# ECE2072 Assignment Report

Name: Jeremy Vasic, Student Id: 27771458
Lab: 2pm Tuesday, Submission Date: 22/10/2019

**Part 1:**

- **1.1:** Explain how your design has made the time duration spent displaying your last count value 100 millisecond.
  - **The design triggers a reset of the timer when 132 seconds is reached. This is achieved by comparing the value of the "Timer" modules output "oTime_100msec16" with the value 132 (in binary just easier visualisation).**
  - **When this condition is met the timer is reset. Thus the final value displayed is 132, instead of 100.**

- **1.2:** Include pretty printed Verilog just for your assign2019 and MyPart1 module – do not include other modules. How to "pretty print" is described on Moodle.

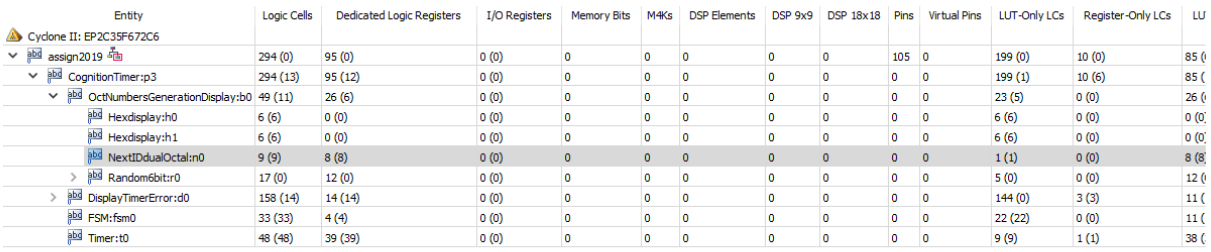
```
module Assign2019(CLOCK_50, KEY, SW, LEDR, LEDG, HEX0, HEX1, HEX2, HEX3,
HEX4, HEX5, HEX6, HEX7);
    input CLOCK_50;
    input [3:0] KEY;
    input [17:0] SW;
    output [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7;
    output [17:0] LEDR;
    output [7:0] LEDG;

    assign LEDG[3:0] = ~KEY[3:0];
    assign LEDR[17:0] = SW[17:0];
    //: Note:
    //: I am using "//:" to easily differentiate my own comments

  MyPart1 part1(
      .iClk(CLOCK_50),
      .iRst(~KEY[0]),
      .iErrorCodes(SW[17:16]),
      .iEnableDisplay(SW[15]),
      .iFreezeDisplay(SW[14]),
      .oHEX0(HEX0[6:0]),
      .oHEX1(HEX1[6:0]),
      .oHEX2(HEX2[6:0]),
      .oHEX3(HEX3[6:0]),
      .oHEX6(HEX6[6:0]),
      .oHEX7(HEX7[6:0])
  );  //: Part 1 Legend:
      //: SW[17:16]=error codes, SW[15]=enable, SW[14]=freeze output,
KEY[0]=reset,


endmodule

module MyPart1(iClk, iRst, iErrorCodes, iEnableDisplay, iFreezeDisplay,
oHEX0, oHEX1, oHEX2, oHEX3, oHEX6, oHEX7);
    input iClk, iRst, iEnableDisplay, iFreezeDisplay;
    input [1:0] iErrorCodes;
    output [6:0] oHEX0, oHEX1, oHEX2, oHEX3, oHEX6, oHEX7;
    wire [15:0] timer_Output; //: Stores the timer100ms value
    reg reset_IDlimit;

    //: Resesting display at timer limit
    //: My student ID#277714{58} -> 158sec timer limit
    always@(posedge iClk) begin
        if (timer_Output >= 16'd158) //: Checks whether the timer has
reached 158 seconds
            reset_IDlimit = 1'b1; //: if it has hit the limit, reset the
timer
        else
            reset_IDlimit = 1'b0;
    end

    //: Instantiating Timer Modules
    Timer func0(
        .iClk(iClk),
        .iRst(iRst),
        .iRstCE(reset_IDlimit),
        .oTime_100msec16(timer_Output)
    );
    DisplayTimerError func1(
        .iClk(iClk),
        .iSWEntryError(iErrorCodes[1:0]),
        .iEnable(iEnableDisplay),
        .iFreezeDisplayTimer(iFreezeDisplay),
        .iTimer_100msec(timer_Output),
        .oHEX0(oHEX0[6:0]),
        .oHEX1(oHEX1[6:0]),
        .oHEX2(oHEX2[6:0]),
        .oHEX3(oHEX3[6:0])
    );

    //: Setting the first two hex digits to my ID#
        assign oHEX7[6:0] = 7'b0010010; //:5 Replacing P1 with my student
ID#277714{58}
        assign oHEX6[6:0] = 7'b0000000; //:8

endmodule
```

- Include a file named MyPart1.sof (rename the file that Quartus produces for download to the FPGA).
    - **Done**

**Part 2:**

- **2.1:** How many states are there in a minimal state diagram for the FSM generated by a good solution for the NextIDdualOctal module?
    - **There are 8 states, one for each unique ID digit.**

- **2.2:** How many flip-flops should a minimal solution for module NextIDdualOctal generate? How many does yours generate?
    - **As there are $2^3$ states for this design the minimum number of flip-flops needed is 3. The number of flips-flops generated by my design is 8, this may have been because when I compiled it, I was not using user encoded settings.**



| Entity | Logic Cells | Dedicated Logic Registers | I/O Registers | Memory Bits | M4Ks | DSP Elements | DSP 9x9 | DSP 18x18 | Pins | Virtual Pins | LUT-Only LCs | Register-Only LCs | LU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cyclone II: EP2C35F672C6 | | | | | | | | | | | | | |
| assign2019 | 294 (0) | 95 (0) | 0 (0) | 0 | 0 | 0 | 0 | 0 | 105 | 0 | 199 (0) | 10 (0) | 85 ( |
| CognitionTimer:p3 | 294 (13) | 95 (12) | 0 (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 199 (1) | 10 (6) | 85 ( |
| OctNumbersGenerationDisplay:b0 | 49 (11) | 26 (6) | 0 (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 23 (5) | 0 (0) | 26 ( |
| Hexdisplay:h0 | 6 (6) | 0 (0) | 0 (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 (6) | 0 (0) | 0 (0 |
| Hexdisplay:h1 | 6 (6) | 0 (0) | 0 (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 (6) | 0 (0) | 0 (0 |
| NextIDdualOctal:n0 | 9 (9) | 8 (8) | 0 (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 (1) | 0 (0) | 8 (8 |
| Random6bit:r0 | 17 (0) | 12 (0) | 0 (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 (0) | 0 (0) | 12 ( |
| DisplayTimerError:d0 | 158 (14) | 14 (14) | 0 (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 144 (0) | 3 (3) | 11 ( |
| FSM:fsm0 | 33 (33) | 4 (4) | 0 (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 22 (22) | 0 (0) | 11 ( |
| Timer:t0 | 48 (48) | 39 (39) | 0 (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9 (9) | 1 (1) | 38 ( |

- **2.3:** Include pretty printed Verilog for your assign2019 and NextIDdualOctal modules – do not include other modules since they are not to be modified.

```
module Assign2019(CLOCK_50, KEY, SW, LEDR, LEDG, HEX0, HEX1, HEX2, HEX3,
HEX4, HEX5, HEX6, HEX7);
    input CLOCK_50;
    input [3:0] KEY;
    input [17:0] SW;
    output [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7;
    output [17:0] LEDR;
    output [7:0] LEDG;

    assign LEDG[3:0] = ~KEY[3:0];
    assign LEDR[17:0] = SW[17:0];
    //: Note:
    //: I am using "//:" to easily differentiate my own comments

  OctNumbersGenerationDisplay part2(
      .iClk(~KEY[3]),
      .iRst(~KEY[0]),
      .iBlankDisplay(0),
      .iNewOctNumsReq(SW[0]),
      .iChooseRandID(SW[5]),
      .oDualOctGenerated(),
      .oHEXA(HEX5),
      .oHEXB(HEX4)
  );  //: Part 2 Legend:
      //: SW[5]= (random number if 1) or (ID# if 0), SW[0]=enable,
KEY[3]=next digit pair, KEY[0]=reset,

endmodule

module NextIDdualOctal(iClk, iRst, iNext, oIDdualOctal);
    input iClk,
        iNext,
        iRst;
    output reg [5:0] oIDdualOctal;

    reg [7:0] state, next_state;
    always @(state) begin
        case(state) //: There are 8 states, one for each ID# digit
            1: begin //: 1st Digit of ID (2)
                next_state <= 2;
                oIDdualOctal <= 6'b010110; //: DISPLAY{2,6},  (Set output
octal value, based on the state. i.e 2->22->10110->010110)
                end
            2: begin //: 2nd Digit of ID (7)
                next_state <= 3;
                oIDdualOctal <= 6'b001101;  //: DISPLAY{1,5}
                end
            3: begin //: 3rd Digit of ID (7)
                next_state <= 4;
                oIDdualOctal <= 6'b001101;  //: DISPLAY{1,5}
                end
            4: begin //: 4th Digit of ID (7)
                next_state <= 5;
                oIDdualOctal <= 6'b001101; //: DISPLAY{1,5}
                end
            5: begin //: 5th Digit of ID (1)
                next_state <= 6;
                oIDdualOctal <= 6'b001011; //: DISPLAY{1,3}
                end
            6: begin //: 6th Digit of ID (4)
                next_state <= 7;
                oIDdualOctal = 6'b101100; //: DISPLAY{5,4}
                end
            7: begin //: 7th Digit of ID (5)
                next_state <= 8;
                oIDdualOctal <= 6'b110111; //: DISPLAY{6,7}
                end
            8: begin //: 8th Digit of ID (8)
                next_state <= 1;
                oIDdualOctal <= 6'b011000; //: DISPLAY{3,0}
                end
            default: begin
            next_state <= 3'bxxx;
            oIDdualOctal <= 6'bxxxxxx;
            end
        endcase
    end

    always @(posedge iClk) begin
        if (iRst == 1'b1) //: if reset hit, go back to last digit state
            state <= 8;
        else if (iNext == 1'b1)
            state <= next_state ; //: otherwise shifts to the next id
digit/next state
    end

endmodule
```

- Include a .sof file (the file that Quartus produces for download to the FPGA). Rename the file MyPart2.sof
    - **Done**

**Part 3:**

- 3.1: Include pretty printed Verilog for your FSM module. Ensure that you document the Verilog code. This must include a description of each state and the use of parameters with brief state names that describe each state. State S1, S2 etc are NOT acceptable. Clearly the states need different values and the number of states will determine how many bits are needed for the state. All your state bits must be contained in the Verilog signal state in your code.

```
module Assign2019(CLOCK_50, KEY, SW, LEDR, LEDG, HEX0, HEX1, HEX2, HEX3,
HEX4, HEX5, HEX6, HEX7);
    input CLOCK_50;
    input [3:0] KEY;
    input [17:0] SW;
    output [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7;
    output [17:0] LEDR;
    output [7:0] LEDG;

    assign LEDG[3:0] = ~KEY[3:0];
    assign LEDR[17:0] = SW[17:0];
    //: Note:
    //: I am using "//:" to easily differentiate my own comments

    CognitionTimer part3(
        .iClk(CLOCK_50),
        .iRst(~KEY[0]),
        .iChooseRandID(SW[5]),
        .iUser4bitSW(SW[3:0]),
        .iSubmitSW(SW[17]),
        .oHEX0(HEX0[6:0]),
        .oHEX1(HEX1[6:0]),
        .oHEX2(HEX2[6:0]),
        .oHEX3(HEX3[6:0]),
        .oHEX4(HEX4[6:0]),
        .oHEX5(HEX5[6:0]),
        .oHEX6(HEX6[6:0]),
        .oHEX7(HEX7[6:0])
    );  //: Part 3 Legend:
        //: SW[17]= submit answer, SW[5]= (random number if 1) or (ID# if
0), SW[3:0]= user inputs, KEY[0]=reset


endmodule

module FSM(iClk, iRst, iUser4bitSW, iSubmitSW, iTimer16, iDualOctGenerated,
oResetTimer, oUser4bitNumError, oFreezeDisplayTimer, oBlankOctNumsDisplay,
oShowTimerErrorDisplay, oNewOctNumsReq, oState);

    input iClk, iRst, iSubmitSW;
    input [3:0] iUser4bitSW;
    input [15:0] iTimer16;
    input [5:0] iDualOctGenerated;
    output reg  oResetTimer,
                oFreezeDisplayTimer,
                oBlankOctNumsDisplay,
                oNewOctNumsReq,
                oShowTimerErrorDisplay;
    output reg [1:0] oUser4bitNumError;
    output [2:0] oState;

    assign oState = state;

    parameter RESET_STATE = 0, INPUT_STATE = 1, SUBMIT_STATE = 2,
CHECK_STATE = 3, ERROR_1 = 4, ERROR_2 = 5; /* student to complete other
states here */

    reg [2:0] state, next_state; //: 2 bit regs as less than 8 states
needed

    reg correct_Answer;

    always@(state, iTimer16, iUser4bitSW, iDualOctGenerated, iSubmitSW)
begin
            oFreezeDisplayTimer=0;
            oResetTimer = 0; //: Resets the timer each instance
            oBlankOctNumsDisplay = 1; //: Makes the display blank before
the instance
            oUser4bitNumError = 0;
            oShowTimerErrorDisplay = 0;  //: As no error has occured before
the instance, do not display an error
            next_state = state;

        case(state)
            RESET_STATE: begin
                oShowTimerErrorDisplay = 0;
                correct_Answer = 0;

                if(iTimer16 == 20) begin //: Delays by 2 seconds before
changing to input state
                    oNewOctNumsReq = 1; //: Allows for a generated number
for the next state, input state

                    next_state = INPUT_STATE;
                end
                else if (iRst) begin //: Reset the timer if reset pressed
                    oResetTimer = 1;

                    next_state = RESET_STATE;
                end
            end

            INPUT_STATE: begin
                oNewOctNumsReq = 0; //: Stop constantly generating octal
values
                oBlankOctNumsDisplay = 0; //: Display the required octal
values

                if(iSubmitSW == 1) begin
                    oResetTimer = 1; //: Reset timer so that the next delay
8 sec starts from zero

                    next_state = SUBMIT_STATE;
                end
                else if (iTimer16 == 80) begin  //: ERROR 2: If the user
takes more than 8 seconds, display error 2/transition to error 2 state
                    next_state = ERROR_2;
                end
                else if (iRst) begin
```

```
                    oResetTimer = 1;

                    next_state = RESET_STATE;
                end
            end

            SUBMIT_STATE: begin
                oNewOctNumsReq = 0;
                oShowTimerErrorDisplay = 1;

                if (iRst) begin
                    oResetTimer = 1;

                    next_state = RESET_STATE;
                end
                else if (correct_Answer == 1) begin //: So that if switch
is flicked off, timer stays frozen
                    oFreezeDisplayTimer = 1;
                    next_state = SUBMIT_STATE;
                    if (0 == iUser4bitSW) //: Reset if user changes input
to zero
                        next_state = RESET_STATE;
                end
                else if (iSubmitSW == 1) begin //: CORRECT ANSWERS LIE
HERE!!!
                    if (((iDualOctGenerated[5:3]) +
(iDualOctGenerated[2:0])) == iUser4bitSW) begin  //: Conveting the DUAL
OCTAL value to a SINGLE 4 BIT SUM
                        oFreezeDisplayTimer = 1; //: Display the timer
value
                        oBlankOctNumsDisplay = 1;
                        correct_Answer = 1;

                        next_state = SUBMIT_STATE;
                    end
                end
            end

            CHECK_STATE: begin
                if (correct_Answer == 1) begin
                    oFreezeDisplayTimer = 1;
                    next_state = SUBMIT_STATE;
                end
                else if (iSubmitSW == 1) begin
                    if (((iDualOctGenerated[5:3]) +
(iDualOctGenerated[2:0])) != iUser4bitSW) begin  //: ERROR 1: If the user
inputs the wrong answer
                        oUser4bitNumError = 1;
                        oShowTimerErrorDisplay = 1;

                        next_state = ERROR_1;
                    end
                end
                else if (iRst) begin
                    oResetTimer = 1;

                    next_state = RESET_STATE;
                end
                else //: If incorrect go back to submit state
                    next_state = SUBMIT_STATE;
            end
            ERROR_1: begin
                if(0 != iUser4bitSW) begin
                    oUser4bitNumError = 1; //: Let this CogntionTimer know
that this error code 2
                    oShowTimerErrorDisplay = 1; //: and display the error
                    next_state <= ERROR_1; //: Stay in this state until
user resets
                end
                else if (0 == iUser4bitSW)
                    next_state = RESET_STATE;
                else if (iRst) begin
                    oResetTimer = 1;

                    next_state = RESET_STATE;
                end
            end

            ERROR_2: begin
                if(0 != iUser4bitSW) begin
                    oUser4bitNumError = 2;
                    oShowTimerErrorDisplay = 1;
                    next_state = ERROR_2;
                end
                else if (0 == iUser4bitSW)
                    next_state = RESET_STATE;
                else if (iRst) begin
                    oResetTimer = 1;

                    next_state = RESET_STATE;
                end
            end

            default: next_state = RESET_STATE;
        endcase
    end

    always @(posedge iClk) begin
        if (iRst) state <= RESET_STATE;
        else state <= next_state;
    end

endmodule
```

- Include MyPart3.sof file (rename the file that Quartus produces for download to the FPGA).
    - **Done**