Scott Will, Kyle Thompson 6 December 2014

EE478 Final Report

1. **Objectives**

The objective of this assignment was to design a model vending machine using a finite-state machine, and implement the design on the Spartan-6 FPGA.

1. **Specifications**

Our design was required to accept monetary inputs of $0.25, $0.50, and $1.00, and dispense four possible items (labeled Item A, Item B, Item C, and Item D for convenience), which carried costs of $0.25, $0.50, $0.75, $1.00, respectively. The implementation was additionally required to have a reset input, allowing any money contained in the machine to be immediately returned to the user, and to return an excess money should the sum contained in the machine exceed $1.00. Finally, the design was required to display the current monetary total contained in the machine, the purchase selected by the user (if any), and, for a short period of time, any change remaining after a purchase.

1. **Technical Discussion**
2. *Physical Input/Output*

We chose to implement the three possible inputs, as well as the reset button, on four of the five push buttons on the Spartan-6 board. The four available items, meanwhile, were implemented on four of the eight switches on the board. All inputs and outputs were displayed using the eight LEDs above the switches.

1. *Signal Encoding*

We chose to encode the inputs as two separate four-bit vectors. One for the pushbuttons to input money/reset, and one for the switches that choose the item. We encoded the output into an eight-bit vector that represented all of the LEDs.

1. *State Reduction*

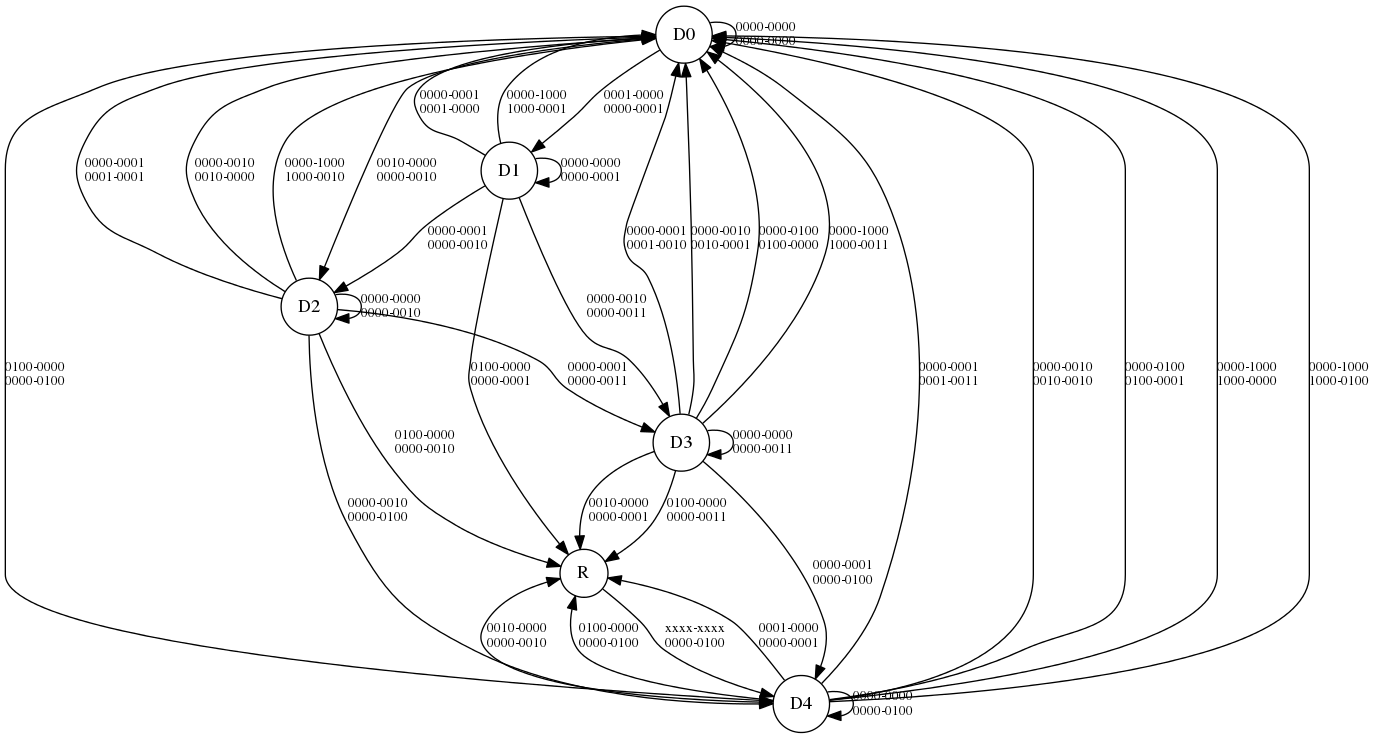
When we first started working on the state logic, we had a state for every possible transaction you could make, being able to enter up to $2.00 into the machine. We quickly realized this was not very efficient and decided to have separate states for each amount of money you could have in the machine, and each amount of change that could be returned. This about halved the number of states, but we realized that we could further reduce it by not allowing the user to enter more than a dollar, and immediately return any money in excess of a dollar. This allowed us to remove the change states, and replace them with four return states, that returned extra money and then returned to the state it came from. At this point we had nine states. Our final reduction was when we realized that we only needed one return state, whose output changed based upon which state came before it.

1. *VHDL Implementation*

To implement our state machine in VHDL, there were a few things we needed to address. Most importantly, the state machine logic, and the state machine clock. For the clock we used a clock divider to divide the board’s 100MHz clock down to 1Hz. This clock allowed us to achieve displaying of the item and change for one second. It also allowed us to not worry about button contact bounce, as the slower clock cannot detect changes that are that quick. For the state machine logic we used a case statement for the states, with nested if statements for the different interactions. The nested if statements allowed us to catch different conditions in the else, which saved us from coding every possible combination of inputs. The case statement made it very easy to debug, because we could just go to the case for the state that was not operating correctly.

1. **State Diagram**

The state diagram of our implementation was generated using the GraphViz specification package (which is built on the DOT language) and is shown below. We were able to achieve our design using only six states- five corresponding to dollar amounts ranging from $0.00 to $1.00 in increments of $0.25 (D0-D4 below), and one return state (denoted ‘R’ below) used to handle cases of money overflow, i.e. when the user attempted to insert more money than the $1.00 machine limit.



**Figure 1:** Implementation state diagram. The Mealy inputs and outputs are stacked vertically to reduce the amount of horizontal space required to generate the graph.

1. **State Table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **CS** | **AMT** | **Money** | **Item** | **NS** | **AMT** | **Item** | **Balance** |
| D0 | $ - | 0000 | 0000 | D0 | $ - | 0000 | 0000 |
| D0 | $ - | 0001 | 0000 | D1 | $ 0.25 | 0000 | 0001 |
| D0 | $ - | 0010 | 0000 | D2 | $ 0.50 | 0000 | 0010 |
| D0 | $ - | 0100 | 0000 | D4 | $ 1.00 | 0000 | 0100 |
| D0 | $ - | 1000 | 0000 | D0 | $ - | 0000 | 0000 |
| D1 | $ 0.25 | 0000 | 0000 | D1 | $ 0.25 | 0000 | 0001 |
| D1 | $ 0.25 | 0001 | 0000 | D2 | $ 0.50 | 0000 | 0010 |
| D1 | $ 0.25 | 0010 | 0000 | D3 | $ 0.75 | 0000 | 0011 |
| D1 | $ 0.25 | 0100 | 0000 | R | $ 0.25 | 0000 | 0001 |
| D1 | $ 0.25 | 1000 | 0000 | D0 | $ - | 0000 | 0001 |
| D1 | $ 0.25 | 0000 | 0001 | D0 | $ - | 0001 | 0000 |
| D2 | $ 0.50 | 0000 | 0000 | D2 | $ 0.50 | 0000 | 0010 |
| D2 | $ 0.50 | 0001 | 0000 | D3 | $ 0.75 | 0000 | 0011 |
| D2 | $ 0.50 | 0010 | 0000 | D4 | $ 1.00 | 0000 | 0100 |
| D2 | $ 0.50 | 0100 | 0000 | R | $ 0.50 | 0000 | 0010 |
| D2 | $ 0.50 | 1000 | 0000 | D0 | $ - | 0000 | 0010 |
| D2 | $ 0.50 | 0000 | 0001 | D0 | $ 0.25 | 0001 | 0001 |
| D2 | $ 0.50 | 0000 | 0010 | D0 | $ - | 0010 | 0000 |
| D3 | $ 0.75 | 0000 | 0000 | D3 | $ 0.75 | 0000 | 0011 |
| D3 | $ 0.75 | 0001 | 0000 | D4 | $ 1.00 | 0000 | 0100 |
| D3 | $ 0.75 | 0010 | 0000 | R | $ 0.25 | 0000 | 0001 |
| D3 | $ 0.75 | 0100 | 0000 | R | $ 0.75 | 0000 | 0011 |
| D3 | $ 0.75 | 1000 | 0000 | D0 | $ - | 0000 | 0011 |
| D3 | $ 0.75 | 0000 | 0001 | D0 | $ 0.50 | 0001 | 0010 |
| D3 | $ 0.75 | 0000 | 0010 | D0 | $ 0.25 | 0010 | 0010 |
| D3 | $ 0.75 | 0000 | 0100 | D0 | $ - | 0100 | 0000 |
| D4 | $ 1.00 | 0000 | 0000 | D4 | $ 1.00 | 0000 | 0100 |
| D4 | $ 1.00 | 0001 | 0000 | R | $ 0.25 | 0000 | 0001 |
| D4 | $ 1.00 | 0010 | 0000 | R | $ 0.50 | 0000 | 0010 |
| D4 | $ 1.00 | 0100 | 0000 | R | $ 1.00 | 0000 | 0100 |
| D4 | $ 1.00 | 1000 | 0000 | D0 | $ - | 0000 | 0100 |
| D4 | $ 1.00 | 0000 | 0001 | D0 | $ 0.75 | 0001 | 0011 |
| D4 | $ 1.00 | 0000 | 0010 | D0 | $ 0.50 | 0010 | 0010 |
| D4 | $ 1.00 | 0000 | 0100 | D0 | $ 0.25 | 0100 | 0001 |
| D4 | $ 1.00 | 0000 | 1000 | D0 | $ - | 1000 | 0000 |
| R | (Return) | xxxx | xxxx | D4 | $ 1.00 | 0000 | 0100 |

|  |
| --- |
| **Legend** |
| Input |
| Purchase |
| Return |

Note on the above table: a cell value of ‘$ - ’ represents the value of $0.00 (the representation shown is the Excel default for items in the “Accounting” format. For an explanation of color coding, see the legend at the bottom of the page above.

1. **Source Code**

*----------------------------------------------------------------------------------*

*-- Company:*

*-- Engineer: Scott Will (50003395)*

*-- Kyle Thompson (36870784)*

*--*

*-- Create Date: 21:59:26 11/29/2014*

*-- Design Name: EE478 Vending Machine*

*-- Module Name: main - Behavioral*

*-- Project Name:*

*-- Target Devices:*

*-- Tool versions:*

*-- Description:*

*--*

*-- Dependencies:*

*--*

*-- Revision:*

*-- Revision 0.01 - File Created*

*-- Additional Comments:*

*--*

*----------------------------------------------------------------------------------*

**library** **IEEE**;

**use** **IEEE.STD\_LOGIC\_1164.ALL**;

*-- Uncomment the following library declaration if using*

*-- arithmetic functions with Signed or Unsigned values*

*--use IEEE.NUMERIC\_STD.ALL;*

*-- Uncomment the following library declaration if instantiating*

*-- any Xilinx primitives in this code.*

*--library UNISIM;*

*--use UNISIM.VComponents.all;*

**entity** **main** **is**

**Port** (

LED : **out** STD\_LOGIC\_VECTOR(7 **DOWNTO** 0);

*-- 3 pushbuttons for 25, 50, and 100 cents*

button : **in** STD\_LOGIC\_VECTOR(3 **DOWNTO** 0);

*-- 4 switches for the four different items*

item : **in** STD\_LOGIC\_VECTOR(3 **DOWNTO** 0);

clk\_100MHz : **IN** STD\_LOGIC

);

**end** **main**;

**architecture** **Behavioral** **of** **main** **is**

*-- Create a new datatype called state\_type whose legal values are any dollar value*

*-- between $0.00 and $1.00 (states d0-d4) and overflow return value (r4)*

*-- Inputs are 8 bits*

*-- Outputs are 8 bits*

**type** state\_type **is** (d0, d1, d2, d3, d4, r4);

*-- Create an internal signal of the state\_type type*

**signal** state: state\_type;

*-- Create an internal signal to hold the signal from the money*

**signal** money: STD\_LOGIC\_VECTOR(3 **downto** 0);

*-- Signal for 1Hz clock*

**signal** clk\_1Hz : STD\_LOGIC;

**component** **ck\_divider** **is**

**Port** ( CK\_IN : **in** STD\_LOGIC;

CK\_OUT : **out** STD\_LOGIC);

**end** **component**;

**begin**

*-- Divides the clock to 1Hz*

divide : ck\_divider **port** **map** (clk\_100MHz, clk\_1Hz);

machine: **process**

*-- We have three possible money inputs, so they will be encoded as a four-bit vector on the buttons*

*-- $0.25 <==> 0001*

*-- $0.50 <==> 0010*

*-- $1.00 <==> 0100*

*-- RESET <==> 1000 (top button on pad)*

*-- We have five possible vending inputs (A, B, C, D, nothing), so they will be encoded as a four-bit vector on the*

*-- the switches*

*-- Ø <==> 0000*

*-- A <==> 0001*

*-- B <==> 0010*

*-- C <==> 0100*

*-- D <==> 1000*

*-- All inputs are 8-bit vectors structured as xxxx|xxxx where the left four bits are the money input and the right four*

*-- are the item selection*

**begin**

**if** rising\_edge(clk\_1Hz) **then**

**case** state **is**

*-- $0.00 dollar state*

**when** d0 =>

LED <= "00000000";

*-- All possible inputs and no item selected*

**if** item = "0000" **then**

*-- $0.00 -> $0.25*

**if** button = "0001" **then**

state <= d1;

LED <= "00000001";

*-- $0.00 -> $0.50*

**elsif** button = "0010" **then**

state <= d2;

LED <= "00000010";

*-- $0.00 -> $1.00*

**elsif** button = "0100" **then**

state <= d4;

LED <= "00000100";

**end** **if**;

*-- If we selected some item to be purchased, then stay in the current state because we don't have any* *money to purchase things. For THIS STATE ONLY, the reset case (input = 1000-0000) is included, since there's no change to refund when the reset signal comes in*

**else**

state <= d0;

LED <= "00000000";

**end** **if**;

*-- $0.25 dollar state*

**when** d1 =>

*-- No purchase*

**if** item = "0000" **then**

*-- $0.25 -> $0.50*

**if** button = "0001" **then**

state <= d2;

LED <= "00000010";

*-- $0.25 -> $0.75*

**elsif** button = "0010" **then**

state <= d3;

LED <= "00000011";

*-- $0.25 -> $1.25*

**elsif** button = "0100" **then**

state <= r4;

LED <= "00000001";

*-- RESET*

**elsif** button = "1000" **then**

state <= d0;

LED <= "00000001";

**end** **if**;

*-- $0.25 purchase*

**elsif** item = "0001" **then**

state <= d0;

LED <= "00010000";

*-- No input, or purchase of something that is too expensive*

**else**

state <= d1;

LED <= "00000001";

**end** **if**;

*-- $0.50 dollar state*

**when** d2 =>

*-- No purchase*

**if** item = "0000" **then**

*-- $0.50 -> $0.75*

**if** button = "0001" **then**

state <= d3;

LED <= "00000011";

*-- $0.50 -> $1.00*

**elsif** button = "0010" **then**

state <= d4;

LED <= "00000100";

*-- $.50 -> $1.50*

**elsif** button = "0100" **then**

state <= r4;

LED <= "00000010";

*-- RESET*

**elsif** button = "1000" **then**

state <= d0;

LED <= "00000010";

**end** **if**;

*-- $0.25 purchase*

**elsif** item = "0001" **then**

state <= d0;

LED <= "00010001";

*-- $0.50 purchase*

**elsif** item = "0010" **then**

state <= d0;

LED <= "00100000";

*-- No input, or purchase of something that is too expensive*

**else**

state <= d2;

LED <= "00000010";

**end** **if**;

*-- $0.75 dollar state*

**when** d3 =>

*-- No purchase*

**if** item = "0000" **then**

*-- $0.75 -> $1.00*

**if** button = "0001" **then**

state <= d4;

LED <= "00000100";

*-- $0.75 -> $1.25*

**elsif** button = "0010" **then**

state <= r4;

LED <= "00000001";

*-- $0.75 -> $1.75*

**elsif** button = "0100" **then**

state <= r4;

LED <= "00000011";

*-- RESET*

**elsif** button = "1000" **then**

state <= d0;

LED <= "00000011";

**end** **if**;

*-- $0.25 purchase*

**elsif** item = "0001" **then**

state <= d0;

LED <= "00010010";

*-- $0.50 purchase*

**elsif** item = "0010" **then**

state <= d0;

LED <= "00100001";

*-- $0.75 purchase*

**elsif** item = "0100" **then**

state <= d0;

LED <= "01000000";

*-- No input, or purchase of something that is too expensive*

**else**

state <= d3;

LED <= "00000011";

**end** **if**;

*-- $1.00 dollar state*

**when** d4 =>

*-- No purchase*

**if** item = "0000" **then**

*-- $1.00 -> $1.25*

**if** button = "0001" **then**

state <= r4;

LED <= "00000001";

*-- $1.00 -> $1.50*

**elsif** button = "0010" **then**

state <= r4;

LED <= "00000010";

*-- $1.00 -> $2.00*

**elsif** button = "0100" **then**

state <= r4;

LED <= "00000100";

*-- RESET*

**elsif** button = "1000" **then**

state <= d0;

LED <= "00000000";

**end** **if**;

*-- $0.25 purchase*

**elsif** item = "0001" **then**

state <= d0;

LED <= "00010011";

*-- $0.50 purchase*

**elsif** item = "0010" **then**

state <= d0;

LED <= "00100010";

*-- $0.75 purchase*

**elsif** item = "0100" **then**

state <= d0;

LED <= "01000001";

*-- $1.00 purchase*

**elsif** item = "1000" **then**

state <= d0;

LED <= "10000000";

*-- No input*

**else**

state <= d4;

**end** **if**;

*-- Excess return state*

**when** r4 =>

state <= d4;

LED <= "00000100";

**end** **case**;

**end** **if**;

**end** **process**;

**end** **Behavioral**;