`timescale 1ns / 1ps

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

//- Verilog Model: univ\_sseg.v

//-

//- Company: Ratface Engineering

//- Engineer: James Ratner

//-

//- Description: Special seven segment display driver. This file

//- interprets the input(s) as an unsigned binary number

//- and displays the result on the four seven-segment

//- displays on the development board. This module implements

//- the required multiplexing of the display based

//- upon the CLK input. For this module, the CLK frequency

//- is expected to be in 100MHz but will work for other

//- relatively fast frequencies. The CLK input connects to a

//- clock signal from the development board; you may need to

//- adjust the clock divider if there are display issues.

//-

//- The description below describes the device's control inputs:

//-

//- valid: if valid = 0, four dashes are displayed

//- if valid = 1, decimal number appears on display

//-

//- dp\_en: if dp\_oe = 0, no decimal point is displayed

//- if dp\_oe = 1, displays one dp according to dp\_sel inputs

//-

//- dp\_sel: if dp = 00, dp displayed on right-most 7-seg display

//- if dp = 01, dp displayed on middle-right 7-seg display

//- if dp = 10, dp displayed on middle-left 7-seg display

//- if dp = 11, dp displayed on left-most 7-seg display

//- NOTE: only one dp is active at any given time

//-

//- mod\_sel: if sel = 00, displays one count [0,255] with optional '-' sign

//- if sel = 01, displays two counts [0,99] (no sign)

//- if sel = 10, displays one count [0,9999] (no sign)

//- if sel = 11, displays average academic administrator IQ

//-

//- sign: if sign = 0, no minus sign appears

//- if sign = 1, minus sign appears on left-most display

//-

//- cnt1: count for mod\_sel = 00 ([0,255]) & mod\_sel = 10 [0,9999]

//- count for left-most 2-digit count mod sel = 01 mode([0,99])

//- NOTE: 14-bit count

//-

//- cnt2: right-most 2-digit count ([0,99]) for mod\_sel = 01

//- ignored in all other mod\_sel values

//-

//-

//- USEAGE (instantiation template)

//-

//- univ\_sseg my\_univ\_sseg (

//- .cnt1 (),

//- .cnt2 (),

//- .valid (),

//- .dp\_en (),

//- .dp\_sel (),

//- .mod\_sel (),

//- .sign (),

//- .clk (),

//- .ssegs (),

//- .disg\_en () );

//-

//- Revision History:

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

//- Created: 07-01-2018 v1.00

//- 10-17-2018 v1.01 fixed system Verilog conversion error

//- 10-24-2018 v1.02 added instantiation template to header

//-

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

module univ\_sseg(

input [13:0] cnt1,

input [6:0] cnt2,

input valid,

input dp\_en,

input [1:0] dp\_sel,

input [1:0] mod\_sel,

input sign,

input clk,

output reg [7:0] ssegs,

output reg [3:0] disp\_en

);

wire [3:0] c1\_d3; // nodes for cnt1 convert results

wire [3:0] c1\_d2;

wire [3:0] c1\_d1;

wire [3:0] c1\_d0;

wire [3:0] c2\_d1; // nodes for cnt2 convert results

wire [3:0] c2\_d0;

reg [3:0] A\_1000\_digit; // final digit-based BCD values

reg [3:0] B\_100\_digit;

reg [3:0] C\_10\_digit;

reg [3:0] D\_1\_digit;

reg [3:0] final\_BCD;

reg [1:0] m\_cnt; // multiplex counter output

reg [7:0] seg\_val; // intermediate segment value

// instantiate 14-bit code converter

cnt\_convert\_14b CC\_14(cnt1, mod\_sel, c1\_d3, c1\_d2, c1\_d1, c1\_d0);

// instantiate 7-bit code converter

cnt\_convert\_7b CC\_7(cnt2, c2\_d1, c2\_d0);

// clock divider

clk\_divder CLK\_DIV (clk,sclk);

//- handle 1000's digit

//- mode 0: case 0 & 4

//- mode 1: case 1 & 5

//- mode 2: case 2 & 6

//- mode 3: case 3 & 7

always @ (sign,mod\_sel,c1\_d1,c1\_d3)

begin

case ({sign,mod\_sel})

0: A\_1000\_digit = 'hF;

1: A\_1000\_digit = c1\_d1;

2: A\_1000\_digit = c1\_d3;

3: A\_1000\_digit = 0;

4: A\_1000\_digit = 'hA; // dash

5: A\_1000\_digit = c1\_d1;

6: A\_1000\_digit = c1\_d3;

7: A\_1000\_digit = 0;

default A\_1000\_digit = 'hA;

endcase

end

//- handle 100's digit

always @ (mod\_sel,c1\_d2,c1\_d0)

begin

case (mod\_sel)

0: B\_100\_digit = c1\_d2;

1: B\_100\_digit = c1\_d0;

2: B\_100\_digit = c1\_d2;

3: B\_100\_digit = 0;

default B\_100\_digit = 0;

endcase

end

//- handle 10's digit

always @ (mod\_sel,c1\_d1,c2\_d1)

begin

case (mod\_sel)

0: C\_10\_digit = c1\_d1;

1: C\_10\_digit = c2\_d1;

2: C\_10\_digit = c1\_d1;

3: C\_10\_digit = 0;

default C\_10\_digit = 0;

endcase

end

//- handle 1's digit

always @ (mod\_sel,c1\_d0,c2\_d0)

begin

case (mod\_sel)

0: D\_1\_digit = c1\_d0;

1: D\_1\_digit = c2\_d0;

2: D\_1\_digit = c1\_d0;

3: D\_1\_digit = 0;

default D\_1\_digit = 0;

endcase

end

//- MUX to handle digits

always @ (m\_cnt, A\_1000\_digit, B\_100\_digit, C\_10\_digit, D\_1\_digit)

begin

case (m\_cnt)

3: final\_BCD = D\_1\_digit;

2: final\_BCD = C\_10\_digit;

1: final\_BCD = B\_100\_digit;

0: final\_BCD = A\_1000\_digit;

default final\_BCD = 0;

endcase

end

//- standard decoder for display multiplex

always @ (m\_cnt)

begin

case (m\_cnt)

0: disp\_en = 4'b1110;

1: disp\_en = 4'b1101;

2: disp\_en = 4'b1011;

3: disp\_en = 4'b0111;

default disp\_en = 0;

endcase

end

//- 7 seg decoder for display multiplex

always @ (final\_BCD)

begin

case (final\_BCD)

0: seg\_val = 'h03; // one

1: seg\_val = 'h9F; // two

2: seg\_val = 'h25; // three

3: seg\_val = 'h0D;

4: seg\_val = 'h99;

5: seg\_val = 'h49;

6: seg\_val = 'h41;

7: seg\_val = 'h1F;

8: seg\_val = 'h01;

9: seg\_val = 'h09;

10: seg\_val = 'hFD; // dash

15: seg\_val = 'hFF; // off

default seg\_val = 0;

endcase

end

//- handle the decimal point stuff

always @ (dp\_sel, m\_cnt, dp\_en, seg\_val, valid)

begin

if (valid == 0) //- handles valid

ssegs = 'hFD;

else

begin

if (dp\_sel == m\_cnt)

ssegs = {seg\_val[7:1],~dp\_en};

else

ssegs = seg\_val;

end

end

//- counter that drives the mulitplexed display

always @ (posedge sclk)

begin

m\_cnt <= m\_cnt + 1'b1;

end

endmodule

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

//- Module: clk\_divider

//-

//- Description: parameterized clock divder

//-

//- USAGE: (example overrides width default)

//-

//- rca\_nb #(16) MY\_RCA (

//- .clockin (my\_clk\_in),

//- .clockout (my\_clk\_out)

//- );

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

module clk\_divder(clockin, clockout);

input clockin;

output wire clockout;

parameter n = 13;

reg [n:0] count;

always@(posedge clockin)

begin

count <= count + 1;

end

assign clockout = count[n];

endmodule

//- converts 14-bit binary to 4 digit decimal

module cnt\_convert\_14b(

input [13:0] cnt,

input [1:0] sel,

output reg [3:0] c1\_d3,

output reg [3:0] c1\_d2,

output reg [3:0] c1\_d1,

output reg [3:0] c1\_d0

);

reg [13:0] cnt\_new; // working value

reg [13:0] cnt\_new\_w; // working value

//- BCD intermediates for digits

reg [3:0] d3; // 1000's digit

reg [3:0] d2; // 100's digit

reg [3:0] d1; // 10's digit

reg [3:0] d0; // 1's digit

//- clear unused bits on per mode basis

always @ (cnt,sel)

case(sel)

0: cnt\_new = {6'b000000,cnt[7:0]}; // clear zeros for [0,255] mode

1: cnt\_new = {7'b00000000,cnt[6:0]}; // clear zeros for [0,99] mode

2: cnt\_new = cnt; // 14-bit mode

3: cnt\_new = cnt; // unused mode

default cnt\_new = cnt;

endcase

always @ (cnt\_new)

begin

d3=0; d2=0; d1=0; d0=0; // pre-assign digits

cnt\_new\_w = cnt\_new; // copy value

//- find 1000's digit

if (cnt\_new\_w > 8999) begin d3=9; cnt\_new\_w = cnt\_new\_w - 9000; end

else if (cnt\_new\_w > 7999) begin d3=8; cnt\_new\_w = cnt\_new\_w - 8000; end

else if (cnt\_new\_w > 6999) begin d3=7; cnt\_new\_w = cnt\_new\_w - 7000; end

else if (cnt\_new\_w > 5999) begin d3=6; cnt\_new\_w = cnt\_new\_w - 6000; end

else if (cnt\_new\_w > 4999) begin d3=5; cnt\_new\_w = cnt\_new\_w - 5000; end

else if (cnt\_new\_w > 3999) begin d3=4; cnt\_new\_w = cnt\_new\_w - 4000; end

else if (cnt\_new\_w > 2999) begin d3=3; cnt\_new\_w = cnt\_new\_w - 3000; end

else if (cnt\_new\_w > 1999) begin d3=2; cnt\_new\_w = cnt\_new\_w - 2000; end

else if (cnt\_new\_w > 999) begin d3=1; cnt\_new\_w = cnt\_new\_w - 1000; end

else begin d3=0; end

//- find 100's digit

if (cnt\_new\_w > 899) begin d2=9; cnt\_new\_w = cnt\_new\_w - 900; end

else if (cnt\_new\_w > 799) begin d2=8; cnt\_new\_w = cnt\_new\_w - 800; end

else if (cnt\_new\_w > 699) begin d2=7; cnt\_new\_w = cnt\_new\_w - 700; end

else if (cnt\_new\_w > 599) begin d2=6; cnt\_new\_w = cnt\_new\_w - 600; end

else if (cnt\_new\_w > 499) begin d2=5; cnt\_new\_w = cnt\_new\_w - 500; end

else if (cnt\_new\_w > 399) begin d2=4; cnt\_new\_w = cnt\_new\_w - 400; end

else if (cnt\_new\_w > 299) begin d2=3; cnt\_new\_w = cnt\_new\_w - 300; end

else if (cnt\_new\_w > 199) begin d2=2; cnt\_new\_w = cnt\_new\_w - 200; end

else if (cnt\_new\_w > 99) begin d2=1; cnt\_new\_w = cnt\_new\_w - 100; end

else begin d2=0; end

//- find 10's digit

if (cnt\_new\_w > 89) begin d1=9; cnt\_new\_w = cnt\_new\_w - 90; end

else if (cnt\_new\_w > 79) begin d1=8; cnt\_new\_w = cnt\_new\_w - 80; end

else if (cnt\_new\_w > 69) begin d1=7; cnt\_new\_w = cnt\_new\_w - 70; end

else if (cnt\_new\_w > 59) begin d1=6; cnt\_new\_w = cnt\_new\_w - 60; end

else if (cnt\_new\_w > 49) begin d1=5; cnt\_new\_w = cnt\_new\_w - 50; end

else if (cnt\_new\_w > 39) begin d1=4; cnt\_new\_w = cnt\_new\_w - 40; end

else if (cnt\_new\_w > 29) begin d1=3; cnt\_new\_w = cnt\_new\_w - 30; end

else if (cnt\_new\_w > 19) begin d1=2; cnt\_new\_w = cnt\_new\_w - 20; end

else if (cnt\_new\_w > 9) begin d1=1; cnt\_new\_w = cnt\_new\_w - 10; end

else begin d1=0; end

//- assign 1's digit

d0 = cnt\_new\_w[3:0];

end

//- handles the lead zero blanking

always @ (cnt\_new, d3,d2,d1,d0)

begin

if (cnt\_new < 1000) // 1000's digit

c1\_d3 = 'hF;

else

c1\_d3 = d3;

if (cnt\_new < 100) // 100's digit

c1\_d2 = 'hF;

else

c1\_d2 = d2;

if (cnt\_new < 10) // 10's digit

c1\_d1 = 'hF;

else

c1\_d1 = d1;

c1\_d0 = d0; // 1's digit never blank

end

endmodule

//- converts 7-bit binary to 2 digit decimal

module cnt\_convert\_7b(

input [6:0] cnt,

output reg [3:0] c2\_d1,

output reg [3:0] c2\_d0

);

reg [7:0] cnt\_new; // working value

reg [7:0] cnt\_new\_w; // working value

// BCD intermediates

reg [3:0] d1; // 10's digit

reg [3:0] d0; // 1's digit

//- decimal (2 digit) to BCD conversion

always @ (cnt\_new)

begin

d1=0; d0=0;

cnt\_new\_w = cnt;

// handle 10's digit

if (cnt\_new\_w > 89) begin d1=9; cnt\_new\_w = cnt\_new\_w - 90; end

else if (cnt\_new\_w > 79) begin d1=8; cnt\_new\_w = cnt\_new\_w - 80; end

else if (cnt\_new\_w > 69) begin d1=7; cnt\_new\_w = cnt\_new\_w - 70; end

else if (cnt\_new\_w > 59) begin d1=6; cnt\_new\_w = cnt\_new\_w - 60; end

else if (cnt\_new\_w > 49) begin d1=5; cnt\_new\_w = cnt\_new\_w - 50; end

else if (cnt\_new\_w > 39) begin d1=4; cnt\_new\_w = cnt\_new\_w - 40; end

else if (cnt\_new\_w > 29) begin d1=3; cnt\_new\_w = cnt\_new\_w - 30; end

else if (cnt\_new\_w > 19) begin d1=2; cnt\_new\_w = cnt\_new\_w - 20; end

else if (cnt\_new\_w > 9) begin d1=1; cnt\_new\_w = cnt\_new\_w - 10; end

else begin d1=0; end

// handle 1's digit

d0 = cnt\_new\_w[3:0];

end

// handles the lead zero blanking

always @ (cnt, d1, d0)

begin

if (cnt < 10) // 10's digit

c2\_d1 = 'hF;

else

c2\_d1 = d1;

c2\_d0 = d0; // 1's digit never blank

end

endmodule