ENGR 378 Digital Systems Design

Exp. 4: Multi-code Converter Module

Submitted by:

Björn Franzén
Kristian Lauszus

Oct. 9, 2014

Demonstration……………………..…..	____/5
Report: Presentation …………...	____/2
Problem analysis……….………….	____/2
Design work……………………….	____/5
Results………………………..………..	____/5
Conclusion/Discussion……………..…….	____/1
Total…………………………..…….	____/ 20

Problem Analysis

For this lab we had to describe and synthesize a multi-code converter module.

Hardware Design

Our approach was to create a top module that decided which the current mode is and routed the output from two different sub-modules to the output of the display. These sub-modules implements the three different modes outlined in the assignment. Before being routed to the display, the modules output was routed through a look-up table to convert binary to the appropriate form for the 7-segment displays.

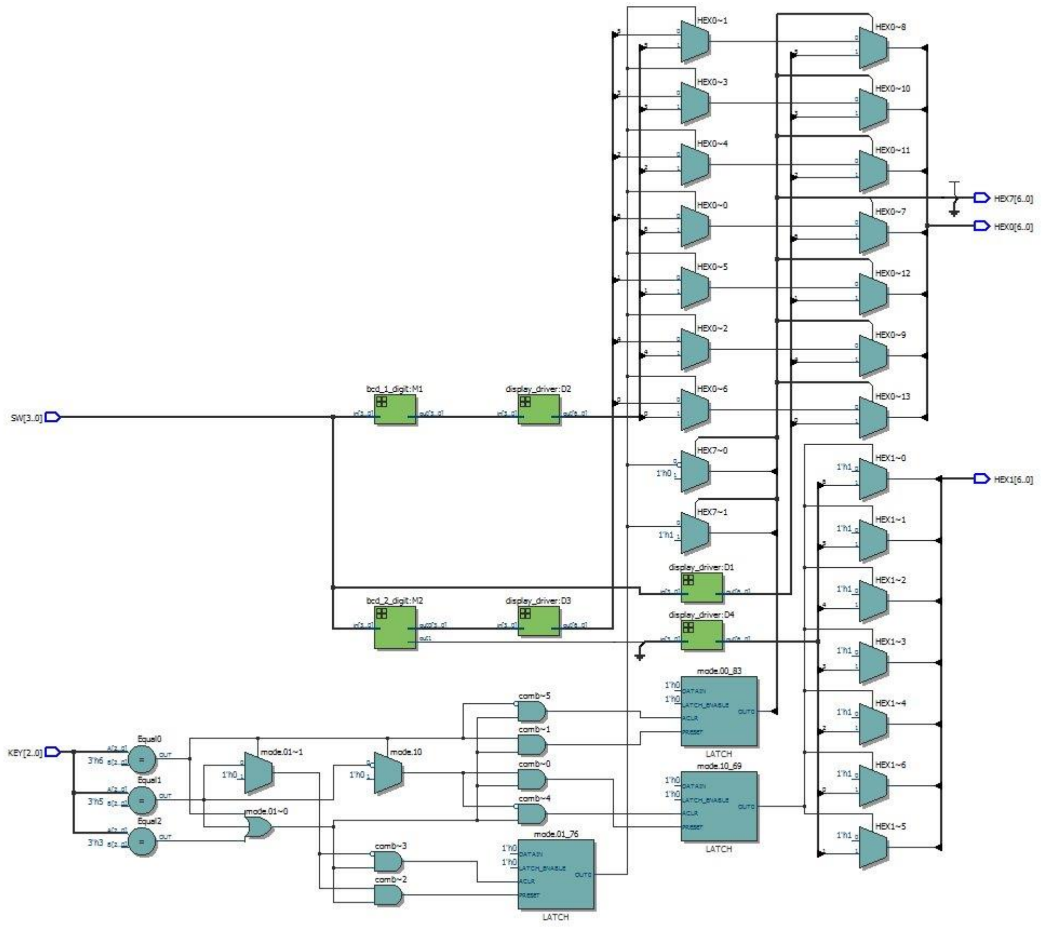

Figure 1 - Hardware synthesis of the design. .

Verilog Modelling

We implemented the modules for the binary to decimal and binary to BCD in the same fashion, that is we used an always statement with the appropriate if statements. The hex converter didn't need a module, the way we implemented it.

In the top design we used an always statement to remember the value of the push keys so that we didn't need to have them pressed down during conversion. This always statement will be synthesised into a latch, in this case desirable.

The look-up table simply converts a 4-bit number into a 7-bit number that can be written directly into the display. This look-up table is used for all the conversion functions.

Results/Verification

In our test bench we used a nested for loop to generate signals to go through the different modes. In the inner for loop we went through all the possible values of the four switches. The outcome of this is shown in the figure below.

After we verified our test bench we uploaded the design to the FPGA through JTAG and verified the design once again.

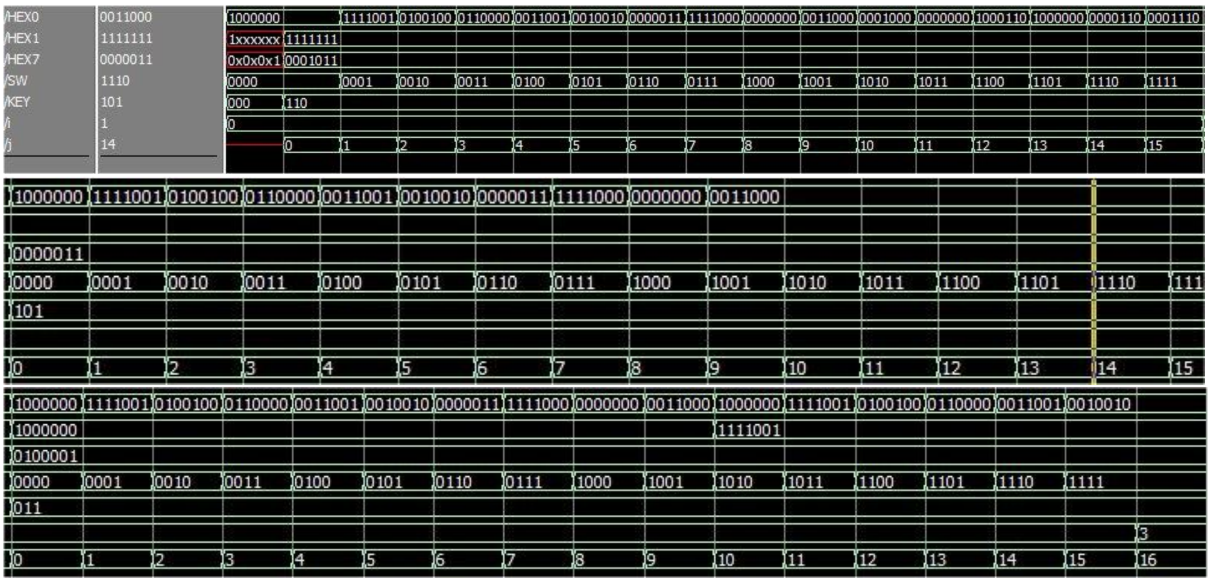

Figure 2 - Simulation waveform of the design..

Conclusions and Discussion

Implementing the latched function of the keys caused a few problems as we needed to figure out a solution that synthesised into what we actually wanted. At first we tried to latch all the outputs but then the problem was that the output only got updated when the keys changed. Instead we used key as an argument to an always statement that affected a register to retain which mode the design was in. This mode was then used to assign the different outputs to the displays to the outputs from the display driver modules.

Work Breakdown

All the assignment was created by a joint effort.

HDL Source Code

```
module code_converter(
                output [6:0] HEX0, HEX1, HEX7, // 7-segment displays
                input[3:0] SW, // Input switches
                input[2:0] KEY // Input buttons
        );
        wire[6:0] disp0, disp1, disp2, disp3; // Stores the output from the
differet display modules
        reg[1:0] mode; // This is used to remember which mode we are in
        wire[3:0] bcd_1_out, bcd_2_out0; // This is the output from the BCD
digit modules
        wire bcd_2_out1; // This is the second output from the BCD 2 digit
module

        display_driver D1(disp0, SW); // HEX driver can just use the
display driver directly

        bcd_1_digit M1(bcd_1_out, SW); // BCD to 1 digit driver
        display_driver D2(disp1, bcd_1_out); // Display driver for the BCD
to 1 digit driver

        bcd_2_digit M2(bcd_2_out0, bcd_2_out1, SW); // BCD to 2 digit
driver
        display_driver D3(disp2, bcd_2_out0); // Display driver for the
first display
        display_driver D4(disp3, {3'b000, bcd_2_out1}); // Display driver
for the second display

        assign HEX1 = mode == 2'b10 ? disp3 : ~7'h0; // If the third button
is pressed we output the value from the bcd_2_digit module, if not we
simply turn off the display
        assign HEX0 = mode == 2'b00 ? disp0 : mode == 2'b01 ? disp1 :
disp2; // We output the differet modules outputs depending on which mode we
are in i.e. which button was pressed last time

        // This control the display on the left which indicates which mode
we are in
        assign HEX7 = mode == 2'b00 ? 7'b0001011 : // 'h
                                         mode == 2'b01 ? 7'b0000011 : // 'b
                                         7'b0100001; // 'd'

        always@(*) begin
                // This if-statement check if a button is pressed and
updates the mode - the mode is latched, so it remembers it's value
                if (KEY == ~3'b001) // First button is pressed
                        mode = 2'b00; // Mode 0
                else if (KEY == ~3'b010) // Second button is pressed
                        mode = 2'b01; // Mode 1
                else if (KEY == ~3'b100) // Third button is pressed
                        mode = 2'b10; // Mode 2
        end
endmodule

module t_code_converter;
        wire [6:0] HEX0, HEX1, HEX7; // 7-segment displays
        reg[3:0] SW; // Input switches
        reg[2:0] KEY; // Input buttons

        integer i, j;
        code_converter M1(.HEX0(HEX0), .HEX1(HEX1), .HEX7(HEX7), .SW(SW),
.KEY(KEY));

        initial begin
                SW = 0; KEY = 0; // Set initial values
        end

        always begin
                for (i = 0; i < 3; i = i + 1) begin
                        #10 KEY = ~(1 << i); // Select mode
                        for (j = 0; j < 16; j = j + 1) begin
                                SW = j; // Set switches
                                #10; // Wait 10 units after the switches is
set
                        end
                end
                #10 $stop;
        end

endmodule

module bcd_2_digit(
                output reg[3:0] out0,
                output reg out1,
                input[3:0] in
        );

        always @ (in) begin
                if (in > 9) begin
                        out1 = 1'b1; // Output '1' on the second display
                        out0 = in - 4'd10; // Subtract 10 from the value
                end
                else begin
                        out1 = 1'b0; // Output '0' on the second display
                        out0 = in; // Output the input directly on the first
display
                end
        end
endmodule

module bcd_1_digit(
                output reg[3:0] out,
                input[3:0] in
        );

        always @ (in) begin
                if (in > 9)
                        out = 9; // Show '9' if the value is above 9
                else
                        out = in; // Output the input directly on the
display
        end
endmodule

module display_driver(
                output reg[6:0] out,
                input [3:0] in
        );

        always @ (in) begin
                case (in)
```

```
                        4'b0000 : out = 7'b1000000; // 0
                        4'b0001 : out = 7'b1111001; // 1
                        4'b0010 : out = 7'b0100100; // 2
                        4'b0011 : out = 7'b0110000; // 3
                        4'b0100 : out = 7'b0011001; // 4
                        4'b0101 : out = 7'b0010010; // 5
                        4'b0110 : out = 7'b0000011; // 6
                        4'b0111 : out = 7'b1111000; // 7
                        4'b1000 : out = 7'b0000000; // 8
                        4'b1001 : out = 7'b0011000; // 9

                        4'b1010 : out = 7'b0001000; // A
                        4'b1011 : out = 7'b0000000; // B (8)
                        4'b1100 : out = 7'b1000110; // C
                        4'b1101 : out = 7'b1000000; // D (0)
                        4'b1110 : out = 7'b0000110; // E
                        4'b1111 : out = 7'b0001110; // F

                        default : out = 7'b1111111; // OFF
                endcase
        end

endmodule
```