ELEX 7660 – Digital System Design

COINSAVER 3000

Project Report

D. Retonel
J. Young
2017-02-14

Table of Contents

Table of Figures

1. Abstract

The CoinSaver 3000 counts the number of coins inputted into the shoots and displays the total value on a quad 7 segment LED display as shown in Figure 1, the maximum amount the LED can display is $99.99. The CoinSaver can also dispense the coins one at a time using servo motors attached to the bottom of the shoot. The servo motors are controlled by a 4x4 matrix keypad. When a designated key on the keypad is pressed, the corresponding servo will rotate 180 degrees pushing a single coin through a slit on the bottom of the shoot.

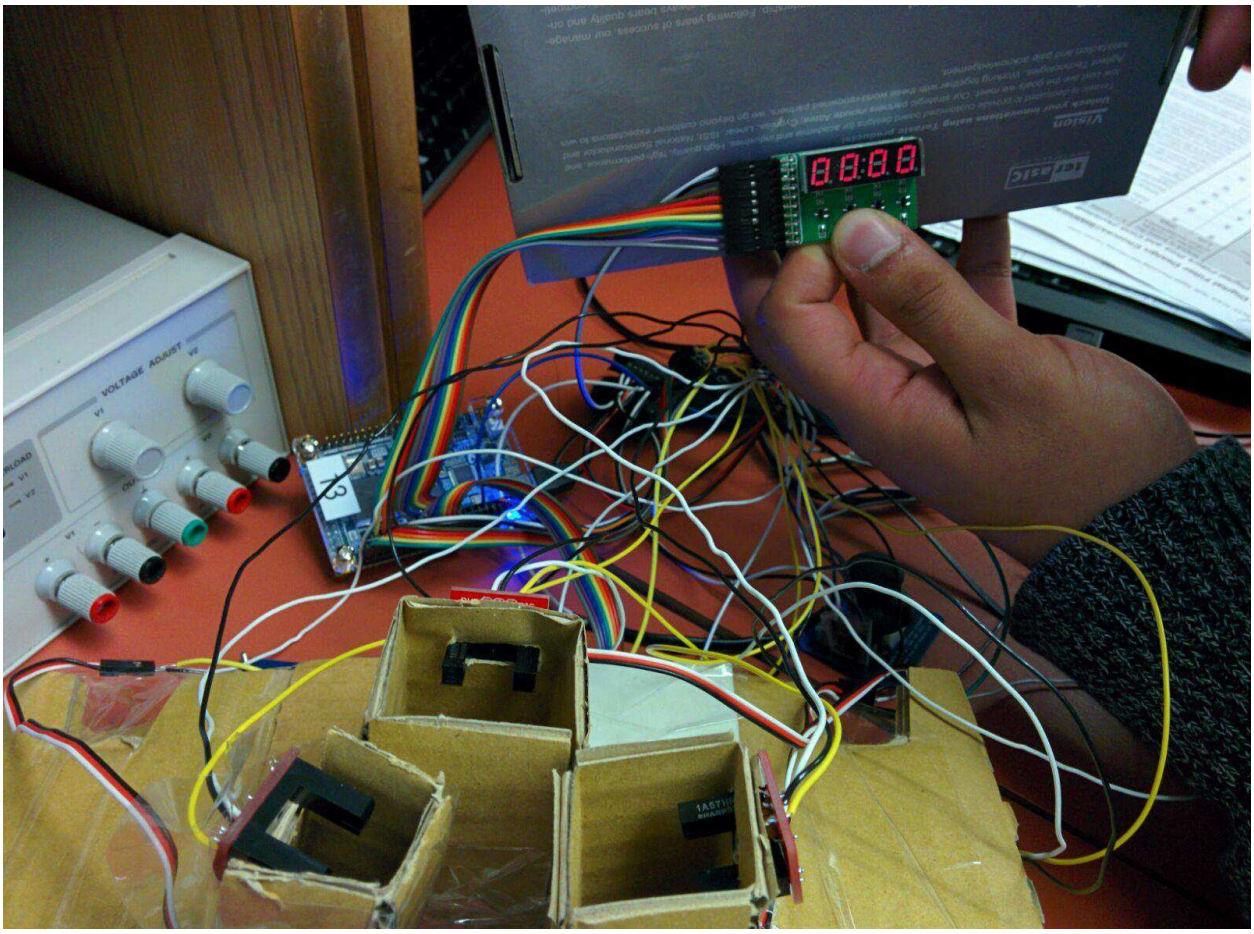

Figure 1: LED display of Value upon reset

2. Introduction

The CoinSaver 3000 is a digital piggy bank that tells you how much money is deposited into it and will also dispense coins one at a time. For this project, we only used $2.00 (toonies), $0.25 (quarters), and $0.10 (dimes) since we could only afford 3 photo interrupters and 3 servos with the budget given. The motivation for this project is our fascination of how a vending machine accepts the proper value in coins as payment as well as to find a way of organizing the change that we accumulate. The coins are inserted into the appropriate shoots and will pass through a photo interrupter. The photo interrupters are active-low, that is they output a high (3.3V VCC) when nothing is in between the gates but will give a low signal (~0.5V) when a coin passes through to the FPGA. The FPGA will then output the appropriate amount to the quad 7-segment display. The user can make a withdrawal simply by pressing the appropriate key on the keypad to dispense one coin at a time. The FPGA always outputs a PWM signal of ~2.2ms pulse width which corresponds to a counter clockwise motion [4]. Once the key is pressed the FPGA will output a ~0.7ms pulse width modulated signal which will create a clockwise motion [4]. The arm attached to the motor will then push the coin out through a slit on the bottom of the shoot. You must hold the button down long enough until the coin drops out (<1s).

3. Changes to proposal

Due to time constraints as well as issues with coding, the password needed to make a withdrawal was not included in the project. Dispensing of a set amount was also changed to dispensing a single coin from the shoot when the appropriate key on the keypad was pressed. The mechanical sort was also swapped out with a manual sort where the user inserts the coins into the appropriate shoot because the mechanical sorter needed to be designed and then 3D printed.

4. Overview

4.1. Manual Sort

The sorting mechanism for the coin counter was to simply separate the coins into the corresponding shoot. These shoots were made of cardboard and slits were cut into them for the photo interrupters to fit into. The original proposal was for a plastic sorter that sorts the coins by size. However, we failed to realize that this sorter was only available for 3D printing and could not be bought. Due to the nature of this project we believed that the actual sorting of the coin themselves was not as important as the underlying project at hand.

4.2. Photo Interrupters

In order to keep track of how many coins are inserted into the shoots we used photo-interrupters that were placed at the top of the shoot. The photo interrupter (GP1A57HRJ00F) were ordered from SparkFun along with the breakout board (Breakout Board - GP1A57HRJ00F). As the coin passes through the gate of the photo-interrupter it will output a ~0.5V signal to the FPGA. When nothing is between the gates it outputs VCC, in this case 3.3V, from the FPGA. On the datasheet, the photo interrupters specified that it needed a minimum of 5 volts but we experimented with 3.3V volts and found that it worked just fine. This greatly simplified the work need to be done since we were able to connect them directly to the FPGA board without the risk of damaging it. The photo interrupter was soldered onto the breakout board and 3 wires (VCC, GND, SIG) were also soldered onto each breakout board.

4.3. Universal Servos

Universal Servos were used as coin dispensers these can be found almost anywhere but where ordered from SparkFun (Servo - Generic (Sub-Micro Size) ROB-09065). The universal servos have ~180 degree of motion and will move counter clockwise when given a pulse train of ~2.2ms and will move clockwise when given a pulse train of ~0.7ms [1]. When the coin falls to the bottom of the shoot the servo-arm will rotate its full 180 degrees and push the coin out of a slit at the bottom of the shoot releasing the coin.

4.4. Quad 7-Segment Display

The 4-digit 7-segment display was supplied to us by our lab instructor. It is a simple 4-digit 7-segment display that is driven a 3.3V VCC. The display works through “a multiplexed display driver that enables each VCC pin in sequence and sinks the current on the segments that should be lit” [2]. Since the clock is running fast enough the display looks like it is always on. The code used in the modules were previously written for a lab in this course [2].

4.5. 4x4 Matrix Keypad

As with the quad 7-segment Display the 4x4 matrix keypad was supplied to us by our lab instructor. The keypad “contains SPST switches at intersection of rows and column lines, switch matrix decoder scans the rows and columns to determine which switch is closed” [3]. The code used in the modules were also used in a previous lab [3].

4.6. Wiring

All wiring was done through a wiring harness that was supplied by our lab instructor. This connects all components of this project together. The Wiring diagram can be found in figure [1]. Since this project only included the 4x4 matrix keypad and the quad 7-segment display we disconnected the RBG OLED display and used those pins for in the input from the photo-interrupters and output to the servo motors.

5. Modules

5.1. coinsavertop.sv

5.1.1. Code

```
// Filename:            coinsavertop.sv
// Author:              Dan Retonel, Josh Young
// Date:                9 April 2017
// Modified from: lab1.sv by Ed Casas 2017-1-9
//
// Top level module of CoinSaver 3000. Instatiates modules and describes
// the operation of CoinSaver's two modes: count and dispense.
// Currently, a keypress on the first 3 rows of the keypad will control
// 1 of 3 servo motors

module coinsavertop (
      input logic CLOCK_50,         // 50 MHz clock
      input logic rst_n,            // active-low reset
      input logic keymode_n,        // active-low mode key
      input logic [2:0] coins,      // array of coin signals
      input logic [3:0] kpr,        // keypad row signals
      output logic [3:0] kpc,       // keypad column signals
      output logic [7:0] leds,      // 7-seg LED cathodes
      output logic [3:0] ct,        // digit enable
      output logic [2:0] sctrl,     // servo control signals
      output logic pwmod ) ;

      // operation enumeration of 2 modes: count and dispense
      enum logic {
            count,
            dispense
      } op;

      logic [1:0] digit;      // 2-bit digit value mapped to 7-seg
      logic [3:0] money;      // converted tens/ones/dimes/cents to display
      logic [15:0] value;     // 16-bit value of coins counted
      logic [3:0] digits;     // 4-bit digits to display based on op
      logic [3:0] num;        // 4-bit digit of keypad press

      logic enable;           // pwm enable
      logic clk;              // system clock, wizard generated below
      logic kphit;            // keypad pressed flag
      logic count_en;         // count enable


      int mode = 1'b0;        // integer to determine mode

      // instantiate modules

      decode2 decode2_0 (.digit,.ct);
      decode7 decode7_0 (.num(digits),.leds);
      coincount coincount_0  (.rst_n,.count_en,.coins,.clk, .value);
      count2money count2money_0 (.digit,.value,.money);
      pwm pwm_0 (.clk,.rst_n,.enable,.pwmod);
      colseq colseq_0 (.reset_n(rst_n),.clk,.kpr,.kpc);
      kpdecode kpdecode_0 (.kpr,.kpc(kpc),.kphit,.num);
```

```
      always_ff @(posedge clk)  begin

            // toggle mode on keypress
            if (!keymode_n)
                  mode <= ~mode;

            // is count ? rotate ct for 7-seg display and enable count
            if (op == count) begin
                  digit <= digit + 1'b1 ;
                  count_en <= 1'b1;
                  end
            // is dispense ? show keypress on right most LED,
            //                        disable count, enable pwm, poll keypad
for servo ctrl

            else if (op == dispense) begin
                  digit <= '0;
                  enable <= 1'b1;
                  count_en <= 1'b0;
                  unique case (kpr)
                        4'b0111: sctrl[0] <= pwmod;
                        4'b1011: sctrl[1] <= pwmod;
                        4'b1101: sctrl[2] <= pwmod;
                  endcase
            end

      end

      always_comb begin
            // in counting mode ? display money : display keypresses
            if (mode) begin
                  op = count;
                  digits = money;
            end
            else begin
                  digits = num;
                  op = dispense;
            end
      end

   lab1clk lab1clk_0 ( CLOCK_50, clk ) ;

endmodule

// megafunction wizard: %ALTPLL%
// ...
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
// ...

module lab1clk ( inclk0, c0);

        input     inclk0;
        output    c0;
```

```
wire [0:0] sub_wire2 = 1'h0;
wire [4:0] sub_wire3;
wire  sub_wire0 = inclk0;
wire [1:0] sub_wire1 = {sub_wire2, sub_wire0};
wire [0:0] sub_wire4 = sub_wire3[0:0];
wire  c0 = sub_wire4;

altpll altpll_component ( .inclk (sub_wire1), .clk
  (sub_wire3), .activeclock (), .areset (1'b0), .clkbad
  (), .clkena ({6{1'b1}}), .clkloss (), .clkswitch
  (1'b0), .configupdate (1'b0), .enable0 (), .enable1 (),
  .extclk (), .extclkena ({4{1'b1}}), .fbin (1'b1),
  .fbmimicbidir (), .fbout (), .fref (), .icdrclk (),
  .locked (), .pfdena (1'b1), .phasecounterselect
  ({4{1'b1}}), .phasedone (), .phasestep (1'b1),
  .phaseupdown (1'b1), .pllena (1'b1), .scanaclr (1'b0),
  .scanclk (1'b0), .scanclkena (1'b1), .scandata (1'b0),
  .scandataout (), .scandone (), .scanread (1'b0),
  .scanwrite (1'b0), .sclkout0 (), .sclkout1 (),
  .vcooverrange (), .vcounderrange ());

defparam
        altpll_component.bandwidth_type = "AUTO",
        altpll_component.clk0_divide_by = 25000,
        altpll_component.clk0_duty_cycle = 50,
        altpll_component.clk0_multiply_by = 1,
        altpll_component.clk0_phase_shift = "0",
        altpll_component.compensate_clock = "CLK0",
        altpll_component.inclk0_input_frequency = 20000,
        altpll_component.intended_device_family = "Cyclone IV E",
        altpll_component.lpm_hint = "CBX_MODULE_PREFIX=lab1clk",
        altpll_component.lpm_type = "altpll",
        altpll_component.operation_mode = "NORMAL",
        altpll_component.pll_type = "AUTO",
        altpll_component.port_activeclock = "PORT_UNUSED",
        altpll_component.port_areset = "PORT_UNUSED",
        altpll_component.port_clkbad0 = "PORT_UNUSED",
        altpll_component.port_clkbad1 = "PORT_UNUSED",
        altpll_component.port_clkloss = "PORT_UNUSED",
        altpll_component.port_clkswitch = "PORT_UNUSED",
        altpll_component.port_configupdate = "PORT_UNUSED",
        altpll_component.port_fbin = "PORT_UNUSED",
        altpll_component.port_inclk0 = "PORT_USED",
        altpll_component.port_inclk1 = "PORT_UNUSED",
        altpll_component.port_locked = "PORT_UNUSED",
        altpll_component.port_pfdena = "PORT_UNUSED",
        altpll_component.port_phasecounterselect = "PORT_UNUSED",
        altpll_component.port_phasedone = "PORT_UNUSED",
        altpll_component.port_phasestep = "PORT_UNUSED",
        altpll_component.port_phaseupdown = "PORT_UNUSED",
        altpll_component.port_pllena = "PORT_UNUSED",
        altpll_component.port_scanaclr = "PORT_UNUSED",
        altpll_component.port_scanclk = "PORT_UNUSED",
        altpll_component.port_scanclkena = "PORT_UNUSED",
        altpll_component.port_scandata = "PORT_UNUSED",
        altpll_component.port_scandataout = "PORT_UNUSED",
```

```
                altpll_component.port_scandone = "PORT_UNUSED",
                altpll_component.port_scanread = "PORT_UNUSED",
                altpll_component.port_scanwrite = "PORT_UNUSED",
                altpll_component.port_clk0 = "PORT_USED",
                altpll_component.port_clk1 = "PORT_UNUSED",
                altpll_component.port_clk2 = "PORT_UNUSED",
                altpll_component.port_clk3 = "PORT_UNUSED",
                altpll_component.port_clk4 = "PORT_UNUSED",
                altpll_component.port_clk5 = "PORT_UNUSED",
                altpll_component.port_clkena0 = "PORT_UNUSED",
                altpll_component.port_clkena1 = "PORT_UNUSED",
                altpll_component.port_clkena2 = "PORT_UNUSED",
                altpll_component.port_clkena3 = "PORT_UNUSED",
                altpll_component.port_clkena4 = "PORT_UNUSED",
                altpll_component.port_clkena5 = "PORT_UNUSED",
                altpll_component.port_extclk0 = "PORT_UNUSED",
                altpll_component.port_extclk1 = "PORT_UNUSED",
                altpll_component.port_extclk2 = "PORT_UNUSED",
                altpll_component.port_extclk3 = "PORT_UNUSED",
                altpll_component.width_clock = 5;


endmodule
```

5.2. count2money.sv

5.2.1. Code

```
// Filename:        count2money.sv
// Name:            Dan Retonel
//
// Date:            2 April 2017
// Converts 2-bit digit value and 16 bit coin value into 4 corresponding
digits
// of monetary value

module count2money(
      input logic [1:0] digit,      // 2-bit digit value
      input logic [15:0] value,     // 16-bit coin value
      output logic [3:0] money      // 4-bit output value
);

      always_comb begin
            unique case (digit)
                  2'b00: money = value % 10;          // cents
                  2'b01: money = value % 100 / 10;    // dimes
                  2'b10: money = value % 1000 / 100;  // dollars
                  2'b11: money = value / 1000;        // tens
            endcase
      end

endmodule
```

5.3. coincount.sv

5.3.1. Code

```
// Filename:      coincount.sv
// Name:          Dan Retonel,      Josh Young
// Date:          1 April 2017
//
// Accepts signals from photo interrupters and outputs the corresponding
// value in cents. Value is only counted when a enable signal is high.
// Resets value to zero on active low reset.

module coincount(
      input logic rst_n,            // active-low reset
      input logic count_en,         // count enable
      input logic [2:0] coins,      // array of coin signals
      input logic clk,              // system clock
      output logic [15:0] value     // monetary values in cents
);

      parameter TOONIE  = 8'd200;
      parameter QUARTER       = 8'd25;
      parameter DIME          = 8'd10;
      parameter INTERVALC     = 1000;

      enum logic {
            count,
            hold
      }     op;                     // operation logic: count or hold value

      int timer = INTERVALC;        // limit coin count with delays

      // while clocking, decrement counter.
      // if in counting operation, poll photointerrupters at delayed times
      always_ff @(posedge clk) begin

            timer <= timer - 1;

            if (!rst_n) begin
                  value <= '0;
                  timer <= INTERVALC;
            end
            else if (op == count && timer <= 0)

                  // check photointerrupters, reset timer on succesful count
                  unique case (coins)
                        3'b011:     begin       value <= value  + TOONIE;
                                                timer <= INTERVALC;
                                    end
                        3'b101:     begin       value <= value  + QUARTER;
                                                timer <= INTERVALC;
                                    end
                        3'b110:     begin       value <= value  + DIME;
                                                timer <= INTERVALC;
                                    end
                        3'b111: value <= value;
                  endcase
```

```
	end

	// count only if it is enabled
	always_comb begin
		if (count_en)
			op = count;
		else
			op = hold;
	end

endmodule
```

5.4. pwm.sv

5.4.1. Code

```
// Filename:      pmw.sv
// Name:          Dan Retonel,      Josh Young
// Date:          9 April 2017
//
// Creates a pulse-width modulated signal to contol servo motors.
// Uses a counting scheme that will output high for a specific
// value of count to produce appropriate pulse width.

module pwm (
      input clk,              // system clock; 1kHz
      input rst_n,            // active low reset
      input enable,           // pulse width out of 256 bits
      output pwmod            // output waveform
);


      parameter MAX_COUNT = 8'd40;  // highest value of count
      logic [7:0] count = 8'd0;           // 8-bit logic count
      logic [7:0] compare;                // 8-bit logic to compare count

      always_ff @(posedge clk) begin
            if (!rst_n)
                  count <= 8'd0;
            else begin
                  count <= count + 1'b1;
                  if (count >= MAX_COUNT)
                        count <= 8'd0;

                  if (count < compare)
                        pwmod = 1'b1;
                  else
                        pwmod = 1'b0;
            end
      end

      // produce 2 types pulse widths: 1 for CCW and 1 for CW rotation
      always_comb begin
            if (enable)
                  compare = 8'd2;
            else
                  compare = 8'd5;
      end
endmodule
```

5.5. decode2.sv

5.5.1. Code

```
// Filename:            decode2.sv
// Name:                Dan Retonel
//
// Date:                16 Jan 2017
// Converts 2-bit values into 4 1-bit active-high VCC enables
// i.e. 2-to-4 bit decoder
// Controls 4 transistors that supply current for the LEDs

module decode2(
      input logic [1:0] digit,      // 2-bit input value
      output logic [3:0] ct         // 4-bit output value
);

      always_comb begin
            unique case (digit)
                  2'b00: ct = 4'b0001;    // 00 -> 0001
                  2'b01: ct = 4'b0010;    // 01 -> 0010
                  2'b10: ct = 4'b0100;    // 10 -> 0100
                  2'b11: ct = 4'b1000;    // 11 -> 1000
            endcase
      end

endmodule
```

5.6. decode7.sv

5.6.1. Code

```
// Filename:        decode7.sv
// Name:            Dan Retonel
//
// Date:            23 Jan 2017
// Converts 4-bit input value into an active low 7-seg LED
// encoded output

module decode7(
      input logic [3:0] num,                // 4-bit input number
      output logic [7:0] leds               // 8-bit LED output
);
      always_comb begin
            unique case (num)
                  4'h0: leds = 8'b1100_0000;          // 0
                  4'h1: leds = 8'b1111_1001;          // 1
                  4'h2: leds = 8'b1010_0100;          // 2
                  4'h3: leds = 8'b1011_0000;          // 3
                  4'h4: leds = 8'b1001_1001;          // 4
                  4'h5: leds = 8'b1001_0010;          // 5
                  4'h6: leds = 8'b1000_0010;          // 6
                  4'h7: leds = 8'b1111_1000;          // 7
                  4'h8: leds = 8'b1000_0000;          // 8
                  4'h9: leds = 8'b1001_0000;          // 9
            endcase
      end

endmodule
```

5.7. colseq.sv

5.7.1. Code

```
// Filename:      colseq.sv
// Name:          Dan Retonel
// Date:          23 Jan 2017
//
// Sequences through keypad column output states on each rising edge of a
clock.
// Resets the column output to 0111 on an active low reset and holds state if
// an acitve low keypad row input is detected.

module colseq(
      input logic       reset_n,          // active low sync reset
      input logic       clk,              // clock
      input logic       [3:0] kpr,        // 4-bit keypad row input
      output logic      [3:0] kpc         // 4-bit keypad column output
);

      enum logic {
            scan,       // scan operation
            hold        // hold operation
      } op;

      always_ff @(posedge clk)
            if (!reset_n)                 // reset kpc on active low reset
                  kpc <= 4'b0111;
            else if (op == scan)          // scan operation -> sequence
                  case (kpc)
                        4'b0111: kpc <= 4'b1011;
                        4'b1011: kpc <= 4'b1101;
                        4'b1101: kpc <= 4'b1110;
                        4'b1110: kpc <= 4'b0111;
                  endcase

            else
                  kpc <= kpc;             // hold

      always_comb begin
            if (kpr != 4'b1111)           // hold on row input signal
                  op = hold;
            else
                  op = scan;
      end

endmodule
```

5.8. kpdecode.sv

5.8.1. Code

```
// Filename:        kpdecode.sv
// Name:            Dan Retonel
// Date:            23 Jan 2017
//
// Outputs 4-bit value of key press by decoding the keypad row and column
// inputs. Also outputs kphit logic to drive least significant digit of LED
// display.

module kpdecode(
      input logic [3:0] kpc,  // 4-bit keypad column value
      input logic [3:0] kpr,  // 4-bit keypad row value
      output logic      kphit,      // high when key pressed
      output logic [3:0] num  // 4-bit value being pressed
);

      always_comb begin

            // Nested cases to give num appropriate value.
            // Keypad mapping can be found in Lab 2 Manual

            unique case (kpc)
                  4'b1110: case (kpr)
                              4'b1110: num = 4'hd;
                              4'b1101: num = 4'hc;
                              4'b1011: num = 4'hb;
                              4'b0111: num = 4'ha;
                              default     num = 4'h0;
                        endcase
                  4'b1101: case (kpr)
                              4'b1110: num = 4'hf;
                              4'b1101: num = 4'h9;
                              4'b1011: num = 4'h6;
                              4'b0111: num = 4'h3;
                              default     num = 4'h0;
                        endcase
                  4'b1011: case (kpr)
                              4'b1110: num = 4'h0;
                              4'b1101: num = 4'h8;
                              4'b1011: num = 4'h5;
                              4'b0111: num = 4'h2;
                              default     num = 4'h0;
                        endcase
                  4'b0111: case (kpr)
                              4'b1110: num = 4'he;
                              4'b1101: num = 4'h7;
                              4'b1011: num = 4'h4;
                              4'b0111: num = 4'h1;
                              default     num = 4'h0;
                        endcase
            endcase

            // kphit enable
```

```
            if (kpc == 4'b1111 || kpr == 4'b1111)
                  kphit = 0;
            else
                  kphit = 1;
      end
endmodule
```

6. Pin Assignments

6.1. Coinsaver.qsf

```
set_location_assignment PIN_R8 -to CLOCK_50
set_location_assignment PIN_A15 -to LED[0]
set_location_assignment PIN_A13 -to LED[1]
set_location_assignment PIN_B13 -to LED[2]
set_location_assignment PIN_A11 -to LED[3]
set_location_assignment PIN_D1 -to LED[4]
set_location_assignment PIN_F3 -to LED[5]
set_location_assignment PIN_B1 -to LED[6]
set_location_assignment PIN_L3 -to LED[7]
# set_location_assignment PIN_J15 -to KEY[0]

set_location_assignment PIN_J15 -to rst_n

set_location_assignment PIN_E1 -to keymode_n
set_location_assignment PIN_M1 -to SW[0]
set_location_assignment PIN_T8 -to SW[1]
set_location_assignment PIN_B9 -to SW[2]
set_location_assignment PIN_M15 -to SW[3]
set_location_assignment PIN_P2 -to DRAM_ADDR[0]
set_location_assignment PIN_N5 -to DRAM_ADDR[1]
set_location_assignment PIN_N6 -to DRAM_ADDR[2]
set_location_assignment PIN_M8 -to DRAM_ADDR[3]
set_location_assignment PIN_P8 -to DRAM_ADDR[4]
set_location_assignment PIN_T7 -to DRAM_ADDR[5]
set_location_assignment PIN_N8 -to DRAM_ADDR[6]
set_location_assignment PIN_T6 -to DRAM_ADDR[7]
set_location_assignment PIN_R1 -to DRAM_ADDR[8]
set_location_assignment PIN_P1 -to DRAM_ADDR[9]
set_location_assignment PIN_N2 -to DRAM_ADDR[10]
set_location_assignment PIN_N1 -to DRAM_ADDR[11]
set_location_assignment PIN_L4 -to DRAM_ADDR[12]
set_location_assignment PIN_M7 -to DRAM_BA[0]
set_location_assignment PIN_M6 -to DRAM_BA[1]
set_location_assignment PIN_L7 -to DRAM_CKE
set_location_assignment PIN_R4 -to DRAM_CLK
set_location_assignment PIN_P6 -to DRAM_CS_N
set_location_assignment PIN_G2 -to DRAM_DQ[0]
set_location_assignment PIN_G1 -to DRAM_DQ[1]
set_location_assignment PIN_L8 -to DRAM_DQ[2]
set_location_assignment PIN_K5 -to DRAM_DQ[3]
set_location_assignment PIN_K2 -to DRAM_DQ[4]
set_location_assignment PIN_J2 -to DRAM_DQ[5]
set_location_assignment PIN_J1 -to DRAM_DQ[6]
set_location_assignment PIN_R7 -to DRAM_DQ[7]
set_location_assignment PIN_T4 -to DRAM_DQ[8]
set_location_assignment PIN_T2 -to DRAM_DQ[9]
set_location_assignment PIN_T3 -to DRAM_DQ[10]
set_location_assignment PIN_R3 -to DRAM_DQ[11]
set_location_assignment PIN_R5 -to DRAM_DQ[12]
set_location_assignment PIN_P3 -to DRAM_DQ[13]
set_location_assignment PIN_N3 -to DRAM_DQ[14]
set_location_assignment PIN_K1 -to DRAM_DQ[15]
```

```
set_location_assignment PIN_R6 -to DRAM_DQM[0]
set_location_assignment PIN_T5 -to DRAM_DQM[1]
set_location_assignment PIN_L1 -to DRAM_CAS_N
set_location_assignment PIN_L2 -to DRAM_RAS_N
set_location_assignment PIN_C2 -to DRAM_WE_N
set_location_assignment PIN_F2 -to I2C_SCLK
set_location_assignment PIN_F1 -to I2C_SDAT
set_location_assignment PIN_G5 -to G_SENSOR_CS_N
set_location_assignment PIN_M2 -to G_SENSOR_INT
set_location_assignment PIN_A14 -to GPIO_2[0]
set_location_assignment PIN_B16 -to GPIO_2[1]
set_location_assignment PIN_C14 -to GPIO_2[2]
set_location_assignment PIN_C16 -to GPIO_2[3]
set_location_assignment PIN_C15 -to GPIO_2[4]
set_location_assignment PIN_D16 -to GPIO_2[5]
set_location_assignment PIN_D15 -to GPIO_2[6]
set_location_assignment PIN_D14 -to GPIO_2[7]
set_location_assignment PIN_F15 -to GPIO_2[8]
set_location_assignment PIN_F16 -to GPIO_2[9]
set_location_assignment PIN_F14 -to GPIO_2[10]
set_location_assignment PIN_G16 -to GPIO_2[11]
set_location_assignment PIN_G15 -to GPIO_2[12]
set_location_assignment PIN_E15 -to GPIO_2_IN[0]
set_location_assignment PIN_E16 -to GPIO_2_IN[1]
set_location_assignment PIN_M16 -to GPIO_2_IN[2]
set_location_assignment PIN_A8 -to sctrl[0]
set_location_assignment PIN_D3 -to GPIO_0[0]
set_location_assignment PIN_B8 -to sctrl[1]
set_location_assignment PIN_C3 -to GPIO_0[1]
#set_location_assignment PIN_A2 -to count_en
set_location_assignment PIN_A3 -to GPIO_0[3]
set_location_assignment PIN_B3 -to sctrl[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to sctrl[0]

set_location_assignment PIN_B4 -to GPIO_0[5]
set_location_assignment PIN_A4 -to GPIO_0[6]
set_location_assignment PIN_B5 -to GPIO_0[7]
set_location_assignment PIN_A5 -to GPIO_0[8]
set_location_assignment PIN_D5 -to GPIO_0[9]
set_location_assignment PIN_B6 -to GPIO_0[10]
set_location_assignment PIN_A6 -to GPIO_0[11]
set_location_assignment PIN_B7 -to GPIO_0[12]
set_location_assignment PIN_D6 -to GPIO_0[13]
set_location_assignment PIN_A7 -to GPIO_0[14]
set_location_assignment PIN_C6 -to GPIO_0[15]
set_location_assignment PIN_C8 -to GPIO_0[16]
set_location_assignment PIN_E6 -to GPIO_0[17]
set_location_assignment PIN_E7 -to GPIO_0[18]
set_location_assignment PIN_D8 -to GPIO_0[19]
set_location_assignment PIN_E8 -to GPIO_0[20]
set_location_assignment PIN_F8 -to GPIO_0[21]
set_location_assignment PIN_F9 -to GPIO_0[22]
set_location_assignment PIN_E9 -to GPIO_0[23]
set_location_assignment PIN_C9 -to GPIO_0[24]
set_location_assignment PIN_D9 -to coins[0]
set_location_assignment PIN_E11 -to GPIO_0[26]
```

```
set_location_assignment PIN_E10 -to coins[1]
set_location_assignment PIN_C11 -to GPIO_0[28]
set_location_assignment PIN_B11 -to coins[2]
set_location_assignment PIN_A12 -to GPIO_0[30]
set_location_assignment PIN_D11 -to sctrl[1]
set_location_assignment PIN_D12 -to GPIO_0[32]
set_location_assignment PIN_B12 -to sctrl[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to sctrl[2]
set_location_assignment PIN_T9 -to GPIO_1_IN[0]
set_location_assignment PIN_F13 -to GPIO_1[0]
set_location_assignment PIN_R9 -to GPIO_1_IN[1]
set_location_assignment PIN_T15 -to GPIO_1[1]
set_location_assignment PIN_T14 -to GPIO_1[2]
set_location_assignment PIN_T13 -to GPIO_1[3]
set_location_assignment PIN_R13 -to GPIO_1[4]
set_location_assignment PIN_T12 -to GPIO_1[5]
set_location_assignment PIN_R12 -to GPIO_1[6]
set_location_assignment PIN_T11 -to GPIO_1[7]
set_location_assignment PIN_T10 -to GPIO_1[8]
set_location_assignment PIN_R11 -to GPIO_1[9]
set_location_assignment PIN_P11 -to GPIO_1[10]
set_location_assignment PIN_R10 -to GPIO_1[11]
set_location_assignment PIN_N12 -to GPIO_1[12]
set_location_assignment PIN_P9 -to GPIO_1[13]
set_location_assignment PIN_N9 -to GPIO_1[14]
set_location_assignment PIN_N11 -to GPIO_1[15]
set_location_assignment PIN_L16 -to GPIO_1[16]
set_location_assignment PIN_K16 -to GPIO_1[17]
set_location_assignment PIN_R16 -to GPIO_1[18]
set_location_assignment PIN_L15 -to GPIO_1[19]
set_location_assignment PIN_P15 -to GPIO_1[20]
set_location_assignment PIN_P16 -to GPIO_1[21]
set_location_assignment PIN_R14 -to GPIO_1[22]
set_location_assignment PIN_N16 -to GPIO_1[23]
set_location_assignment PIN_N15 -to GPIO_1[24]
set_location_assignment PIN_P14 -to GPIO_1[25]
set_location_assignment PIN_L14 -to GPIO_1[26]
set_location_assignment PIN_N14 -to GPIO_1[27]
set_location_assignment PIN_M10 -to GPIO_1[28]
set_location_assignment PIN_L13 -to GPIO_1[29]
set_location_assignment PIN_J16 -to GPIO_1[30]
set_location_assignment PIN_K15 -to GPIO_1[31]
set_location_assignment PIN_J13 -to GPIO_1[32]
set_location_assignment PIN_J14 -to GPIO_1[33]

set_location_assignment PIN_A2 -to qspb
set_location_assignment PIN_A8 -to qsa
set_location_assignment PIN_B8 -to qsb
set_location_assignment PIN_A12 -to ct[0]
set_location_assignment PIN_A5 -to leds[0]
set_location_assignment PIN_C11 -to ct[1]
set_location_assignment PIN_B6 -to leds[1]
set_location_assignment PIN_E11 -to ct[2]
set_location_assignment PIN_B7 -to leds[2]
set_location_assignment PIN_C9 -to ct[3]
set_location_assignment PIN_A7 -to leds[3]
```

```
set_location_assignment PIN_C8 -to leds[4]
set_location_assignment PIN_E7 -to leds[5]
set_location_assignment PIN_E8 -to leds[6]
set_location_assignment PIN_F9 -to leds[7]

set_location_assignment PIN_D5 -to kpr[3]
set_location_assignment PIN_A6 -to kpr[2]
set_location_assignment PIN_D6 -to kpr[1]
set_location_assignment PIN_C6 -to kpr[0]
set_location_assignment PIN_E6 -to kpc[0]
set_location_assignment PIN_D8 -to kpc[1]
set_location_assignment PIN_E9 -to kpc[3]
set_location_assignment PIN_F8 -to kpc[2]

set_location_assignment PIN_D9 -to rgb_din
set_location_assignment PIN_E10 -to rgb_clk
set_location_assignment PIN_B11 -to rgb_cs
set_location_assignment PIN_D11 -to sctrl[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to sctrl[1]


set_location_assignment PIN_G15 -to jstk_sel
set_location_assignment PIN_A10 -to adc_cs_n
set_location_assignment PIN_B10 -to adc_saddr
set_location_assignment PIN_A9 -to adc_sdat
set_location_assignment PIN_B14 -to adc_sclk

set_location_assignment PIN_D12 -to point

set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to jstk_sel
```

6.2. Pin Layout

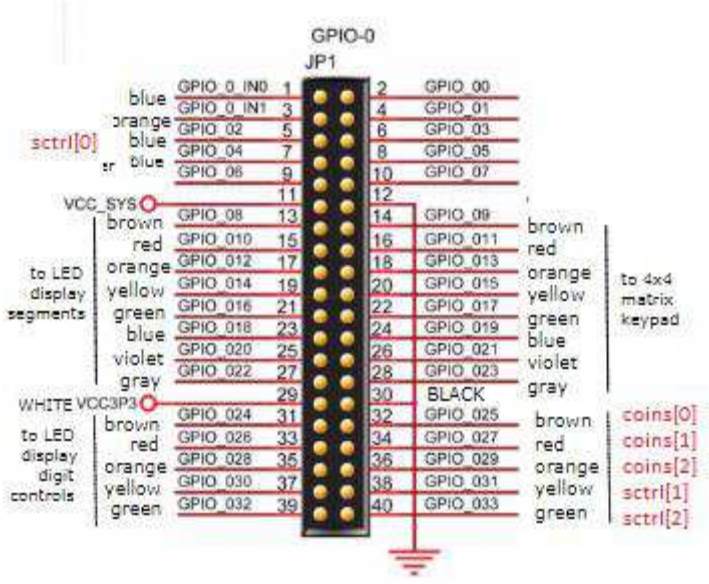

Figure 2: GPIO-0 I/O Pinout with Corresponding Signals

7. Schematics

7.1. Top Level Schematic

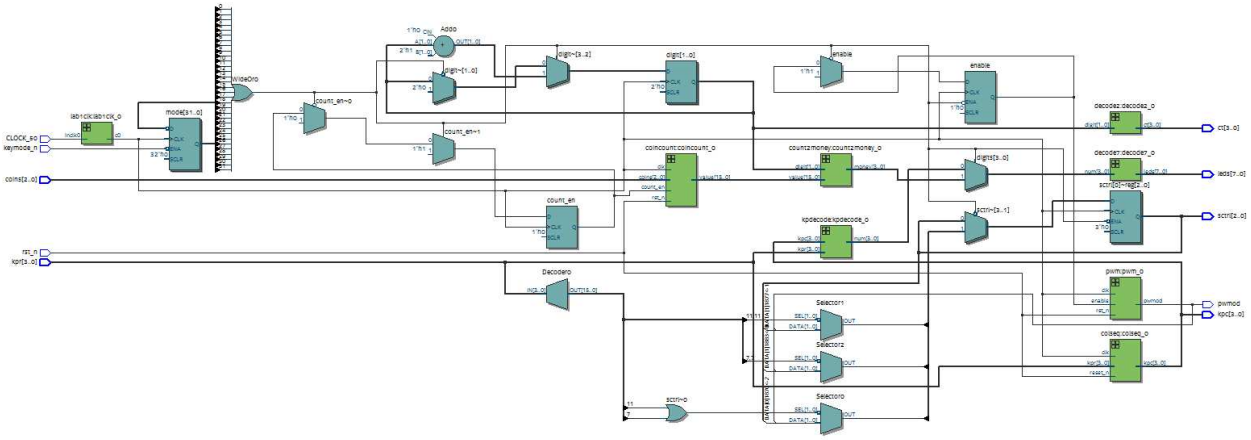

Figure 3: coinsavertop.sv – Top Level Module Schematic

7.2. Design Schematics

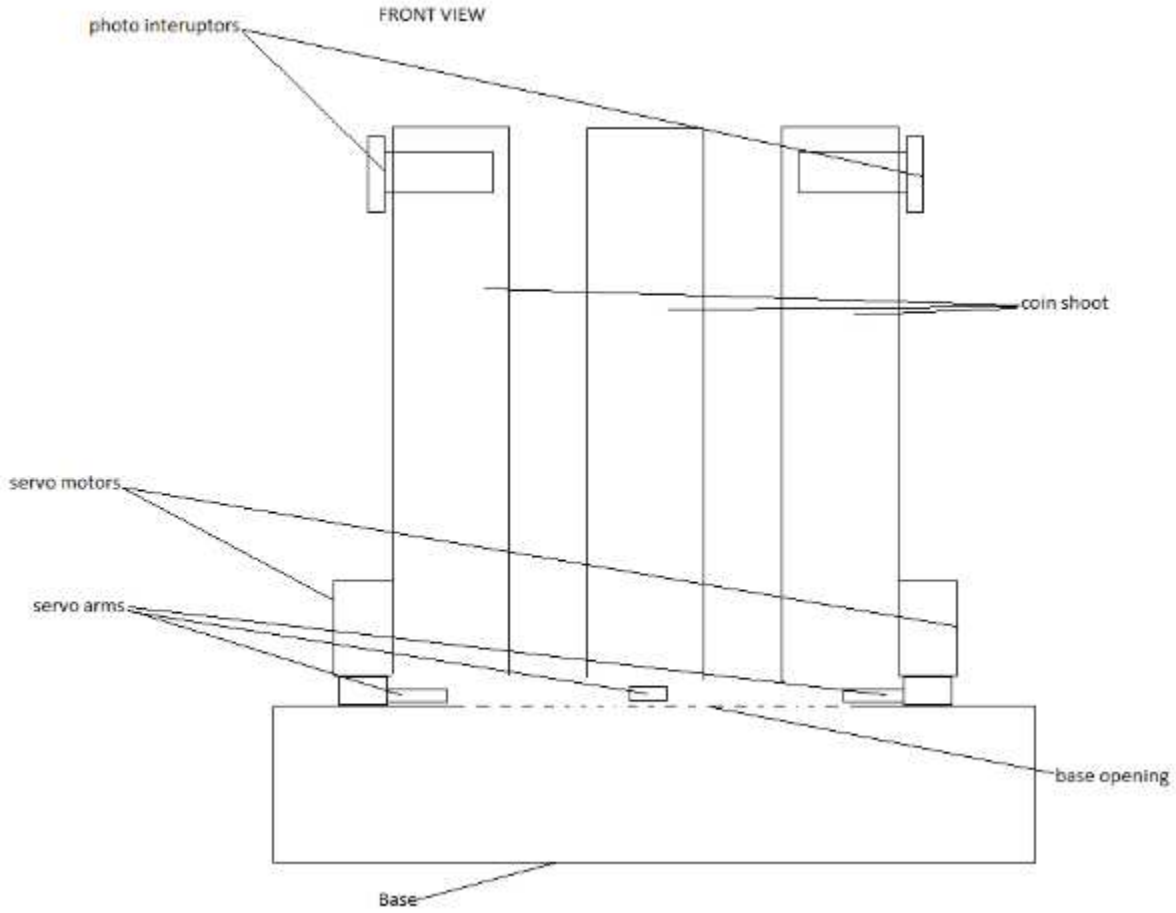

Figure 4: Front View Schematic

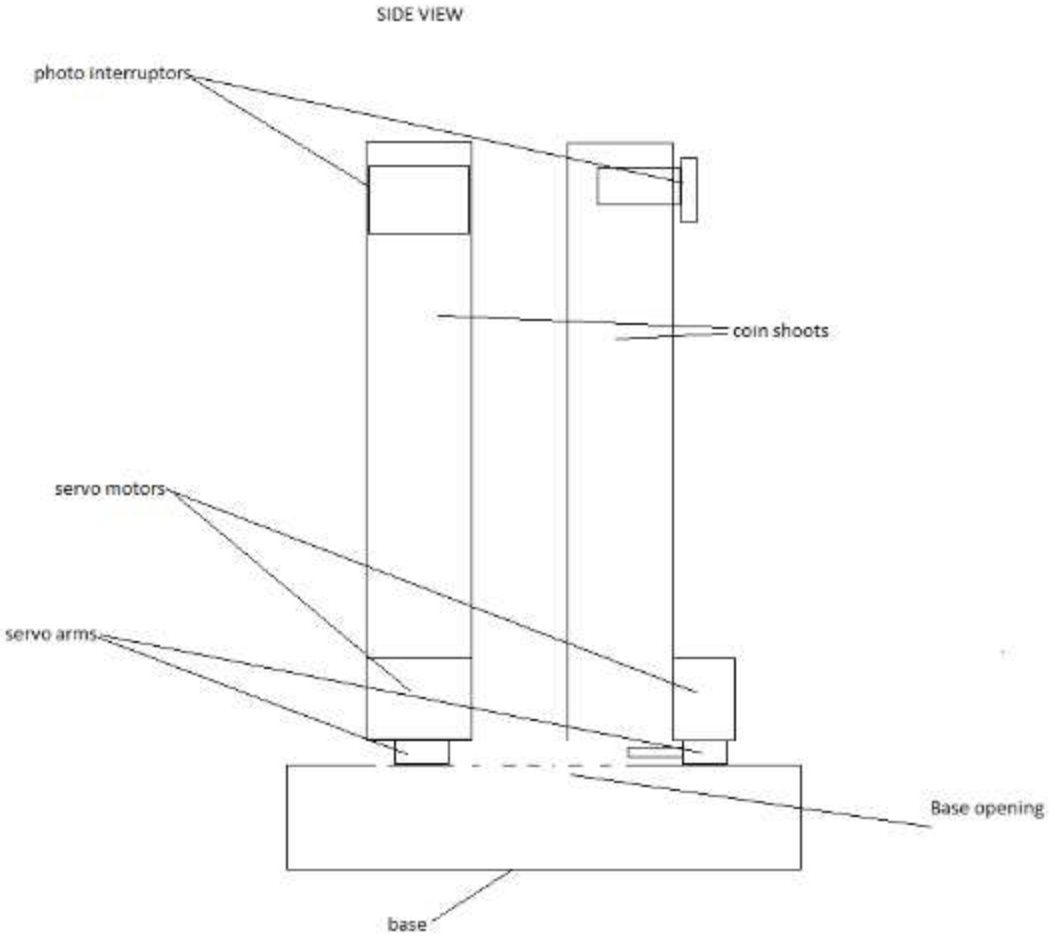

Figure 5: Side View Schematic

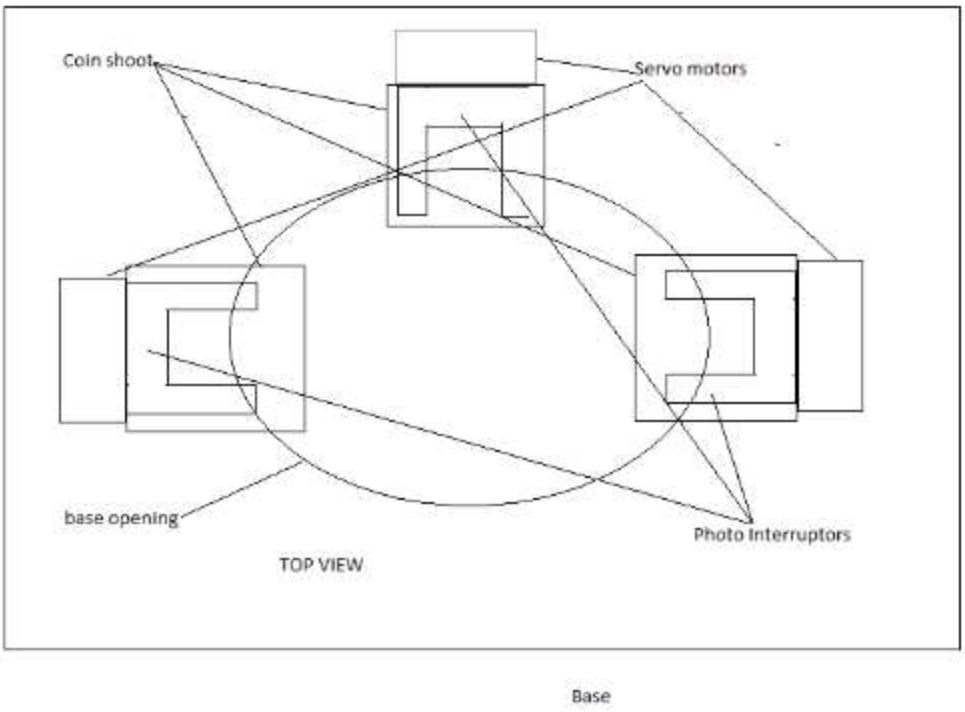

Figure 6: Top View Schematic

8. Compilation Report

Flow Summary

<<Filter>>

Flow Status	Successful - Mon Apr 17 03:21:06 2017
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	coinsaver
Top-level Entity Name	coinsavertop
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	1,108 / 22,320 (5 %)
Total registers	68
Total pins	30 / 154 (19 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	1 / 4 (25 %)

Figure 7: Compilation Report of CoinSaver 3000

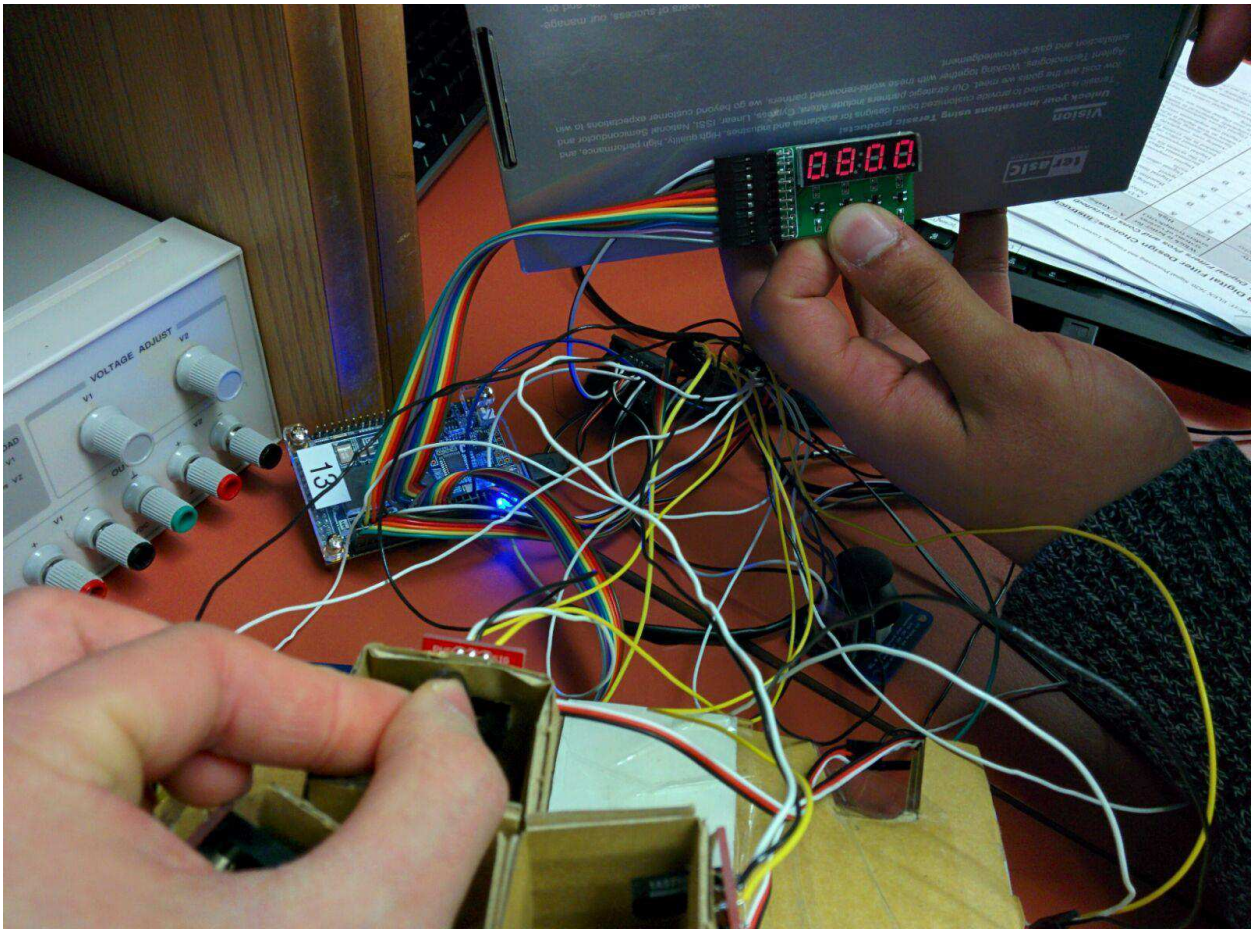

Figure 8: LED display showing $08.00 after inserting 4 toonies

9. Conclusion/Suggestions for Future Work

The motivation for this project was that our interest in the workings of vending machines and a way to organize the loose change that we accumulate. The completed project was very rudimentary. However, given more time as well as a larger budget, many more improvements could be accomplished. The simplest improvements can be made by implementing an automatic mechanical sorter and adding more photo interrupters allowing all coin denominations to be counted. Furthermore, adding a timer on the output on the servos would also let the user to only push the button once instead of holding it down until the coin drops out. More advanced additions would be to have a password in order to make a withdrawal and also an input for the withdrawal amount that the FPGA would dispense automatically without needing to push a button.

10. References

[1] B. J. MIKEGRUSIN, "Hobby Servo Tutorial," [Online]. Available: https://learn.sparkfun.com/tutorials/hobby-servo-tutorial. [Accessed 2017].

[2] E. Cases, "Lab 1 7-Segment LED Decoder," 2017. [Online]. Available: https://learn.bcit.ca/d2l/le/content/372579/viewContent/2228872/View. [Accessed 2017].

[3] E. Casas, "Lab 2 Matrix Keypad Decoder," 2017. [Online]. Available: https://learn.bcit.ca/d2l/le/content/372579/viewContent/2243842/View. [Accessed 2017].

[4] E. Casas, "Lab 0 Lab Wiring Harness," 2017. [Online]. Available: https://learn.bcit.ca/d2l/le/content/372579/viewContent/2228871/View. [Accessed 2017].

11. Appendix

Link to parts

Photo Interrupters

https://www.SparkFun.com/products/9299

Photo Interrupter break out board

https://www.SparkFun.com/products/9322

Micro Universal Servo

https://www.SparkFun.com/products/9065

Coin Sorter

http://www.thingiverse.com/thing:499177