```
‘timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Walker Chan
//
// Create Date:    18:49:49 05/02/2007
// Design Name:
// Module Name:    laser_driver
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

// top level module for laser driver. Takes segment data from vector drawing module
// and outputs to laser. Also handles laser parameters (translation, rotation, etc.)
// input from the keyboard.
module laser_driver(clock, reset, pattern_source,
x1_in, y1_in, x2_in, y2_in, next_seg,
dac_data, dac_addr, dac_wr, laser_en,
key_dat, key_clk,
vga_start, vga_busy, vga_x, vga_y, vga_pixel, vga_wr);

input clock, reset;

// pattern
input pattern_source;
input [7:0] x1_in, y1_in, x2_in, y2_in;
output next_seg;

// dac
output [7:0] dac_data;
output [1:0] dac_addr;
output dac_wr;
output laser_en;
```

```
// keyboard
input key_dat;
input key_clk;

// vga
input vga_start;
output vga_busy;
output [9:0] vga_x;
output [8:0] vga_y;
output vga_pixel;
output vga_wr;

// interconnects
wire [7:0] x, y;
wire [7:0] newx, newy;
wire [7:0] a, b, c, d, xoffset, yoffset, xbias, ybias, speed;
wire strobe;
wire [3:0] param;
wire [1:0] op;

// keyboard input for laser parameters
wire [7:0]ascii;
wire ascii_ready;
assign strobe = ascii_ready;

ps2_ascii_input keyboard(clock, reset, key_clk, key_dat, ascii, ascii_ready);
keyboard_decoder keydec1(clock, reset, ascii, param, op);

// input from Huy or rom
laser_input laser_input1(clock, reset, pattern_source,
x1_in, y1_in, x2_in, y2_in, next_seg,
x, y, speed, laser_en);

// transformation
transformation trans1(clock, reset, a, b, c, d, xoffset, yoffset, x, y, newx, newy);

// dac
dac dac1(clock, reset, newx, newy, xbias, ybias, dac_data, dac_addr, dac_wr);

// laser parameters
laser_parameters laser_params(clock, reset, param, op, strobe,
a, b, c, d, xoffset, yoffset, xbias, ybias, speed,
vga_start, vga_busy, vga_x, vga_y, vga_pixel, vga_wr);
endmodule


module transformation(clock, reset, a, b, c, d, xoff, yoff, oldx, oldy, newx, newy);
input clock, reset;
input [7:0]oldx, oldy;
input signed [7:0] a, b, c, d, xoff, yoff;
output [7:0]newx, newy;
wire signed [7:0] trans_inx, trans_iny;
wire signed [15:0] trans_outx, trans_outy;

// shift to [-128, 127] range
assign trans_inx = oldx - 128;
assign trans_iny = oldy - 128;

// do transformation R' = AR + B
assign trans_outx = trans_inx * a + trans_iny * b;
assign trans_outy = trans_inx * c + trans_iny * d;

// truncate back to 8 bits, move back to [0,255] range
assign newx = trans_outx[15:8] + 128 + xoff;
assign newy = trans_outy[15:8] + 128 + yoff;
endmodule


////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:  Walker Chan
//
// Create Date:    17:43:32 05/02/2007
// Design Name:
// Module Name:    pattern
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module laser_input(clock, reset, pattern_source, x1_in, y1_in, x2_in, y2_in, next_seg, x
input clock, reset;
input pattern_source;
input [7:0] x1_in, y1_in, x2_in, y2_in;
output next_seg;
output [7:0] x, y; // --> transformation --> DACs
input [7:0] speed;
output laser_en;

// timer
wire end_sel;
laser_timer timer1(clock, reset, next_seg, end_sel, laser_en, speed);

// pattern rom for testing
wire [7:0] test_x1, test_y1, test_x2, test_y2;
pattern pattern1(clock, reset, next_seg, test_x1, test_y1, test_x2, test_y2);

// pattern source selection
wire [7:0] x1, y1, x2, y2;
assign x1 = pattern_source ? test_x1 : x1_in;
assign y1 = pattern_source ? test_y1 : y1_in;
assign x2 = pattern_source ? test_x2 : x2_in;
assign y2 = pattern_source ? test_y2 : y2_in;

// end point selection
assign x = end_sel ? x2 : x1;
assign y = end_sel ? y2 : y1;
endmodule


// reads sequential data out of a rom to generate a test pattern
module pattern(clock, reset, next, x1, y1, x2, y2);
input clock, reset;
input next;
output [7:0] x1, y1, x2, y2;

// number of lines in testpattern rom to read
```

```
parameter ROM_SIZE = 120;

wire [31:0] segment;
reg [7:0] addr;

// increment address counter
always @(posedge clock)
begin
if (reset)
addr <= 0;
else if (addr > ROM_SIZE)
addr <= 0;
else
if (next)
addr <= addr + 1;
end

// look up addr in the pattern rom
testpattern rom1 (addr, clock, segment);

// break up segment into the endpoints
assign x1 = segment[31:24];
assign y1 = segment[23:16];
assign x2 = segment[15:8];
assign y2 = segment[7:0];
endmodule


module laser_timer(clock, reset, next, mux, laser_en, speed);
input clock, reset;
output next, mux, laser_en;
input [7:0] speed;
reg [25:0] count;
reg next, mux, laser_en;
wire [25:0]max;
assign max = {speed, 10'b1111111111};

always @(posedge clock)
begin
next <= 0;
mux <= 0;
laser_en <= 0;
```

```
if(reset) count <= 0;
else count <= count + 1;

if(count == max)
begin
count <= 0;
next <= 1;
end
// generate signal to flip between endpoints
if (count > max/2) mux <= 1;

// generate signal to blank laser
// needs work
if (count > 100000 && count < max-10000) laser_en <= 1;
end
endmodule


////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:  Walker Chan
//
// Create Date:    17:37:03 04/25/2007
// Design Name:
// Module Name:    dac
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

// dac driver
module dac(clock, reset, ch1, ch2, ch3, ch4, data, addr, wr);
input clock;
input reset;
input [7:0] ch1;
input [7:0] ch2;
input [7:0] ch3;
input [7:0] ch4;
output [7:0] data;
output [1:0] addr;
output wr;

reg [7:0] data;
reg [1:0] addr;
reg wr;
reg [3:0] state;

parameter INIT = 0;
parameter CH1 = 1;
parameter EN1a = 2;
parameter EN1b = 3;
parameter HOLD1 = 4;
parameter CH2 = 5;
parameter EN2a = 6;
parameter EN2b = 7;
parameter HOLD2 = 8;
parameter CH3 = 9;
parameter EN3a = 10;
parameter EN3b = 11;
parameter HOLD3 = 12;
parameter CH4 = 13;
parameter EN4a = 14;
parameter EN4b = 15;
parameter HOLD4 = 16;


always @(posedge clock)
begin
if (reset)
state <= INIT;
else
begin
// defaults
wr <= 1;
```

```
case (state)
INIT:
state <= CH1;

// output channel 1
CH1:
begin
addr <= 0;
data <= ch1;
state <= EN1a;
end
EN1a:
begin
wr <= 0;
state <= EN1b;
end
EN1b:
begin
wr <= 0;
state <= HOLD1;
end
HOLD1:
state <= CH2;

// output channel 2
CH2:
begin
addr <= 1;
data <= ch2;
state <= EN2a;
end
EN2a:
begin
wr <= 0;
state <= EN2b;
end
EN2b:
begin
wr <= 0;
state <= HOLD2;
end
HOLD2:
```

```
state <= CH3;

// output channel 3
CH3:
begin
addr <= 2;
data <= ch3;
state <= EN3a;
end
EN3a:
begin
wr <= 0;
state <= EN3b;
end
EN3b:
begin
wr <= 0;
state <= HOLD3;
end
HOLD3:
state <= CH4;

// output channel 4
CH4:
begin
addr <= 3;
data <= ch4;
state <= EN4a;
end
EN4a:
begin
wr <= 0;
state <= EN4b;
end
EN4b:
begin
wr <= 0;
state <= HOLD4;
end
HOLD4:
state <= CH1;
```

```
default:
state <= INIT;
endcase
end
end
endmodule

‘timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:  Walker Chan
//
// Create Date:    19:38:47 05/02/2007
// Design Name:
// Module Name:    laser_parameters
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

// maintains laser paramters in registers. Performs an operation specified by
// op on a parameter specified by param when strobe is pulsed high for one
// clock cycle. Displays the parameters to the framebuffer.
module laser_parameters(clock, reset, param, op, strobe, a,
b, c, d, xoffset, yoffset, xbias, ybias, speed,
vga_start, vga_busy, vga_x, vga_y, vga_color, vga_wr);
input clock, reset;

// parameter modification
input [3:0] param;
input [1:0] op;
input strobe;
```

```
// parameter output
output signed [7:0] a, b, c, d; //transformation matrix
output signed [7:0] xoffset, yoffset;
output signed [7:0] xbias, ybias;
output [7:0] speed;

// vga
input vga_start;
output vga_busy;
output [9:0] vga_x;
output [8:0] vga_y;
output vga_color;
output vga_wr;

reg [7:0] a, b, c, d;
reg [7:0] xoffset, yoffset;
reg [7:0] xbias, ybias;
reg [7:0] speed;

// op
parameter RESET = 0;
parameter INC = 1;
parameter DEC = 2;

// params
parameter A = 0;
parameter B = 1;
parameter C = 2;
parameter D = 3;
parameter XOFF = 4;
parameter YOFF = 5;
parameter XBIAS = 6;
parameter YBIAS = 7;
parameter SPEED = 8;

always @(posedge clock)
begin
if (reset)
begin
a <= -59;
b <= 0;
```

```
c <= 0;
d <= 54;
xoffset <= 0;
yoffset <= 0;
xbias <= 0;
ybias <= 0;
speed <= 16;
end
else if (strobe)
begin
case (param)
A:
begin
case (op)
RESET: a <= 1;
INC: a <= a + 1;
DEC: a <= a - 1;
endcase
end
B:
begin
case (op)
RESET: b <= 0;
INC: b <= b + 1;
DEC: b <= b - 1;
endcase
end
C:
begin
case (op)
RESET: c <= 0;
INC: c <= c + 1;
DEC: c <= c - 1;
endcase
end
D:
begin
case (op)
RESET: d <= 1;
INC: d <= d + 1;
DEC: d <= d - 1;
endcase
```

```
end
XOFF:
begin
case (op)
RESET: xoffset <= 0;
INC: xoffset <= xoffset + 1;
DEC: xoffset <= xoffset - 1;
endcase
end
YOFF:
begin
case (op)
RESET: yoffset <= 0;
INC: yoffset <= yoffset + 1;
DEC: yoffset <= yoffset - 1;
endcase
end
XBIAS:
begin
case (op)
RESET: xbias <= 0;
INC: xbias <= xbias + 1;
DEC: xbias <= xbias - 1;
endcase
end
YBIAS:
begin
case (op)
RESET: ybias <= 0;
INC: ybias <= ybias + 1;
DEC: ybias <= ybias - 1;
endcase
end
SPEED:
begin
case (op)
RESET: speed <= 0;
INC: speed <= speed + 1;
DEC: speed <= speed - 1;
endcase
end
default:
```

```
begin
end
endcase
end
end

textdisplay disp1(clock, reset, vga_start, vga_busy, vga_x, vga_y, vga_color, vga_wr,
a, b, c, d, xoffset, yoffset, xbias, ybias, speed);
endmodule



// combinational logic to change an ascii value to a laser parameter index and
// an operation to be performed on the parameter
module keyboard_decoder(clock, reset, ascii, param, op);
input clock, reset;
input [7:0] ascii;
output [3:0] param;
output [1:0] op;

reg [3:0]param;
reg [1:0]op;

// op
parameter RESET = 0;
parameter INC = 1;
parameter DEC = 2;

// params
parameter A = 0;
parameter B = 1;
parameter C = 2;
parameter D = 3;
parameter XOFF = 4;
parameter YOFF = 5;
parameter XBIAS = 6;
parameter YBIAS = 7;
parameter SPEED = 8;

// keys
parameter KEY_A = 8'h41;
parameter KEY_D = 8'h44;
parameter KEY_W = 8'h57;
parameter KEY_S = 8'h53;
parameter KEY_PLUS = 8'h3d;
parameter KEY_MINUS = 8'h2d;
parameter KEY_U = 8'h55;
parameter KEY_I = 8'h49;
parameter KEY_O = 8'h4F;
parameter KEY_P = 8'h50;
parameter KEY_J = 8'h4A;
parameter KEY_K = 8'h4B;
parameter KEY_L = 8'h4C;
parameter KEY_SEMI = 8'h3B;
parameter KEY_Z = 8'h5A;
parameter KEY_X = 8'h58;
parameter KEY_C = 8'h43;
parameter KEY_V = 8'h56;


always @(posedge clock)
begin
case (ascii)
KEY_U:
begin
param <= A;
op <= INC;
end
KEY_I:
begin
param <= A;
op <= DEC;
end
KEY_O:
begin
param <= B;
op <= INC;
end
KEY_P:
begin
param <= B;
op <= DEC;
end
KEY_J:
```

```
begin
param <= C;
op <= INC;
end
KEY_K:
begin
param <= C;
op <= DEC;
end
KEY_L:
begin
param <= D;
op <= INC;
end
KEY_SEMI:
begin
param <= D;
op <= DEC;
end
KEY_A:
begin
param <= XOFF;
op <= INC;
end
KEY_D:
begin
param <= XOFF;
op <= DEC;
end
KEY_W:
begin
param <= YOFF;
op <= DEC;
end
KEY_S:
begin
param <= YOFF;
op <= INC;
end
KEY_PLUS:
begin
param <= SPEED;
```

```
op <= INC;
end
KEY_MINUS:
begin
param <= SPEED;
op <= DEC;
end
KEY_Z:
begin
param <= XBIAS;
op <= DEC;
end
KEY_X:
begin
param <= XBIAS;
op <= INC;
end
KEY_C:
begin
param <= YBIAS;
op <= DEC;
end
KEY_V:
begin
param <= YBIAS;
op <= INC;
end
default:
begin
param <= 9;
op <= 0;
end
endcase
end
endmodule


// outputs all laser dirver parameters to the VGA module using drawnumber and drawchar
module textdisplay(clock, reset, start, busy, x, y, pixel, wr,
a, b, c, d, xoffset, yoffset, xbias, ybias, speed);
input clock, reset;
input start;
```

```
output busy;
output [9:0] x;
output [8:0] y;
output pixel;
output wr;
input signed [7:0] a, b, c, d, xoffset, yoffset, xbias, ybias, speed;

reg select;
parameter CHAR = 1;
parameter NUM = 0;

reg [7:0] state;
parameter WAIT = 120;
// display transformation matrix
parameter A1 = 1;
parameter A2 = 2;
parameter A3 = 3;
parameter A4 = 4;
parameter B1 = 5;
parameter B2 = 6;
parameter B3 = 7;
parameter B4 = 8;
parameter C1 = 9;
parameter C2 = 10;
parameter C3 = 11;
parameter C4 = 12;
parameter D1 = 13;
parameter D2 = 14;
parameter D3 = 15;
parameter D4 = 16;
// display offsets
parameter E1 = 17;
parameter E2 = 18;
parameter E3 = 19;
parameter E4 = 20;
parameter F1 = 21;
parameter F2 = 22;
parameter F3 = 23;
parameter F4 = 24;
// display bias currents
parameter G1 = 25;
parameter G2 = 26;
parameter G3 = 27;
parameter G4 = 28;
parameter H1 = 29;
parameter H2 = 30;
parameter H3 = 31;
parameter H4 = 32;
// display period
parameter I1 = 33;
parameter I2 = 34;
parameter I3 = 35;
parameter I4 = 36;

reg [9:0] xorg;
reg [8:0] yorg;
reg num_start, char_start;
reg busy;
reg [7:0] number, char;
reg num_is_signed;

wire char_busy, num_busy;
wire [9:0]char_x, num_x;
wire [8:0]char_y, num_y;
wire char_pixel, num_pixel;
wire char_wr, num_wr;

always @(posedge clock)
begin
if (reset)
state <= WAIT;
else
begin
// defaults
busy <= 1;
char_start <= 0;
num_start <= 0;
num_is_signed <= 1;

case (state)
WAIT:
begin
busy <= 0;
if (start) state <= A1;
```

```
end
A1:
begin
number <= a;
xorg <= 70;
yorg <= 460;
select <= NUM;
num_start <= 1;
state <= state + 1;
end
A2: state <= state + 1;
A3: state <= state + 1;
A4: if (!num_busy) state <= B1;
B1:
begin
number <= b;
xorg <= 100;
yorg <= 460;
select <= NUM;
num_start <= 1;
state <= state + 1;
end
B2: state <= state + 1;
B3: state <= state + 1;
B4: if (!num_busy) state <= C1;
C1:
begin
number <= c;
xorg <= 70;
yorg <= 470;
select <= NUM;
num_start <= 1;
state <= state + 1;
end
C2: state <= state + 1;
C3: state <= state + 1;
C4: if (!num_busy) state <= D1;
D1:
begin
number <= d;
xorg <= 100;
yorg <= 470;
```

```
select <= NUM;
num_start <= 1;
state <= state + 1;
end
D2: state <= state + 1;
D3: state <= state + 1;
D4: if (!num_busy) state <= E1;
E1:
begin
number <= xoffset;
xorg <= 150;
yorg <= 470;
select <= NUM;
num_start <= 1;
state <= state + 1;
end
E2: state <= state + 1;
E3: state <= state + 1;
E4: if (!num_busy) state <= F1;
F1:
begin
number <= yoffset;
xorg <= 150;
yorg <= 460;
select <= NUM;
num_start <= 1;
state <= state + 1;
end
F2: state <= state + 1;
F3: state <= state + 1;
F4: if (!num_busy) state <= G1;
G1:
begin
number <= xbias;
xorg <= 200;
yorg <= 470;
select <= NUM;
num_start <= 1;
state <= state + 1;
end
G2: state <= state + 1;
G3: state <= state + 1;
```

```
G4: if (!num_busy) state <= H1;
H1:
begin
number <= ybias;
xorg <= 200;
yorg <= 460;
select <= NUM;
num_start <= 1;
state <= state + 1;
end
H2: state <= state + 1;
H3: state <= state + 1;
H4: if (!num_busy) state <= I1;
I1:
begin
num_is_signed <= 0;
number <= speed;
xorg <= 250;
yorg <= 465;
select <= NUM;
num_start <= 1;
state <= state + 1;
end
I2:
begin
state <= state + 1;
num_is_signed <= 0;
end
I3:
begin
state <= state + 1;
num_is_signed <= 0;
end
I4:
begin
num_is_signed <= 0;
if (!num_busy) state <= state + 1;
end
37: begin
char <= 33; // line icon
xorg <= 60;
yorg <= 40;
```

```
select <= CHAR;
char_start <= 1;
state <= state + 1;
end
38: state <= state + 1;
39: state <= state + 1;
40: if (!char_busy) state <= state + 1;
41: begin
char <= 32; // circle icon
xorg <= 190;
yorg <= 40;
select <= CHAR;
char_start <= 1;
state <= state + 1;
end
42: state <= state + 1;
43: state <= state + 1;
44: if (!char_busy) state <= state + 1;
45: begin
char <= 37; // right arrow
xorg <= 280;
yorg <= 60;
select <= CHAR;
char_start <= 1;
state <= state + 1;
end
46: state <= state + 1;
47: state <= state + 1;
48: if (!char_busy) state <= state + 1;
49: begin
char <= 35; // down arrow
xorg <= 320;
yorg <= 60;
select <= CHAR;
char_start <= 1;
state <= state + 1;
end
50: state <= state + 1;
51: state <= state + 1;
52: if (!char_busy) state <= state + 1;
53: begin
char <= 36; // left arrow -- don't need
```

```
xorg <= 320;
yorg <= 20;
select <= CHAR;
char_start <= 1;
state <= state + 1;
end
54: state <= state + 1;
55: state <= state + 1;
56: if (!char_busy) state <= state + 1;
57: begin
char <= 34; // up arrow
xorg <= 320;
yorg <= 20;
select <= CHAR;
char_start <= 1;
state <= state + 1;
end
58: state <= state + 1;
59: state <= state + 1;
60: if (!char_busy) state <= state + 1;
61: begin
char <= 36; // right arrow
xorg <= 360;
yorg <= 60;
select <= CHAR;
char_start <= 1;
state <= state + 1;
end
62: state <= state + 1;
63: state <= state + 1;
64: if (!char_busy) state <= state + 1;
65: begin
char <= 38; // rotate ccw
xorg <= 415;
yorg <= 40;
select <= CHAR;
char_start <= 1;
state <= state + 1;
end
66: state <= state + 1;
67: state <= state + 1;
68: if (!char_busy) state <= state + 1;
```

```
69: begin
char <= 39; // rotate cw
xorg <= 475;
yorg <= 40;
select <= CHAR;
char_start <= 1;
state <= state + 1;
end
70: state <= state + 1;
71: state <= state + 1;
72: if (!char_busy) state <= state + 1;
73: begin
char <= 22; // remove
xorg <= 580;
yorg <= 40;
select <= CHAR;
char_start <= 1;
state <= state + 1;
end
74: state <= state + 1;
75: state <= state + 1;
76: if (!char_busy) state <= WAIT;
endcase
end
end


drawchar char1(clock, reset, char_start, char_busy, char, xorg, yorg,
char_x, char_y, char_pixel, char_wr);
drawnumber num1(clock, reset, num_start, num_busy, number, num_is_signed, xorg, yorg,
num_x, num_y, num_pixel, num_wr);
assign x = select ? char_x : num_x;
assign y = select ? char_y : num_y;
assign pixel = select ? char_pixel : num_pixel;
assign wr = select ? char_wr : num_wr;
endmodule


// draws a number in hex at (xorg, yorg). Three characters are drawn, a sign and two
// hex digits. If is_signed==1, the number is treated as 2's complement.
module drawnumber(clock, reset, start, busy, number, is_signed, xorg, yorg, x, y, pixel,
input clock, reset;
input start;
output busy;
input [7:0] number;
input is_signed;
input [9:0] xorg;
input [8:0] yorg;
output [9:0] x;
output [8:0] y;
output pixel;
output wr;

reg[3:0] state;
parameter WAIT = 15;
parameter SIGN1 = 1;
parameter SIGN2 = 2;
parameter SIGN3 = 3;
parameter SIGN4 = 4;
parameter DIG11 = 5;
parameter DIG12 = 6;
parameter DIG13 = 7;
parameter DIG14 = 8;
parameter DIG21 = 9;
parameter DIG22 = 10;
parameter DIG23 = 11;
parameter DIG24 = 12;

reg char_start;
wire char_busy;
reg [9:0] char_xorg;
reg [8:0] char_yorg;
reg [7:0] char;
reg busy;


wire [7:0] mag;
wire sign;
assign mag = is_signed ? ({8{number[7]}} ^ number) + number[7] : number;
assign sign = is_signed ? number[7] : 1;

always @(posedge clock)
begin
if (reset)
```

```
state <= WAIT;
else
begin
// defaults
busy <= 1;
char_start <= 0;

case (state)
WAIT:
begin
busy <= 0;
if (start) state <= SIGN1;
end

// display sign
SIGN1:
begin
char_xorg <= xorg;
char_yorg <= yorg;
if (sign)
char <= 8'h10;
else
char <= 8'h11;
char_start <= 1;
state <= state + 1;
end
SIGN2:
state <= state + 1;
SIGN3:
state <= state + 1;
SIGN4:
if (!char_busy) state <= DIG11;

// display 1st digit
DIG11:
begin
char_xorg <= xorg + 6;
char_yorg <= yorg;
char <= {4'h0, mag[7:4]};
char_start <= 1;
state <= state + 1;
end
```

```
DIG12:
state <= state + 1;
DIG13:
state <= state + 1;
DIG14:
if (!char_busy) state <= DIG21;

// display 2nd digit
DIG21:
begin
char_xorg <= xorg + 12;
char_yorg <= yorg;
char <= {4'h0, mag[3:0]};
char_start <= 1;
state <= state + 1;
end
DIG22:
state <= state + 1;
DIG23:
state <= state + 1;
DIG24:
if (!char_busy) state <= WAIT;

default:
state <= WAIT;
endcase
end
end

drawchar drawchar1(clock, reset, char_start, char_busy, char, char_xorg, char_yorg,
x, y, pixel, wr);
endmodule



// outputs a sequence of (x, y, pixel) corresponding to char at (xorg, yorg)
// when start goes high
module drawchar(clock, reset, start, busy, char, xorg, yorg, x, y, pixel, wr);
input clock, reset;
input start;
output busy;
input [7:0] char;
input [9:0] xorg;
input [8:0] yorg;
output [9:0] x;
output [8:0] y;
output pixel;
output wr;
wire [34:0]bitmap;
wire bit;
wire [3:0] xoff;
wire [2:0] yoff;

// look up character
charrom rom1(char, clock, bitmap);

// serialize bitmap, generate signals for framebuffer
charser serial1(clock, reset, start, busy, bitmap, xoff, yoff, pixel, wr);

// offset to (xorg, yorg)
assign x = xorg + xoff;
assign y = yorg + yoff;
endmodule





// converts the character data stored in the 35 bit wide (5x7) bitmap
// to a serial (x, y, serial) sequence when start goes high. Pulses wr
// when data is valid.
module charser(clock, reset, start, busy, bitmap, xoff, yoff, serial, wr);
input clock, reset;
input start;
input [34:0] bitmap;
output [3:0] xoff;
output [2:0] yoff;
output serial;
output busy;
output wr;

reg serial;
reg busy;
reg wr;
reg [7:0]state;
parameter WAIT = 127;

reg [3:0] xoff;
reg [2:0] yoff;

always @(posedge clock)
begin
if (reset)
begin
xoff<=0;
yoff<=0;
state <= WAIT;
end
else
begin
busy <= 1;
wr <= 0;
case (state)
// idle state
WAIT:
begin
busy <= 0;
serial <= 0;
if (start) state <= 0;
end
// row 0:
0:
begin
serial <= bitmap[34];
xoff <= 0;
yoff <= 0;
state <= state + 1;
end
1:
begin
wr <= 1;
state <= state + 1;
end
2:
begin
```

```
serial <= bitmap[33];
xoff <= 1;
yoff <= 0;
state <= state + 1;
end
3:
begin
wr <= 1;
state <= state + 1;
end
4:
begin
serial <= bitmap[32];
xoff <= 2;
yoff <= 0;
state <= state + 1;
end
5:
begin
wr <= 1;
state <= state + 1;
end
6:
begin
serial <= bitmap[31];
xoff <= 3;
yoff <= 0;
state <= state + 1;
end
7:
begin
wr <= 1;
state <= state + 1;
end
8:
begin
serial <= bitmap[30];
xoff <= 4;
yoff <= 0;
state <= state + 1;
end
9:
```

```
begin
wr <= 1;
state <= state + 1;
end
10:
begin
serial <= bitmap[29];
xoff <= 0;
yoff <= 1;
state <= state + 1;
end
11:
begin
wr <= 1;
state <= state + 1;
end
// row 1:
12:
begin
serial <= bitmap[28];
xoff <= 1;
yoff <= 1;
state <= state + 1;
end
13:
begin
wr <= 1;
state <= state + 1;
end
14:
begin
serial <= bitmap[27];
xoff <= 2;
yoff <= 1;
state <= state + 1;
end
15:
begin
wr <= 1;
state <= state + 1;
end
16:
```

```
begin
serial <= bitmap[26];
xoff <= 3;
yoff <= 1;
state <= state + 1;
end
17:
begin
wr <= 1;
state <= state + 1;
end
18:
begin
serial <= bitmap[25];
xoff <= 4;
yoff <= 1;
state <= state + 1;
end
19:
begin
wr <= 1;
state <= state + 1;
end
// row 2
20:
begin
serial <= bitmap[24];
xoff <= 0;
yoff <= 2;
state <= state + 1;
end
21:
begin
wr <= 1;
state <= state + 1;
end
22:
begin
serial <= bitmap[23];
xoff <= 1;
yoff <= 2;
state <= state + 1;
```

```
end
23:
begin
wr <= 1;
state <= state + 1;
end
24:
begin
serial <= bitmap[22];
xoff <= 2;
yoff <= 2;
state <= state + 1;
end
25:
begin
wr <= 1;
state <= state + 1;
end
26:
begin
serial <= bitmap[21];
xoff <= 3;
yoff <= 2;
state <= state + 1;
end
27:
begin
wr <= 1;
state <= state + 1;
end
28:
begin
serial <= bitmap[20];
xoff <= 4;
yoff <= 2;
state <= state + 1;
end
29:
begin
wr <= 1;
state <= state + 1;
end
```

```
// row 3:
30:
begin
serial <= bitmap[19];
xoff <= 0;
yoff <= 3;
state <= state + 1;
end
31:
begin
wr <= 1;
state <= state + 1;
end
32:
begin
serial <= bitmap[18];
xoff <= 1;
yoff <= 3;
state <= state + 1;
end
33:
begin
wr <= 1;
state <= state + 1;
end
34:
begin
serial <= bitmap[17];
xoff <= 2;
yoff <= 3;
state <= state + 1;
end
35:
begin
wr <= 1;
state <= state + 1;
end
36:
begin
serial <= bitmap[16];
xoff <= 3;
yoff <= 3;
```

```
state <= state + 1;
end
37:
begin
wr <= 1;
state <= state + 1;
end
38:
begin
serial <= bitmap[15];
xoff <= 4;
yoff <= 3;
state <= state + 1;
end
39:
begin
wr <= 1;
state <= state + 1;
end
// row 4:
40:
begin
serial <= bitmap[14];
xoff <= 0;
yoff <= 4;
state <= state + 1;
end
41:
begin
wr <= 1;
state <= state + 2;
end
43:
begin
serial <= bitmap[13];
xoff <= 1;
yoff <= 4;
state <= state + 1;
end
44:
begin
wr <= 1;
```

```
state <= state + 1;
end
45:
begin
serial <= bitmap[12];
xoff <= 2;
yoff <= 4;
state <= state + 1;
end
46:
begin
wr <= 1;
state <= state + 1;
end
47:
begin
serial <= bitmap[11];
xoff <= 3;
yoff <= 4;
state <= state + 1;
end
48:
begin
wr <= 1;
state <= state + 1;
end
49:
begin
serial <= bitmap[10];
xoff <= 4;
yoff <= 4;
state <= state + 1;
end
50:
begin
wr <= 1;
state <= state + 1;
end
// row 5:
51:
begin
serial <= bitmap[9];
```

```
xoff <= 0;
yoff <= 5;
state <= state + 1;
end
52:
begin
wr <= 1;
state <= state + 1;
end
53:
begin
serial <= bitmap[8];
xoff <= 1;
yoff <= 5;
state <= state + 1;
end
54:
begin
wr <= 1;
state <= state + 1;
end
55:
begin
serial <= bitmap[7];
xoff <= 2;
yoff <= 5;
state <= state + 1;
end
56:
begin
wr <= 1;
state <= state + 1;
end
57:
begin
serial <= bitmap[6];
xoff <= 3;
yoff <= 5;
state <= state + 1;
end
58:
begin
```

```
wr <= 1;
state <= state + 1;
end
59:
begin
serial <= bitmap[5];
xoff <= 4;
yoff <= 5;
state <= state + 1;
end
60:
begin
wr <= 1;
state <= state + 1;
end
// row 6:
61:
begin
serial <= bitmap[4];
xoff <= 0;
yoff <= 6;
state <= state + 1;
end
62:
begin
wr <= 1;
state <= state + 1;
end
63:
begin
serial <= bitmap[3];
xoff <= 1;
yoff <= 6;
state <= state + 1;
end
64:
begin
wr <= 1;
state <= state + 1;
end
65:
begin
```

```
serial <= bitmap[2];
xoff <= 2;
yoff <= 6;
state <= state + 1;
end
66:
begin
wr <= 1;
state <= state + 1;
end
67:
begin
serial <= bitmap[1];
xoff <= 3;
yoff <= 6;
state <= state + 1;
end
68:
begin
wr <= 1;
state <= state + 1;
end
69:
begin
serial <= bitmap[0];
xoff <= 4;
yoff <= 6;
state <= WAIT;
end
70:
begin
wr <= 1;
state <= WAIT;
end
endcase
end
end
endmodule
```

```
//////////////////////////////////////////////////////////////////////////////
//
// File:   ps2_kbd.v
// Date:   24-Oct-05
// Author: C. Terman / I. Chuang
//
// PS2 keyboard input for 6.111 labkit
//
// INPUTS:
//
//   clock_27mhz   - master clock
//   reset         - active high
//   clock         - ps2 interface clock
//   data          - ps2 interface data
//
// OUTPUTS:
//
//   ascii         - 8 bit ascii code for current character
//   ascii_ready   - one clock cycle pulse indicating new char received
//////////////////////////////////////////////////////////////////////////////

module ps2_ascii_input(clock_27mhz, reset, clock, data, ascii, ascii_ready);

   // module to generate ascii code for keyboard input
   // this is module works synchronously with the system clock

   input clock_27mhz;
   input reset;   // Active high asynchronous reset
   input clock;   // PS/2 clock
   input data;    // PS/2 data
   output [7:0] ascii;   // ascii code (1 character)
   output ascii_ready;  // ascii ready (one clock_27mhz cycle active high)

   reg [7:0]   ascii_val; // internal combinatorial ascii decoded value
   reg [7:0]   lastkey; // last keycode
   reg [7:0]   curkey; // current keycode
   reg [7:0]   ascii; // ascii output (latched & synchronous)
   reg        ascii_ready; // synchronous one-cycle ready flag

   // get keycodes

   wire        fifo_rd; // keyboard read request
   wire [7:0]  fifo_data; // keyboard data
   wire        fifo_empty; // flag: no keyboard data
   wire        fifo_overflow; // keyboard data overflow

   key_ps2 myps2(reset, clock_27mhz, clock, data, fifo_rd, fifo_data,
     fifo_empty,fifo_overflow);

   assign      fifo_rd = ~fifo_empty; // continous read
   reg         key_ready;

   always @(posedge clock_27mhz)
     begin

// get key if ready

curkey <= ~fifo_empty ? fifo_data : curkey;
lastkey <= ~fifo_empty ? curkey : lastkey;
key_ready  <= ~fifo_empty;

// raise ascii_ready for last key which was read

ascii_ready <= key_ready & ~(curkey[7]|lastkey[7]);
ascii <=  (key_ready & ~(curkey[7]|lastkey[7])) ? ascii_val : ascii;

     end


   always @(curkey) begin //convert PS/2 keyboard make code ==> ascii code
     case (curkey)
       8'h1C: ascii_val = 8'h41; //A
       8'h32: ascii_val = 8'h42; //B
       8'h21: ascii_val = 8'h43; //C
       8'h23: ascii_val = 8'h44; //D
       8'h24: ascii_val = 8'h45; //E
       8'h2B: ascii_val = 8'h46; //F
       8'h34: ascii_val = 8'h47; //G
       8'h33: ascii_val = 8'h48; //H
       8'h43: ascii_val = 8'h49; //I
       8'h3B: ascii_val = 8'h4A; //J
       8'h42: ascii_val = 8'h4B; //K
       8'h4B: ascii_val = 8'h4C; //L
       8'h3A: ascii_val = 8'h4D; //M
```

```
8’h31: ascii_val = 8’h4E; //N
8’h44: ascii_val = 8’h4F; //O
8’h4D: ascii_val = 8’h50; //P
8’h15: ascii_val = 8’h51; //Q
8’h2D: ascii_val = 8’h52; //R
8’h1B: ascii_val = 8’h53; //S
8’h2C: ascii_val = 8’h54; //T
8’h3C: ascii_val = 8’h55; //U
8’h2A: ascii_val = 8’h56; //V
8’h1D: ascii_val = 8’h57; //W
8’h22: ascii_val = 8’h58; //X
8’h35: ascii_val = 8’h59; //Y
8’h1A: ascii_val = 8’h5A; //Z

8’h45: ascii_val = 8’h30; //0
8’h16: ascii_val = 8’h31; //1
8’h1E: ascii_val = 8’h32; //2
8’h26: ascii_val = 8’h33; //3
8’h25: ascii_val = 8’h34; //4
8’h2E: ascii_val = 8’h35; //5
8’h36: ascii_val = 8’h36; //6
8’h3D: ascii_val = 8’h37; //7
8’h3E: ascii_val = 8’h38; //8
8’h46: ascii_val = 8’h39; //9

8’h0E: ascii_val = 8’h60; // ‘
8’h4E: ascii_val = 8’h2D; // -
8’h55: ascii_val = 8’h3D; // =
8’h5C: ascii_val = 8’h5C; // backslash
8’h29: ascii_val = 8’h20; // (space)
8’h54: ascii_val = 8’h5B; // [
8’h5B: ascii_val = 8’h5D; // ]
8’h4C: ascii_val = 8’h3B; // ;
8’h52: ascii_val = 8’h27; // ’
8’h41: ascii_val = 8’h2C; // ,
8’h49: ascii_val = 8’h2E; // .
8’h4A: ascii_val = 8’h2F; // /

8’h5A: ascii_val = 8’h0D; // enter (CR)
8’h66: ascii_val = 8’h08; // backspace

//  8’hF0: ascii_val = 8’hF0; // BREAK CODE
```

```
       default: ascii_val = 8'h23; // #
     endcase
   end
endmodule // ps2toascii



///////////////////////////////////////////////////////////////////////////////
// new synchronous ps2 keyboard driver, with built-in fifo, from Chris Terman

module key_ps2(reset, clock_27mhz, ps2c, ps2d, fifo_rd, fifo_data,
   fifo_empty,fifo_overflow);

   input clock_27mhz,reset;
   input ps2c; // ps2 clock
   input ps2d; // ps2 data
   input fifo_rd; // fifo read request (active high)
   output [7:0] fifo_data; // fifo data output
   output  fifo_empty; // fifo empty (active high)
   output  fifo_overflow; // fifo overflow - too much kbd input

  reg [3:0] count;      // count incoming data bits
  reg [9:0] shift;      // accumulate incoming data bits

  reg [7:0] fifo[7:0];   // 8 element data fifo
  reg fifo_overflow;
  reg [2:0] wptr,rptr;   // fifo write and read pointers

  wire [2:0] wptr_inc = wptr + 1;

  assign fifo_empty = (wptr == rptr);
  assign fifo_data = fifo[rptr];

  // synchronize PS2 clock to local clock and look for falling edge
  reg [2:0] ps2c_sync;
  always @ (posedge clock_27mhz) ps2c_sync <= {ps2c_sync[1:0],ps2c};
  wire sample = ps2c_sync[2] & ~ps2c_sync[1];

  always @ (posedge clock_27mhz) begin
    if (reset) begin
      count <= 0;
      wptr <= 0;
      rptr <= 0;
      fifo_overflow <= 0;
    end
    else if (sample) begin
           // order of arrival: 0,8 bits of data (LSB first),odd parity,1
           if (count==10) begin
              // just received what should be the stop bit
              if (shift[0]==0 && ps2d==1 && (^shift[9:1])==1) begin
 fifo[wptr] <= shift[8:1];
 wptr <= wptr_inc;
 fifo_overflow <= fifo_overflow | (wptr_inc == rptr);
              end
              count <= 0;
   end else begin
              shift <= {ps2d,shift[9:1]};
              count <= count + 1;
   end
         end
    // bump read pointer if we're done with current value.
    // Read also resets the overflow indicator
    if (fifo_rd && !fifo_empty) begin
      rptr <= rptr + 1;
      fifo_overflow <= 0;
    end
  end

endmodule


`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:59:07 04/29/2007
// Design Name:
// Module Name:    textdisplay
// Project Name:
// Target Devices:
// Tool versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

/*
// outputs characters to the VGA module
module textdisplay(clock, reset, start, busy, x, y, color, char_start, char_busy, pixel,
rotation, xscale, yscale, xoffset, yoffset, xbias, ybias, speed);
input clock, reset;
input start;
output busy;
output [9:0] x;
output [8:0] y;
output [23:0] color;
input signed [7:0] rotation, xscale, yscale, xoffset, yoffset, xbias, ybias, speed;
output char_start, char_busy, pixel;

reg[3:0] state;
parameter WAIT = 0;

reg char_start;
reg [9:0]xorg;
reg [8:0]yorg;
reg [7:0]char;
wire char_busy;
reg busy;

always @(posedge clock)
begin
if(reset)
state <= WAIT;
else
begin
// assign defaults
char_start <= 0;
busy <= 1;
```

```
case (state)
WAIT:
begin
busy <= 0;
if (start)
state <= 1;
end
1:
begin
xorg <= 10;
yorg <= 100;
char <= 1;
char_start <= 1;
state <= state + 1;
end
2: state <= state + 1;
3: state <= state + 1;
4:
begin
if (!char_busy)
state <= state + 1;
end
5:
begin
xorg <= 100;
yorg <= 10;
char <= 0;
char_start <= 1;
state <= state + 1;
end
6: state <= state + 1;
7: state <= state + 1;
8:
begin
if (!char_busy)
state <= WAIT;
end

endcase
end
end
```

```
wire pixel;
drawchar char1(clock, reset, char_start, char_busy, char, xorg, yorg, x, y, pixel);
assign color = pixel ? 24'hffffff : 24'h000000;

endmodule
*/


// outputs a sequence of (x, y, pixel) corresponding to char at (xorg, yorg)
// when start goes high
module drawchar(clock, reset, start, busy, char, xorg, yorg, x, y, pixel, wr);
input clock, reset;
input start;
output busy;
input [7:0] char;
input [9:0] xorg;
input [9:0] yorg;
output [9:0] x;
output [9:0] y;
output pixel;
output wr;

wire [34:0]bitmap;
charrom rom1(char, clock, bitmap);

wire bit;
wire [3:0] xoff;
wire [2:0] yoff;
charser serial1(clock, reset, start, busy, bitmap, xoff, yoff, pixel, wr);

assign x = xorg + xoff;
assign y = yorg + yoff;
endmodule

// converts the character data stored in the 35 bit wide (5x7) bitmap
// to a serial (xoff, yoff, pixel) sequence when start goes high
module charser(clock, reset, start, busy, bitmap, xoff, yoff, serial, wr);
input clock, reset;
input start;
input [34:0] bitmap;
output [3:0] xoff;
output [2:0] yoff;
output serial;
output busy;
output wr;

reg serial;
reg busy;
reg wr;

reg [7:0]state;
parameter WAIT = 127;
// [0,35] used to clock out bitmap

reg [3:0] xoff;
reg [2:0] yoff;

always @(posedge clock)
begin
if (reset)
begin
xoff<=0;
yoff<=0;
state <= WAIT;
end
else
begin
busy <= 1;
wr <= 0;
case (state)
// idle state
WAIT:
begin
busy <= 0;
serial <= 0;
if (start) state <= 0;
end
// row 0:
0:
begin
serial <= bitmap[34];
xoff <= 0;
yoff <= 0;
state <= state + 1;
```

```
wr <= 1;
end
1:
begin
wr <= 0;
state <= state + 1;
end
2:
begin
wr <= 1;
serial <= bitmap[33];
xoff <= 1;
yoff <= 0;
state <= state + 1;
end
3:
begin
wr <= 0;
state <= state + 1;
end
4:
begin
wr <= 1;
serial <= bitmap[32];
xoff <= 2;
yoff <= 0;
state <= state + 1;
end
5:
begin
wr <= 0;
state <= state + 1;
end
6:
begin
wr <= 1;
serial <= bitmap[31];
xoff <= 3;
yoff <= 0;
state <= state + 1;
end
7:
```

```
begin
wr <= 0;
state <= state + 1;
end
8:
begin
wr <= 1;
serial <= bitmap[30];
xoff <= 4;
yoff <= 0;
state <= state + 1;
end
9:
begin
wr <= 0;
state <= state + 1;
end
10:
begin
wr <= 1;
serial <= bitmap[29];
xoff <= 0;
yoff <= 1;
state <= state + 1;
end
11:
begin
wr <= 0;
state <= state + 1;
end
// row 1:
12:
begin
wr <= 1;
serial <= bitmap[28];
xoff <= 1;
yoff <= 1;
state <= state + 1;
end
13:
begin
wr <= 0;
```

```
state <= state + 1;
end
14:
begin
wr <= 1;
serial <= bitmap[27];
xoff <= 2;
yoff <= 1;
state <= state + 1;
end
15:
begin
wr <= 0;
state <= state + 1;
end
16:
begin
wr <= 1;
serial <= bitmap[26];
xoff <= 3;
yoff <= 1;
state <= state + 1;
end
17:
begin
wr <= 0;
state <= state + 1;
end
18:
begin
wr <= 1;
serial <= bitmap[25];
xoff <= 4;
yoff <= 1;
state <= state + 1;
end
19:
begin
wr <= 0;
state <= state + 1;
end
// row 2
```

```
20:
begin
wr <= 1;
serial <= bitmap[24];
xoff <= 0;
yoff <= 2;
state <= state + 1;
end
21:
begin
wr <= 0;
state <= state + 1;
end
22:
begin
wr <= 1;
serial <= bitmap[23];
xoff <= 1;
yoff <= 2;
state <= state + 1;
end
23:
begin
wr <= 0;
state <= state + 1;
end
24:
begin
wr <= 1;
serial <= bitmap[22];
xoff <= 2;
yoff <= 2;
state <= state + 1;
end
25:
begin
wr <= 0;
state <= state + 1;
end
26:
begin
wr <= 1;
```

```
serial <= bitmap[21];
xoff <= 3;
yoff <= 2;
state <= state + 1;
end
27:
begin
wr <= 0;
state <= state + 1;
end
28:
begin
wr <= 1;
serial <= bitmap[20];
xoff <= 4;
yoff <= 2;
state <= state + 1;
end
29:
begin
wr <= 0;
state <= state + 1;
end
// row 3:
30:
begin
wr <= 1;
serial <= bitmap[19];
xoff <= 0;
yoff <= 3;
state <= state + 1;
end
31:
begin
wr <= 0;
state <= state + 1;
end
32:
begin
wr <= 1;
serial <= bitmap[18];
xoff <= 1;
```

```
yoff <= 3;
state <= state + 1;
end
33:
begin
wr <= 0;
state <= state + 1;
end
34:
begin
wr <= 1;
serial <= bitmap[17];
xoff <= 2;
yoff <= 3;
state <= state + 1;
end
35:
begin
wr <= 0;
state <= state + 1;
end
36:
begin
wr <= 1;
serial <= bitmap[16];
xoff <= 3;
yoff <= 3;
state <= state + 1;
end
37:
begin
wr <= 0;
state <= state + 1;
end
38:
begin
wr <= 1;
serial <= bitmap[15];
xoff <= 4;
yoff <= 3;
state <= state + 1;
end
```

```
39:
begin
wr <= 0;
state <= state + 1;
end
// row 4:
40:
begin
wr <= 1;
serial <= bitmap[14];
xoff <= 0;
yoff <= 4;
state <= state + 1;
end
41:
begin
wr <= 0;
state <= state + 2;
end
43:
begin
wr <= 1;
serial <= bitmap[13];
xoff <= 1;
yoff <= 4;
state <= state + 1;
end
44:
begin
wr <= 0;
state <= state + 1;
end
45:
begin
wr <= 1;
serial <= bitmap[12];
xoff <= 2;
yoff <= 4;
state <= state + 1;
end
46:
begin
```

```
wr <= 0;
state <= state + 1;
end
47:
begin
wr <= 1;
serial <= bitmap[11];
xoff <= 3;
yoff <= 4;
state <= state + 1;
end
48:
begin
wr <= 0;
state <= state + 1;
end
49:
begin
wr <= 1;
serial <= bitmap[10];
xoff <= 4;
yoff <= 4;
state <= state + 1;
end
50:
begin
wr <= 0;
state <= state + 1;
end
// row 5:
51:
begin
wr <= 1;
serial <= bitmap[9];
xoff <= 0;
yoff <= 5;
state <= state + 1;
end
52:
begin
wr <= 0;
state <= state + 1;
```

```
end
53:
begin
wr <= 1;
serial <= bitmap[8];
xoff <= 1;
yoff <= 5;
state <= state + 1;
end
54:
begin
wr <= 0;
state <= state + 1;
end
55:
begin
wr <= 1;
serial <= bitmap[7];
xoff <= 2;
yoff <= 5;
state <= state + 1;
end
56:
begin
wr <= 0;
state <= state + 1;
end
57:
begin
wr <= 1;
serial <= bitmap[6];
xoff <= 3;
yoff <= 5;
state <= state + 1;
end
58:
begin
wr <= 0;
state <= state + 1;
end
59:
begin
```

```
wr <= 1;
serial <= bitmap[5];
xoff <= 4;
yoff <= 5;
state <= state + 1;
end
60:
begin
wr <= 0;
state <= state + 1;
end
// row 6:
61:
begin
wr <= 1;
serial <= bitmap[4];
xoff <= 0;
yoff <= 6;
state <= state + 1;
end
62:
begin
wr <= 0;
state <= state + 1;
end
63:
begin
wr <= 1;
serial <= bitmap[3];
xoff <= 1;
yoff <= 6;
state <= state + 1;
end
64:
begin
wr <= 0;
state <= state + 1;
end
65:
begin
wr <= 1;
serial <= bitmap[2];
```

```
xoff <= 2;
yoff <= 6;
state <= state + 1;
end
66:
begin
wr <= 0;
state <= state + 1;
end
67:
begin
wr <= 1;
serial <= bitmap[1];
xoff <= 3;
yoff <= 6;
state <= state + 1;
end
68:
begin
wr <= 0;
state <= state + 1;
end
69:
begin
wr <= 1;
serial <= bitmap[0];
xoff <= 4;
yoff <= 6;
state <= WAIT;
end
70:
begin
wr <= 0;
state <= WAIT;
end
endcase
end
end
endmodule
```

## Table of Contents

## Double buffer

```
// This module implements the double buffer for displaying on the screen
// By: Huy Nguyen
module ram_buffer(
        reset,
        clock,
        switch_buffer,
        write_pixel_count,
        write_line_count,
        write_color_and_z,
        read_pixel_count,
        read_line_count,
        rgb,
        start,
        busy, state, write_addr);
        output [2:0] state;
        output [18:0] write_addr;

        //parameter DEFAULT_COLOR = 24'h00FF0F; // background color
        parameter SCREEN_WIDTH   = 640;
        parameter SCREEN_HEIGHT  = 480;


        input reset;
        input clock;
        input switch_buffer;
        input [9:0] write_pixel_count;
        input [9:0] write_line_count;
        input write_color_and_z;
        input [9:0] read_pixel_count;
        input [9:0] read_line_count;
```

```
output  [23:0]  rgb;
input  start;
output  busy;

//  state  of  write  buffer
reg  [2:0]  state,  next_state;
reg  we,  next_we;
reg  busy,  next_busy;

wire  we0,  we1;

localparam  CLEAR  =  0;
localparam  CLEAR2  =  1;
localparam  WRITE  =  2;
localparam  IDLE  =  3;

reg  start0,  start1,  start2;
wire  color;

reg  [18:0]  write_addr,  next_write_addr;

reg  w_idx;

wire  dout0,  dout1;

reg  write_data,  next_write_data;

wire  [18:0]  read_addr0,  read_addr1;

buffer_ram  buffer_ram1(
.clka(clock),
.dina(write_data),
.addra(write_addr),
.wea(we1),
.clkb(clock),
.addrb(read_addr1),
.doutb(dout1));

buffer_ram  buffer_ram0(
.clka(clock),
.dina(write_data),
.addra(write_addr),
.wea(we0),
.clkb(clock),
.addrb(read_addr0),
.doutb(dout0));
```

```
	assign read_addr0 = !w_idx ? 19'bZ : read_line_count * SCREEN_WIDTH +
read_pixel_count;
	assign read_addr1 = w_idx ? 19'bZ : read_line_count * SCREEN_WIDTH +
read_pixel_count;

	assign we0 = !w_idx ? we : 0;
	assign we1 = w_idx ? we : 0;
	assign color = w_idx ? dout0 : dout1;
	//assign color = dout0;
	//assign we0 = we;
	//assign read_addr0 = read_line_count * SCREEN_WIDTH + read_pixel_count;

	assign rgb = (color) ? 24'h00FF00 : 24'h000000;

	// write buffer logic
	always @ (posedge clock)
	begin
		// change state
		if (reset)
		begin
			state <= IDLE;
			busy <= 0;
			we <= 0;
			write_addr <= 0;
			w_idx <= 0;
		end
		else begin
			state <= next_state;
			we <= next_we;
			write_addr <= next_write_addr;
			w_idx <= switch_buffer ? !w_idx : w_idx;
			busy <= next_busy;
			write_data <= next_write_data;
		end
	end

	always @ (start or write_addr or state or we or write_line_count or
write_pixel_count or busy or switch_buffer or write_data
	or write_color_and_z)
	begin
		//default values
		next_write_addr = write_addr;
		next_state = state;
		next_we = we;
		next_busy = busy;
		next_write_data = write_data;
```

```
        case  (state)
            IDLE:begin
                if  (switch_buffer)  begin
                    next_busy  =  1;
                    next_state  =  CLEAR;
                    next_we  =  1;
                    next_write_addr  =  0;
                    next_write_data  =  1'b0;
                end
                else  if  (start)
                begin
                    next_state  =  WRITE;
                    next_we  =  1;
                    next_busy  =  1;
                    next_write_addr  =  write_line_count  *  SCREEN_WIDTH
+  write_pixel_count;
                    next_write_data  =  write_color_and_z;
                end
            end
            WRITE:  begin
                if  (switch_buffer)  begin
                    next_busy  =  1;
                    next_state  =  CLEAR;
                    next_we  =  1;
                    next_write_addr  =  0;
                    next_write_data  =  1'b0;
                end
                else  begin
                    next_state  =  IDLE;
                    next_we  =  0;
                    next_busy  =  0;
                end
            end
            CLEAR:begin
                    next_state  =  CLEAR2;
                    next_we  =  0;
                    next_busy  =  1;
                end
            CLEAR2:begin
                if  (write_addr  <  SCREEN_WIDTH  *  SCREEN_HEIGHT)
                begin
                    next_state  =  CLEAR;
                    next_we  =  1;
                    next_write_addr  =  write_addr  +  1;
                end
                else  begin
                    next_state  =  IDLE;
```

```
                                        next_busy = 0;
                                        next_we = 0;
                                  end
                            end
                      endcase
                end
endmodule
```

## Render

```
// By: Huy Nguyen
module render(reset, clock, objcount, busy, ram_data, ram_addr, x1_vga, y1_vga,
x2_vga, y2_vga, color_vga, z_vga, x1_laser, y1_laser, x2_laser, y2_laser, next_laser,
start_laser, start_vga, busy_vga, shape_out, switch_buffer,blank,state,shape_laser,
      drawchar_start, drawchar_busy, buffer_write_mux, ix2_laser, iy2_laser, radius,
buffer_busy);
      output signed [31:0] ix2_laser, iy2_laser, radius;

      input buffer_busy;

      parameter PRECISION = 12;
      input reset;
      input clock;
      input [10:0] objcount;
      output busy;
      input [PRECISION * 5 +24 :0] ram_data;//12x1 + 12y1 + 12x2 + 12y2
+ 12z + 24color + 1shape
      output [10:0] ram_addr;
      output [PRECISION - 1:0] x1_vga;
      output [PRECISION - 1:0] y1_vga;
      output [PRECISION - 1:0] x2_vga;
      output [PRECISION - 1:0] y2_vga;
      output [23:0] color_vga;
      output [PRECISION - 1:0] z_vga;
      output [7:0] x1_laser;
      output [7:0] y1_laser;
      output [7:0] x2_laser;
      output [7:0] y2_laser;
      input next_laser;
      output start_laser;
      output start_vga;
      input busy_vga;
      output shape_out;
      output switch_buffer;
      input blank;
      output [4:0] state;
      output [3:0] shape_laser;
```

```
        output   drawchar_start;
        input   drawchar_busy;
        output   buffer_write_mux;

        localparam   IDLE   =   0;
        localparam   READ1   =   1;
        localparam   READ2   =   2;
        localparam   READ3   =   3;
        localparam   READ4   =   4;
        localparam   SWITCH_BUFFER1   =   5;
        localparam   READ_LASER1   =   6;
        localparam   READ_LASER2   =   7;
        localparam   FETCH_LASER   =   8;
        localparam   WRITE_PARAMS   =   9;
        localparam   WRITE_PARAMS2   =   10;
        localparam   WRITE_PARAMS3   =   11;
        localparam   SWITCH_BUFFER2   =   12;
        localparam   BACK_TO_NORMAL   =   13;
        localparam   READ_LASER0   =   14;
        localparam   SWITCH_BUFFER3   =   15;
        localparam   SWITCH_BUFFER4   =   16;

        reg   [4:0]   state,   new_state,   old_state,   new_old_state;
        reg   [PRECISION   -   1:0]   x1_vga,   y1_vga,   x2_vga,   y2_vga,   new_x1_vga,
new_y1_vga,   new_x2_vga,   new_y2_vga;
        wire   [23:0]   color;
        wire   [11:0]   z;

        reg   [10:0]   read_addr,   new_read_addr,   old_read_addr,   new_old_read_addr;
        reg   [10:0]   counter,   new_counter,   counter_laser,   new_counter_laser;
        reg   busy,   new_busy,   old_busy,   new_old_busy;
        reg   start_vga,   new_start_vga,   start_laser,   new_start_laser;
        reg   switch_buffer,   new_switch_buffer;
        reg   signed   [31:0]   ix1_laser,   iy1_laser,   ix2_laser,   iy2_laser,   new_ix1_laser,
new_iy1_laser,   new_ix2_laser,   new_iy2_laser,   xc_laser,   yc_laser,   new_xc_laser,
new_yc_laser,   radius,   new_radius;
        reg   [3:0]   shape_laser,   new_shape_laser;
        reg   shape_out,   new_shape;

        reg   drawchar_start,   new_drawchar_start;

        assign   ram_addr   =   read_addr;
        //assign   x1_vga   =   ram_data[84:73];
        //assign   y1_vga   =   ram_data[72:61];
        //assign   x2_vga   =   ram_data[60:49];
        //assign   y2_vga   =   ram_data[48:37];
        assign   z_vga   =   ram_data[36:25];
        assign color_vga = ram_data[24:1];
        //assign shape_out = ram_data[0];

        assign x1_laser = ix1_laser >> 16;
        assign x2_laser = ix2_laser >> 16;
        assign y1_laser = iy1_laser >> 16;
        assign y2_laser = iy2_laser >> 16;

        assign buffer_write_mux = (state != WRITE_PARAMS && state !=
WRITE_PARAMS2 && state != WRITE_PARAMS3);

        always @ (posedge clock)
        begin
                if (reset)
                begin
                        state <= IDLE;
                        start_vga <= 0;
                        busy <= 0;
                        switch_buffer <= 0;
                        read_addr <= 0;
                        old_state <= 0;
                        shape_laser <= 0;
                        counter_laser <= 0;
                        counter <= 0;
                        old_busy <= 0;
                        start_laser <= 0;
                        x1_vga <= 0;
                        y1_vga <= 0;
                        x2_vga <= 0;
                        y2_vga <= 0;

                        xc_laser <= 0;
                        yc_laser <= 0;
                        radius <= 0;

                        drawchar_start <= 0;
                end
                else begin
                        state <= new_state;
                        busy <= new_busy;
                        counter <= new_counter;
                        read_addr <= new_read_addr;
                        start_vga <= new_start_vga;
                        switch_buffer <= new_switch_buffer;
                        x1_vga <= new_x1_vga;
                        y1_vga <= new_y1_vga;
                        x2_vga <= new_x2_vga;
                        y2_vga  <=  new_y2_vga;
                        shape_out  <=  new_shape;

                        old_state  <=  new_old_state;
                        ix1_laser  <=  new_ix1_laser;
                        iy1_laser  <=  new_iy1_laser;
                        ix2_laser  <=  new_ix2_laser;
                        iy2_laser  <=  new_iy2_laser;
                        xc_laser  <=  new_xc_laser;
                        yc_laser  <=  new_yc_laser;
                        shape_laser  <=  new_shape_laser;
                        counter_laser  <=  new_counter_laser;
                        old_busy  <=  new_old_busy;
                        start_laser  <=  new_start_laser;
                        radius  <=  new_radius;

                        old_read_addr  <=  new_old_read_addr;

                        drawchar_start  <=  new_drawchar_start;
                end
        end

        always @ (state or read_addr or counter or busy or objcount or busy_vga
or blank or next_laser or shape_laser
        or x1_vga or y1_vga or x2_vga or y2_vga or shape_out or ix1_laser or
iy1_laser or ix2_laser or iy2_laser or xc_laser or yc_laser
        or old_state or counter_laser or ram_data or old_busy or drawchar_busy or
drawchar_start or xc_laser or yc_laser
        or radius or old_read_addr)
        begin
                // default values
                new_state = state;
                new_read_addr = read_addr;
                new_counter = counter;
                new_busy = busy;
                new_start_vga = 0;
                new_switch_buffer = 0;
                new_old_state = old_state;
                new_old_busy = old_busy;

                new_x1_vga = x1_vga;
                new_y1_vga = y1_vga;
                new_x2_vga = x2_vga;
                new_y2_vga = y2_vga;
                new_shape = shape_out;

                new_ix1_laser = ix1_laser;
                new_iy1_laser = iy1_laser;
                new_ix2_laser = ix2_laser;
                new_iy2_laser = iy2_laser;
                new_xc_laser = xc_laser;
                new_yc_laser = yc_laser;
                new_shape_laser = shape_laser;
                new_counter_laser = counter_laser;
                new_start_laser = 0;
                new_radius = radius;

                new_drawchar_start = 0;

                if (next_laser)
                begin
                        new_old_state = state;
                        new_old_busy = busy;
                        new_old_read_addr = read_addr;
                        new_busy = 1;
                        if (shape_laser == 0 || shape_laser == 8) //finished previous
shape, 0<->line, 8<->circle
                        begin
                              new_state = READ_LASER0;
                        end
                        else begin
                              new_state = FETCH_LASER;
                              new_start_laser = 1;
                              //cos(2pi/8)=181/256
                              //x_new = x_old * cos - y_old * sin
                              //y_new = x_old * sin + y_old * cos
                              new_ix1_laser = ix2_laser;
                              new_iy1_laser = iy2_laser;
                              new_ix2_laser = (ix2_laser +yc_laser > xc_laser +
iy2_laser) ?
                              xc_laser + (ix2_laser - xc_laser - iy2_laser +
yc_laser) * 181/256:
                              xc_laser - (xc_laser + iy2_laser - yc_laser -
ix2_laser) * 181/256;
                              new_iy2_laser = (ix2_laser + iy2_laser > xc_laser +
yc_laser) ?
                              yc_laser + (ix2_laser + iy2_laser - xc_laser -
yc_laser) * 181/256:
                              yc_laser - (xc_laser + yc_laser - ix2_laser -
iy2_laser) * 181/256;
                              new_shape_laser = shape_laser + 1;
                        end
                end
                else
```

```
case   (state)
        IDLE:begin
                //if   (start)
                //begin
                        new_state   =   READ1;
                        new_busy   =   1;
                        new_counter   =   0;
                //end
        end
        READ1:begin
                if   (counter   <   objcount)
                begin
                        new_read_addr   =   counter;
                        new_state   =   READ2;
                end
                else   begin
                        new_state   =   WRITE_PARAMS;
                        new_drawchar_start   =   1;
                end
        end
        READ2:begin
                new_state   =   READ3;
                new_start_vga   =   1;
                new_x1_vga   =   ram_data[84:73];
                new_y1_vga   =   ram_data[72:61];
                new_x2_vga   =   ram_data[60:49];
                new_y2_vga   =   ram_data[48:37];
                new_shape   =   ram_data[0];
        end
        READ3:begin
                new_state   =   READ4;
        end
        READ4:begin
                if   (!busy_vga)
                begin
                        new_state   =   READ1;
                        new_counter   =   counter   +   1;
                end
        end
        WRITE_PARAMS:begin
                new_state   =   WRITE_PARAMS2;
        end
        WRITE_PARAMS2:begin
                new_state   =   WRITE_PARAMS3;
        end
        WRITE_PARAMS3:begin
                if(!drawchar_busy)
                        new_state  =  SWITCH_BUFFER1;
        end
        SWITCH_BUFFER1:begin
                new_state  =  SWITCH_BUFFER2;
        end
        SWITCH_BUFFER2:begin
                if  (!blank)
                begin
                        new_switch_buffer  =  1;
                        new_busy  =  1;
                        new_state  =  SWITCH_BUFFER3;
                end
        end
        SWITCH_BUFFER3:begin
                new_state  =  SWITCH_BUFFER4;
        end
        SWITCH_BUFFER4:begin
                if  (!buffer_busy)
                begin
                        new_busy  =  0;
                        new_state  =  IDLE;
                end
        end
        READ_LASER0:begin
                new_state  =  READ_LASER1;
                new_read_addr  =  counter_laser;
                if  (counter_laser  <  objcount  -  1)
                        new_counter_laser  =  counter_laser  +  1;
                else  new_counter_laser  =  0;
        end
        READ_LASER1:begin
                new_state  =  READ_LASER2;
        end
        READ_LASER2:begin
                new_state  =  FETCH_LASER;
                new_start_laser  =  1;
                if  (ram_data[0]  ==  0)  //line
                begin
                        new_ix1_laser  =  ram_data[84:73]  <<  14;
                        new_iy1_laser  =  ram_data[72:61]  <<  14;
                        new_ix2_laser  =  ram_data[60:49]  <<  14;
                        new_iy2_laser  =  ram_data[48:37]  <<  14;
                        new_shape_laser  =  0;
                end
                else  //circle
                begin
                        new_xc_laser  =  ram_data[84:73]  <<  14;
```

```
                                        new_yc_laser   =   ram_data[72:61]   <<   14;
                                        new_radius   =   ram_data[60:49]   <<   14;
                                        new_ix1_laser   =   (ram_data[84:73]+ram_data[60:49])   <<
14;
                                        new_iy1_laser   =   ram_data[72:61]   <<   14;
                                        new_ix2_laser   =   (ram_data[84:73]   <<   14)   +
ram_data[60:49]*11585;
                                        new_iy2_laser   =   (ram_data[72:61]   <<   14)   +
ram_data[60:49]*11585;
                                        new_shape_laser   =   1;
                                end
                        end
                        FETCH_LASER:begin
                                new_state   =   BACK_TO_NORMAL;
                                new_read_addr   =   old_read_addr;
                        end
                        BACK_TO_NORMAL:begin
                                new_state   =   old_state;
                                new_busy   =   old_busy;
                        end
                endcase
        end
endmodule
```

## Rotate

```
`timescale   1ns   /   1ps
//   rotate   by   roughly   pi/10,   sin(pi/10)   ~   79/256,   cos(pi/10)   ~   243/256
module   rotate(reset,   clock,   start,   busy,   objcount,   direction,   ram_data_in,   ram_data_out,
ram_addr,   ram_we,new_x2,new_y2,   aaa);
          output   [31:0]   new_x2,   new_y2;
          output   [47:0]   aaa;

        parameter   PRECISION   =   12;

        input   reset;
        input   clock;
        input   start;
        output   busy;
        input   [10:0]   objcount;
        input   direction;
```

```
	output   [84:0]   ram_data_in;
	input   [84:0]   ram_data_out;
	output   [10:0]   ram_addr;
	output   ram_we;

	reg[84:0]   ram_data_in,   new_ram_data_in;
	reg   [10:0]   ram_addr,   new_ram_addr;
	reg   ram_we,   new_ram_we;

	wire   signed   [31:0]   x1,   y1,   x2,   y2,   new_x1,   new_y1,   new_x2,   new_y2;
	assign   x1   =   ram_data_out[84:73];
	assign   y1   =   ram_data_out[72:61];
	assign   x2   =   ram_data_out[60:49];
	assign   y2   =   ram_data_out[48:37];
	assign   new_x1   =   direction   ?   (x1   *   243   -   y1   *   79)   /   256   :   (x1   *
243   +   y1   *   79)   /   256;
	assign   new_y1   =   direction   ?   (x1   *   79   +   y1   *   243)   /   256   :   (-x1   *
79   +   y1   *   243)   /   256;
	assign   new_x2   =   direction   ?   (x2   *   243   -   y2   *   79)   /   256   :   (x2   *
243   +   y2   *   79)   /   256;
	assign   new_y2   =   direction   ?   (x2   *   79   +   y2   *   243)   /   256   :   (-x2   *
79   +   y2   *   243)   /   256;
	assign   aaa   =   {new_x1[PRECISION-1:0],   new_y1[PRECISION-1:0],
new_x2[PRECISION-1:0],   new_y2[PRECISION-1:0]};

	localparam   IDLE   =   0;
	localparam   READ1   =   1;
	localparam   READ2   =   2;
	localparam   WRITE1   =   3;
	localparam   WRITE2   =   4;
	localparam   WRITE3   =   5;
	localparam   READ0   =   6;

	reg   [2:0]   state,   new_state;
reg busy, new_busy;

always @ (posedge clock)
begin
    if (reset)
    begin
        state <= IDLE;
        busy <= 0;
        ram_data_in <= 0;
        ram_addr <= 0;
        ram_we <= 0;
    end
    else begin
        state <= new_state;
        busy <= new_busy;
        ram_data_in <= new_ram_data_in;
        ram_addr <= new_ram_addr;
        ram_we <= new_ram_we;
    end
end

always @(state or busy or ram_data_in or ram_addr or ram_we
or start or ram_data_out or new_x1 or new_x2 or new_y1 or new_y2 or
objcount)
begin
    //default values
    new_state     = state;
    new_busy = busy;
    new_ram_data_in = ram_data_in;
    new_ram_addr = ram_addr;
    new_ram_we = 0;

    case (state)
```

```
        IDLE:begin
            if (start)
            begin
                new_state = READ0;
                new_ram_addr = 0;
                new_busy = 1;
            end
        end
        READ0:begin
            new_state = READ1;
        end
        READ1:begin
            if (ram_addr < objcount)
                new_state = READ2;
            else begin
                new_state = IDLE;
                new_busy = 0;
            end
        end
        READ2:begin
            if (ram_data_out[0] == 0)
            begin
                new_ram_data_in = {new_x1[PRECISION-1:0],
new_y1[PRECISION-1:0], new_x2[PRECISION-1:0], new_y2[PRECISION-1:0],
ram_data_out[36:0]};
            end
            else
                new_ram_data_in = {new_x1[PRECISION-1:0],
new_y1[PRECISION-1:0], ram_data_out[60:0]};
            new_state = WRITE1;
            new_ram_we = 1;
        end
        WRITE1:begin
            new_state = WRITE2;
```

```
                    new_ram_we = 0;
                end
                WRITE2:begin
                    new_state = WRITE3;
                    new_ram_addr = ram_addr + 1;
                end
                WRITE3:begin
                    new_state = READ1;
                end
            endcase
        end
endmodule
```

## Add

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:43:54 05/01/2007
// Design Name:
// Module Name:    add
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
```

```
////////////////////////////////////////////////////////////////////////////////
module add(clock, reset, x1, y1, x2, y2, shape, objcount, start, busy, ram_data,
ram_addr, ram_we);

        parameter PRECISION=12;

        input clock;
        input reset;
        input [PRECISION-1:0] x1;
        input [PRECISION-1:0] y1;
        input [PRECISION-1:0] x2;
        input [PRECISION-1:0] y2;
        input shape;
        input [10:0] objcount;
        input start;
        output busy;

        output [84:0] ram_data;
        output [10:0] ram_addr;
        output ram_we;

        localparam IDLE = 0;
        localparam WRITE1 = 1;
        localparam WRITE2 = 2;
        localparam WRITE3 = 3;
        reg [1:0] state, new_state;
        reg ram_we, new_ram_we, busy, new_busy;
        reg [84:0] ram_data, new_ram_data;
        reg [10:0] ram_addr, new_ram_addr;

        always @ (posedge clock)
        begin
                if (reset)
                begin
                    state <= IDLE;
                    ram_we <= 0;
                    busy <= 0;
                end
                else
                begin
                    state <= new_state;
                    ram_we <= new_ram_we;
                    ram_data <= new_ram_data;
                    ram_addr <= new_ram_addr;
                    busy <= new_busy;
                end
        end

        always @ (state or busy or start or x1 or y1 or x2 or y2 or shape
or objcount)
        begin
                //default values
                new_state = state;
                new_ram_we = 0;
                new_busy = busy;
                case (state)
                    IDLE:begin
                        if (start)
                        begin
                            new_state = WRITE1;
                            new_ram_we = 0;
                            new_ram_addr = objcount;
                            new_ram_data = {x1,y1,x2,y2,36'b1,shape};
                            new_busy = 1;
                        end
                    end
```

```
                    WRITE1:begin
                        new_state  =  WRITE2;
                        new_ram_we  =  1;
                    end
                    WRITE2:begin
                        new_state  =  WRITE3;
                        new_busy  =  1;
                    end
                    WRITE3:begin
                        new_state  =  IDLE;
                        new_busy  =  0;
                    end
                endcase
        end
endmodule
```

## Draw line and circle

```
// This module implements Bresenham's line drawing algorithm

module draw_line(
        clock,
        reset,
        x1,
        y1,
        x2,
        y2,
        color,
        z,
        shape,
        draw_line_start,
        draw_line_busy,
        buffer_start,
        buffer_busy,
        buffer_line_count,
        buffer_pixel_count,
        buffer_data,error,state);
        output   [5:0]   state;

        parameter   PRECISION   =   12;
        output   signed   [PRECISION   -   1   :   0]   error;
        //output   signed   [PRECISION   -1   :   0]   next_x;

        input   clock;
        input   reset;
        input   signed   [PRECISION   -   1   :   0]   x1;
        input   signed   [PRECISION   -   1   :   0]   y1;
        input   signed   [PRECISION   -   1   :   0]   x2;
        input   signed   [PRECISION   -   1   :   0]   y2;
        input   signed   [PRECISION   -   1   :   0]   z;
        input   [23:0]   color;
        input   draw_line_start;
        output   draw_line_busy;
        output   buffer_start;
        input   buffer_busy;
        output   buffer_data;
        output   [9:0]   buffer_line_count;
        output[9:0]   buffer_pixel_count;
        input   shape;

        localparam   IDLE   =   0;
        localparam   DRAW   =   1;
        localparam   DRAW2   =   2;
        localparam   CORNER1   =   3;
        localparam   CORNER2   =   4;
        localparam   CORNER3   =   5;
        localparam   CORNER4   =   6;
```

```
    localparam   CIRCLE_DRAW   =   7;
    localparam   CIRCLE_DRAW1   =   8;
    localparam   CIRCLE_DRAW2   =   9;
    localparam   CIRCLE_DRAW3   =   10;
    localparam   CIRCLE_DRAW4   =   11;
    localparam   CIRCLE_DRAW5   =   12;
    localparam   CIRCLE_DRAW6   =   13;
    localparam   CIRCLE_DRAW7   =   14;
    localparam   CIRCLE_DRAW8   =   15;
    localparam   WAIT_CORNER1   =   16;
    localparam   WAIT_CORNER2   =   17;
    localparam   WAIT_CORNER3   =   18;
    localparam   WAIT_CORNER0   =   19;
    localparam   WAIT_CIRCLE_DRAW0   =   20;
    localparam   WAIT_CIRCLE_DRAW1   =   21;
    localparam   WAIT_CIRCLE_DRAW2   =   22;
    localparam   WAIT_CIRCLE_DRAW3   =   23;
    localparam   WAIT_CIRCLE_DRAW4   =   24;
    localparam   WAIT_CIRCLE_DRAW5   =   25;
    localparam   WAIT_CIRCLE_DRAW6   =   26;
    localparam   WAIT_CIRCLE_DRAW7   =   27;
    localparam   WAIT_CIRCLE_DRAW8   =   28;
    localparam   WAIT_DRAW_LINE   =   29;


    reg   [5:0]   state,   next_state;

    wire   signed   [PRECISION   -   1   :   0]   absx,   absy,   tx1,   tx2,   ty1,
ty2,fx1,fx2,fy1,fy2;
    wire   signed   [PRECISION   :   0]   deltax,   absdeltay;
    reg   signed[PRECISION   -   1   :   0]   x,   y,   next_x,   next_y;
    reg   signed   [PRECISION   -   1:   0]   error,   next_error;
    reg ibuffer_start, next_ibuffer_start;
    reg [9:0] ibuffer_line_count, next_ibuffer_line_count;
    reg [9:0] ibuffer_pixel_count, next_ibuffer_pixel_count;
    reg [35:0] ibuffer_data, next_ibuffer_data;

    //assign buffer_start = (draw_line_busy) ? ibuffer_start : 1'bZ;
    //assign buffer_line_count = (draw_line_busy) ? ibuffer_line_count : 10'bZ;
    //assign buffer_pixel_count = (draw_line_busy) ? ibuffer_pixel_count : 10'bZ;
    //assign buffer_data = (draw_line_busy) ? ibuffer_data : 36'bZ;
    assign buffer_start = ibuffer_start;
    assign buffer_line_count = ibuffer_line_count;
    assign buffer_pixel_count = ibuffer_pixel_count;
    assign buffer_data = ibuffer_data;

    reg signed [PRECISION-1:0] dx, dy, next_dx, next_dy;

    wire [PRECISION - 1 : 0] sign_dy;

    reg draw_line_busy, next_busy;

    wire swapped_xy;

    wire signed [PRECISION-1:0] radius;

    assign radius = x2;

    assign absx = (x1 > x2) ? (x1 - x2) : (x2 - x1);
    assign absy = (y1 > y2) ? (y1 - y2) : (y2 - y1);

    assign swapped_xy = absy > absx;
    assign tx1 = swapped_xy ? y1 : x1;
    assign tx2 = swapped_xy ? y2 : x2;
    assign ty1 = swapped_xy ? x1 : y1;
    assign  ty2  =  swapped_xy  ?  x2  :  y2;

    assign  fx1  =  (tx1  >  tx2)  ?  tx2  :  tx1;
    assign  fx2  =  (tx1  >  tx2)  ?  tx1  :  tx2;
    assign  fy1  =  (tx1  >  tx2)  ?  ty2  :  ty1;
    assign  fy2  =  (tx1  >  tx2)  ?  ty1  :  ty2;

    assign  deltax  =  fx2  -  fx1;
    assign  absdeltay  =  (fy2  >  fy1)  ?  (fy2  -  fy1)  :  (fy1  -  fy2);
    assign  sign_dy  =  fy2  >  fy1  ?  1  :  -1;

    always  @  (posedge  clock)
    begin
            if  (reset)
            begin
                    state  <=  IDLE;
                    draw_line_busy  <=  0;
                    ibuffer_start  <=  0;
            end
            else  begin
                    state  <=  next_state;
                    draw_line_busy  <=  next_busy;
                    x  <=  next_x;
                    y  <=  next_y;
                    dx  <=  next_dx;
                    dy  <=  next_dy;
                    error  <=  next_error;
                    ibuffer_line_count  <=  next_ibuffer_line_count;
                    ibuffer_pixel_count  <=  next_ibuffer_pixel_count;
                    ibuffer_data  <=  next_ibuffer_data;
                    ibuffer_start  <=  next_ibuffer_start;
            end
    end
```

```
        always @ (state or fx1 or fx2 or fy1 or fy2 or x or y or
draw_line_busy or ibuffer_data or ibuffer_line_count or ibuffer_pixel_count
        or error or draw_line_start or z or color or deltax or buffer_busy or
swapped_xy or absdeltay or sign_dy
        or dx or dy or x1 or y1 or radius or shape)
        begin
                next_state = state;
                next_x = x;
                next_y = y;
                next_dx = dx;
                next_dy = dy;
                next_error = error;
                next_busy = draw_line_busy;
                next_ibuffer_start = 0;
                next_ibuffer_data = ibuffer_data;
                next_ibuffer_line_count = ibuffer_line_count;
                next_ibuffer_pixel_count = ibuffer_pixel_count;

                case (state)
                        IDLE: begin
                                if (draw_line_start && !shape)
                                begin
                                        next_state = WAIT_DRAW_LINE;
                                        next_x = fx1;
                                        next_y = fy1;
                                        next_error = - deltax / 2;
                                        next_busy = 1;
                                        next_ibuffer_start = 1;
                                        next_ibuffer_data = {z, color};
                                        if (swapped_xy)
                                        begin
                                                next_ibuffer_line_count = fx1;
                next_ibuffer_pixel_count = fy1;
            end
            else begin
                next_ibuffer_line_count = fy1;
                next_ibuffer_pixel_count = fx1;
            end
        end else if (draw_line_start && shape)
        begin
            next_state = WAIT_CORNER0;
            next_busy = 1;
            next_x = 0;
            next_y = radius;
            next_dx = 0;
            next_dy = - 2 * radius;
            next_error = 1 - radius;
            //draw the first corner
            next_ibuffer_start = 1;
            next_ibuffer_data = {z, color};
            next_ibuffer_pixel_count = x1;
            next_ibuffer_line_count = y1 + radius;
        end
    end
    WAIT_DRAW_LINE:begin
        next_state = DRAW;
    end
    DRAW: begin
        if (x < fx2)
        begin
            if (buffer_busy == 0)
            begin
                next_state = DRAW2;
                next_error = (error + absdeltay > 0) ? error
+ absdeltay - deltax : error + absdeltay;
```

```
                              next_x  =  x  +  1;
                              next_y  =  (error  +  absdeltay  >  0)  ?  y  +
sign_dy  :  y;
                              if  (swapped_xy)
                              begin
                                  next_ibuffer_line_count  =  next_x;
                                  next_ibuffer_pixel_count  =  next_y;
                              end
                              else  begin
                                  next_ibuffer_line_count  =  next_y;
                                  next_ibuffer_pixel_count  =  next_x;
                              end
                              next_busy  =  1;
                              next_ibuffer_start  =  1;
                              next_ibuffer_data  =  {z,  color};
                          end
                      end
                      else  begin
                          next_state  =  IDLE;
                          next_x  =  0;
                          next_y  =  0;
                          next_error  =  0;
                          next_busy  =  0;
                          next_ibuffer_start  =  0;
                          next_ibuffer_data  =  0;
                      end
                  end
                  DRAW2:  begin
                      next_state  =  DRAW;
                  end
                  WAIT_CORNER0:begin
                      next_state  =  CORNER1;
                  end
```

```
CORNER1:begin
        if  (!buffer_busy)
        begin
                next_state  =  WAIT_CORNER1;
                //draw  the  second  corner
                next_ibuffer_start  =  1;
                next_ibuffer_data  =  {z,  color};
                next_ibuffer_pixel_count  =  x1;
                next_ibuffer_line_count  =  y1  -  radius;
        end
end
WAIT_CORNER1:begin
        next_state  =  CORNER2;
end
CORNER2:begin
        if  (!buffer_busy)
        begin
                next_state  =  WAIT_CORNER2;
                //draw  the  third  corner
                next_ibuffer_start  =  1;
                next_ibuffer_data  =  {z,  color};
                next_ibuffer_pixel_count  =  x1  +  radius;
                next_ibuffer_line_count  =  y1;
        end
end
WAIT_CORNER2:begin
        next_state  =  CORNER3;
end
CORNER3:begin
        if  (!buffer_busy)
        begin
                next_state  =  WAIT_CORNER3;
                //draw  the  fourth  corner
                next_ibuffer_start  =  1;
                next_ibuffer_data  =  {z,  color};
                next_ibuffer_pixel_count  =  x1  -  radius;
                next_ibuffer_line_count  =  y1;
        end
end
WAIT_CORNER3:begin
        next_state  =  CORNER4;
end
CORNER4:begin
        if  (!buffer_busy)
        begin
                next_state  =  CIRCLE_DRAW;
        end
end
CIRCLE_DRAW:begin
        if  (x  >=  y)
        begin
                next_state  =  IDLE;
                next_busy  =  0;
        end
        else  begin
                next_x  =  x  +  1;
                next_dx  =  next_dx  +  2;
                if  (error  >=  0)
                begin
                        next_y  =  y  -  1;
                        next_dy  =  dy  +  2;
                        next_error  =  error  +  next_dy  +  next_dx  +  1;
                end
                else
                begin
                        next_error  =  error  +  next_dx  +  1;
                end
                next_state    =    WAIT_CIRCLE_DRAW0;
                //    draw    point    in    1st    i.e.    top    right
                //    octant
                next_ibuffer_start    =    1;
                next_ibuffer_data    =    {z,    color};
                next_ibuffer_pixel_count    =    x1    +    next_x;
                next_ibuffer_line_count    =    y1    +    next_y;
        end
end
WAIT_CIRCLE_DRAW0:begin
        next_state    =    CIRCLE_DRAW1;
end
CIRCLE_DRAW1:begin
        if    (!buffer_busy)
        begin
                next_state    =    WAIT_CIRCLE_DRAW1;
                //    draw    point    in    2nd    octant    in
                //    clockwise    direction
                next_ibuffer_start    =    1;
                next_ibuffer_data    =    {z,    color};
                next_ibuffer_pixel_count    =    x1    +    y;
                next_ibuffer_line_count    =    y1    +    x;
        end
end
WAIT_CIRCLE_DRAW1:begin
        next_state    =    CIRCLE_DRAW2;
end
CIRCLE_DRAW2:begin
        if    (!buffer_busy)
        begin
                next_state    =    WAIT_CIRCLE_DRAW2;
                //    draw    point    in    3rd    octant    in
                // clockwise direction
                next_ibuffer_start = 1;
                next_ibuffer_data = {z, color};
                next_ibuffer_pixel_count = x1 + y;
                next_ibuffer_line_count = y1 - x;
        end
end
WAIT_CIRCLE_DRAW2:begin
        next_state = CIRCLE_DRAW3;
end
CIRCLE_DRAW3:begin
        if (!buffer_busy)
        begin
                next_state = WAIT_CIRCLE_DRAW3;
                // draw point in 4th octant in
                // clockwise direction
                next_ibuffer_start = 1;
                next_ibuffer_data = {z, color};
                next_ibuffer_pixel_count = x1 + x;
                next_ibuffer_line_count = y1 - y;
        end
end
WAIT_CIRCLE_DRAW3:begin
        next_state = CIRCLE_DRAW4;
end
CIRCLE_DRAW4:begin
        if (!buffer_busy)
        begin
                next_state = WAIT_CIRCLE_DRAW4;
                // draw point in 5th octant in
                // clockwise direction
                next_ibuffer_start = 1;
                next_ibuffer_data = {z, color};
                next_ibuffer_pixel_count   =   x1   -   x;
                next_ibuffer_line_count   =   y1   -   y;
        end
end
WAIT_CIRCLE_DRAW4:begin
        next_state   =   CIRCLE_DRAW5;
end
CIRCLE_DRAW5:begin
        if   (!buffer_busy)
        begin
                next_state   =   WAIT_CIRCLE_DRAW5;
                //   draw   point   in   6th   octant   in
                //   clockwise   direction
                next_ibuffer_start   =   1;
                next_ibuffer_data   =   {z,   color};
                next_ibuffer_pixel_count   =   x1   -   y;
                next_ibuffer_line_count   =   y1   -   x;
        end
end
WAIT_CIRCLE_DRAW5:begin
        next_state   =   CIRCLE_DRAW6;
end
CIRCLE_DRAW6:begin
        if   (!buffer_busy)
        begin
                next_state   =   WAIT_CIRCLE_DRAW6;
                //   draw   point   in   7th   octant   in
                //   clockwise   direction
                next_ibuffer_start   =   1;
                next_ibuffer_data   =   {z,   color};
                next_ibuffer_pixel_count   =   x1   -   y;
                next_ibuffer_line_count   =   y1   +   x;
        end
```

```
                end
                WAIT_CIRCLE_DRAW6:begin
                        next_state = CIRCLE_DRAW7;
                end
                CIRCLE_DRAW7:begin
                        if (!buffer_busy)
                        begin
                                next_state = WAIT_CIRCLE_DRAW7;
                                // draw point in 2nd octant in
                                // clockwise direction
                                next_ibuffer_start = 1;
                                next_ibuffer_data = {z, color};
                                next_ibuffer_pixel_count = x1 - x;
                                next_ibuffer_line_count = y1 + y;
                        end
                end
                WAIT_CIRCLE_DRAW7:begin
                        next_state = CIRCLE_DRAW8;
                end
                CIRCLE_DRAW8:begin
                        if (!buffer_busy)
                        begin
                                next_state = CIRCLE_DRAW;
                        end
                end
        endcase
    end
endmodule
```

## Translate

```
`timescale 1ns / 1ps
// translate by 4 pixels
// direction = 0 up
//                                  1   down
//                                  2   left
//                                  3   right
module translate(reset, clock, start, busy, objcount, direction, ram_data_in,
ram_data_out, ram_addr, ram_we,new_x2,new_y2, aaa);
            output [31:0] new_x2, new_y2;
            output [47:0] aaa;

        parameter PRECISION = 12;

        input reset;
        input clock;
        input start;
        output busy;
        input [10:0] objcount;
        input [1:0] direction;
        output [84:0] ram_data_in;
        input [84:0] ram_data_out;
        output [10:0] ram_addr;
        output ram_we;

        reg[84:0] ram_data_in, new_ram_data_in;
        reg [10:0] ram_addr, new_ram_addr;
        reg ram_we, new_ram_we;

        localparam UP = 0;
        localparam DOWN = 1;
        localparam LEFT = 2;
        localparam RIGHT = 3;

        wire signed [31:0] x1, y1, x2, y2, new_x1, new_y1, new_x2, new_y2;
        assign x1 = ram_data_out[84:73];
        assign y1 = ram_data_out[72:61];
        assign   x2   =   ram_data_out[60:49];
        assign   y2   =   ram_data_out[48:37];
        assign   new_x1   =   (direction   ==   UP   ||   direction   ==   DOWN)   ?   x1   :
(direction   ==   LEFT)   ?   x1   -   4   :   x1   +   4;
        assign   new_y1   =   (direction   ==   LEFT   ||   direction   ==   RIGHT)   ?   y1   :
(direction   ==   UP)   ?   y1   -   4   :   y1   +   4;
        assign   new_x2   =   (direction   ==   UP   ||   direction   ==   DOWN)   ?   x2   :
(direction   ==   LEFT)   ?   x2   -   4   :   x2   +   4;
        assign   new_y2   =   (direction   ==   LEFT   ||   direction   ==   RIGHT)   ?   y2   :
(direction   ==   UP)   ?   y2   -   4   :   y2   +   4;
        assign   aaa   =   {new_x1[PRECISION-1:0],   new_y1[PRECISION-1:0],
new_x2[PRECISION-1:0],   new_y2[PRECISION-1:0]};

        localparam   IDLE   =   0;
        localparam   READ1   =   1;
        localparam   READ2   =   2;
        localparam   WRITE1   =   3;
        localparam   WRITE2   =   4;
        localparam   WRITE3   =   5;
        localparam   READ0   =   6;

        reg   [2:0]   state,   new_state;
        reg   busy,   new_busy;

        always   @   (posedge   clock)
        begin
                if   (reset)
                begin
                        state   <=   IDLE;
                        busy   <=   0;
                        ram_data_in   <=   0;
                        ram_addr   <=   0;
                        ram_we   <=   0;
                end
                else    begin
                        state   <=   new_state;
                        busy   <=   new_busy;
                        ram_data_in   <=   new_ram_data_in;
                        ram_addr   <=   new_ram_addr;
                        ram_we   <=   new_ram_we;
                end
        end

        always  @(state  or  busy  or  ram_data_in  or  ram_addr  or  ram_we  or  start  or
ram_data_out   or   objcount
        or   new_x1   or   new_y1   or   new_x2   or   new_y2)
        begin
                //default   values
                new_state      =   state;
                new_busy   =   busy;
                new_ram_data_in   =   ram_data_in;
                new_ram_addr   =   ram_addr;
                new_ram_we   =   0;

                case   (state)
                        IDLE:begin
                        if   (start)
                        begin
                            new_state   =   READ0;
                            new_ram_addr   =   0;
                            new_busy   =   1;
                        end
                        end
                        READ0:begin
                        new_state   =   READ1;
                        end
                        READ1:begin
```

```
                    if   (ram_addr   <   objcount)
                         new_state   =   READ2;
                    else   begin
                         new_state   =   IDLE;
                         new_busy   =   0;
                    end
               end
               READ2:begin
                    if   (ram_data_out[0]   ==   0)
                    begin
                         new_ram_data_in   =   {new_x1[PRECISION-1:0],
new_y1[PRECISION-1:0],   new_x2[PRECISION-1:0],   new_y2[PRECISION-1:0],
ram_data_out[36:0]};
                    end
                    else
                         new_ram_data_in   =   {new_x1[PRECISION-1:0],
new_y1[PRECISION-1:0],   ram_data_out[60:0]};
                    new_state   =   WRITE1;
                    new_ram_we   =   1;
               end
               WRITE1:begin
                    new_state   =   WRITE2;
                    new_ram_we   =   0;
               end
               WRITE2:begin
                    new_state   =   WRITE3;
                    new_ram_addr   =   ram_addr   +   1;
               end
               WRITE3:begin
                    new_state   =   READ1;
               end
          endcase
     end
endmodule
```

## Display

```
`timescale 1ns / 1ps
module display(pixel_clock, reset, vga_out_red, vga_out_green, vga_out_blue,
vga_out_sync_b, vga_out_blank_b,
        vga_out_pixel_clock, vga_out_hsync, vga_out_vsync, drawchar_start, drawchar_busy,
        drawchar_pixel_count,drawchar_line_count, drawchar_data, drawchar_buffer_start,
x1_laser, y1_laser, x2_laser, y2_laser,
        next_laser, write_pixel_count, write_line_count, render_state,
        up_sync, down_sync, left_sync, right_sync, rotate1_sync, rotate2_sync,
        ram_dinb, ram_addrb, ram_web, ram_doutb, mouse_data, mouse_clock, mx, my,
btn_click, object_count, m_fsm_state);
        output [5:0] m_fsm_state;
        output [11:0] mx, my;
        output [2:0] btn_click;
        output [10:0] object_count;

        output [84:0] ram_dinb;
        output [10:0] ram_addrb;
        output ram_web;
        output [84:0] ram_doutb;

        inout mouse_data, mouse_clock;

        input up_sync, down_sync, left_sync, right_sync, rotate1_sync, rotate2_sync;

        output [9:0] write_pixel_count, write_line_count;
        output [3:0] render_state;

        input pixel_clock;
        input reset;
        output [7:0] vga_out_red;
        output [7:0] vga_out_green;
        output [7:0] vga_out_blue;
        output  vga_out_sync_b;
        output  vga_out_blank_b;
        output  vga_out_pixel_clock;
        output  vga_out_hsync;
        output  vga_out_vsync;

        input  [9:0]  drawchar_line_count;
        input  [9:0]  drawchar_pixel_count;
        input  drawchar_data;
        input  drawchar_buffer_start;
        output  drawchar_start;
        input  drawchar_busy;

        output  [7:0]  x1_laser,  y1_laser,  x2_laser,  y2_laser;
        input  next_laser;

        assign  vga_out_pixel_clock  =  pixel_clock;
        assign  vga_out_sync_b  =  1'b1;

        wire  [9:0]  pixel;
        wire  [9:0]  line;
        wire  neg_hsync;
        wire  neg_vsync;
        wire  button1_sync;
        wire  switch_buffer;
        wire  [9:0]  write_pixel_count;
        wire  [9:0]  write_line_count;
        wire  write_data;
        wire  buffer_start;
        wire  buffer_busy;

        wire  add_start,  add_busy,  remove_start,  remove_busy,  render_busy;
        wire  [10:0]  object_count;
```

```
        wire   [84:0]   ram_data_in,   ram_data_out;
        wire   [10:0]   ram_addr;
        wire   ram_we   =   0;
        wire   blank;

        wire   buffer_write_mux;

        wire   [9:0]   drawline_pixel_count,   drawline_line_count;
        wire   drawline_buffer_start;
        wire   drawline_data;

        wire   [11:0]   line_x1;
        wire   [11:0]   line_y1;
        wire   [11:0]   line_x2;
        wire   [11:0]   line_y2;
        wire   [23:0]   line_color;
        wire   [11:0]   line_z;
        wire   render_shape;
        wire   line_busy;
        wire   render_start;

        assign   vga_out_blank_b   =   blank;

        vga   vga1(.clk(pixel_clock),   .reset(reset),   .neg_hsync(neg_hsync),
.neg_vsync(neg_vsync),   .pixel_count(pixel),   .line_count(line),   .blank(blank));

        delay   #(3)   delay1(reset,   pixel_clock,   !neg_hsync,   vga_out_hsync);
        delay   #(3)   delay2(reset,   pixel_clock,   !neg_vsync,   vga_out_vsync);

        wire   [18:0]   write_addr0;
        wire   [84:0]   ram_dinb;
        wire   [10:0]   ram_addrb;
        wire   ram_web;
```

```
	wire [84:0] ram_doutb;
	wire [23:0] rgb;

	// double buffer
	/*buffer buffer1(.clock(pixel_clock), .reset(reset_sync), .switch_buffer(switch_buffer),
.write_pixel_count(write_pixel_count),
	.write_line_count(write_line_count), .write_color_and_z(write_data),
.read_pixel_count(pixel), .read_line_count(line),
	.rgb({vga_out_red, vga_out_green,
vga_out_blue}),.ram0_we(ram0_we_b),.ram0_data(ram0_data),
	.ram0_addr(ram0_address), .ram1_we(ram1_we_b),.ram1_data(ram1_data),
.ram1_addr(ram1_address),
	.start(buffer_start), .busy(buffer_busy), .write_addr0(write_addr0));*/

	ram_buffer buffer1(.clock(pixel_clock), .reset(reset),
.switch_buffer(switch_buffer),.write_pixel_count(write_pixel_count),
	.write_line_count(write_line_count), .write_color_and_z(write_data),
.read_pixel_count(pixel), .read_line_count(line),
	.rgb(rgb),.start(buffer_start), .busy(buffer_busy));

	wire rotate_dir;
	wire [1:0] translate_dir;
	wire [10:0] trans_ram_addr, rotate_ram_addr;
	wire [84:0] trans_ram_data, rotate_ram_data;

	/*assign vga_out_red = 0;
	assign vga_out_green = 0;
	assign vga_out_blue = 8'hFF;*/

	/*main_fsm main_fsm1(.reset(reset), .clock(pixel_clock), .mouse_x(0), .mouse_y(0),
.keyboard(0),
	.render_start(render_start), .render_object_count(object_count),
	.render_busy(render_busy), .add_start(add_start), .add_busy(add_busy),
	.remove_start(remove_start), .remove_busy(remove_busy), .translate_start(translate_start),
	.translate_busy(translate_busy), .rotate_start(rotate_start),
	.rotate_busy(rotate_busy),    .ram_mux(ram_mux),    .up_sync(up_sync),
.down_sync(down_sync),
	.left_sync(left_sync),    .right_sync(right_sync),    .rotate1_sync(rotate1_sync),
	.rotate2_sync(rotate2_sync),    .rotate_dir(rotate_dir),    .translate_dir(translate_dir));*/


	ram    ram1(
	.clka(pixel_clock),
	.dina(85'b0),
	.addra(ram_addr),
	.wea(0),
	.douta(ram_data_out),
	.clkb(pixel_clock),
	.dinb(ram_dinb),
	.addrb(ram_addrb),
	.web(ram_web),
	.doutb(ram_doutb));


	//main_fsm    main_fsm1(.reset(reset),    .clock(pixel_clock),    .mouse_x(0),    .mouse_y(0),
.keyboard(0),    .render_start(render_start),
.render_object_count(object_count),.render_busy(render_busy),    .add_start(add_start),
.add_busy(add_busy),    .remove_start(remove_start),    .remove_busy(remove_busy));


	//ram
ram1(.clka(pixel_clock),.dina(ram_data_in),.addra(ram_addr),.wea(0),.douta(ram_data_out));


	//wire    [11:0]    x1_vga,    y1_vga,    x2_vga,    y2_vga;


	render    render1(.reset(reset),    .clock(pixel_clock),    .objcount(object_count),
.busy(render_busy),
	.ram_data(ram_data_out),    .ram_addr(ram_addr),    .x1_vga(line_x1),    .y1_vga(line_y1),
.x2_vga(line_x2),
	.y2_vga(line_y2),    .color_vga(line_color),    .z_vga(line_z),.x1_laser(x1_laser),
.y1_laser(y1_laser),    .x2_laser(x2_laser),    .y2_laser(y2_laser),
	.next_laser(next_laser),    .start_laser(start_laser),    .start_vga(line_start),
.busy_vga(line_busy),    .shape_out(render_shape),
        .switch_buffer(switch_buffer),   .blank(blank),   .buffer_write_mux(buffer_write_mux),
        .drawchar_start(drawchar_start),   .drawchar_busy(drawchar_busy),   .state(render_state),
.buffer_busy(buffer_busy));


        draw_line   draw_line1(.clock(pixel_clock),   .reset(reset),        .x1(line_x1),   .y1(line_y1),
.x2(line_x2),
        .y2(line_y2),   .color(line_color),   .z(line_z),   .draw_line_start(line_start),
.draw_line_busy(line_busy),
        .buffer_start(drawline_buffer_start),   .buffer_busy(buffer_busy),
.buffer_line_count(drawline_line_count),
        .buffer_pixel_count(drawline_pixel_count),   .buffer_data(drawline_data),
.shape(render_shape));


        assign   write_pixel_count   =   ({10{buffer_write_mux   ==   0}}   &
drawchar_pixel_count)   |
                                              ({10{buffer_write_mux   ==   1}}   &
drawline_pixel_count);
        assign   write_line_count   =   ({10{buffer_write_mux   ==   0}}   &   drawchar_line_count)
|
                                              ({10{buffer_write_mux   ==   1}}   &
drawline_line_count);
        assign   write_data   =   ({buffer_write_mux   ==   0}   &   drawchar_data)   |
                                      ({buffer_write_mux   ==   1}   &   drawline_data);
        assign   buffer_start   =   ((buffer_write_mux   ==   0)   &   drawchar_buffer_start)   |
                                                   ((buffer_write_mux   ==   1)   &
drawline_buffer_start);




        //   mouse
        wire   [11:0]   mx,my;




        wire   [2:0]      btn_click;
        ps2_mouse_xy   m1(.clk(pixel_clock),   .reset(reset),   .ps2_clk(mouse_clock),
.ps2_data(mouse_data),   .mx(mx),   .my(my),   .btn_click(btn_click));
        defparam          m1.MAX_X   =   640-3;      //   max   -   blob   size
        defparam          m1.MAX_Y   =   480-3;


        assign   {vga_out_red,   vga_out_green,   vga_out_blue}   =   (pixel   >=   mx   -   1   &&
pixel   <=   mx   +   1   &&   line   >=   my   -   1   &&   line   <=   my   +   1)   ?
24'hFF0000   :   rgb;


        wire   [11:0]   x1,   y1,   x2,   y2,   dx,   dy,   radius;
        wire   [1:0]   ram_mux;
        //wire   [4:0]   numobj;
        //wire   shape,   remove_success,   bl,   bc,   bt,   br,   ba,   sl,   sc,   st,   sr,   sa;
        //wire   [1:0]   dt;   wire   dr;


        //wire   [2:0]   bpixel;
        //wire                  fpixel   =   (drawchar_line_count==0   |   drawchar_line_count==639
|
        //                                              drawchar_pixel_count==0   |
drawchar_pixel_count==479);
        //blob   ball(.x(mx),   .y(my),   .hcount(drawchar_line_count),
.vcount(drawchar_pixel_count),   .pixel(bpixel));


        //wire   [2:0]   pixels;
        //wire   blank;
        //assign   pixels   =   (switch[0]   ?   drawchar_line_count[7:5]   :   {3{fpixel}}   |   bpixel)
&   ~blank;


        wire   [63:0]   dispdata   =   {mx,my,1'b0,btn_click,36'b0};
        wire   [84:0]   add_ram_data;
        wire   [10:0]   add_ram_addr;
        wire   add_ram_we;
        wire   [84:0]   remove_ram_data;
        wire   [10:0]   remove_ram_addr;
        wire   remove_ram_we;
        display_16hex   d1(.reset(reset),   .clock_27mhz(pixel_clock),   .data(dispdata),
                        .disp_blank(disp_blank),   .disp_clock(disp_clock),   .disp_rs(disp_rs),
.disp_ce_b(disp_ce_b),
                        .disp_reset_b(disp_reset_b),   .disp_data_out(disp_data_out));

        wire   [5:0]   m_fsm_state;
        //   mouse   FSM
        m_fsm   mouseFSM(.clk(pixel_clock),   .reset(reset),   .mx(mx),   .my(my),
.buttons(btn_click),
                             .x1(x1),   .y1(y1),   .x2(x2),   .y2(y2),   .radius(radius),
.shape(shape),   .numobj(object_count),
                             .remove_success(remove_success),
                             .busy_trans(translate_busy),   .busy_rotate(rotate_busy),
.busy_add(ba),
                             .start_trans(translate_start),   .start_rotate(rotate_start),
.start_add(sa),
                             .direction_trans(translate_dir),   .direction_rotate(rotate_dir),
.state(m_fsm_state),
                             .ram_mux(ram_mux)
                             );

        add   add1(.clock(pixel_clock),   .reset(reset),   .x1(x1),   .y1(y1),   .x2(x2),   .y2(y2),
.shape(shape),
                           .objcount(object_count),   .start(sa),   .busy(ba),
.ram_data(add_ram_data),   .ram_addr(add_ram_addr),   .ram_we(add_ram_we));

        rotate   rotate1(.reset(reset),   .clock(pixel_clock),   .start(rotate_start),   .busy(rotate_busy),
.objcount(object_count),
        .direction(rotate_dir),   .ram_data_in(rotate_ram_data),   .ram_data_out(ram_doutb),
.ram_addr(rotate_ram_addr),   .ram_we(rotate_ram_we));

        translate   translate1(.reset(reset),   .clock(pixel_clock),   .start(translate_start),
.busy(translate_busy),
        .objcount(object_count),   .direction(translate_dir),   .ram_data_in(trans_ram_data),
.ram_data_out(ram_doutb),   .ram_addr(trans_ram_addr),   .ram_we(trans_ram_we));
```

```
        //muxes  for  ram
        assign  ram_dinb  =  (ram_mux  <=  1)  ?  (ram_mux  ==  0  ?  add_ram_data  :
remove_ram_data)
                                                      :  (ram_mux  ==  2  ?  trans_ram_data  :
rotate_ram_data);
        assign  ram_addrb  =  (ram_mux  <=  1)  ?  (ram_mux  ==  0  ?  add_ram_addr  :
remove_ram_addr)
                                                      :  (ram_mux  ==  2  ?  trans_ram_addr  :
rotate_ram_addr);
        assign  ram_web  =  (ram_mux  <=  1)  ?  (ram_mux  ==  0  ?  add_ram_we  :
remove_ram_we)
                                                      :  (ram_mux  ==  2  ?  trans_ram_we  :
rotate_ram_we);

        /*always  @  (ram_mux  or  add_ram_data  or  add_ram_addr  or  add_ram_we  or
                                  remove_ram_data  or  remove_ram_addr  or  remove_ram_we
or
                                  trans_ram_data  or  trans_ram_addr  or  trans_ram_we  or
                                  rotate_ram_data  or  rotate_ram_addr  or  rotate_ram_we)
        begin
                case  (ram_mux)
                        0:  begin
                                ram_dinb  =  add_ram_data;
                                ram_addrb  =  add_ram_addr;
                                ram_web  =  add_ram_we;
                        end
                        1:  begin
                                ram_dinb  =  remove_ram_data;
                                ram_addrb  =  remove_ram_addr;
                                ram_web  =  remove_ram_we;
                        end
                        2:  begin
                                ram_dinb  =  trans_ram_data;
                                ram_addrb  =  trans_ram_addr;
```

```
                    ram_web   =   trans_ram_we;
                end
                3:   begin
                    ram_dinb   =   rotate_ram_data;
                    ram_addrb   =   rotate_ram_addr;
                    ram_web   =   rotate_ram_we;
                end
            endcase
        end*/

endmodule
```

## VGA

```
//   VGA   controller
module   vga(clk,   reset,   neg_hsync,   neg_vsync,   pixel_count,   line_count,   blank);
        input   clk;
        input   reset;
        output   neg_hsync;
        output   neg_vsync;
        output   [9:0]   pixel_count;
        output   [9:0]   line_count;
        output   blank;
        reg   neg_hsync;
        reg   neg_vsync;
        reg   [9:0]   pixel_count;
        reg   [9:0]   line_count;
        reg   blank;
        parameter   PIXELS   =   800;
        parameter   LINES   =   525;
        parameter   HACTIVE_VIDEO   =   640;
        parameter   HFRONT_PORCH   =   16;
        parameter   HSYNC_PERIOD   =   96;
        parameter   HBACK_PORCH   =   48;
        parameter   VACTIVE_VIDEO   =   480;
        parameter   VFRONT_PORCH   =   11;
        parameter   VSYNC_PERIOD   =   2;
        parameter   VBACK_PORCH   =   32;

        always   @   (posedge   clk)
        begin
                if   (reset)
                begin
                        pixel_count  <=  0;
                        line_count  <=  0;
                end  else
                begin
                        if  (pixel_count  ==  PIXELS-1)
                        begin
                                pixel_count  <=  0;
                                if  (line_count  ==  LINES-1)
                                        line_count  <=  0;
                                else
                                        line_count  <=  line_count  +  1;
                        end
                        else
                                pixel_count  <=  pixel_count  +  1;
                end
        end

        always  @  (pixel_count  or  line_count  or  reset)
        begin
                neg_hsync  =  ((pixel_count  <  HACTIVE_VIDEO+HFRONT_PORCH)  ||
(pixel_count  >=  HACTIVE_VIDEO+HFRONT_PORCH+HSYNC_PERIOD))  &&  (reset  ==
0);
                neg_vsync  =  ((line_count  <  VACTIVE_VIDEO+VFRONT_PORCH)  ||
(line_count  >=  VACTIVE_VIDEO+VFRONT_PORCH+VSYNC_PERIOD))  &&  (reset  ==
0);
                blank  =  ~((line_count  >=  VACTIVE_VIDEO)  ||  (pixel_count  >
HACTIVE_VIDEO));
        end
endmodule
```

## Delay

```
module  delay  (reset,  clock,  signal_in,  delayed_signal);

      input  reset;
      input  clock;
      input  signal_in;
      output  delayed_signal;

      parameter  NUM_CYCLE  =  5;
      reg  [NUM_CYCLE-1:0]  tmp;

      assign  delayed_signal  =  tmp[NUM_CYCLE-1];
      always  @  (posedge  clock)
      begin
```

```
                if   (reset)
                begin
                        tmp   <=   0;
                end
                else   begin
                        tmp   <=   {tmp[NUM_CYCLE-2:0],   signal_in};
                end
        end
endmodule
```