ELEE 204 Design Project

Kaleb Dekker

JD Elsey

David Fritts

Domenic Rodriguez

**I. Problem Statement**

In our experience, dormitory washing machines are particularly prone to breaking down given their high volume of usage. Often they are unable to safely recover from error conditions, leaving the user’s clothing potentially soaked or causing flooding. Our goal is to design a robust control system for a dormitory washing machine that is able to provide useful diagnostic information and safely recovery from error states.

**II. Design Criteria**

1. Robust error handling and informative diagnostic information
2. Multiple temperature/cycle options
3. Manual user override and refund option

**III. Design Details**

**A. General Overview**

The control system is divided into the following subsystems: payment, cycle control, and error handling. The subsystems are implemented as independent finite state machines, communicating via VHDL signals. The payment treats asynchronous quarter input as its clock signal, while the cycle control state machine is driven by the 50 MHz clock on the Altera DE2 board. The error handling exists as a subsection of the cycle control state machine (see Appendix A for the final code).

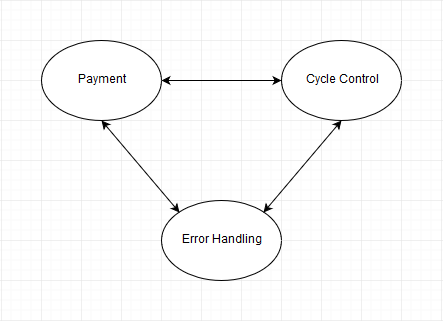


Figure 1 - Overall Washing Machine Control System Structure

**B. Payment System**

The payment system is a finite state machine driven with states to represent the amount of money placed into the machine. The state machine treats quarter input as a clock signal. If a coin jam is detected, the payment system can asynchronously jump to the error state. The payment system coordinates with the cycle control system to determine if the machine is ready to run. The cycle control system cannot begin until the payment system detects either $1.00 or $1.25 has been placed into the machine. If the user places $1.25 into the machine, a SuperCycle will be run instead of a regular cycle. At any point during the payment process the user can request a refund from the machine. The machine will then enter the refund state where it will return all of the quarters to the user.

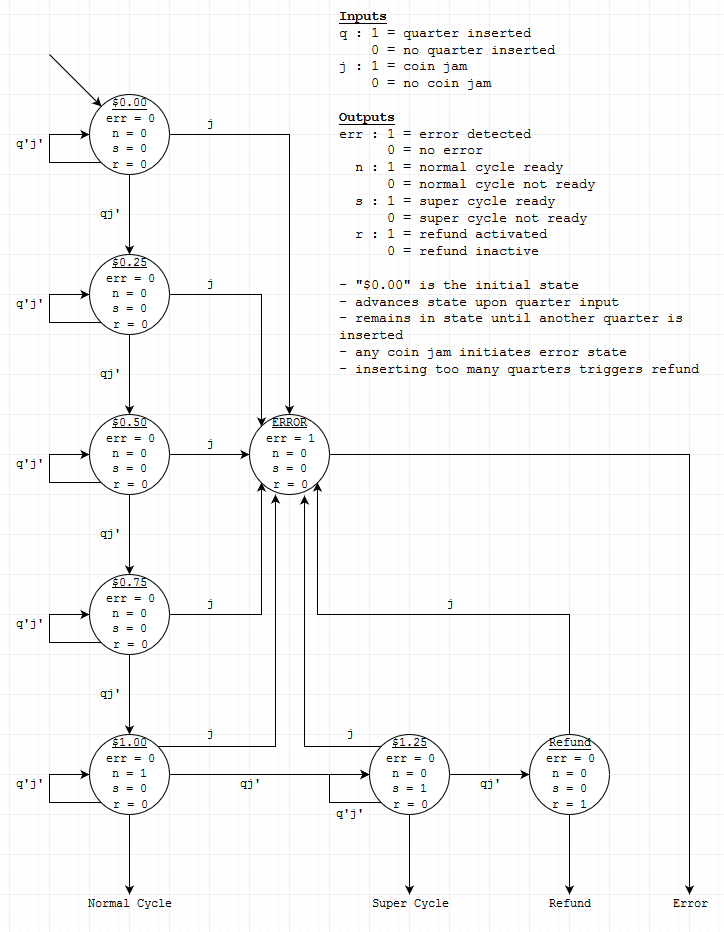


Figure 2 - Payment System

**C. Cycle Control**

The cycle control system runs side-by-side with the payment system. It begins in the cycle\_select state and remains there until sufficient payment has been entered into the washing machine. Once payment has been entered, the cycle control system waits for the user to select a cycle type (whites, colors, or brights) A whites cycle washes with hot water, colors cycle washes with warm water, and brights cycle washes with cold water. Once a wash cycle has begun, the control flow is as follows:

1. Fill: fill the washer with the correct temperature water.
2. Wash: add detergent and spin/agitate the washer compartment
3. Wash (Extended): extended wash cycle for a SuperCycle.
4. Rinse: run additional water of the same temperature through the machine to rinse the clothing.
5. Spin: spin out the water and drain the washing machine.

For simulation purposes, each of the cycle stages runs for 30 seconds. If at any point during the cycle stage the machine has an error, then the cycle controller will move to the error state. The two errors that can be encountered by the machine are an offset in the main compartment’s balance and an error in the water level. Either of the events will cause the machine to halt its processes immediately and enter the ERROR state. The machine will remain in this state until the reset input is given, signaling that maintenance has been done. The next state will return the user’s money before returning to the payment stage of the system.

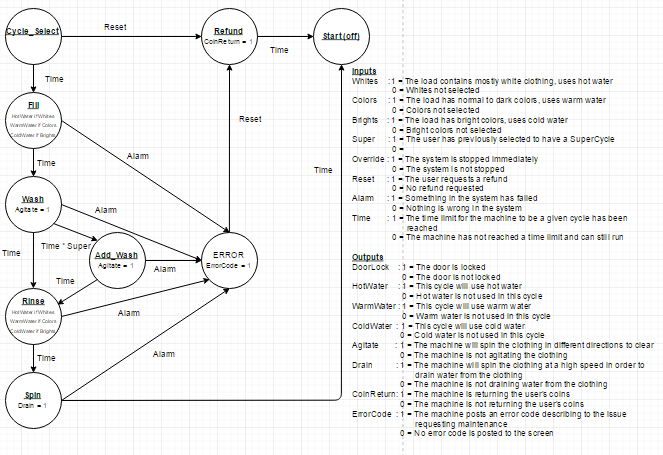


Figure 3 - Cycle Control System

**IV. User Interface Overview**

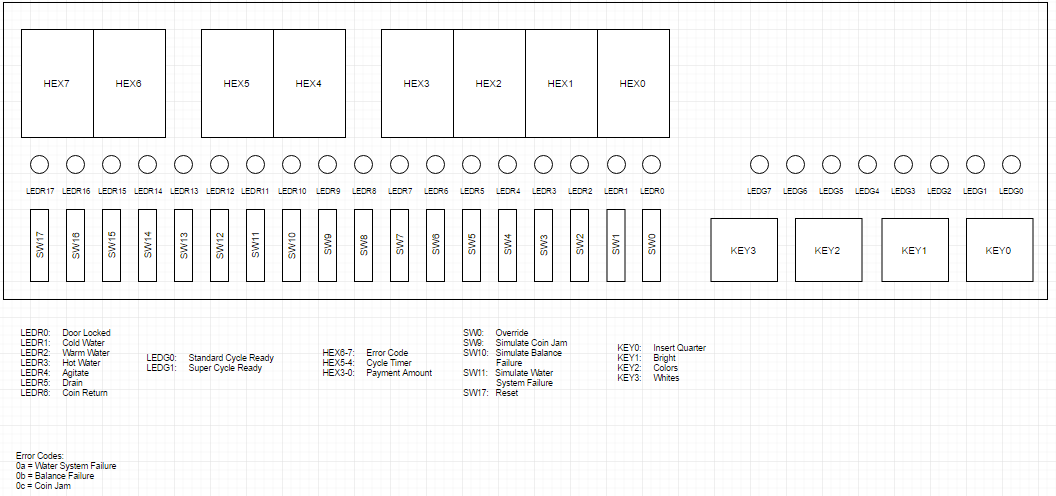


Figure 4 - User Interface Diagram

The washing machine control system is implemented in VHDL on an Altera DE2 board. For the purposes of simulation, the push-buttons and switches were used to represents inputs, and the LEDs and 7-segment displays were used to display outputs.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Description** | **Output** | **Description** |
| KEY0 | Insert Quarter | LEDG0 | Wash Ready |
| KEY1 | Brights Cycle | LEDG1 | SuperCycle Ready |
| KEY2 | Colors Cycle | LEDR0 | Door Lock |
| KEY3 | Whites Cycle | LERD1 | Cold Water |
| SW0 | User Override | LEDR2 | Warm Water |
| SW9 | Simulate Coin Jam | LEDR3 | Hot Water |
| SW10 | Simulate Water Level Error | LEDR4 | Agitate |
| SW11 | Simulate Balance Error | LEDR5 | Drain |
| SW17 | Reset | LEDR6 | Coin Return |
|  |  | HEX[3-0] | Payment Indicator |
|  |  | HEX[5-4] | Cycle Timer |
|  |  | HEX[7-6] | Error Code |

Table – Input/Output Descriptions

**V. Results / Design Evaluation**

The system’s complexity led to an implementation in VHDL on an Altera DE2 board rather than with 74x chips on a breadboard. The final design had a large number of input signals that lead to many branches between each state. The design met the criteria of graceful error recovery by allowing for a synchronous error checking process that could stop the machine and return to the starting state while refunding the user. Temperature and cycle control were implemented using multiple button inputs on the Altera board. Working with the onboard clock for timing proved to be difficult since some standard arithmetic operators (such as division or modulo) were not easily available. To overcome this, a lookup table was built for an appropriate range of values. Finally, the user override was implemented by allowing for a single switch to be thrown to jump the machine forward to the Spin stage, draining any unwanted water from the machine before exiting the cycle completely. An LED on the board was used to indicate if a refund would be given. A complete listing of the VHDL code can be found in Appendix A.

Over the course of the project, the focus of our objective design criteria shifted slightly (see Appendix B for the original requirements). Detection of laundry additives was removed as we noted that the washing machines which we were modeling do not require a control signal for their additives. We decided that a refund would be offered only if the machine encountered an error or during the payment phase before a cycle has started. Finally we removed the ability for the user to switch between different cycles after beginning a load as this would lead to either inconsistencies in the water temperature or ineffective use of resources by draining and refilling the compartment.

**Appendix A: VHDL Code Listing**

--

-- WashingMachineController.vhd

--

-- Top level component for the washing machine control system.

--

-- ELEE 204 Design Project

-- Kaleb Dekker, JD Elsey, David Fritts, Domenic Rodriguez

--

-- April 12, 2016

--

-- Import required libraries

library IEEE;

use IEEE.std\_logic\_1164.all;

-- WashingMachineController is the top level entity for the washer control system.

entity WashingMachineController is

port(CLOCK\_50 : in std\_logic;

KEY : in std\_logic\_vector(3 downto 0);

SW : in std\_logic\_vector(17 downto 0);

LEDR : out std\_logic\_vector(17 downto 0);

LEDG : out std\_logic\_vector(3 downto 0);

HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 : out std\_logic\_vector(0 to 6));

end;

-- Architecture definition for the washer control system

architecture WashingMachineController\_arch of WashingMachineController is

-- Decoder for the 7-segment hex display on the DE2 board

component SevenSegDecoder

port(hex : in std\_logic\_vector(3 downto 0);

display : out std\_logic\_vector(0 to 6));

end component;

type payment\_state\_type is (zero, twentyfive, fifty, seventyfive, onedollar, dollartwentyfive, coin\_jam);

-- The numbered states indicate how much money has been placed into the machine

-- coin\_jam is a state to handle a mechanical coin jam condition

type cycle\_state\_type is (cycle\_select, fill, wash, wash\_ext, rinse, spin, done, refund, error);

-- cycle\_select is the state where the machine waits for payment and for the user to select a cycle

-- fill is the state where the machine is filling with water

-- wash is the state where the machine "agitates" the load to clean the clothing

-- wash\_ext is an extended wash state for SuperCycle

-- rinse is the state where additional water is run through the machine

-- spin is the state where the water is spun out of the machine

-- coin\_jam is an error state to handle coin jams

-- refund is a state to handle user refunds

-- error is a general error state

type cycle\_type is (hot, warm, cold);

-- cycle\_type represents the different water temperature wash cycles

signal quarter, whites, colors, brights, override : std\_logic;

-- quarter represents a quarter input

-- whites represents selecting a hot water cycle

-- colors represents selecting a warm water cycle

-- brights represents selecting a cold water cycle

-- override represents a request from the user to stop the wash cycle

signal jam\_sensor, balance\_sensor, water\_sensor : std\_logic;

-- jam\_sensor is 1 when the system detects a coin jam, and 0 otherwise

-- balance\_sensor is 1 when the sensor detects the washer is out of balance, and 0 otherwise

-- water\_sensor is 1 when the system detects a water level related error, and 0 otherwise

signal reset : std\_logic;

-- reset is used to manually reset the system after entering an error state\_type

signal hot\_water, warm\_water, cold\_water, door\_lock, agitate, drain, coin\_return : std\_logic;

-- [hot/warm/cold]\_water are active high signals to fill the machine with the appropriate temperature water

-- door\_lock is '0' when the door is unlocked, and '1' when the door is locked

-- agitate is an active high signal that controls the motor during the wash state

-- drain is an active high signal to spin the water out from the machine

-- coin\_return is an active high signal that indicates funds should be returned to the user

signal payment\_state : payment\_state\_type := zero;

-- Represents the current state of the payment FSM

signal current\_cycle\_state, next\_cycle\_state : cycle\_state\_type;

-- cur\_state represents the current state of the washer cycle control FSM

-- next\_state represents the next state that should be transitioned to

signal selected\_cycle : cycle\_type;

-- selected\_cycle stores the information regarding the selected water type

signal hex0\_dat, hex1\_dat, hex2\_dat, hex3\_dat, hex4\_dat, hex5\_dat, hex6\_dat, hex7\_dat : std\_logic\_vector(3 downto 0);

-- hexN\_dat are signals to hold the data for the hex displays on the DE2 board

signal counter : integer := 0;

signal counter\_bits : std\_logic\_vector(7 downto 0);

signal reset\_counter: std\_logic;

begin

-- Assign names input signals

quarter <= KEY(0);

whites <= KEY(3);

colors <= KEY(2);

brights <= KEY(1);

override <= SW(0);

jam\_sensor <= SW(9);

balance\_sensor <= SW(10);

water\_sensor <= SW(11);

reset <= SW(17);

-- Instantiate the hex-display decoders

hex0\_decoder : SevenSegDecoder port map (hex0\_dat, HEX0);

hex1\_decoder : SevenSegDecoder port map (hex1\_dat, HEX1);

hex2\_decoder : SevenSegDecoder port map (hex2\_dat, HEX2);

hex3\_decoder : SevenSegDecoder port map (hex3\_dat, HEX3);

hex4\_decoder : SevenSegDecoder port map (hex4\_dat, HEX4);

hex5\_decoder : SevenSegDecoder port map (hex5\_dat, HEX5);

hex6\_decoder : SevenSegDecoder port map (hex6\_dat, HEX6);

hex7\_decoder : SevenSegDecoder port map (hex7\_dat, HEX7);

-- Handle quarter inputs

insert\_quarter: process (quarter, jam\_sensor, current\_cycle\_state, reset)

begin

-- Check for a coin jam

if reset = '1' then

payment\_state <= zero;

elsif jam\_sensor = '1' then

if current\_cycle\_state = cycle\_select then

payment\_state <= zero;

else

payment\_state <= payment\_state;

end if;

elsif current\_cycle\_state = done then

payment\_state <= zero;

else

-- No coin jam; check for a button press and advance the FSM

if quarter'event and quarter = '1' and current\_cycle\_state = cycle\_select and jam\_sensor = '0' then

case payment\_state is

when zero =>

payment\_state <= twentyfive;

when twentyfive =>

payment\_state <= fifty;

when fifty =>

payment\_state <= seventyfive;

when seventyfive =>

payment\_state <= onedollar;

when onedollar =>

payment\_state <= dollartwentyfive;

when dollartwentyfive =>

payment\_state <= dollartwentyfive;

when others =>

payment\_state <= zero;

end case;

end if;

end if;

end process insert\_quarter;

-- Update the hex display to show the payment state

update\_payment\_display: process (payment\_state)

begin

case payment\_state is

when zero =>

hex2\_dat <= x"0";

hex1\_dat <= x"0";

hex0\_dat <= x"0";

when twentyfive =>

hex2\_dat <= x"0";

hex1\_dat <= x"2";

hex0\_dat <= x"5";

when fifty =>

hex2\_dat <= x"0";

hex1\_dat <= x"5";

hex0\_dat <= x"0";

when seventyfive =>

hex2\_dat <= x"0";

hex1\_dat <= x"7";

hex0\_dat <= x"5";

when onedollar =>

hex2\_dat <= x"1";

hex1\_dat <= x"0";

hex0\_dat <= x"0";

when dollartwentyfive =>

hex2\_dat <= x"1";

hex1\_dat <= x"2";

hex0\_dat <= x"5";

when others =>

hex2\_dat <= x"0";

hex1\_dat <= x"0";

hex0\_dat <= x"0";

end case;

hex3\_dat <= x"0";

end process update\_payment\_display;

-- Figure out what the next state for the cycle FSM should be

determine\_next\_cycle\_state: process (whites, brights, colors, override, balance\_sensor, water\_sensor, reset, current\_cycle\_state, payment\_state, selected\_cycle, counter)

begin

case current\_cycle\_state is

when cycle\_select =>

-- Check if payment is sufficient

if payment\_state = twentyfive then

coin\_return <= '0';

end if;

if jam\_sensor = '1' then

next\_cycle\_state <= error;

elsif payment\_state = onedollar or payment\_state = dollartwentyfive then

-- Check if any of the cycle select buttons are pressed

if whites = '0' then

next\_cycle\_state <= fill;

selected\_cycle <= hot;

elsif colors = '0' then

next\_cycle\_state <= fill;

selected\_cycle <= warm;

elsif brights = '0' then

next\_cycle\_state <= fill;

selected\_cycle <= cold;

else

next\_cycle\_state <= cycle\_select;

selected\_cycle <= selected\_cycle;

end if;

else

next\_cycle\_state <= cycle\_select;

selected\_cycle <= selected\_cycle;

end if;

reset\_counter <= '1';

when fill =>

-- Fill the washer with water for 30 seconds

if counter > 1500000000 then

next\_cycle\_state <= wash;

reset\_counter <= '1';

else

if balance\_sensor = '1' OR water\_sensor = '1' then

next\_cycle\_state <= error;

reset\_counter <= '0';

elsif override = '1' then

next\_cycle\_state <= spin;

reset\_counter <= '1';

else

next\_cycle\_state <= fill;

reset\_counter <= '0';

end if;

end if;

selected\_cycle <= selected\_cycle;

when wash =>

-- Run the wash stage for 30 seconds

if counter > 1500000000 then

-- Check for a super cycle

if payment\_state = dollartwentyfive then

next\_cycle\_state <= wash\_ext;

else

next\_cycle\_state <= rinse;

end if;

reset\_counter <= '1';

else

if balance\_sensor = '1' OR water\_sensor = '1' then

next\_cycle\_state <= error;

elsif override = '1' then

next\_cycle\_state <= spin;

reset\_counter <= '0';

else

next\_cycle\_state <= wash;

end if;

reset\_counter <= '0';

end if;

selected\_cycle <= selected\_cycle;

when wash\_ext =>

-- Additional wash time for the super cycle

if counter > 1500000000 then

next\_cycle\_state <= rinse;

reset\_counter <= '1';

else

if balance\_sensor = '1' OR water\_sensor = '1' then

next\_cycle\_state <= error;

elsif override = '1' then

next\_cycle\_state <= spin;

reset\_counter <= '0';

else

next\_cycle\_state <= wash\_ext;

end if;

reset\_counter <= '0';

end if;

selected\_cycle <= selected\_cycle;

when rinse =>

-- Run the rinse cycle for 30 seconds

if counter > 1500000000 then

next\_cycle\_state <= spin;

reset\_counter <= '1';

else

if balance\_sensor = '1' OR water\_sensor = '1' then

next\_cycle\_state <= error;

elsif override = '1' then

next\_cycle\_state <= spin;

reset\_counter <= '0';

else

next\_cycle\_state <= rinse;

end if;

reset\_counter <= '0';

end if;

selected\_cycle <= selected\_cycle;

when spin =>

-- Spin out and drain the water from the machine

if counter > 1500000000 then

next\_cycle\_state <= done;

reset\_counter <= '1';

else

if balance\_sensor = '1' OR water\_sensor = '1' then

next\_cycle\_state <= error;

else

next\_cycle\_state <= spin;

end if;

reset\_counter <= '0';

end if;

selected\_cycle <= selected\_cycle;

when done =>

next\_cycle\_state <= cycle\_select;

reset\_counter <= '1';

selected\_cycle <= selected\_cycle;

when error =>

-- Handle error conditions

if reset = '1' then

next\_cycle\_state <= cycle\_select;

coin\_return <= '0';

else

next\_cycle\_state <= error;

coin\_return <= '1';

end if;

reset\_counter <= '1';

selected\_cycle <= selected\_cycle;

when others =>

-- Unknown state

next\_cycle\_state <= error;

reset\_counter <= '1';

selected\_cycle <= selected\_cycle;

end case;

end process determine\_next\_cycle\_state;

-- Show the value of the timer on the 7-segment displays

update\_cycle\_timer: process (counter)

begin

-- Brute force check the value of the counter, and set the 7-segment display

-- to appropriate values to count down from 30. Ugly, but it works.

if counter > 0 and counter <= 50000000 then

hex5\_dat <= x"3";

hex4\_dat <= x"0";

elsif counter > 50000000 and counter <= 100000000 then

hex5\_dat <= x"2";

hex4\_dat <= x"9";

elsif counter > 100000000 and counter <= 150000000 then

hex5\_dat <= x"2";

hex4\_dat <= x"8";

elsif counter > 150000000 and counter <= 200000000 then

hex5\_dat <= x"2";

hex4\_dat <= x"7";

elsif counter > 200000000 and counter <= 250000000 then

hex5\_dat <= x"2";

hex4\_dat <= x"6";

elsif counter > 250000000 and counter <= 300000000 then

hex5\_dat <= x"2";

hex4\_dat <= x"5";

elsif counter > 300000000 and counter <= 350000000 then

hex5\_dat <= x"2";

hex4\_dat <= x"4";

elsif counter > 350000000 and counter <= 400000000 then

hex5\_dat <= x"2";

hex4\_dat <= x"3";

elsif counter > 400000000 and counter <= 450000000 then

hex5\_dat <= x"2";

hex4\_dat <= x"2";

elsif counter > 450000000 and counter <= 500000000 then

hex5\_dat <= x"2";

hex4\_dat <= x"1";

elsif counter > 500000000 and counter <= 550000000 then

hex5\_dat <= x"2";

hex4\_dat <= x"0";

elsif counter > 550000000 and counter <= 600000000 then

hex5\_dat <= x"1";

hex4\_dat <= x"9";

elsif counter > 600000000 and counter <= 650000000 then

hex5\_dat <= x"1";

hex4\_dat <= x"8";

elsif counter > 650000000 and counter <= 700000000 then

hex5\_dat <= x"1";

hex4\_dat <= x"7";

elsif counter > 700000000 and counter <= 750000000 then

hex5\_dat <= x"1";

hex4\_dat <= x"6";

elsif counter > 750000000 and counter <= 800000000 then

hex5\_dat <= x"1";

hex4\_dat <= x"5";

elsif counter > 800000000 and counter <= 850000000 then

hex5\_dat <= x"1";

hex4\_dat <= x"4";

elsif counter > 850000000 and counter <= 900000000 then

hex5\_dat <= x"1";

hex4\_dat <= x"3";

elsif counter > 900000000 and counter <= 950000000 then

hex5\_dat <= x"1";

hex4\_dat <= x"2";

elsif counter > 950000000 and counter <= 1000000000 then

hex5\_dat <= x"1";

hex4\_dat <= x"1";

elsif counter > 1000000000 and counter <= 1050000000 then

hex5\_dat <= x"1";

hex4\_dat <= x"0";

elsif counter > 1050000000 and counter <= 1100000000 then

hex5\_dat <= x"0";

hex4\_dat <= x"9";

elsif counter > 1100000000 and counter <= 1150000000 then

hex5\_dat <= x"0";

hex4\_dat <= x"8";

elsif counter > 1150000000 and counter <= 1200000000 then

hex5\_dat <= x"0";

hex4\_dat <= x"7";

elsif counter > 1200000000 and counter <= 1250000000 then

hex5\_dat <= x"0";

hex4\_dat <= x"6";

elsif counter > 1250000000 and counter <= 1300000000 then

hex5\_dat <= x"0";

hex4\_dat <= x"5";

elsif counter > 1300000000 and counter <= 1350000000 then

hex5\_dat <= x"0";

hex4\_dat <= x"4";

elsif counter > 1350000000 and counter <= 1400000000 then

hex5\_dat <= x"0";

hex4\_dat <= x"3";

elsif counter > 1400000000 and counter <= 1450000000 then

hex5\_dat <= x"0";

hex4\_dat <= x"2";

elsif counter > 1450000000 and counter <= 1500000000 then

hex5\_dat <= x"0";

hex4\_dat <= x"1";

else

hex5\_dat <= x"0";

hex4\_dat <= x"0";

end if;

end process update\_cycle\_timer;

-- Handles the displaying of error codes on the hex7 and hex6 displays DF

display\_error\_code: process (current\_cycle\_state, balance\_sensor, water\_sensor)

begin

if current\_cycle\_state = cycle\_select then

hex7\_dat <= x"1";

hex6\_dat <= x"0";

elsif current\_cycle\_state = fill then

hex7\_dat <= x"2";

hex6\_dat <= x"0";

elsif current\_cycle\_state = wash then

hex7\_dat <= x"3";

elsif current\_cycle\_state = rinse then

hex7\_dat <= x"4";

elsif current\_cycle\_state = spin then

hex7\_dat <= x"5";

elsif current\_cycle\_state = wash\_ext then

hex7\_dat <= x"E";

elsif current\_cycle\_state = error or jam\_sensor = '1'then

hex7\_dat <= x"0";

if water\_sensor = '1' then

hex6\_dat <= x"A";

elsif balance\_sensor = '1' then

hex6\_dat <= x"B";

elsif jam\_sensor = '1' and current\_cycle\_state = cycle\_select then

hex6\_dat <= x"C";

else

hex6\_dat <= x"0";

end if;

end if;

end process display\_error\_code;

-- Handle updates that occur on the clock cycle

update\_clock: process (CLOCK\_50)

begin

if CLOCK\_50'event and CLOCK\_50 = '1' then

-- Move the cycle controller to the next state

current\_cycle\_state <= next\_cycle\_state;

-- Advance the counter

if reset\_counter = '1' then

counter <= 0;

else

counter <= counter + 1;

end if;

end if;

end process update\_clock;

-- Determine washer output signals

hot\_water <= '1' when (current\_cycle\_state = fill or current\_cycle\_state = rinse) and selected\_cycle = hot else '0';

warm\_water <= '1' when (current\_cycle\_state = fill or current\_cycle\_state = rinse) and selected\_cycle = warm else '0';

cold\_water <= '1' when (current\_cycle\_state = fill or current\_cycle\_state = rinse) and selected\_cycle = cold else '0';

door\_lock <= '0' when current\_cycle\_state = cycle\_select or

current\_cycle\_state = error or

current\_cycle\_state = done

else '1';

agitate <= '1' when current\_cycle\_state = wash or

current\_cycle\_state = wash\_ext

else '0';

drain <= '1' when current\_cycle\_state = spin else '0';

-- Show payment status

LEDG(0) <= '1' when payment\_state = onedollar or payment\_state = dollartwentyfive else '0';

LEDG(1) <= '1' when payment\_state = dollartwentyfive else '0';

-- Show washer controller outputs

LEDR(0) <= door\_lock;

LEDR(1) <= cold\_water;

LEDR(2) <= warm\_water;

LEDR(3) <= hot\_water;

LEDR(4) <= agitate;

LEDR(5) <= drain;

LEDR(6) <= coin\_return;

-- Display the cycle controller state

LEDR(17) <= '1' when current\_cycle\_state = cycle\_select else '0';

LEDR(16) <= '1' when current\_cycle\_state = fill else '0';

LEDR(15) <= '1' when current\_cycle\_state = wash else '0';

LEDR(14) <= '1' when current\_cycle\_state = wash\_ext else '0';

LEDR(13) <= '1' when current\_cycle\_state = rinse else '0';

LEDR(12) <= '1' when current\_cycle\_state = spin else '0';

LEDR(11) <= '1' when current\_cycle\_state = error else '0';

end WashingMachineController\_arch;

-- 7-segment decoder taken from ELEE 252 Lab 5

library IEEE;

use IEEE.std\_logic\_1164.all;

entity SevenSegDecoder is

port (hex : in std\_logic\_vector(3 downto 0);

display : out std\_logic\_vector(0 to 6));

END SevenSegDecoder;

architecture SevenSegDecoder\_arch of SevenSegDecoder is

begin

--

-- 0

-- ---

-- | |

-- 5| |1

-- | 6 |

-- ---

-- | |

-- 4| |2

-- | |

-- ---

-- 3

--

process (hex)

begin

-- a conditional VHDL statement

case hex is

when "0000" => display <= "0000001";

when "0001" => display <= "1001111";

when "0010" => display <= "0010010";

when "0011" => display <= "0000110";

when "0100" => display <= "1001100";

when "0101" => display <= "0100100";

when "0110" => display <= "0100000";

when "0111" => display <= "0001111";

when "1000" => display <= "0000000";

when "1001" => display <= "0001100";

when "1010" => display <= "0001000";

when "1011" => display <= "1100000";

when "1100" => display <= "0110001";

when "1101" => display <= "1000010";

when "1110" => display <= "0110000";

when others => display <= "0111000";

end case;

end process;

end SevenSegDecoder\_arch;

**Appendix B: Original Brainstorming / Design Criteria**

* Problem Statement: Due to heavy usage, the washing machines in Hopeman tend to break down often and aren't designed to handle any sort of error conditions.
* Objective Criteria:
  + Recognize and handle error statements
    - Notify when in error state
    - Recover from error state (ex. Drain water from washer if not done)
    - Require servicing in order to function again after error state is thrown.
    - Recognize varying errors during cycles.
  + Differentiate between different load cycles (3 distinct cycles and super cycles)
  + Note variance in additives
  + Cycle types:
    - Be able to switch between cycle types
    - Be able to refund the user given a certain time limit.
  + Clock to keep track of time.