A User Interface Source Codes

```
//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ADV7185 Video Decoder Configuration Init
//
// Created:
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
// Register 0
//////////////////////////////////////////////////////////////////////////////

`define INPUT_SELECT                                    4'h0
  // 0: CVBS on AIN1 (composite video in)
  // 7: Y on AIN2, C on AIN5 (s-video in)
  // (These are the only configurations supported by the 6.111 labkit hardware)
`define INPUT_MODE                                      4'h0
  // 0: Autodetect: NTSC or PAL (BGHID), w/o pedestal
  // 1: Autodetect: NTSC or PAL (BGHID), w/pedestal
  // 2: Autodetect: NTSC or PAL (N), w/o pedestal
  // 3: Autodetect: NTSC or PAL (N), w/pedestal
  // 4: NTSC w/o pedestal
  // 5: NTSC w/pedestal
  // 6: NTSC 4.43 w/o pedestal
  // 7: NTSC 4.43 w/pedestal
  // 8: PAL BGHID w/o pedestal
  // 9: PAL N w/pedestal
  // A: PAL M w/o pedestal
  // B: PAL M w/pedestal
  // C: PAL combination N
  // D: PAL combination N w/pedestal
  // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

//////////////////////////////////////////////////////////////////////////////
// Register 1
//////////////////////////////////////////////////////////////////////////////

`define VIDEO_QUALITY                                   2'h0
  // 0: Broadcast quality
  // 1: TV quality
  // 2: VCR quality
  // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE                            1'b0
  // 0: Normal mode
  // 1: Square pixel mode
`define DIFFERENTIAL_INPUT                              1'b0
```

```
  // 0: Single-ended inputs
  // 1: Differential inputs
`define FOUR_TIMES_SAMPLING                     1'b0
  // 0: Standard sampling rate
  // 1: 4x sampling rate (NTSC only)
`define BETACAM                                 1'b0
  // 0: Standard video input
  // 1: Betacam video input
`define AUTOMATIC_STARTUP_ENABLE                1'b1
  // 0: Change of input triggers reacquire
  // 1: Change of input does not trigger reacquire

`define ADV7185_REGISTER_1 {`AUTOMATIC_STARTUP_ENABLE, 1'b0, `BETACAM,
                            `FOUR_TIMES_SAMPLING, `DIFFERENTIAL_INPUT,
                            `SQUARE_PIXEL_IN_MODE, `VIDEO_QUALITY}

////////////////////////////////////////////////////////////////////////////
// Register 2
////////////////////////////////////////////////////////////////////////////

`define Y_PEAKING_FILTER                        3'h4
  // 0: Composite =  4.5dB,  s-video =  9.25dB
  // 1: Composite =  4.5dB,  s-video =  9.25dB
  // 2: Composite =  4.5dB,  s-video =  5.75dB
  // 3: Composite =  1.25dB, s-video =  3.3dB
  // 4: Composite =  0.0dB,  s-video =  0.0dB
  // 5: Composite = -1.25dB, s-video = -3.0dB
  // 6: Composite = -1.75dB, s-video = -8.0dB
  // 7: Composite = -3.0dB,  s-video = -8.0dB
`define CORING                                  2'h0
  // 0: No coring
  // 1: Truncate if Y < black+8
  // 2: Truncate if Y < black+16
  // 3: Truncate if Y < black+32

`define ADV7185_REGISTER_2 {3'b000, `CORING, `Y_PEAKING_FILTER}

////////////////////////////////////////////////////////////////////////////
// Register 3
////////////////////////////////////////////////////////////////////////////

`define INTERFACE_SELECT                        2'h0
  // 0: Philips-compatible
  // 1: Broktree API A-compatible
  // 2: Broktree API B-compatible
  // 3: [Not valid]
`define OUTPUT_FORMAT                           4'h0
  // 0: 10-bit @ LLC, 4:2:2 CCIR656
  // 1: 20-bit @ LLC, 4:2:2 CCIR656
  // 2: 16-bit @ LLC, 4:2:2 CCIR656
  // 3: 8-bit @ LLC, 4:2:2 CCIR656
```

```
  // 4: 12-bit @ LLC, 4:1:1
  // 5-F: [Not valid]
  // (Note that the 6.111 labkit hardware provides only a 10-bit interface to
  // the ADV7185.)
`define TRISTATE_OUTPUT_DRIVERS                 1'b0
  // 0: Drivers tristated when ~OE is high
  // 1: Drivers always tristated
`define VBI_ENABLE                              1'b0
  // 0: Decode lines during vertical blanking interval
  // 1: Decode only active video regions

`define ADV7185_REGISTER_3 {`VBI_ENABLE, `TRISTATE_OUTPUT_DRIVERS, `OUTPUT_FORMAT,
                            `INTERFACE_SELECT}

////////////////////////////////////////////////////////////////////////////////
// Register 4
////////////////////////////////////////////////////////////////////////////////

`define OUTPUT_DATA_RANGE                       1'b0
  // 0: Output values restricted to CCIR-compliant range
  // 1: Use full output range
`define BT656_TYPE                              1'b0
  // 0: BT656-3-compatible
  // 1: BT656-4-compatible

`define ADV7185_REGISTER_4 {`BT656_TYPE, 3'b000, 3'b110, `OUTPUT_DATA_RANGE}

////////////////////////////////////////////////////////////////////////////////
// Register 5
////////////////////////////////////////////////////////////////////////////////


`define GENERAL_PURPOSE_OUTPUTS                 4'b0000
`define GPO_0_1_ENABLE                          1'b0
  // 0: General purpose outputs 0 and 1 tristated
  // 1: General purpose outputs 0 and 1 enabled
`define GPO_2_3_ENABLE                          1'b0
  // 0: General purpose outputs 2 and 3 tristated
  // 1: General purpose outputs 2 and 3 enabled
`define BLANK_CHROMA_IN_VBI                     1'b1
  // 0: Chroma decoded and output during vertical blanking
  // 1: Chroma blanked during vertical blanking
`define HLOCK_ENABLE                            1'b0
  // 0: GPO 0 is a general purpose output
  // 1: GPO 0 shows HLOCK status

`define ADV7185_REGISTER_5 {`HLOCK_ENABLE, `BLANK_CHROMA_IN_VBI, `GPO_2_3_ENABLE,
                            `GPO_0_1_ENABLE, `GENERAL_PURPOSE_OUTPUTS}

////////////////////////////////////////////////////////////////////////////////
// Register 7
```

```
////////////////////////////////////////////////////////////////////////////

`define FIFO_FLAG_MARGIN                        5'h10
  // Sets the locations where FIFO almost-full and almost-empty flags are set
`define FIFO_RESET                              1'b0
  // 0: Normal operation
  // 1: Reset FIFO. This bit is automatically cleared
`define AUTOMATIC_FIFO_RESET                    1'b0
  // 0: No automatic reset
  // 1: FIFO is autmatically reset at the end of each video field
`define FIFO_FLAG_SELF_TIME                     1'b1
  // 0: FIFO flags are synchronized to CLKIN
  // 1: FIFO flags are synchronized to internal 27MHz clock

`define ADV7185_REGISTER_7 {`FIFO_FLAG_SELF_TIME, `AUTOMATIC_FIFO_RESET,
                            `FIFO_RESET, `FIFO_FLAG_MARGIN}

////////////////////////////////////////////////////////////////////////////
// Register 8
////////////////////////////////////////////////////////////////////////////

`define INPUT_CONTRAST_ADJUST                       8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register 9
////////////////////////////////////////////////////////////////////////////

`define INPUT_SATURATION_ADJUST                     8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register A
////////////////////////////////////////////////////////////////////////////

`define INPUT_BRIGHTNESS_ADJUST                     8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register B
////////////////////////////////////////////////////////////////////////////

`define INPUT_HUE_ADJUST                            8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

////////////////////////////////////////////////////////////////////////////
// Register C
```

```
////////////////////////////////////////////////////////////////////////////

`define DEFAULT_VALUE_ENABLE                        1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values
`define DEFAULT_VALUE_AUTOMATIC_ENABLE              1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values if lock is lost
`define DEFAULT_Y_VALUE                             6'h0C
  // Default Y value

`define ADV7185_REGISTER_C {`DEFAULT_Y_VALUE, `DEFAULT_VALUE_AUTOMATIC_ENABLE,
                            `DEFAULT_VALUE_ENABLE}

////////////////////////////////////////////////////////////////////////////
// Register D
////////////////////////////////////////////////////////////////////////////

`define DEFAULT_CR_VALUE                            4'h8
  // Most-significant four bits of default Cr value
`define DEFAULT_CB_VALUE                            4'h8
  // Most-significant four bits of default Cb value

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

////////////////////////////////////////////////////////////////////////////
// Register E
////////////////////////////////////////////////////////////////////////////

`define TEMPORAL_DECIMATION_ENABLE                  1'b0
  // 0: Disable
  // 1: Enable
`define TEMPORAL_DECIMATION_CONTROL                 2'h0
  // 0: Supress frames, start with even field
  // 1: Supress frames, start with odd field
  // 2: Supress even fields only
  // 3: Supress odd fields only
`define TEMPORAL_DECIMATION_RATE                    4'h0
  // 0-F: Number of fields/frames to skip

`define ADV7185_REGISTER_E {1'b0, `TEMPORAL_DECIMATION_RATE,
                            `TEMPORAL_DECIMATION_CONTROL,
                            `TEMPORAL_DECIMATION_ENABLE}

////////////////////////////////////////////////////////////////////////////
// Register F
////////////////////////////////////////////////////////////////////////////

`define POWER_SAVE_CONTROL                          2'h0
  // 0: Full operation
  // 1: CVBS only
```

```
  // 2: Digital only
  // 3: Power save mode
`define POWER_DOWN_SOURCE_PRIORITY              1'b0
  // 0: Power-down pin has priority
  // 1: Power-down control bit has priority
`define POWER_DOWN_REFERENCE                    1'b0
  // 0: Reference is functional
  // 1: Reference is powered down
`define POWER_DOWN_LLC_GENERATOR                1'b0
  // 0: LLC generator is functional
  // 1: LLC generator is powered down
`define POWER_DOWN_CHIP                         1'b0
  // 0: Chip is functional
  // 1: Input pads disabled and clocks stopped
`define TIMING_REACQUIRE                        1'b0
  // 0: Normal operation
  // 1: Reacquire video signal (bit will automatically reset)
`define RESET_CHIP                              1'b0
  // 0: Normal operation
  // 1: Reset digital core and I2C interface (bit will automatically reset)

`define ADV7185_REGISTER_F {`RESET_CHIP, `TIMING_REACQUIRE, `POWER_DOWN_CHIP,
                            `POWER_DOWN_LLC_GENERATOR, `POWER_DOWN_REFERENCE,
                            `POWER_DOWN_SOURCE_PRIORITY, `POWER_SAVE_CONTROL}

////////////////////////////////////////////////////////////////////////////////
// Register 33
////////////////////////////////////////////////////////////////////////////////

`define PEAK_WHITE_UPDATE                       1'b1
  // 0: Update gain once per line
  // 1: Update gain once per field
`define AVERAGE_BIRIGHTNESS_LINES               1'b1
  // 0: Use lines 33 to 310
  // 1: Use lines 33 to 270
`define MAXIMUM_IRE                             3'h0
  // 0: PAL: 133, NTSC: 122
  // 1: PAL: 125, NTSC: 115
  // 2: PAL: 120, NTSC: 110
  // 3: PAL: 115, NTSC: 105
  // 4: PAL: 110, NTSC: 100
  // 5: PAL: 105, NTSC: 100
  // 6-7: PAL: 100, NTSC: 100
`define COLOR_KILL                              1'b1
  // 0: Disable color kill
  // 1: Enable color kill

`define ADV7185_REGISTER_33 {1'b1, `COLOR_KILL, 1'b1, `MAXIMUM_IRE,
                             `AVERAGE_BIRIGHTNESS_LINES,
                             `PEAK_WHITE_UPDATE}
```

```
`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
`define ADV7185_REGISTER_14 8'h18
`define ADV7185_REGISTER_15 8'h60
`define ADV7185_REGISTER_16 8'h00
`define ADV7185_REGISTER_17 8'h01
`define ADV7185_REGISTER_18 8'h00
`define ADV7185_REGISTER_19 8'h10
`define ADV7185_REGISTER_1A 8'h10
`define ADV7185_REGISTER_1B 8'hF0
`define ADV7185_REGISTER_1C 8'h16
`define ADV7185_REGISTER_1D 8'h01
`define ADV7185_REGISTER_1E 8'h00
`define ADV7185_REGISTER_1F 8'h3D
`define ADV7185_REGISTER_20 8'hD0
`define ADV7185_REGISTER_21 8'h09
`define ADV7185_REGISTER_22 8'h8C
`define ADV7185_REGISTER_23 8'hE2
`define ADV7185_REGISTER_24 8'h1F
`define ADV7185_REGISTER_25 8'h07
`define ADV7185_REGISTER_26 8'hC2
`define ADV7185_REGISTER_27 8'h58
`define ADV7185_REGISTER_28 8'h3C
`define ADV7185_REGISTER_29 8'h00
`define ADV7185_REGISTER_2A 8'h00
`define ADV7185_REGISTER_2B 8'hA0
`define ADV7185_REGISTER_2C 8'hCE
`define ADV7185_REGISTER_2D 8'hF0
`define ADV7185_REGISTER_2E 8'h00
`define ADV7185_REGISTER_2F 8'hF0
`define ADV7185_REGISTER_30 8'h00
`define ADV7185_REGISTER_31 8'h70
`define ADV7185_REGISTER_32 8'h00
`define ADV7185_REGISTER_34 8'h0F
`define ADV7185_REGISTER_35 8'h01
`define ADV7185_REGISTER_36 8'h00
`define ADV7185_REGISTER_37 8'h00
`define ADV7185_REGISTER_38 8'h00
`define ADV7185_REGISTER_39 8'h00
`define ADV7185_REGISTER_3A 8'h00
`define ADV7185_REGISTER_3B 8'h00

`define ADV7185_REGISTER_44 8'h41
`define ADV7185_REGISTER_45 8'hBB

`define ADV7185_REGISTER_F1 8'hEF
`define ADV7185_REGISTER_F2 8'h80
```

```
module adv7185init (reset, clock_27mhz, source, tv_in_reset_b,
                    tv_in_i2c_clock, tv_in_i2c_data);

   input reset;
   input clock_27mhz;
   output tv_in_reset_b; // Reset signal to ADV7185
   output tv_in_i2c_clock; // I2C clock output to ADV7185
   output tv_in_i2c_data; // I2C data line to ADV7185
   input source; // 0: composite, 1: s-video

   initial begin
      $display("ADV7185 Initialization values:");
      $display("  Register 0:  0x%X", `ADV7185_REGISTER_0);
      $display("  Register 1:  0x%X", `ADV7185_REGISTER_1);
      $display("  Register 2:  0x%X", `ADV7185_REGISTER_2);
      $display("  Register 3:  0x%X", `ADV7185_REGISTER_3);
      $display("  Register 4:  0x%X", `ADV7185_REGISTER_4);
      $display("  Register 5:  0x%X", `ADV7185_REGISTER_5);
      $display("  Register 7:  0x%X", `ADV7185_REGISTER_7);
      $display("  Register 8:  0x%X", `ADV7185_REGISTER_8);
      $display("  Register 9:  0x%X", `ADV7185_REGISTER_9);
      $display("  Register A:  0x%X", `ADV7185_REGISTER_A);
      $display("  Register B:  0x%X", `ADV7185_REGISTER_B);
      $display("  Register C:  0x%X", `ADV7185_REGISTER_C);
      $display("  Register D:  0x%X", `ADV7185_REGISTER_D);
      $display("  Register E:  0x%X", `ADV7185_REGISTER_E);
      $display("  Register F:  0x%X", `ADV7185_REGISTER_F);
      $display("  Register 33: 0x%X", `ADV7185_REGISTER_33);
   end

   //
   // Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
   //

   reg [7:0] clk_div_count, reset_count;
   reg clock_slow;
   wire reset_slow;

   initial
     begin
        clk_div_count <= 8'h00;
        // synthesis attribute init of clk_div_count is "00";
        clock_slow <= 1'b0;
        // synthesis attribute init of clock_slow is "0";
     end

   always @(posedge clock_27mhz)
     if (clk_div_count == 26)
       begin
          clock_slow <= ~clock_slow;
          clk_div_count <= 0;
```

```
    end
  else
    clk_div_count <= clk_div_count+1;

always @(posedge clock_27mhz)
  if (reset)
    reset_count <= 100;
  else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign reset_slow = reset_count != 0;

//
// I2C driver
//

reg load;
reg [7:0] data;
wire ack, idle;

i2c i2c(.reset(reset_slow), .clock4x(clock_slow), .data(data), .load(load),
        .ack(ack), .idle(idle), .scl(tv_in_i2c_clock),
        .sda(tv_in_i2c_data));

//
// State machine
//

reg [7:0] state;
reg tv_in_reset_b;
reg old_source;

always @(posedge clock_slow)
   if (reset_slow)
     begin
        state <= 0;
        load <= 0;
        tv_in_reset_b <= 0;
        old_source <= 0;
     end
   else
     case (state)
       8'h00:
         begin
            // Assert reset
            load <= 1'b0;
            tv_in_reset_b <= 1'b0;
            if (!ack)
              state <= state+1;
         end
       8'h01:
```

```
  state <= state+1;
8'h02:
  begin
     // Release reset
     tv_in_reset_b <= 1'b1;
     state <= state+1;
          end
8'h03:
  begin
     // Send ADV7185 address
     data <= 8'h8A;
     load <= 1'b1;
     if (ack)
       state <= state+1;
  end
8'h04:
  begin
     // Send subaddress of first register
     data <= 8'h00;
     if (ack)
       state <= state+1;
  end
8'h05:
  begin
     // Write to register 0
     data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
     if (ack)
       state <= state+1;
  end
8'h06:
  begin
     // Write to register 1
     data <= `ADV7185_REGISTER_1;
     if (ack)
       state <= state+1;
  end
8'h07:
  begin
     // Write to register 2
     data <= `ADV7185_REGISTER_2;
     if (ack)
       state <= state+1;
  end
8'h08:
  begin
     // Write to register 3
     data <= `ADV7185_REGISTER_3;
     if (ack)
       state <= state+1;
  end
8'h09:
```

```
  begin
     // Write to register 4
     data <= `ADV7185_REGISTER_4;
     if (ack)
       state <= state+1;
  end
8'h0A:
  begin
     // Write to register 5
     data <= `ADV7185_REGISTER_5;
     if (ack)
       state <= state+1;
  end
8'h0B:
  begin
     // Write to register 6
     data <= 8'h00; // Reserved register, write all zeros
     if (ack)
       state <= state+1;
  end
8'h0C:
  begin
     // Write to register 7
     data <= `ADV7185_REGISTER_7;
     if (ack)
       state <= state+1;
  end
8'h0D:
  begin
     // Write to register 8
     data <= `ADV7185_REGISTER_8;
     if (ack)
       state <= state+1;
  end
8'h0E:
  begin
     // Write to register 9
     data <= `ADV7185_REGISTER_9;
     if (ack)
       state <= state+1;
  end
8'h0F: begin
    // Write to register A
    data <= `ADV7185_REGISTER_A;
 if (ack)
    state <= state+1;
end
8'h10:
  begin
     // Write to register B
     data <= `ADV7185_REGISTER_B;
```

```
      if (ack)
        state <= state+1;
  end
8'h11:
  begin
      // Write to register C
      data <= `ADV7185_REGISTER_C;
      if (ack)
        state <= state+1;
  end
8'h12:
  begin
      // Write to register D
      data <= `ADV7185_REGISTER_D;
      if (ack)
        state <= state+1;
  end
8'h13:
  begin
      // Write to register E
      data <= `ADV7185_REGISTER_E;
      if (ack)
        state <= state+1;
  end
8'h14:
  begin
      // Write to register F
      data <= `ADV7185_REGISTER_F;
      if (ack)
        state <= state+1;
  end
8'h15:
  begin
      // Wait for I2C transmitter to finish
      load <= 1'b0;
      if (idle)
        state <= state+1;
  end
8'h16:
  begin
      // Write address
      data <= 8'h8A;
      load <= 1'b1;
      if (ack)
        state <= state+1;
  end
8'h17:
  begin
      data <= 8'h33;
      if (ack)
        state <= state+1;
```

```
  end
8'h18:
  begin
     data <= `ADV7185_REGISTER_33;
     if (ack)
       state <= state+1;
  end
8'h19:
  begin
     load <= 1'b0;
     if (idle)
       state <= state+1;
  end

8'h1A: begin
   data <= 8'h8A;
   load <= 1'b1;
   if (ack)
     state <= state+1;
end
8'h1B:
  begin
     data <= 8'h33;
     if (ack)
       state <= state+1;
  end
8'h1C:
  begin
     load <= 1'b0;
     if (idle)
       state <= state+1;
  end
8'h1D:
  begin
     load <= 1'b1;
     data <= 8'h8B;
     if (ack)
       state <= state+1;
  end
8'h1E:
  begin
     data <= 8'hFF;
     if (ack)
       state <= state+1;
  end
8'h1F:
  begin
     load <= 1'b0;
     if (idle)
       state <= state+1;
  end
```

```
            8'h20:
              begin
                 // Idle
                 if (old_source != source) state <= state+1;
                 old_source <= source;
              end
            8'h21: begin
                 // Send ADV7185 address
                 data <= 8'h8A;
                 load <= 1'b1;
                 if (ack) state <= state+1;
            end
            8'h22: begin
                 // Send subaddress of register 0
                 data <= 8'h00;
                 if (ack) state <= state+1;
            end
            8'h23: begin
                 // Write to register 0
                 data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
                 if (ack) state <= state+1;
            end
            8'h24: begin
                 // Wait for I2C transmitter to finish
                 load <= 1'b0;
                 if (idle) state <= 8'h20;
            end
          endcase

endmodule

// i2c module for use with the ADV7185

module i2c (reset, clock4x, data, load, idle, ack, scl, sda);

   input reset;
   input clock4x;
   input [7:0] data;
   input load;
   output ack;
   output idle;
   output scl;
   output sda;

   reg [7:0] ldata;
   reg ack, idle;
   reg scl;
   reg sdai;

   reg [7:0] state;
```

```
assign sda = sdai ? 1'bZ : 1'b0;

always @(posedge clock4x)
  if (reset)
    begin
       state <= 0;
       ack <= 0;
    end
  else
    case (state)
      8'h00: // idle
        begin
           scl <= 1'b1;
           sdai <= 1'b1;
           ack <= 1'b0;
           idle <= 1'b1;
           if (load)
             begin
                ldata <= data;
                ack <= 1'b1;
                state <= state+1;
             end
        end
      8'h01: // Start
        begin
           ack <= 1'b0;
           idle <= 1'b0;
           sdai <= 1'b0;
           state <= state+1;
        end
      8'h02:
        begin
           scl <= 1'b0;
           state <= state+1;
        end
      8'h03: // Send bit 7
        begin
           ack <= 1'b0;
           sdai <= ldata[7];
           state <= state+1;
        end
      8'h04:
        begin
           scl <= 1'b1;
           state <= state+1;
        end
      8'h05:
        begin
           state <= state+1;
        end
      8'h06:
```

```
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h07:
  begin
     sdai <= ldata[6];
     state <= state+1;
  end
8'h08:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h09:
  begin
     state <= state+1;
  end
8'h0A:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h0B:
  begin
     sdai <= ldata[5];
     state <= state+1;
  end
8'h0C:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h0D:
  begin
     state <= state+1;
  end
8'h0E:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h0F:
  begin
     sdai <= ldata[4];
     state <= state+1;
  end
8'h10:
  begin
     scl <= 1'b1;
     state <= state+1;
```

```
  end
8'h11:
  begin
     state <= state+1;
  end
8'h12:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h13:
  begin
     sdai <= ldata[3];
     state <= state+1;
  end
8'h14:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h15:
  begin
     state <= state+1;
  end
8'h16:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h17:
  begin
     sdai <= ldata[2];
     state <= state+1;
  end
8'h18:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h19:
  begin
     state <= state+1;
  end
8'h1A:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h1B:
  begin
     sdai <= ldata[1];
```

```
      state <= state+1;
    end
  8'h1C:
    begin
      scl <= 1'b1;
      state <= state+1;
    end
  8'h1D:
    begin
      state <= state+1;
    end
  8'h1E:
    begin
      scl <= 1'b0;
      state <= state+1;
    end
  8'h1F:
    begin
      sdai <= ldata[0];
      state <= state+1;
    end
  8'h20:
    begin
      scl <= 1'b1;
      state <= state+1;
    end
  8'h21:
    begin
      state <= state+1;
    end
  8'h22:
    begin
      scl <= 1'b0;
      state <= state+1;
    end
  8'h23: // Acknowledge bit
    begin
      state <= state+1;
    end
  8'h24:
    begin
      scl <= 1'b1;
      state <= state+1;
    end
  8'h25:
    begin
      state <= state+1;
    end
  8'h26:
    begin
      scl <= 1'b0;
```

```
            if (load)
              begin
                 ldata <= data;
                 ack <= 1'b1;
                 state <= 3;
              end
            else
              state <= state+1;
          end
        8'h27:
          begin
             sdai <= 1'b0;
             state <= state+1;
          end
        8'h28:
          begin
             scl <= 1'b1;
             state <= state+1;
          end
        8'h29:
          begin
             sdai <= 1'b1;
             state <= 0;
          end
      endcase

endmodule
```

```
//
// File:   zbt_6111.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Simple ZBT driver for the MIT 6.111 labkit, which does not hide the
// pipeline delays of the ZBT from the user.  The ZBT memories have
// two cycle latencies on read and write, and also need extra-long data hold
// times around the clock positive edge to work reliably.
//

////////////////////////////////////////////////////////////////////////////
// Ike's simple ZBT RAM driver for the MIT 6.111 labkit
//
// Data for writes can be presented and clocked in immediately; the actual
// writing to RAM will happen two cycles later.
//
// Read requests are processed immediately, but the read data is not available
// until two cycles after the intial request.
//
// A clock enable signal is provided; it enables the RAM clock when high.

module zbt_6111(clk, cen, we, addr, write_data, read_data,
                ram_clk, ram_we_b, ram_address, ram_data, ram_cen_b);

   input clk;                   // system clock
   input cen;                   // clock enable for gating ZBT cycles
   input we;                    // write enable (active HIGH)
   input [18:0] addr;           // memory address
   input [35:0] write_data;     // data to write
   output [35:0] read_data;     // data read from memory
   output        ram_clk;       // physical line to ram clock
   output        ram_we_b;      // physical line to ram we_b
   output [18:0] ram_address;   // physical line to ram address
   inout [35:0]  ram_data;      // physical line to ram data
   output        ram_cen_b;     // physical line to ram clock enable

   // clock enable (should be synchronous and one cycle high at a time)
   wire          ram_cen_b = ~cen;

   // create delayed ram_we signal: note the delay is by two cycles!
   // ie we present the data to be written two cycles after we is raised
   // this means the bus is tri-stated two cycles after we is raised.

   reg [1:0]     we_delay;

   always @(posedge clk)
     we_delay <= cen ? {we_delay[0],we} : we_delay;

   // create two-stage pipeline for write data
```

```
    reg [35:0]     write_data_old1;
    reg [35:0]     write_data_old2;
    always @(posedge clk)
      if (cen)
        {write_data_old2, write_data_old1} <= {write_data_old1, write_data};

    // wire to ZBT RAM signals

    assign         ram_we_b = ~we;
    assign         ram_clk = ~clk;      // RAM is not happy with our data hold
    // times if its clk edges equal FPGA's
    // so we clock it on the falling edges
    // and thus let data stabilize longer
    assign         ram_address = addr;

    assign         ram_data = we_delay[1] ? write_data_old2 : {36{1'bZ}};
    assign         read_data = ram_data;

endmodule // zbt_6111
```

```
//
// File:   ntsc2zbt.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Example for MIT 6.111 labkit showing how to prepare NTSC data
// (from Javier's decoder) to be loaded into the ZBT RAM for video
// display.
//
//Modified so that 2 sets of 18 bit data corresponding to 2 pixels
// are written during each write
//
//////////////////////////////////////////////////////////////////////////
// Prepare data and address values to fill ZBT memory with NTSC data

module ntsc_to_zbt(clk, vclk, fvh, dv, din, ntsc_addr, ntsc_data, ntsc_we);

   input          clk;   // system clock
   input          vclk;  // video clock from camera
   input [2:0]    fvh;
   input          dv;
   input [29:0]   din;

   output [18:0] ntsc_addr;
   output [35:0] ntsc_data;
   output        ntsc_we;       // write enable for NTSC data

   parameter     COL_START = 10'd0;
   parameter     ROW_START = 10'd0;

   // here put the luminance data from the ntsc decoder into the ram
   // this is for 1024 x 768 XGA display

   reg [9:0]     col = 0;
   reg [9:0]     row = 0;
   reg [17:0]    vdata = 0;
   reg           vwe;
   reg           old_dv;
   reg           old_frame;     // frames are even / odd interlaced
   reg           old_v;
   reg           even_odd;      // decode interlaced frame to this wire

   wire          frame = fvh[2];
   wire          frame_edge = frame & ~old_frame;
   wire          v_edge = !fvh[2] & !fvh[1] & old_v ;

   always @ (posedge vclk) //LLC1 is reference
     begin
        old_dv <= dv;
        vwe <= dv && !fvh[2] & ~old_dv; // if data valid, write it
        old_frame <= frame;
```

```
      old_v = fvh[1];
      even_odd = frame_edge ? ~even_odd : even_odd;


      if (!fvh[2])
        begin
           col <= fvh[0] ? COL_START :
                  (!fvh[2] && dv && (col < 640+COL_START)) ? col + 1 : col;
           row <= v_edge ? ROW_START :
                  (!fvh[2] && fvh[0] && (row < 480+ROW_START)) ? row + 1 : row;
           vdata <= (dv && !fvh[2]) ? {din[29:22],din[19:15],din[9:5]} : vdata;
        end
  end

// synchronize with system clock

reg [9:0] x[1:0];
reg [9:0] y[1:0];
reg [17:0] data[1:0];
reg        we[1:0];
reg        eo[1:0];

always @(posedge clk)
  begin
     {x[1],x[0]} <= {x[0],col};
     {y[1],y[0]} <= {y[0],row};
     {data[1],data[0]} <= {data[0],vdata};
     {we[1],we[0]} <= {we[0],vwe};
     {eo[1],eo[0]} <= {eo[0],even_odd};
  end

// edge detection on write enable signal

reg old_we;
wire we_edge = we[1] & ~old_we;
always @(posedge clk) old_we <= we[1];

// shift each set of four bytes into a large register for the ZBT

reg [35:0] mydata;
always @(posedge clk)
  if (we_edge)
    mydata <= { mydata[17:0], data[1] };

// compute address to store data in

wire [18:0] myaddr = {y[1][8:0],1'b0, x[1][9:1]};

// update the output address and data only when four bytes ready

reg [18:0]  ntsc_addr;
```

```
    reg [35:0]   ntsc_data;
    wire         ntsc_we = we_edge & (x[1][0]==1'b0);

    always @(posedge clk)
      if ( ntsc_we)
        begin
           ntsc_addr <= myaddr;  // normal and expanded modes
           ntsc_data <= mydata;
        end

endmodule // ntsc_to_zbt
```

```
//
// File:   ntsc_decode.v
// Date:   31-Oct-05
// Author: J. Castro (MIT 6.111, fall 2005)
//
// This file contains the ntsc_decode modules
//
// This module (w/ADV7185init) is used to grab input NTSC video data from the RCA
// phono jack on the right hand side of the 6.111 labkit (connect
// the camera to the LOWER jack).
//
// Modified so that v is low during the entire odd field
//////////////////////////////////////////////////////////////////////////
//
// NTSC decode - 16-bit CCIR656 decoder
// By Javier Castro
// This module takes a stream of LLC data from the adv7185
// NTSC/PAL video decoder and generates the corresponding pixels,
// that are encoded within the stream, in YCrCb format.

// Make sure that the adv7185 is set to run in 16-bit LLC2 mode.

module ntsc_decode(clk, reset, tv_in_ycrcb, ycrcb, f, v, h, data_valid);

   // clk - line-locked clock (in this case, LLC1 which runs at 27Mhz)
   // reset - system reset
   // tv_in_ycrcb - 10-bit input from chip. should map to pins [19:10]
   // ycrcb - 24 bit luminance and chrominance (8 bits each)
   // f - field: 1 indicates an even field, 0 an odd field
   // v - vertical sync: 1 means vertical sync
   // h - horizontal sync: 1 means horizontal sync

   input clk;
   input reset;
   input [9:0] tv_in_ycrcb; // modified for 10 bit input - should be P[19:10]
   output [29:0] ycrcb;
   output        f;
   output        v;
   output        h;
   output        data_valid;
   // output [4:0] state;

   parameter     SYNC_1 = 0;
   parameter     SYNC_2 = 1;
   parameter     SYNC_3 = 2;
   parameter     SAV_f1_cb0 = 3;
   parameter     SAV_f1_y0 = 4;
   parameter     SAV_f1_cr1 = 5;
   parameter     SAV_f1_y1 = 6;
   parameter     EAV_f1 = 7;
   parameter     SAV_VBI_f1 = 8;
```

```
parameter      EAV_VBI_f1 = 9;
parameter      SAV_f2_cb0 = 10;
parameter      SAV_f2_y0 = 11;
parameter      SAV_f2_cr1 = 12;
parameter      SAV_f2_y1 = 13;
parameter      EAV_f2 = 14;
parameter      SAV_VBI_f2 = 15;
parameter      EAV_VBI_f2 = 16;



// In the start state, the module doesn't know where
// in the sequence of pixels, it is looking.

// Once we determine where to start, the FSM goes through a normal
// sequence of SAV process_YCrCb EAV... repeat

// The data stream looks as follows
// SAV_FF|SAV_00|SAV_00|SAV_XY |Cb0|Y0|Cr1|Y1|Cb2|Y |...|EAV sequence
// There are two things we need to do:
//   1. Find the two SAV blocks (stands for Start Active Video perhaps?)
//   2. Decode the subsequent data

reg [4:0]     current_state = 5'h00;
reg [9:0]     y = 10'h000;  // luminance
reg [9:0]     cr = 10'h000; // chrominance
reg [9:0]     cb = 10'h000; // more chrominance

assign        state = current_state;

always @ (posedge clk)
  begin
     if (reset)
       begin

       end
     else
       begin
     // these states don't do much except allow us to know where we are in the stream.
     // whenever the synchronization code is seen, go back to the sync_state before
          // transitioning to the new state
          case (current_state)
            SYNC_1: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_2 : SYNC_1;
            SYNC_2: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_3 : SYNC_1;
            SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
                                     (tv_in_ycrcb == 10'h274) ? EAV_f1 :
                                     (tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
                                     (tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
                                     (tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
                                     (tv_in_ycrcb == 10'h368) ? EAV_f2 :
```

```
                                        (tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
                                        (tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC_1;

            SAV_f1_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_y0;
            SAV_f1_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_cr1;
            SAV_f1_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_y1;
            SAV_f1_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_cb0;

            SAV_f2_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_y0;
            SAV_f2_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_cr1;
            SAV_f2_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_y1;
            SAV_f2_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f2_cb0;

            // These states are here in the event that we want to cover these signals
            // in the future. For now, they just send the state machine back to SYNC_1
            EAV_f1: current_state <= SYNC_1;
            SAV_VBI_f1: current_state <= SYNC_1;
            EAV_VBI_f1: current_state <= SYNC_1;
            EAV_f2: current_state <= SYNC_1;
            SAV_VBI_f2: current_state <= SYNC_1;
            EAV_VBI_f2: current_state <= SYNC_1;

          endcase
        end
    end // always @ (posedge clk)

// implement our decoding mechanism

wire y_enable;
wire cr_enable;
wire cb_enable;

// if y is coming in, enable the register
// likewise for cr and cb
assign y_enable = (current_state == SAV_f1_y0) ||
                  (current_state == SAV_f1_y1) ||
                  (current_state == SAV_f2_y0) ||
                  (current_state == SAV_f2_y1);
assign cr_enable = (current_state == SAV_f1_cr1) ||
                   (current_state == SAV_f2_cr1);
assign cb_enable = (current_state == SAV_f1_cb0) ||
                   (current_state == SAV_f2_cb0);

// f, v, and h only go high when active
assign {v,h} = (current_state == SYNC_3) ? tv_in_ycrcb[7:6] : 2'b00;

// data is valid when we have all three values: y, cr, cb
assign data_valid = y_enable;
assign ycrcb = {y,cr,cb};

reg     f = 0;
```

```
    always @ (posedge clk)
      begin
         y <= y_enable ? tv_in_ycrcb : y;
         cr <= cr_enable ? tv_in_ycrcb : cr;
         cb <= cb_enable ? tv_in_ycrcb : cb;
         f <= (current_state == SYNC_3) ? tv_in_ycrcb[8] : f;
      end

endmodule
```

```
/*************************************************************************
 **
 ** Module: ycrcb2rgb
 **
 * Convert ycrcb to rgb color space
 *************************************************************************/

module YCrCb2RGB ( R, G, B, clk, rst, Y, Cr, Cb );

   output [7:0]  R, G, B;

   input         clk,rst;
   input [9:0]   Y, Cr, Cb;

   wire [7:0]    R,G,B;
   reg [20:0]    R_int,G_int,B_int,X_int,A_int,B1_int,B2_int,C_int;
   reg [9:0]     const1,const2,const3,const4,const5;
   reg [9:0]     Y_reg, Cr_reg, Cb_reg;

   //registering constants
   always @(posedge clk)
     begin
        const1 = 10'b 0100101010; //1.164 = 01.00101010
        const2 = 10'b 0110011000; //1.596 = 01.10011000
        const3 = 10'b 0011010000; //0.813 = 00.11010000
        const4 = 10'b 0001100100; //0.392 = 00.01100100
        const5 = 10'b 1000000100; //2.017 = 10.00000100
     end

   always @(posedge clk or posedge rst)
     if(rst)
       begin
          Y_reg <= 0; Cr_reg <= 0; Cb_reg <= 0;
       end
     else
       begin
          Y_reg <= Y; Cr_reg <= Cr; Cb_reg <= Cb;
       end

   always @(posedge clk or posedge rst)
     if(rst)
       begin
          A_int <= 0; B1_int <= 0; B2_int <= 0; C_int <= 0; X_int <= 0;
       end
     else
       begin
          X_int <= (const1 * (Y_reg - 'd64)) ;
          A_int <= (const2 * (Cr_reg - 'd512));
          B1_int <= (const3 * (Cr_reg - 'd512));
          B2_int <= (const4 * (Cb_reg - 'd512));
          C_int <= (const5 * (Cb_reg - 'd512));
```

```
        end

    always @(posedge clk or posedge rst)
      if (rst)
        begin
           R_int <= 0; G_int <= 0; B_int <= 0;
        end
      else
        begin
           R_int <= X_int + A_int;
           G_int <= X_int - B1_int - B2_int;
           B_int <= X_int + C_int;
        end

    /* limit output to 0 - 4095, <0 equals 0 and >4095 equals 4095 */
    assign R = (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] : 8'b11111111;
    assign G = (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] : 8'b11111111;
    assign B = (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] : 8'b11111111;

endmodule
```

```
////////////////////////////////////////////////////////////////////////////
// parameterized delay line

module delayN(clk,in,out);
   input clk;
   input in;
   output out;

   parameter NDELAY = 3;

   reg [NDELAY-1:0] shiftreg;
   wire             out = shiftreg[NDELAY-1];

   always @(posedge clk)
     shiftreg <= {shiftreg[NDELAY-2:0],in};

endmodule // delayN
```

```
//////////////////////////////////////////////////////////////////////////
// generate display pixels from reading the ZBT ram
// Each data has 2 18 bit package corresponding to 2 pixels.

module vram_display(reset,clk,hcount,vcount,vr_pixel,
                    vram_addr,vram_read_data);

   input reset, clk;
   input [9:0] hcount;
   input [9:0] vcount;
   output [17:0] vr_pixel;
   output [18:0] vram_addr;
   input [35:0]  vram_read_data;

   wire [9:0]    vshift = vcount + 21;
   wire [9:0]    hshift = hcount + 14;
   wire [18:0]   vram_addr = {vshift[9:1],1'b0,hshift[9:1]};

   wire          hc4 = hcount[0];
   reg [17:0]    vr_pixel;
   reg [35:0]    vr_data_latched;
   reg [35:0]    last_vr_data;

   always @(posedge clk)
     begin
        vr_data_latched <= (hc4) ? vram_read_data : vr_data_latched;
        last_vr_data <= (hc4) ? vr_data_latched : last_vr_data;
     end

   always @(*)
     begin
        case (hc4)
          1'd1 : vr_pixel <= last_vr_data[17:0];
          1'd0 : vr_pixel <= last_vr_data[35:18];
        endcase
     end
endmodule // vram_display
```

```
//module searches for two clumps of bright lights
//these two clumps should correspond to the left and right hands of the user

module findlights(clk, reset, vcount, hcount, validcolor, vcenl, hcenl, vcenr, hcenr);
   input clk, reset;
   input [9:0] vcount, hcount;
   input       validcolor;
   output [9:0] vcenl, hcenl;
   output [9:0] vcenr, hcenr;

   reg [9:0]    vcenl, hcenl;
   reg [9:0]    vcenr, hcenr;

   reg [9:0]    vcen1, hcen1;
   reg [9:0]    vup1,vdown1,hleft1,hright1;
   reg          onevalid;

   wire         up1 = vcount < vup1;
   wire         down1 = vcount > vdown1;
   wire         left1 = hcount < hleft1;
   wire         right1 = hcount > hright1;
   wire [9:0]   vdist1 = vdown1 - vup1;
   wire [9:0]   hdist1 = hright1 - hleft1;
   wire [9:0]   tempvcen1 = vup1 + {0,vdist1[9:1]};
   wire [9:0]   temphcen1 = hleft1 + {0,hdist1[9:1]};
   wire [9:0]   tempdist1 = (hcount>=temphcen1) ?
                (hcount - temphcen1) : (temphcen1 - hcount);
   wire         cantbeone = ((tempdist1>=10) || ((vcount - tempvcen1)>=10));

   reg [9:0]    vcen2, hcen2;
   reg [9:0]    vup2,vdown2,hleft2,hright2;
   reg          twovalid;

   wire         up2 = vcount < vup2;
   wire         down2 = vcount > vdown2;
   wire         left2 = hcount < hleft2;
   wire         right2 = hcount > hright2;
   wire [9:0]   vdist2 = vdown2 - vup2;

   wire [9:0]   hdist2 = hright2 - hleft2;
   wire [9:0]   tempvcen2 = vup2 + {0,vdist2[9:1]};
   wire [9:0]   temphcen2 = hleft2 + {0,hdist2[9:1]};
   wire [9:0]   tempdist2 = (hcount>=temphcen2) ?
                (hcount - temphcen2) : (temphcen2 - hcount);
   wire         cantbetwo = ((tempdist2>=10) || ((vcount - tempvcen2)>=10));

   reg [5:0]    validhist;

   wire         validpixel = (((hcount>9) && (hcount<628)) &&
                              ((vcount>=200) && (vcount<=479)));
```

```
wire          newframe = ((vcount==524) && (hcount==799));

wire          oneisleft = (temphcen1 < temphcen2);

always @ (posedge clk)
  begin
     if (reset)
       begin
          vcen1 <= 0;
          hcen1 <= 0;
          vup1 <= 10'd479;
          vdown1 <= 10'd0;
          hleft1 <= 10'd639;
          hright1 <= 10'd0;
          onevalid <= 0;

          vcen2 <= 0;
          hcen2 <= 0;
          vup2 <= 10'd479;
          vdown2 <= 10'd0;
          hleft2 <= 10'd639;
          hright2 <= 10'd0;
          twovalid <= 0;

          validhist <= 4'd0;
       end
     else
       begin
          validhist <= {validhist[4:0], validcolor};
          if (&(validhist) & validpixel) begin
             if (!onevalid) begin
                onevalid <= 1;
                vup1 <= up1 ? vcount : vup1;
                vdown1 <= down1 ? vcount : vdown1;
                hleft1 <= left1 ? hcount-4 : hleft1;
                hright1 <= right1 ? hcount-4 : hright1;
             end
             else begin
                if (cantbeone) begin
                   if (!twovalid | (twovalid & !cantbetwo)) begin
                      twovalid <= 1;
                      vup2 <= up2 ? vcount : vup2;
                      vdown2 <= down2 ? vcount : vdown2;
                      hleft2 <= left2 ? hcount-4 : hleft2;
                      hright2 <= right2 ? hcount-4 : hright2;
                   end
                end
                else begin
                   onevalid <= 1;
                   vup1 <= up1 ? vcount : vup1;
                   vdown1 <= down1 ? vcount : vdown1;
```

```
                hleft1 <= left1 ? hcount-4 : hleft1;
                hright1 <= right1 ? hcount-4 : hright1;
            end
        end
    end

    if(newframe) begin
        vcen1 <= 0;
        hcen1 <= 0;
        vup1 <= 10'd479;
        vdown1 <= 10'd0;
        hleft1 <= 10'd639;
        hright1 <= 10'd0;
        onevalid <= 0;

        vcen2 <= 0;
        hcen2 <= 0;
        vup2 <= 10'd479;
        vdown2 <= 10'd0;
        hleft2 <= 10'd639;
        hright2 <= 10'd0;
        twovalid <=0;

        validhist <= 4'd0;

        vcenl   <=      oneisleft ? tempvcen1 : tempvcen2;
        hcenl   <=      oneisleft ? temphcen1 : temphcen2;

        vcenr   <=      oneisleft ? tempvcen2 : tempvcen1;
        hcenr   <=      oneisleft ? temphcen2 : temphcen1;

    end
  end
end
endmodule
```

```
//passes the x value of the hand positions
//and passes computes a binary up/down y position

module sendgloveheight(gli_x, gli_y, gri_x, gri_y, glo_x, glo_y, gro_x, gro_y);
   input  [9:0] gli_x, gli_y, gri_x, gri_y;
   output [9:0] glo_x, gro_x;
   output       glo_y, gro_y;

   wire [9:0]   glo_x, gro_x;
   wire         glo_y, gro_y;

   parameter  UPDOWNBOUNDARY = 10'd350;

   assign     glo_x = gli_x;
   assign     gro_x = gri_x;

   assign     glo_y = (gli_y<UPDOWNBOUNDARY);
   assign     gro_y = (gri_y<UPDOWNBOUNDARY);

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Uzoma A. Orji
// Description: Counter takes an input clock and send a pulse on endcount
//
//////////////////////////////////////////////////////////////////////////////////

module counter(clk,reset,enable);
   input clk, reset;
   output enable;

   reg    enable;
   reg [25:0] count;

   parameter  endcount = 3374999;

   always @(posedge clk)
     begin
        if (reset) begin
           count <= 26'd0;
           enable<=1'b0;
        end
        else if (count==endcount) begin
           count<=26'd0;
           enable<=1'b1;
        end
        else begin
           count<=count+1;
           enable<=1'b0;
        end
     end
endmodule
```

```
//controller that inputs the serial data from the ADC and computes the 8bit signal

module serial2parallel(clk, pclk, reset, datain, RFS, CONVST, dataout, valid);
   input clk, pclk;
   input reset;
   input datain;
   input RFS;
   input CONVST;
   output [7:0] dataout;
   output       valid;

   reg [1:0]    state;
   parameter    IDLE = 0;
   parameter    WAITFORRFS = 1;
   parameter    GETDATA = 2;
   parameter    DATAVALID = 3;

   reg          valid;
   reg          oldstate;
   reg [7:0]    dataout, datatemp;

   always @ (negedge clk)
     begin

        if (reset) begin
           datatemp <= 8'b0;
           state <= IDLE;
        end
        else begin
           case (state)
             IDLE:
               if (!CONVST)
                 state <= WAITFORRFS;
               else
                 state <= IDLE;
             WAITFORRFS:
               if (!RFS) begin
                  datatemp <= {datatemp[6:0],datain};
                  state <= GETDATA;
               end
               else
                 state <= WAITFORRFS;
             GETDATA:
               if (RFS) begin
                  state <= DATAVALID;
               end
               else begin
                  datatemp <= {datatemp[6:0],datain};
                  state <= GETDATA;
               end
             DATAVALID:
```

```
                state <= IDLE;
            endcase
        end
    end


  always @ (posedge pclk)
    begin
        oldstate <= state;
        valid <=0;
        if ((state == DATAVALID) && (oldstate!=DATAVALID)) begin
            valid <= 1;
            dataout<= datatemp;
        end
    end
endmodule
```

```
//sendpunch averages four samples from the ADC and computes the maximum punch during
//the interval. It outputs the 2bit punchlevel

module sendpunch(clk, refresh, datain, validin, punchlevel);
   input clk;
   input refresh;
   input [7:0] datain;
   input       validin;

   output [1:0] punchlevel;

   wire [1:0]   punchlevel;

   reg [7:0]    datamax;
   reg [2:0]    num;

   reg [9:0]    tempsum;

   reg          oldvalidin;

   wire         validedge = (validin & !oldvalidin);

   wire         update = (tempsum[9:2]>datamax);

   parameter    HARDPUNCH = 8'b11111000;
   parameter    MEDPUNCH =  8'b11101100;
   parameter    SOFTPUNCH = 8'b11010000;

   wire         hard = (datamax>=HARDPUNCH);
   wire         med = (datamax>=MEDPUNCH);
   wire         soft = (datamax>=SOFTPUNCH);

   assign       punchlevel = hard ? 2'b11 :
                             med ? 2'b10 :
                             soft ? 2'b01 : 2'b00;

   always @(negedge clk)
     begin

        if (refresh) begin
           num <= 3'b0;
           datamax <= 8'b0;
           tempsum <= 10'b0;
        end
        else begin
           if (num==3'b100) begin
              num <= 3'b0;
              datamax <= update ? tempsum[9:2] : datamax;
              tempsum <= 10'b0;
           end
           else if (validedge) begin
```

```
                num <= num + 1;
                tempsum <= tempsum + datain;
            end
            else begin
                num <= num;
                tempsum <= tempsum;
            end
        end

    end

  always @(posedge clk)
    oldvalidin <= validin;

endmodule
```

```
//punchdetector is the top level module for the user interface
//it controls the CONVST that starts a conversion for the ADC
//it also controls the refresh bit that clears the sendpunch module
// it outputs the positions of the hands and the punchlevels

module punchdetector (clock_27mhz, pclk, sclk, reset, CONVST, datainl, RFSl, datainr, RFSr
                      gli_x, gli_y, gri_x, gri_y,
                      glo_x, glo_y, gro_x, gro_y, punchlevell, punchlevelr);

   input clock_27mhz, pclk, reset;
   input datainl, RFSl;
   input datainr, RFSr;
   input [9:0] gli_x, gli_y, gri_x, gri_y;

   output      sclk, CONVST;
   output [9:0] glo_x, gro_x;
   output       glo_y, gro_y;
   output [1:0] punchlevell, punchlevelr;

   reg [3:0]    clkcount;
   always @ (posedge clock_27mhz)
     clkcount <= clkcount + 1;

   BUFG pixel_clock_buf (.I(clkcount[2]), .O(sclk));
   //sclk is 3.375 MHz

   wire         startpulse;
   counter count(sclk,reset,startpulse);
   defparam     count.endcount =        900;
   wire         CONVST = ~startpulse;

   wire [7:0]   dataoutl;
   wire         validl;
   wire [7:0]   dataoutr;
   wire         validr;

   serial2parallel s2pl(sclk, pclk, reset, datainl, RFSl, CONVST, dataoutl, validl);
   serial2parallel s2pr(sclk, pclk, reset, datainr, RFSr, CONVST, dataoutr, validr);

   wire [9:0]   glo_x, gro_x;
   wire         glo_y, gro_y;

   sendgloveheight sgh(gli_x, gli_y, gri_x, gri_y, glo_x, glo_y, gro_x, gro_y);

   wire         refresh;
   counter refresher(pclk, reset, refresh);
        defparam refresher.endcount =   4200000;

   wire [1:0]    punchlevell, punchlevelr;

   sendpunch spl(pclk, refresh, dataoutl, validl, punchlevell);
```

```
    sendpunch spr(pclk, refresh, dataoutr, validr, punchlevelr);
endmodule
```

B Control Unit Source Codes

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    13:45:29 03/19/06
// Design Name:
// Module Name:    controlunit
// Project Name:
// Target Device:
// Tool versions:
// Description: Major FSM for Pong game that also stores the ball's position
//              and velocity, the paddle's position, and gravity well's position and
//              pulse information.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module controlunit(clk, reset, start,
        plx, ply, plp,
        prx, pry, prp,

        player_l_hand_x, player_l_hand_y, player_l_hand_punch,
        player_r_hand_x, player_r_hand_y, player_r_hand_punch,

        opponent_head_x,
        opponent_picture,

        player_life, opponent_life,
        player_energy, opponent_energy,
        time_left, mode, round, game_type, state,

        index_in, data_in, enable_in,
        index_out, data_out, enable_out
        );

//inputs and outputs
input clk, reset, start;
input[9:0] plx, prx;
input      ply, pry;
input[1:0] plp, prp;

output[9:0] player_l_hand_x, player_r_hand_x;
output      player_l_hand_y, player_r_hand_y;
output[1:0] player_l_hand_punch, player_r_hand_punch;
```

```
output[9:0] opponent_head_x;
output[4:0] opponent_picture,
        player_life, opponent_life, player_energy, opponent_energy;
output[7:0] time_left;
output[1:0] mode, round, game_type;
output[2:0] state;

input[2:0] index_in;
input[9:0] data_in;
input enable_in;
output[2:0] index_out;
output[9:0] data_out;
output enable_out;

//Major FSM data
reg[9:0] player_l_hand_x, player_r_hand_x, opponent_l_hand_x, opponent_r_hand_x;
reg      player_l_hand_y, player_r_hand_y, opponent_l_hand_y, opponent_r_hand_y;
reg[1:0] player_l_hand_punch, player_r_hand_punch
        ,opponent_l_hand_punch, opponent_r_hand_punch;
reg[9:0] player_head_x, opponent_head_x;
reg[4:0] player_picture, opponent_picture,
        player_life, opponent_life, player_energy, opponent_energy;
reg[7:0] time_left;
reg endround;

reg[9:0] olx, orx;
reg      oly, ory;
reg[1:0] olp, orp;

reg[1:0] mode, round;
reg[1:0] game_type;
reg[2:0] state;
reg[5:0] energypulse;

//outputs from minor FSMS
wire[9:0] title_l_hand_x, title_r_hand_x;
wire      title_l_hand_y, title_r_hand_y;
wire[1:0] next_mode, next_game_type;

wire[9:0] olx_in, orx_in;
wire      oly_in, ory_in;
wire[1:0] olp_in, orp_in;

wire[9:0] olx_ai, orx_ai;
wire      oly_ai, ory_ai;
wire[1:0] olp_ai, orp_ai;

wire[9:0] plx_in, prx_in, ohx_in;
wire      ply_in, pry_in;
wire[4:0] opic_in, plb_in, olb_in, peb_in, oeb_in;
wire[7:0] time_left_in;
```

```
wire[1:0] plp_in, prp_in;
wire endround_in;

wire[7:0] next_time_left;
wire[9:0] next_player_l_hand_x, next_player_r_hand_x, next_player_head_x;
wire      next_player_l_hand_y, next_player_r_hand_y;
wire[1:0] next_player_l_hand_punch, next_player_r_hand_punch;
wire[4:0] next_player_picture, next_opponent_life, next_player_energy;
wire[9:0] next_opponent_l_hand_x, next_opponent_r_hand_x, next_opponent_head_x;
wire      next_opponent_l_hand_y, next_opponent_r_hand_y;
wire[1:0] next_opponent_l_hand_punch, next_opponent_r_hand_punch;
wire[4:0] next_opponent_picture, next_player_life, next_opponent_energy;

wire[1:0] pstate, ostate;

wire[2:0] index_in, index_in_e, index_out, index_out_su, index_out_sd, index_out_gu,
          index_out_gd;
wire[9:0] data_in, data_out, data_out_u, data_out_d;
wire enable_in, enable_out, enable_out_u, enable_out_d;
wire valid_u, valid_d;

//game modes
parameter TITLE = 2'd0;
parameter ROUND = 2'd1;
parameter REST          = 2'd2;
parameter GAMEOVER= 2'd3;

//game types
parameter ONEPLAYER     = 2'd0;
parameter TWOPLAYERM    = 2'd1;
parameter TWOPLAYERS    = 2'd2;

//game states
parameter IDLE                          = 3'd0;
parameter STARTGETDATA  = 3'd1;
parameter WAITGETDATA   = 3'd2;
parameter STARTCOMPUTE  = 3'd3;
parameter WAITCOMPUTE   = 3'd4;
parameter STARTSENDDATA = 3'd5;
parameter WAITSENDDATA  = 3'd6;

assign index_out[2:0] = (game_type == TWOPLAYERM) ?
        (enable_out ? index_out_sd[2:0] : index_out_gu[2:0]) :
        (enable_out ? index_out_su[2:0] : index_out_gd[2:0]);
assign data_out[9:0] = (game_type == TWOPLAYERM) ? data_out_d[9:0] : data_out_u[9:0];
assign enable_out = (game_type == TWOPLAYERM) ?
        (state==WAITSENDDATA ? enable_out_d : 1'b0) :
        (enable_in || state!=WAITSENDDATA?  1'b0 : enable_out_u);
//assign index_in_e = enable_in ? index_in : 3'd0;
assign index_in_e = enable_out || ~enable_in ? 3'd0 : index_in;
```

```
//Major/Minor FSM controls
reg titlefsm_start;
wire titlefsm_busy;

reg getuserdata_start, senduserdata_start;
wire getuserdata_busy, senduserdata_busy;

reg tlfsm_start, pfsm_start, ofsm_start;
wire tlfsm_busy, pfsm_busy, ofsm_busy;

reg getdispdata_start, senddispdata_start;
wire getdispdata_busy, senddispdata_busy;

reg p_knockouts;
reg o_knockouts;

reg ai_start;
wire ai_busy;

//interfighter connections
wire[6:0] player_stall_inc, opponent_stall_inc;
wire[1:0] player_hit, opponent_hit;

//fighter to ai
wire[6:0] player_stall, opponent_stall;

//data bus communication
//communication comm(data_bus, game_type,
        //index_in,  data_in,  ack_out, enable_in,
        //index_out, data_out, ack_in,  enable_out);

//Minor FSMs
titlefsm title(clk, reset, titlefsm_start, titlefsm_busy,
        plx, prx, ply, pry, plp, prp,
        title_l_hand_x, title_r_hand_x, title_l_hand_y, title_r_hand_y,
        next_mode, next_game_type);

timeleftfsm tl(clk, reset, tlfsm_start, tlfsm_busy, time_left, next_time_left);

fighterfsm      player(clk, reset, pfsm_start, pfsm_busy,
        plx, prx, ply, pry, plp, prp,
        player_energy,  player_stall_inc, player_hit,
        opponent_r_hand_y, ostate, opponent_head_x, opponent_life,

        next_player_l_hand_x, next_player_r_hand_x,
        next_player_l_hand_y, next_player_r_hand_y,
        next_player_l_hand_punch, next_player_r_hand_punch, next_player_head_x,
        next_player_picture, next_opponent_life, next_player_energy,
        opponent_stall_inc,opponent_hit,
        pstate, player_stall);
```

```
fighterfsm      opponent(clk, reset, ofsm_start, ofsm_busy,
        olx, orx, oly, ory, olp, orp,
        opponent_energy, opponent_stall_inc, opponent_hit,
        player_r_hand_y, pstate, player_head_x, player_life,

        next_opponent_l_hand_x, next_opponent_r_hand_x,
        next_opponent_l_hand_y, next_opponent_r_hand_y,
        next_opponent_l_hand_punch, next_opponent_r_hand_punch,
        next_opponent_head_x,
        next_opponent_picture, next_player_life, next_opponent_energy,
        player_stall_inc,
        player_hit,
        ostate, opponent_stall);

ai aifighter(clk, reset, ai_start, ai_busy,
        opponent_stall, player_head_x, player_l_hand_x, player_r_hand_x,
        olx_ai, orx_ai, oly_ai, ory_ai, olp_ai, orp_ai);

//wire sclk;
//reg[3:0] clkcount;
//always @ (posedge clk)
//      clkcount <= clkcount + 1;
//BUFG pixel_clock_buf (.I(clkcount[2]), .O(sclk));
////clk is 3.375 MHz

getuserdata gud(clk, reset, getuserdata_start, getuserdata_busy,
        index_in_e, data_in, enable_in, index_out_gu,
        olx_in, orx_in, oly_in, ory_in, olp_in, orp_in,valid_u);

senduserdata sud(clk, reset, senduserdata_start, senduserdata_busy,
        index_out_su, data_out_u, enable_out_u, index_in,
        plx, prx, ply, pry, plp, prp);

senddispdata sdd(clk, reset, senddispdata_start, senddispdata_busy,
        index_out_sd, data_out_d, enable_out_d, index_in,
        opponent_l_hand_x, opponent_r_hand_x, opponent_l_hand_y,
        opponent_r_hand_y,
        player_head_x, player_picture, time_left,
        opponent_life, player_life, opponent_energy, player_energy,
        opponent_l_hand_punch, opponent_r_hand_punch, endround);

getdispdata gdd(clk, reset, getdispdata_start, getdispdata_busy,
        index_in_e, data_in, enable_in, index_out_gd,
        plx_in, prx_in, ply_in, pry_in, ohx_in, opic_in,
        time_left_in, plb_in, olb_in, peb_in, oeb_in, plp_in, prp_in,
        endround_in,valid_d);

always @(posedge clk) begin
        tlfsm_start <= 0;
        pfsm_start      <= 0;
        ofsm_start      <= 0;
```

```
ai_start                <= 0;
getuserdata_start   <= 0;
senduserdata_start <= 0;
getdispdata_start   <= 0;
senddispdata_start <= 0;
if(reset) begin
        time_left <= 180;
        state    <= IDLE;
        mode             <= TITLE;
        round <=0;
        player_picture                          <= 0;
        opponent_picture                        <= 0;
        player_life                                     <= 31;
        opponent_life                           <= 31;
        player_energy                                   <= 31;
        opponent_energy                         <= 31;
        energypulse <= 0;

        olx<=300;
        orx<=340;
        oly<=1;
        ory<=1;
        olp<=0;
        orp<=0;

        opponent_l_hand_x<=300;
        opponent_r_hand_x<=1;
        opponent_l_hand_y<=340;
        opponent_r_hand_y<=1;
        opponent_l_hand_punch<=0;
        opponent_r_hand_punch<=0;
        player_head_x<=0;
        opponent_head_x<=0;

        endround<=0;

        p_knockouts<=0;
        o_knockouts<=0;

end else begin
        case (mode)
        TITLE: begin
           case (state)
             IDLE:
               if(start) begin
                  titlefsm_start        <= 1;
                  state <= STARTGETDATA;
               end else begin
                  state <= IDLE;
               end
             STARTGETDATA: begin
```

```
    if(~titlefsm_busy) begin
       titlefsm_start <= 1;
       state <= STARTGETDATA;
    end else begin
       state <= WAITGETDATA;
    end
            end
WAITGETDATA:        begin
    if(titlefsm_busy) begin
       state <= WAITGETDATA;
    end else begin
       player_l_hand_x                      <= title_l_hand_x;
       player_r_hand_x                      <= title_r_hand_x;

       player_l_hand_y                      <= title_l_hand_y;
       player_r_hand_y                      <= title_r_hand_y;

       mode                         <= next_mode;
       game_type              <= next_game_type;

       round<=1;
       player_l_hand_punch          <= 0;
       player_r_hand_punch          <= 0;
       player_head_x                                <= 320;
       player_picture                               <= 0;
       player_energy                                <= 31;
       player_life                          <= 31;

       olx <= 300;
       orx <= 340;
       oly <= 1;
       ory <= 1;
       olp <= 0;
       orp <= 0;

       opponent_l_hand_x                    <= 300;
       opponent_r_hand_x                    <= 340;
       opponent_l_hand_y                    <= 1;
       opponent_r_hand_y                    <= 1;
       opponent_l_hand_punch        <= 0;
       opponent_r_hand_punch        <= 0;
       opponent_head_x                      <= 320;
       opponent_picture                     <= 0;
       opponent_energy                      <= 31;
       opponent_life                                <= 31;

       time_left <=180;

       /*if(next_game_type==TWOPLAYERM) begin
        getuserdata_start <= 1;
        state<=STARTGETDATA;
```

```
                end else if(next_game_type==TWOPLAYERS) begin
                senduserdata_start <= 1;
                state <= STARTSENDDATA;
                                        end else*/
               state <= IDLE;
               //init <=1;
            end
         end
         default: state<=IDLE;
      endcase
   end
     ROUND: begin
         if(round==0) begin
            mode<=GAMEOVER;
         end else if (endround) begin
            mode <=REST;
            round <= round + 1;
            time_left <= 30;
            endround <= 0;
         end else
           case (game_type)
            TWOPLAYERS: begin
               //pass through punch data
               case (state)
                 IDLE:
                    if(start) begin
                       senduserdata_start <= 1;
                       state <= STARTSENDDATA;
                    end else begin
                       state <= IDLE;
                    end
                 STARTSENDDATA: begin
                     if(~senduserdata_busy) begin
                        //if(~enable_in)
                        senduserdata_start <= 1;
                        state <= STARTSENDDATA;
                     end else begin
                        state <= WAITSENDDATA;
                     end
                  end
                  WAITSENDDATA: begin
                     if(enable_in) begin
                        getdispdata_start <= 1;
                        state <= STARTGETDATA;
                     end else if(senduserdata_busy) begin
                        state <= WAITSENDDATA;
                     end else begin
                        getdispdata_start <= 1;
                        state <= STARTGETDATA;
                     end
                  end
```

```
        STARTGETDATA: begin
            if(~getdispdata_busy) begin
                getdispdata_start <= 1;
                state <= STARTGETDATA;
            end else begin
                state <= WAITGETDATA;
            end
        end
        WAITGETDATA: begin
            if(getdispdata_busy) begin
                state <= WAITGETDATA;
            end else begin
                if(valid_d && plx_in !=0 && prx_in!=0
                    && plx_in < prx_in) begin
                    player_l_hand_x <= plx_in;
                    player_r_hand_x <= prx_in;
                    player_l_hand_y <= ply_in;
                    player_r_hand_y <= pry_in;
                    player_l_hand_punch <= plp_in;
                    player_r_hand_punch <= prp_in;
                    opponent_head_x <= ohx_in;
                    opponent_picture<= opic_in;
                    time_left                <= time_left_in;
                    player_life              <= plb_in;
                    opponent_life     <= olb_in;
                    player_energy     <= peb_in;
                    opponent_energy <= oeb_in;
                    endround                 <= endround_in;
                end
                senduserdata_start <= 1;
                state <= STARTSENDDATA;
                //state <= IDLE;
                //slave never goes idle
            end
        end
        default: begin
            if(enable_in && state!=WAITGETDATA &&
                state!=STARTGETDATA) begin
                getdispdata_start <= 1;
                state<=STARTGETDATA;
            end else begin
                senduserdata_start <= 1;
                state <= STARTSENDDATA;
            end
            //state<=IDLE;
        end
    endcase
end
default: begin  //for two player m or single player
    case (state)
```

```
IDLE:
   if(start) begin
      if(game_type == ONEPLAYER)
        ai_start <= 1;
      else getuserdata_start <= 1;
      state <= STARTGETDATA;
   end else begin
      state <= IDLE;
   end
STARTGETDATA: begin
    if((game_type == TWOPLAYERM && ~getuserdata_busy)
       || (game_type == ONEPLAYER && ~ai_busy)) begin

       if(game_type == ONEPLAYER)
         ai_start <= 1;
       else getuserdata_start <= 1;

       state <= STARTGETDATA;
    end else begin
       state <= WAITGETDATA;
    end
end
WAITGETDATA: begin
    //forces continue if another start pulse occurs
    if(( (game_type == TWOPLAYERM && getuserdata_busy)
         || (game_type == ONEPLAYER && ai_busy))
       /* && ~start*/) begin
       state <= WAITGETDATA;
    end else begin
       if(game_type == ONEPLAYER && ~ai_busy) begin
          olx <= olx_ai;
          orx <= orx_ai;
          oly <= oly_ai;
          ory <= ory_ai;
          olp <= olp_ai;
          orp <= orp_ai;
       end else if(game_type == TWOPLAYERM
                   && ~getuserdata_busy) begin
          if(valid_u) begin
             // && olx_in!=0 && orx_in!=0 &&
             //  olx_in < orx_in) begin
             olx <= olx_in;
             orx <= orx_in;
             oly <= oly_in;
             ory <= ory_in;
             olp <= olp_in;
             orp <= orp_in;
          end
       end else begin
          /*olx <= 300;
           orx <= 340;
```

```
            oly <= 1;
                                ory <= 1;
            olp <= 0;
            orp <= 0;*/
        end

        tlfsm_start <= 1;
        pfsm_start  <= 1;
        ofsm_start  <= 1;
        energypulse <= energypulse + 1;
        state <= STARTCOMPUTE;
    end
end
STARTCOMPUTE: begin
    //start minor FSM's and repeat until all have started
    //and have reported that they are busy (in computation)
    if(~tlfsm_busy || ~pfsm_busy || ~ofsm_busy) begin
        tlfsm_start <= 1;
        pfsm_start  <= 1;
        ofsm_start  <= 1;
        state <= STARTCOMPUTE;
    end else begin
        state <= WAITCOMPUTE;
    end
end
WAITCOMPUTE: begin
    //wait until all minor FSM's have finished
    // calculating their next values
    if(tlfsm_busy || pfsm_busy || ofsm_busy) begin
        state <= WAITCOMPUTE;
    end else begin //if finished update next values
        // and wait until next computation
        time_left <= next_time_left;

        player_l_hand_x     <= next_player_l_hand_x;
        player_r_hand_x    <= next_player_r_hand_x;
        player_l_hand_y     <= next_player_l_hand_y;
        player_r_hand_y     <= next_player_r_hand_y;
        player_l_hand_punch <= next_player_l_hand_punch;
        player_r_hand_punch   <= next_player_r_hand_punch;
        player_head_x       <= 640 - next_player_head_x;
        player_picture<= next_player_picture;
        player_energy <=
            next_player_energy + &(energypulse) >= 31 ?
            31 : next_player_energy + &(energypulse);
        player_life <= next_player_life;

        opponent_l_hand_x   <= next_opponent_l_hand_x;
        opponent_r_hand_x   <= next_opponent_r_hand_x;
        opponent_l_hand_y   <= next_opponent_l_hand_y;
        opponent_r_hand_y   <= next_opponent_r_hand_y;
```

```
                    opponent_l_hand_punch <= next_opponent_l_hand_punch;
                    opponent_r_hand_punch <= next_opponent_r_hand_punch;
                    opponent_head_x <= 640 - next_opponent_head_x;
                    opponent_picture     <= next_opponent_picture;
                    opponent_energy <=
                          (next_opponent_energy + &(energypulse)) >= 31 ?
                              31 : next_opponent_energy + &(energypulse);
                    opponent_life        <= next_opponent_life;

                    if(time_left==0 ||
                       opponent_life < next_opponent_life ||
                       opponent_life < next_opponent_life) begin
                       if(player_life < next_player_life)
                         p_knockouts<=p_knockouts+1;
                       if(opponent_life < next_opponent_life)
                         o_knockouts<=o_knockouts+1;
                       endround<=1;
                    end

                    if(game_type==ONEPLAYER)
                      state<=IDLE;
                    else begin
                       senddispdata_start <= 1;
                       state <= STARTSENDDATA;
                    end
                 end
              end
              STARTSENDDATA: begin
                 if(~senddispdata_busy) begin
                    senddispdata_start <= 1;
                    state <= STARTSENDDATA;
                 end else begin
                    state <= WAITSENDDATA;
                 end
              end
              WAITSENDDATA: begin
                 /*if (start) begin
                  senddispdata_start <= 1;
                        end else*/ if(senddispdata_busy/* && ~start*/)
                          begin
                            state <= WAITSENDDATA;
                          end else begin
                            state <= IDLE;
                          end
              end
              default: state<=IDLE;
            endcase
          end
        endcase
end
REST: begin
```

```
case(state)
  IDLE: begin
      if(start) begin
         tlfsm_start <= 1;
         state <= STARTCOMPUTE;
      end else begin
         state <= IDLE;
      end
  end
  STARTCOMPUTE: begin
      if(~tlfsm_busy) begin
         tlfsm_start <= 1;
         state <= STARTCOMPUTE;
      end else begin
                  state <= WAITCOMPUTE;
      end
  end

  WAITCOMPUTE: begin
      time_left <= next_time_left;
      player_l_hand_punch          <= 0;
      player_r_hand_punch          <= 0;
      player_head_x                          <= 320;
      player_picture                                 <= 0;
      player_energy                                  <= 31;
      player_life                            <= 31;

      olx <= 300;
      orx <= 340;
      oly <= 1;
      ory <= 1;
      olp <= 0;
      orp <= 0;

      opponent_l_hand_x                      <= 300;
      opponent_r_hand_x                      <= 340;
      opponent_l_hand_y                      <= 1;
      opponent_r_hand_y                      <= 1;
      opponent_l_hand_punch <= 0;
      opponent_r_hand_punch <= 0;
            opponent_head_x                  <= 320;
      opponent_picture                       <= 0;
      opponent_energy                        <= 31;
      opponent_life                                  <= 31;

      if(time_left==0||next_time_left==0) begin
         mode<=ROUND;
                  time_left <=180;
      end
      state<=IDLE;
  end
```

```
                default: state<=IDLE;
            endcase
        end
          GAMEOVER: begin
              //done
           end
        endcase
    end
end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:44:05 04/23/06
// Design Name:
// Module Name:    fighterfsm
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module fighterfsm(clk, reset, start, busy,
        l_hand_x, r_hand_x, l_hand_y, r_hand_y, l_hand_punch, r_hand_punch,
        energy, stall_inc, hit,
        op_hand_y, op_state, op_head_x, op_life,

        next_l_hand_x, next_r_hand_x, next_l_hand_y, next_r_hand_y,
        next_l_hand_punch, next_r_hand_punch, next_head_x,
        picture, next_op_life, next_energy, op_stall_inc, op_hit,
        state, stall);

input clk, reset, start;
input[9:0] l_hand_x, r_hand_x, op_head_x;
input      l_hand_y, r_hand_y, op_hand_y;
input[1:0] l_hand_punch, r_hand_punch;
input[4:0] energy;
input[1:0] op_state;
input[4:0] op_life;
input[6:0] stall_inc;
input[1:0] hit;

output busy;
output[9:0]     next_l_hand_x, next_r_hand_x, next_head_x;
output      next_l_hand_y, next_r_hand_y;
output[1:0]     next_l_hand_punch, next_r_hand_punch;
output[4:0] picture, next_op_life, next_energy;
output[1:0] state;
output[6:0] op_stall_inc;
output[1:0] op_hit;
output[6:0] stall;

reg fsmstate;
```

```
reg[1:0] state;
reg busy;
reg[9:0]        next_l_hand_x, next_r_hand_x, next_head_x;
reg      next_l_hand_y, next_r_hand_y;
reg[1:0]        next_l_hand_punch, next_r_hand_punch;
reg[4:0] picture, next_op_life, next_energy;

reg[1:0] lrpunch;
reg[9:0] punch_x;
reg      punch_y;
reg[1:0] punch_str;
reg[6:0] stall;
reg[6:0] combostall;
reg[6:0] op_stall_inc;
reg[1:0] op_hit;
reg impact;

parameter IDLE = 1'b0;
parameter ACTIVE= 1'b1;

parameter STAND =2'd0;
parameter BLOCK =2'd1;
parameter PUNCH =2'd2;
parameter RECOIL        =2'd3;

parameter STANDING00    = 0;
parameter STANDING01    = 1;
parameter STANDING10    = 2;
parameter STANDING11    = 3;
parameter BLOCKING0     = 4;
parameter BLOCKING1     = 5;
parameter LJAB                  = 6;
parameter RJAB                  = 7;
parameter LBODY         = 8;
parameter RBODY         = 9;
parameter LHOOK         = 10;
parameter RHOOK         = 11;
parameter LHOOKLOW      = 12;
parameter RHOOKLOW      = 13;
parameter LCROSS                = 14;
parameter RCROSS                = 15;
parameter LUPPER                = 16;
parameter RUPPER                = 17;
parameter BOTHLOW               = 18;
parameter BOTHHIGH      = 19;
parameter LRECOIL               = 20;
parameter RRECOIL               = 21;
parameter BOTHRECOIL    = 22;
parameter HITLEFT               = 23;
parameter HITRIGHT      = 24;
parameter HITLOW                = 25;
```

```
always @ (posedge clk) begin
        if(reset) begin
                state <= STAND;
                lrpunch<=0;
                stall <= 0;
                op_stall_inc <=0;
                impact <= 0;
                busy<=0;
                next_energy <= energy;
                next_op_life <= op_life;
                next_op_life <= 31;
                next_l_hand_x <= 300;
                next_r_hand_x <= 340;
                next_l_hand_y <= 0;
                next_r_hand_y <= 0;
                next_l_hand_punch <= 0;
                next_r_hand_punch <= 0;
                next_head_x <= 320;
                op_hit <=0;
        end else begin
                case(fsmstate)
                IDLE: begin
                        if(start) begin
                                busy<=1;
                                fsmstate<=ACTIVE;
                        end else begin
                                busy<=0;
                                fsmstate<=IDLE;
                        end
                end
                ACTIVE: begin
                        stall <= stall + stall_inc;
                        case (hit)
                        2'b00:begin picture<=picture;end
                        2'b01:begin picture<=HITRIGHT;end
                        2'b10:begin picture<=HITLEFT;end
                        2'b11:begin picture<=HITLOW;end
                        endcase

                        case (state)
                        STAND: begin
                            next_energy <= energy;
                            next_op_life <= op_life;
                            op_hit <=0;
                            next_l_hand_x <= l_hand_x;
                            next_r_hand_x <= r_hand_x;
                            next_l_hand_y <= l_hand_y;
                            next_r_hand_y <= r_hand_y;
                            next_l_hand_punch <= l_hand_punch;
                            next_r_hand_punch <= r_hand_punch;
```

```
next_head_x <= ((l_hand_x>>1) + (r_hand_x>>1));
//stall <= 0;
if(stall>0)begin //might want to move before data updates
   stall <= stall - 1;
   state <= STAND;
   //if punch initiated
end else if((l_hand_punch > 0 && energy >= l_hand_punch)
            || (r_hand_punch > 0 && energy >= r_hand_punch)
            || (l_hand_punch > 0 && r_hand_punch > 0
                && energy >= l_hand_punch + r_hand_punch))
  begin
   state <= PUNCH;
   stall <= 7'd20;
   if(r_hand_punch == 0) begin //left punch
      lrpunch<=2'b10;
      punch_x <= l_hand_x;
      punch_y <= l_hand_y;
      punch_str<=l_hand_punch;
      case(l_hand_punch)
        //2'b00: not possible
        2'b01: begin picture <= l_hand_y ? LJAB:LBODY;end
        2'b10: begin picture <= l_hand_y ? LHOOK:LHOOKLOW;end
        2'b11: begin picture <= l_hand_y ? LCROSS:LUPPER;end
      endcase
   end else if(l_hand_punch ==0) begin       //right punch
      lrpunch<=2'b01;
      punch_x <= r_hand_x;
      punch_y <= r_hand_y;
      punch_str<=r_hand_punch;
      case(r_hand_punch)
        //2'b00: not possible
        2'b01: begin picture <= r_hand_y ? RJAB:RBODY;end
        2'b10: begin picture <= r_hand_y ? RHOOK:RHOOKLOW;end
        2'b11: begin picture <= r_hand_y ? RCROSS:RUPPER;end
      endcase
   end else begin    //someone thinks you punch with both hands
      lrpunch<=2'b11;
      punch_x <= ((r_hand_x>>1) + (l_hand_x>>1));
      punch_y <= r_hand_y;
      punch_str<=((r_hand_punch>>1) + (l_hand_punch>>1));
      picture <= r_hand_y ? BOTHHIGH:BOTHLOW;
   end

   //if block initiated
end else if(l_hand_x > r_hand_x - 128
            && ~(l_hand_y ^ r_hand_y)) begin
   state <= BLOCK;
   stall <= 7'd0;
   picture <= r_hand_y? BLOCKING1:BLOCKING0;
end else begin
   state <= STAND;
```

```
            case({l_hand_y,r_hand_y})
              2'b00: begin picture <= STANDING00; end
              2'b01: begin picture <= STANDING01; end
              2'b10: begin picture <= STANDING10; end
              2'b11: begin picture <= STANDING11; end
            endcase
          end
        end //STAND
        BLOCK: begin
            next_energy <= energy;
            next_op_life <= op_life;
            next_l_hand_x <= l_hand_x;
            next_r_hand_x <= r_hand_x;
            next_l_hand_y <= l_hand_y;
            next_r_hand_y <= r_hand_y;
            next_l_hand_punch <= l_hand_punch;
            next_r_hand_punch <= r_hand_punch;
            next_head_x <= ((l_hand_x>>1) + (r_hand_x>>1));
            picture <= r_hand_y? BLOCKING1:BLOCKING0;

               if(stall>0)begin
                  stall <= stall - 1;
                  state <=BLOCK;
               end else if(l_hand_x > r_hand_x - 128
                           && ~(l_hand_y ^ r_hand_y)
                           && l_hand_punch==0 && r_hand_punch==0) begin
                  state <= BLOCK;
               end else begin
                  case({l_hand_y,r_hand_y})
                    2'b00: begin picture <= STANDING00; end
                    2'b01: begin picture <= STANDING01; end
                    2'b10: begin picture <= STANDING10; end
                    2'b11: begin picture <= STANDING11; end
                  endcase
                  state <= STAND;
               end
        end //BLOCK
        PUNCH: begin
            next_energy <= energy;
            next_op_life <= op_life;
            op_hit <= 0;
            if(~impact) begin
               if(op_head_x > punch_x - 50 && op_head_x < punch_x + 50 &&
                  ~(op_state==BLOCK && op_hand_y == punch_y)) begin
                  next_op_life <= op_life - punch_str;
                  case(punch_str)
                    2'b01: begin op_stall_inc <= 7'd15; end
                    2'b10: begin op_stall_inc <= 7'd25; end
                    2'b11: begin op_stall_inc <= 7'd35; end
                  endcase
                  op_hit <= (punch_y == 0) ? 2'b11 : lrpunch;
```

```
            end
            next_energy <= energy - punch_str;
            impact <= 1;
        end else begin
            if(stall>0)begin
                stall <= stall - 1;
                state <=PUNCH;
            end else begin
                impact <= 0;
                case(punch_str)
                  //2'b00: begin stall <= ;combostall <= ; end
                  2'b01: begin stall <= 7'd20; combostall <= 7'd10; end
                  2'b10: begin stall <= 7'd40; combostall <= 7'd20; end
                  2'b11: begin stall <= 7'd60; combostall <= 7'd30; end
                endcase
                case(lrpunch)
                  //2'b00: not possible
                  2'b01: begin picture <= RRECOIL; end
                  2'b10: begin picture <= LRECOIL; end
                  2'b11: begin picture <= BOTHRECOIL; end
                endcase
                state <= RECOIL;
            end
        end
    end
    RECOIL:begin
        next_energy <= energy;
        next_op_life <= op_life;
        if(stall>0)begin
            stall <= stall - 1;
            //if halfway through recoil check for combo
            if(combostall >= stall &&
                (     //jab/body->opposite anything
                      (punch_str==1 &&
                       (
                        (lrpunch==2'b10 && l_hand_punch==0
                         && r_hand_punch>0)
                        ||
                        (lrpunch==2'b01 && l_hand_punch>0
                         && r_hand_punch==0)
                       )
                       )

                      ||

                      //cross->opposite hook
                      //hook->opposite hook
                      (
                       (punch_str==2 || (punch_str==3 && punch_y == 0))
                       && (
                            (lrpunch==2'b10 && l_hand_punch==0
```

```
                                    && r_hand_punch==2)
                                   ||
                                   (lrpunch==2'b01 && l_hand_punch==2
                                    && r_hand_punch==0)
                                   )
                                )

                               ||

                               //hook->opposite upper
                               (
                                (punch_str==2 && punch_y==1) &&
                                (
                                 (lrpunch==2'b10 && l_hand_punch==0 &&
                                  r_hand_punch==3 && r_hand_y==0)
                                 ||
                                 (lrpunch==2'b01 && l_hand_punch==3 &&
                                  r_hand_punch==0 && l_hand_y==0)
                                 )
                                )

                               )
                           ) begin
                           stall<=7'd0;
                           combostall<=7'd0;
                           case({l_hand_y,r_hand_y})
                             2'b00: begin picture <= STANDING00; end
                             2'b01: begin picture <= STANDING01; end
                             2'b10: begin picture <= STANDING10; end
                             2'b11: begin picture <= STANDING11; end
                           endcase
                           state <= STAND;
                       end else
                         state <=RECOIL;
                   end else begin
                       case({l_hand_y,r_hand_y})
                         2'b00: begin picture <= STANDING00; end
                         2'b01: begin picture <= STANDING01; end
                         2'b10: begin picture <= STANDING10; end
                         2'b11: begin picture <= STANDING11; end
                       endcase
                       state <= STAND;
                   end
               end
           default: state <= STAND;
           endcase //state

    busy<=0;
    fsmstate<=IDLE;
end
  default: fsmstate<=IDLE;
```

```
                endcase //fsmstate
        end
end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:58:26 05/01/06
// Design Name:
// Module Name:    ai
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module ai(clk, reset, start, busy,
        fighter_stall, pl_head_x, pl_l_x, pl_r_x,
        lpx, rpx, lpy, rpy, lpp, rpp);

input clk, reset, start;
input[9:0] pl_head_x, pl_l_x, pl_r_x;
input[6:0] fighter_stall;

output busy;
output[9:0] lpx, rpx;
output      lpy, rpy;
output[1:0] lpp, rpp;

//output[9:0] x;

reg busy;
reg[9:0] lpx, rpx;
reg      lpy, rpy;
reg[1:0] lpp, rpp;

reg[9:0] x, xdst;

reg[1:0] state;
reg[7:0] counter;
reg[1:0] movedelay;

reg[5:0] loffset, roffset;

wire[15:0] decision;
wire[15:0] movedist;
wire[15:0] punchstr;
```

```
wire[15:0] hands;

reg enable_decision, enable_movedist,   enable_punchstr, enable_hands;

lfsr decision_prng(clk, reset, 16'd1092 , enable_decision, decision);
lfsr movedist_prng(clk, reset, 16'd30541, enable_movedist, movedist);
lfsr punchstr_prng(clk, reset, 16'd2141 , enable_punchstr, punchstr);
lfsr    hands_prng(clk, reset, 16'd112  ,    enable_hands,    hands);

parameter DECISIONRATE = 64;

parameter IDLE             = 2'd0;
parameter DECISION = 2'd1;
parameter MOVE    = 2'd2;
parameter PUNCH   = 2'd3;

//decision-making        0-65535
parameter  MOVETHRESH = 40000;
parameter PUNCHTHRESH = 0;
//parameter DONOTHING

//movement
parameter MOVELEFT       = 55535;
parameter MOVERIGHT      = 45535;
parameter MOVECLOSER     = 20000;
parameter MOVEAWAY       = 15000;

//punches
parameter WEAK           = 30000;
parameter MEDIUM         = 20000;
//parameter STRONG       =

//hands
parameter FACEBLOCK      = 50000;
parameter BODYBLOCK      = 40000;
parameter FACEOPEN       = 30000;
parameter BODYOPEN       = 20000;
parameter LUPRDOWN       = 15000;
parameter RUPLDOWN       = 10000;


always @(posedge clk) begin
        enable_decision<=0;
        enable_movedist<=0;
        enable_punchstr<=0;
        enable_hands<=0;
        lpp<=0;
        rpp<=0;
        if(reset) begin
                lpx <=10'd220;
                rpx <=10'd420;
```

```
        lpy <=1'd1;
        rpy <=1'd1;
        lpp <=2'd0;
        rpp <=2'd0;
        busy<=0;
        state<=0;
        counter<=0;
        movedelay<=0;
        x<=320;
        xdst <= 320;
        loffset<=50;
        roffset<=50;
end else begin

        case(state)
        IDLE:begin
                if(start) begin
                        if(fighter_stall==0) begin
                                counter <= counter + 1;
                                movedelay <= movedelay + 1;
                        end
                        busy <= 1;
                        state<=DECISION;
                end else begin
                        busy<=0;
                        state<=IDLE;
                end
        end
        DECISION: begin

                if(&(movedelay)) begin
                        if(xdst > x) x <= x + 1;
                        else if(xdst < x) x <= x - 1;
                end

                lpx <= x - loffset;
                rpx <= x + roffset;
                if      (x>pl_l_x&&x-pl_l_x<50 ||
                         x<pl_l_x&&pl_l_x-x<50 ||
                         x>pl_r_x&&x-pl_r_x<50 ||
                         x<pl_r_x&&pl_r_x-x<50 ) begin
                         if(hands > 20000) begin
                                 loffset <= 50;
                                 roffset <= 50;
                                 lpy <= 1;
                                 rpy <= 1;
                         end else begin
                                 loffset <= 50;
                                 roffset <= 50;
                                 lpy <= 0;
                                 rpy <= 0;
```

```
        end
end else if(hands > FACEBLOCK) begin
        loffset <= 50;
        roffset <= 50;
        lpy <= 1;
        rpy <= 1;
end else if(hands > BODYBLOCK) begin
        loffset <= 50;
        roffset <= 50;
        lpy <= 0;
        rpy <= 0;
end else if(hands > FACEOPEN) begin
        loffset <= 100;
        roffset <= 100;
        lpy <= 1;
        rpy <= 1;
end else if(hands > BODYOPEN) begin
        loffset <= 100;
        roffset <= 100;
        lpy <= 0;
        rpy <= 0;
end else if(hands > LUPRDOWN) begin
        loffset <= 50;
        roffset <= 50;
        lpy <= 1;
        rpy <= 0;
end else if(hands > RUPLDOWN) begin
        loffset <= 50;
        roffset <= 50;
        lpy <= 0;
        rpy <= 1;
end else begin

end

if(counter==DECISIONRATE) begin
        counter <= 0;
        enable_hands <= 1;
        enable_decision <= 1;
        busy<=1;
        if(decision > MOVETHRESH) begin
                enable_decision <= 1;
                enable_movedist<=1;
                state <= MOVE;
        end else if(decision > PUNCHTHRESH /*&&
                  (x>pl_head_x&&(x-pl_head_x<64))
                ||(x<pl_head_x&&(pl_head_x-x<64))*/) begin
                enable_punchstr<=1;
                state <= PUNCH;
        end else begin
                busy<=0;
```

```
                        state <= IDLE;
                end

        end else begin
                busy<=0;
                state <= IDLE;
        end
end
MOVE: begin
        busy<=0;
        movedelay <=0;
        state <= IDLE;
        enable_decision <= 1;//for next decision
        if(decision > MOVELEFT) begin
                if      (x > 128 + movedist[5:0])
                  xdst <= x - movedist[5:0];
                else xdst <= 128;
        end else if(decision > MOVERIGHT) begin
                if      (x < 512 - movedist[5:0])
                  xdst <= x + movedist[5:0];
                else xdst <= 512;
        end else if(decision > MOVECLOSER) begin
                if(x > pl_head_x) begin
                        if(x > pl_head_x + 64)
                          xdst <= x - movedist[5:0];
                        else xdst <= pl_head_x;
                end else begin
                        if(x < pl_head_x - 64)
                          xdst <= x + movedist[5:0];
                        else xdst <= pl_head_x;
                end
        end else if(decision > MOVEAWAY) begin
                if(x > pl_head_x) begin
                        if      (x < 512 - movedist[5:0])
                          xdst <= x + movedist[5:0];
                        else xdst <= 512;
                end else begin
                        if      (x > 128 + movedist[5:0])
                          xdst <= x - movedist[5:0];
                        else xdst <= 128;
                end
        end else begin
                xdst<=x;
        end
end
PUNCH: begin
        xdst<=x;
        busy<=0;
        state <= IDLE;
        if(punchstr > WEAK) begin
                if(punchstr[0]) begin
```

```
                                        lpy <= punchstr[10];
                                        lpp <= 1;
                                end else begin
                                        rpy <= punchstr[10];
                                        rpp <= 1;
                                end
                        end else if(punchstr > MEDIUM) begin
                                if(punchstr[0]) begin
                                        lpy <= punchstr[10];
                                        lpp <= 2;
                                end else begin
                                        rpy <= punchstr[10];
                                        rpp <= 2;
                                end
                        end else begin
                                if(punchstr[0]) begin
                                        lpy <= punchstr[10];
                                        lpp <= 3;
                                end else begin
                                        rpy <= punchstr[10];
                                        rpp <= 3;
                                end
                        end
                end
                default: state<=IDLE;
                endcase
        end
end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    13:54:01 05/03/06
// Design Name:
// Module Name:    communication
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module communication(bus, game_type,
        get_index, get_data, get_ack, get_enable,
        send_index, send_data, send_ack, send_enable);

//input clk, reset;
input[1:0] game_type;

inout[15:0] bus;

output[2:0] get_index;
output[9:0] get_data;
input  get_ack;
output get_enable;

input[2:0] send_index;
input[9:0] send_data;
output  send_ack;
input send_enable;

//game types
parameter ONEPLAYER     = 2'd0;
parameter TWOPLAYERM    = 2'd1;
parameter TWOPLAYERS    = 2'd2;

   //data pins  index   data    ack     m_enable        s_enable
//bus address   15:13   12:3    2       1               0


wire get_ack_override;
assign get_ack_override = ~send_enable && ~get_enable ? 1'hZ : get_ack;
```

```
wire slave_backout;
assign slave_backout = (game_type==TWOPLAYERM || (game_type==TWOPLAYERS && ~get_enable));

assign bus[15:2] = (//game_type!=TWOPLAYERM && game_type!=TWOPLAYERS) ? 14'hZ :(
        (send_enable && slave_backout)
                                        ? {send_index[2:0],send_data[9:0],1'hZ    }
                                        : {3'hZ, 10'hZ,   get_ack}//_override}

);

assign bus[1:0] = (//game_type!=TWOPLAYERM && game_type!=TWOPLAYERS) ? 2'hZ :(
        (game_type==TWOPLAYERM) ? (
                /*master*/
                {send_enable,1'hZ}
        ):(
                /*slave*/
                {1'hZ,send_enable}
        )
);

assign {get_index[2:0],get_data[9:0],send_ack} = bus[15:2];
assign get_enable = (game_type==TWOPLAYERM) ? bus[0] : bus[1];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:21:01 04/30/06
// Design Name:
// Module Name:    getuserdata
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module getdispdata(clk, reset, start, busy, index, data, enable, ack,
        plx, prx, ply, pry, ohx, opic, time_left, plb, olb, peb, oeb, plp,
        prp, endround, valid);
input clk, reset, start, enable;
input[2:0] index;
output busy;
output[2:0] ack;
output valid;

input[9:0] data;
output[9:0] plx, prx, ohx;
output ply, pry;
output[4:0] opic, plb, olb, peb, oeb;
output[7:0]     time_left;
output[1:0] plp, prp;
output endround;

reg busy;
reg[2:0] ack;
reg[9:0] plx, prx, ohx;
reg ply, pry;
reg[4:0] opic, plb, olb, peb, oeb;
reg[7:0]        time_left;
reg[1:0] plp, prp;
reg endround;
reg valid;

reg state;

reg enable_sync1, enable_sync;
reg[9:0] data_sync1, data_sync;
```

```
reg[2:0] index_sync2, index_sync1, index_sync;

parameter IDLE = 0;
parameter GET = 1;

parameter DONE           = 3'd0;
parameter PLX            = 3'd1; //10
parameter PRX            = 3'd2; //10
parameter PLRY_TL = 3'd3; //1,1,8
parameter OHX            = 3'd4; //10
parameter OPIC_ENDROUND_PLRP= 3'd5; //5,o,2,2
parameter LIFEBAR = 3'd6; //5,5
parameter ENERBAR = 3'd7; //5,5


always @(posedge clk) begin
        enable_sync1 <= enable;
        enable_sync  <= enable_sync1;
        data_sync1 <= data;
        data_sync  <= data_sync1;
        index_sync2 <= index;
        index_sync1 <= index_sync2;
        index_sync  <= index_sync1;

        if(reset) begin
                state <=IDLE;
                busy<=0;
                ack<=2'd0;
                plx<=300;
                prx<=340;
                ohx<=300;
                ply<=1;
                pry<=1;
                opic<=0;
                plb<=31;
                olb<=31;
                peb<=31;
                oeb<=31;
                time_left<=180;
                endround<=0;
                valid<=0;
        end else if(start) begin
                state <=GET;
                ack<=2'd0;
                busy<=1;
                valid<=0;
        end else begin
                case(state)
                IDLE: begin
                        state <=IDLE;
                        busy<=0;
```

```
end
GET: begin
        busy <=1;
        state<=GET;
        if(index_sync!=ack)begin
                ack <= index_sync;

                case(index_sync)
                DONE: begin
                        state <= IDLE;
                        busy <=0;
                        valid<=1;
                end
                PLX: begin
                        plx <= data;
                end
                PRX: begin
                        prx <= data;
                end
                PLRY_TL: begin
                        ply <= data[9];
                        pry <= data[8];
                        time_left <= data[7:0];
                end
                OHX: begin
                        ohx <= data;
                end
                OPIC_ENDROUND_PLRP: begin
                        opic <= data[9:5];
                        endround <=data[4];
                        plp<= data[3:2];
                        prp <= data[1:0];
                end
                LIFEBAR: begin
                        plb <= data[9:5];
                        olb <= data[4:0];
                end
                ENERBAR: begin
                        peb <= data[9:5];
                        oeb <= data[4:0];
                        //state <= IDLE;
                        //busy <=0;
                        valid<=1;
                end
                default: begin
                        state <= IDLE;
                        busy <=0;
                end
                endcase

        end
```

```
                end
                endcase
        end
end


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:21:01 04/30/06
// Design Name:
// Module Name:    getuserdata
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module getuserdata(clk, reset, start, busy, index, data, enable, ack, lpx, rpx,
                   lpy, rpy, lpp, rpp, valid);
input clk, reset, start, enable;
input[2:0] index;
output busy;
output[2:0] ack;
output[9:0] lpx, rpx;
output lpy, rpy;
output[1:0] lpp, rpp;
input[9:0] data;
output valid;

reg busy;
reg[2:0] ack;
reg[9:0] lpx, rpx;
reg lpy, rpy;
reg[1:0] lpp, rpp;
reg valid;
reg state;

reg enable_sync1, enable_sync;
reg[9:0] data_sync1, data_sync;
reg[2:0] index_sync2, index_sync1, index_sync;

//reg debouncereset;

//debounce dbi0(debouncereset, clk, index[0], index_sync[0]);
//debounce dbi1(debouncereset, clk, index[1], index_sync[1]);
//debounce dbi2(debouncereset, clk, index[2], index_sync[2]);

parameter IDLE = 0;
```

```
parameter GET = 1;

parameter DONE = 3'd0;
parameter LPX  = 3'd1;
parameter RPX  = 3'd2;
parameter LPYP = 3'd3;
parameter RPYP = 3'd4;


always @ (posedge clk) begin
        enable_sync1 <= enable;
        enable_sync  <= enable_sync1;
        data_sync1 <= data;
        data_sync  <= data_sync1;
        index_sync2 <= index;
        index_sync1 <= index_sync2;
        index_sync  <= index_sync1;
        //debouncereset <=0;
        if(reset) begin
                state <=IDLE;
                busy<=0;
                ack<=2'd0;
                lpx<=0;
                rpx<=0;
                lpy<=0;
                rpy<=0;
                lpp<=0;
                rpp<=0;
                valid<=0;
                //debouncereset<=1;
        end else if(start) begin
                state <=GET;
                //last_index<=0;
                ack<=2'd0;
                busy<=1;
                valid<=0;
        end else begin
                case(state)
                IDLE: begin
                        state <=IDLE;
                        busy<=0;
                end
                GET: begin
                        busy <=1;
                        state<=GET;

                        if(/*enable && */index_sync!=ack) begin
                                ack <= index_sync;
                                //if(enable)
                                case(index_sync)
                                DONE: begin
```

```
                                        state <= IDLE;
                                        busy <=0;
                                        valid <=1;
                                    end
                                    LPX: begin
                                        lpx <= data;//_sync;
                                    end
                                    RPX: begin
                                        rpx <= data;//_sync;
                                    end
                                    LPYP: begin
                                        lpy <= data[9];//_sync[9];
                                        lpp <= data[0];//_sync[1:0];
                                    end
                                    RPYP: begin
                                        rpy <= data[9];//_sync[9];
                                        rpp <= data[0];//_sync[1:0];
                                        //state <= IDLE;
                                        //busy <=0;
                                        valid <=1;
                                    end
                                    default: begin
                                        state <= IDLE;
                                        busy <=0;
                                    end
                                    endcase


                            end
                        end

                        endcase
                end
        end


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    19:39:49 04/30/06
// Design Name:
// Module Name:    senddispdata
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module senddispdata(clk, reset, start, busy, index, data, enable, ack,
        plx, prx, ply, pry, ohx, opic, time_left, plb, olb, peb, oeb, plp, prp, endround);
input clk, reset, start;
output busy, enable;

input[9:0] plx, prx, ohx;
input ply, pry;
input[4:0] opic, plb, olb, peb, oeb;
input [7:0] time_left;
input [1:0] plp, prp;
input endround;

output[2:0] index;
input [2:0] ack;
output[9:0] data;

reg[2:0] ack_sync1;
reg[2:0] ack_sync;
reg[2:0] index;
reg busy;
reg state;
reg[9:0] data;

parameter IDLE = 0;
parameter SEND = 1;

parameter DONE          = 3'd0;
parameter PLX           = 3'd1; //11
parameter PRX           = 3'd2; //11
parameter PLRY_TL = 3'd3; //1,1,8
parameter OHX           = 3'd4; //11
```

```
parameter OPIC_ENDROUND_PLRP= 3'd5; //5,1,2,2
parameter LIFEBAR = 3'd6; //5,5
parameter ENERBAR = 3'd7; //5,5


assign enable = busy;

always @(posedge clk) begin
        ack_sync1 <= ack;
        ack_sync <= ack_sync1;
        if(reset) begin
                ack_sync1 <= 2'b00;
                ack_sync <= 2'b00;
                busy <=0;
                state <= IDLE;
                data <=0;
                index <=DONE;
        end else if(start) begin
                state <=SEND;
                index<=PLX;
                busy <=1;
                data <=plx;
        end else begin
                case(state)
                IDLE: begin
                        state <=IDLE;
                        index<=DONE;
                        busy <=0;
                        data <=0;
                end
                SEND: begin
                        busy <= 1;
                        if(ack_sync==index) begin
                                case(index)
                                DONE: begin
                                        busy <=0;
                                        state <= IDLE;
                                        index <= DONE;
                                        data <=0;
                                end
                                PLX: begin
                                        state <= SEND;
                                        index <= PRX;
                                        data  <= prx;
                                end
                                PRX: begin
                                        state <= SEND;
                                        index <= PLRY_TL;
                                        data  <= {ply,pry,time_left};
                                end
                                PLRY_TL: begin
```

```
                                        state <= SEND;
                                        index <= OHX;
                                        data  <= ohx;
                                    end
                                    OHX: begin
                                        state <= SEND;
                                        index <= OPIC_ENDROUND_PLRP;
                                        data  <= {opic,enround,plp,prp};
                                    end
                                    OPIC_ENDROUND_PLRP: begin
                                        state <= SEND;
                                        index <= LIFEBAR;
                                        data  <= {plb,olb};
                                    end
                                    LIFEBAR: begin
                                        state <= SEND;
                                        index <= ENERBAR;
                                        data  <= {peb,oeb};
                                    end
                                    ENERBAR: begin
                                        busy <=0;
                                        state <= IDLE;
                                        index <= DONE;
                                        data <=0;
                                    end
                                    endcase
                                end
                            end
                            default:state <=IDLE;
                            endcase
                    end
end


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:20:50 04/30/06
// Design Name:
// Module Name:    senduserdata
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module senduserdata(clk, reset, start, busy, index, data, enable, ack, lpx, rpx,
                    lpy, rpy, lpp, rpp);
input clk, reset, start;
input[9:0] lpx, rpx;
input lpy, rpy;
output busy, enable;
input[1:0] lpp, rpp;
output[2:0] index;
input [2:0] ack;
output[9:0] data;

reg[2:0] ack_sync1;
reg[2:0] ack_sync;
reg[2:0] index;
reg busy;
reg state;
reg[9:0] data;

parameter IDLE = 0;
parameter SEND = 1;

parameter DONE = 3'd0;
parameter LPX  = 3'd1;
parameter RPX  = 3'd2;
parameter LPYP = 3'd3;
parameter RPYP = 3'd4;

assign enable = busy;

always @(posedge clk) begin
        ack_sync1 <= ack;
```

```
ack_sync <= ack_sync1;
if(reset) begin
        ack_sync1 <= 2'd0;
        ack_sync <= 2'd0;
        busy <=0;
        state <= IDLE;
        data <=0;
        index <=DONE;
end else if(start) begin
        state <=SEND;
        index<=LPX;
        busy <=1;
        data <=lpx;
end else begin
        case(state)
        IDLE: begin
                state <=IDLE;
                index<=DONE;
                busy <=0;
                data <=0;
        end
        SEND: begin
                busy <= 1;
                if(ack_sync==index) begin
                        case(index)
                        DONE:   begin
                                state <= IDLE;
                                index<=DONE;
                                busy <=0;
                                data <=0;
                        end
                        LPX: begin
                                state <= SEND;
                                index <= RPX;
                                data  <= rpx;
                        end
                        RPX: begin
                                state <= SEND;
                                index <= LPYP;
                                data  <= {lpy,7'd0,lpp};
                        end
                        LPYP: begin
                                state <= SEND;
                                index <= RPYP;
                                data  <= {rpy,7'd0,rpp};
                        end
                        RPYP: begin
                                //state <= IDLE;
                                state <= SEND;
                                index <= DONE;
                                //busy <=0;
```

```
                                        data <=0;
                                end
                                default:        begin
                                        state <= IDLE;
                                        index<=DONE;
                                        busy <=0;
                                        data <=0;
                                end
                                endcase
                        end

                end
                default:state <=IDLE;
                endcase

        end
end



endmodule
```

```
// Switch Debounce Module
// use your system clock for the clock input
// to produce a synchronous, debounced output
module debounce (reset, clock, noisy, clean);
   parameter DELAY = 270000;   // .01 sec with a 27Mhz clock
   input reset, clock, noisy;
   output clean;

   reg [18:0] count;
   reg new, clean;

   always @(posedge clock)
     if (reset)
       begin
          count <= 0;
          new <= noisy;
          clean <= noisy;
       end
     else if (noisy != new)
       begin
          new <= noisy;
          count <= 0;
       end
     else if (count == DELAY)
       clean <= new;
     else
       count <= count+1;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:25:54 03/12/06
// Design Name:
// Module Name:    delay
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module delay(clk, in, out);
input clk, in;

output out;

reg del1, del2, out;

always @(posedge clk) begin
        del1 <= in;
        del2 <= del1;
        out <= del2;
end



endmodule
```

```
//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Hex display driver
//
//
// File:   display_16hex.v
// Date:   24-Sep-05
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
// This module drives the labkit hex displays and shows the value of
// 8 bytes (16 hex digits) on the displays.
//
// 24-Sep-05 Ike: updated to use new reset-once state machine, remove clear
// 02-Nov-05 Ike: updated to make it completely synchronous
//
// Inputs:
//
//   reset       - active high
//   clock_27mhz - the synchronous clock
//   data        - 64 bits; each 4 bits gives a hex digit
//
// Outputs:
//
//    disp_*     - display lines used in the 6.111 labkit (rev 003 & 004)
//
//////////////////////////////////////////////////////////////////////////////

module display_16hex (reset, clock_27mhz, data_in,
               disp_blank, disp_clock, disp_rs, disp_ce_b,
               disp_reset_b, disp_data_out);

   input reset, clock_27mhz;    // clock and reset (active high reset)
   input [63:0] data_in;               // 16 hex nibbles to display

   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
          disp_reset_b;

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

   //////////////////////////////////////////////////////////////////////////
   //
   // Display Clock
   //
   // Generate a 500kHz clock for driving the displays.
   //
   //////////////////////////////////////////////////////////////////////////

   reg [5:0] count;
```

```
   reg [7:0] reset_count;
//   reg             old_clock;
   wire      dreset;
   wire      clock = (count<27) ? 0 : 1;

   always @(posedge clock_27mhz)
     begin
        count <= reset ? 0 : (count==53 ? 0 : count+1);
        reset_count <= reset ? 100 : ((reset_count==0) ? 0 : reset_count-1);
//      old_clock <= clock;
     end

   assign dreset = (reset_count != 0);
   assign disp_clock = ~clock;
   wire   clock_tick = ((count==27) ? 1 : 0);
//   wire   clock_tick = clock & ~old_clock;

   ////////////////////////////////////////////////////////////////////////
   //
   // Display State Machine
   //
   ////////////////////////////////////////////////////////////////////////

   reg [7:0] state;             // FSM state
   reg [9:0] dot_index;         // index to current dot being clocked out
   reg [31:0] control;          // control register
   reg [3:0] char_index;        // index of current character
   reg [39:0] dots;             // dots for a single digit
   reg [3:0] nibble;            // hex nibble of current character
   reg [63:0] data;

   assign disp_blank = 1'b0; // low <= not blanked

   always @(posedge clock_27mhz)
     if (clock_tick)
       begin
          if (dreset)
            begin
               state <= 0;
               dot_index <= 0;
               control <= 32'h7F7F7F7F;
            end
          else
            casex (state)
              8'h00:
                begin
                   // Reset displays
                   disp_data_out <= 1'b0;
                   disp_rs <= 1'b0; // dot register
                   disp_ce_b <= 1'b1;
                   disp_reset_b <= 1'b0;
```

```
      dot_index <= 0;
      state <= state+1;
    end

  8'h01:
    begin
      // End reset
      disp_reset_b <= 1'b1;
      state <= state+1;
    end

  8'h02:
    begin
      // Initialize dot register (set all dots to zero)
      disp_ce_b <= 1'b0;
      disp_data_out <= 1'b0; // dot_index[0];
      if (dot_index == 639)
        state <= state+1;
      else
        dot_index <= dot_index+1;
    end

  8'h03:
    begin
      // Latch dot data
      disp_ce_b <= 1'b1;
      dot_index <= 31;              // re-purpose to init ctrl reg
      state <= state+1;
    end

  8'h04:
    begin
      // Setup the control register
      disp_rs <= 1'b1; // Select the control register
      disp_ce_b <= 1'b0;
      disp_data_out <= control[31];
      control <= {control[30:0], 1'b0};     // shift left
      if (dot_index == 0)
        state <= state+1;
      else
        dot_index <= dot_index-1;
    end

  8'h05:
    begin
      // Latch the control register data / dot data
      disp_ce_b <= 1'b1;
      dot_index <= 39;              // init for single char
      char_index <= 15;             // start with MS char
      data <= data_in;
      state <= state+1;
```

```
            end

          8'h06:
            begin
               // Load the user's dot data into the dot reg, char by char
               disp_rs <= 1'b0;                    // Select the dot register
               disp_ce_b <= 1'b0;
               disp_data_out <= dots[dot_index]; // dot data from msb
               if (dot_index == 0)
                 if (char_index == 0)
                   state <= 5;                     // all done, latch data
                 else
                   begin
                      char_index <= char_index - 1; // goto next char
                      data <= data_in;
                      dot_index <= 39;
                   end
               else
                 dot_index <= dot_index-1;  // else loop thru all dots
            end

        endcase // casex(state)
    end

always @(data or char_index)
  case (char_index)
    4'h0:            nibble <= data[3:0];
    4'h1:            nibble <= data[7:4];
    4'h2:            nibble <= data[11:8];
    4'h3:            nibble <= data[15:12];
    4'h4:            nibble <= data[19:16];
    4'h5:            nibble <= data[23:20];
    4'h6:            nibble <= data[27:24];
    4'h7:            nibble <= data[31:28];
    4'h8:            nibble <= data[35:32];
    4'h9:            nibble <= data[39:36];
    4'hA:            nibble <= data[43:40];
    4'hB:            nibble <= data[47:44];
    4'hC:            nibble <= data[51:48];
    4'hD:            nibble <= data[55:52];
    4'hE:            nibble <= data[59:56];
    4'hF:            nibble <= data[63:60];
  endcase

always @(nibble)
  case (nibble)
    4'h0: dots <= 40'b00111110_01010001_01001001_01000101_00111110;
    4'h1: dots <= 40'b00000000_01000010_01111111_01000000_00000000;
    4'h2: dots <= 40'b01100010_01010001_01001001_01001001_01000110;
    4'h3: dots <= 40'b00100010_01000001_01001001_01001001_00110110;
    4'h4: dots <= 40'b00011000_00010100_00010010_01111111_00010000;
```

```
        4'h5: dots <= 40'b00100111_01000101_01000101_01000101_00111001;
        4'h6: dots <= 40'b00111100_01001010_01001001_01001001_00110000;
        4'h7: dots <= 40'b00000001_01110001_00001001_00000101_00000011;
        4'h8: dots <= 40'b00110110_01001001_01001001_01001001_00110110;
        4'h9: dots <= 40'b00000110_01001001_01001001_00101001_00011110;
        4'hA: dots <= 40'b01111110_00001001_00001001_00001001_01111110;
        4'hB: dots <= 40'b01111111_01001001_01001001_01001001_00110110;
        4'hC: dots <= 40'b00111110_01000001_01000001_01000001_00100010;
        4'hD: dots <= 40'b01111111_01000001_01000001_01000001_00111110;
        4'hE: dots <= 40'b01111111_01001001_01001001_01001001_01000001;
        4'hF: dots <= 40'b01111111_00001001_00001001_00001001_00000001;
      endcase

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    20:50:52 04/03/06
// Design Name:
// Module Name:    lsfr
// Project Name:
// Target Device:
// Tool versions:
// Description:  Generates pseudorandom numbers from the seed value
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module lfsr(clk, reset, seed, enable, out);
input clk;
input enable;
input reset;
input [15:0] seed;
output [15:0] out;

reg[15:0] out;
wire feedback, x1513, xx12;

assign x1513 = (out[15] ^ out[13]);
assign xx12  = (out[12] ^ x1513);
assign feedback = (out[10] ^ xx12);

always @(posedge clk) begin
        if (reset) begin
                out[15:0] <= seed;
        end else if (enable) begin
                out[15:0] <= {out[14:0], feedback};
        end
end

endmodule
```

```
module timer(pixel_clock, reset, time_in, mode, minutes, seconds, enable);
    input pixel_clock;
    input reset;
    input [1:0] mode;
         input [7:0] time_in;
    output [1:0] minutes;
    output [5:0] seconds;
         output enable;

         reg [1:0] minutes;
         reg [5:0] seconds;
         reg [7:0] counter;

         wire enable;
         wire time_reset;
         assign time_reset = (time_in==180);

         counter ct(.clk(pixel_clock),.reset(Reset_Sync),.enable(enable));

         always @(posedge pixel_clock) begin
/*               if ((!reset)||time_reset||(mode!=1)) begin
                         //time_in<=180;
                         //last_time_in<=0;
                end
                else if (time_in==0) counter<=0;
                else if (enable && (last_time_in!=counter)) begin
                         last_time_in <= time_in;
                         counter<=(counter-1);
                end*/

                /*if ((!reset)||time_reset) minutes<=3;
                else if (time_in>120) minutes<=2;
                else if (time_in>60) minutes<=1;
                else minutes<=0;

                if ((!reset)||time_reset) seconds<=0;
                else if ((seconds==0)&&(minutes!=0)) seconds<=59;
                else if (seconds==0) seconds<=0;
                else if (minutes==2) seconds<=time_in-120;
                else if (minutes==1) seconds<=time_in-60;        */

                if(!reset||time_reset) begin
                         minutes<=3;
                         seconds<=0;
                end else if(time_in>120) begin
                         minutes<=2;
                         seconds<=time_in - 120;
                end else if(time_in>60) begin
                         minutes<=1;
                         seconds<=time_in - 60;
                end else begin
```

```
                    minutes<=0;
                    seconds<=time_in;
                end

        end



endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    15:45:33 04/23/06
// Design Name:
// Module Name:    timeleftfsm
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module timeleftfsm(clk, reset, start, busy, time_left, next_time_left);
input   clk, reset, start;
output busy;
input[7:0] time_left;
output[7:0] next_time_left;

reg[6:0] counter;
reg[7:0] next_time_left;
reg state;
reg busy;

//parameter FRAMERATE = 60;
parameter FRAMERATE = 30;//twice as fast

parameter IDLE  = 1'b0;
parameter START = 1'b1;

always @(posedge clk) begin
        next_time_left <= time_left;
        if(reset) begin
                counter <= FRAMERATE;
                busy <= 0;
        end else begin
                case(state)
                        IDLE: begin
                                if(start) begin
                                        state <= START;
                                        busy <= 1;
                                end else begin
                                        state <= IDLE;
                                        busy <= 0;
```

```
                        end
                    end
                    START: begin
                        if(counter == 0) begin
                            next_time_left <= time_left - 1;
                            counter <= FRAMERATE;
                        end else begin
                            counter <= counter - 1;
                        end
                        busy <=0;
                        state <=IDLE;
                    end
                    default: state <=IDLE;
                endcase
        end
end


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    21:50:16 04/26/06
// Design Name:
// Module Name:    title_fsm
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module titlefsm(clk, reset, start, busy,
        l_hand_x, r_hand_x, l_hand_y, r_hand_y, l_hand_punch, r_hand_punch,
        next_l_hand_x, next_r_hand_x,   next_l_hand_y, next_r_hand_y,
        next_mode, game_type);

input clk, reset, start;
input[9:0] l_hand_x, r_hand_x;
input      l_hand_y, r_hand_y;
input[1:0] l_hand_punch, r_hand_punch;

output busy;
output[9:0]     next_l_hand_x, next_r_hand_x;
output      next_l_hand_y, next_r_hand_y;
output[1:0] next_mode, game_type;

reg busy;
reg[9:0]        next_l_hand_x, next_r_hand_x;
reg      next_l_hand_y, next_r_hand_y;
reg[1:0] next_mode, game_type;

reg[1:0] state;

parameter IDLE = 1'b0;
parameter ACTIVE= 1'b1;

always @(posedge clk) begin
        if(reset) begin
                busy                            <=      0;
                next_l_hand_x   <= 300;
                next_r_hand_x   <= 340;
                next_l_hand_y   <= 0;
```

```
                next_r_hand_y    <= 0;
                next_mode                <= 2'b00;
                game_type                <= 0;
        end else begin
                case(state)
                IDLE: begin
                        if(start) begin
                                busy<=1;
                                state<=ACTIVE;
                        end else begin
                                busy<=0;
                                state<=IDLE;
                        end
                end
                ACTIVE: begin
                        next_l_hand_x    <= l_hand_x;
                        next_r_hand_x    <= r_hand_x;
                        next_l_hand_y    <= l_hand_y;
                        next_r_hand_y    <= r_hand_y;
                        next_mode                <= 2'b00;
                        game_type                <= 0;

                   if(l_hand_punch > 0 || r_hand_punch > 0) begin
                      next_mode <= 2'b01;
                      game_type<= 2'b00;
                      if(r_hand_punch > 0 && r_hand_x < 213 ||
                         l_hand_punch > 0 && l_hand_x < 213)
                        game_type<= 2'b00;
                      else if(r_hand_punch > 0 && r_hand_x > 414 ||
                              l_hand_punch > 0 && l_hand_x > 414)
                        game_type<= 2'b10;
                      else
                        game_type<= 2'b01;
                   end
                   busy<=0;
                        state<=IDLE;
                end
                default: state<=IDLE;
                endcase //mstate
        end
end

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module for Lab 4 (Spring 2006)
//
//
// Created: March 13, 2006
// Author: Nathan Ickes
//
////////////////////////////////////////////////////////////////////////////////

module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
               ac97_bit_clock,

               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,

               tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
               tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
               tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

               tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
               tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
               tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
               tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

               ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
               ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

               ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
               ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

               clock_feedback_out, clock_feedback_in,

               flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
               flash_reset_b, flash_sts, flash_byte_b,

               rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

               mouse_clock, mouse_data, keyboard_clock, keyboard_data,

               clock_27mhz, clock1, clock2,

               disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
               disp_reset_b, disp_data_in,

               button0, button1, button2, button3, button_enter, button_right,
               button_left, button_down, button_up,

               switch,
```

```
            led,

            user1, user2, user3, user4,

            daughtercard,

            systemace_data, systemace_address, systemace_ce_b,
            systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

            analyzer1_data, analyzer1_clock,
            analyzer2_data, analyzer2_clock,
            analyzer3_data, analyzer3_clock,
            analyzer4_data, analyzer4_clock);

output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
input  ac97_bit_clock, ac97_sdata_in;

output [7:0] vga_out_red, vga_out_green, vga_out_blue;
output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
       vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb;
output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
       tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
       tv_out_subcar_reset;

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
       tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
       tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

inout  [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;

input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input  flash_sts;
```

```
output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input  disp_data_in;
output  disp_data_out;

input  button0, button1, button2, button3, button_enter, button_right,
       button_left, button_down, button_up;
input  [7:0] switch;
output [7:0] led;

inout [31:0] user1, user2, user3, user4;

inout [43:0] daughtercard;

inout  [15:0] systemace_data;
output [6:0]  systemace_address;
output systemace_ce_b, systemace_we_b, systemace_oe_b;
input  systemace_irq, systemace_mpbrdy;

output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
              analyzer4_data;
output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

////////////////////////////////////////////////////////////////////////
//
// I/O Assignments
//
////////////////////////////////////////////////////////////////////////

// Audio Input and Output
assign beep= 1'b0;
assign audio_reset_b = 1'b0;
assign ac97_synch = 1'b0;
assign ac97_sdata_out = 1'b0;

    // Video Output
assign tv_out_ycrcb = 10'h0;
assign tv_out_reset_b = 1'b0;
assign tv_out_clock = 1'b0;
assign tv_out_i2c_clock = 1'b0;
assign tv_out_i2c_data = 1'b0;
assign tv_out_pal_ntsc = 1'b0;
assign tv_out_hsync_b = 1'b1;
assign tv_out_vsync_b = 1'b1;
assign tv_out_blank_b = 1'b1;
```

```
   assign tv_out_subcar_reset = 1'b0;

   // Video Input
   //assign tv_in_i2c_clock = 1'b0;
   assign tv_in_fifo_read = 1'b1;
   assign tv_in_fifo_clock = 1'b0;
   assign tv_in_iso = 1'b1;
   //assign tv_in_reset_b = 1'b0;
   assign tv_in_clock = clock_27mhz;//1'b0;
   //assign tv_in_i2c_data = 1'bZ;
   // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
   // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

   // SRAMs
/* change lines below to enable ZBT RAM bank0 */

/*
   assign ram0_data = 36'hZ;
   assign ram0_address = 19'h0;
   assign ram0_clk = 1'b0;
   assign ram0_we_b = 1'b1;
   assign ram0_cen_b = 1'b0;     // clock enable
*/

/* enable RAM pins */

   assign ram0_ce_b = 1'b0;
   assign ram0_oe_b = 1'b0;
   assign ram0_adv_ld = 1'b0;
   assign ram0_bwe_b = 4'h0;

/**********/
   assign ram1_data = 36'hZ;
   assign ram1_address = 19'h0;
   assign ram1_adv_ld = 1'b0;
   assign ram1_clk = 1'b0;
   assign ram1_cen_b = 1'b1;
   assign ram1_ce_b = 1'b1;
   assign ram1_oe_b = 1'b1;
   assign ram1_we_b = 1'b1;
   assign ram1_bwe_b = 4'hF;
   assign clock_feedback_out = 1'b0;

   // Flash ROM
   assign flash_data = 16'hZ;
   assign flash_address = 24'h0;
   assign flash_ce_b = 1'b1;
   assign flash_oe_b = 1'b1;
   assign flash_we_b = 1'b1;
   assign flash_reset_b = 1'b0;
   assign flash_byte_b = 1'b1;
```

```
// RS-232 Interface
assign rs232_txd = 1'b1;
assign rs232_rts = 1'b1;

// LED Displays
assign disp_blank = 1'b1;
assign disp_clock = 1'b0;
assign disp_rs = 1'b0;
assign disp_ce_b = 1'b1;
assign disp_reset_b = 1'b0;
assign disp_data_out = 1'b0;

// Buttons, Switches, and Individual LEDs
assign led = 8'hFF;

// User I/Os
assign user1 = 32'hZ;
assign user2 = 32'hZ;
//assign user3 = 32'hZ;
assign user4[31:18] = 16'hZ;

// Daughtercard Connectors
assign daughtercard = 44'hZ;

// SystemACE Microprocessor Port
assign systemace_data = 16'hZ;
assign systemace_address = 7'h0;
assign systemace_ce_b = 1'b1;
assign systemace_we_b = 1'b1;
assign systemace_oe_b = 1'b1;

// Logic Analyzer
assign analyzer1_data = 16'h0;
assign analyzer1_clock = 1'b1;
assign analyzer2_data = 16'h0;
assign analyzer2_clock = 1'b1;
assign analyzer3_data = 16'h0;
assign analyzer3_clock = 1'b1;
assign analyzer4_data = 16'h0;
assign analyzer4_clock = 1'b1;

////////////////////////////////////////////////////////////////////////
//
// Lab 4 Components
//
////////////////////////////////////////////////////////////////////////

//
// Generate a 31.5MHz pixel clock from clock_27mhz
//
```

```
   wire pclk, pixel_clock;
   DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
   // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 15
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 14
        // synthesis attribute CLKIN_PERIOD of pixel_clock_dcm is 37
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
   BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

   //
   // VGA output signals
   //

   // Inverting the clock to the DAC provides half a clock period for signals
   // to propagate from the FPGA to the DAC.
   assign vga_out_pixel_clock = ~pixel_clock;

        wire reset_sync;
        debounce debounceReset(1'b0, pixel_clock, button_enter, reset_sync);

        wire[9:0] lpx, rpx;
        wire          lpy, rpy;
        wire[1:0] lpp, rpp;

        //assign led = 8'b11111111;

        //display_16hex alphadisp(~reset_sync, pixel_clock, hex,
        //      disp_blank, disp_clock, disp_rs, disp_ce_b,
        //      disp_reset_b, disp_data_out);


///START oF UZOMA's CODE
        wire clk;
        assign clk = pixel_clock;
        wire [9:0] hcount;
   wire [9:0]  vcount;
        wire start;

        //vgacontroller vga(clk, ~reset_sync, hsync_b, vsync_b, vga_out_sync_b,
        //vga_out_blank_b, hcount, vcount, start);

        //delay dhsync(clk,hsync_b,vga_out_hsync);
        //delay dvsync(clk,vsync_b,vga_out_vsync);

        //vga vga1(clk,hcount,vcount,hsync,vsync,blank);
   // wire up to ZBT ram



   wire [35:0] vram_write_data;
   wire [35:0] vram_read_data;
```

```
wire [18:0] vram_addr;
wire        vram_we;

zbt_6111 zbt1(clk, 1'b1, vram_we, vram_addr,
              vram_write_data, vram_read_data,
              ram0_clk, ram0_we_b, ram0_address, ram0_data, ram0_cen_b);

// generate pixel value from reading ZBT memory
wire [17:0]  vr_pixel;
wire [18:0]  vram_addr1;

vram_display vd1(~reset_sync,clk,hcount,vcount,vr_pixel,
                 vram_addr1,vram_read_data);

// ADV7185 NTSC decoder interface code
// adv7185 initialization module
adv7185init adv7185(.reset(~reset_sync), .clock_27mhz(clock_27mhz),
                    .source(1'b0), .tv_in_reset_b(tv_in_reset_b),
                    .tv_in_i2c_clock(tv_in_i2c_clock),
                    .tv_in_i2c_data(tv_in_i2c_data));

wire [29:0] ycrcb;    // video data (luminance, chrominance)
wire [2:0] fvh;       // sync for field, vertical, horizontal
wire       dv;        // data valid

ntsc_decode decode (.clk(tv_in_line_clock1), .reset(~reset_sync),
                    .tv_in_ycrcb(tv_in_ycrcb[19:10]),
                    .ycrcb(ycrcb), .f(fvh[2]),
                    .v(fvh[1]), .h(fvh[0]), .data_valid(dv));

// code to write NTSC data to video memory

wire [18:0] ntsc_addr;
wire [35:0] ntsc_data;
wire        ntsc_we;

ntsc_to_zbt n2z (clk, tv_in_line_clock1, fvh, dv, ycrcb,
                 ntsc_addr, ntsc_data, ntsc_we);

     wire    my_we = hcount[1:0] == 2'd2;
//wire my_we = ntsc_we;
     assign  vram_addr = my_we ? ntsc_addr : vram_addr1;
assign       vram_we = my_we;
assign       vram_write_data = ntsc_data;

//wire       bl,hs,vs;
     //delayN dn1(clk,hsync,hs);      // delay by 3 cycles to sync with ZBT read
//delayN dn2(clk,vsync,vs);
//delayN dn3(clk,blank,bl);

wire valid_pixel = ((hcount>9) & (hcount<628)); //639-11
```

```
    //wire valid_pixel = 1;
    reg[9:0] Y,Cr,Cb;

    always @(posedge clk)
  begin
          Y <= {vr_pixel[17:10],2'b0};
      Cr <={vr_pixel[9:5],5'b0};
      Cb <={vr_pixel[4:0],5'b0};
  end

    wire[7:0] tR,tG,tB;
    YCrCb2RGB convcolor(tR,tG,tB, clk, ~reset_sync, Y, Cr, Cb);

    //reg[7:0] L_thres, R_thres, G_thres, B_thres;
/*  debounce up1(reset, clk, ~button_up, uplevel);
    debounce down1(reset, clk, ~button_down, downlevel);

    reg upold, downold;
    always @ (posedge clk)
          begin
                upold <= uplevel;
                downold <= downlevel;
          end

    wire up = uplevel & !upold;
    wire down = downlevel & !downold;

    wire sw0,sw1,sw3,sw7;
    debounce s0(reset, clk, switch[0], sw0);
    debounce s1(reset, clk, switch[1], sw1);
    debounce s3(reset, clk, switch[3], sw3);
    debounce s7(reset, clk, switch[7], sw7);        */

    parameter L_thres = 8'b10000111;
    parameter R_thres = 8'b11111011;
    parameter G_thres = 8'b01111111;
    parameter B_thres = 8'b11111111;

    //wire[9:0] vcen, hcen;
    wire validcolor;
    //find_center fc(clk, reset, vcount, hcount, validcolor, vcen, hcen);

    wire[9:0] vcenl, hcenl, vcenr, hcenr;
    findlights fl(clk, ~reset_sync, vcount, hcount, validcolor, vcenl, hcenl,
                vcenr, hcenr);

    wire [23:0] color, color2, colorleft,colorright;
    inrectangle leftglove(vcount, hcount, hcenl, vcenl, colorleft);
    inrectangle rightglove(vcount, hcount, hcenr, vcenr, colorright);

    //inrectangle leftglove(vcount, hcount, 9'd200, 9'd150, colorleft);
```

```
    //inrectangle rightglove(vcount, hcount, 9'd300, 9'd300, colorright);

    defparam rightglove.TRUE_COLOR = 24'h00FF00;

    assign color2 = (colorleft|colorright);

    assign color = (color2==24'd0) ? {tR,tG,tB} : color2;

    assign validcolor = (Y[9:2]>=8'b10000111) &
                                                    (color[23:16]>=R_thres) &
                                                    (color[15:8]>=G_thres) &
                                                    (color[7:0]<B_thres);


    wire sclk, CONVST;
    assign user3    = {2'hZ, sclk, CONVST, 26'hZ, sclk, CONVST};
    wire datainr = user3[2];
    wire RFSr = user3[3];
    wire datainl = user3[30];
    wire RFSl = user3[31];

    wire[1:0] punchlevell, punchlevelr;

    wire [7:0] dataoutl;
    wire validl;

    wire [9:0] glo_x, gro_x;
    wire glo_y, gro_y;

punchdetector pd(clock_27mhz, clk, sclk, ~reset_sync, CONVST, datainl, RFSl,
                 datainr, RFSr,
                 hcenl, vcenl, hcenr, vcenr,
                 glo_x, glo_y, gro_x, gro_y, punchlevell, punchlevelr);
//end OF UZOMA'S CODE









    wire[2:0] index_in;
    wire[9:0] data_in;
    wire enable_in;
    wire[2:0] index_out;
    wire[9:0] data_out;
    wire enable_out;
```

```
wire[9:0] player_l_hand_x, player_r_hand_x;
wire      player_l_hand_y, player_r_hand_y;
wire[1:0] player_l_hand_punch, player_r_hand_punch;
wire[9:0] opponent_head_x;
wire[4:0] opponent_picture,
        player_life, opponent_life, player_energy, opponent_energy;
wire[7:0] time_left;
wire[1:0] mode, round, game_type;
wire[2:0] state;

assign start = (hcount==10'd639 && vcount==10'd479);

//game types
parameter ONEPLAYER     = 2'd0;
parameter TWOPLAYERM    = 2'd1;
parameter TWOPLAYERS    = 2'd2;

//wire  m_index s_index data    m_enable        s_enable
//pins  17:15           14:12           11:2    1                       0

wire sbo;//slave back out on collision
assign sbo = (game_type==TWOPLAYERM || (game_type==TWOPLAYERS && ~enable_in));

assign enable_in                = (game_type==TWOPLAYERM) ? user4[0] : user4[1];
assign index_in         = (game_type==TWOPLAYERM) ? user4[14:12] : user4[17:15];

assign user4[17:15]     = (game_type==TWOPLAYERM) ? index_out[2:0] : 3'hZ;
assign user4[14:12]     = (game_type==TWOPLAYERM) ? 3'hZ : index_out[2:0];
assign user4[1]         = (game_type==TWOPLAYERM) ? enable_out : 1'hZ;
assign user4[0]         = (game_type==TWOPLAYERM) ? 1'hZ : enable_out;

//assign data_in[9:0]   = user4[11:2];// : 10'b0;
assign data_in[9:0]     = enable_in ? user4[11:2] : 10'b0;
assign user4[11:2]      = enable_out && sbo ? data_out[9:0] : 10'hZ;

controlunit cu(clk, ~reset_sync, start,
10'd640-gro_x, gro_y, punchlevelr,
10'd640-glo_x, glo_y, punchlevell,

player_l_hand_x, player_l_hand_y, player_l_hand_punch,
player_r_hand_x, player_r_hand_y, player_r_hand_punch,

opponent_head_x,
opponent_picture,

player_life, opponent_life,
player_energy, opponent_energy,
time_left, mode, round, game_type, state,

index_in,  data_in,  enable_in,
```

```
        index_out, data_out, enable_out
        );

        wire hsync, vsync, lhstatus, rhstatus;
        reg hsync_b, vsync_b;

        assign lhstatus = |(player_l_hand_punch);
        assign rhstatus = |(player_r_hand_punch);

        top top1(
                .pixel_clock(pixel_clock),
                .reset_sync(reset_sync),
                .f1_lhand_x(player_l_hand_x),
                .f1_rhand_x(player_r_hand_x),
                .f1_lhand_y(player_l_hand_y),
                .f1_rhand_y(player_r_hand_y),
                .f1_lh_status(lhstatus),
                .f1_rh_status(rhstatus),
                .f2_pic_x(opponent_head_x),
                .f2_pic_y(10'd250),
                .f2_status(opponent_picture),
                .hsync(hsync),
                .vsync(vsync),
                .sync_b(vga_out_sync_b),
                .blank_b(vga_out_blank_b),
                .red(vga_out_red),
                .green(vga_out_green),
                .blue(vga_out_blue),
                .f1_lifebar(player_life),
                .f1_energybar(player_energy),
                .f2_lifebar(opponent_life),
                .f2_energybar(opponent_energy),
                .mode(mode),
                .round(round),
                .pixel_count(hcount),
                .line_count(vcount),
                .time_in(time_left));


    always @(posedge pixel_clock) begin
                        hsync_b <= ~hsync;
                        vsync_b <= ~vsync;
     end

          assign vga_out_vsync = vsync_b;
          assign vga_out_hsync = hsync_b;

endmodule
```

C Imaging Source Codes

```
module boxing_ring(pixel_clock, x, y, red, green, blue);
   input pixel_clock;
   input [9:0] x, y;
   output [7:0] red, green, blue;

   parameter    PIXELS = 800;
   parameter    LINES = 525;
   parameter    FLOOR_TOP_LX = 10'd100;
   parameter    FLOOR_TOP_RX = 10'd540;
   parameter    MAX_X = 10'd640;
   parameter    FLOOR_TOP_Y = 10'd320;
   parameter    MAX_Y = 10'd420;
   parameter    SPACE = 10'd25;
   parameter    WIDTH = 10'd5;
   parameter    POLE_WIDTH = 10'd10;
   parameter    POLE_TOP_Y = FLOOR_TOP_Y-3*SPACE-5*WIDTH;
   parameter    POLE_BOTTOM_Y = FLOOR_TOP_Y+WIDTH;
   parameter    POLE_LX = FLOOR_TOP_LX-10'd5;
   parameter    POLE_RX = FLOOR_TOP_RX-10'd5;
   parameter    FLOOR_COLOR = 24'hffffff;
   parameter    ROPE1_COLOR = 24'h0000ff;
   parameter    ROPE2_COLOR = 24'hff0000;
   parameter    ROPE3_COLOR = 24'hffffff;
   parameter    POLE_COLOR = 24'hff0000;
   parameter    BLACK = 24'h000000;

   wire [23:0]  next_RGB;
   wire [9:0]   next_x, next_y;
   wire         out_of_bounds, horizontal_rope1, horizontal_rope2, horizontal_rope3,
                diagonal_rope1, diagonal_rope2,
                diagonal_rope3, pole;
   wire [7:0]   red, green, blue;

   reg [23:0]   RGB;

   always @(posedge pixel_clock) begin
      //assigns next state
      RGB<=next_RGB;
   end

   //computes whether next pixel position is within boundaries of boxing ring
   assign out_of_bounds = (((next_x>=0)&&(next_x<=FLOOR_TOP_LX)&&
                            (next_y<(MAX_Y-next_x)))||
                           ((next_x>=FLOOR_TOP_RX)&&(next_x<=MAX_X)&&
                            (next_y<(FLOOR_TOP_Y+next_x-FLOOR_TOP_RX)))||
                           ((next_x>=FLOOR_TOP_LX)&&(next_x<=FLOOR_TOP_RX)&&
                            (next_y<FLOOR_TOP_Y)));
   assign horizontal_rope1 = (((next_y>=(FLOOR_TOP_Y-SPACE-WIDTH))&&
                               (next_y<=(FLOOR_TOP_Y-SPACE)))&&
```

```
                                (next_x>=FLOOR_TOP_LX)&&(next_x<=FLOOR_TOP_RX));
    assign horizontal_rope2 = (((next_y>=(FLOOR_TOP_Y-2*SPACE-2*WIDTH))&&
                                 (next_y<=(FLOOR_TOP_Y-2*SPACE-WIDTH)))&&
                                (next_x>=FLOOR_TOP_LX)&&(next_x<=FLOOR_TOP_RX));
    assign horizontal_rope3 = (((next_y>=(FLOOR_TOP_Y-3*SPACE-3*WIDTH))&&
                                 (next_y<=(FLOOR_TOP_Y-3*SPACE-2*WIDTH)))&&
                                (next_x>=FLOOR_TOP_LX)&&(next_x<=FLOOR_TOP_RX));
    assign diagonal_rope1 = (((next_x>=0)&&(next_x<=FLOOR_TOP_LX)&&
                               (next_y>=(MAX_Y-next_x-SPACE-WIDTH))&&
                               (next_y<=(MAX_Y-next_x-SPACE)))||
                              ((next_x>=FLOOR_TOP_RX)&&(next_x<=MAX_X)&&
                               (next_y>=(FLOOR_TOP_Y+next_x-FLOOR_TOP_RX-SPACE-WIDTH))
                              &&(next_y<=(FLOOR_TOP_Y+next_x-FLOOR_TOP_RX-SPACE))));
    assign diagonal_rope2 = (((next_x>=0)&&(next_x<=FLOOR_TOP_LX)&&
                               (next_y>=(MAX_Y-next_x-2*SPACE-2*WIDTH))&&
                               (next_y<=(MAX_Y-next_x-2*SPACE-WIDTH)))||
                              ((next_x>=FLOOR_TOP_RX)&&(next_x<=MAX_X)&&
                               (next_y>=(FLOOR_TOP_Y+next_x-FLOOR_TOP_RX-2*SPACE-2*WIDTH))
                             &&(next_y<=(FLOOR_TOP_Y+next_x-FLOOR_TOP_RX-2*SPACE-WIDTH))));
    assign diagonal_rope3 = (((next_x>=0)&&(next_x<=FLOOR_TOP_LX)&&
                               (next_y>=(MAX_Y-next_x-3*SPACE-3*WIDTH))&&
                               (next_y<=(MAX_Y-next_x-3*SPACE-2*WIDTH)))||
                              ((next_x>=FLOOR_TOP_RX)&&(next_x<=MAX_X)&&
                               (next_y>=(FLOOR_TOP_Y+next_x-FLOOR_TOP_RX-3*SPACE-3*WIDTH))
                            &&(next_y<=(FLOOR_TOP_Y+next_x-FLOOR_TOP_RX-3*SPACE-2*WIDTH))));
    assign pole = (((next_y<=POLE_BOTTOM_Y)&&(next_y>=POLE_TOP_Y))&&
                   (((next_x>=POLE_LX) &&(next_x<=(POLE_LX+POLE_WIDTH)))||
                    ((next_x>=POLE_RX)&&(next_x<=(POLE_RX+POLE_WIDTH)))));

    //computes next pixel position
    assign next_x = (x == (PIXELS-1)) ? 10'd0 : x + 1'b1;
    assign next_y = (x == (PIXELS-1)) ? (y == LINES-1) ? 10'd0 : y + 1'b1 : y;

    //computes next state
    assign next_RGB = (pole) ? POLE_COLOR :
                      (horizontal_rope1||diagonal_rope1) ? ROPE1_COLOR :
                      (horizontal_rope2||diagonal_rope2) ? ROPE2_COLOR :
                      (horizontal_rope3||diagonal_rope3) ? ROPE3_COLOR :
                      (!out_of_bounds) ? FLOOR_COLOR : BLACK;

    assign red = RGB[23:16];
    assign green = RGB[15:8];
    assign blue = RGB[7:0];


endmodule
```

```
// This module provides control signals to the ADV7125.
// The resolution is 640x480 and the pixel frequency
// is about 25MHz
// hsync is active low: high for 640 pixels of active video,
// high for 16 pixels of front porch,
// low for 96 pixels of hsync,
// high for 48 pixels of back porch
// vsync is active low: high for 480 lines of active video,
// high for 11 lines of front porch,
// low for 2 lines of vsync,
// high for 31 lines of back porch
module vga (pixel_clock, reset, hsync, vsync, sync_b,
      blank_b, pixel_count, line_count, frame_pulse);
   input pixel_clock; // 25.175 MHz pixel clock
   input reset; // system reset
   output hsync; // horizontal sync
   output vsync; // vertical sync
   output sync_b; // hardwired to Vdd
   output blank_b; // composite blank
   output [9:0] pixel_count; // number of the current pixel
   output [9:0] line_count; // number of the current line
        output frame_pulse;

   // 640x480 60Hz parameters
   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 = 31;

   // current pixel count
   reg [9:0] pixel_count = 10'b0;
   reg [9:0] line_count = 10'b0;

   // registered outputs
   //reg hsync = 1'b1;
   //reg vsync = 1'b1;
   //reg blank_b = 1'b1;
   wire sync_b; // connected to Vdd
        wire frame_pulse;
   wire pixel_clock;
   wire [9:0] next_pixel_count;
   wire [9:0] next_line_count;

   reg hsync = 1'b1;
        reg vsync = 1'b1;
```

```
      reg blank_b = 1'b1;

      always @(posedge pixel_clock) begin
    if (!reset) begin
       pixel_count <= 10'b0;
       line_count <= 10'b0;
       hsync <= 1'b1;
       vsync <= 1'b1;
       blank_b <= 1'b1;
    end
    else begin
       pixel_count <= next_pixel_count;
       line_count <= next_line_count;
       hsync <=
          (next_pixel_count < HACTIVE_VIDEO + HFRONT_PORCH) |
          (next_pixel_count >= HACTIVE_VIDEO+HFRONT_PORCH+
          HSYNC_PERIOD);
       vsync <=
          (next_line_count < VACTIVE_VIDEO+VFRONT_PORCH) |
          (next_line_count >= VACTIVE_VIDEO+VFRONT_PORCH+
          VSYNC_PERIOD);
    // this is the and of hblank and vblank
       blank_b <=
          (next_pixel_count < HACTIVE_VIDEO) &
          (next_line_count < VACTIVE_VIDEO);
    end
  end

  // next state is computed with combinational logic
  assign next_pixel_count = (pixel_count == PIXELS-1) ?
     10'h000 : pixel_count + 1'b1;
  assign next_line_count = (pixel_count == PIXELS-1) ?
     (line_count == LINES-1) ? 10'h000 :
     line_count + 1'b1 : line_count;
      assign frame_pulse = ((pixel_count == HACTIVE_VIDEO-1) &&
                            (line_count == VACTIVE_VIDEO-1));

  // since we are providing hsync and vsync to the display, we
  // can hardwire composite sync to Vdd.
  assign sync_b = 1'b1;

endmodule
```

```
module Display_Field(pixel_clock, reset_sync, pixel_count, line_count, f1_lifebar,
                     f1_energybar, f1_lhand_x, f1_rhand_x, f1_lhand_y, f1_rhand_y,
                     f1_lh_status, f2_lh_status, f2_lifebar, f2_energybar, f2_status,
                     red, green, blue, frame_pulse, mode, round, time_in);

   input pixel_clock, reset_sync, frame_pulse;
   input [1:0] mode;
   input [3:0] round;
   input [9:0] line_count, pixel_count;
   input [4:0] f1_lifebar, f2_lifebar, f1_energybar, f2_energybar;
   input [9:0] f1_lhand_x, f1_rhand_x, f2_pic_x, f2_pic_y;
   input [4:0] f2_status;
   input       f1_lhand_y, f1_rhand_y;
   input       f1_lh_status, f1_rh_status;
   input [7:0] time_in;
   output [7:0] red;
   output [7:0] green;
   output [7:0] blue;

    //color parameters for RGB
   parameter    WHITE = 24'hffffff;
   parameter    BLUE = 24'h0000ff;
   parameter    LT_BLUE = 24'h00007f;
   parameter    RED = 24'hff0000;
   parameter    LT_RED = 24'h7f0000;

   //parameters for widths, heights, and starting points
   parameter    PLAYER2_PIXEL = 10'd380;
   parameter    PLAYER1_PIXEL = 10'd4;
   parameter    TIME_PIXEL = 10'd277;
   parameter    COLON = 10'd5;
   parameter    BAR_HEIGHT = 10'd10;
   parameter    TOP = 10'd15;
   parameter    ONE_PLAYER = 10'd0;
   parameter    TWO_PLAYER = 10'd427;
   parameter    SCREEN_HEIGHT = 10'd480;
   parameter    SCREEN_WIDTH = 10'd640;
   parameter    MODE_WIDTH = 10'd213;

   //defining colors of instances of the rectangle module
   defparam     f1e.COLOR = BLUE;
   defparam     f2e.COLOR = BLUE;
   defparam     f1l.COLOR = RED;
   defparam     f2l.COLOR = RED;
   defparam     f2l_fade.COLOR = LT_RED;
   defparam     f1l_fade.COLOR = LT_RED;
   defparam     f2e_fade.COLOR = LT_BLUE;
   defparam     f1e_fade.COLOR = LT_BLUE;
   defparam     top_colon.COLOR = WHITE;
   defparam     bottom_colon.COLOR = WHITE;
   defparam     one_player.COLOR = 24'h0f820d;
```

```
defparam      two_player.COLOR = 24'h40d040;
defparam      card.COLOR = WHITE;

//defining colors, widths, and heights of instances of the number module
defparam      round_num.HDIM =         10'd59;
defparam      round_num.VDIM =         10'd79;
defparam      round_num.WIDTH = 10'd10;
defparam      round_num.SPACE = 10'd24;
defparam      round_num.COLOR = 24'h000000;
defparam      round_num.BACKGROUND = WHITE;
defparam      one.HDIM =       10'd39;
defparam      one.VDIM =       10'd59;
defparam      one.WIDTH = 10'd6;
defparam      one.SPACE = 10'd20;
defparam      two.HDIM =       10'd39;
defparam      two.VDIM =       10'd59;
defparam      two.WIDTH = 10'd6;
defparam      two.SPACE = 10'd20;



reg [7:0]     f1l_pixels, f2l_pixels, f1e_pixels, f2e_pixels;

wire          enable;
wire [1:0]    minutes;
wire [5:0]    seconds;
wire [3:0]    sec_ones, sec_tens;
wire [7:0]    f1e_width, f2e_width;
wire [7:0]    next_f1l_pixels, next_f2l_pixels, next_f1e_pixels,
              next_f2e_pixels;
wire [7:0]    red, blue, green;
wire [7:0]    red_f1l, red_f1e, red_f2l, red_f2e, red_f1l_fade, red_f1e_fade,
              red_f2l_fade, red_f2e_fade,
              red_min, red_sec_tens, red_sec_ones, red_top_colon, red_bottom_colon,
              red_bxr, red_one_player,
              red_two_player, red_round, red_card, red_one, red_two, green_f1l,
              green_f1e, green_f2l, green_f2e,
              green_f1l_fade, green_f1e_fade, green_f2l_fade, green_f2e_fade,
              green_min, green_sec_tens,
              green_sec_ones, green_top_colon, green_bottom_colon, green_bxr,
              green_one_player, green_two_player,
              green_round, green_card, green_one, green_two, blue_f1l, blue_f1e,
              blue_f2l, blue_f2e, blue_f1l_fade,
              blue_f1e_fade, blue_f2l_fade, blue_f2e_fade, blue_min, blue_sec_tens,
              blue_sec_ones, blue_top_colon,
              blue_bottom_colon, blue_bxr, blue_one_player, blue_two_player,
              blue_round, blue_card, blue_one, blue_two;

wire [23:0]   f1_RGB, f2_RGB;

 //player's life bar
```

```
rectangle f1l(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
              .corner_x(PLAYER1_PIXEL+(10'd260-next_f1l_pixels)),
             .corner_y(TOP),.height(BAR_HEIGHT),.width({2'b0,next_f1l_pixels}),
              .red(red_f1l),.green(green_f1l),
              .blue(blue_f1l),.blink(1'b0));
//opponent's life bar
rectangle f2l(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
              .corner_x(PLAYER2_PIXEL),.corner_y(TOP),
              .height(BAR_HEIGHT),.width({2'b0,next_f2l_pixels}),.red(red_f2l),
              .green(green_f2l),.blue(blue_f2l), .blink(1'b0));
//player's energy bar
rectangle f1e(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
              .corner_x(PLAYER1_PIXEL+(10'd260-f1e_width)),
              .corner_y(TOP+10'd15),.height(BAR_HEIGHT),.width({2'b0,f1e_width}),
              .red(red_f1e),.green(green_f1e),
              .blue(blue_f1e),.blink(1'b0));
//opponent's energy bar
rectangle f2e(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
              .corner_x(PLAYER2_PIXEL),.corner_y(TOP+10'd15),
              .height(BAR_HEIGHT),.width({2'b0,f2e_width}),.red(red_f2e),
              .green(green_f2e),.blue(blue_f2e), .blink(1'b0));
//rectangle module that causes fade effect of player's energy bar
rectangle f1e_fade(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
                   .corner_x(PLAYER1_PIXEL+(10'd260-f1e_pixels)),
                   .corner_y(TOP+10'd15),.height(BAR_HEIGHT),.width({2'b0,f1e_pixels}),
                   .red(red_f1e_fade),
                   .green(green_f1e_fade),.blue(blue_f1e_fade),.blink(1'b0));
//rectangle module that causes fade effect of opponent's energy bar
rectangle f2e_fade(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
                   .corner_x(PLAYER2_PIXEL),.corner_y(TOP+10'd15),
                   .height(BAR_HEIGHT),.width({2'b0,f2e_pixels}),.red(red_f2e_fade),
                   .green(green_f2e_fade),
                   .blue(blue_f2e_fade),.blink(1'b0));
//rectangle module that causes fade effect of opponent's life bar
rectangle f2l_fade(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
                   .corner_x(PLAYER2_PIXEL),.corner_y(TOP),
                   .height(BAR_HEIGHT),.width({2'b0,f2l_pixels}),.red(red_f2l_fade),
                   .green(green_f2l_fade),
                   .blue(blue_f2l_fade),.blink(1'b0));
//rectangle module that causes fade effect of player's life bar
rectangle f1l_fade(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
                   .corner_x(PLAYER1_PIXEL+(10'd260-f1l_pixels)),
                   .corner_y(TOP),.height(BAR_HEIGHT),        .width({2'b0,f1l_pixels}),
                   .red(red_f1l_fade),
                   .green(green_f1l_fade),.blue(blue_f1l_fade),.blink(1'b0));

//rectangle module for one player option on start screen
rectangle one_player(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
                     .corner_x(ONE_PLAYER),.corner_y(10'd0),
                     .height(SCREEN_HEIGHT), .width(MODE_WIDTH),.red(red_one_player),
                     .green(green_one_player),
```

```
                    .blue(blue_one_player),.blink(1'b0));
//rectangle module for two player option on start screen
rectangle two_player(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
                    .corner_x(TWO_PLAYER),.corner_y(10'd0),
                    .height(SCREEN_HEIGHT), .width(MODE_WIDTH),.red(red_two_player),
                    .green(green_two_player),
                    .blue(blue_two_player),.blink(1'b0));
//number module for number one displayed over the one player option on start screen
number one(.x(pixel_count),.y(line_count),.corner_x(ONE_PLAYER+10'd86),
          .corner_y(10'd220),
          .number(4'd1),.red(red_one),.green(green_one),.blue(blue_one));
//number module for number two displayed over the one player option on start screen
number two(.x(pixel_count),.y(line_count),.corner_x(TWO_PLAYER+10'd86),
          .corner_y(10'd220),
          .number(4'd2),.red(red_two),.green(green_two),.blue(blue_two));

//timer module to determine the time left in minutes and seconds
timer tm(.pixel_clock(pixel_clock),.reset(reset_sync),.minutes(minutes),
        .seconds(seconds),.time_in(time_in));
//number module for minutes in the time left displayed
number dmin(.x(pixel_count),.y(line_count),.corner_x(TIME_PIXEL),.corner_y(TOP),
           .number({2'b0,minutes}),.red(red_min),.green(green_min),.blue(blue_min));
//number module for tens of the seconds in the time left displayed
number dsec_tens(.x(pixel_count),.y(line_count),.corner_x(TIME_PIXEL+10'd45),
                .corner_y(TOP),
                .number(sec_tens),.red(red_sec_tens),.green(green_sec_tens),
                .blue(blue_sec_tens));
//number module for ones of the seconds in the time left displayed
number dsec_ones(.x(pixel_count),.y(line_count),.corner_x(TIME_PIXEL+10'd75),
                .corner_y(TOP),
                .number(sec_ones),.red(red_sec_ones),.green(green_sec_ones),
                .blue(blue_sec_ones));
//rectangle module for top colon in the time left displayed
// (blinking because blink is set high)
rectangle top_colon(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
                   .corner_x(TIME_PIXEL+10'd30),.corner_y(TOP+10'd5),.height(COLON),
                   .width(COLON),.red(red_top_colon),
                   .green(green_top_colon),.blue(blue_top_colon),
                   .reset_sync(reset_sync), .blink(1'b1));
//rectangle module for bottom colon in the time left displayed
// (blinking because blink is set high)
rectangle bottom_colon(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
                      .corner_x(TIME_PIXEL+10'd30),.corner_y(TOP+10'd15),
                      .height(COLON),
                      .width(COLON),.red(red_bottom_colon),
                      .green(green_bottom_colon),.blue(blue_bottom_colon),
                      .reset_sync(reset_sync),.blink(1'b1));

//boxing ring module to display boxing ring
boxing_ring bxr(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),.red(red_bxr),
               .green(green_bxr),.blue(blue_bxr));
```

```
//number module for round number displayed
number round_num(.x(pixel_count),.y(line_count),.corner_x(10'd290),.corner_y(10'd200),
                 .number(round),.red(red_round),.green(green_round),.blue(blue_round));
//rectangle module for round number card displayed
rectangle card(.pixel_clock(pixel_clock),.x(pixel_count),.y(line_count),
               .corner_x(10'd128),
               .corner_y(10'd80),.height(10'd320),
               .width(10'd384),.red(red_card),.green(green_card),.blue(blue_card),
               .blink(1'b0));

//Fighter1 Control module for player gloves displayed
Fighter1_Control f1c(.pixel_clock(pixel_clock),.reset(reset_sync),
                     .line_count(line_count),
                     .pixel_count(pixel_count),.f1_lhand_x(f1_lhand_x),
                     .f1_rhand_x(f1_rhand_x),
                     .f1_lhand_y(f1_lhand_y),
                     .lh_status(f1_lh_status),.rh_status(f1_rh_status),
                     .f1_rhand_y(f1_rhand_y),
                     .RGB(f1_RGB),.mode(mode));
//Fighter2 Control module for opponent displayed
Fighter2_Control f2c(.pixel_clock(pixel_clock),.reset(reset_sync),
                     .line_count(line_count),
                     .pixel_count(pixel_count),.pic_x(f2_pic_x),.pic_y(f2_pic_y),
                     .f2_status(f2_status),.RGB(f2_RGB),.mode(mode));

always @(posedge pixel_clock) begin
   //Decrements and increments the fade life and energy bars by one until it
   // reaches the actual
   //energy and life left. Increments the actual energy bars until it reaches
   // the actual energy left.
   //Creates fade effect and makes transitions smoother.
   if (!reset_sync) f1l_pixels<=next_f1l_pixels;
   else if ((f1l_pixels>next_f1l_pixels)&& frame_pulse)
     f1l_pixels <= f1l_pixels-1;
   if (!reset_sync) f1e_pixels<=next_f1e_pixels;
   else if ((f1e_pixels>next_f1e_pixels)&& frame_pulse)
     f1e_pixels <= f1e_pixels-1;
            else if ((f1e_pixels<next_f1e_pixels)&& frame_pulse)
              f1e_pixels <= f1e_pixels+1;
   if (!reset_sync) f2l_pixels <= next_f2l_pixels;
   else if ((f2l_pixels>next_f2l_pixels)&& frame_pulse)
     f2l_pixels <= f2l_pixels-1;
   if (!reset_sync) f2e_pixels <= next_f2e_pixels;
   else if ((f2e_pixels>next_f2e_pixels) && frame_pulse)
     f2e_pixels <= f2e_pixels-1;
   else if ((f2e_pixels<next_f2e_pixels) && frame_pulse)
     f2e_pixels <= f2e_pixels+1;
end

//converts energy and life left from 5 to 8 bits
```

```
assign next_f1l_pixels = {f1_lifebar, 3'b0};
assign next_f2l_pixels = {f2_lifebar, 3'b0};
assign next_f1e_pixels = {f1_energybar, 3'b0};
     assign next_f2e_pixels = {f2_energybar, 3'b0};
//calculates the energy bar widths
assign      f1e_width = (f1e_pixels>=next_f1e_pixels) ? next_f1e_pixels :
                        f1e_pixels;
assign      f2e_width = (f2e_pixels>=next_f2e_pixels) ? next_f2e_pixels :
                        f2e_pixels;

//computes red, green, and blue signals based on the mode
assign      red = ((mode==0) && (f1_RGB!=24'd0)) ? (f1_RGB[23:16]) :
                  (mode==0) ? (red_one_player|red_two_player|red_one|red_two) :
                  ((mode==2)&&((red_round==8'd0)||(red_card==8'd0))) ? 8'd0 :
                  (mode==2) ? (red_round|red_card) :
                  (f1_RGB!=24'd0) ? (f1_RGB[23:16]) :
                  (f2_RGB!=24'd0) ? (f2_RGB[23:16]) :
                  (red_f1l|red_f1e|red_f2l|red_f2e|red_f1l_fade|red_f1e_fade|red_f2l_fad
                   red_f2e_fade|red_min|red_sec_tens|red_sec_ones|red_top_colon|
                   red_bottom_colon|red_bxr);

assign      green = ((mode==0) && (f1_RGB!=24'd0)) ? (f1_RGB[15:8]) :
                    (mode==0) ? (green_one_player|green_two_player|green_one|green_two)
                    ((mode==2)&&((green_round==8'd0)||(green_card==8'd0))) ? 8'd0 :
                                                (mode==2) ? (green_round|green_card) :
                    (f1_RGB!=24'd0) ? (f1_RGB[15:8]) :
                    (f2_RGB!=24'd0) ? (f2_RGB[15:8]) :
                    (green_f1l|green_f1e|green_f2l|green_f2e|green_f1l_fade|green_f1e_fa
                     green_f2l_fade|green_f2e_fade|green_min|green_sec_tens|green_sec_or
                     green_top_colon|
                     green_bottom_colon|green_bxr);

assign      blue = ((mode==0) && (f1_RGB!=24'd0)) ? (f1_RGB[7:0]) :
                   (mode==0) ? (blue_one_player|blue_two_player|blue_one|blue_two) :
                   ((mode==2)&&((blue_round==8'd0)||(blue_card==8'd0))) ? 8'd0 :
                   (mode==2) ? (blue_round|blue_card) :
                   (f1_RGB!=24'd0) ? (f1_RGB[7:0]) :
                                                (f2_RGB!=24'd0) ? (f2_RGB[7:0]) :
                   (blue_f1l|blue_f1e|blue_f2l|blue_f2e|blue_f1l_fade|blue_f1e_fade|
                    blue_f2l_fade|
                    blue_f2e_fade|blue_min|blue_sec_tens|blue_sec_ones|blue_top_colon|
                    blue_bottom_colon| blue_bxr);

//computes tens of the seconds
assign      sec_tens = ((seconds>=0)&&(seconds<10)) ? 0 :
                       ((seconds>=10)&&(seconds<20)) ? 1 :
                       ((seconds>=20)&&(seconds<30)) ? 2 :
                       ((seconds>=30)&&(seconds<40)) ? 3 :
                       ((seconds>=40)&&(seconds<50)) ? 4 :
                       ((seconds>=50)&&(seconds<60)) ? 5 : 0;
//computes ones of the seconds
```

```
    assign      sec_ones = (seconds-(10*sec_tens));

endmodule
```

```
module Fighter1_Control(pixel_clock, reset, line_count, pixel_count, f1_lhand_x,
                        f1_rhand_x, f1_lhand_y, f1_rhand_y, lh_status, rh_status,
                        mode, RGB);

    input [9:0] line_count;
    input [9:0] pixel_count;
    input       pixel_clock, reset;
    input [9:0] f1_lhand_x, f1_rhand_x;
    input       f1_lhand_y, f1_rhand_y;
    input       lh_status, rh_status;
    input [1:0] mode;
    output [23:0] RGB;

    wire [23:0]   RGB;
    wire [9:0]    r_end_pixel, l_end_pixel, l_end_line, r_end_line, next_x, next_y,
                  lhand_y, rhand_y;
    wire          righthand;
    wire          lefthand;
    wire          righthand_next;
    wire          lefthand_next;
    wire          gm_data, game_over, game_over_next;
    wire [11:0]   address;
    wire [3:0]    glove_data, glove0_data, glove1_data, glove2_data, glove3_data;

    reg           r_enable_x, r_enable_y, l_enable_x, l_enable_y, g_enable_x,
                  g_enable_y;
    reg [11:0]    lhand_address;
    reg [11:0]    rhand_address;
    reg [13:0]    g_address;
    reg [23:0]    data;

    //ROMS containing images of the player's gloves
    glove0_rom g0r(.addr(lhand_address),.clk(pixel_clock),.dout(glove0_data));
    glove1_rom g1r(.addr(lhand_address),.clk(pixel_clock),.dout(glove1_data));
    glove2_rom g2r(.addr(rhand_address),.clk(pixel_clock),.dout(glove2_data));
    glove3_rom g3r(.addr(rhand_address),.clk(pixel_clock),.dout(glove3_data));
    gameover_rom gm1(.addr(g_address),.clk(pixel_clock),.dout(gm_data));

    //dimensions of the pictures (after being scaled by 2)
    parameter     PIC_DIMENSION_X = 100;
    parameter     PIC_DIMENSION_Y = 100;

    //last pixel count and line count in a frame
    parameter     PIXELS = 639;
    parameter     LINES = 479;

    //color parameters for RGB
    parameter     BLACK = 24'b0000_0000_0000_0000_0000_0000;
    parameter     DK_GRAY =24'b0001_0100_0001_0100_0001_0100;
    parameter     GRAY=24'b1001_0010_1001_0010_1001_0010;
    parameter     WHITE=24'b1111_1111_1111_1111_1111_1111;
```

```
parameter      DKR_RED = 24'b0100_0000_0000_0000_0000_0000;
parameter      DK_RED=24'b0110_1101_0000_0000_0000_0000;
parameter      RED=24'b1011_0110_0000_0000_0000_0000;
parameter      LT_RED=24'b1101_1011_0000_0000_0000_0000;
parameter      LT_BLUE=24'b0000_0000_0000_0000_1011_0110;

//starting and ending line and pixel of the game over image
// (after being scaled by 2)
parameter      G_START_LINE = 195;
parameter      G_START_PIXEL = 80;
parameter      G_END_LINE = 285;
parameter      G_END_PIXEL = 560;

always @ (posedge pixel_clock) begin
   //Computes enable_x for the left glove ROM which goes high every other
   // cycle which enables the
   //address to increment to the next address. This scales the image in the
   // ROM horizontally by two
   if ((next_x==(PIXELS-3)) && (next_y==(LINES-1))) l_enable_x<=1;
   else if (lefthand_next) l_enable_x<=~l_enable_x;

   //Computes enable_y for the left glove ROM which goes high every other line
   // which enables the
   //address to increment to the next address on the next line instead of forcing
   // it to decrement the
   //address by the number of horizontal pixels that correspond to the image's width.
   // This scales the
   //image in the ROM vertically by two
   if ((next_x==(PIXELS-3)) && (next_y==(LINES-1))) l_enable_y<=1;
   else if (next_x==l_end_pixel) l_enable_y<=~l_enable_y;

   //Computes the address for the left glove image
   if ((next_x==(f1_lhand_x-50))&&(line_count==lhand_y)) lhand_address<=12'd0;
   else if ((next_y>=lhand_y) && (next_x==(f1_lhand_x-50)) && l_enable_y)
     lhand_address<=(lhand_address-12'd49);
   else if (lefthand_next && l_enable_x) lhand_address<=lhand_address+1;

   //Computes enable_x for the right glove ROM...
   if ((next_x==(PIXELS-3)) && (next_y==(LINES-1))) r_enable_x<=1;
   else if (righthand_next) r_enable_x<=~r_enable_x;

   //Computes enable_y for the right glove ROM...
   if ((next_x==(PIXELS-2)) && (next_y==(LINES-1))) r_enable_y<=1;
   else if (next_x==r_end_pixel) r_enable_y<=~r_enable_y;

   //Computes the address for the right glove ROM
   if ((next_x==(f1_rhand_x-50))&&(line_count==rhand_y)) rhand_address<=12'd0;
   else if ((next_y>=rhand_y) && (next_x==(f1_rhand_x-50)) && r_enable_y)
     rhand_address<=(rhand_address-12'd49);
   else if (righthand_next && r_enable_x) rhand_address<=rhand_address+1;
```

```
    //Computes enable_x for the game over ROM...
    if ((next_x==(PIXELS-3)) && (next_y==(LINES-1))) g_enable_x<=1;
    else if (game_over_next) g_enable_x<=~g_enable_x;

    //Computes enable_y for the game over ROM...
    if ((next_x==(PIXELS-3)) && (next_y==(LINES-1))) g_enable_y<=1;
    else if (next_x==G_END_PIXEL) g_enable_y<=~g_enable_y;

    //Computes the address for the game over ROM
    if ((next_x==G_START_PIXEL)&&(line_count==G_START_LINE)) g_address<=14'd0;
    else if ((next_y>=G_START_LINE) && (next_x==G_START_PIXEL) && g_enable_y)
      g_address<=(g_address-14'd239);
    else if (game_over_next && g_enable_x) g_address<=g_address+1;

    //Computes the color based on the 4-bit number outputted from the ROM
    data <=  (glove_data==8) ? LT_BLUE:
             (glove_data==0) ? BLACK :
             (glove_data==1) ? DK_GRAY:
             (glove_data==2) ? GRAY:
             (glove_data==3) ? WHITE:
             (glove_data==4) ? DKR_RED:
             (glove_data==5) ? DK_RED:
             (glove_data==6) ? RED: LT_RED;

end

//Computes whether the current position is within the boundary of the left glove
assign lefthand = ((pixel_count<l_end_pixel)&&(pixel_count>=(f1_lhand_x-50))&&
                   (line_count<l_end_line)&&(line_count>=lhand_y));

//Computes whether the current position is within the boundary of the right glove
assign righthand = ((pixel_count<r_end_pixel)&&(pixel_count>=(f1_rhand_x-50))&&
                    (line_count<r_end_line)&&(line_count>=rhand_y));

//Computes whether the next position is within the boundary of the left glove
assign lefthand_next = ((next_x<l_end_pixel)&&(next_x>=(f1_lhand_x-50))&&
                        (next_y<l_end_line)&&(next_y>=lhand_y));

//Computes whether the next position is within the boundary of the right glove
assign righthand_next = ((next_x<r_end_pixel)&&(next_x>=(f1_rhand_x-50))&&
                         (next_y<r_end_line)&&(next_y>=rhand_y));

//Computes whether the current position is within the boundary of the game over image
assign game_over = (next_x>=G_START_PIXEL)&&(next_x<G_END_PIXEL)&&
                   (next_y>=G_START_LINE) &&(next_y<G_END_LINE);
assign game_over_next = (pixel_count>=G_START_PIXEL)&&(pixel_count<G_END_PIXEL)&&
                        (line_count>=G_START_LINE)&&(line_count<G_END_LINE);

//Computes whether the next position is within the boundary of the game over image
assign next_x = (pixel_count == (PIXELS-3)) ? 10'd0 : (pixel_count==(PIXELS-2)) ?
                10'd1 :
```

```
                    (pixel_count==(PIXELS-1)) ? 10'd2 : pixel_count + 3;
    assign next_y = ((pixel_count >= (PIXELS-3))&&(pixel_count<=(PIXELS-1))) ?
                    (line_count == (LINES-1)) ? 10'd0 : line_count + 1'b1 : line_count;


    //Determines which ROM to read from based on the status
    assign glove_data = (game_over && (mode==3) && gm_data)? 4'd8 :
                        (lefthand && lh_status) ? glove1_data :
                        (lefthand ? glove0_data:
                         (righthand && rh_status) ? glove3_data :
                         (righthand) ? glove2_data : 4'b0;

     //Computes the last pixel of the line of the left glove image
     assign l_end_pixel = (f1_lhand_x-50)+PIC_DIMENSION_X;

     //Computes the last pixel of the line of the right glove image
     assign r_end_pixel = (f1_rhand_x-50)+PIC_DIMENSION_X;

     //Computes the last line of the left glove image
     assign l_end_line = lhand_y+PIC_DIMENSION_Y;

    //Computes the last line of the right glove image
     assign r_end_line = rhand_y+PIC_DIMENSION_Y;

     //Assigns RGB the data if its current position is within the boundary of the either
     // glove or the game over image,
     //otherwise it assigns it black
     assign RGB = (righthand||lefthand||game_over)? data : 24'h0;

    //Assigns the gloves' y position based on a 1-bit number
     assign lhand_y=(f1_lhand_y) ? 10'd125 : 10'd235;
     assign rhand_y=(f1_rhand_y) ? 10'd125 : 10'd235;
endmodule
```

```
module Fighter2_Control(pixel_clock, reset, line_count, pixel_count,
                        pic_x, pic_y, f2_status, mode, RGB);
   input pixel_clock;
   input reset;
   input [1:0] mode;
   input [9:0] line_count;
   input [9:0] pixel_count;
   input [9:0] pic_x, pic_y;
   input [4:0] f2_status;
   output [23:0] RGB;


   wire [23:0]   RGB;
   wire [9:0]    end_pixel, end_line, next_x, next_y;
   wire          fighter;
   wire          fighter_next;

   wire [3:0]    fighter_data, fighter0_data, fighter1_data, fighter2_data, fighter3_data,
                 fighter4_data, fighter5_data,
                 fighter6_data, fighter7_data, fighter8_data, fighter9_data,
                 fighter12_data, fighter13_data, fighter16_data, fighter17_data,
                 fighter18_data, fighter19_data, fighter20_data, fighter21_data,
                 fighter23_data, fighter24_data, fighter25_data;

   reg [14:0]    address;
   reg           enable_x, enable_y;
   reg [23:0]    data;

   //ROMs that contain the bitmaps of the opponent
   fighter0_rom f0r(.addr(address),.clk(pixel_clock),.dout(fighter0_data));
   fighter1_rom f1r(.addr(address),.clk(pixel_clock),.dout(fighter1_data));
   fighter2_rom f2r(.addr(address),.clk(pixel_clock),.dout(fighter2_data));
   fighter3_rom f3r(.addr(address),.clk(pixel_clock),.dout(fighter3_data));
   fighter4_rom f4r(.addr(address),.clk(pixel_clock),.dout(fighter4_data));
   fighter5_rom f5r(.addr(address),.clk(pixel_clock),.dout(fighter5_data));
   fighter6_rom f6r(.addr(address),.clk(pixel_clock),.dout(fighter6_data));
   fighter7_rom f7r(.addr(address),.clk(pixel_clock),.dout(fighter7_data));
   fighter8_rom f8r(.addr(address),.clk(pixel_clock),.dout(fighter8_data));
   fighter9_rom f9r(.addr(address),.clk(pixel_clock),.dout(fighter9_data));
   fighter16_rom f16r(.addr(address),.clk(pixel_clock),.dout(fighter16_data));
   fighter17_rom f17r(.addr(address),.clk(pixel_clock),.dout(fighter17_data));
   fighter18_rom f18r(.addr(address),.clk(pixel_clock),.dout(fighter18_data));
   fighter19_rom f19r(.addr(address),.clk(pixel_clock),.dout(fighter19_data));
   fighter20_rom f20r(.addr(address),.clk(pixel_clock),.dout(fighter20_data));
   fighter21_rom f21r(.addr(address),.clk(pixel_clock),.dout(fighter21_data));
   fighter23_rom f23r(.addr(address),.clk(pixel_clock),.dout(fighter23_data));
   fighter24_rom f24r(.addr(address),.clk(pixel_clock),.dout(fighter24_data));
   fighter25_rom f25r(.addr(address),.clk(pixel_clock),.dout(fighter25_data));


   //dimensions of the pictures (after being scaled by 2)
```

```
parameter      PIC_DIMENSION_X = 200;
parameter      PIC_DIMENSION_Y = 500;

//last pixel count and line count in a frame
parameter      PIXELS = 639;
parameter      LINES = 479;

//color parameters for RGB
parameter      BLACK = 24'b0000_0000_0000_0000_0000_0000;
parameter      DK_GRAY =24'b0001_0100_0001_0100_0001_0100;
parameter      GRAY=24'b1001_0010_1001_0010_1001_0010;
parameter      WHITE=24'b1111_1111_1111_1111_1111_1111;
parameter      DKR_RED = 24'b0100_0000_0000_0000_0000_0000;
parameter      DK_RED=24'b0110_1101_0000_0000_0000_0000;
parameter      RED=24'b1011_0110_0000_0000_0000_0000;
parameter      LT_RED=24'b1101_1011_0000_0000_0000_0000;
parameter      DK_BLUE=24'b0000_0000_0000_0000_0110_1101;
parameter      LT_BLUE=24'b0000_0000_0000_0000_1011_0110;
parameter      DKR_BROWN=24'b0101_0000_0010_1000_0000_0000;
parameter      DK_BROWN=24'b0110_0100_0011_0010_0000_0000;
parameter      LT_BROWN=24'b1000_0000_0100_0000_0000_0000;
parameter      DK_YELLOW=24'b0111_0110_0101_0101_0000_1010;
parameter      LT_YELLOW=24'b1010_1100_0111_1101_0000_1111;

always @ (posedge pixel_clock) begin
   //Computes enable_x which goes high every other cycle which enables the
   // address to increment to the next address
   //This scales the image in the ROM horizontally by two
   if ((next_x==(PIXELS-3)) && (next_y==(LINES-1))) enable_x<=1;
   else if (fighter_next) enable_x<=~enable_x;

   //Computes enable_y which goes high every other line which enables the address
   // to increment to the next address on
   //the next line instead of forcing it to decrement the address by the number of
   // horizontal pixels that correspond
   //to the image's width. This scales the image in the ROM vertically by two
   if ((next_x==(PIXELS-3)) && (next_y==(LINES-1))) enable_y<=1;
   else if (next_x==end_pixel) enable_y<=~enable_y;

   //Computes the address
   if ((next_x==(pic_x-100))&&(line_count==(pic_y-250))) address<=15'd0;
   else if ((next_y>=(pic_y-250)) && (next_x==(pic_x-100)) && enable_y)
     address<=(address-15'd99);
   else if (fighter_next && enable_x) address<=address+1;

   //Computes the color based on the 4-bit number outputted from the ROM
   data <= (fighter_data==0) ? BLACK :
           (fighter_data==1) ? GRAY:
           (fighter_data==2) ? WHITE:
           (fighter_data==3) ? DK_RED:
           (fighter_data==4) ? RED:
```

```
            (fighter_data==5) ? LT_RED:
            (fighter_data==6) ? DK_BLUE:
            (fighter_data==7) ? LT_BLUE:
            (fighter_data==8) ? DK_BROWN:
            (fighter_data==9) ? LT_BROWN:
            (fighter_data==10) ? DK_YELLOW:
            (fighter_data==12) ? DKR_BROWN:
            (fighter_data==13) ? DKR_RED :
            (fighter_data==14) ? DKR_BROWN: LT_YELLOW;
end

//Computes whether the next position is within the boundary of the opponent
assign fighter_next = (next_x>=(pic_x-100))&&(next_x<end_pixel)&&(next_y>=(pic_y-250))
  &&(next_y<end_line);

//Computes whether the current position is within the boundary of the opponent
assign fighter = (pixel_count>=(pic_x-100))&&(pixel_count<end_pixel)&&
                 (line_count>=(pic_y-250))&&(line_count<end_line);

//Computes the next position
assign next_x = (pixel_count == (PIXELS-3)) ? 10'd0 : (pixel_count==(PIXELS-2)) ?
                10'd1 :
                (pixel_count==(PIXELS-1)) ? 10'd2 : pixel_count + 3;
assign next_y = ((pixel_count >= (PIXELS-3))&&(pixel_count<=(PIXELS-1))) ?
                (line_count == (LINES-1)) ? 10'd0 : line_count + 1'b1 : line_count;

//Computes the last line of the image
assign end_line = pic_y+PIC_DIMENSION_Y-250;

//Computes the last pixel of the line of the image
assign end_pixel = pic_x+PIC_DIMENSION_X-100;

//Assigns RGB the data if its current position is within the boundary of the opponent,
// otherwise it assigns it black
assign RGB = (fighter)? data : 24'h0;

//Determines which ROM to read from based on the status
assign fighter_data = ((f2_status==0)||(f2_status==22)) ? fighter0_data :
                      (f2_status==1) ? fighter1_data :
                      (f2_status==2) ? fighter2_data :
                      (f2_status==3) ? fighter3_data :
                      (f2_status==4) ? fighter4_data :
                      (f2_status==5) ? fighter5_data :
                      ((f2_status==6)||(f2_status==10)||(f2_status==14)) ?
                      fighter6_data :
                      ((f2_status==7)||(f2_status==11)||(f2_status==15)) ?
                      fighter7_data :
                      ((f2_status==8)||(f2_status==12)) ? fighter8_data :
                      ((f2_status==9)||(f2_status==13)) ? fighter9_data :
                      (f2_status==16) ? fighter16_data :
                      (f2_status==17) ? fighter17_data :
```

```
                        (f2_status==18) ? fighter18_data :
                        (f2_status==19) ? fighter19_data :
                        (f2_status==20) ? fighter20_data :
                        (f2_status==21) ? fighter21_data :
                        (f2_status==23) ? fighter23_data :
                        (f2_status==24) ? fighter24_data : fighter25_data;
endmodule
```

```
//                  a
//                 ____
//                |    |
//              f | g  | b
//                ____
//                |    |
//              e |    |  c
//                ____
//                  d
module number(x, y, corner_x, corner_y, num, red, green, blue);
   input [9:0] x, y, corner_x, corner_y;
   input [3:0]       num;
   output [7:0] red;
   output [7:0]  green;
   output [7:0]  blue;

   reg [23:0]    RGB;

   parameter         ZERO = 0;
   parameter        ONE = 1;
   parameter        TWO = 2;
   parameter        THREE = 3;
   parameter        FOUR = 4;
   parameter        FIVE = 5;
   parameter        SIX = 6;
   parameter        SEVEN = 7;
   parameter        EIGHT = 8;
   parameter        NINE = 9;
   parameter        HDIM = 14;
   parameter        VDIM = 24;
   parameter        WIDTH = 3;
   parameter        SPACE = 6;
   parameter        COLOR = 24'hffffff;
   parameter        BACKGROUND = 24'd0;

   wire             a,b,c,d,e,f,g;//segments as defined above

   always @ (num or a or b or c or d or e or f or g) begin
      case (num)
        ZERO:            RGB = (a||b||c||d||e||f) ? COLOR : BACKGROUND;
        ONE:             RGB = (b||c) ?  COLOR :          BACKGROUND;
        TWO:             RGB = (a||b||d||e||g) ? COLOR : BACKGROUND;
        THREE:  RGB = (a||b||c||d||g) ? COLOR : BACKGROUND;
        FOUR:            RGB = (b||c||f||g) ? COLOR : BACKGROUND;
        FIVE:            RGB = (a||c||d||f||g) ? COLOR : BACKGROUND;
        SIX:             RGB = (a||c||d||e||f||g) ? COLOR : BACKGROUND;
        SEVEN:  RGB = (a||b||c) ? COLOR : BACKGROUND;
        EIGHT:  RGB = (a||b||c||d||e||f||g) ? COLOR : BACKGROUND;
        NINE:            RGB = (a||b||c||d||f||g) ? COLOR : BACKGROUND;
        default: RGB = BACKGROUND;
      endcase
```

```
    end

    assign a = ((x>=(corner_x+WIDTH))&&(x<=(corner_x+HDIM-WIDTH))&&(y>=corner_y)
                &&(y<=(corner_y+WIDTH)));
    assign b = ((x>=(corner_x+HDIM-WIDTH))&&(x<=(corner_x+HDIM))&&
                (y>=(corner_y+WIDTH))&&(y<=(corner_y+WIDTH+SPACE)));
    assign c = ((x>=(corner_x+HDIM-WIDTH))&&(x<=(corner_x+HDIM))&&
                (y>=(corner_y+2*WIDTH+SPACE))&&(y<=(corner_y+VDIM-WIDTH)));
    assign d = ((x>=(corner_x+WIDTH))&&(x<=(corner_x+HDIM-WIDTH))&&
                (y>=(corner_y+VDIM-WIDTH))&&(y<=(corner_y+VDIM)));
    assign e = ((x>=corner_x)&&(x<=(corner_x+WIDTH))&&(y>=(corner_y+2*WIDTH+SPACE))
                &&(y<=(corner_y+VDIM-WIDTH)));
    assign f = ((x>=corner_x)&&(x<=(corner_x+WIDTH))&&(y>=(corner_y+WIDTH))&&
                (y<=(corner_y+WIDTH+SPACE)));
    assign g = ((x>=(corner_x+WIDTH))&&(x<=(corner_x+HDIM-WIDTH))&&
                (y>=(corner_y+WIDTH+SPACE))&&(y<=(corner_y+2*WIDTH+SPACE)));

    assign red = RGB[23:16];
    assign green = RGB[15:8];
    assign blue = RGB[7:0];

endmodule
```

```
module rectangle(pixel_clock, reset, x, y, corner_x, corner_y, height, width, red, green,
                 blue, blink);
   input pixel_clock, reset, blink;
   input [9:0] x, y, corner_x, corner_y, height, width;
   output [7:0] red, green, blue;

   wire [7:0]   red, green, blue;
   wire [23:0]  RGB;

   parameter    COLOR=24'b0101_1111_0001_1111_0001_1111;
   defparam     ct.endcount = 25174999;

   counter ct(.clk(pixel_clock),.Reset_Sync(reset),.enable(enable));

   always @(posedge pixel_clock) begin
      if (!reset_sync) counter<=0;
      else if (enable && (counter==2) && blink) counter<=0;
      else if (enable && blink) counter<=(counter+1);
   end

   assign RGB = ((x>=corner_x) && (y>=corner_y) && (x<(corner_x+width)) &&
                 (y<(corner_y+height)) && (counter!=2)) ? COLOR : 24'h0;
   assign RGB = ((x>=corner_x) && (y>=corner_y) && (x<(corner_x+width)) &&
                 (y<(corner_y+height))) ? COLOR : 24'h0;

   assign red = RGB[23:16];
   assign green = RGB[15:8];
   assign blue = RGB[7:0];

endmodule
```

```
module timer(pixel_clock, reset, mode, minutes, seconds);
    input pixel_clock;
    input reset;
    input mode;
    output [1:0] minutes;
    output [5:0] seconds;

        reg enable;
        reg [7:0] counter;

        always @(posedge pixel_clock) begin
                if (!reset) counter<=180;
                else if (enable && (counter==0)) counter<=0;
                else if (enable) counter<=counter-1;

                if (!reset) minutes<=3;
                else if ((counter==180) && enable) minutes<=2;
                else if ((counter==121) && enable) minutes<=1;
                else if ((counter==61) && enable) minutes<=0;

                if (!reset) seconds<=0;
                else if ((seconds==0) && enable && (minutes!=0)) seconds<=59;
                else if ((seconds==0) && enable) seconds<=0;
                else if (enable) seconds<=seconds-1;

        end


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:18:06 04/23/06
// Design Name:
// Module Name:    top
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module top(pixel_clock, reset_sync, round, mode, f1_lhand_x, f1_rhand_x, f1_lhand_y,
           f1_rhand_y,f1_lh_status, f1_rh_status, f1_lifebar, f1_energybar, f2_pic_x,
           f2_pic_y, f2_status, f2_lifebar, f2_energybar, hsync, vsync,
           sync_b, blank_b, red, green, blue, pixel_count, line_count, time_in,
           frame_pulse);

   input pixel_clock;
   input reset_sync;
   input [1:0] mode;
   input [3:0] round;
   input [4:0] f1_lifebar, f1_energybar, f2_lifebar, f2_energybar;
   input [9:0] f1_lhand_x, f1_rhand_x, f2_pic_x, f2_pic_y;
   input [4:0] f2_status;
   input       f1_lhand_y, f1_rhand_y;
   input       f1_lh_status, f1_rh_status;
   input [7:0] time_in;
   output      hsync, vsync, sync_b, blank_b, frame_pulse;
   output [7:0] red, green, blue;
   output [9:0] pixel_count, line_count;

   wire [9:0]   pixel_count, line_count;
   wire [23:0]  f1_RGB, f2_RGB;
   wire         reset_sync, frame_pulse;

   Display_Field df(.pixel_clock(pixel_clock),.reset_sync(reset_sync),
                    .pixel_count(pixel_count),.line_count(line_count),
                    .f1_lifebar(f1_lifebar),.f1_energybar(f1_energybar),
                    .f2_lifebar(f2_lifebar), .f2_energybar(f2_energybar),
                    .f1_lhand_x(f1_lhand_x),.f1_rhand_x(f1_rhand_x),
                    .f1_lhand_y(f1_lhand_y), .lh_status(f1_lh_status),
                    .rh_status(f1_rh_status),.f1_rhand_y(f1_rhand_y),
```

```
                    .f2_pic_x(f2_pic_x),.f2_pic_y(f2_pic_y),.f2_status(f2_status),
                    .red(red),.green(green),.blue(blue),.frame_pulse(frame_pulse),
                    .mode(mode),.round(round),.time_in(time_in));

  vga vga1(.pixel_clock(pixel_clock),.reset(reset_sync),.hsync(hsync),.vsync(vsync),
           .sync_b(sync_b), .blank_b(blank_b),.pixel_count(pixel_count),
           .line_count(line_count),.frame_pulse(frame_pulse));


endmodule
```

```
import java.awt.image.BufferedImage;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;
import java.util.ArrayList;

import javax.imageio.ImageIO;

/**
 *
 */

/**
 * @author David Blau
 *
 */
public class NewImageConvert {

    /**
     * @param args
     */
    public static void main(String[] args) {
        for (int i = 0; i<4; i++){
            String imagefilename, coefilename;
            if (i<10){
                imagefilename = "glove000"+i+".bmp";
                coefilename = "newcoe/glove"+i+".coe";
            }
            else{
                imagefilename = "fighter00"+i+".bmp";
                coefilename = "newcoe/fighter"+i+".coe";
            }
            //List<String> elements = new ArrayList<String>();
            if(args.length!=0) {
                imagefilename = args[0];
                        coefilename = args[1];
            }

            File imageFile = new File(imagefilename);

            BufferedImage image = null;
            try{
                image = ImageIO.read(imageFile);
            } catch (IOException ioe){
                System.err.println("Error reading image: "+imagefilename);
                return;
            }

            int height = image.getHeight();
```

```
int width = image.getWidth();
int rgb=0;
String rgbString;
int red;
int green;
int blue;
String outputString;

if(!coefilename.endsWith(".coe")) coefilename+=".coe";
try {
    BufferedWriter out = new BufferedWriter(new FileWriter(coefilename));
    out.write("memory_initialization_radix=2;\n"+
              "memory_initialization_vector=\n\n");

    for(int y=0;y<height;y++){
        for(int x=0;x<width;x++){
            rgb = image.getRGB(x,y);

            rgbString = Integer.toBinaryString(rgb);
            //System.out.println(rgbString);
            red = Integer.parseInt(rgbString.substring(8,11),2);
            green = Integer.parseInt(rgbString.substring(16,19),2);
            blue = Integer.parseInt(rgbString.substring(24,27),2);
            System.out.println(red);
            if ((red==green) && (green==blue)) {
                if (red==0) outputString="0000";
                else if (red<3) outputString = "1100";
                else if (red<6) outputString="0001";
                else outputString="0010";
            }
            else if ((green==0)&&(blue==0)){
                if (red==1) outputString="1101";
                else if (red<4) outputString="0011";
                else if (red<6) outputString="0100";
                else outputString = "0101";
            }
            else if ((red==0)&&(green==0)){
                if (blue<2) outputString="1100";
                else if (blue<5) outputString="0110";
                else outputString="0111";
            }
            else if (((red-1)==green)&&((green-1)==blue)){
                if (red<3) outputString ="1110";
                else if (red<5) outputString="1000";
                else outputString="1001";
            }
            else if ((red==green) && (blue<red)){
                if (red<3) outputString="1100";
                else if ((red<5) && (red-blue>1)) outputString="1010";
                else if (red-blue>1) outputString="1011";
                else if (red<6) outputString="0001";
```

```
                        else outputString="0010";
                    }
                    else if ((red>green) && (red>blue) && (green!=blue)){
                        if (red<3) outputString="1110";
                        else if (red<5) outputString="1000";
                        else outputString="1001";
                    }
                    else if ((red>green)&&(red>blue)){
                        if (red==1) outputString="1101";
                        else if (red<4) outputString="0011";
                        else if (red<6) outputString="0100";
                        else outputString = "0101";
                    }
                    else if ((green>red) && (green>blue)){
                        if (green<3) outputString="1110";
                        else if ((green<5) && ((green-blue)>1)) outputString="1010";
                        else if ((green-blue)>1) outputString="1011";
                        else if (green<6) outputString="0001";
                        else outputString="0010";
                    }
                    else if ((blue>red) && (blue>green)){
                        if (blue<2) outputString="1100";
                        else if (blue<5) outputString="0110";
                        else outputString="0111";
                    }
                    else {
                        if (blue<3) outputString="1100";
                        else if (blue<6) outputString="0001";
                        else outputString="0010";
                    }


                    while(rgbString.length()<8)
                        rgbString = "0"+rgbString;

                    //if (!elements.contains(outputString)){
                    //elements.add(outputString);
                    if(x==width-1 && y==height-1)
                        out.write(outputString+";\n");
                    else
                        out.write(outputString+",\n");
                    //}
                }
            }
            out.close();
        } catch (IOException e) {
            System.err.println("Error writing coe:"+coefilename);
        }
    }
    }
}
```