**ECE211-L LAB 5**

**Lab 5 Report: Binary Adders/Subtractors**

*Efe CIVISOKEN*

*Fahendrena RAMIANDRASOA*

*03/12/2025*

*Statement of Collaboration:* Each team member contributed to the design, implementation, and testing of the code conversion module. Efe and Fahendrena both worked on implementing half and full adder designs. Efe wrote the half adder and debugged the full adder. Fahendrena wrote the full adder and double-checked for bugs in the half adder design. They also worked together in implementing the 6-bit adder module and incorporating the “two’s complement” to account for subtraction. In the implementation part of the negative value display, even though both worked together, it was mostly needed to be done by Efe. This lab report is also written entirely by Efe.

**Time Spent:** 4 hours.

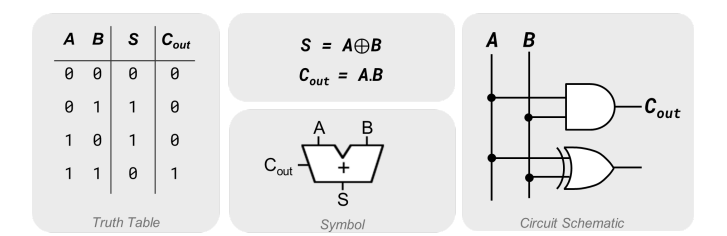
**INTRODUCTION**

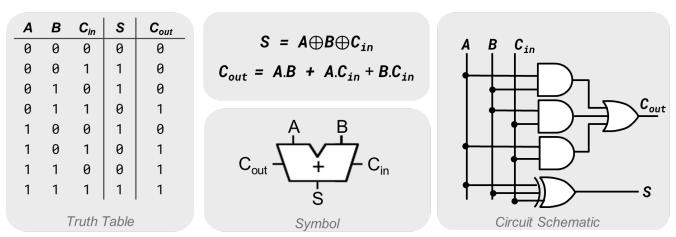
This lab focused on implementing binary adders and subtractors using SystemVerilog at different abstraction levels. We first built gate-level models for half adder and full adder modules and then used them to construct a 6-bit ripple-carry adder. By leveraging the properties of 2’s complement arithmetic, we extended our design to perform both addition and subtraction, displaying results on the Nexys A7 FPGA’s seven-segment display, for which the modules were already provided by the lab instructor. The lab emphasized modular design, hierarchical structuring, and hardware implementation using SystemVerilog.

**DESIGN**

1. **Design of Half Adder and Full Adder Modules**

The logic behind half adders and full adders were already provided by the lab instructor as can be seen in Figures 1 & 2.



**Figure 1:** **Half Adder Truth Tables, Equations, and Schematic**

V: 1

V: 0

**Figure 2: Full Adder Truth Tables, Equations, and Schematic**

Leveraging these provided equations, we designed our half adder and full adder modules –to be further utilized in ripple 6-bit adder module– in the following way:

module full\_adder(input logic a, b, cin, output logic s, cout

);

xor(s, a, b, cin);

or(cout, a&b, a&cin, b&cin);

endmodule

//////////////////////////////////////////////////////////////////////////////////

module half\_adder( input logic a, b, output logic s, cout

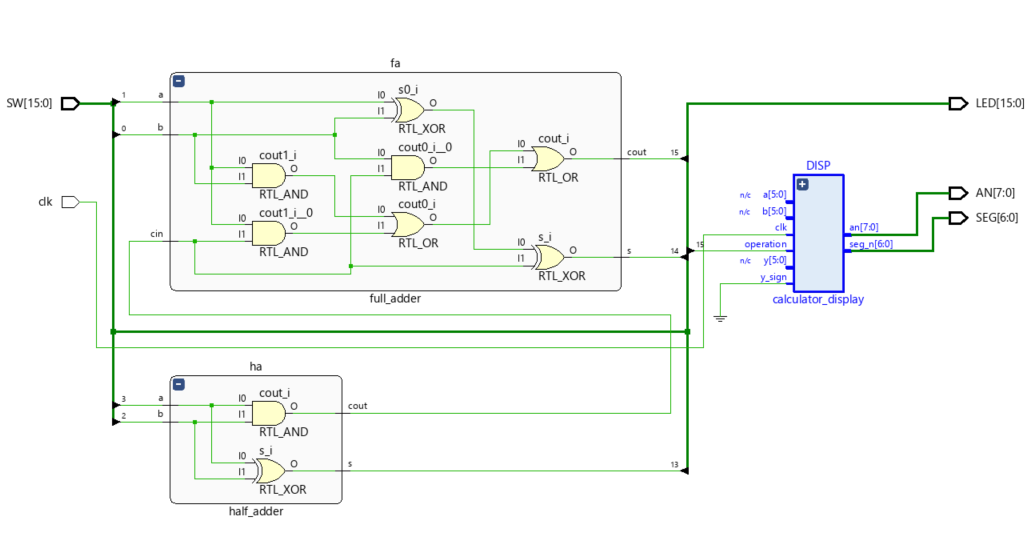
);

xor(s, a, b);

and(cout,a,b);

endmodule

**Code 1: SystemVerilog Codes for the Full Adder Module (full\_adder.sv) and Half Adder (half\_adder.sv) Module, respectively.**

****

**Schema 1. Final Vivado Circuit Schematic View of the Modules full\_adder and half\_adder.**

1. **Design of the 6-bit Adder Module**

A 6-bit ripple-carry adder was implemented by connecting multiple –five to be precise– full adders in series, with a single half adder used for the least significant bit (LSB) operation. The design takes two 6-bit binary inputs/vectors, denoted as A and B, and produces a 7-bit sum output, Y, whose most significant bit (MSB) from the last full-adder’s carry-out is going to be ignored in the next sections for simplicity purposes. The carry-out from each full adder serves as the carry-in for the subsequent full adder, effectively propagating the carry through all six stages.

The ripple-carry adder operates by performing bitwise addition from the LSB to MSB. At each stage, the sum of two corresponding bits from the inputs, along with any carry-in from the previous stage, is computed. The propagation of carry signals through the adder creates a delay, which can affect performance in larger designs but is sufficient for this 6-bit implementation.

To construct the ripple-carry adder, we instantiated five full adder modules along with a half adder module within a new SystemVerilog module called “adder\_6b”. This approach leveraged the modularity of the adders designed in the previous section, simplifying the implementation while maintaining reusability. The interconnection between adders ensured that the circuit could correctly perform binary addition without requiring direct handling of individual logic gates.

A schematic representation of the 6-bit ripple-carry adder is also provided illustrating how the full and half adders were connected to form the complete circuit.

module adder\_6b( input logic [5:0] A, input logic [5:0] B, output logic [6:0] Y

);

logic cout0,cout1,cout2,cout3,cout4,cout5,cout6;

half\_adder ha1(.a(A[0]), .b(B[0]), .s(Y[0]), .cout(cout0));

full\_adder fa1(.a(A[1]), .b(B[1]), .cin(cout0), .s(Y[1]), .cout(cout1));

full\_adder fa2(.a(A[2]), .b(B[2]), .cin(cout1), .s(Y[2]), .cout(cout2));

full\_adder fa3(.a(A[3]), .b(B[3]), .cin(cout2), .s(Y[3]), .cout(cout3));

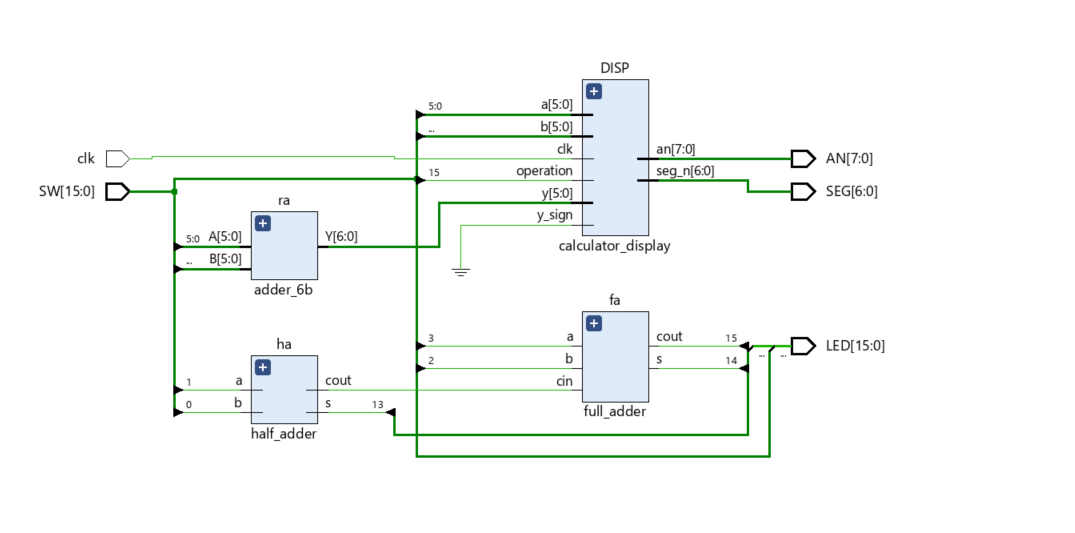
full\_adder fa4(.a(A[4]), .b(B[4]), .cin(cout3), .s(Y[4]), .cout(cout4));

full\_adder fa5(.a(A[5]), .b(B[5]), .cin(cout4), .s(Y[5]), .cout(cout5));

assign Y[6] = cout5;

endmodule

**Code 2: The SystemVerilog Code of the “adder\_6b” Module Using half\_adder and full\_adder Modules.**

****

**Schema 2. Final Vivado Circuit Schematic View of the Module adder\_6b.**

**FINAL IMPLEMENTATION OF THE BINARY ADDER/SUBSTRACTOR:**

To extend the functionality of our design to perform both addition and subtraction, we incorporated 2’s complement arithmetic. When performing subtraction, the circuit negates the second operand B by computing its 2’s complement representation. This transformation is achieved by inverting all bits of B and adding one to the least significant bit.

To facilitate this dual operation, we introduced a control signal, SW[15], which determines whether the circuit operates in addition or subtraction mode. When SW[15] is low, the original values of A and B are passed into the adder, resulting in straightforward binary addition. However, when SW[15] is high, B is transformed into its two’s complement form before being added, effectively performing subtraction.

Also, initially, our design did not account for overflow scenarios, which led to incorrect outputs when subtraction resulted in negative values. In our initial implementation, the result of a subtraction operation would appear as a large unsigned binary value, as the circuit displayed the direct 2’s complement representation. To resolve this issue, we implemented logic to detect when the result was negative and appropriately modify the displayed output.

To improve the readability of results, particularly when the subtraction yielded a negative number, we modified the design to display the correct sign and magnitude. The most significant bit (MSB) of the result was used to determine whether the output should be negative, and when necessary, the circuit converted the output into its absolute value before displaying it by reversing the two’s complement form. This approach ensured that the result appeared in a human-readable format rather than its raw binary form.

Finally, the implementation of this dual-operation logic can be achieved using either a multiplexer (MUX) or XOR gates as specified by the lab instructor. A MUX-based approach involves selecting between B and its negated version based on the control signal, while the XOR approach negates each bit of B when subtraction is selected. Both methods yield functionally correct results, but the XOR-based design can be more efficient in terms of hardware utilization. As efficiency was not a concern in 6-bit addition/subtraction implementation, we chose MUX-based approach leveraging ternary operators for convenience in the coding process. The final code and schematic, incorporating the explained method of implementing the binary adder/subtractor, are given in Code 3 & Schema 3.

module lab05\_top(

input logic clk,

input logic [15:0] SW,

output logic [15:0] LED,

output logic [6:0] SEG,

output logic [7:0] AN

);

// logic extra;

// Implement your circuit here!

// full\_adder fa(.a(SW[3]), .b(SW[2]), .cin(extra), .s(LED[14]), .cout(LED[15]));

// half\_adder ha(.a(SW[1]), .b(SW[0]), .s(LED[13]), .cout(extra));

logic [6:0] youtputp;

adder\_6b rap(.A({2'b00,SW[7:4]}), .B({2'b00,SW[3:0]}), .Y(youtputp));

logic [6:0] youtputn;

logic [6:0] complementb;

assign complementb = {1'd0, ~{1'd0,SW[3:0]}} + 000001; // debug d0 to d1

adder\_6b ran(.A({2'd0,SW[7:4]}), .B(complementb), .Y(youtputn));

logic [6:0] youtputneg;

assign youtputneg = ~youtputn[5] ? (~youtputn + 000001) : youtputn; // debug 6 to 5

// Connect signals to 7-segmenet display (with additional formatting)

calculator\_display DISP(

.a(SW[7:4]), /\* Connect input A here \*/

.b(SW[3:0]), /\* Connect input B here \*/

.operation(SW[15]), /\* Indicates add or subtract \*/

.y( SW[15] ? youtputneg[4:0] : youtputp[5:0]), /\* Connect result Y here \*/

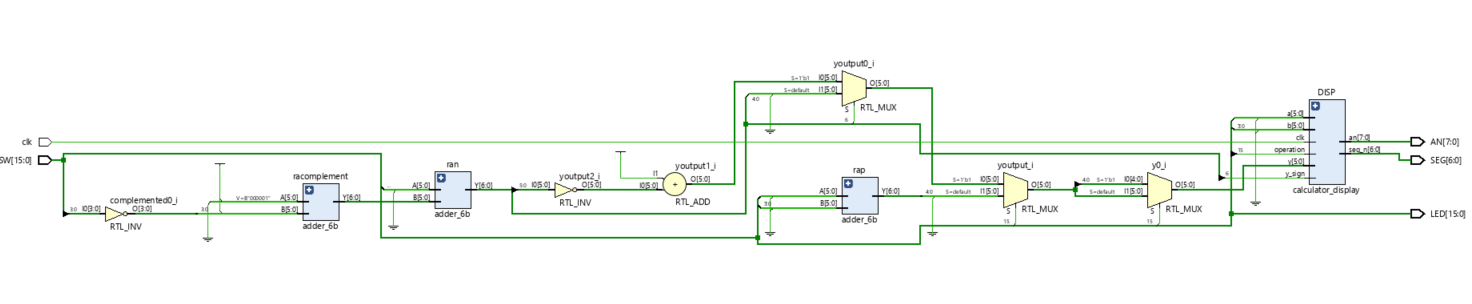
.y\_sign(SW[15] ? ~youtputn[5] : 0), /\* Replace connection for Part 4\*/ // debug 6 to 5

.clk(clk), .seg\_n(SEG), .an(AN));

assign LED[15:0] = SW[15:0];

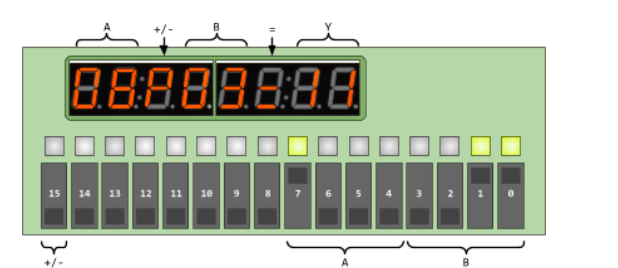
endmodule

**Code 3: Final SystemVerilog Code of “lab5\_top” Module.**



**Schema 3. Final Vivado Circuit Schematic View of the Complete Implementation of the Binary Adder/Substractor.**

**TESTING & RESULTS:**



**Figure 3: Input/Output Configuration for Lab 5.**

Our implementation was tested in each stage of the design process for ensuring bug-free code according to the specifications given in Figure 3. We first tested the addition functionality using 4-bit A and 4-bit B vectors and then incorporated for possible negative values of B depending on the signal coming from the SW[15]. After Vivado synthesis, implementation, and hardware programming of our top module, we were able to successfully confirm the error-free operation of our implementation. After that, we implemented the negative-value displaying part instead of overflow values in two’s complement form. It initially was still erroneous and after a through debugging process, we successfully incorporated for the negative values in the output part of the 7-segment displayer. Furthermore, the correctness of our implementation was confirmed by the lab instructor, verifying that all required functionality was present and working as intended. Some finalized test cases, including edge cases, are provided for enhancing the further understandability of our testing process in Table 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | A (6-bit Form) SW[7:4] | B (6-bit Form)  SW[3:0] | Mode (Add/Sub)  SW[15] | Expected Output (6-bit Form) | Observed Output (6-bit Form) |
| 1 | 000110 | 000001 | Add | 000111 | 000111 |
| 2 | 000101 | 000011 | Add | 000110 | 000110 |
| 3 | 000011 | 000100 | Add | 000111 | 000111 |
| 4 | 001111 | 000001 | Add | 010000 | 010000 |
| 5 | 000001 | 000010 | Sub | 111111 (-1) | 111111 (-1) |
| 6 | 000011 | 000101 | Sub | 111110 (-2) | 111110 (-2) |
| 7 | 000100 | 000011 | Sub | 000001 | 000001 |
| 8 | 001111 | 001111 | Add | 011110 | 011110 |
| 9 | 001000 | 000001 | Sub | 000111 | 000111 |
| 10 | 000000 | 000000 | Add | 000000 | 000000 |

**Table 1. Test Cases for Lab 5’s Final Implementation, Blue for Addition, Green for Subtraction.**

**DISCUSSION & CONCLUSION:**

Throughout this lab, we successfully implemented a 6-bit binary adder/subtractor using modular SystemVerilog design principles. The project allowed us to apply theoretical concepts from digital logic design and explore the implementation of arithmetic circuits using hierarchical structures. By utilizing half adders and full adders, we constructed a functional ripple-carry adder capable of handling both addition and subtraction operations through two’s complement representation.

During the implementation, we encountered several challenges, with the most time-consuming issue being the incorporation of the negative-valued result displaying process on the seven-segment display. Initially, we followed our own approach, ensuring that the positive and negative results were handled separately. However, after incorporating advice that suggested a different approach—one that was ultimately incompatible with our design—we spent a very considerable amount of time trying to make it work. This advice recommended combining the outputs into a single variable, which led to unforeseen conflicts within our implementation structure and disrupted the transition between addition and subtraction. the outputs into a single variable, which ultimately conflicted with our implementation structure. After recognizing this incompatibility, we reverted to our original design, in which the positive and negative results were stored separately in “youtputp” and “youtputneg”, respectively. This correction, as seen in Code 3, resolved the issue and ensured proper handling of both positive and negative results on the display. Once this major obstacle was overcome, we encountered a few minor errors that needed debugging. However, after systematically addressing each issue, we finally achieved a fully functional implementation that met all the lab’s requirements.

Upon final testing, our circuit successfully performed binary addition and subtraction, correctly displaying both positive and negative results while maintaining an accurate representation of the numerical values, which was also confirmed by the lab instructor Prof. Biernacki during the actual lab time, as mentioned in TESTING & RESULTS part of this document. To summarize, the final design demonstrated a smooth and reliable execution of arithmetic operations, with the seven-segment display correctly reflecting signed values.

END.

**APPENDIX**

* **Lab05 Demo File on Moodle Page.**
* **OTHER SYSTEMVERILOG MODULES:**

//////////////////////////////////////////////////////////////////////////////////

// Company: Lafayette College

// Create Date: 01/09/2024 10:11:26 AM

// Design Name:

// Module Name: adv\_seven\_seg\_n

// Project Name: ECE 211 Digital Circuits 1

//-----------------------------------------------------------------------------

// Author : John Nestor <nestorj@lafayette.edu>

// Created : Feb 2020

// Revised : Lauren Biernacki <biernacl@lafayette.edu> Jan 2024

//-----------------------------------------------------------------------------

// Description : Advanced seven-segment decoder displaying

// hexidecimal values 0-F with active low outputs

// Segments are ordered as follows: segs\_n[6]=g, segs\_n[0]=a

//-----------------------------------------------------------------------------

//////////////////////////////////////////////////////////////////////////////////

module adv\_seven\_seg\_n(

input logic [3:0] data,

output logic [6:0] segs\_n // ordered g(6) - a(0)

);

always\_comb

case (data) // Seg: gfedcba

4'd0: segs\_n = 7'b1000000;

4'd1: segs\_n = 7'b1111001;

4'd2: segs\_n = 7'b0100100;

4'd3: segs\_n = 7'b0110000;

4'd4: segs\_n = 7'b0011001;

4'd5: segs\_n = 7'b0010010;

4'd6: segs\_n = 7'b0000010;

4'd7: segs\_n = 7'b1111000;

4'd8: segs\_n = 7'b0000000;

4'd9: segs\_n = 7'b0010000;

4'd10: segs\_n = 7'b0001000;

4'd11: segs\_n = 7'b0000011;

4'd12: segs\_n = 7'b1000110;

4'd13: segs\_n = 7'b0100001;

4'd14: segs\_n = 7'b0000110;

4'd15: segs\_n = 7'b0001110;

default: segs\_n = 7'b1111111;

endcase

endmodule: adv\_seven\_seg\_n

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company: Lafayette College

// Create Date: 02/26/2024

// Design Name:

// Module Name: calculator\_display

// Project Name: ECE 211 Digital Circuits 1

//-----------------------------------------------------------------------------

// Author : Lauren Biernacki <biernacl@lafayette.edu>

// Created : Feb 2024

//-----------------------------------------------------------------------------

// Description : Takes inputs a, b, and y to display on the seven-segement

// display in the format ``a - b = y''.

// The input operation determines wether a minus sign or the

// letter 'p' is displayed. Additional inputs can be used to

// display the sign of the result (i.e., ``- y'') or display

// ``Err'' in the case of an overflow.

//-----------------------------------------------------------------------------

//////////////////////////////////////////////////////////////////////////////////

module calculator\_display(

input logic [5:0] a, b,

input logic [5:0] y,

input logic y\_sign,

input logic operation,

input logic clk,

output logic [6:0] seg\_n, //connect to board seg pin

output logic [7:0] an //connect to board en pin

);

// Make A, B, and Y Binary-Coded Decimal

// "tens" are really ones, "ones" are really tenths

logic [3:0] a\_tens, a\_ones, b\_tens, b\_ones, y\_tens, y\_ones;

dbl\_dabble\_ext U\_DBLX\_A (.b(a), .tens(a\_tens), .ones(a\_ones));

dbl\_dabble\_ext U\_DBLX\_B (.b(b), .tens(b\_tens), .ones(b\_ones));

dbl\_dabble\_ext U\_DBLX\_Y (.b(y), .tens(y\_tens), .ones(y\_ones));

// Decode the binary values a, b, and y to

// get the representation on the seven-segement display

logic [6:0] seg\_a\_tens, seg\_a\_ones, seg\_b\_tens, seg\_b\_ones, seg\_y\_tens, seg\_y\_ones;

adv\_seven\_seg\_n D0(.data(a\_tens), .segs\_n(seg\_a\_tens));

adv\_seven\_seg\_n D1(.data(b\_tens), .segs\_n(seg\_b\_tens));

adv\_seven\_seg\_n D2(.data(y\_tens), .segs\_n(seg\_y\_tens));

adv\_seven\_seg\_n D3(.data(a\_ones), .segs\_n(seg\_a\_ones));

adv\_seven\_seg\_n D4(.data(b\_ones), .segs\_n(seg\_b\_ones));

adv\_seven\_seg\_n D5(.data(y\_ones), .segs\_n(seg\_y\_ones));

// Decode the + or - sign operator

logic [6:0] seg\_operator = operation ? 7'b0111111 : 7'b0001100;

// Decode the sign bit for y

logic [6:0] seg\_sign = y\_sign ? 7'b0111111 : 7'b1111111;

logic [6:0] seg\_equals = 7'b0110111;

logic [6:0] zero = 7'b1000000;

logic [7:0] seg7, seg6, seg5, seg4, seg3, seg2, seg1, seg0;

always\_comb begin

// Select output configuration based on inputs for Part 4

// (overflow detection or sign/magnitude of output)

// ugh, its complicated to fit everything!

if(y\_sign == 0) begin

//Result is positive, display AA+BB=YY

seg7 = seg\_a\_tens;

seg6 = seg\_a\_ones;

seg5 = seg\_operator;

seg4 = seg\_b\_tens;

seg3 = seg\_b\_ones;

seg2 = seg\_equals;

seg1 = seg\_y\_tens;

seg0 = seg\_y\_ones;

end

else if (y\_sign == 1 & a\_tens == 0) begin

//Result is negative with 2-sig-digits, display A+BB=-YY

seg7 = seg\_a\_ones;

seg6 = seg\_operator;

seg5 = seg\_b\_tens;

seg4 = seg\_b\_ones;

seg3 = seg\_equals;

seg2 = seg\_sign;

seg1 = seg\_y\_tens;

seg0 = seg\_y\_ones;

end

else begin

//Result is negative with 1-sig-digits, display AA+BB=-Y

seg7 = seg\_a\_tens;

seg6 = seg\_a\_ones;

seg5 = seg\_operator;

seg4 = seg\_b\_tens;

seg3 = seg\_b\_ones;

seg2 = seg\_equals;

seg1 = seg\_sign;

seg0 = seg\_y\_ones;

end

end

disp\_mux\_seven\_seg D6(.clk(clk), .reset(1'b0),

.seg7(seg7), /\* a \*/

.seg6(seg6), /\* a \*/

.seg5(seg5), /\* add/subtract \*/

.seg4(seg4), /\* b \*/

.seg3(seg3), /\* b \*/

.seg2(seg2), /\* equals \*/

.seg1(seg1), /\* y\_tens or - \*/

.seg0(seg0), /\* y\_ones \*/

.seg\_dis(seg\_disable), .seg\_n(seg\_n), .an(an)

);

Endmodule

//////////////////////////////////////////////////////////////////////////////////

// Company: Lafayette College

// Create Date: 01/09/2024 10:11:26 AM

// Design Name:

// Module Name: disp\_mux\_seven\_seg

// Project Name: ECE 211 Digital Circuits 1

//-----------------------------------------------------------------------------

// Author : Lauren Biernacki <biernacl@lafayette.edu>

// Created : Jan 2024

// Acknowledgment: Faraz Khan <https://simplefpga.blogspot.com/2012/07/seven-segment-led-multiplexing-circuit.html>, July 2012

//-----------------------------------------------------------------------------

// Description : Time-multiplexing circuit to display multi-digit seven-segment

// display with active-low outputs

// Takes four decoder values (seg0-seg3) and to activate each corresponding

// digit on the seven-segment display

// The refresh rate should be near 1000 Hz to achieve the desired

// visual effect. The refresh rate is determined by the clock period

// and the parameter N (i.e., clk\_period/(2^N-2))

//-----------------------------------------------------------------------------

//////////////////////////////////////////////////////////////////////////////////

module disp\_mux\_seven\_seg(

input logic clk,

input logic reset,

input logic [6:0] seg0, //data for seg LED 0

input logic [6:0] seg1, // data for seg LED 1

input logic [6:0] seg2, // data for seg LED 2

input logic [6:0] seg3, // data for seg LED 3

input logic [6:0] seg4, //data for seg LED 4

input logic [6:0] seg5, // data for seg LED 5

input logic [6:0] seg6, // data for seg LED 6

input logic [6:0] seg7, // data for seg LED 7

input logic [8:0] seg\_dis, //8-bit value to disable each seg LED(active-low)

output logic [6:0] seg\_n, //connect to board seg pin

output logic [7:0] an //connect to board en pin

);

localparam N = 19; // Refresh rate of ~800Hz for a 100MHz clock

// Update counter on clock tick (for time multiplexing)

logic [N-1:0] q\_reg;

always\_ff @(posedge clk)

begin

if (reset) q\_reg <=0;

else q\_reg <= q\_reg + 1;

end

// Multiplex which seg LED is displayed

always\_comb

begin

case(q\_reg[N-1:N-3])

3'b000: begin

seg\_n = seg0;

an = 8'b11111110 | seg\_dis;

end

3'b001: begin

seg\_n = seg1;

an = 8'b11111101 | seg\_dis;

end

3'b010: begin

seg\_n = seg2;

an = 8'b11111011 | seg\_dis;

end

3'b011: begin

seg\_n = seg3;

an = 8'b11110111 | seg\_dis;

end

3'b100: begin

seg\_n = seg4;

an = 8'b11101111 | seg\_dis;

end

3'b101: begin

seg\_n = seg5;

an = 8'b11011111 | seg\_dis;

end

3'b110: begin

seg\_n = seg6;

an = 8'b10111111 | seg\_dis;

end

3'b111: begin

seg\_n = seg7;

an = 8'b01111111 | seg\_dis;

end

endcase

end

endmodule: disp\_mux\_seven\_seg

## This file is a general .xdc for the Nexys A7-100T

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -period 10.00 -waveform {0 5} [get\_ports {clk}];

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { SW[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { SW[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { SW[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { SW[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { SW[4] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { SW[5] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { SW[6] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { SW[7] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { SW[8] }]; #IO\_L24N\_T3\_34 Sch=sw[8]

set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { SW[9] }]; #IO\_25\_34 Sch=sw[9]

set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { SW[10] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { SW[11] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { SW[12] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { SW[13] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { SW[14] }]; #IO\_L19N\_T3\_A09\_D25\_VREF\_14 Sch=sw[14]

set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { SW[15] }]; #IO\_L21P\_T3\_DQS\_14 Sch=sw[15]

## LEDs

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { LED[0] }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { LED[1] }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { LED[2] }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { LED[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { LED[4] }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { LED[5] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]

set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { LED[6] }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]

set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { LED[7] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { LED[8] }]; #IO\_L16N\_T2\_A15\_D31\_14 Sch=led[8]

set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { LED[9] }]; #IO\_L14N\_T2\_SRCC\_14 Sch=led[9]

set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { LED[10] }]; #IO\_L22P\_T3\_A05\_D21\_14 Sch=led[10]

set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { LED[11] }]; #IO\_L15N\_T2\_DQS\_DOUT\_CSO\_B\_14 Sch=led[11]

set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { LED[12] }]; #IO\_L16P\_T2\_CSI\_B\_14 Sch=led[12]

set\_property -dict { PACKAGE\_PIN V14 IOSTANDARD LVCMOS33 } [get\_ports { LED[13] }]; #IO\_L22N\_T3\_A04\_D20\_14 Sch=led[13]

set\_property -dict { PACKAGE\_PIN V12 IOSTANDARD LVCMOS33 } [get\_ports { LED[14] }]; #IO\_L20N\_T3\_A07\_D23\_14 Sch=led[14]

set\_property -dict { PACKAGE\_PIN V11 IOSTANDARD LVCMOS33 } [get\_ports { LED[15] }]; #IO\_L21N\_T3\_DQS\_A06\_D22\_14 Sch=led[15]

## RGB LEDs

#set\_property -dict { PACKAGE\_PIN R12 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_B }]; #IO\_L5P\_T0\_D06\_14 Sch=led16\_b

#set\_property -dict { PACKAGE\_PIN M16 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_G }]; #IO\_L10P\_T1\_D14\_14 Sch=led16\_g

#set\_property -dict { PACKAGE\_PIN N15 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_R }]; #IO\_L11P\_T1\_SRCC\_14 Sch=led16\_r

#set\_property -dict { PACKAGE\_PIN G14 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_B }]; #IO\_L15N\_T2\_DQS\_ADV\_B\_15 Sch=led17\_b

#set\_property -dict { PACKAGE\_PIN R11 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_G }]; #IO\_0\_14 Sch=led17\_g

#set\_property -dict { PACKAGE\_PIN N16 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_R }]; #IO\_L11N\_T1\_SRCC\_14 Sch=led17\_r

##7 segment display (SPECIFIC TO THE Nexys-A7-100T)

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { SEG[0] }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { SEG[1] }]; #IO\_25\_14 Sch=cb

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { SEG[2] }]; #IO\_25\_15 Sch=cc

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { SEG[3] }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { SEG[4] }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { SEG[5] }]; #IO\_L19P\_T3\_A10\_D26\_14 Sch=cf

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { SEG[6] }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { DP }]; #IO\_L19N\_T3\_A21\_VREF\_15 Sch=dp

set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { AN[0] }]; #IO\_L23P\_T3\_FOE\_B\_15 Sch=an[0]

set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { AN[1] }]; #IO\_L23N\_T3\_FWE\_B\_15 Sch=an[1]

set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { AN[2] }]; #IO\_L24P\_T3\_A01\_D17\_14 Sch=an[2]

set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { AN[3] }]; #IO\_L19P\_T3\_A22\_15 Sch=an[3]

set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { AN[4] }]; #IO\_L8N\_T1\_D12\_14 Sch=an[4]

set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { AN[5] }]; #IO\_L14P\_T2\_SRCC\_14 Sch=an[5]

set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { AN[6] }]; #IO\_L23P\_T3\_35 Sch=an[6]

set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { AN[7] }]; #IO\_L23N\_T3\_A02\_D18\_14 Sch=an[7]

##Buttons

#set\_property -dict { PACKAGE\_PIN C12 IOSTANDARD LVCMOS33 } [get\_ports { CPU\_RESETN }]; #IO\_L3P\_T0\_DQS\_AD1P\_15 Sch=cpu\_resetn

#set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { BTNC }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

#set\_property -dict { PACKAGE\_PIN M18 IOSTANDARD LVCMOS33 } [get\_ports { BTNU }]; #IO\_L4N\_T0\_D05\_14 Sch=btnu

#set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { BTNL }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

#set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { BTNR }]; #IO\_L10N\_T1\_D15\_14 Sch=btnr

#set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { BTND }]; #IO\_L9N\_T1\_DQS\_D13\_14 Sch=btnd

##Pmod Headers

##Pmod Header JA

#set\_property -dict { PACKAGE\_PIN C17 IOSTANDARD LVCMOS33 } [get\_ports { JA[0] }]; #IO\_L20N\_T3\_A19\_15 Sch=ja[0]

#set\_property -dict { PACKAGE\_PIN D18 IOSTANDARD LVCMOS33 } [get\_ports { JA[1] }]; #IO\_L21N\_T3\_DQS\_A18\_15 Sch=ja[1]

#set\_property -dict { PACKAGE\_PIN E18 IOSTANDARD LVCMOS33 } [get\_ports { JA[2] }]; #IO\_L21P\_T3\_DQS\_15 Sch=ja[2]

#set\_property -dict { PACKAGE\_PIN G17 IOSTANDARD LVCMOS33 } [get\_ports { JA[3] }]; #IO\_L18N\_T2\_A23\_15 Sch=ja[3]

#set\_property -dict { PACKAGE\_PIN D17 IOSTANDARD LVCMOS33 } [get\_ports { JA[4] }]; #IO\_L16N\_T2\_A27\_15 Sch=ja[4]

#set\_property -dict { PACKAGE\_PIN E17 IOSTANDARD LVCMOS33 } [get\_ports { JA[5] }]; #IO\_L16P\_T2\_A28\_15 Sch=ja[5]

#set\_property -dict { PACKAGE\_PIN F18 IOSTANDARD LVCMOS33 } [get\_ports { JA[6] }]; #IO\_L22N\_T3\_A16\_15 Sch=ja[6]

#set\_property -dict { PACKAGE\_PIN G18 IOSTANDARD LVCMOS33 } [get\_ports { JA[7] }]; #IO\_L22P\_T3\_A17\_15 Sch=ja[7]

##Pmod Header JB

#set\_property -dict { PACKAGE\_PIN D14 IOSTANDARD LVCMOS33 } [get\_ports { JB[0] }]; #IO\_L1P\_T0\_AD0P\_15 Sch=jb[0]

#set\_property -dict { PACKAGE\_PIN F16 IOSTANDARD LVCMOS33 } [get\_ports { JB[1] }]; #IO\_L14N\_T2\_SRCC\_15 Sch=jb[1]

#set\_property -dict { PACKAGE\_PIN G16 IOSTANDARD LVCMOS33 } [get\_ports { JB[2] }]; #IO\_L13N\_T2\_MRCC\_15 Sch=jb[2]

#set\_property -dict { PACKAGE\_PIN H14 IOSTANDARD LVCMOS33 } [get\_ports { JB[3] }]; #IO\_L15P\_T2\_DQS\_15 Sch=jb[3]

#set\_property -dict { PACKAGE\_PIN E16 IOSTANDARD LVCMOS33 } [get\_ports { JB[4] }]; #IO\_L11N\_T1\_SRCC\_15 Sch=jb[4]

#set\_property -dict { PACKAGE\_PIN F13 IOSTANDARD LVCMOS33 } [get\_ports { JB[5] }]; #IO\_L5P\_T0\_AD9P\_15 Sch=jb[5]

#set\_property -dict { PACKAGE\_PIN G13 IOSTANDARD LVCMOS33 } [get\_ports { JB[6] }]; #IO\_0\_15 Sch=jb[6]

#set\_property -dict { PACKAGE\_PIN H16 IOSTANDARD LVCMOS33 } [get\_ports { JB[7] }]; #IO\_L13P\_T2\_MRCC\_15 Sch=jb[7]

##Pmod Header JC

#set\_property -dict { PACKAGE\_PIN K1 IOSTANDARD LVCMOS33 } [get\_ports { JC[1] }]; #IO\_L23N\_T3\_35 Sch=jc[1]

#set\_property -dict { PACKAGE\_PIN F6 IOSTANDARD LVCMOS33 } [get\_ports { JC[2] }]; #IO\_L19N\_T3\_VREF\_35 Sch=jc[2]

#set\_property -dict { PACKAGE\_PIN J2 IOSTANDARD LVCMOS33 } [get\_ports { JC[3] }]; #IO\_L22N\_T3\_35 Sch=jc[3]

#set\_property -dict { PACKAGE\_PIN G6 IOSTANDARD LVCMOS33 } [get\_ports { JC[4] }]; #IO\_L19P\_T3\_35 Sch=jc[4]

#set\_property -dict { PACKAGE\_PIN E7 IOSTANDARD LVCMOS33 } [get\_ports { JC[7] }]; #IO\_L6P\_T0\_35 Sch=jc[7]

#set\_property -dict { PACKAGE\_PIN J3 IOSTANDARD LVCMOS33 } [get\_ports { JC[8] }]; #IO\_L22P\_T3\_35 Sch=jc[8]

#set\_property -dict { PACKAGE\_PIN J4 IOSTANDARD LVCMOS33 } [get\_ports { JC[9] }]; #IO\_L21P\_T3\_DQS\_35 Sch=jc[9]

#set\_property -dict { PACKAGE\_PIN E6 IOSTANDARD LVCMOS33 } [get\_ports { JC[10] }]; #IO\_L5P\_T0\_AD13P\_35 Sch=jc[10]

##Pmod Header JD

#set\_property -dict { PACKAGE\_PIN H4 IOSTANDARD LVCMOS33 } [get\_ports { JD[1] }]; #IO\_L21N\_T3\_DQS\_35 Sch=jd[1]

#set\_property -dict { PACKAGE\_PIN H1 IOSTANDARD LVCMOS33 } [get\_ports { JD[2] }]; #IO\_L17P\_T2\_35 Sch=jd[2]

#set\_property -dict { PACKAGE\_PIN G1 IOSTANDARD LVCMOS33 } [get\_ports { JD[3] }]; #IO\_L17N\_T2\_35 Sch=jd[3]

#set\_property -dict { PACKAGE\_PIN G3 IOSTANDARD LVCMOS33 } [get\_ports { JD[4] }]; #IO\_L20N\_T3\_35 Sch=jd[4]

#set\_property -dict { PACKAGE\_PIN H2 IOSTANDARD LVCMOS33 } [get\_ports { JD[7] }]; #IO\_L15P\_T2\_DQS\_35 Sch=jd[7]

#set\_property -dict { PACKAGE\_PIN G4 IOSTANDARD LVCMOS33 } [get\_ports { JD[8] }]; #IO\_L20P\_T3\_35 Sch=jd[8]

#set\_property -dict { PACKAGE\_PIN G2 IOSTANDARD LVCMOS33 } [get\_ports { JD[9] }]; #IO\_L15N\_T2\_DQS\_35 Sch=jd[9]

#set\_property -dict { PACKAGE\_PIN F3 IOSTANDARD LVCMOS33 } [get\_ports { JD[10] }]; #IO\_L13N\_T2\_MRCC\_35 Sch=jd[10]

##Pmod Header JXADC

#set\_property -dict { PACKAGE\_PIN A14 IOSTANDARD LVCMOS33 } [get\_ports { XA\_N[1] }]; #IO\_L9N\_T1\_DQS\_AD3N\_15 Sch=xa\_n[1]

#set\_property -dict { PACKAGE\_PIN A13 IOSTANDARD LVCMOS33 } [get\_ports { XA\_P[1] }]; #IO\_L9P\_T1\_DQS\_AD3P\_15 Sch=xa\_p[1]

#set\_property -dict { PACKAGE\_PIN A16 IOSTANDARD LVCMOS33 } [get\_ports { XA\_N[2] }]; #IO\_L8N\_T1\_AD10N\_15 Sch=xa\_n[2]

#set\_property -dict { PACKAGE\_PIN A15 IOSTANDARD LVCMOS33 } [get\_ports { XA\_P[2] }]; #IO\_L8P\_T1\_AD10P\_15 Sch=xa\_p[2]

#set\_property -dict { PACKAGE\_PIN B17 IOSTANDARD LVCMOS33 } [get\_ports { XA\_N[3] }]; #IO\_L7N\_T1\_AD2N\_15 Sch=xa\_n[3]

#set\_property -dict { PACKAGE\_PIN B16 IOSTANDARD LVCMOS33 } [get\_ports { XA\_P[3] }]; #IO\_L7P\_T1\_AD2P\_15 Sch=xa\_p[3]

#set\_property -dict { PACKAGE\_PIN A18 IOSTANDARD LVCMOS33 } [get\_ports { XA\_N[4] }]; #IO\_L10N\_T1\_AD11N\_15 Sch=xa\_n[4]

#set\_property -dict { PACKAGE\_PIN B18 IOSTANDARD LVCMOS33 } [get\_ports { XA\_P[4] }]; #IO\_L10P\_T1\_AD11P\_15 Sch=xa\_p[4]

##VGA Connector

#set\_property -dict { PACKAGE\_PIN A3 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_R[0] }]; #IO\_L8N\_T1\_AD14N\_35 Sch=vga\_r[0]

#set\_property -dict { PACKAGE\_PIN B4 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_R[1] }]; #IO\_L7N\_T1\_AD6N\_35 Sch=vga\_r[1]

#set\_property -dict { PACKAGE\_PIN C5 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_R[2] }]; #IO\_L1N\_T0\_AD4N\_35 Sch=vga\_r[2]

#set\_property -dict { PACKAGE\_PIN A4 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_R[3] }]; #IO\_L8P\_T1\_AD14P\_35 Sch=vga\_r[3]

#set\_property -dict { PACKAGE\_PIN C6 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_G[0] }]; #IO\_L1P\_T0\_AD4P\_35 Sch=vga\_g[0]

#set\_property -dict { PACKAGE\_PIN A5 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_G[1] }]; #IO\_L3N\_T0\_DQS\_AD5N\_35 Sch=vga\_g[1]

#set\_property -dict { PACKAGE\_PIN B6 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_G[2] }]; #IO\_L2N\_T0\_AD12N\_35 Sch=vga\_g[2]

#set\_property -dict { PACKAGE\_PIN A6 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_G[3] }]; #IO\_L3P\_T0\_DQS\_AD5P\_35 Sch=vga\_g[3]

#set\_property -dict { PACKAGE\_PIN B7 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_B[0] }]; #IO\_L2P\_T0\_AD12P\_35 Sch=vga\_b[0]

#set\_property -dict { PACKAGE\_PIN C7 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_B[1] }]; #IO\_L4N\_T0\_35 Sch=vga\_b[1]

#set\_property -dict { PACKAGE\_PIN D7 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_B[2] }]; #IO\_L6N\_T0\_VREF\_35 Sch=vga\_b[2]

#set\_property -dict { PACKAGE\_PIN D8 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_B[3] }]; #IO\_L4P\_T0\_35 Sch=vga\_b[3]

#set\_property -dict { PACKAGE\_PIN B11 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_HS }]; #IO\_L4P\_T0\_15 Sch=vga\_hs

#set\_property -dict { PACKAGE\_PIN B12 IOSTANDARD LVCMOS33 } [get\_ports { VGA\_VS }]; #IO\_L3N\_T0\_DQS\_AD1N\_15 Sch=vga\_vs

##Micro SD Connector

#set\_property -dict { PACKAGE\_PIN E2 IOSTANDARD LVCMOS33 } [get\_ports { SD\_RESET }]; #IO\_L14P\_T2\_SRCC\_35 Sch=sd\_reset

#set\_property -dict { PACKAGE\_PIN A1 IOSTANDARD LVCMOS33 } [get\_ports { SD\_CD }]; #IO\_L9N\_T1\_DQS\_AD7N\_35 Sch=sd\_cd

#set\_property -dict { PACKAGE\_PIN B1 IOSTANDARD LVCMOS33 } [get\_ports { SD\_SCK }]; #IO\_L9P\_T1\_DQS\_AD7P\_35 Sch=sd\_sck

#set\_property -dict { PACKAGE\_PIN C1 IOSTANDARD LVCMOS33 } [get\_ports { SD\_CMD }]; #IO\_L16N\_T2\_35 Sch=sd\_cmd

#set\_property -dict { PACKAGE\_PIN C2 IOSTANDARD LVCMOS33 } [get\_ports { SD\_DAT[0] }]; #IO\_L16P\_T2\_35 Sch=sd\_dat[0]

#set\_property -dict { PACKAGE\_PIN E1 IOSTANDARD LVCMOS33 } [get\_ports { SD\_DAT[1] }]; #IO\_L18N\_T2\_35 Sch=sd\_dat[1]

#set\_property -dict { PACKAGE\_PIN F1 IOSTANDARD LVCMOS33 } [get\_ports { SD\_DAT[2] }]; #IO\_L18P\_T2\_35 Sch=sd\_dat[2]

#set\_property -dict { PACKAGE\_PIN D2 IOSTANDARD LVCMOS33 } [get\_ports { SD\_DAT[3] }]; #IO\_L14N\_T2\_SRCC\_35 Sch=sd\_dat[3]

##Accelerometer

#set\_property -dict { PACKAGE\_PIN E15 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_MISO }]; #IO\_L11P\_T1\_SRCC\_15 Sch=acl\_miso

#set\_property -dict { PACKAGE\_PIN F14 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_MOSI }]; #IO\_L5N\_T0\_AD9N\_15 Sch=acl\_mosi

#set\_property -dict { PACKAGE\_PIN F15 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_SCLK }]; #IO\_L14P\_T2\_SRCC\_15 Sch=acl\_sclk

#set\_property -dict { PACKAGE\_PIN D15 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_CSN }]; #IO\_L12P\_T1\_MRCC\_15 Sch=acl\_csn

#set\_property -dict { PACKAGE\_PIN B13 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_INT[1] }]; #IO\_L2P\_T0\_AD8P\_15 Sch=acl\_int[1]

#set\_property -dict { PACKAGE\_PIN C16 IOSTANDARD LVCMOS33 } [get\_ports { ACL\_INT[2] }]; #IO\_L20P\_T3\_A20\_15 Sch=acl\_int[2]

##Temperature Sensor

#set\_property -dict { PACKAGE\_PIN C14 IOSTANDARD LVCMOS33 } [get\_ports { TMP\_SCL }]; #IO\_L1N\_T0\_AD0N\_15 Sch=tmp\_scl

#set\_property -dict { PACKAGE\_PIN C15 IOSTANDARD LVCMOS33 } [get\_ports { TMP\_SDA }]; #IO\_L12N\_T1\_MRCC\_15 Sch=tmp\_sda

#set\_property -dict { PACKAGE\_PIN D13 IOSTANDARD LVCMOS33 } [get\_ports { TMP\_INT }]; #IO\_L6N\_T0\_VREF\_15 Sch=tmp\_int

#set\_property -dict { PACKAGE\_PIN B14 IOSTANDARD LVCMOS33 } [get\_ports { TMP\_CT }]; #IO\_L2N\_T0\_AD8N\_15 Sch=tmp\_ct

##Omnidirectional Microphone

#set\_property -dict { PACKAGE\_PIN J5 IOSTANDARD LVCMOS33 } [get\_ports { M\_CLK }]; #IO\_25\_35 Sch=m\_clk

#set\_property -dict { PACKAGE\_PIN H5 IOSTANDARD LVCMOS33 } [get\_ports { M\_DATA }]; #IO\_L24N\_T3\_35 Sch=m\_data

#set\_property -dict { PACKAGE\_PIN F5 IOSTANDARD LVCMOS33 } [get\_ports { M\_LRSEL }]; #IO\_0\_35 Sch=m\_lrsel

##PWM Audio Amplifier

#set\_property -dict { PACKAGE\_PIN A11 IOSTANDARD LVCMOS33 } [get\_ports { AUD\_PWM }]; #IO\_L4N\_T0\_15 Sch=aud\_pwm

#set\_property -dict { PACKAGE\_PIN D12 IOSTANDARD LVCMOS33 } [get\_ports { AUD\_SD }]; #IO\_L6P\_T0\_15 Sch=aud\_sd

##USB-RS232 Interface

#set\_property -dict { PACKAGE\_PIN C4 IOSTANDARD LVCMOS33 } [get\_ports { UART\_TXD\_IN }]; #IO\_L7P\_T1\_AD6P\_35 Sch=uart\_txd\_in

#set\_property -dict { PACKAGE\_PIN D4 IOSTANDARD LVCMOS33 } [get\_ports { UART\_RXD\_OUT }]; #IO\_L11N\_T1\_SRCC\_35 Sch=uart\_rxd\_out

#set\_property -dict { PACKAGE\_PIN D3 IOSTANDARD LVCMOS33 } [get\_ports { UART\_CTS }]; #IO\_L12N\_T1\_MRCC\_35 Sch=uart\_cts

#set\_property -dict { PACKAGE\_PIN E5 IOSTANDARD LVCMOS33 } [get\_ports { UART\_RTS }]; #IO\_L5N\_T0\_AD13N\_35 Sch=uart\_rts

##USB HID (PS/2)

#set\_property -dict { PACKAGE\_PIN F4 IOSTANDARD LVCMOS33 } [get\_ports { PS2\_CLK }]; #IO\_L13P\_T2\_MRCC\_35 Sch=ps2\_clk

#set\_property -dict { PACKAGE\_PIN B2 IOSTANDARD LVCMOS33 } [get\_ports { PS2\_DATA }]; #IO\_L10N\_T1\_AD15N\_35 Sch=ps2\_data

##SMSC Ethernet PHY

#set\_property -dict { PACKAGE\_PIN C9 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_MDC }]; #IO\_L11P\_T1\_SRCC\_16 Sch=eth\_mdc

#set\_property -dict { PACKAGE\_PIN A9 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_MDIO }]; #IO\_L14N\_T2\_SRCC\_16 Sch=eth\_mdio

#set\_property -dict { PACKAGE\_PIN B3 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_RSTN }]; #IO\_L10P\_T1\_AD15P\_35 Sch=eth\_rstn

#set\_property -dict { PACKAGE\_PIN D9 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_CRSDV }]; #IO\_L6N\_T0\_VREF\_16 Sch=eth\_crsdv

#set\_property -dict { PACKAGE\_PIN C10 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_RXERR }]; #IO\_L13N\_T2\_MRCC\_16 Sch=eth\_rxerr

#set\_property -dict { PACKAGE\_PIN C11 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_RXD[0] }]; #IO\_L13P\_T2\_MRCC\_16 Sch=eth\_rxd[0]

#set\_property -dict { PACKAGE\_PIN D10 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_RXD[1] }]; #IO\_L19N\_T3\_VREF\_16 Sch=eth\_rxd[1]

#set\_property -dict { PACKAGE\_PIN B9 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_TXEN }]; #IO\_L11N\_T1\_SRCC\_16 Sch=eth\_txen

#set\_property -dict { PACKAGE\_PIN A10 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_TXD[0] }]; #IO\_L14P\_T2\_SRCC\_16 Sch=eth\_txd[0]

#set\_property -dict { PACKAGE\_PIN A8 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_TXD[1] }]; #IO\_L12N\_T1\_MRCC\_16 Sch=eth\_txd[1]

#set\_property -dict { PACKAGE\_PIN D5 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_REFCLK }]; #IO\_L11P\_T1\_SRCC\_35 Sch=eth\_refclk

#set\_property -dict { PACKAGE\_PIN B8 IOSTANDARD LVCMOS33 } [get\_ports { ETH\_INTN }]; #IO\_L12P\_T1\_MRCC\_16 Sch=eth\_intn

##Quad SPI Flash

#set\_property -dict { PACKAGE\_PIN K17 IOSTANDARD LVCMOS33 } [get\_ports { QSPI\_DQ[0] }]; #IO\_L1P\_T0\_D00\_MOSI\_14 Sch=qspi\_dq[0]

#set\_property -dict { PACKAGE\_PIN K18 IOSTANDARD LVCMOS33 } [get\_ports { QSPI\_DQ[1] }]; #IO\_L1N\_T0\_D01\_DIN\_14 Sch=qspi\_dq[1]

#set\_property -dict { PACKAGE\_PIN L14 IOSTANDARD LVCMOS33 } [get\_ports { QSPI\_DQ[2] }]; #IO\_L2P\_T0\_D02\_14 Sch=qspi\_dq[2]

#set\_property -dict { PACKAGE\_PIN M14 IOSTANDARD LVCMOS33 } [get\_ports { QSPI\_DQ[3] }]; #IO\_L2N\_T0\_D03\_14 Sch=qspi\_dq[3]

#set\_property -dict { PACKAGE\_PIN L13 IOSTANDARD LVCMOS33 } [get\_ports { QSPI\_CSN }]; #IO\_L6P\_T0\_FCS\_B\_14 Sch=qspi\_csn