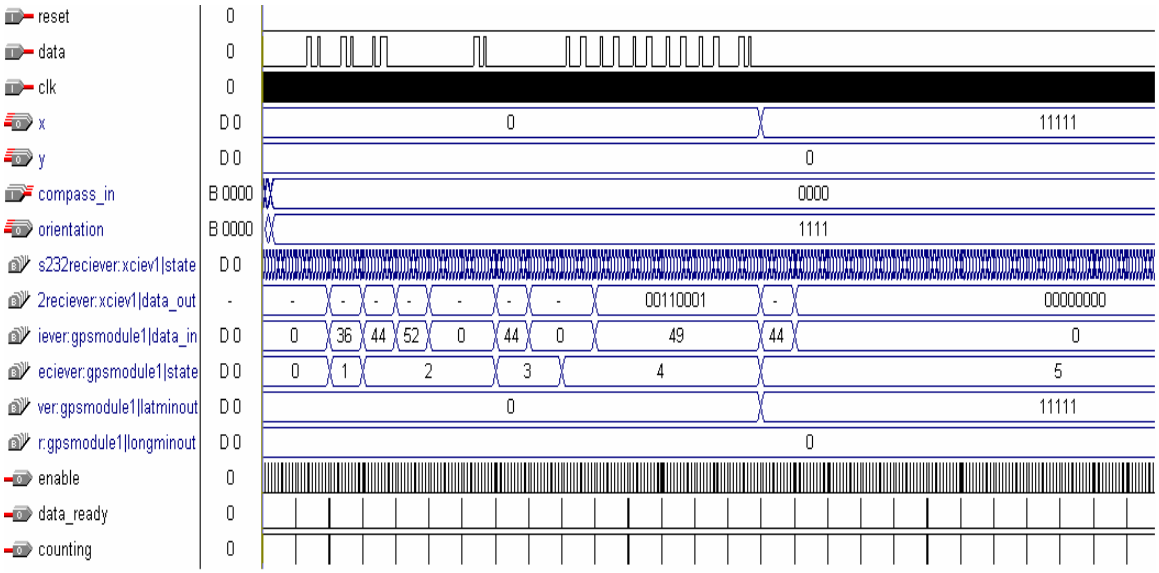
Appendix

GPS Top simulation



GPS Top

```
module GPStop(clk, reset, data, data2, x, y, compass_in, orientation,
                                    counting, enable, data_ready, fix, state, ascii_data);

input clk, reset, data, data2;
input [3:0] compass_in;

output[5:0] x,y;
output[3:0] orientation;
output fix;
output [4:0] state;

output counting, enable; //commmunication between serialclock and rs232

output data_ready;              //communication between rs232 and gps
//wire [7:0] ascii_data;        //adat fom rs232 to GPS
output [7:0] ascii_data;

wire [7:0] ascii_data2;
```

```
wire data_sync2, enable2, counting2, data_ready2,
                gps_ready, wireless_ready;

wire [5:0] x_gps, y_gps, x_wireless, y_wireless;

sync sreset(clk, reset, reset_sync);

sync sdata(clk, data, data_sync);

sync sdata2(clk, data2, data_sync2);


sync scompass0(clk, ~compass_in[0], orientation[0]);
sync scompass1(clk, ~compass_in[1], orientation[1]);
sync scompass2(clk, ~compass_in[2], orientation[2]);
sync scompass3(clk, ~compass_in[3], orientation[3]);

serialclock sclk1(clk, reset_sync, counting, enable);

serialclock sclk2(clk, reset_sync, counting2, enable2);

rs232reciever xciev1(reset_sync, clk, enable, data_sync, counting,
                                        data_ready, ascii_data);

rs232reciever xciev2(reset_sync, clk, enable2, data_sync2, counting2,
                                        data_ready2, ascii_data2);


GPSreciever gpsmodule1(clk, reset_sync, data_ready, ascii_data,
                                                x_gps,y_gps, fix, state, gps_ready);

wirelessreciever wireless1(clk, reset_sync, data_ready2, ascii_data2,
                                                x_wireless, y_wireless, wireless_ready);

always @(negedge gps_ready or negedge wireless_ready) begin
x <= fix ? x_gps : x_wireless;
y <= fix ? y_gps : y_wireless;
end

endmodule
```

Synchronizer

```
module sync(clk, in, out);
```

```
input clk, in;
output out;
reg r1, out;
always @(posedge clk) begin
        r1 <= in;
        out <= r1;
end
endmodule
```

Serial Clock

```
module serialclock(clk, Reset_sync, counting, enable);

        input clk, Reset_sync, counting;
        output enable;

        //parameter clock_frequency = 27000000;
//      parameter baud_rate = 9600;

        //parameter clock_frequency = 200;
        //parameter baud_rate = 10;


        parameter cycles_per_bit = 2812;       //must be even

        reg[12:0] count;
        reg enable;


        always @(posedge clk)
                begin
                if(Reset_sync==1)
                        begin
                        count <= 13'd0;
                        enable <= 1'b0;
                        end


                if (counting) begin
                        if (count == ((cycles_per_bit/2)-1)) begin
                                enable <= 1'b1;
                                count <= count + 1;
                                end
```

```
                else if (count == (cycles_per_bit-1))
                //      else if (count == 15'd0015)
                        count <= 13'd0;

                else
                        begin
                        count <= count+1;
                        enable <= 1'b0;
                        end
                end

        else begin
                count <= 13'd0;
                enable <= 1'b0;
        end
        end
endmodule
```

RS232 Decoder

```
module rs232reciever (reset, clock, enable, data, counting, data_ready, data_out);

  input reset; // Active high asynchronous reset
  input clock; // system clock
  input enable; //from serial clock
  input data; // PS/2 data
  output counting; //starts serial clock
  output data_ready;  //indicates that the data is ready
  output [7:0] data_out;
  //output [39:0] disp; // Bitmap for display (1 character)

  reg [7:0] keycode, data_out;
  reg [39:0] disp;
  reg [3:0] state;
  reg counting, data_ready, decoding, decoding2;
  reg [3:0] start_count;

  always @(posedge clock)
    begin
        if (reset)
          begin
            state <= 0;
            keycode <= 0;
            counting <= 0;
```

```
    data_ready <= 0;
    data_out <= 0;
        start_count <= 0;
        decoding <= 0;
        decoding2 <= 0;
  end
else    begin
        if(decoding == 0) begin
                state <= 0;
                data_ready <= 0;
                decoding2 <= 0;
                if(data) begin
                        counting <= 1;
                        if (enable)  begin
                                if (start_count[3])
                                start_count <= start_count;
                                else
                                start_count <= start_count + 1;
                        end
                end
                else begin
                        if (start_count[3]) begin
                                decoding <= 1;
                                counting <= 0;
                                state <= 0;
                                start_count <= 0;
                                end
                        else begin
                                counting <= 1;
                                start_count <= 0;
                                end

                end
        end


        else begin

        data_ready <= 0;


    if (decoding2)
                if (state == 0)
                        if (!data)
                                counting <= 1;
```

```
        if (decoding2 == 0)
                counting <= 1;



        case(state)
4'd0: // Start Bit
  if (enable)
                state <= 1;

4'd1: // Bit 0
  if(enable) begin
  keycode[0] <= data;
  state <= state+1;
  end

4'd2: if (enable) begin
  // Bit 1
  keycode[1] <= data;
  state <= state+1;
end
4'd3: if(enable) begin
  // Bit 2
  keycode[2] <= data;
  state <= state+1;
end
4'd4: if(enable) begin
  // Bit 3
  keycode[3] <= data;
  state <= state+1;
end
4'd5: if(enable) begin
  // Bit 4
  keycode[4] <= data;
  state <= state+1;
end
4'd6: if(enable) begin
  // Bit 5
  keycode[5] <= data;
  state <= state+1;
end
4'd7: if(enable) begin
  // Bit 6
  keycode[6] <= data;
  state <= state+1;
end
```

```
            4'd8: if(enable) begin
              // Bit 7
              keycode[7] <= data;
              state <= state+1;
            end
            4'd9: if(enable) begin
              // Stop bit
              state <= 0;
              counting <= 0;
              data_out <= keycode;
              data_ready <= 1;
                    decoding2 <= 1;
            end
          endcase
          end
          end
    end

endmodule
```

GPS FSM

```
module GPSreciever(clk, reset, data_ready, data_in,
                   latminout, longminout, fix, state, position_ready);

input clk, data_ready, reset;
input [7:0] data_in;

output [5:0] latminout, longminout;
output fix;
output [4:0] state;
output position_ready;

reg fix;
reg [4:0] state;
reg [4:0] count;
reg [16:0] latitude_minutes_out, longitude_minutes_out;
reg [5:0] latminout, longminout;
reg position_ready;


parameter IDLE = 0;
parameter HEADER = 1;
parameter TIME = 2;
parameter LATDEG = 3;
```

```
parameter LATMIN = 4;
parameter NS = 5;
parameter LONGDEG = 6;
parameter LONGMIN = 7;
parameter EW = 8;
parameter FIX = 9;
parameter SATELLITES = 10;
parameter HDOP = 11;
parameter ALT = 12;
parameter UNITS = 13;
parameter AGE = 14;
parameter HEADER2 = 15;
parameter HEADER3 = 16;
parameter HEADER4 = 17;
parameter HEADER5 = 18;
parameter HEADER6 = 19;

always @(posedge clk) begin

if (reset) begin
        state <= IDLE;
        latitude_minutes_out <= 17'd0;
        longitude_minutes_out <= 17'd0;
        latminout <= 0;
        longminout <= 0;
        fix <= 0;
        position_ready <= 0;
        end

else if (data_ready)
        case(state)

// go through all parts of message for increased functionality, if desired...

        IDLE:  if (data_in == 36) state <= HEADER;
                else state <= IDLE;

        HEADER:      if (data_in == 71) state <= HEADER2;
                else state <= IDLE;

        HEADER2:    if (data_in == 80) state <= HEADER3;
                else state <= IDLE;

        HEADER3:    if (data_in == 71) state <= HEADER4;
                else state <= IDLE;
```

```
HEADER4:    if (data_in == 71) state <= HEADER5;
        else state <= IDLE;

HEADER5:    if (data_in == 65) state <= HEADER6;
        else state <= IDLE;

HEADER6:    if (data_in == 44) state <= TIME;
        else state <= IDLE;

TIME: if (data_in == 44) begin count <= 1; state <= LATDEG; end
        else state <= TIME;

LATDEG:     if (count == 0) begin count <= 4; state <= LATMIN; end
        else begin count <= count - 1; state <= LATDEG; end

LATMIN:     if (data_in == 44) begin state <= NS;
                    latminout <= (55586 - latitude_minutes_out) / 22;
                end
        else begin
                if (data_in != 46)
                    latitude_minutes_out <=
                    latitude_minutes_out*10 + (data_in-48);
                count <= count - 1;
                state <= LATMIN;
        end

NS:     if (data_in == 44) begin count <= 1; state <= LONGDEG; end
        else state <= NS;

LONGDEG: if (count == 0) begin count <= 4; state <= LONGMIN; end
        else begin count <= count - 1; state <= LONGDEG; end

LONGMIN: if (data_in == 44) begin state <= EW;
                    longminout <= (217001 - longitude_minutes_out) / 16;
                end
        else begin
                if (data_in != 46)
                    longitude_minutes_out <=
                    longitude_minutes_out*10 + (data_in-48);
                count <= count - 1;
                state <= LONGMIN;
        end

EW:     if (data_in == 44) state <= FIX;
        else state <= EW;
```

```
        FIX:    if(data_in == 44) state <= SATELLITES;
                else begin fix <= data_in[0];
                        state <= FIX;
                        end


        SATELLITES:         if (data_in == 44) state <= HDOP;
                else state <= SATELLITES;

        HDOP:           if (data_in == 44) state <= ALT;
                else state <= HDOP;


        ALT:    if (data_in == 44) state <= UNITS;
                else state <= ALT;

        UNITS:          if (data_in == 44) state <= AGE;
                else state <= UNITS;

        AGE:    if (data_in == 44) begin
                                state <= IDLE;
                                position_ready <= 1;
                                end
                else state <= AGE;

        default: begin  state <= IDLE;
                        latitude_minutes_out <= 0;
                        longitude_minutes_out <= 0;
                 end

endcase
else
        position_ready <= 0;
end // always @ clk

endmodule
```

Wireless Reciever (VBScript)

```
Option Explicit

Dim strength ,cnt, countMax, xPosition, yPosition, str, cntI, doIt
Dim Sources()
Dim acObj
```

```
Set acObj = CreateObject( "ActivXperts.Comport" )
   acObj.Baudrate = 9600
   acObj.PortID = 1 'Comport can be changed here
   acObj.Open

SetDefaults

Sub SetDefaults
        cnt = 0
        cntI = 0
        countMax = 1
        xPosition = 0
        yPosition = 0
        str = 0
        doIt = 0
End Sub


Sub OnScanComplete (FoundNew, SeenBefore, LostContact, BestSNR)
        If cnt = 0 Then
                countMax = 0
        Else
                countMax = UBound(Sources,2)
        End If
        cnt = 0
        cntI = 0
        doIt = 1
        xPosition = 0
        yPosition = 0
        str = 0
        GetPosition

End Sub


Sub OnScanResult (SSID, BSSID, CapFlags, Signal, Noise, LastSeen)
        If cnt = 0 Then
                ReDim Sources(3,1)
        Else
                ReDim Preserve Sources(3,cnt+1)
        End If

        If IsNull(Signal) = False Then
                Select Case BSSID
```

```
                Case "00022D4B6C87" 'ee-wireless-08
                        Sources(0,cnt) = 4
                        Sources(1,cnt) = 21
                        Sources(2,cnt) = 148 - ((Asc(Mid(Signal,2,1)) - 48) * 10) -
Asc(Mid(Signal,3,1))
                        cnt = cnt + 1

                Case "00022D4B6C8D" 'ee-wireless-09
                        Sources(0,cnt) = 10
                        Sources(1,cnt) = 19
                        Sources(2,cnt) = 148 - ((Asc(Mid(Signal,2,1)) - 48) * 10) -
Asc(Mid(Signal,3,1))
                        cnt = cnt + 1

                Case Else
                        Sources(0,cnt) = cnt
                        Sources(1,cnt) = cnt
                        Sources(2,cnt) = 1

                End Select

        '       cnt = cnt + 1

'       Else
'               Sources(cnt,0) = cnt
'               Sources(cnt,1) = cnt
'               Sources(cnt,2) = 0
'               cnt = cnt + 1

        End If

        End Sub



Sub GetPosition
        If cntI = countMax Then
                SendToKit
        Else
                str = str + Sources(2,cntI)
                xPosition = xPosition + CInt(((Sources(0,cntI) - xPosition ) *
(Sources(2,cntI) / str)))
                yPosition = yPosition + CInt(((Sources(1,cntI) - yPosition ) *
(Sources(2,cntI) / str)))
                cntI = cntI + 1
                GetPosition
```

```
        End If


End Sub

Sub SendToKit
If doIt = 1 Then
doIt = 0
cntI = 0
cnt = 0
countMax  = 1
'MsgBox         "x = " & CInt(xPosition) & " and y= " & cInt(yPosition) & " signal
strength = " & str


        acObj.WriteByte 036
        acObj.WriteBytes xPosition
        'acObj.WriteByte 065 'building 34
        'acObj.WriteByte 058 'no building
        acObj.WriteByte 044
        acObj.WriteBytes yPosition
        'acObj.WriteByte 066 'building 34
        'acObj.WriteByte 075 'no building
        acObj.WriteByte 044
        End If
End Sub
```

Wireless Reciever

```
module wirelessreciever(clk, reset, data_ready, data_in, x_out, y_out, position_ready);

input clk, reset, data_ready;
input [7:0] data_in;

output [5:0] x_out, y_out;
output position_ready;

reg [1:0] state;
reg [5:0] x, y, x_out, y_out;
reg position_ready;

parameter IDLE = 0;
parameter XPOS = 1;
```

```
parameter YPOS = 2;

always @(posedge clk) begin

if (reset) begin
        state <= IDLE;
        x_out <= 0;
        y_out <= 0;
        x <= 0;
        y <= 0;
        position_ready <= 0;

        end

else if (data_ready)
        case(state)

        IDLE:  begin
                                x <= 0;
                                y <= 0;
                                position_ready <= 0;
                                if (data_in == 36) state <= XPOS;
                                else state <= IDLE;
                        end

        XPOS: if (data_in == 44) state <= YPOS;
                        else x <= x*10 + data_in - 48;

        YPOS: if (data_in == 44) begin
                                state <= IDLE;
                                if ((x == 0) && (y == 0)) position_ready <= 0;
                                else begin
                                        x_out <= x;
                                        y_out <= y;
                                        position_ready <= 1;
                                end
                        else y <= y*10 + data_in - 48;
        endcase

else
        position_ready <= 0;

end //always@ clk

endmodule
```

Indicator Code

```
//Major FSM used to control Identifier

module controlident(clock, reset, sample, info, t_mode, pointin,
pttosearch, closebuilding, ptresponse, pointout,
                              start1, busy1, start2, busy2, start3,
busy3, start4, busy4, start5, busy5, start0, busy0, state,
                              skipDetbuilding, devbuildingreg);

input clock, reset, sample, busy1, busy2, busy3, busy4, busy5, busy0;
input info, t_mode;
input [14:0] pointin;                 //in from Jon
input [5:0] ptresponse;                       //responses from all
memory modules
input [2:0] closebuilding;            //in from null object (outside)
memory module

output start1, start2, start3, start4, start5, start0;        //trigger
signals
output [11:0] pttosearch;                     //to memory modules
output [14:0] pointout;                       //to Justin
output [3:0] state;
output skipDetbuilding;
output [2:0] devbuildingreg;

reg start1, start2, start3, start4, start5, start0;
reg [2:0] dir;
reg [3:0] state, next;
reg [5:0] xreg, yreg, xin, yin;
reg [2:0] devbuilding, devbuildingreg, building, buildingreg;
reg skipDetbuilding;
reg [14:0] pointout_int, pointout;
reg [11:0] pttosearch;
reg [3:0] countpts;
reg [11:0] xyreg;
reg ce, resetcount;

parameter INITIAL=0;
parameter WAIT=1;
parameter CHECKMODE=2;
parameter GETNEXTPOINT=3;
parameter WAITFORXYREG=4;
parameter PREPSEARCH=5;
parameter STARTSEARCH=6;
parameter WAITSEARCH=7;
parameter DONESEARCH=8;
parameter STARTOUTSIDESEARCH=9;
parameter WAITOUTSIDESEARCH=10;
parameter DONEOUTSIDESEARCH=11;
parameter DETBUILDING=12;
parameter CHECKRESPONSE=13;
parameter DETSKIPBUILDING=14;
parameter DONEFORMAT=15;

parameter N=0;
```

```
parameter NE=1;
parameter E=2;
parameter SE=3;
parameter S=4;
parameter SW=5;
parameter W=6;
parameter NW=7;

parameter YMAX=45;
parameter XMAX=45;

always @ (posedge clock or posedge reset) begin
   if (reset) state<=INITIAL;
   else state<=next;

end


always @ (posedge clock or posedge reset) begin
   if (reset) begin xreg <=6'd0; yreg <= 6'd0; devbuildingreg <=
3'b000; skipDetbuilding <= 1'b0; end
   else begin

        //Parses input point
        dir <= pointin[14:12];
        xin <= pointin[11:6];
        yin <= pointin[5:0];

        //Handles info button press stuff
        if (ce) countpts <= countpts + 4'd1;
        else if (resetcount) countpts <= 4'd0;
        else countpts <= countpts;

        //Handles search point
        //if (!info) begin
        //      if (state==CHECKMODE) pttosearch <= pointin[11:0];
        //      else if (state==GETNEXTPOINT) pttosearch <= {xreg,
yreg};
        //      else pttosearch <= pttosearch;
        //      end
        //else pttosearch <= pointin[11:0];


        //Handles formatting of output point
        if (state==DONEFORMAT) begin
                if (t_mode | !info) pointout_int <={buildingreg,
pointin[11:0]};
                else pointout_int <= {3'b000, pointin[11:0]};
                end
        else pointout_int <= pointout_int;

        //Handles info button press stuff (skipping detbuilding state)
     if (state==DETBUILDING|state==DONEOUTSIDESEARCH) skipDetbuilding
<= 1'b1;
     else if (state==CHECKMODE) skipDetbuilding <= 1'b0;
     else skipDetbuilding <= skipDetbuilding;
```

```
        //Handles getting next point
        if (state==GETNEXTPOINT) begin
           case(dir)
                N: if (yreg!=6'd0) begin yreg <= yreg-6'd1; xreg <=
xreg; end
                else begin yreg <= yreg; xreg <= xreg; end
                NE: if (yreg!=6'd0 & xreg!=XMAX) begin yreg <= yreg-
6'd1; xreg <= xreg+6'd1; end
                else begin yreg <= yreg; xreg <= xreg; end
                E: if (xreg!=XMAX) begin yreg <= yreg; xreg <=
xreg+6'd1; end
                else begin yreg <= yreg; xreg <= xreg; end
                SE: if (yreg!=YMAX & xreg!=XMAX) begin yreg <=
yreg+6'd1; xreg <= xreg+6'd1; end
                        else begin yreg <= yreg; xreg <= xreg; end
                S: if (yreg!=YMAX) begin yreg <= yreg+1; xreg <= xreg;
end
                        else begin yreg <= yreg; xreg <= xreg; end
                SW: if (yreg!=YMAX & xreg!=6'd0) begin yreg <=
yreg+6'd1; xreg <= xreg-6'd1; end
                                else begin yreg <= yreg; xreg <= xreg;
end
                W: if (xreg!=6'd0) begin yreg <= yreg; xreg <= xreg-
6'd1; end
                        else begin yreg <= yreg; xreg <= xreg; end
                NW: if (yreg!=6'd0 & xreg!=6'd0) begin yreg <= yreg-
6'd1; xreg <= xreg-6'd1; end
                        else begin yreg <= yreg; xreg <= xreg; end
                default: begin yreg <= yreg; xreg <= xreg; end
                endcase

                pttosearch <= pttosearch;
                end
         else if (state==WAITFORXYREG) begin
                if (!info) pttosearch <= {xreg, yreg};
                else pttosearch <= pointin[11:0];
                end

         else if (state==CHECKMODE) begin
             xreg<=xin;
                yreg<=yin;
                pttosearch <= pointin[11:0];
                end
         else  pttosearch <= pttosearch;

         if (state==DETBUILDING)
            devbuildingreg <= 3'd0;
         else if (state==GETNEXTPOINT)
            devbuildingreg <= devbuilding;
       else devbuildingreg <= devbuildingreg;


        pointout <= pointout_int;
        buildingreg <= building;
        //devbuildingreg <= devbuilding;
        //ptresponse <= ptresponse_int;
        end
```

```
end

always @ (state or busy1 or busy2 or busy3 or busy4 or busy5 or busy0
or sample or ptresponse) begin
   resetcount=1'b0;    ce=1'b0;
   case(state)
       INITIAL:  begin resetcount=1'b1; start1=0; start2=0; start3=0;
start4=0; start5=0; start0=0; building=3'd0; next=WAIT; end

         WAIT: begin if (sample) begin resetcount=1'b1;
next=CHECKMODE; end
                     else next=WAIT;

                          start1=0; start2=0; start3=0; start4=0;
start5=0; start0=0;
                          end

         CHECKMODE: begin if (!info | t_mode)  next=PREPSEARCH;
                                  else next=DONEFORMAT;

                                start1=0; start2=0; start3=0; start4=0;
start5=0; start0=0;
                                end


         //USED TO GET NEXT LINE OF SIGHT COORDINATE
         //USED ONLY IN INFO MODE
         GETNEXTPOINT: begin
                               start1=0; start2=0; start3=0; start4=0;
start5=0; start0=0;
                                      next=WAITFORXYREG;

        WAITFORXYREG: begin   start1=0; start2=0; start3=0; start4=0;
start5=0; start0=0;
                                      next=PREPSEARCH;

         //MEMORY SEARCH (SEARCH THROUGH ALL BUILDINGS)
         PREPSEARCH: begin start1=0; start2=0; start3=0; start4=0;
start5=0; start0=0;
                                    if (busy1 | busy2 | busy3 | busy4 |
busy5) next=PREPSEARCH;
                                      else next=STARTSEARCH;    end

         STARTSEARCH: begin start1=1; start2=1; start3=1; start4=1;
start5=1; start0=0;
                                      if (!busy1 | !busy2 | !busy3 |
!busy4 | !busy5) next=STARTSEARCH;
                                      else next=WAITSEARCH;   end

         WAITSEARCH: begin if (busy1 & busy2 & busy3 & busy4 & busy5)
begin
                                                 start1=1; start2=1;
start3=1; start4=1; start5=1; start0=0; next=WAITSEARCH; end
                                           else begin start1=0;
start2=0; start3=0; start4=0; start5=0; start0=0; next=DONESEARCH; end
                               end
```

```
          DONESEARCH: begin start1=0; start2=0; start3=0; start4=0;
start5=0; start0=0;
                    if (ptresponse[1] | ptresponse[2] | ptresponse[3] |
ptresponse[4] | ptresponse[5])
                                  begin
                                        if (!info & !skipDetbuilding)
next=DETBUILDING;
                                        else next=CHECKRESPONSE;
                                  end
                              else next=STARTOUTSIDESEARCH;
                              end

          STARTOUTSIDESEARCH:  begin start1=0; start2=0; start3=0;
start4=0; start5=0; start0=1;
                                              if (!busy0)
next=STARTOUTSIDESEARCH;
                                              else
next=WAITOUTSIDESEARCH;
                                           end

           WAITOUTSIDESEARCH: if (busy0) begin start1=0; start2=0;
start3=0; start4=0; start5=0; start0=1; next=WAITOUTSIDESEARCH; end
                                            else begin start1=0; start2=0;
start3=0; start4=0; start5=0; start0=0;  next=DONEOUTSIDESEARCH; end

           DONEOUTSIDESEARCH:  if (!info) next=CHECKRESPONSE;
                                              else begin
building=closebuilding; next=DONEFORMAT; end

        //FIND BUILDING THAT DEVICE IS IN
        //USED ONLY IN INFOMODE
           DETBUILDING: begin start1=0; start2=0; start3=0; start4=0;
start5=0; start0=0;
                                         case (ptresponse)
                                             6'b000010: devbuilding=3'h1;
                                                6'b000100:
devbuilding=3'h2;
                                                6'b001000:
devbuilding=3'h3;
                                                6'b010000:
devbuilding=3'h4;
                                                6'b100000:
devbuilding=3'h5;
                                                default: devbuilding=3'h0;
                                           endcase

                                          ce=1; next=GETNEXTPOINT; end
        ////////////////

          CHECKRESPONSE: begin start1=0; start2=0; start3=0; start4=0;
start5=0; start0=0;
                                           case (ptresponse)
                                             6'b000010: building=3'h1;
                                                6'b000100: building=3'h2;
                                                6'b001000: building=3'h3;
                                                6'b010000: building=3'h4;
                                                6'b100000: building=3'h5;
```

```
                                            default: building=3'h0;
                                         endcase

                                        if (!info & countpts==4'd15)
begin building=3'h0; next=DONEFORMAT; end
                                        else if (!info &
(building==devbuildingreg | building==3'h0)) begin ce=1;
next=GETNEXTPOINT; end
                                        else next=DONEFORMAT;
                                end


          DONEFORMAT: begin  start1=0; start2=0; start3=0; start4=0;
start5=0; start0=0;
                                       next=WAIT;
                                       end

          default: begin next=INITIAL; building=3'd7; end
   endcase
end



endmodule
```

Identdivideren

```
module identdivideren(clock, reset, enable);
//Clock divider used for enable
//System clock is 27MHz. Need enable(1Hz)

input clock, reset;
output enable;

reg enable;
//reg enable_int;

reg [24:0]count1;

parameter Cycles1=25'd26999999;
//parameter Cycles2=15'd26999;

always @ (posedge clock or posedge reset)
begin

    if (reset) begin
                      count1<=25'd0;
                      enable <= 1'b0;
                      //enable_int <= enable;
                      end
    else if (count1==Cycles1) begin
                      count1<=25'd0;
                      enable <= 1'b1;
```

```
                        //enable_int <= enable;
                        end
    else begin count1<=count1+25'd1;
                        enable<=1'b0;
                                //enable_int <= enable;
                        end
end

endmodule
```

Ident Testbench

```
`timescale 1ns/10ps

module identtestbench2;

reg clk, reset;
reg info, t_mode;
reg enable;
reg sample;
reg [14:0] pointin;

wire [7:0] addr1, addr2, addr3, addr4, addr5;
wire [10:0] addr0;
wire [14:0] data0;
wire [11:0] data1, data2, data3, data4, data5;
wire start1, start2, start3, start4, start5;
wire foundpt1, foundpt2, foundpt3, foundpt4, foundpt5, foundpt0;
wire [5:0] ptresponse;
wire busy1, busy2, busy3, busy4, busy5;
wire [3:0] state1, state2, state3, state4, state5, state0;
wire [3:0] statec;
wire [11:0] ckpoint;
wire [2:0] closebuilding;
wire [14:0] pointout;
wire skipdet;
wire [2:0] devbuild;


building1 b1(addr1, clk, data1);
readobject rb1(clk, reset, start1, ckpoint, data1, addr1, foundpt1,
busy1, state1);
building2 b2(addr2, clk, data2);
readobject rb2(clk, reset, start2, ckpoint, data2, addr2, foundpt2,
busy2, state2);
building3 b3(addr3, clk, data3);
readobject rb3(clk, reset, start3, ckpoint, data3, addr3, foundpt3,
busy3, state3);
building4 b4(addr4, clk, data4);
readobject rb4(clk, reset, start4, ckpoint, data4, addr4, foundpt4,
busy4, state4);
building5 b5(addr5, clk, data5);
```

```
readobject rb5(clk, reset, start5, ckpoint, data5, addr5, foundpt5,
busy5, state5);
readoutside rb0(clk, reset, start0, ckpoint, data0, closebuilding,
addr0, foundpt0, busy0, state0);
outside b0(addr0, clk, data0);


assign ptresponse={foundpt5, foundpt4, foundpt3, foundpt2, foundpt1,
foundpt0};


controlident topc(clk, reset, sample, info, t_mode, pointin, ckpoint,
closebuilding, ptresponse, pointout,
                        start1, busy1, start2, busy2, start3, busy3,
start4, busy4, start5, busy5, start0, busy0, statec,
                        skipdet, devbuild);

initial
  begin
        clk=1'b0;
        reset=1'b0;
        pointin=15'b011000111010111;
        info=1'b1;
        t_mode=1'b0;
        sample=1'b0;
  end

always
        #25 clk=~clk;

initial
  begin
        #50 reset=1'b1;
        #1000 enable=1'b1;
        #100 enable=1'b0;
        #1000 enable=1'b1;
        #100 enable=1'b0;
        #100 reset=1'b0;

        #100 sample=1'b1; info=1'b0;
        #50 sample=1'b0;

  end

 endmodule



//Module holds info pulse for 10 seconds

module inforeg(clock, reset, info, startcount, stophold, inforeg);

input clock, reset, info, stophold;
output inforeg, startcount;

reg inforeg, startcount;
```

```
always @ (posedge clock or posedge reset) begin
   if (reset) begin inforeg<=1'b1;    startcount<=1'b0; end
   else if (!info) begin inforeg<=1'b0; startcount<=1'b1; end
   else if (stophold) begin inforeg<=1'b1; startcount<=1'b0; end
   else begin inforeg<=inforeg; startcount<=startcount; end

end

endmodule



//Timer for info hold register

module infotimer(clock, reset, enable, start, expire);

input clock, reset, enable, start;
output expire;
//output [3:0] count;

reg [3:0]count;
reg expire;
//reg enable_int;

//always @ (posedge clock)
//  enable_int<=enable;

//always @ (posedge enable_int or posedge start)
//begin
//    if (reset) begin count<=4'd0; expire<=1'b0; end
//    else if (start & count<4'd10) begin count<=count+4'd1;
expire<=1'b0;  end
//    else if (start & count==4'd10) begin count<=4'd0; expire<=1'b1;
end
//    else begin count<=4'd0; expire<=1'b0; end
//end

always @ (posedge clock or posedge reset) begin
        if (reset) begin count<=4'd0; expire<=1'b0; end
        else if (start & enable & count<4'd10) begin count<=count+4'd1;
expire<=1'b0; end
    else if (start & enable & count==4'd10) begin count<=4'd0;
expire<=1'b1; end
    else begin count<=count; expire<=1'b0; end
end



endmodule
```

Readobject

```
//Minor FSM interface to building ROMs

module readobject(clock, reset, start, ckpoint, point, romaddr,
ptfound, busy, state);

input clock, reset, start;
input [11:0]ckpoint, point;

output busy;
output [7:0]romaddr;
//output ptfound_int;
output ptfound;
output [3:0] state;

reg [3:0] state, next;
reg [7:0] romaddr;
reg busy, busy_int, ce;
reg ptfound, ptfound_int;
reg resetaddr;

parameter INITIAL=0;
parameter WAITING=1;
parameter LDADDRESS=2;
parameter WAITLOAD=3;
parameter READCHECKPT=4;
parameter SENDGOODRESPONSE=5;
parameter SENDBADRESPONSE=6;
parameter STOPPED=7;
parameter STOPPED2=8;

always @ (posedge clock or posedge reset) begin
   if (reset) begin state<=INITIAL; romaddr<=8'd0; end
   else if (ce) begin state<=next; romaddr<=romaddr+8'd1; end
   else if (resetaddr) begin state<=next; romaddr<=8'd0; end
   else begin state<=next; romaddr<=romaddr; end


end

always @ (posedge clock) begin
   busy <= busy_int;
   //ptfound_int <= ptfound;
   if (state==SENDGOODRESPONSE) ptfound<=1'b1;
   else if (state==SENDBADRESPONSE | state==LDADDRESS) ptfound<=1'b0;
   else ptfound<=ptfound;

end

always @ (state or start or ckpoint) begin
   resetaddr=1'b0;
   case (state)
     INITIAL:   begin busy_int=0; ce=0;  next=WAITING; end

     WAITING:   begin resetaddr=1'b1; busy_int=0; ce=0;
```

```
                                if (start) next=LDADDRESS;
                        else next=WAITING;
                            end

     LDADDRESS: begin  busy_int=1; ce=0;
                                  if (!start) next=STOPPED;
                                  else next=WAITLOAD;
                                 end

     WAITLOAD: begin busy_int=1; ce=0; next=READCHECKPT; end

     READCHECKPT: begin busy_int=1; ce=1;
                         if (point==ckpoint) next=SENDGOODRESPONSE;
                                 else if (romaddr==8'd255)
next=SENDBADRESPONSE;
                         else next=LDADDRESS;     end

        SENDBADRESPONSE: begin busy_int=0; ce=0;  next=STOPPED; end

     SENDGOODRESPONSE: begin busy_int=0; ce=0; next=STOPPED; end


        STOPPED: begin busy_int=0; ce=0; next=STOPPED2; end

        STOPPED2: begin busy_int=0; ce=0;  next=WAITING; end

     default: next=INITIAL;
    endcase
end


endmodule
```

Readoutside

```
//Minor FSM interface to building ROMs

module readoutside(clock, reset, start, ckpoint, point, closebuilding,
romaddr, ptfound, busy, state);

input clock, reset, start;
input [11:0]ckpoint;          //from major fsm
input [14:0] point;           //from ROM

output busy;                  //to major fsm
output [2:0] closebuilding;   //to major fsm
output [10:0]romaddr;         //to ROM
output ptfound;               //to major fsm
output [3:0] state;

reg [3:0] state, next;
reg [10:0] romaddr;
```

```
reg busy, busy_int, ce;
reg ptfound, ptfound_int;
reg resetaddr;
reg [2:0] closebuilding, closebuilding_int;

parameter INITIAL=0;
parameter WAITING=1;
parameter LDADDRESS=2;
parameter WAITLOAD=3;
parameter READCHECKPT=4;
parameter SENDGOODRESPONSE=5;
parameter SENDBADRESPONSE=6;
parameter STOPPED=7;
parameter STOPPED2=8;

always @ (posedge clock or posedge reset) begin
   if (reset) begin state<=INITIAL; romaddr<=11'd0; end
   else if (ce) begin state<=next; romaddr<=romaddr+11'd1; end
   else if (resetaddr) begin state<=next; romaddr<=11'd0; end
   else begin state<=next; romaddr<=romaddr; end

end

always @ (posedge clock) begin
   busy<=busy_int;
   //ptfound_int <= ptfound;
   closebuilding <= closebuilding_int;

    if (state==SENDGOODRESPONSE) ptfound<=1'b1;
    //else if (state==SENDBADRESPONSE | state==LDADDRESS)
ptfound<=1'b0;
    else ptfound<=1'b0;

end

always @ (state or start) begin
   resetaddr=1'b0;
   case (state)
     INITIAL:   begin busy_int=0; ce=0; next=WAITING; end

     WAITING:   begin busy_int=0; ce=0; resetaddr=1'b1;
                                if (start)  next=LDADDRESS;
              else next=WAITING;
                end

     LDADDRESS: begin busy_int=1; ce=0;
                                if (!start) next=STOPPED;
                                else  next=WAITLOAD;
                      end

     WAITLOAD: begin busy_int=1; ce=0; next=READCHECKPT; end

     READCHECKPT: begin busy_int=1; ce=1;
                  if (point[11:0]==ckpoint) begin
closebuilding_int=point[14:12];
                                                                    next=SENDGO
```

```
                          else if (romaddr==11'd2047)
next=SENDBADRESPONSE;
                  else next=LDADDRESS;     end

     SENDGOODRESPONSE: begin busy_int=0; ce=0;  next=STOPPED; end

          SENDBADRESPONSE: begin busy_int=0; ce=0;  next=STOPPED; end

        STOPPED: begin busy_int=0; ce=0; next=STOPPED2; end

        STOPPED2: begin busy_int=0; ce=0; next=WAITING; end

     default: next=INITIAL;
    endcase
end


endmodule
```

Top Identify

```
//Top module for identifier
module topidentify(clk, reset, info, t_mode, pointin_int, pointout,
statec, state0, romaddr0, searchpt,
                    response1, response2, response3, response4,
response5, response0, memresponses, closebuilding,
                           skipdetstate, devbuilding);

input clk, reset, info, t_mode;
input [15:0] pointin_int;

output [14:0] pointout;
output [3:0] statec;
output [3:0] state0;
output [10:0] romaddr0;
output [11:0] searchpt;
output response1, response2, response3, response4, response5,
response0;
output [5:0] memresponses;
output [2:0] closebuilding;
output skipdetstate;
output [2:0] devbuilding;

wire [14:0] pointout;
wire [3:0] statec;
wire [3:0] state1, state2, state3, state4, state5, state0;

reg [14:0] pointin;

wire reset_sync, t_modesync, info_sync, enable, sample;
wire start, expire, info_held;
wire [11:0] searchpt;
```

```
wire [2:0] closebuilding;
reg [5:0] memresponses;
wire start1, busy1, start2, busy2, start3, busy3, start4, busy4,
start5, busy5, start0, busy0;
wire [11:0] mempoint1, mempoint2, mempoint3, mempoint4, mempoint5;
wire [14:0] mempoint0;
wire [7:0] romaddr1, romaddr2, romaddr3, romaddr4, romaddr5;
wire [10:0] romaddr0;
wire response0, response1, response2, response3, response4, response5;
wire skipdetstate;
wire [2:0] devbuilding;

always @ (pointin_int) begin
   case (pointin_int[15:12])
         4'b1101: pointin={3'b000, pointin_int[11:0]};
         4'b1001: pointin={3'b001, pointin_int[11:0]};
         4'b1011: pointin={3'b010, pointin_int[11:0]};
         4'b0011: pointin={3'b011, pointin_int[11:0]};
         4'b0111: pointin={3'b100, pointin_int[11:0]};
         4'b0110: pointin={3'b101, pointin_int[11:0]};
         4'b1110: pointin={3'b110, pointin_int[11:0]};
         4'b1100: pointin={3'b111, pointin_int[11:0]};
         default: pointin={3'b000, pointin_int[11:0]};
   endcase
end

identdivideren iendiv(clk, reset_sync, enable);
identdividersamp isampdiv(clk, reset_sync, sample);
syncident isync(clk, reset, t_mode, info, reset_sync, t_modesync,
info_sync);

inforeg ireg(clk, reset_sync, info_sync, start, expire, info_held);
infotimer itimer(clk, reset_sync, enable, start, expire);

controlident icontrol(clk, reset_sync, sample, info_held, t_modesync,
pointin, searchpt,
                                closebuilding, memresponses, pointout,
                                start1, busy1, start2, busy2, start3,
busy3, start4, busy4, start5, busy5, start0, busy0, statec,
                                skipdetbstate, devbuilding);

readobject b1(clk, reset_sync, start1, searchpt, mempoint1, romaddr1,
response1, busy1, state1);
readobject b2(clk, reset_sync, start2, searchpt, mempoint2, romaddr2,
response2, busy2, state2);
readobject b3(clk, reset_sync, start3, searchpt, mempoint3, romaddr3,
response3, busy3, state3);
readobject b4(clk, reset_sync, start4, searchpt, mempoint4, romaddr4,
response4, busy4, state4);
readobject b5(clk, reset_sync, start5, searchpt, mempoint5, romaddr5,
response5, busy5, state5);
readoutside b0(clk, reset_sync, start0, searchpt, mempoint0,
closebuilding, romaddr0, response0, busy0, state0);

building1 b1rom(romaddr1, clk, mempoint1);
building2 b2rom(romaddr2, clk, mempoint2);
building3 b3rom(romaddr3, clk, mempoint3);
```

```
building4 b4rom(romaddr4, clk, mempoint4);
building5 b5rom(romaddr5, clk, mempoint5);
outside outrom(romaddr0, clk, mempoint0);

always @ (posedge clk)
   memresponses <= {response5, response4, response3, response2,
response1, response0};


endmodule
```

Video Code

Next page…

```
module videov3 (reset, clock_27mhz, x, y, building, 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);

        input reset;
        input clock_27mhz;
        input [5:0] x, y;
        input [5:0] building;

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

        reg [5:0] x_coord, y_coord;
        reg [5:0] building_num;
                                                        // 6-bit building #!

        wire vga_out_pixel_clock;
        wire [5:0] data;
        wire data24, data26, data34, data36, data38;
                                                                        // 9
bits per pixel?!
        wire [14:0] address, address24, address26, address34, address36, address38;
//      wire [7:0] red, green, blue;

        displayv3 vga_output (reset, clock_27mhz, x_coord, y_coord, building_num,
                                data, data24, data26, data34, data36, data38,
                                address, address24, address26, address34,
address36, address38,
                                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);
//      bldglookup info (reset, vga_out_pixel_clock, building_num, data38, address38,
red, green, blue);
        mitmap fullmap (address, vga_out_pixel_clock, data);
        bldg24 info24 (address24, vga_out_pixel_clock, data24);
        bldg26 info26 (address26, vga_out_pixel_clock, data26);
        bldg34 info34 (address34, vga_out_pixel_clock, data34);
        bldg36 info36 (address36, vga_out_pixel_clock, data36);
        bldg38 info38 (address38, vga_out_pixel_clock, data38);

        always @(x)
                begin
```

```
                x_coord <= x;
        end

    always @(y)
        begin
                y_coord <= y;
        end

    always @(building)
        begin
                building_num <= building;
        end

endmodule
```

```
// Use smaller map and allow for showing building information
module displayv3 (reset, clock_27mhz, x_coord, y_coord, building,
                                  next_pixel, data24, data26, data34, data36, data38,
                                  address, address24, address26, address34, address36,
address38,
                                  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);

        input reset; // Active high reset, synchronous with 27MHz clock
        input clock_27mhz; // 27MHz input clock
        input [5:0] x_coord, y_coord;
        input [5:0] building;
        input [5:0] next_pixel;
                                  // Using 6 bits for building #
        input data24, data26, data34, data36, data38;
//      input [7:0] red, green, blue;

//      input [8:0] building info;

        output [14:0] address, address24, address26, address34, address36, address38;
        output [7:0] vga_out_red, vga_out_green, vga_out_blue; // Outputs to DAC
        output vga_out_sync_b, vga_out_blank_b; // Composite sync/blank outputs to
DAC
        output vga_out_pixel_clock; // Pixel clock for DAC
        output vga_out_hsync, vga_out_vsync; // Sync outputs to VGA connector

        reg [14:0] address, address24, address26, address34, address36, address38;


        ////////////////////////////////////////////////////////////////////////
        //
        // Timing values
        //
        ////////////////////////////////////////////////////////////////////////

        // 800 X 600 @ 60Hz with a 40.000MHz pixel clock
`define H_ACTIVE              800 // pixels
`define H_FRONT_PORCH          40 // pixels
`define H_SYNCH               128 // pixels
`define H_BACK_PORCH           88 // pixels
`define H_TOTAL               1056 // pixels

`define V_ACTIVE              600 // lines
`define V_FRONT_PORCH           1 // lines
`define V_SYNCH                 4 // lines
```

```
`define V_BACK_PORCH      23 // lines
`define V_TOTAL           628 // lines

        //////////////////////////////////////////////////////////////////////
        //
        // Internal signals
        //
        //////////////////////////////////////////////////////////////////////

        wire pixel_clock;
        reg prst, pixel_reset; // Active high reset, synchronous with pixel clock

        reg [7:0] vga_out_red, vga_out_blue, vga_out_green;
        wire vga_out_sync_b, vga_out_blank_b;
        reg hsync1, hsync2, vga_out_hsync, vsync1, vsync2, vga_out_vsync;

        reg [11:0] pixel_count; // Counts pixels in each line
        reg [10:0] line_count; // Counts lines in each frame

        //////////////////////////////////////////////////////////////////////
        //
        // Generate the pixel clock (40.000MHz)
        //
        //////////////////////////////////////////////////////////////////////

        // synthesis attribute period of clock_27mhz is 37ns;

        DCM vga_dcm (.CLKIN(clock_27mhz),
                                .RST(1'b0),
                                .CLKFX(pixel_clock));
        // synthesis attribute DLL_FREQUENCY_MODE of vga_dcm is "LOW"
        // synthesis attribute DUTY_CYCLE_CORRECTION of vga_dcm is "TRUE"
        // synthesis attribute STARTUP_WAIT of vga_dcm is "TRUE"
        // synthesis attribute DFS_FREQUENCY_MODE of vga_dcm is "LOW"
        // synthesis attribute CLKFX_DIVIDE of vga_dcm is 13.5
        // synthesis attribute CLKFX_MULTIPLY of vga_dcm is 20
        // synthesis attribute CLK_FEEDBACK of vga_dcm  is "1X"
        // synthesis attribute CLKOUT_PHASE_SHIFT of vga_dcm is "NONE"
        // synthesis attribute PHASE_SHIFT of vga_dcm is 0
        // synthesis attribute clkin_period of vga_dcm is "37.04ns"

        assign vga_out_pixel_clock = ~pixel_clock;

        always @(posedge pixel_clock)
                begin
                        prst <= reset;
```

```
		pixel_reset <= prst;
	end

////////////////////////////////////////////////////////////////////////
//
// Pixel and Line Counters
//
////////////////////////////////////////////////////////////////////////

always @(posedge pixel_clock)
	if (pixel_reset)
		begin
			pixel_count <= 0;
			line_count <= 0;
			address <= 0;
					// address management
			address24 <= 0;
			address26 <= 0;
			address34 <= 0;
			address36 <= 0;
			address38 <= 0;
		end
	else if (pixel_count == (`H_TOTAL-1)) // last pixel in the line
		begin
			pixel_count <= 0;
			address <= ((line_count + 1) * 190);
			// modified address management
			if (line_count >= 164)
				begin
					address24 <= ((line_count - 163) * 256);
					address26 <= ((line_count - 163) * 256);
					address34 <= ((line_count - 163) * 256);
					address36 <= ((line_count - 163) * 256);
					address38 <= ((line_count - 163) * 256);
				end
			if (line_count == (`V_TOTAL-1)) // last line of the frame
				line_count <= 0;
			else
				line_count <= line_count + 1;
		end
	else
		begin
			pixel_count <= pixel_count + 1;
			address <= address + 1;
					// address management
			if (line_count >= 164)
```

```
                                begin
                                        address24 <= address38 + 1;
                                        address26 <= address38 + 1;
                                        address34 <= address38 + 1;
                                        address36 <= address38 + 1;
                                        address38 <= address38 + 1;
                                end
                        end

        ////////////////////////////////////////////////////////////////////////
        //
        // Sync and Blank Signals
        //
        ////////////////////////////////////////////////////////////////////////

        always @ (posedge pixel_clock)
                begin
                        if (pixel_reset)
                                begin
                                        hsync1 <= 1;
                                        hsync2 <= 1;
                                        vga_out_hsync <= 1;
                                        vsync1 <= 1;
                                        vsync2 <= 1;
                                        vga_out_vsync <= 1;
                                end
                        else
                                begin

                                        // Horizontal sync
                                        if (pixel_count ==
(`H_ACTIVE+`H_FRONT_PORCH))
                                                hsync1 <= 0; // start of h_sync
                                        else if (pixel_count ==
(`H_ACTIVE+`H_FRONT_PORCH+`H_SYNCH))
                                                hsync1 <= 1; // end of h_sync

                                        // Vertical sync
                                        if (pixel_count == (`H_TOTAL-1))
                                                begin
                                                        if (line_count ==
(`V_ACTIVE+`V_FRONT_PORCH))
                                                                vsync1 <= 0; // start of
v_sync
                                                        else if (line_count ==
(`V_ACTIVE+`V_FRONT_PORCH+`V_SYNCH))
```

```
                                                vsync1 <= 1; // end of v_sync
                                        end

                                end

                        // Delay hsync and vsync by two cycles to compensate for 2 cycles
of
                        // pipeline delay in the DAC.
                        hsync2 <= hsync1;
                        vga_out_hsync <= hsync2;
                        vsync2 <= vsync1;
                        vga_out_vsync <= vsync2;

                end

        // Blanking
        assign vga_out_blank_b = ((pixel_count<`H_ACTIVE) &
(line_count<`V_ACTIVE));

        // Composite sync
        assign vga_out_sync_b = hsync1 ^ vsync1;

        ////////////////////////////////////////////////////////////////////////
        //
        // Generate the display
        //
        ////////////////////////////////////////////////////////////////////////

        reg [15:0] frame_count;

        always @(posedge pixel_clock)
                if (pixel_reset)
                        frame_count <= 0;
                else if ((pixel_count == `H_TOTAL-1) && (line_count == `V_TOTAL-
1))
                        frame_count <= frame_count + 1;

        always @ (posedge pixel_clock)
                begin

                        if (((pixel_count >= (((x_coord + 15) * 3) - 3)) && (pixel_count
<= (((x_coord + 15) * 3) + 3)))
                                && ((line_count >= ((y_coord * 3) - 3)) && (line_count
<= ((y_coord * 3) + 3))))
                                begin
                                        vga_out_red <= 0;
```

```
                        vga_out_green <= 255;
                        vga_out_blue <= 0;
                    end
                else
                    begin
                        if ((next_pixel == 0 || pixel_count >= 189) &&
(line_count < 164))         // Modified for coloring black after edge of small map
                            begin
                                vga_out_red <= 0;
                                vga_out_green <= 0;
                                vga_out_blue <= 0;
                            end


////////////////////////// Begin building info management \\\\\\\\\\\\\\\\\\\\\\\\\

                        else if (line_count >= 164)
                            begin
                                if (pixel_count >= 255 || line_count
>= 262)
                                    begin
                                        vga_out_red <= 0;
                                        vga_out_green <= 0;
                                        vga_out_blue <= 0;
                                    end
                                else
                                    begin
                                        if (building == 24)
                                            begin
                                                if
(data24 == 0)

        begin

        vga_out_red <= 0;

        vga_out_green <= 0;

        vga_out_blue <= 0;

        end
                                                else if
(data24 == 1)

        begin
```

```
            vga_out_red <= 255;
            vga_out_green <= 255;
            vga_out_blue <= 255;
            end
                                                end
                                        else    if (building ==
26)
                                                begin
                                                        if
(data26 == 0)
            begin
            vga_out_red <= 0;
            vga_out_green <= 0;
            vga_out_blue <= 0;
            end
                                                        else if
(data26 == 1)
            begin
            vga_out_red <= 255;
            vga_out_green <= 255;
            vga_out_blue <= 255;
            end
                                                end
                                        else    if (building ==
34)
                                                begin
                                                        if
(data34 == 0)
            begin
            vga_out_red <= 0;
```

```
            vga_out_green <= 0;
            vga_out_blue <= 0;
            end
                                                                    else if
(data34 == 1)
            begin
            vga_out_red <= 255;
            vga_out_green <= 255;
            vga_out_blue <= 255;
            end
                                                                end
                                                        else    if (building ==
36)
                                                                begin
                                                                    if
(data36 == 0)
            begin
            vga_out_red <= 0;
            vga_out_green <= 0;
            vga_out_blue <= 0;
            end
                                                                    else if
(data36 == 1)
            begin
            vga_out_red <= 255;
            vga_out_green <= 255;
            vga_out_blue <= 255;
            end
```

```
                                                                    end
                                                                else    if (building ==
38)
                                                                    begin
                                                                        if
(data38 == 0)
        begin
        vga_out_red <= 0;
        vga_out_green <= 0;
        vga_out_blue <= 0;
        end
                                                                        else if
(data38 == 1)
        begin
        vga_out_red <= 255;
        vga_out_green <= 255;
        vga_out_blue <= 255;
        end
                                                                    end
                                                                else
                                                                    begin
        vga_out_red <= 0;
        vga_out_green <= 0;
        vga_out_blue <= 0;
                                                                    end
                                                                //vga_out_red <= red;
                // Check how building info image is stored in ROM
                                                                //vga_out_green <=
green;
                                                                //vga_out_blue <=
blue;
                                                            end
                                                    end
```

```
//////////////////////  End building info management  \\\\\\\\\\\\\\\\\\\\\\\\

                                        else if (next_pixel == building)
                                                begin
                                                        vga_out_red <= 255;
                                                        vga_out_green <= 0;
                                                        vga_out_blue <= 0;
                                                end
                                        else
                                                begin
                                                        vga_out_red <= 0;
                                                        vga_out_green <= 0;
                                                        vga_out_blue <= 255;
                                                end
                                end
                        end

endmodule
```

// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

```
// You must compile the wrapper file mitmap.v when simulating
// the core, mitmap. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Guide".

module mitmap (
        addr,
        clk,
        dout);    // synthesis black_box

input [14 : 0] addr;
input clk;
output [5 : 0] dout;

// synopsys translate_off

        BLKMEMSP_V6_1 #(
                15,       // c_addr_width
                "0",      // c_default_data
                32768, // c_depth
                0,        // c_enable_rlocs
                0,        // c_has_default_data
                0,        // c_has_din
                0,        // c_has_en
                0,        // c_has_limit_data_pitch
                0,        // c_has_nd
                0,        // c_has_rdy
                0,        // c_has_rfd
                0,        // c_has_sinit
                0,        // c_has_we
                18,       // c_limit_data_pitch
                "mitmap.mif", // c_mem_init_file
                0,        // c_pipe_stages
                0,        // c_reg_inputs
                "0",      // c_sinit_value
                6,        // c_width
                0,        // c_write_mode
                "0",      // c_ybottom_addr
                1,        // c_yclk_is_rising
                1,        // c_yen_is_high
                "hierarchy1",   // c_yhierarchy
                0,        // c_ymake_bmm
                "16kx1",          // c_yprimitive_type
                1,        // c_ysinit_is_high
                "1024",           // c_ytop_addr
                0,        // c_yuse_single_primitive
```

```
                1,      // c_ywe_is_high
                1)      // c_yydisable_warnings
        inst (
                .ADDR(addr),
                .CLK(clk),
                .DOUT(dout),
                .DIN(),
                .EN(),
                .ND(),
                .RFD(),
                .RDY(),
                .SINIT(),
                .WE());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of mitmap is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of mitmap is "black_box"

endmodule
```

// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

```
// You must compile the wrapper file bldg24.v when simulating
// the core, bldg24. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Guide".

module bldg24 (
        addr,
        clk,
        dout);    // synthesis black_box

input [14 : 0] addr;
input clk;
output [0 : 0] dout;

// synopsys translate_off

        BLKMEMSP_V6_1 #(
                15,       // c_addr_width
                "0",      // c_default_data
                32768, // c_depth
                0,        // c_enable_rlocs
                0,        // c_has_default_data
                0,        // c_has_din
                0,        // c_has_en
                0,        // c_has_limit_data_pitch
                0,        // c_has_nd
                0,        // c_has_rdy
                0,        // c_has_rfd
                0,        // c_has_sinit
                0,        // c_has_we
                18,       // c_limit_data_pitch
                "bldg24.mif",  // c_mem_init_file
                0,        // c_pipe_stages
                0,        // c_reg_inputs
                "0",      // c_sinit_value
                1,        // c_width
                0,        // c_write_mode
                "0",      // c_ybottom_addr
                1,        // c_yclk_is_rising
                1,        // c_yen_is_high
                "hierarchy1",   // c_yhierarchy
                0,        // c_ymake_bmm
                "16kx1",          // c_yprimitive_type
                1,        // c_ysinit_is_high
                "1024",           // c_ytop_addr
                0,        // c_yuse_single_primitive
```

```
                1,      // c_ywe_is_high
                1)      // c_yydisable_warnings
        inst (
                .ADDR(addr),
                .CLK(clk),
                .DOUT(dout),
                .DIN(),
                .EN(),
                .ND(),
                .RFD(),
                .RDY(),
                .SINIT(),
                .WE());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of bldg24 is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of bldg24 is "black_box"

endmodule
```

// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

```
// You must compile the wrapper file bldg26.v when simulating
// the core, bldg26. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Guide".

module bldg26 (
        addr,
        clk,
        dout);    // synthesis black_box

input [14 : 0] addr;
input clk;
output [0 : 0] dout;

// synopsys translate_off

        BLKMEMSP_V6_1 #(
                15,       // c_addr_width
                "0",      // c_default_data
                32768, // c_depth
                0,        // c_enable_rlocs
                0,        // c_has_default_data
                0,        // c_has_din
                0,        // c_has_en
                0,        // c_has_limit_data_pitch
                0,        // c_has_nd
                0,        // c_has_rdy
                0,        // c_has_rfd
                0,        // c_has_sinit
                0,        // c_has_we
                18,       // c_limit_data_pitch
                "bldg26.mif",  // c_mem_init_file
                0,        // c_pipe_stages
                0,        // c_reg_inputs
                "0",      // c_sinit_value
                1,        // c_width
                0,        // c_write_mode
                "0",      // c_ybottom_addr
                1,        // c_yclk_is_rising
                1,        // c_yen_is_high
                "hierarchy1",   // c_yhierarchy
                0,        // c_ymake_bmm
                "16kx1",          // c_yprimitive_type
                1,        // c_ysinit_is_high
                "1024",           // c_ytop_addr
                0,        // c_yuse_single_primitive
```

```
                1,      // c_ywe_is_high
                1)      // c_yydisable_warnings
        inst (
                .ADDR(addr),
                .CLK(clk),
                .DOUT(dout),
                .DIN(),
                .EN(),
                .ND(),
                .RFD(),
                .RDY(),
                .SINIT(),
                .WE());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of bldg26 is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of bldg26 is "black_box"

endmodule
```

// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

```
// You must compile the wrapper file bldg34.v when simulating
// the core, bldg34. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Guide".

module bldg34 (
        addr,
        clk,
        dout);    // synthesis black_box

input [14 : 0] addr;
input clk;
output [0 : 0] dout;

// synopsys translate_off

        BLKMEMSP_V6_1 #(
                15,     // c_addr_width
                "0",    // c_default_data
                32768, // c_depth
                0,      // c_enable_rlocs
                0,      // c_has_default_data
                0,      // c_has_din
                0,      // c_has_en
                0,      // c_has_limit_data_pitch
                0,      // c_has_nd
                0,      // c_has_rdy
                0,      // c_has_rfd
                0,      // c_has_sinit
                0,      // c_has_we
                18,     // c_limit_data_pitch
                "bldg34.mif",  // c_mem_init_file
                0,      // c_pipe_stages
                0,      // c_reg_inputs
                "0",    // c_sinit_value
                1,      // c_width
                0,      // c_write_mode
                "0",    // c_ybottom_addr
                1,      // c_yclk_is_rising
                1,      // c_yen_is_high
                "hierarchy1",   // c_yhierarchy
                0,      // c_ymake_bmm
                "16kx1",        // c_yprimitive_type
                1,      // c_ysinit_is_high
                "1024",         // c_ytop_addr
                0,      // c_yuse_single_primitive
```

```
                1,      // c_ywe_is_high
                1)      // c_yydisable_warnings
        inst (
                .ADDR(addr),
                .CLK(clk),
                .DOUT(dout),
                .DIN(),
                .EN(),
                .ND(),
                .RFD(),
                .RDY(),
                .SINIT(),
                .WE());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of bldg34 is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of bldg34 is "black_box"

endmodule
```

// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

```
// You must compile the wrapper file bldg36.v when simulating
// the core, bldg36. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Guide".

module bldg36 (
        addr,
        clk,
        dout);    // synthesis black_box

input [14 : 0] addr;
input clk;
output [0 : 0] dout;

// synopsys translate_off

        BLKMEMSP_V6_1 #(
                15,       // c_addr_width
                "0",      // c_default_data
                32768, // c_depth
                0,        // c_enable_rlocs
                0,        // c_has_default_data
                0,        // c_has_din
                0,        // c_has_en
                0,        // c_has_limit_data_pitch
                0,        // c_has_nd
                0,        // c_has_rdy
                0,        // c_has_rfd
                0,        // c_has_sinit
                0,        // c_has_we
                18,       // c_limit_data_pitch
                "bldg36.mif",  // c_mem_init_file
                0,        // c_pipe_stages
                0,        // c_reg_inputs
                "0",      // c_sinit_value
                1,        // c_width
                0,        // c_write_mode
                "0",      // c_ybottom_addr
                1,        // c_yclk_is_rising
                1,        // c_yen_is_high
                "hierarchy1",   // c_yhierarchy
                0,        // c_ymake_bmm
                "16kx1",          // c_yprimitive_type
                1,        // c_ysinit_is_high
                "1024",           // c_ytop_addr
                0,        // c_yuse_single_primitive
```

```
                1,      // c_ywe_is_high
                1)      // c_yydisable_warnings
        inst (
                .ADDR(addr),
                .CLK(clk),
                .DOUT(dout),
                .DIN(),
                .EN(),
                .ND(),
                .RFD(),
                .RDY(),
                .SINIT(),
                .WE());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of bldg36 is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of bldg36 is "black_box"

endmodule
```

// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity synthesis
// tools. Ensure they are correct for your synthesis tool(s).

```
// You must compile the wrapper file bldg38.v when simulating
// the core, bldg38. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Guide".

module bldg38 (
        addr,
        clk,
        dout);    // synthesis black_box

input [14 : 0] addr;
input clk;
output [0 : 0] dout;

// synopsys translate_off

        BLKMEMSP_V6_1 #(
                15,       // c_addr_width
                "0",      // c_default_data
                32768, // c_depth
                0,        // c_enable_rlocs
                0,        // c_has_default_data
                0,        // c_has_din
                0,        // c_has_en
                0,        // c_has_limit_data_pitch
                0,        // c_has_nd
                0,        // c_has_rdy
                0,        // c_has_rfd
                0,        // c_has_sinit
                0,        // c_has_we
                18,       // c_limit_data_pitch
                "bldg38.mif",  // c_mem_init_file
                0,        // c_pipe_stages
                0,        // c_reg_inputs
                "0",      // c_sinit_value
                1,        // c_width
                0,        // c_write_mode
                "0",      // c_ybottom_addr
                1,        // c_yclk_is_rising
                1,        // c_yen_is_high
                "hierarchy1",   // c_yhierarchy
                0,        // c_ymake_bmm
                "16kx1",          // c_yprimitive_type
                1,        // c_ysinit_is_high
                "1024",           // c_ytop_addr
                0,        // c_yuse_single_primitive
```

```
                1,      // c_ywe_is_high
                1)      // c_yydisable_warnings
        inst (
                .ADDR(addr),
                .CLK(clk),
                .DOUT(dout),
                .DIN(),
                .EN(),
                .ND(),
                .RFD(),
                .RDY(),
                .SINIT(),
                .WE());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of bldg38 is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of bldg38 is "black_box"

endmodule
```