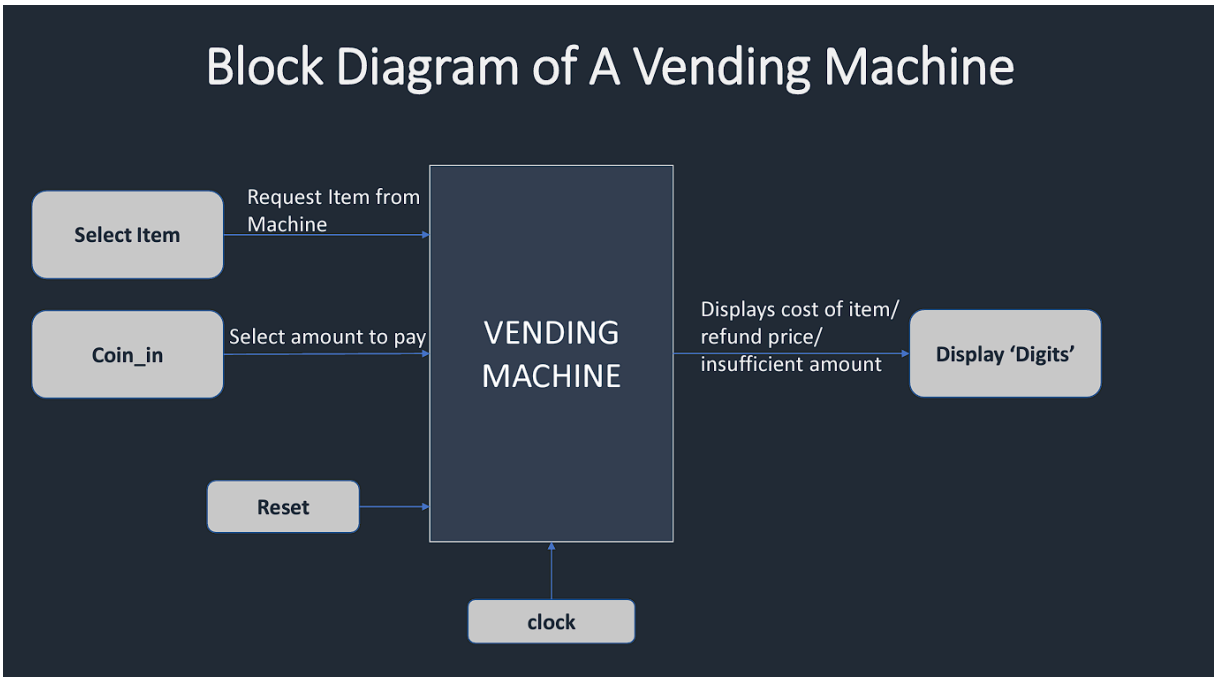

Figure 2: Vending Machine Block Diagram

3 Project Team management

We used Scrum which is a subset of agile methodology. Every week we would discuss the project's progress and come up with to-do tasks. In the beginning, the team had a planned meetings to decide what to work on during the upcoming week and the idea was discussed. We followed a collective way of solving different tasks in which each of us would work together at every step of the project.

The following were the tasks worked upon:

Brainstorming and concept creation: All team members worked together to brainstorm and formulate the initial idea for the smart greenhouse. Each team member contributed their unique perspectives and ideas to create a viable project plan.

Block Diagram and Circuit Design: All three members contributed to the design and development of the block diagram. Also, the schematic, PCB design and 3D view layout of the circuit on KiCAD was accomplished as a collective effort. Each member brought their own ideas, and through discussion and iteration, they came up with the final diagrams.

Coding and Implementation: The coding and implementation of the vending machine concept and circuit design was a shared task. Each team member wrote sections of the code using different ideas

to troubleshoot our issues. We worked together to ensure a compilation and simulation of the code in model SIM and proper synthesis and implementation of our code on the FPGA board.

4 Implementation

KiCAD: This tool is used to create the schematic for the vending machine circuit based on the functionalities as defined in the concept section. Here an evaluation board is given that contains libraries of the footprint and schematic for designing hardware with an FPGA board. This serves as the basis for designing the power supply, utilities and vending machine hardware circuit schematics as shown in figures 3, 4 and 5 respectively. In Figure 5, we make use of the switches and 7 segment display of the FPGA thus the reason for adding the symbols to the schematic. In order to generate the PCB layout, we must assign components to the symbols and, by creating the footprint of the schematic and assigning components from the footprint libraries provided.

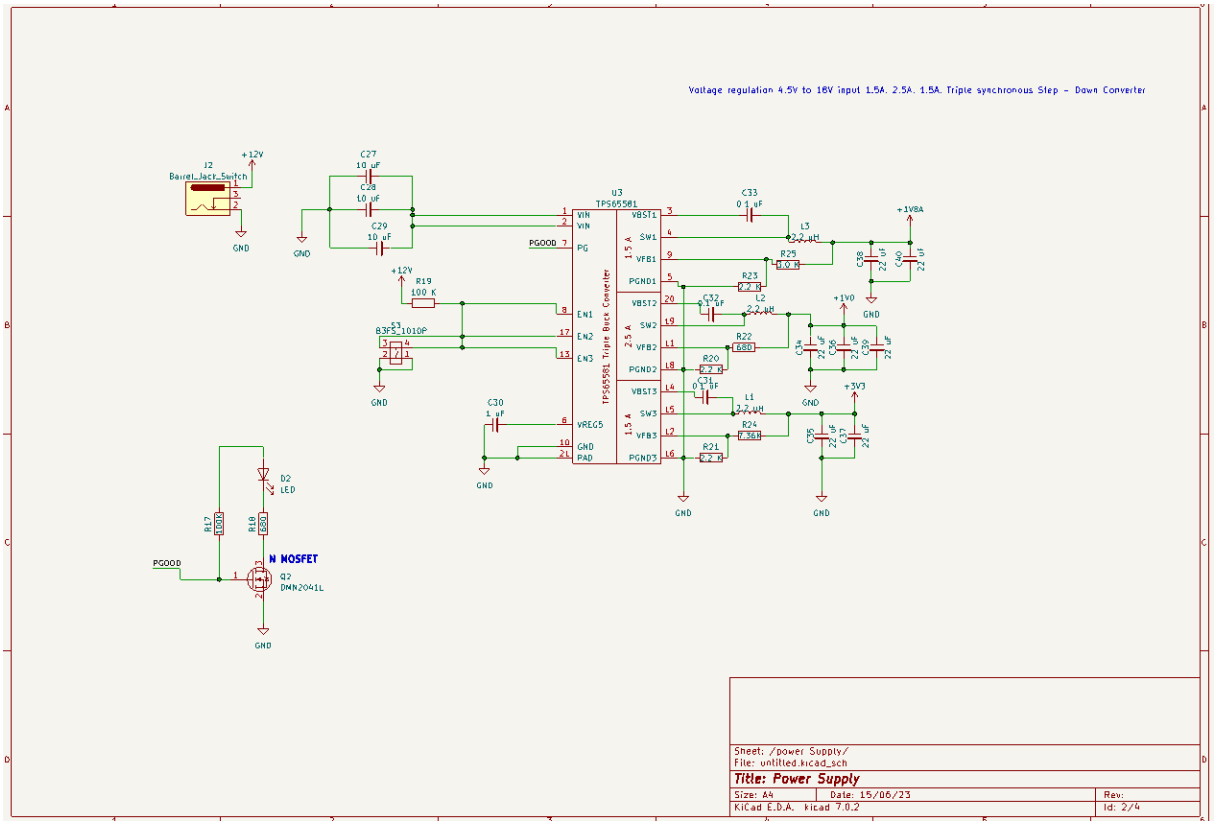

Figure 3: Circuit Power Supply Schematic

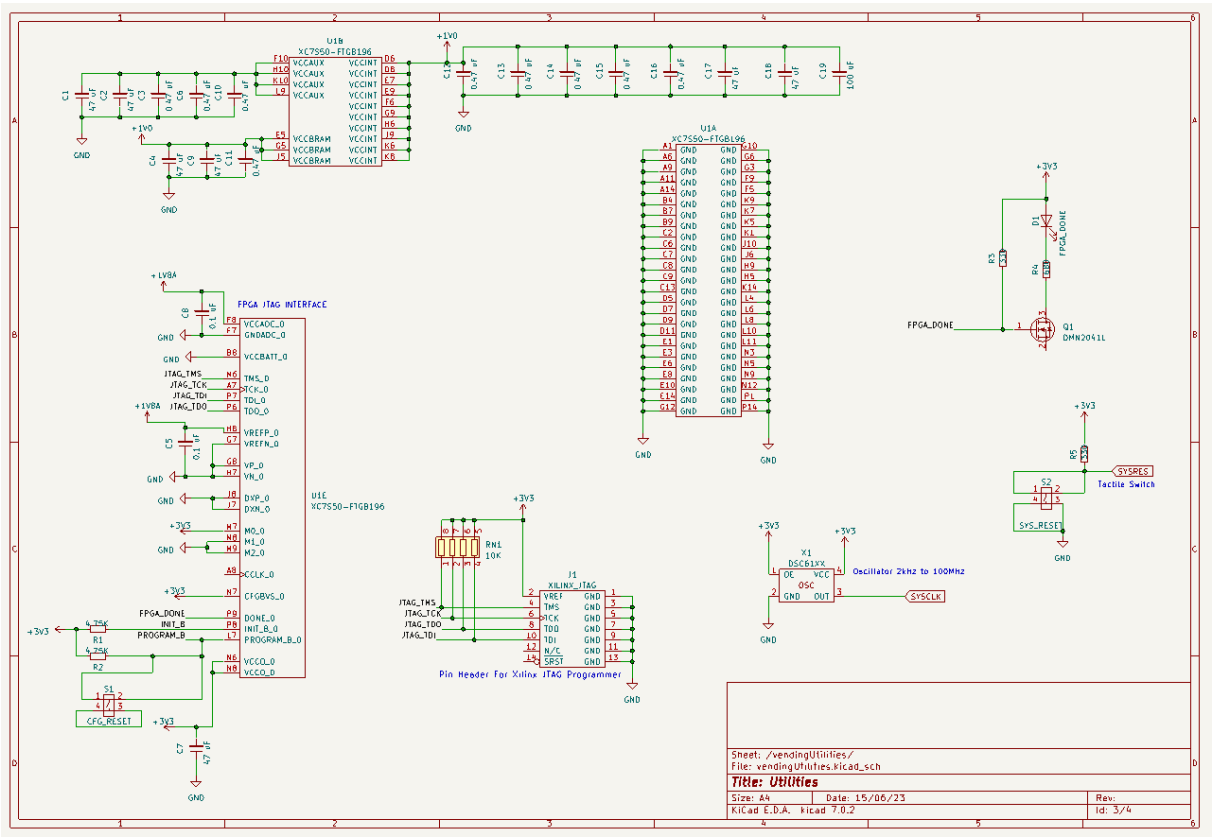

Figure 4: Circuit Utilities Schematic

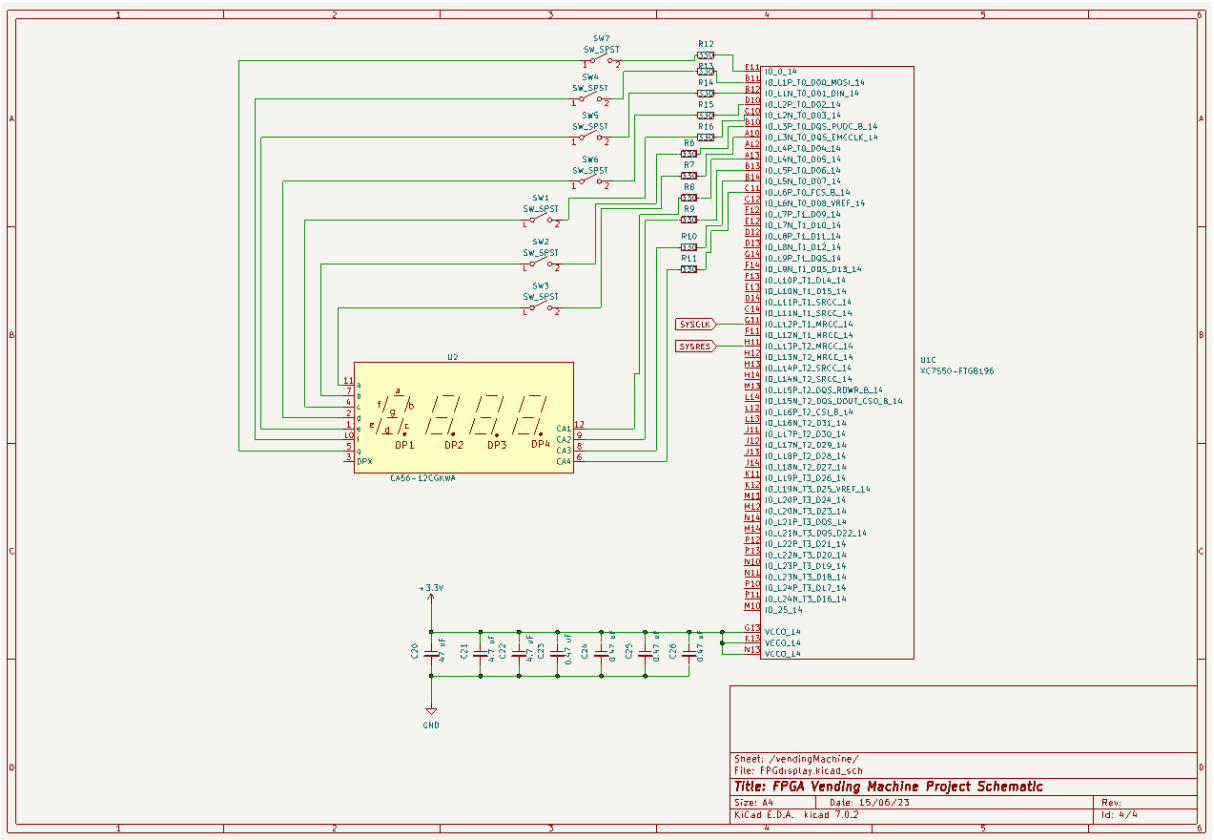

Figure 5: Circuit Vending Machine Schematic

These changes made in the schematic by assigning components are updated to the PCB and then an outline is placed around as seen in figure 6, which corresponds to the size of the board. Auto-routing is performed to route the tracks on the board using an online free routing tool. A 3D view of the board can also be seen and saved as a .png file as shown in figure 7.

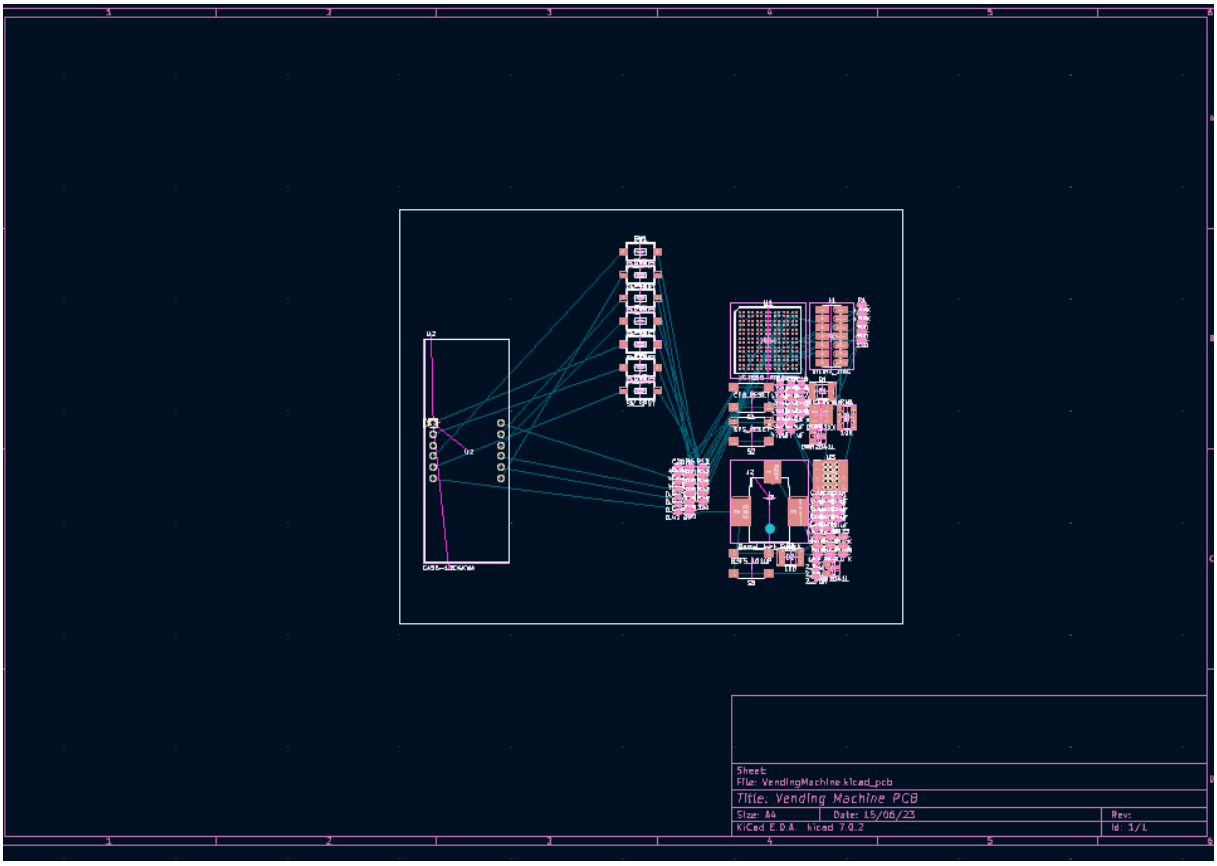

Figure 6: PCB Vending Machine without Auto Routing

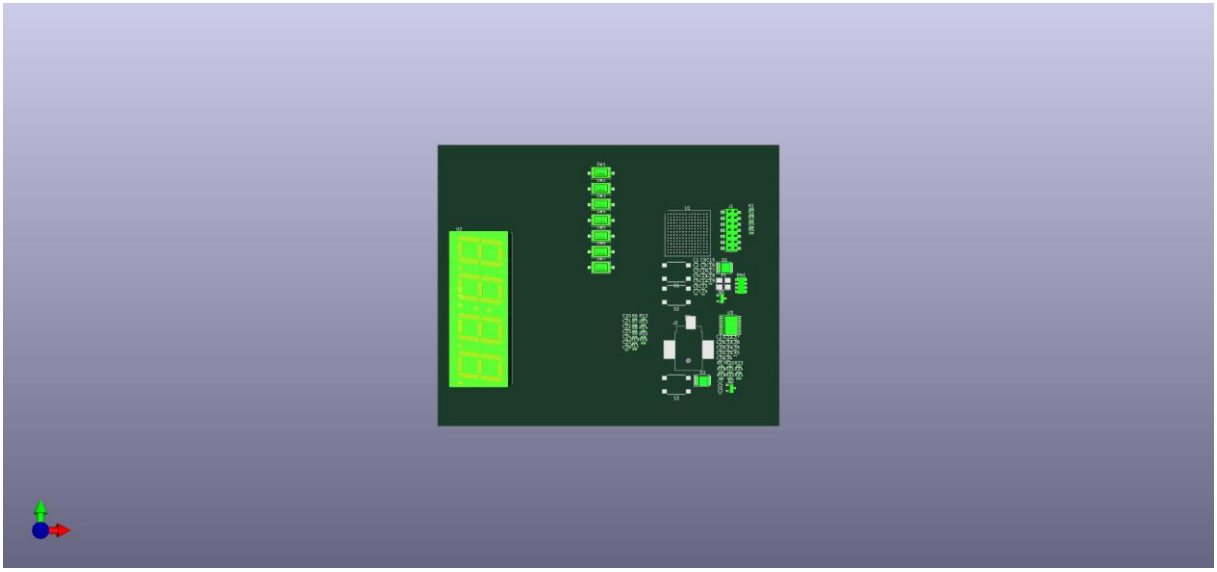

Figure 7: 3D view Vending Machine PCB

MODELSIM: Here the code for implementing the vending state machine is written. Three items are assumed to be contained in the vending machine. The users are expected to select an item and select the amount being placed into the machine. These items are being stored as products to identify the items being selected, and specific prices are assigned to each item. Signals are being used to determine the state of the machine from the idle state till an item has been dispensed from the machine. The DecoderBIt component enables the output values to be displayed on the 7-segment display.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Vending is
  port (
    CLK100MHZ, reset: in std_logic;
        pickItem: in std_logic_vector(2 downto 0);
        selectedItem: out std_logic_vector(2 downto 0);
        coin_in: in std_logic_vector(3 downto 0);
    dispense: out std_logic_vector(6 downto 0));
end Vending;

architecture VendingData of Vending is
component DECODERBIT is
        port(BCD: in std_logic_vector(3 downto 0);
            segment: out std_logic_vector(6 downto 0));
end component;


  type state is (idle, select_item, review, output);
  signal current_state, next_state: state;

begin
    --replaced every CLK with CLK100MHZ
  -- set memory state
  process(CLK100MHZ, reset)
    variable temp: integer;
  begin
    if (reset = '1') then
      current_state <= idle;
    elsif (CLK100MHZ 'event and CLK100MHZ = '1') then
      current_state <= next_state; -- Assign current_state based on next_state
    end if;
  end process;
```

Figure 8: VHDL Code - 1

```
-- deciding the next state
process(current_state, pickItem(0), pickItem(1), pickItem(2), coin_in(0), coin_in(1), coin_in(2), coin_in(3))
      variable total, price, product, refund: integer;
begin
  case (current_state) is
    when idle =>
      selectedItem <= "000";
      dispense <= "0000000";
      if (pickItem(0) = '1') then
              price := 7;
              product := 1;
              total := 0;
              next_state <= select_item;
      end if;
      if (pickItem(1) = '1') then
              price := 9;
              product := 2;
              total := 0;
              next_state <= select_item;
      end if;
      if (pickItem(2) = '1') then
              price := 2;
              product := 3;
              total := 0;
              next_state <= select_item;
      end if;

      when select_item =>
      if (coin_in(0) = '1') then
              total := total + 1;
      elsif (coin_in(1) = '1') then
              total := total + 2;
      elsif (coin_in(2) = '1') then
              total := total + 4;
      elsif (coin_in(3) = '1') then
              total := total + 8;
      end if;
```

Figure 9: VHDL Code - 2

When a user inputs the value for money being used in purchasing an item. When the value amount is more than the price of the item, the machine displays the refund value, else it waits for the user to put in the remainder amount. The display then shows the item being dispensed.

```
      if (price > total) then
              next_state <= select_item;
      elsif (price <= total) then
              refund := total - price;
              next_state <= review;
      end if;

      when review =>
              if (refund >= 4) then
              selectedItem <= "100";
              refund := refund - 4;
              elsif (refund >= 2) then
              selectedItem <= "010";
              refund := refund - 2;
              elsif (refund >= 1) then
              selectedItem <= "001";
              refund := refund - 1;
              end if;
              if (refund > 0) then
              next_state <= review;
              elsif (refund = 0) then
              next_state <= output;
              end if;

      when output =>
              if (product = 1) then
              dispense <= "0000001";
              elsif (product = 2) then
              dispense <= "0000010";
              elsif (product = 3) then
              dispense <= "0000100";
              end if;
              --C <= dispense;
              next_state <= idle;

      ---when others => next_state <= idle;
      end case;
  end process;
DISPLAY: DECODERBIT port map(coin_in, dispense);
end VendingData;
```

VIVADO:

```
## This file is a general .xdc for the Nexys A7-100T
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

# Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];


#Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { pickItem[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { pickItem[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { pickItem[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { coin_in[0] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { coin_in[1] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { coin_in[2] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { coin_in[3] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]

#7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { BCD[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { BCD[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { BCD[2] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { BCD[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { BCD[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { BCD[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { BCD[6] }]; #IO_L4P_T0_D04_14 Sch=cg
#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
#set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { AN[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
#set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { AN[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
#set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { AN[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
```