



Electronic Engineering
Advanced Embedded System Lab
Summer Term 2023
Prof. Dr. Hayek Ali

Documentation for Vending Machine

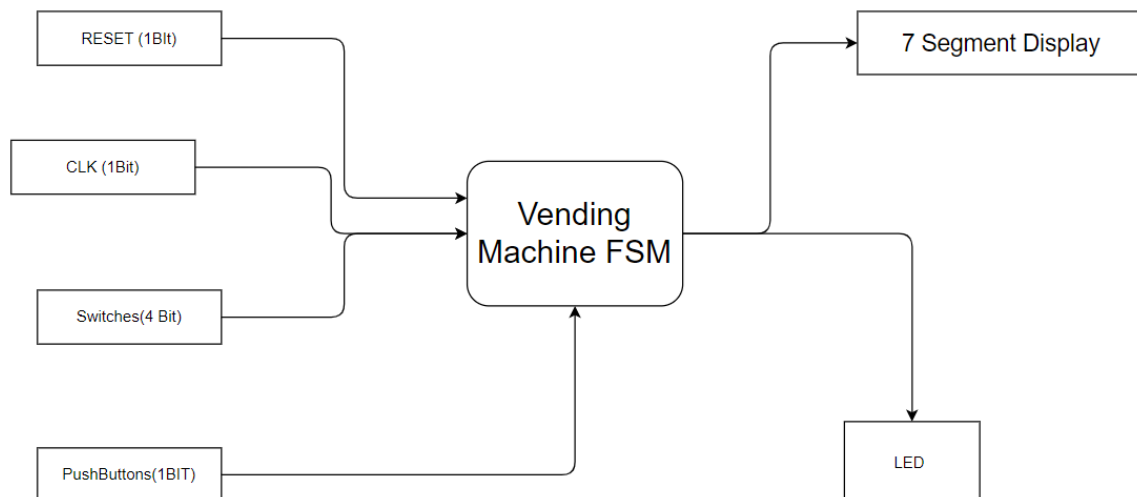
Team F
Shihab Ud Doula
Neaz Mahmud
Yoonsuk Choi

I. INTRODUCTION

A typical Vending machine which is programmed with advanced hardware using an FPGA development board. It will be designed in such way where every aspect is processed with high accuracy by creating a VHDL code which will then be implemented in a FPGA board. This machine gives the user the option of having various goods on the go by proving secure payment method and refund. The specialty of this machine is designed in its code where an FPGA development will play a crucial role as this whole system will be based on the board inputs and outputs in real time.

II. CONCEPT DESCRIPTION






Block Diagram:



Project Description:

Our Vending machine is programmed with advanced hardware using an FPGA development board. It will be designed in such a way where every aspect is processed with high accuracy by creating a VHDL code which will then be implemented in a FPGA board. This machine gives the user the option of having various goods on the go by proving secure payment method and refund. The specialty of this machine is designed in its code where an FPGA development will play a crucial role as this whole system will be based on the board inputs and outputs in real time.

Hardware and Software Used:

-  Nexys A7 100T FPGA board
-  Xilinx Vivado Software
-  ModelSim for Vhdl
-  UML diagrams
-  KICAD

III. PROJECT AND TEAM MANAGEMENT

A. *Used Methodology and Task Breakdown*

In our vending machine project implemented on an FPGA, we employed the following project methods to ensure effective management and coordination:

Agile Methodology: We adopted an agile approach to project management, specifically Scrum. This methodology allowed us to break down the project into smaller, manageable tasks and work on them in short iterations called sprints. We held regular sprint planning meetings, daily stand-up meetings, and sprint review meetings to track progress, discuss any challenges, and adapt the project plan as needed.

Breakdown: To manage our tasks efficiently, we followed the following breakdown;

Requirements Gathering: We started by clearly defining the functional and non-functional requirements of the vending machine. This step involved understanding the desired features, user interactions, and system constraints.

VHDL Modelling: One part of our project involved VHDL modelling. We divided the VHDL modelling tasks into logical modules, such as the vending machine controller, 7-segment display interface, switch and button inputs, and price calculation. Each team member was assigned specific modules to work on based on their expertise.

FPGA Implementation: Another major part of our project was implementing the VHDL code on the FPGA. We allocated tasks for FPGA implementation, including synthesis, implementation and bitstream generation. Team members

responsible for this stage worked closely with the VHDL modellers to ensure seamless integration and functionality.

PCB Design: The final part of our project involved designing the printed circuit board (PCB) for the vending machine. This task required layout design, component placement, and signal routing. Team members with experience in PCB design were assigned these responsibilities.

B. Team Management:

VHDL Modelers: The VHDL modelers were responsible for designing and implementing the various VHDL modules required for the vending machine. They worked on tasks such as creating the controller logic, interfacing with the 7-segment display, and handling switch and button inputs.

FPGA Implementation Specialist: This team member had expertise in FPGA implementation and was responsible for synthesizing the VHDL code, performing placement and routing, and configuring the FPGA with the generated bitstream. They ensured that the VHDL code was correctly translated and executed on the FPGA hardware.

PCB Designer: The PCB designer was responsible for designing the PCB layout, including component placement, signal routing, and ensuring the manufacturability and functionality of the board. They collaborated with the VHDL modellers and FPGA implementation specialist to integrate the FPGA and other components onto the PCB.

Specific Assigned Task

Shihab Ud Doula: ModelSim Simulation

Neaz Mahmud : FPGA Implementation

Younsuk Choi : PCB Design

IV. VHDL & FPGA IMPLEMENTATION

VHDL

In this we will explain the overall architecture of the vending machine that was done by vhdL:

VHDL code for design:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- Update the library import

entity VendingMachine is
port (
    clk : in std_logic;
    reset : in std_logic;
    coin : in std_logic_vector(1 downto 0);
    product1 : in std_logic;
    product2 : in std_logic;
    product3 : in std_logic;
    dispense : out std_logic;
    refund : out std_logic
);
end entity VendingMachine;

architecture Behavioral of VendingMachine is
    type state_type is (idle, product_select, insert_coin, check, dispensing, refunding);
    signal current_state, next_state : state_type;
    signal change_state : std_logic;
    signal amount : unsigned(7 downto 0);
    constant product1_price : unsigned(7 downto 0) := to_unsigned(20, 8); -- Price for
    product 1 (20 cents)
    constant product2_price : unsigned(7 downto 0) := to_unsigned(50, 8); -- Price for
    product 2 (50 cents)
    constant product3_price : unsigned(7 downto 0) := to_unsigned(100, 8); -- Price for
    product 3 (1 euro)

begin
    -- Sequential process to change states
    state_process : process(clk, reset)
    begin
        if reset = '1' then
            current_state <= idle;
            amount <= (others => '0');
            change_state <= '0';
        elsif rising_edge(clk) then
            if change_state = '1' then
                current_state <= next_state;
            end if;
        end if;
    end process state_process;

    -- Combinational process to determine next states and outputs
    vending_process : process(current_state, coin, product1, product2, product3, amount)
    begin
        next_state <= current_state;

        case current_state is
            when idle =>
                if product1 = '1' then
                    next_state <= product_select;
                elsif product2 = '1' then
```

```

    next_state <= product_select;
elseif product3 = '1' then
    next_state <= product_select;
end if;

when product_select =>
    if coin = "01" or coin = "10" or coin = "11" then -- Valid coin denominations
        next_state <= insert_coin;
        case amount is -- Use 'amount' instead of 'product1_price' here
            when product1_price => amount <= product1_price;
            when product2_price => amount <= product2_price;
            when product3_price => amount <= product3_price;
            when others => null; -- Unexpected state
        end case;
    end if;

when insert_coin =>
    if coin = "01" or coin = "10" or coin = "11" then -- Valid coin denominations
        amount <= amount + unsigned(coin);
    end if;

    if (amount >= product1_price and product1 = '1') or
       (amount >= product2_price and product2 = '1') or
       (amount >= product3_price and product3 = '1') then
        next_state <= check;
    end if;

when check =>
    if amount >= product1_price and product1 = '1' then
        next_state <= dispensing;
        amount <= amount - product1_price;
    elseif amount >= product2_price and product2 = '1' then
        next_state <= dispensing;
        amount <= amount - product2_price;
    elseif amount >= product3_price and product3 = '1' then
        next_state <= dispensing;
        amount <= amount - product3_price;
    else
        next_state <= refunding;
    end if;

when dispensing =>
    next_state <= idle;

when refunding =>
    next_state <= idle;

end case;

change_state <= '1'; -- Set the signal to indicate state change
end process vending_process;

-- Output assignments
dispense <= '1' when current_state = dispensing else '0';
refund <= '1' when current_state = refunding else '0';

end architecture Behavioral;

```

Code explanation:

For better understanding each part of the code is explained:

1. **Library and Use Clauses:** The library and use clauses import necessary libraries, such as **ieee.std_logic_1164** and **ieee.numeric_std**, which provide standard data types and operations used in the design.
2. **Entity Declaration:** The entity declaration defines the interface of the **VendingMachine** module. It specifies the input and output ports of the module. In this case, the **VendingMachine** entity has inputs (**clk**, **reset**, **coin**, **product1**, **product2**, **product3**) and outputs (**dispense**, **refund**).
3. **Architecture Declaration:** The architecture declaration, named Behavioral, specifies the behavior of the **VendingMachine** module. It contains the processes and logic to define the behavior of the module.
4. **Type Declaration:** The **state_type** is a user-defined enumeration type that represents the different states of the vending machine. The possible states are **idle**, **product_select**, **insert_coin**, **check**, **dispensing**, and **refunding**.
5. **Signal Declarations:** Signals are declared to represent various internal states and values within the **VendingMachine** module. The **current_state** and **next_state** signals track the current and next states of the vending machine. The **change_state** signal indicates a state change. The **amount** signal holds the total amount of inserted coins.
6. **Constant Declarations:** Constants are declared to hold the prices of the three products in the vending machine. The constants **product1_price**, **product2_price**, and **product3_price** define the prices of product 1, product 2, and product 3, respectively.
7. **Sequential Process:** The **state_process** is a sequential process triggered by changes in the **clk** and **reset** signals. It updates the **current_state** based on the **next_state** and sets the initial values when the **reset** signal is active.
8. **Combinational Process:** The **vending_process** is a combinational process triggered by changes in the **current_state**, **coin**, **product1**, **product2**, **product3**, and **amount** signals. It determines the **next_state** based on the

current_state and input conditions. It performs actions based on the current state, such as handling coin insertion, product selection, and price comparison.

9. **Case Statement:** The case statement inside the **vending_process** handles different cases based on the **current_state**. It checks the current state and updates the **next_state** accordingly. It also handles coin insertion, product selection, and price comparison using the **amount** signal.
10. **Output Assignments:** The output assignments assign values to the dispense and refund outputs based on the **current_state**. When the **current_state** is **dispensing**, the **dispense** signal is set to '1', indicating that a product is being dispensed. Similarly, when the **current_state** is **refunding**, the refund signal is set to '1', indicating that a refund is being initiated.

Testbench code for VHDL:

```
library ieee;
use ieee.std_logic_1164.all;

entity VendingMachine_TB is
end entity VendingMachine_TB;

architecture Behavioral of VendingMachine_TB is
-- Component declaration for the unit under test (UUT)
component VendingMachine is
port (
    clk : in std_logic;
    reset : in std_logic;
    coin : in std_logic_vector(1 downto 0);
    product1 : in std_logic;
    product2 : in std_logic;
    product3 : in std_logic;
    dispense : out std_logic;
    refund : out std_logic
);
end component VendingMachine;

-- Signal declarations
signal clk : std_logic := '0';
signal reset : std_logic := '0';
signal coin : std_logic_vector(1 downto 0) := "00";
signal product1 : std_logic := '0';
signal product2 : std_logic := '0';
signal product3 : std_logic := '0';
signal dispense : std_logic;
signal refund : std_logic;

begin
-- Instantiate the unit under test (UUT)
ut: VendingMachine
port map (
```



```

    clk => clk,
    reset => reset,
    coin => coin,
    product1 => product1,
    product2 => product2,
    product3 => product3,
    dispense => dispense,
    refund => refund
);

-- Stimulus process
stim_proc: process
begin
    -- Initialize inputs
    reset <= '1';
    coin <= "00";
    product1 <= '0';
    product2 <= '0';
    product3 <= '0';

    wait for 10 ns;
    reset <= '0';

    wait for 10 ns;
    product1 <= '1';

    wait for 10 ns;
    product1 <= '0';
    product2 <= '1';

    wait for 10 ns;
    coin <= "01";

    wait for 10 ns;
    coin <= "10";

    wait for 10 ns;
    coin <= "11";

    wait for 10 ns;
    product3 <= '1';

    wait for 100 ns;
    -- Add more stimulus as needed

    wait;
end process stim_proc;

end architecture Behavioral;

```

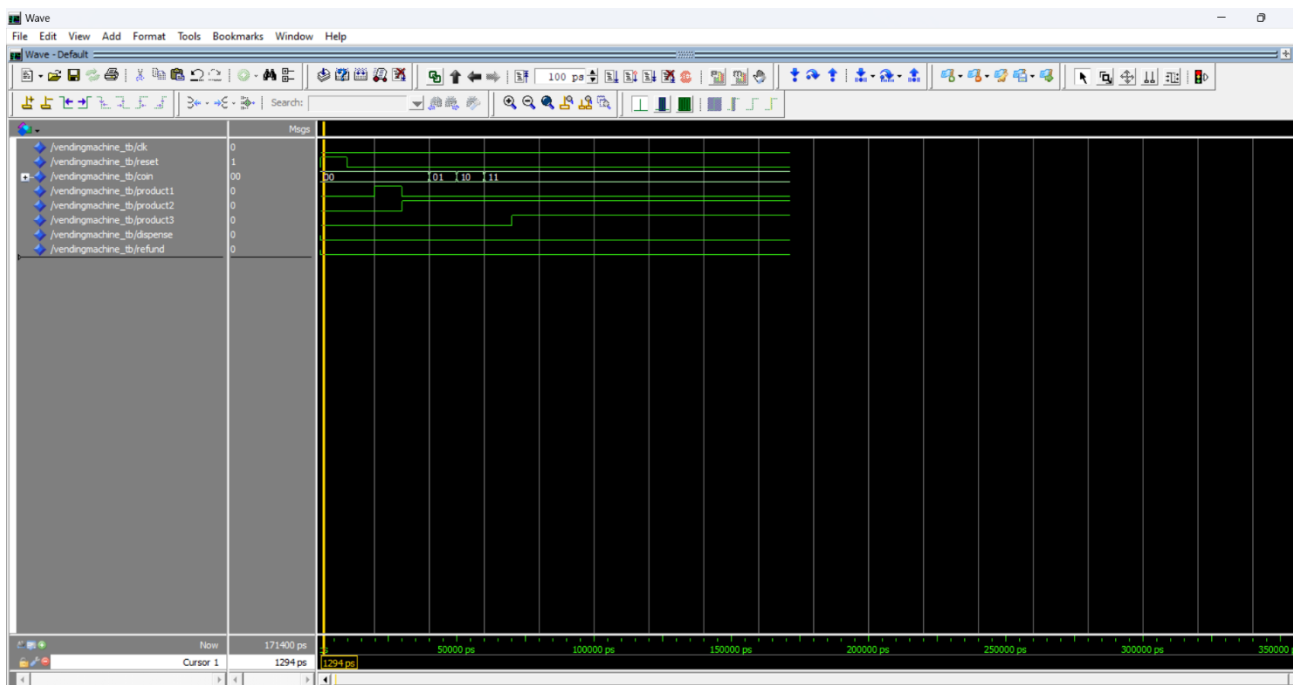
Testbench Code Explanation:

1. Library and Use Clauses:
 - **library ieee;** and **use ieee.std_logic_1164.all;** import the necessary IEEE standard libraries, including the **std_logic** type.
2. Entity Declaration:
 - entity **VendingMachine_TB** is declares the testbench entity.
3. Architecture Declaration:
 - **architecture Behavioral of VendingMachine_TB is** defines the behavioral architecture of the testbench.
4. Component Declaration:
 - **component VendingMachine** declares the component for the unit under test (UUT). It specifies the ports of the UUT.
5. Signal Declarations:
 - Signals are declared to connect to the ports of the UUT and stimulate its behavior during simulation.
6. UUT Instantiation:
 - **uut: VendingMachine** instantiates the unit under test (VendingMachine) using the declared component. It maps the signals to the ports of the UUT.
7. Stimulus Process:
 - **stim_proc: process** defines a process for generating stimulus to the UUT.
 - The process initializes input signals and applies stimulus values at specific time intervals using wait statements.
 - The wait statements introduce time delays between stimulus actions.
 - Additional stimulus can be added as needed by extending the process.

This testbench code sets up the simulation environment, initializes input signals, applies stimulus, and captures the output responses from the UUT. The stimulus process drives inputs to the UUT, while the UUT itself is instantiated and connected to the signals. This allows the simulation to observe the behavior of the Vending Machine module and verify its functionality.

Output and Simulation:

The testbench was run in a simulation of vhdl and following screenshot is given:



Note: The full code and other necessary files are uploaded on GITHUB:

<https://github.com/Shihab-007/Advanced-Embedded-System-Hardware-Engineering/tree/main>

FPGA

V. PCB DESIGN

To design the PCB for our FPGA evaluation board, the following steps are taken.

1. We selected the FPGA which was Artix-7.
2. Then we identified the necessary modules to be included in the PCB design. Those are FPGA Artix 7, power supply, JTAG programming interface, clock (CLK) interface, and reset interface. 7-segment displays, and 7 switches.
3. For the next step, we planned the board layout. In this step, we had to consider the size and form factor of our PCB. We determined the placement and orientation of each module on the board, ensuring that there is enough space for traces, power supplies, etc.
4. After this stage, we created the schematic: We drew our schematics using KiCAD tool. The layout of the schematic comprises of three layers. Those are (1) Power Supply Schematics, (2) FPGA Utilities Schematics, and (3) FPGA Externals

Schematics. Power Supply schematics ensure that power supply and ground connections are properly distributed throughout the design. FPGA Utilities Schematics include Artix-7 chip, JTAG programmer interface, Reset module, and Clock module. FPGA Externals Schematics include 7-segment display and 7 switches.

5. Then we moved on to designing the PCB layout. This step required us to place each component on the board according to our planned layout. In this step, we focused on criteria such as minimizing trace lengths and avoiding any interference between signals.

References

1. Alaattinyilmaz. (n.d.). vending-machine/vending_machine.v at master · alaattinyilmaz/vending-machine. GitHub. https://github.com/alaattinyilmaz/vending-machine/blob/master/vending_machine.v
2. comp.arch.fpga | Simulation of VHDL code for a vending machine. (n.d.). <https://www.fpgarelated.com/showthread/comp.arch.fpga/89364-1.php>
3. Falcao, R. E. (2022, April 23). Vending Machine in Verilog HDL | Medium. Medium. <https://medium.com/@rebsfalcao15/design-of-vending-machine-using-verilog-hdl-42cede6cf8bb>
4. Nation Innovation. (2019, April 13). Vending Machine| VHDL Code | Simulation | FSM Based | Nation Innovation | B.Tech Final Year Project [Video]. YouTube. <https://www.youtube.com/watch?v=ywrSG9J3taE>
5. Zhang, J. (2012). The Design, Simulation, Verification and Implementation of Vending Machine Based on FPGA. <https://kb.osu.edu/dspace/handle/1811/52066>

