module lcd\_driver(

input CLOCK\_50, // 50 MHz clock

input [17:0] SW, // Toggle Switch[17:0]

input K0\_clk, //Control Display

output LCD\_ON, // LCD Power ON/OFF

output LCD\_BLON, // LCD Back Light ON/OFF

output LCD\_RW, // LCD Read/Write Select, 0 = Write, 1 = Read

output LCD\_EN, // LCD Enable

output LCD\_RS, // LCD Command/Data Select, 0 = Command, 1 = Data

inout [7:0] LCD\_DATA // LCD Data bus 8 bits

);

// reset delay gives some time for peripherals to initialize

wire DLY\_RST;

Reset\_Delay r0( .iCLK(CLOCK\_50),.oRESET(DLY\_RST) );

// turn LCD ON

assign LCD\_ON = 1'b1;

assign LCD\_BLON = 1'b1;

//Store SW

wire [3:0] hex3,hex2,hex1,hex0;

assign hex3 = SW[17:14];

assign hex2 = SW[13:10];

assign hex1 = SW[7:4];

assign hex0 = SW[3:0];

LCD\_Display u1(

.iCLK\_50MHZ(CLOCK\_50),

.iRST\_N(DLY\_RST),

.K0\_clk(K0\_clk),

.hex0(hex0),

.hex1(hex1),

.hex2(hex2),

.hex3(hex3),

.DATA\_BUS(LCD\_DATA),

.LCD\_RW(LCD\_RW),

.LCD\_E(LCD\_EN),

.LCD\_RS(LCD\_RS)

);

endmodule

//LCD\_Display.v

module LCD\_Display(iCLK\_50MHZ, iRST\_N,K0\_clk, hex3,hex2,hex1, hex0,

LCD\_RS,LCD\_E,LCD\_RW,DATA\_BUS);

input iCLK\_50MHZ, iRST\_N;

input K0\_clk;

input [3:0] hex3,hex2,hex1, hex0;

output LCD\_RS, LCD\_E, LCD\_RW;

output [7:0] DATA\_BUS;//change inout to output

parameter

HOLD = 4'h0,

FUNC\_SET = 4'h1,

DISPLAY\_ON = 4'h2,

MODE\_SET = 4'h3,

Print\_String = 4'h4,

LINE2 = 4'h5,

RETURN\_HOME = 4'h6,

DROP\_LCD\_E = 4'h7,

RESET1 = 4'h8,

RESET2 = 4'h9,

RESET3 = 4'ha,

DISPLAY\_OFF = 4'hb,

DISPLAY\_CLEAR = 4'hc;

reg [3:0] state, next\_command;

// Enter new ASCII hex data above for LCD Display

reg [7:0] DATA\_BUS\_VALUE;

wire [7:0] Next\_Char;

reg [19:0] CLK\_COUNT\_400HZ;

reg [4:0] CHAR\_COUNT;

reg CLK\_400HZ, LCD\_RW\_INT, LCD\_E, LCD\_RS;

// BIDIRECTIONAL TRI STATE LCD DATA BUS

assign DATA\_BUS = (LCD\_RW\_INT? 8'bZZZZZZZZ: DATA\_BUS\_VALUE);

LCD\_display\_string u1(

.index(CHAR\_COUNT),

.out(Next\_Char),

.K0\_clk(K0\_clk),

.hex3(hex3),

.hex2(hex2),

.hex1(hex1),

.hex0(hex0));

assign LCD\_RW = LCD\_RW\_INT;

always @(posedge iCLK\_50MHZ or negedge iRST\_N)

if (!iRST\_N)

begin

CLK\_COUNT\_400HZ <= 20'h00000;

CLK\_400HZ <= 1'b0;

end

///

else if (CLK\_COUNT\_400HZ < 20'h0F424)

begin

CLK\_COUNT\_400HZ <= CLK\_COUNT\_400HZ + 1'b1;

end

else

begin

CLK\_COUNT\_400HZ <= 20'h00000;

CLK\_400HZ <= ~CLK\_400HZ;

end

// State Machine to send commands and data to LCD DISPLAY

always @(posedge CLK\_400HZ or negedge iRST\_N)

if (!iRST\_N)

begin

state <= RESET1;

end

else

case (state)

RESET1:

// Set Function to 8-bit transfer and 2 line display with 5x8 Font size

// see Hitachi HD44780 family data sheet for LCD command and timing details

begin

LCD\_E <= 1'b1;

LCD\_RS <= 1'b0;

LCD\_RW\_INT <= 1'b0;

DATA\_BUS\_VALUE <= 8'h38;

state <= DROP\_LCD\_E;

next\_command <= RESET2;

CHAR\_COUNT <= 5'b00000;

end

RESET2:

begin

LCD\_E <= 1'b1;

LCD\_RS <= 1'b0;

LCD\_RW\_INT <= 1'b0;

DATA\_BUS\_VALUE <= 8'h38;

state <= DROP\_LCD\_E;

next\_command <= RESET3;

CHAR\_COUNT <= 5'b00000;

end

RESET3:

begin

LCD\_E <= 1'b1;

LCD\_RS <= 1'b0;

LCD\_RW\_INT <= 1'b0;

DATA\_BUS\_VALUE <= 8'h38;

state <= DROP\_LCD\_E;

next\_command <= FUNC\_SET;

CHAR\_COUNT <= 5'b00000;

end

FUNC\_SET:

begin

LCD\_E <= 1'b1;

LCD\_RS <= 1'b0;

LCD\_RW\_INT <= 1'b0;

DATA\_BUS\_VALUE <= 8'h38;

state <= DROP\_LCD\_E;

next\_command <= DISPLAY\_OFF;

end

// Turn off Display and Turn off cursor

DISPLAY\_OFF:

begin

LCD\_E <= 1'b1;

LCD\_RS <= 1'b0;

LCD\_RW\_INT <= 1'b0;

DATA\_BUS\_VALUE <= 8'h08;

state <= DROP\_LCD\_E;

next\_command <= DISPLAY\_CLEAR;

end

// Clear Display and Turn off cursor

DISPLAY\_CLEAR:

begin

LCD\_E <= 1'b1;

LCD\_RS <= 1'b0;

LCD\_RW\_INT <= 1'b0;

DATA\_BUS\_VALUE <= 8'h01;

state <= DROP\_LCD\_E;

next\_command <= MODE\_SET;

end

// Set write mode to auto increment address and move cursor to the right

MODE\_SET:

begin

LCD\_E <= 1'b1;

LCD\_RS <= 1'b0;

LCD\_RW\_INT <= 1'b0;

DATA\_BUS\_VALUE <= 8'h06;

state <= DROP\_LCD\_E;

next\_command <= DISPLAY\_ON;

end

// Turn on Display and Turn off cursor

DISPLAY\_ON:

begin

LCD\_E <= 1'b1;

LCD\_RS <= 1'b0;

LCD\_RW\_INT <= 1'b0;

DATA\_BUS\_VALUE <= 8'h0C;

state <= DROP\_LCD\_E;

next\_command <= Print\_String;

end

// Write ASCII hex character in first LCD character location

Print\_String:

begin

state <= DROP\_LCD\_E;

LCD\_E <= 1'b1;

LCD\_RS <= 1'b1;

LCD\_RW\_INT <= 1'b0;

// ASCII character to output

if (Next\_Char[7:4] != 4'h0)

DATA\_BUS\_VALUE <= Next\_Char;

// Convert 4-bit value to an ASCII hex digit

else if (Next\_Char[3:0] >9)

// ASCII A...F

DATA\_BUS\_VALUE <= {4'h4,Next\_Char[3:0]-4'h9};

else

// ASCII 0...9

DATA\_BUS\_VALUE <= {4'h3,Next\_Char[3:0]};

// Loop to send out 32 characters to LCD Display (16 by 2 lines)

if ((CHAR\_COUNT < 31) && (Next\_Char != 8'hFE))

CHAR\_COUNT <= CHAR\_COUNT + 1'b1;

else

CHAR\_COUNT <= 5'b00000;

// Jump to second line?

if (CHAR\_COUNT == 15)

next\_command <= LINE2;

// Return to first line?

else if ((CHAR\_COUNT == 31) || (Next\_Char == 8'hFE))

next\_command <= RETURN\_HOME;

else

next\_command <= Print\_String;

end

// Set write address to line 2 character 1

LINE2:

begin

LCD\_E <= 1'b1;

LCD\_RS <= 1'b0;

LCD\_RW\_INT <= 1'b0;

DATA\_BUS\_VALUE <= 8'hC0;

state <= DROP\_LCD\_E;

next\_command <= Print\_String;

end

// Return write address to first character postion on line 1

RETURN\_HOME:

begin

LCD\_E <= 1'b1;

LCD\_RS <= 1'b0;

LCD\_RW\_INT <= 1'b0;

DATA\_BUS\_VALUE <= 8'h80;

state <= DROP\_LCD\_E;

next\_command <= Print\_String;

end

// The next three states occur at the end of each command or data transfer to the LCD

// Drop LCD E line - falling edge loads inst/data to LCD controller

DROP\_LCD\_E:

begin

LCD\_E <= 1'b0;

state <= HOLD;

end

// Hold LCD inst/data valid after falling edge of E line

HOLD:

begin

state <= next\_command;

end

endcase

endmodule

/\*

|----------------|

|0x41 65 'A' 255 11

|abcdefghijkl

|----------------|

\*/

Module LCD\_display\_string(index,out,K0\_clk, hex3,hex2,hex0,hex1);

input K0\_clk;

input [4:0] index;

input [3:0] hex3,hex2,hex1,hex0;

output [7:0] out;

reg [7:0] out;

reg [7:0] dec;

reg [11:0] b;

reg [3:0] count=4'd0;

reg a;

reg [7:0] regs[15:0];

initial

begin

regs[15]= 8'h20;

regs[14]= 8'h20;

regs[13]= 8'h20;

regs[12]= 8'h20;

regs[11]= 8'h20;

regs[10]= 8'h20;

regs[9]= 8'h20;

regs[8]= 8'h20;

regs[7]= 8'h20;

regs[6]= 8'h20;

regs[5]= 8'h20;

regs[4]= 8'h20;

regs[3]= 8'h20;

regs[2]= 8'h20;

regs[1]= 8'h20;

regs[0]= 8'h20;

end

always @(negedge K0\_clk)

if(count == 16)

count <= 1'b0;

else if(!K0\_clk)

begin

regs[count] <= {hex3,hex2};

count<=count+1'b1;

end

// Line 1

always

case (index)

5'h00: out <= 8'h41;

5'h01: out <= 8'h53;

5'h02: out <= 8'h43;

5'h03: out <= 8'h49;

5'h04: out <= 8'h49;

5'h05: out <= 8'h3a;

5'h06: out <= 8'h30;

5'h07: out <= 8'h78;

5'h08: out <= {4'h0,hex1};

5'h09: out <= {4'h0,hex0};

5'h0b: out <= {hex1,hex0};

5'h0d:

begin

dec<={hex1,hex0};

b[11:8]=(dec/100)%10;

out <= {4'h0,b[11:8]};

end

5'h0e:

begin

dec<={hex1,hex0};

b[7:4]=(dec/10)%10;

out <= {4'h0,b[7:4]};

end

5'h0f:

begin

dec<={hex1,hex0};

b[3:0]=dec%10;

out <= {4'h0,b[3:0]};

end

// Line 2

5'h10: out <= regs[0];

5'h11: out <= regs[1];

5'h12: out <= regs[2];

5'h13: out <= regs[3];

5'h14: out <= regs[4];

5'h15: out <= regs[5];

5'h16: out <= regs[6];

5'h17: out <= regs[7];

5'h18: out <= regs[8];

5'h19: out <= regs[9];

5'h1a: out <= regs[10];

5'h1b: out <= regs[11];

5'h1c: out <= regs[12];

5'h1d: out <= regs[13];

5'h1e: out <= regs[14];

5'h1f: out <= regs[15];

default: out <= 8'h20;

endcase

endmodule

//reset\_delay.v

module Reset\_Delay(iCLK,oRESET);

input iCLK;

output reg oRESET;

reg [19:0] Cont;

always@(posedge iCLK)

begin

if(Cont!=20'hFFFFF)

begin

Cont <= Cont+1'b1;

oRESET <= 1'b0;

end

else

oRESET <= 1'b1;

end

endmodule