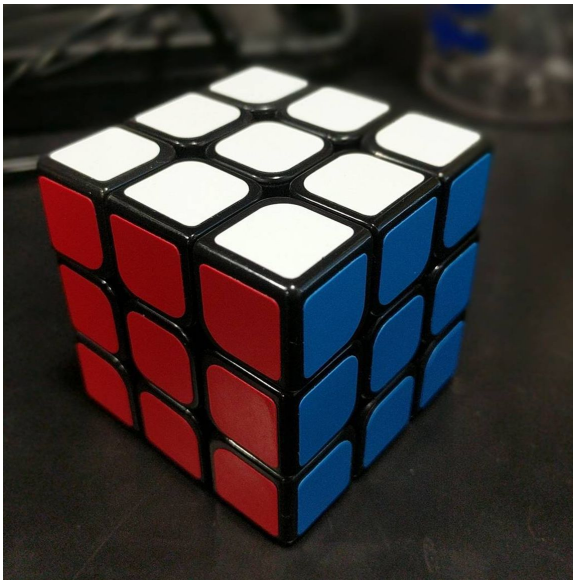
ECE 385
SPRING 2017
Final Project

Rubik's Cube Solver using FPGA and Camera

Siddharth Muralidaran
Simran Patil
Section ABO
Conor Gardner

Introduction:

In this project, we work with the Altera DE2-115 Cyclone IV Board and the TRDB-DC2 Camera to implement a virtual Rubik's solver. The camera input is used to detect the colors in real time and use this data to take in the current configuration of the Rubik's cube. We use the Fridrich algorithm for executing the solving algorithm and display the moves required to reach the solution on the VGA Monitor. The user feeds in the per-face configuration of the cube by showing it to the camera one after the other.



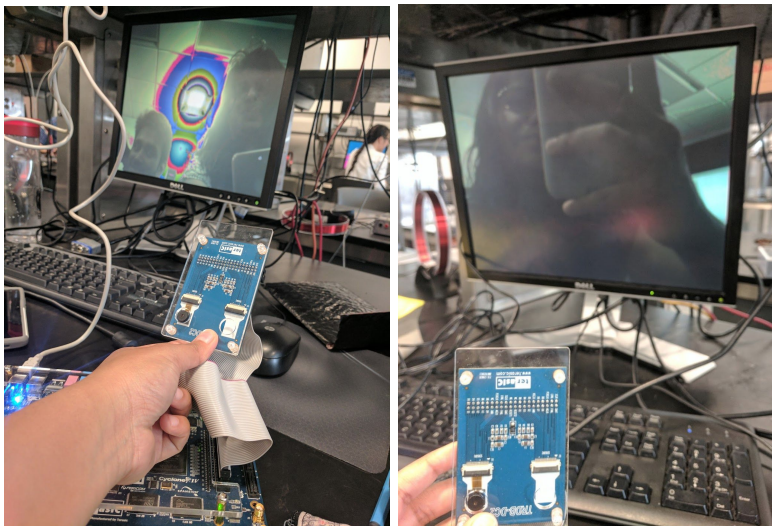
Features:

1. **Camera:** The main input device is the camera. It is connected to the FPGA board using a jtag uart interface and we use verilog code to establish the connection that was provided with the TRDB-DC2 Kit. Because the provided driver had compatibility with the old DE2 Cyclone 2 board, we had to modify the pin assignments and look for the proper signal values in each of the boards to get the camera working with the new board. The output interface involves 10 bits of data while the VGA requires 8 bits of data for display and hence we had to bitshift the output data and thus the colors displayed on the VGA while checking input are not as clear as we would have expected.
2. **Color Detection:** In order to have real-time processing capabilities of color detection, we implemented all of these signal processing modules in system verilog. We aren't doing edge detection and hence we have to provide the cube length in the image to carry out the process of color detection. We average out the colors in each of the blocks per face and pass it on to the next module to check against the thresholds and assign the right colors.

3. **Algorithm:** The Rubik's Cube algorithm implemented in C.We followed the Fridrich algorithm for the actual execution of the solver. We would like to acknowledge the ECE 5760 final project team for the code implementation of the main algorithm. We modified the code to fit our design expectations but the code was inspired by their implementation of Fridrich algorithm for most parts. We took the colors that were detected in hardware and passed it on to software for the execution of the algorithm to create the solver. We had to establish a hardware - software communication FSM in SystemVerilog for the execution of this. The function on the software end that retrieved this data, loaded it into a 1-d array followed by a 2-d array to fit the desired configuration and the created input cube was passed on to the algorithm to solve it. The moves generated are displayed on the VGA along with the initial cube configuration.
4. **VGA Mapping:** The mapping of the input configuration and the moves to VGA display.We planned to map the variable number of moves generated by the algorithm to the VGA along with the input cube configuration.

Camera:

We worked with the TRDB-DC2 camera that was not compatible with the Cyclone IV board and thus we had to modify a lot of pin assignments to get the camera to work with the board. This also involved scrutinizing each of the board documentations to get the appropriate signal values and the connections right in place. Because the camera takes in a 10-bit value while the VGA works with a 8-bit value per color, we had to shift the bits to let the colors to the VGA and because of this the colors on the display weren't as clear as expected. The camera offered additional functionality like capturing the current screen. We were to use this feature in the implementation of our color detection module.



Initially the colors were funky and we managed to fix them a bit using bit shifts and some math.

DE2_CCD.v

This is the toplevel file of the module and has the instantiation of all the other modules. There are 4 important modules in this module that helped us understand the working of the camera and helped us modify the code to suit our requirements and needs: CCD capture, Raw2RGB, SDRAM Controller and VGA controller.

```
always@(posedge CLOCK_50) CCD_MCLK <= ~CCD_MCLK;
always@(posedge CCD_PIXCLK)
begin
 rCCD_DATA <= CCD_DATA;
 rCCD_LVAL <= CCD_LVAL;
 rCCD_FVAL <= CCD_FVAL;
end
VGA_Controller  u1 ( // Host Side
       .oRequest(Read),
       .iRed(Read_DATA2[9:0]),
       .iGreen({Read_DATA1[14:10],Read_DATA2[14:10]}),
       .iBlue(Read_DATA1[9:0]),
       // VGA Side
       .oVGA_R(VGA_R),
       .oVGA_G(VGA_G),
       .oVGA_B(VGA_B),
       .oVGA_H_SYNC(VGA_HS),
       .oVGA_V_SYNC(VGA_VS),
       .oVGA_SYNC(VGA_SYNC_N),
       .oVGA_BLANK(VGA_BLANK_N),
       // Control Signal
       .iCLK(VGA_CTRL_CLK),
       .iRST_N(DLY_RST_2) );
CCD_Capture   u3 ( .oDATA(mCCD_DATA),
       .oDVAL(mCCD_DVAL),
       .oX_Cont(X_Cont),
       .oY_Cont(Y_Cont),
       .oFrame_Cont(Frame_Cont),
       .iDATA(rCCD_DATA),
       .iFVAL(rCCD_FVAL),
       .iLVAL(rCCD_LVAL),
       .iSTART(!KEY[3]),
       .iEND(!KEY[2]),
       .iCLK(CCD_PIXCLK),
       .iRST(DLY_RST_1) );
RAW2RGB    u4 ( .oRed(mCCD_R),
       .oGreen(mCCD_G),
       .oBlue(mCCD_B),
       .oDVAL(mCCD_DVAL_d),
       .iX_Cont(X_Cont),
       .iY_Cont(Y_Cont),
```

```
        .iDATA(mCCD_DATA),
        .iDVAL(mCCD_DVAL),
        .iCLK(CCD_PIXCLK),
        .iRST(DLY_RST_1) );

Sdram_Control_4Port u6 ( // HOST Side
      .REF_CLK(CLOCK_50),
      .RESET_N(1'b1),
       // FIFO Write Side 1
      .WR1_DATA( {sCCD_G[9:5],
            sCCD_B[9:0]}),
       .WR1(sCCD_DVAL),
       .WR1_ADDR(0),
       .WR1_MAX_ADDR(640*512),
       .WR1_LENGTH(9'h100),
       .WR1_LOAD(!DLY_RST_0),
       .WR1_CLK(CCD_PIXCLK),
       // FIFO Write Side 2
      .WR2_DATA( {sCCD_G[4:0],
            sCCD_R[9:0]}),
       .WR2(sCCD_DVAL),
       .WR2_ADDR(22'h100000),
       .WR2_MAX_ADDR(22'h100000+640*512),
       .WR2_LENGTH(9'h100),
       .WR2_LOAD(!DLY_RST_0),
       .WR2_CLK(CCD_PIXCLK),
       // FIFO Read Side 1
      .RD1_DATA(Read_DATA1),
             .RD1(Read),
             .RD1_ADDR(640*16),
       .RD1_MAX_ADDR(640*496),
       .RD1_LENGTH(9'h100),
             .RD1_LOAD(!DLY_RST_0),
       .RD1_CLK(VGA_CTRL_CLK),
       // FIFO Read Side 2
      .RD2_DATA(Read_DATA2),
             .RD2(Read),
             .RD2_ADDR(22'h100000+640*16),
       .RD2_MAX_ADDR(22'h100000+640*496),
       .RD2_LENGTH(9'h100),
             .RD2_LOAD(!DLY_RST_0),
       .RD2_CLK(VGA_CTRL_CLK),
       // SDRAM Side
      .SA(DRAM_ADDR),
      .BA({DRAM_BA[1],DRAM_BA[0]}),
      .CS_N(DRAM_CS_N),
      .CKE(DRAM_CKE),
      .RAS_N(DRAM_RAS_N),
```

```
        .CAS_N(DRAM_CAS_N),
        .WE_N(DRAM_WE_N),
    .DQ(DRAM_DQ),
        .DQM({DRAM_DQM[1],DRAM_DQM[0]}),
     .SDR_CLK(DRAM_CLK) );
```

The Camera worked by capturing frame by frame in the CCD Capture module. It captures the frame using a X counter and a Y counter to help keep track of the coordinates of the image being captured. Line Valid and Frame Valid signals get updated as X counter reaches the end of line and as both coordinates reach end of frame respectively. This way the CCD capture transfers the frame to Raw2RGB module.

```
always@(posedge iCLK or negedge iRST)
begin
 if(!iRST)
 mSTART <= 0;
 else
 begin
  if(iSTART)
  mSTART <= 1;
  if(iEND)
  mSTART <= 0;
 end
end
always@(posedge iCLK or negedge iRST)
begin
 if(!iRST)
 begin
  Pre_FVAL <= 0;
  mCCD_FVAL <= 0;
  mCCD_LVAL <= 0;
  mCCD_DATA <= 0;
  X_Cont  <= 0;
  Y_Cont  <= 0;
 end
 else
 begin
  Pre_FVAL <= iFVAL;
  if( ({Pre_FVAL,iFVAL}==2'b01) && mSTART )
  mCCD_FVAL <= 1;
  else if({Pre_FVAL,iFVAL}==2'b10)
  mCCD_FVAL <= 0;
  mCCD_LVAL <= iLVAL;
  mCCD_DATA <= iDATA;
  if(mCCD_FVAL)
  begin
```

```
   if(mCCD_LVAL)
   begin
    if(X_Cont<1279)
    X_Cont <= X_Cont+1;
    else
    begin
     X_Cont <= 0;
     Y_Cont <= Y_Cont+1;
    end
   end
  end
  else
  begin
   X_Cont <= 0;
   Y_Cont <= 0;
  end
 end
end
always@(posedge iCLK or negedge iRST)
begin
 if(!iRST)
 Frame_Cont <= 0;
 else
 begin
  if( ({Pre_FVAL,iFVAL}==2'b01) && mSTART )
  Frame_Cont <= Frame_Cont+1;
 end
end
```

Raw2RGB converts the raw signals coming from the camera sensor into a 3 channel RGB signal that can be used by the rest of the code to perform functions.

```
always@(posedge iCLK or negedge iRST)
begin
 if(!iRST)
 begin
  mCCD_R <= 0;
  mCCD_G <= 0;
  mCCD_B <= 0;
  mDATAd_0<= 0;
  mDATAd_1<= 0;
  mDVAL <= 0;
 end
 else
 begin
  mDATAd_0 <= mDATA_0;
  mDATAd_1 <= mDATA_1;
  mDVAL  <= {iY_Cont[0]|iX_Cont[0]} ? 1'b0 : iDVAL;
```

```
  if({iY_Cont[0],iX_Cont[0]}==2'b01)
  begin
   mCCD_R <= mDATA_0;
   mCCD_G <= mDATAd_0+mDATA_1;
   mCCD_B <= mDATAd_1;
  end
  else if({iY_Cont[0],iX_Cont[0]}==2'b00)
  begin
   mCCD_R <= mDATAd_0;
   mCCD_G <= mDATA_0+mDATAd_1;
   mCCD_B <= mDATA_1;
  end
  else if({iY_Cont[0],iX_Cont[0]}==2'b11)
  begin
   mCCD_R <= mDATA_1;
   mCCD_G <= mDATA_0+mDATAd_1;
   mCCD_B <= mDATAd_0;
  end
  else if({iY_Cont[0],iX_Cont[0]}==2'b10)
  begin
   mCCD_R <= mDATAd_1;
   mCCD_G <= mDATAd_0+mDATA_1;
   mCCD_B <= mDATA_0;
  end
 end
End
```

The RGB values are stored in the SDRAM using the SDRAM controller. The values are stored in the SDRAM are read from the VGA controller to display on the VGA screen. The VGA controller contains a Horizontal sync and a Vertical sync to help traverse through the screen pixel by pixel. The frame is complete when both horizontal and vertical syncs reach their max limit. The VGA screen displays 8 bits for each color channel while the sensor for the camera sends in 10 bits for each color channel. So the signal is truncated and adjusted to get a clear image on the VGA display.

```
assign oVGA_R = ( H_Cont>=X_START  && H_Cont<X_START+H_SYNC_ACT &&
      V_Cont>=Y_START  && V_Cont<Y_START+V_SYNC_ACT )
      ? iRed[9:2] : 0;
assign oVGA_G = ( H_Cont>=X_START  && H_Cont<X_START+H_SYNC_ACT &&
      V_Cont>=Y_START  && V_Cont<Y_START+V_SYNC_ACT )
      ? iGreen[9:2] : 0;
assign oVGA_B = ( H_Cont>=X_START  && H_Cont<X_START+H_SYNC_ACT &&
      V_Cont>=Y_START  && V_Cont<Y_START+V_SYNC_ACT )
      ? iBlue[9:2] : 0;
// Pixel LUT Address Generator
always@(posedge iCLK or negedge iRST_N)
begin
```

```
 if(!iRST_N)
 oRequest <= 0;
 else
 begin
  if( H_Cont>=X_START-2 && H_Cont<X_START+H_SYNC_ACT-2 &&
   V_Cont>=Y_START && V_Cont<Y_START+V_SYNC_ACT )
  oRequest <= 1;
  else
  oRequest <= 0;
 end
end
// H_Sync Generator, Ref. 25.175 MHz Clock
always@(posedge iCLK or negedge iRST_N)
begin
 if(!iRST_N)
 begin
  H_Cont  <= 0;
  oVGA_H_SYNC <= 0;
 end
 else
 begin
  // H_Sync Counter
  if( H_Cont < H_SYNC_TOTAL )
  H_Cont <= H_Cont+1;
  else
  H_Cont <= 0;
  // H_Sync Generator
  if( H_Cont < H_SYNC_CYC )
  oVGA_H_SYNC <= 0;
  else
  oVGA_H_SYNC <= 1;
 end
end
// V_Sync Generator, Ref. H_Sync
always@(posedge iCLK or negedge iRST_N)
begin
 if(!iRST_N)
 begin
  V_Cont  <= 0;
  oVGA_V_SYNC <= 0;
 end
 else
 begin
  // When H_Sync Re-start
  if(H_Cont==0)
  begin
   // V_Sync Counter
   if( V_Cont < V_SYNC_TOTAL )
```

```
    V_Cont <= V_Cont+1;
    else
    V_Cont <= 0;
    // V_Sync Generator
    if( V_Cont < V_SYNC_CYC )
    oVGA_V_SYNC <= 0;
    else
    oVGA_V_SYNC <= 1;
   end
  end
end
```

The following system verilog files were written by us for carrying out color detection in hardware. We decided to limit this to hardware so that it can be done faster - whether using an image stored in memory or by processing real time images.

We used the script provided on the course website to generate a resized palette of the rubik's cube onto on-chip memory to pass on the values for color detection.

Main Approach and associated modules:

1. Storing a txt file with the palette mapping of an image onto on-chip memory. We achieved that well and the memory block was instantiated with the file. It implied having 6 ram.sv instantiations for the 6 images corresponding to the 6 faces of the cube.
2. Reading in the pixels and comparing against the first edge pixel obtained to detect the start of the cube.
3. Calculating the edge length to get the dimension of the cube (N)
4. Using math and a counter to run through the pixels of the file to get the centers of all the blocks on each side of the cube
5. After getting the centers, running through the pixels all over again, we look for the incoming current location that matches one of the coordinates and and perform color detection with appropriate thresholds and averaging over the nearby 4 pixels to get the color specific to the block.
6. This color is then passed from hardware to software through the io module for being used as a part of the rubik's cube algorithm.
7. We created an array that accepted the inputs from the software port as they were generated and it was then converted in a 6x9 array per face of the cube to pass it on as an input to the Rubik's cube algorithm.
8. The algorithm performed the computation in a series of steps and is an implementation of the basic beginners algorithm used by humans called the Fridrich algorithm. It

involves solving the cube in 5 main steps and the computations of these involved a lot of helper functions that were implemented using matrices and look up tables. A general sequence of math operations was required for the moves and that built up to the solver.

9. Finally, the moves were printed on the terminal to showcase the steps involved
10. We were working on mapping the input configuration of the cube to the VGA. This involved another handshake protocol and FSM to get the appropriate values and implement the algorithm.
11. Mapping the moves was a further more challenging task as the number of moves generated were dependent on the input configuration and thus this implied finding a good way to implement this was to work with hardware software communication across the print functions in c and the FSM in hardware along with user's key input from the FPGA to actually display the move on the VGA.

Modules specific to Color Detection:

Color_Checker.sv

Description: This module takes in the pixel values and compares again upper and lower thresholds of each of the colors to determine which color it was.
Purpose: The main comparison unit in hardware side of the project.

```
module ColorChecker ( input logic Clk, cc,
                                input logic [29:0] Color_in,
                                output logic [29:0] Color_out);

logic [29:0] Red, Green, Blue, Yellow, White, Orange;
real a;
initial
begin
      a = 0.05;
end
Register reg_red (.Clk(Clk), .init(init), .Load(1'b1),
.Data_in(30'b11111111110000000000000000000), .Data_out(Red));
Register reg_green (.Clk(Clk), .init(init), .Load(1'b1),
.Data_in(30'b00000000001111111110000000000), .Data_out(Green));
Register reg_blue (.Clk(Clk), .init(init), .Load(1'b1),
.Data_in(30'b00000000000000000001111111111), .Data_out(Blue));
Register reg_yellow (.Clk(Clk), .init(init), .Load(1'b1),
.Data_in(30'b11111111111111111110000000000), .Data_out(Yellow));
Register reg_white (.Clk(Clk), .init(init), .Load(1'b1),
.Data_in(30'b11111111111111111111111111111), .Data_out(White));
Register reg_orange (.Clk(Clk), .init(init), .Load(1'b1),
.Data_in(30'b11111111111110100101000000000), .Data_out(Orange));
```

```
always_ff @(posedge Clk)
begin
      if(cc)
            begin
                  if( (Color_in - Red) / Red < a || (Red - Color_in) / Red < a)
                        Color_out <= Red;
                  else if ( (Color_in - Green) / Green < a || (Green -
Color_in) / Green < a)
                        Color_out <= Green;
                  else if ( (Color_in - Blue) / Blue < a || (Blue - Color_in) /
Blue < a)
                        Color_out <= Blue;
                  else if ( (Color_in - Yellow) / Yellow < a || (Yellow -
Color_in) / Yellow < a)
                        Color_out <= Yellow;
                  else if ( (Color_in - White) / White < a || (White -
Color_in) / White < a)
                        Color_out <= White;
                  else if ( (Color_in - Orange) / Orange < a || (Orange -
Color_in) / Orange < a)
                        Color_out <= Orange;
                  else
                        Color_out <= 30'b000000000000000000000000000000;
            end
end
endmodule
```

Color_Clubber.sv

Description: This is used to put the colors in the desired format as per the requirements of the file.

Purpose: To club three colors into a 30 bit RGB value

module ColorClubber (input logic [9:0] R, G, B,

```
                              output logic [29:0] Color);


logic [29:0] temp;

always_comb

begin
```

```
	temp = {R, G, B};

end

assign Color = temp;

endmodule
```

Color_Detection.sv

Description: Used for averaging out the nearby pixel values when one of the central coordinates is reached while traversing the file again.

Purpose: To get the appropriate pixel values associated with a block.

```
module ColorDetection (input logic [29:0] pixelValue,
                                    input logic [10:0] X_Cont, Y_Cont,
                                    input logic Clk,cs,init,
                                    input logic [10:0] Block1X, Block1Y,
                                                                Block2X, Block2Y,
                                                                Block3X, Block3Y,
                                                                Block4X, Block4Y,
                                                                Block5X, Block5Y,
                                                                Block6X, Block6Y,
                                                                Block7X, Block7Y,
                                                                Block8X, Block8Y,
                                                                Block9X, Block9Y,
                                        input logic [29:0] Counter_in,
                                        input logic [29:0]   Color1_in,
                                                                        Color2_in,
                                                                        Color3_in,
                                                                        Color4_in,
                                                                        Color5_in,
                                                                        Color6_in,
                                                                        Color7_in,
                                                                        Color8_in,
                                                                        Color9_in,
                                        output logic [29:0]  Color1,
                                                                        Color2,
                                                                        Color3,
                                                                        Color4,
                                                                        Color5,
                                                                        Color6,
                                                                        Color7,
                                                                        Color8,
                                                                        Color9,
                                        output logic [29:0] Counter_out
                                        );

logic [29:0] b1, b2, b3, b4, b5, b6, b7, b8, b9;
logic [29:0] Counter;
logic Load;

always_comb
begin
	if(Counter_in < 30'd45)
```

```
       Load = cs;
else
       Load = 1'b0;

if( X_Cont == Block1X && Y_Cont == Block1Y
       || X_Cont == (Block1X-1) && Y_Cont == Block1Y
    || X_Cont == (Block1X+1) && Y_Cont == Block1Y
    || X_Cont == Block1X && Y_Cont == (Block1Y-1)
    || X_Cont == Block1X && Y_Cont == (Block1Y+1))
       begin
              Counter = Counter_in + 30'h00000001;
              b1 = pixelValue;
       end
else
       b1 = 30'h0000;

if( X_Cont == Block2X && Y_Cont == Block2Y
       || X_Cont == (Block2X-1) && Y_Cont == Block2Y
    || X_Cont == (Block2X+1) && Y_Cont == Block2Y
    || X_Cont == Block2X && Y_Cont == (Block2Y-1)
    || X_Cont == Block2X && Y_Cont == (Block2Y+1))
       begin
              Counter = Counter_in + 30'h00000001;
              b2 = pixelValue;
       end
else
       b2 = 30'h00003;

if( X_Cont == Block3X && Y_Cont == Block3Y
       || X_Cont == (Block3X-1) && Y_Cont == Block3Y
    || X_Cont == (Block3X+1) && Y_Cont == Block3Y
    || X_Cont == Block3X && Y_Cont == (Block3Y-1)
    || X_Cont == Block3X && Y_Cont == (Block3Y+1))
       begin
              Counter = Counter_in + 30'h00000001;
              b3 = pixelValue;
       end
else
       b3 = 30'h0000;

if( X_Cont == Block4X && Y_Cont == Block4Y
       || X_Cont == (Block4X-1) && Y_Cont == Block4Y
    || X_Cont == (Block4X+1) && Y_Cont == Block4Y
    || X_Cont == Block4X && Y_Cont == (Block4Y-1)
    || X_Cont == Block4X && Y_Cont == (Block4Y+1))
       begin
              Counter = Counter_in + 30'h00000001;
              b4 = pixelValue;
       end
else
       b4 = 30'h0000;

if( X_Cont == Block5X && Y_Cont == Block5Y
       || X_Cont == (Block5X-1) && Y_Cont == Block5Y
    || X_Cont == (Block5X+1) && Y_Cont == Block5Y
    || X_Cont == Block5X && Y_Cont == (Block5Y-1)
    || X_Cont == Block5X && Y_Cont == (Block5Y+1))
       begin
              Counter = Counter_in + 30'h00000001;
              b5 = pixelValue;
       end
```

```
	else
		b5 = 30'h0000;

	if( X_Cont == Block6X && Y_Cont == Block6Y
		|| X_Cont == (Block6X-1) && Y_Cont == Block6Y
	    || X_Cont == (Block6X+1) && Y_Cont == Block6Y
	    || X_Cont == Block6X && Y_Cont == (Block6Y-1)
	    || X_Cont == Block6X && Y_Cont == (Block6Y+1))
		begin
			Counter = Counter_in + 30'h00000001;
			b6 = pixelValue;
		end
	else
		b6 = 30'h0000;

	if( X_Cont == Block7X && Y_Cont == Block7Y
		|| X_Cont == (Block7X-1) && Y_Cont == Block7Y
	    || X_Cont == (Block7X+1) && Y_Cont == Block7Y
	    || X_Cont == Block7X && Y_Cont == (Block7Y-1)
	    || X_Cont == Block7X && Y_Cont == (Block7Y+1))
		begin
			Counter = Counter_in + 30'h00000001;
			b7 = pixelValue;
		end
	else
		b7 = 30'h0000;

	if( X_Cont == Block8X && Y_Cont == Block8Y
		|| X_Cont == (Block8X-1) && Y_Cont == Block8Y
	    || X_Cont == (Block8X+1) && Y_Cont == Block8Y
	    || X_Cont == Block8X && Y_Cont == (Block8Y-1)
	    || X_Cont == Block8X && Y_Cont == (Block8Y+1))
		begin
			Counter = Counter_in + 30'h00000001;
			b8 = pixelValue;
		end
	else
		b8 = 30'h0000;

	if( X_Cont == Block9X && Y_Cont == Block9Y
		|| X_Cont == (Block9X-1) && Y_Cont == Block9Y
	    || X_Cont == (Block9X+1) && Y_Cont == Block9Y
	    || X_Cont == Block9X && Y_Cont == (Block9Y-1)
	    || X_Cont == Block9X && Y_Cont == (Block9Y+1))
		begin
			Counter = Counter_in + 30'h00000001;
			b9 = pixelValue;
		end
	else
		b9 = 30'h0000;
end

Register block1_reg (.Clk(Clk), .init(init), .Load(Load), .Data_in(Color1_in+b1/5),
.Data_out(Color1));
Register block2_reg (.Clk(Clk), .init(init), .Load(Load), .Data_in(Color2_in+b2/5),
.Data_out(Color2));
Register block3_reg (.Clk(Clk), .init(init), .Load(Load), .Data_in(Color3_in+b3/5),
.Data_out(Color3));
Register block4_reg (.Clk(Clk), .init(init), .Load(Load), .Data_in(Color4_in+b4/5),
.Data_out(Color4));
Register block5_reg (.Clk(Clk), .init(init), .Load(Load), .Data_in(Color5_in+b5/5),
```

```
.Data_out(Color5));
Register block6_reg (.Clk(Clk), .init(init), .Load(Load), .Data_in(Color6_in+b6/5),
.Data_out(Color6));
Register block7_reg (.Clk(Clk), .init(init), .Load(Load), .Data_in(Color7_in+b7/5),
.Data_out(Color7));
Register block8_reg (.Clk(Clk), .init(init), .Load(Load), .Data_in(Color8_in+b8/5),
.Data_out(Color8));
Register block9_reg (.Clk(Clk), .init(init), .Load(Load), .Data_in(Color9_in+b9/5),
.Data_out(Color9));

Register counter_reg (.Clk(Clk), .init(init), .Load(1'b1), .Data_in(Counter),
.Data_out(Counter_out));
endmodule
```

Detect_Start.sv

Description: This is the starting module that detects the starting coordinates of the cube on the image.

Purpose: This is used for getting the coordinates to do the math associated with the block centers and coordinates used for color detection.

```
module DetectStart (input logic ds, Clk,
                              input logic [10:0] X_Cont, Y_Cont,
                              input logic [9:0] pixelValue,
                              output logic [10:0] CubeX_Start, CubeY_Start,
                              output logic CubeDetected);


always_ff @(posedge Clk)
begin
      if(ds)
            begin
                  if(pixelValue == 16'h0000)
                        begin
                              CubeX_Start = X_Cont;
                              CubeY_Start = Y_Cont;
                              CubeDetected = 1'b1;
                        end
                  else
```

```
                        begin
                                CubeX_Start = 10'b00;
                                CubeY_Start = 10'b00;
                                CubeDetected = 1'b0;
                        end
            end
end
Endmodule
```

getCenters.sv

Description: This is the module used for the computation of the coordinates that are used for averaging out values for color detection.

Purpose: This is used for getting the math values that will be used for computation later.

```
module getCenters ( input logic Clk,gc,
                                input logic [10:0] CubeX_Start, CubeY_Start,
                                input logic [9:0] N,
                                output logic [10:0] Block1X, Block1Y,
                                                                Block2X, Block2Y,
                                                                Block3X, Block3Y,
                                                                Block4X, Block4Y,
                                                                Block5X, Block5Y,
                                                                Block6X, Block6Y,
                                                                Block7X, Block7Y,
                                                                Block8X, Block8Y,
                                                                Block9X, Block9Y,
                                output logic gotCenters
                                );

always_ff @(posedge Clk)
begin
       if(gc)
              begin
                     Block1X <= CubeX_Start + N/6 ;
                     Block1Y <= CubeY_Start + N/6 ;

                     Block2X <= CubeX_Start + N/2 ;
                     Block2Y <= CubeY_Start + N/6 ;

                     Block3X <= CubeX_Start + 5*N/6 ;
                     Block3Y <= CubeY_Start + N/6 ;

                     Block4X <= CubeX_Start + N/6 ;
                     Block4Y <= CubeY_Start + N/2 ;

                     Block5X <= CubeX_Start + N/2 ;
                     Block5Y <= CubeY_Start + N/2 ;
```

```
                    Block6X <= CubeX_Start + 5*N/6 ;
                    Block6Y <= CubeY_Start + N/2 ;

                    Block7X <= CubeX_Start + N/6 ;
                    Block7Y <= CubeY_Start + 5*N/6 ;

                    Block8X <= CubeX_Start + N/2 ;
                    Block8Y <= CubeY_Start + 5*N/6 ;

                    Block9X <= CubeX_Start + 5*N/6 ;
                    Block9Y <= CubeY_Start + 5*N/6 ;

                    gotCenters <= 1'b1;
            end
end

endmodule
```

Toplevel_color.sv

Description: This module is used for the overall hierarchical execution of all the submodules in color detection.

Purpose: To get the color detection files together in conjugation with the io module.

Control_Unit.sv

Description: This module manages the flow of bits across the color detection modules by generating signals required to pass on to the next module for execution

Purpose: This is required to make sure that the execution of the data occurs in the correct order.

```
module ControlUnit ( input logic Clk, Reset, Execute, Face,
                                output logic gc , ds , cs, cc, algstart, init,
facedone
                               );

enum logic [5:0] {  ResetState, Stop_1, Stop_2, Stop_3, Stop_4, Stop_5, Stop_6, Halt,
initState,
                               DetectStart_1, GetCenters_1, ColorDetection_1,
ColorChecker_1,
                               DetectStart_2, GetCenters_2, ColorDetection_2,
ColorChecker_2,
                               DetectStart_3, GetCenters_3, ColorDetection_3,
ColorChecker_3,
                               DetectStart_4, GetCenters_4, ColorDetection_4,
ColorChecker_4,
                               DetectStart_5, GetCenters_5, ColorDetection_5,
ColorChecker_5,
                               DetectStart_6, GetCenters_6, ColorDetection_6,
ColorChecker_6 }   curr_state, next_state;
```

```
always_ff @ (posedge Clk)
    begin
        if (Reset)
            curr_state <= ResetState;
        else
            curr_state <= next_state;
    end

always_comb
    begin
             next_state  = curr_state;
        unique case (curr_state)

            ResetState                  :    if (Execute)
                                                    next_state = initState;
            initState                   :       next_state = DetectStart_1;
            DetectStart_1         :    next_state = GetCenters_1;
            GetCenters_1          :    next_state = ColorDetection_1;
            ColorDetection_1      :    next_state = ColorChecker_1;
            ColorChecker_1              :       next_state = Stop_1;
            Stop_1                      :    if (Face)
                                                    next_state = DetectStart_2;

                    DetectStart_2              :    next_state = GetCenters_2;
            GetCenters_2          :    next_state = ColorDetection_2;
            ColorDetection_2      :    next_state = ColorChecker_2;
            ColorChecker_2              :       next_state = Stop_2;
            Stop_2                      :    if (Face)
                                                    next_state = DetectStart_3;

            DetectStart_3         :    next_state = GetCenters_3;
            GetCenters_3          :    next_state = ColorDetection_3;
            ColorDetection_3      :    next_state = ColorChecker_3;
            ColorChecker_3              :       next_state = Stop_3;
            Stop_3                      :    if (Face)
                                                    next_state = DetectStart_4;

            DetectStart_4         :    next_state = GetCenters_4;
            GetCenters_4          :    next_state = ColorDetection_4;
            ColorDetection_4      :    next_state = ColorChecker_4;
            ColorChecker_4              :       next_state = Stop_4;
            Stop_4                      :    if (Face)
                                                    next_state = DetectStart_5;

            DetectStart_5         :    next_state = GetCenters_5;
            GetCenters_5          :    next_state = ColorDetection_5;
            ColorDetection_5      :    next_state = ColorChecker_5;
            ColorChecker_5              :       next_state = Stop_5;
            Stop_5                      :    if (Face)
                                                    next_state = DetectStart_6;

            DetectStart_6         :    next_state = GetCenters_6;
            GetCenters_6          :    next_state = ColorDetection_6;
            ColorDetection_6      :    next_state = ColorChecker_6;
            ColorChecker_6              :       next_state = Stop_6;
            Stop_6                      :    next_state = Halt;

                           Halt                         :       if (~Execute)
                                                        next_state = ResetState;
        endcase
```

```
	gc = 1'b0;                          // GetCenter
	ds = 1'b0;                          // DetectStart
	cs = 1'b0;                          // ColorStored
	cc = 1'b0;                          // ColorCheck
	algstart = 1'b0;
	facedone = 1'b0;
	init = 1'b0;

case (curr_state)

		ResetState:
			begin
				gc = 1'b0;                          // GetCenter
				ds = 1'b0;                          // DetectStart
				cs = 1'b0;                          // ColorStored
				cc = 1'b0;                          // ColorCheck
				algstart = 1'b0;
				init = 1'b0;
			end

		initState:
			begin
				init = 1'b1;
			end

		DetectStart_1:
			begin
				ds = 1'b1;
			end
		GetCenters_1:
			begin
				gc = 1'b1;
			end
		ColorDetection_1:
			begin
				cs = 1'b1;
			end
		ColorChecker_1:
			begin
				cc = 1'b1;
			end
		Stop_1:
			begin
				facedone = 1'b1;
			end

		DetectStart_2:
			begin
				ds = 1'b1;
			end
		GetCenters_2:
			begin
				gc = 1'b1;
			end
		ColorDetection_2:
			begin
				cs = 1'b1;
			end
		ColorChecker_2:
			begin
				cc = 1'b1;
```

```
        end
Stop_2:
        begin
                facedone = 1'b1;
        end

DetectStart_3:
        begin
                ds = 1'b1;
        end
GetCenters_3:
        begin
                gc = 1'b1;
        end
ColorDetection_3:
        begin
                cs = 1'b1;
        end
ColorChecker_3:
        begin
                cc = 1'b1;
        end
Stop_3:
        begin
                facedone = 1'b1;
        end

DetectStart_4:
        begin
                ds = 1'b1;
        end
GetCenters_4:
        begin
                gc = 1'b1;
        end
ColorDetection_4:
        begin
                cs = 1'b1;
        end
ColorChecker_4:
        begin
                cc = 1'b1;
        end
Stop_4:
        begin
                facedone = 1'b1;
        end

DetectStart_5:
        begin
                ds = 1'b1;
        end
GetCenters_5:
        begin
                gc = 1'b1;
        end
ColorDetection_5:
        begin
                cs = 1'b1;
        end
ColorChecker_5:
```

```
                        begin
                                cc = 1'b1;
                        end
                    Stop_5:
                        begin
                                facedone = 1'b1;
                        end

                    DetectStart_6:
                        begin
                                ds = 1'b1;
                        end
                    GetCenters_6:
                        begin
                                gc = 1'b1;
                        end
                    ColorDetection_6:
                        begin
                                cs = 1'b1;
                        end
                    ColorChecker_6:
                        begin
                                cc = 1'b1;
                        end
                    Stop_6:
                        begin
                                facedone = 1'b1;
                        end

                    Halt:
                        begin
                                algstart = 1'b1;
                        end

                    default:
                    begin
                gc = 1'b0;                      // GetCenter
                        ds = 1'b0;                          // DetectStart
                        cs = 1'b0;                          // ColorStored
                        cc = 1'b0;                          // ColorCheck
                        algstart = 1'b0;
                        init = 1'b0;
                        facedone = 1'b0;
                    end

        endcase
    end
endmodule
```

Color_Mapper.sv

Description: This module takes in an array of 10 bit X and Y coordinates corresponding to the 54 blocks in the cube and the colors associated with it to pass it on for mapping it to the VGA.

Purpose: To display the input configuration on the VGA Display.

```
module  color_mapper (  input   [539:0] CubeX, CubeY,       // Cube coordinates
                        input   [9:0]  CubeS,  DrawX, DrawY,            // Cube size
```

```
                        input logic [431:0] Color_R, Color_G, Color_B,
                        output logic [7:0] VGA_R, VGA_G, VGA_B // VGA RGB output
                    );

    logic [7:0] Red, Green, Blue;

 /* The Cube's (pixelated) circle is generated using the standard circle formula.
Note that while
    the single line is quite powerful descriptively, it causes the synthesis tool to
use up three
    of the 12 available multipliers on the chip! Since the multiplicants are required
to be signed,
    we have to first cast them from logic to int (signed by default) before they are
multiplied. */

    int Size;
    assign Size = CubeS;

    assign VGA_R = Red;
    assign VGA_G = Green;
    assign VGA_B = Blue;

    always_comb
    begin

        if( (DrawX - CubeX[539:530]) * (DrawY - CubeY[539:530]) <= (Size * Size) )
        begin
            // Color Cube
            Red = Color_R[431:424];
            Green = Color_G[431:424];
            Blue = Color_B[431:424];
        end

        else if( (DrawX - CubeX[529:520]) * (DrawY - CubeY[529:520]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[423:416];
            Green = Color_G[423:416];
            Blue = Color_B[423:416];
        end

        else if( (DrawX - CubeX[519:510]) * (DrawY - CubeY[519:510]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[415:408];
            Green = Color_G[415:408];
            Blue = Color_B[415:408];
        end

        else if( (DrawX - CubeX[509:500]) * (DrawY - CubeY[509:500]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[407:400];
            Green = Color_G[407:400];
            Blue = Color_B[407:400];
        end

        else if( (DrawX - CubeX[499:490]) * (DrawY - CubeY[499:490]) <= (Size * Size)
```

```
)
        begin
            // Color Cube
            Red = Color_R[399:392];
            Green = Color_G[399:392];
            Blue = Color_B[399:392];
        end

        else if( (DrawX - CubeX[489:480]) * (DrawY - CubeY[489:480]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[391:384];
            Green = Color_G[391:384];
            Blue = Color_B[391:384];
        end

        else if( (DrawX - CubeX[479:470]) * (DrawY - CubeY[479:470]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[383:376];
            Green = Color_G[383:376];
            Blue = Color_B[383:376];
        end

        else if( (DrawX - CubeX[469:460]) * (DrawY - CubeY[469:460]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[375:368];
            Green = Color_G[375:368];
            Blue = Color_B[375:368];
        end

        else if( (DrawX - CubeX[459:450]) * (DrawY - CubeY[459:450]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[367:360];
            Green = Color_G[367:360];
            Blue = Color_B[367:360];
        end

        else if( (DrawX - CubeX[449:440]) * (DrawY - CubeY[449:440]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[359:352];
            Green = Color_G[359:352];
            Blue = Color_B[359:352];
        end

        else if( (DrawX - CubeX[439:430]) * (DrawY - CubeY[439:430]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[351:344];
            Green = Color_G[351:344];
            Blue = Color_B[351:344];
        end
```

```
        else if( (DrawX - CubeX[429:420]) * (DrawY - CubeY[429:420]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[343:336];
            Green = Color_G[343:336];
            Blue = Color_B[343:336];
        end

        else if( (DrawX - CubeX[419:410]) * (DrawY - CubeY[419:410]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[335:328];
            Green = Color_G[335:328];
            Blue = Color_B[335:328];
        end

        else if( (DrawX - CubeX[409:400]) * (DrawY - CubeY[409:400]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[327:320];
            Green = Color_G[327:320];
            Blue = Color_B[327:320];
        end

        else if( (DrawX - CubeX[399:390]) * (DrawY - CubeY[399:390]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[319:312];
            Green = Color_G[319:312];
            Blue = Color_B[319:312];
        end

        else if( (DrawX - CubeX[389:380]) * (DrawY - CubeY[389:380]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[311:304];
            Green = Color_G[311:304];
            Blue = Color_B[311:304];
        end

        else if( (DrawX - CubeX[379:370]) * (DrawY - CubeY[379:370]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[303:296];
            Green = Color_G[303:296];
            Blue = Color_B[303:296];
        end

        else if( (DrawX - CubeX[369:360]) * (DrawY - CubeY[369:360]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[295:288];
            Green = Color_G[295:288];
```

```
            Blue = Color_B[295:288];
        end

        else if( (DrawX - CubeX[359:350]) * (DrawY - CubeY[359:350]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[287:280];
            Green = Color_G[287:280];
            Blue = Color_B[287:280];
        end

        else if( (DrawX - CubeX[349:340]) * (DrawY - CubeY[349:340]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[279:272];
            Green = Color_G[279:272];
            Blue = Color_B[279:272];
        end

        else if( (DrawX - CubeX[339:330]) * (DrawY - CubeY[339:330]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[271:264];
            Green = Color_G[271:264];
            Blue = Color_B[271:264];
        end

        else if( (DrawX - CubeX[329:320]) * (DrawY - CubeY[329:320]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[263:256];
            Green = Color_G[263:256];
            Blue = Color_B[263:256];
        end

        else if( (DrawX - CubeX[319:310]) * (DrawY - CubeY[319:310]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[255:248];
            Green = Color_G[255:248];
            Blue = Color_B[255:248];
        end

        else if( (DrawX - CubeX[309:300]) * (DrawY - CubeY[309:300]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[247:240];
            Green = Color_G[247:240];
            Blue = Color_B[247:240];
        end

        else if( (DrawX - CubeX[299:290]) * (DrawY - CubeY[299:290]) <= (Size * Size)
)
        begin
            // Color Cube
```

```
            Red = Color_R[239:232];
            Green = Color_G[239:232];
            Blue = Color_B[239:232];
        end

        else if( (DrawX - CubeX[289:280]) * (DrawY - CubeY[289:280]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[231:224];
            Green = Color_G[231:224];
            Blue = Color_B[231:224];
        end

        else if( (DrawX - CubeX[279:270]) * (DrawY - CubeY[279:270]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[223:216];
            Green = Color_G[223:216];
            Blue = Color_B[223:216];
        end

        else if( (DrawX - CubeX[269:260]) * (DrawY - CubeY[269:260]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[215:208];
            Green = Color_G[215:208];
            Blue = Color_B[215:208];
        end

        else if( (DrawX - CubeX[259:250]) * (DrawY - CubeY[259:250]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[207:200];
            Green = Color_G[207:200];
            Blue = Color_B[207:200];
        end

        else if( (DrawX - CubeX[249:240]) * (DrawY - CubeY[249:240]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[199:192];
            Green = Color_G[199:192];
            Blue = Color_B[199:192];
        end

        else if( (DrawX - CubeX[239:230]) * (DrawY - CubeY[239:230]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[191:184];
            Green = Color_G[191:184];
            Blue = Color_B[191:184];
        end

        else if( (DrawX - CubeX[229:220]) * (DrawY - CubeY[229:220]) <= (Size * Size)
)
```

```
        begin
            // Color Cube
            Red = Color_R[183:176];
            Green = Color_G[183:176];
            Blue = Color_B[183:176];
        end

        else if( (DrawX - CubeX[219:210]) * (DrawY - CubeY[219:210]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[175:168];
            Green = Color_G[175:168];
            Blue = Color_B[175:168];
        end

        else if( (DrawX - CubeX[209:200]) * (DrawY - CubeY[209:200]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[167:160];
            Green = Color_G[167:160];
            Blue = Color_B[167:160];
        end

        else if( (DrawX - CubeX[199:190]) * (DrawY - CubeY[199:190]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[159:152];
            Green = Color_G[159:152];
            Blue = Color_B[159:152];
        end

        else if( (DrawX - CubeX[189:180]) * (DrawY - CubeY[189:180]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[151:144];
            Green = Color_G[151:144];
            Blue = Color_B[151:144];
        end

        else if( (DrawX - CubeX[179:170]) * (DrawY - CubeY[179:170]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[143:136];
            Green = Color_G[143:136];
            Blue = Color_B[143:136];
        end

        else if( (DrawX - CubeX[169:160]) * (DrawY - CubeY[169:160]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[135:128];
            Green = Color_G[135:128];
            Blue = Color_B[135:128];
        end
```

```
        else if( (DrawX - CubeX[159:150]) * (DrawY - CubeY[159:150]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[127:120];
            Green = Color_G[127:120];
            Blue = Color_B[127:120];
        end

        else if( (DrawX - CubeX[149:140]) * (DrawY - CubeY[149:140]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[119:112];
            Green = Color_G[119:112];
            Blue = Color_B[119:112];
        end

        else if( (DrawX - CubeX[139:130]) * (DrawY - CubeY[139:130]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[111:104];
            Green = Color_G[111:104];
            Blue = Color_B[111:104];
        end

        else if( (DrawX - CubeX[129:120]) * (DrawY - CubeY[129:120]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[103:96];
            Green = Color_G[103:96];
            Blue = Color_B[103:96];
        end

        else if( (DrawX - CubeX[119:110]) * (DrawY - CubeY[119:110]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[95:88];
            Green = Color_G[95:88];
            Blue = Color_B[95:88];
        end

        else if( (DrawX - CubeX[109:100]) * (DrawY - CubeY[109:100]) <= (Size * Size)
)
        begin
            // Color Cube
            Red = Color_R[87:80];
            Green = Color_G[87:80];
            Blue = Color_B[87:80];
        end

        else if( (DrawX - CubeX[99:90]) * (DrawY - CubeY[99:90]) <= (Size * Size) )
        begin
            // Color Cube
            Red = Color_R[79:72];
            Green = Color_G[79:72];
            Blue = Color_B[79:72];
        end
```

```
else if( (DrawX - CubeX[89:80]) * (DrawY - CubeY[89:80]) <= (Size * Size) )
begin
    // Color Cube
    Red = Color_R[71:64];
    Green = Color_G[71:64];
    Blue = Color_B[71:64];
end

else if( (DrawX - CubeX[79:70]) * (DrawY - CubeY[79:70]) <= (Size * Size) )
begin
    // Color Cube
    Red = Color_R[63:56];
    Green = Color_G[63:56];
    Blue = Color_B[63:56];
end

else if( (DrawX - CubeX[69:60]) * (DrawY - CubeY[69:60]) <= (Size * Size) )
begin
    // Color Cube
    Red = Color_R[55:48];
    Green = Color_G[55:48];
    Blue = Color_B[55:48];
end

else if( (DrawX - CubeX[59:50]) * (DrawY - CubeY[59:50]) <= (Size * Size) )
begin
    // Color Cube
    Red = Color_R[47:40];
    Green = Color_G[47:40];
    Blue = Color_B[47:40];
end

else if( (DrawX - CubeX[49:40]) * (DrawY - CubeY[49:40]) <= (Size * Size) )
begin
    // Color Cube
    Red = Color_R[39:32];
    Green = Color_G[39:32];
    Blue = Color_B[39:32];
end

else if( (DrawX - CubeX[39:30]) * (DrawY - CubeY[39:30]) <= (Size * Size) )
begin
    // Color Cube
    Red = Color_R[31:24];
    Green = Color_G[31:24];
    Blue = Color_B[31:24];
end

else if( (DrawX - CubeX[29:20]) * (DrawY - CubeY[29:20]) <= (Size * Size) )
begin
    // Color Cube
    Red = Color_R[23:16];
    Green = Color_G[23:16];
    Blue = Color_B[23:16];
end

else if( (DrawX - CubeX[19:10]) * (DrawY - CubeY[19:10]) <= (Size * Size) )
begin
    // Color Cube
    Red = Color_R[15:8];
```

```
            Green = Color_G[15:8];
            Blue = Color_B[15:8];
        end

        else if( (DrawX - CubeX[9:0]) * (DrawY - CubeY[9:0]) <= (Size * Size) )
        begin
            // Color Cube
            Red = Color_R[7:0];
            Green = Color_G[7:0];
            Blue = Color_B[7:0];
        end

        else
        begin
            // White Background
            Red = 8'hff;
            Green = 8'hff;
            Blue = 8'hff;
        end
    end

Endmodule
```

VGA

- Mapping the input cube configuration
- Mapping the moves generated (Non-deterministic in number)

Vga_controller.sv

Description: This module is the same as the one provided in lab 8. It is used to work with the VGA and the frame rates with X and Y coordinates.

Purpose: To pass on the RGB values to be written to the VGA.

HexDriver.sv

Description: This is used to make connections to the hexdriver
Purpose: It is used for IO connections to Hex display. Helped a lot with debugging
Input:[3:0] In0
Output:[6:0] Out0

```
module HexDriver (input  logic[3:0]  In0,
                  output logic [6:0]  Out0);

       always_comb
       begin
              unique case (In0)
                 4'b0000   : Out0 = 7'b1000000; // '0'
                 4'b0001   : Out0 = 7'b1111001; // '1'
                 4'b0010   : Out0 = 7'b0100100; // '2'
```

```
                4'b0011   : Out0 = 7'b0110000; // '3'
                4'b0100   : Out0 = 7'b0011001; // '4'
                4'b0101   : Out0 = 7'b0010010; // '5'
                4'b0110   : Out0 = 7'b0000010; // '6'
                4'b0111   : Out0 = 7'b1111000; // '7'
                4'b1000   : Out0 = 7'b0000000; // '8'
                4'b1001   : Out0 = 7'b0010000; // '9'
                4'b1010   : Out0 = 7'b0001000; // 'A'
                4'b1011   : Out0 = 7'b0000011; // 'B'
                4'b1100   : Out0 = 7'b1000110; // 'C'
                4'b1101   : Out0 = 7'b0100001; // 'D'
                4'b1110   : Out0 = 7'b0000110; // 'E'
                4'b1111   : Out0 = 7'b0001110; // 'F'
                default   : Out0 = 7'bXXXXXXX; // undefined output
        endcase
      end

endmodule
```

CubeMap.sv

Description: It has per pixel indexing and generates values associated with the VGA Screen coordinates.

Purpose: This module is used to map the input cubes from the 6x9 array format onto the VGA.

```
 module  Cube ( input         Reset,
                              frame_clk,          // The clock indicating a new frame
(~60Hz)
               input logic [6:0] index,
               input logic [29:0] Color_in,
               output [9:0]  CubeX, CubeY, CubeS, // CubeX and CubeY are the center
coordinates for the blocks
               output logic [29:0] Color_out
              );

    logic [9:0] Cube_X_Pos,  Cube_Y_Pos;
    logic [9:0] Cube_X_Pos_in, Cube_Y_Pos_in;

    parameter [9:0] Cube_X_Center=320;  // Center position on the X axis
    parameter [9:0] Cube_Y_Center=240;  // Center position on the Y axis
    parameter [9:0] Cube_X_Min=0;       // Leftmost point on the X axis
    parameter [9:0] Cube_X_Max=639;     // Rightmost point on the X axis
    parameter [9:0] Cube_Y_Min=0;       // Topmost point on the Y axis
    parameter [9:0] Cube_Y_Max=479;     // Bottommost point on the Y axis
    parameter [9:0] Cube_Size = 4;      // Block size

    assign CubeX = Cube_X_Pos;
    assign CubeY = Cube_Y_Pos;
    assign CubeS = Cube_Size;

    always_ff @ (posedge frame_clk or posedge Reset)
    begin
        if (Reset)
        begin
            Cube_X_Pos <= Cube_X_Center;
```

```
            Cube_Y_Pos <= Cube_Y_Center;
            Color_out <= 30'hffffffff;
        end
        else
        begin
            Cube_X_Pos <= Cube_X_Pos_in;
            Cube_Y_Pos <= Cube_Y_Pos_in;
            Color_out <= Color_in;
        end
    end

    always_comb
    begin

         // Be careful when using comparators with "logic" datatype becuase compiler
treats
        //   both sides of the operator UNSIGNED numbers. (unless with further type
casting)
        // e.g. Cube_Y_Pos - Cube_Size <= Cube_Y_Min
        // If Cube_Y_Pos is 0, then Cube_Y_Pos - Cube_Size will not be -4, but rather
a large positive number.

        unique case (index)

            // U --> (0 to 8)

            7'd0 :
            begin
                Cube_X_Pos_in = 10'd16;
                Cube_Y_Pos_in = 10'd6;
            end

            7'd1 :
            begin
                Cube_X_Pos_in = 10'd20;
                Cube_Y_Pos_in = 10'd6;
            end

            7'd2 :
            begin
                Cube_X_Pos_in = 10'd24;
                Cube_Y_Pos_in = 10'd6;
            end

            7'd3 :
            begin
                Cube_X_Pos_in = 10'd16;
                Cube_Y_Pos_in = 10'd10;
            end

            7'd4 :
            begin
                Cube_X_Pos_in = 10'd20;
                Cube_Y_Pos_in = 10'd10;
            end

            7'd5 :
            begin
                Cube_X_Pos_in = 10'd24;
                Cube_Y_Pos_in = 10'd10;
            end
```

```
7'd6 :
begin
    Cube_X_Pos_in = 10'd16;
    Cube_Y_Pos_in = 10'd14;
end

7'd7 :
begin
    Cube_X_Pos_in = 10'd20;
    Cube_Y_Pos_in = 10'd14;
end

7'd8 :
begin
    Cube_X_Pos_in = 10'd24;
    Cube_Y_Pos_in = 10'd14;
end

// D --> (9 to 17)

7'd9 :
begin
    Cube_X_Pos_in = 10'd16;
    Cube_Y_Pos_in = 10'd30;
end

7'd10 :
begin
    Cube_X_Pos_in = 10'd20;
    Cube_Y_Pos_in = 10'd30;
end

7'd11 :
begin
    Cube_X_Pos_in = 10'd24;
    Cube_Y_Pos_in = 10'd30;
end

7'd12 :
begin
    Cube_X_Pos_in = 10'd16;
    Cube_Y_Pos_in = 10'd34;
end

7'd13 :
begin
    Cube_X_Pos_in = 10'd20;
    Cube_Y_Pos_in = 10'd34;
end

7'd14 :
begin
    Cube_X_Pos_in = 10'd24;
    Cube_Y_Pos_in = 10'd34;
end

7'd15 :
begin
    Cube_X_Pos_in = 10'd16;
    Cube_Y_Pos_in = 10'd38;
```

```
end

7'd16 :
begin
    Cube_X_Pos_in = 10'd20;
    Cube_Y_Pos_in = 10'd38;
end

7'd17 :
begin
    Cube_X_Pos_in = 10'd24;
    Cube_Y_Pos_in = 10'd38;
end

// B --> (18 to 26)

7'd18 :
begin
    Cube_X_Pos_in = 10'd40;
    Cube_Y_Pos_in = 10'd18;
end

7'd19 :
begin
    Cube_X_Pos_in = 10'd44;
    Cube_Y_Pos_in = 10'd18;
end

7'd20 :
begin
    Cube_X_Pos_in = 10'd48;
    Cube_Y_Pos_in = 10'd18;
end

7'd21 :
begin
    Cube_X_Pos_in = 10'd40;
    Cube_Y_Pos_in = 10'd22;
end

7'd22 :
begin
    Cube_X_Pos_in = 10'd44;
    Cube_Y_Pos_in = 10'd22;
end

7'd23 :
begin
    Cube_X_Pos_in = 10'd48;
    Cube_Y_Pos_in = 10'd22;
end

7'd24 :
begin
    Cube_X_Pos_in = 10'd40;
    Cube_Y_Pos_in = 10'd26;
end

7'd25 :
begin
    Cube_X_Pos_in = 10'd44;
```

```
    Cube_Y_Pos_in = 10'd26;
end

7'd26 :
begin
    Cube_X_Pos_in = 10'd48;
    Cube_Y_Pos_in = 10'd26;
end

// F --> (27 to 35)

7'd27 :
begin
    Cube_X_Pos_in = 10'd16;
    Cube_Y_Pos_in = 10'd18;
end

7'd28 :
begin
    Cube_X_Pos_in = 10'd20;
    Cube_Y_Pos_in = 10'd18;
end

7'd29 :
begin
    Cube_X_Pos_in = 10'd24;
    Cube_Y_Pos_in = 10'd18;
end

7'd30 :
begin
    Cube_X_Pos_in = 10'd16;
    Cube_Y_Pos_in = 10'd22;
end

7'd31 :
begin
    Cube_X_Pos_in = 10'd20;
    Cube_Y_Pos_in = 10'd22;
end

7'd32 :
begin
    Cube_X_Pos_in = 10'd24;
    Cube_Y_Pos_in = 10'd22;
end

7'd33 :
begin
    Cube_X_Pos_in = 10'd16;
    Cube_Y_Pos_in = 10'd26;
end

7'd34 :
begin
    Cube_X_Pos_in = 10'd20;
    Cube_Y_Pos_in = 10'd26;
end

7'd35 :
begin
```

```
    Cube_X_Pos_in = 10'd24;
    Cube_Y_Pos_in = 10'd26;
end

// L --> (36 to 44)

7'd36 :
begin
    Cube_X_Pos_in = 10'd4;
    Cube_Y_Pos_in = 10'd18;
end

7'd37 :
begin
    Cube_X_Pos_in = 10'd8;
    Cube_Y_Pos_in = 10'd18;
end

7'd38 :
begin
    Cube_X_Pos_in = 10'd12;
    Cube_Y_Pos_in = 10'd18;
end

7'd39 :
begin
    Cube_X_Pos_in = 10'd4;
    Cube_Y_Pos_in = 10'd22;
end

7'd40 :
begin
    Cube_X_Pos_in = 10'd8;
    Cube_Y_Pos_in = 10'd22;
end

7'd41 :
begin
    Cube_X_Pos_in = 10'd12;
    Cube_Y_Pos_in = 10'd22;
end

7'd42 :
begin
    Cube_X_Pos_in = 10'd4;
    Cube_Y_Pos_in = 10'd26;
end

7'd43 :
begin
    Cube_X_Pos_in = 10'd8;
    Cube_Y_Pos_in = 10'd26;
end

7'd44 :
begin
    Cube_X_Pos_in = 10'd12;
    Cube_Y_Pos_in = 10'd26;
end

// R --> (45 to 53)
```

```
            7'd45 :
            begin
                Cube_X_Pos_in = 10'd28;
                Cube_Y_Pos_in = 10'd18;
            end

            7'd46 :
            begin
                Cube_X_Pos_in = 10'd32;
                Cube_Y_Pos_in = 10'd18;
            end

            7'd47 :
            begin
                Cube_X_Pos_in = 10'd36;
                Cube_Y_Pos_in = 10'd18;
            end

            7'd48 :
            begin
                Cube_X_Pos_in = 10'd28;
                Cube_Y_Pos_in = 10'd22;
            end

            7'd49 :
            begin
                Cube_X_Pos_in = 10'd32;
                Cube_Y_Pos_in = 10'd22;
            end

            7'd50 :
            begin
                Cube_X_Pos_in = 10'd36;
                Cube_Y_Pos_in = 10'd22;
            end

            7'd51 :
            begin
                Cube_X_Pos_in = 10'd28;
                Cube_Y_Pos_in = 10'd26;
            end

            7'd52 :
            begin
                Cube_X_Pos_in = 10'd32;
                Cube_Y_Pos_in = 10'd26;
            end

            7'd53 :
            begin
                Cube_X_Pos_in = 10'd36;
                Cube_Y_Pos_in = 10'd26;
            end

        endcase
    end

endmodule
```

FontMapper.sv

Description: This is used to map the font to the VGA. It works with the font_rom.sv file provided.

Purpose: This module is used for mapping the characters of the moves generated onto the VGA.

```
module  font_mapper (   input        [9:0] FontX, FontY,       // Font
coordinates
                                           DrawX, DrawY,
                        input        [9:0] N,
                        input logic [7:0] R, G, B,
                        output logic [7:0] VGA_R, VGA_G, VGA_B // VGA RGB
output
                    );


    logic Font_on;
    logic [7:0] Red, Green, Blue;
    logic [10:0] shape_x = FontX;
    logic [10:0] shape_y = FontY;
    logic [10:0] shape_size_x = 8;
    logic [10:0] shape_size_y = 16;

    logic [10:0] sprite_addr;
    logic [7:0] sprite_data;
    font_rom(.addr(sprite_addr), .data(sprite_data));


 /* The Font's (pixelated) circle is generated using the standard circle
formula.  Note that while
    the single line is quite powerful descriptively, it causes the synthesis
tool to use up three
    of the 12 available multipliers on the chip! Since the multiplicants are
required to be signed,
    we have to first cast them from logic to int (signed by default) before
they are multiplied. */

    assign VGA_R = Red;
    assign VGA_G = Green;
    assign VGA_B = Blue;

    always_comb
    begin : Font_on_proc
        if ( DrawX >= shape_x && DrawX <= shape_x + shape_size_x && DrawY >=
shape_y && DrawY < shape_y + shape_size_y  )
        begin
             Font_on = 1'b1;
```

```
            sprite_addr = (DrawY - shape_y + 16*N)
        end
        else
        begin
            Font_on = 1'b0;
            sprite_addr = 10'b0;
        end
    end

    always_comb
    begin : RGB_Display
        if ((Font_on == 1'b1) && sprite_data[ DrawX - shape_x ] == 1'b1)
        begin
            Red = R;
            Green = G;
            Blue = B;
        end
        else
        begin
            Red = 8'hff;
            Green = 8'hff;
            Blue = 8'hff;
        end
    end

endmodule
```

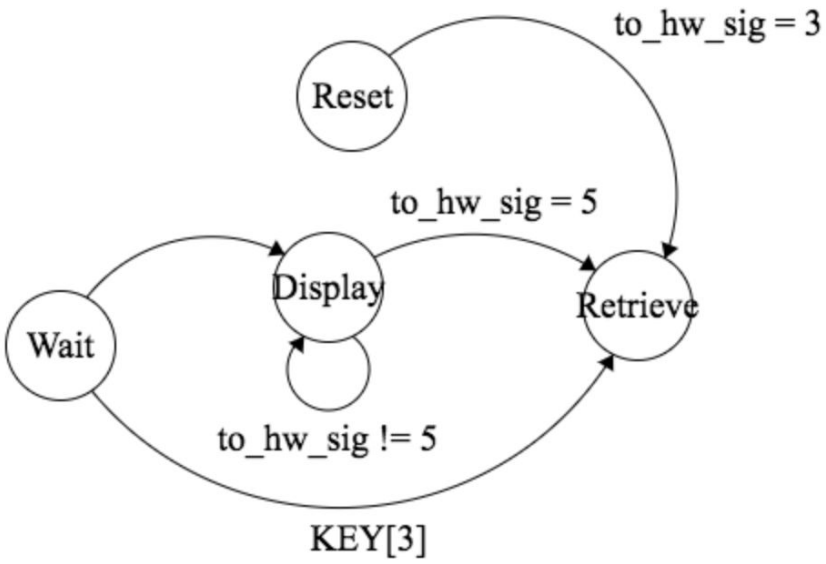
fontFSM.sv

Description: This module is used to establish the handshake protocol between the hardware and the software for the mapping of font onto the VGA.

Purpose: The signal for printing is from software while the state machine is handled in hardware.

```
module fontFSM      (     input logic Clk, Reset, printing, Execute,
                                input logic [2:0] to_hw_sig_font,
                                input logic [29:0] to_hw_port_font,
                                output logic [29:0] to_sw_port_font,
                                output logic [2:0] to_sw_sig_font);

enum logic [2:0] {  ResetState, Wait, Retrieve
                          }  curr_state, next_state;


always_ff @ (posedge Clk)
    begin
        if (Reset)
            curr_state <= ResetState;
        else
            curr_state <= next_state;
    end

always_comb
```

```
    begin
              next_state  = curr_state;
        unique case (curr_state)

              ResetState: next_state = Wait;

                    Wait:  if(to_hw_sig_font == 3'd5)
                                next_state = Retrieve;
              Retrieve: if(to_hw_sig_font == 3'd6)
                                next_state = Wait;
        endcase

        unique case (curr_state)

                    ResetState:
                          begin
                                to_sw_sig_font = 3'd3;
                          end

                    Wait:
                          begin
                                to_sw_sig_font = 3'd0;
                          end

                    Retrieve:
                          begin
                                to_sw_sig_font = 3'd1;
                          end
              endcase
Endmodule
```



FSM according to new code.
The code above was modified in lab but we didn't have access to it while drafting this and hence have included a previous commit snippet.

The Rubik's cube algorithm is implemented using C code and the communication with hardware and this piece of code is established using different PIO blocks that we used according to the requirements.

```
int cubieColor[6][9] = {
{G,W,W,B,W,O,O,G,O}, //U
{B,G,G,W,Y,G,B,O,Y}, //D
{G,R,Y,Y,O,Y,B,W,W}, //B
{G,Y,B,O,R,W,W,O,W}, //F
{R,R,Y,B,G,Y,O,G,R}, //L
{Y,B,O,B,B,R,R,R,R}  //R
};
```

The mainf. And mainf.h files have checks and calls to the other functions that were declared in moveComputations.c and moveComputations.h. We modified the code from the reference of cornell 5760 course to fit our designs. We modified certain functions to make a virtual rubik's cube solver instead of having to include servos and other mechanical components like their code had. We also added the hardware software communication blocks, the solver.c and solver.h files that are used for input block and data extractions and the VGA Fsm handshake protocols, all that were merged with the modified code. We kept the basic algorithm step implementations as they were in the algorithm provided as they are the standard look up tables but we made sincere effort to rule out unnecessary steps and follow the code to fit our design. We also eliminated a few of the functions in teh code because of our newly created functions for data inputs in specific formats. Here is a snippet of moves and starndard lookups:

```
rotateCubeVertical();
rotateCubeVertical();
rotateCubeHorizontal();
rotateCubeHorizontal();

while (!((cubieColor[front][4] == cubieColor[front][3]) &&
(cubieColor[front][4] == cubieColor[front][5]) &&
(cubieColor[left][4] == cubieColor[left][3]) &&
(cubieColor[left][4] == cubieColor[left][5]) &&
(cubieColor[right][4] == cubieColor[right][3]) &&
(cubieColor[right][4] == cubieColor[right][5]) &&
(cubieColor[back][4] == cubieColor[back][3]) &&
(cubieColor[back][4] == cubieColor[back][5]) &&
(cubieColor[front][4] == cubieColor[front][6]) &&
(cubieColor[front][4] == cubieColor[front][8]) &&
(cubieColor[left][4] == cubieColor[left][6]) &&
(cubieColor[left][4] == cubieColor[left][8]) &&
(cubieColor[right][4] == cubieColor[right][6]) &&
(cubieColor[right][4] == cubieColor[right][8]) &&
(cubieColor[back][4] == cubieColor[back][6]) &&
(cubieColor[back][4] == cubieColor[back][8])))
{
                    f2l();
       rotateCubeHorizontal();
       //testf2l();
       //printcube(cubieColor);
}
```

```
                    case rClock:
                    cubieColor[3][2] = cubieTemp[1][2];
                    cubieColor[3][5] = cubieTemp[1][5];
                    cubieColor[3][8] = cubieTemp[1][8];

                    cubieColor[1][2] = cubieTemp[2][6];
                    cubieColor[1][5] = cubieTemp[2][3];
                    cubieColor[1][8] = cubieTemp[2][0];

                    cubieColor[2][0] = cubieTemp[0][8];
                    cubieColor[2][3] = cubieTemp[0][5];
                    cubieColor[2][6] = cubieTemp[0][2];

                    cubieColor[0][2] = cubieTemp[3][2];
                    cubieColor[0][5] = cubieTemp[3][5];
                    cubieColor[0][8] = cubieTemp[3][8];

                    cubieColor[5][0] = cubieTemp[5][6];
                    cubieColor[5][1] = cubieTemp[5][3];
                    cubieColor[5][2] = cubieTemp[5][0];
                    cubieColor[5][3] = cubieTemp[5][7];
                    cubieColor[5][4] = cubieTemp[5][4];
                    cubieColor[5][5] = cubieTemp[5][1];
                    cubieColor[5][6] = cubieTemp[5][8];
                    cubieColor[5][7] = cubieTemp[5][5];
                    cubieColor[5][8] = cubieTemp[5][2];

            break;
```

Solver.h

```
#ifndef SOLVER_H
#define SOLVER_H

#define to_hw_port (volatile char*) 0x00000080 // actual address here
#define to_hw_sig (volatile char*) 0x00000060 // actual address here
#define to_sw_port (char*) 0x00000070 // actual address here
#define to_sw_sig (char*) 0x00000050 // actual address here

#endif
```

Solver.c

```
#include <stdio.h>
#include <stdlib.h>
#include "solver.h"

int cubieColor[6][9];

int solver()
{
       int i, face, f;

       unsigned int colors[54];

       for(i = 0; i<54; i++)
              colors[i] = 0;
```

```
	// Start with pressing reset
	*to_hw_sig = 0;
	*to_hw_port = 0;
	printf("Press reset!\n");
	while (*to_sw_sig != 3);

	while (1)
	{
		*to_hw_sig = 3;

		printf("To Software Sig: %d", *to_sw_sig);
		while (*to_sw_sig != 2);

		*to_hw_sig = 1;
		while (*to_sw_sig != 1);
		face = *to_sw_port;
		*to_hw_sig = 2;
		while (*to_sw_sig != 0);
		f = (face - 1) * 9;

		for (i = f; i < f+9; i++)
		{
			*to_hw_sig = 1;
			while (*to_sw_sig != 1);
			colors[i] = *to_sw_port;
			*to_hw_sig = 2;
			while (*to_sw_sig != 0);
		}

		for(i = f; i<f + 9; i++)
			printf("%2x", colors[i]);
	}

	for(i=0; i<54; i++)
		cubieColor[i/9][i%9] = colors[i];

	return 0;
}
```

IO module (io_module.sv)

This module is used to setup the communication between hardware and software. It is a state machine that works based on the to_hw_sig. It assigns the next_state value and initializes to_hw_port and to_sw_port. The state machines transitions between multiple states like reset, wait, Read_msg, Ack_msg, Send_back, Read_key, Ack_key. The state machine goes through a read key cycle and a read message cycle. During this it assigns values to the to_sw_sig to tell the software the status. At the same time, it receives a to_hw_sig that corresponds to the software's status.

Module Descriptions:

The following are used in Qsys and are used specifically to establish communication between the hardware and the software components. They are used to keep track of the acknowledgement and Read signals between the hardware and the software.

Clock Module:

It is set at 50 MHz default frequency and this is the synchronous clock unit.

NIOS II Block:

This is the NIOS II processor block. It works with the clock input and the reset input. It gives access to data and instruction buses and specifies the hardware that we will be working with.

On-Chip Memory Block:

We add an on-chip memory unit that is of RAM Writable type. We use a total memory size of 16 bytes. This is the fastest memory source available as it is the one directly available on the chip and has minimum latency. We limit the memory size to 16 bytes to save on valuable on-chip memory and instead execute NIOS II programs from the Dynamic Random Access Memory.

SDRAM:

This is the SDRAM Controller unit. The Synchronous Dynamic RAM is used for saving the programs itself as the on-chip memory is limited. This RAM is larger in capacity but has a more complicated access and modify interface. SDRAM cannot be interfaced to the bus directly, as it has a complex addressing scheme and requires constant refreshing to retain data as it is made up of capacitors that save data as charge. Thus, we need a SDRAM controller IP core to interface the SDRAM to the Avalon bus. The table below shows the various parameters used to instantiate the controller:

SDRAM Parameter	Short Name	Parameter Value
Data Width	[width]	32 bits
# of Rows	[nrows]	13
# of Columns	[ncols]	10
# of Chip Selects	[ncs]	1
# of Banks	[nbanks]	4

SDRAM_PLL:

Phase-Locked Loop component is required to provide a clock signal for the SDRAM chip. SDRAM requires precise timings, and the PLL allows us to compensate for clock skew due to the board layout. We add a second clock with a phase shift of -3ns. This puts the clock going out to the SDRAM chip 3ns ahead of the controller clock (synchronous with the rest of the circuit) that is at 0ns.

System ID Peripheral:

This is added to check the compatibility of between hardware and software. It basically prevents software incompatible with NIOS II from loading onto the FPGA. It is used for debug and verification purposes.

jtag_uart_0:

This block gives the ability to use printf commands from the NIOS II Processor which go through the programming cable via USB. This is set on the interrupt request 5 so that the requests from other blocks are not preempted by this one. Sending and receiving texts over the console is a slow task and thus having it on IRQ 5 will ensure that the CPU is not blocked when a print interrupt request is being handled and the control is immediately returned back to the program

with the main execution being handled in a tasklet which is an interrupt setup at the end of transmission for each buffer. The CPU doesn't have to be blocked while waiting for each character to be transmitted, and is only interrupted when the buffer is full, that is, the peripheral needs more data.

to_hw_port_PIO_Block:

It is responsible for taking an 30-bit pixel Value from C to hardware code.

to_hw_sig_PIO_Block:

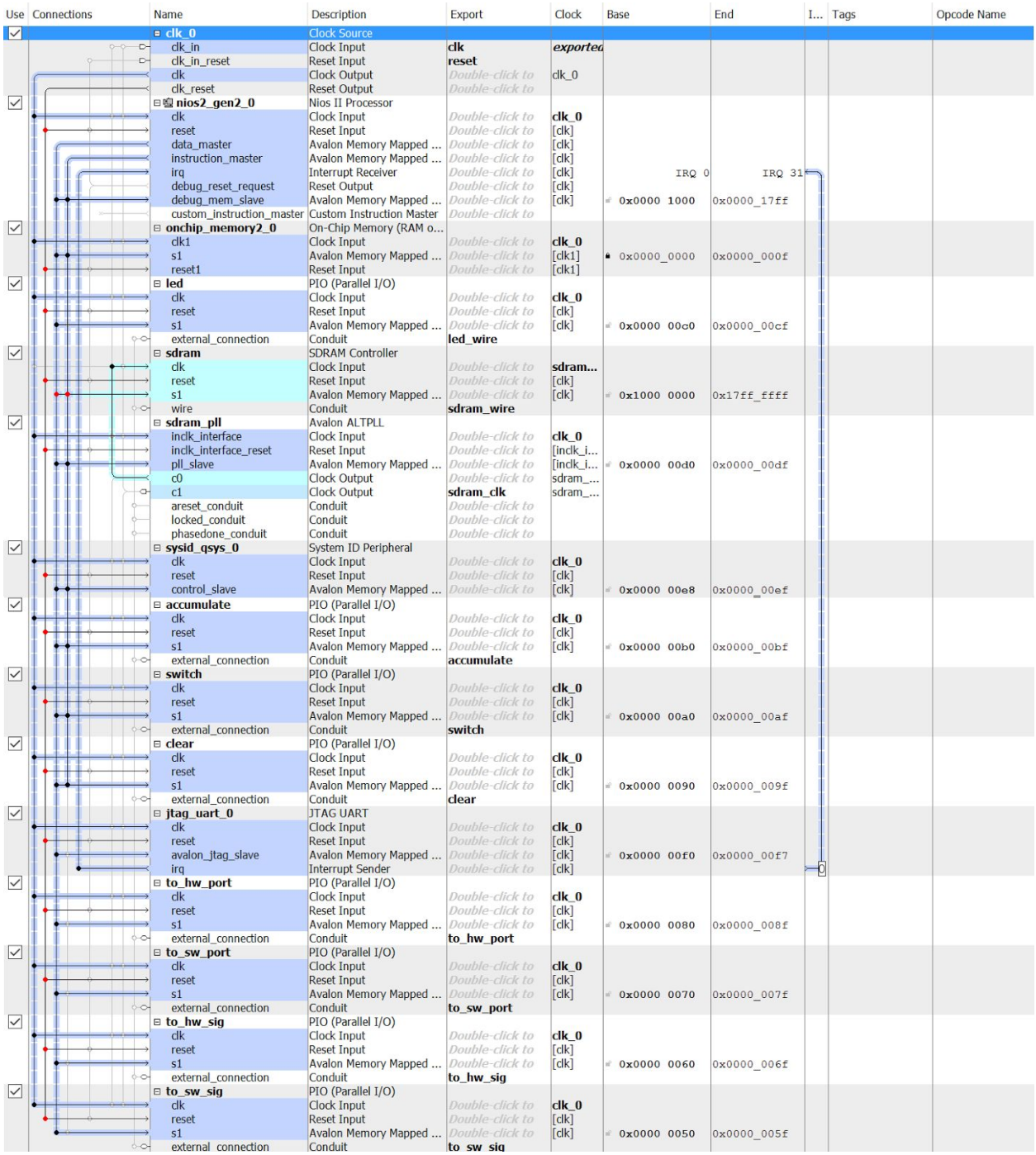
This 2-bit PIO block is used to establish the handshake protocol by notifying the hardware that message has been sent and computations can be carried out on it. This is also an input to the FSM.

to_sw_port_PIO_Block:

The 30 bit PIO Input blocks takes in 30 bit pixel value from hardware code to the C code.

to_sw_sig_PIO_Block:

This 2-bit PIO block is used to establish the handshake protocol by notifying the software C code that message has been received by the hardware.



Use	Connections	Name	Description	Export	Clock	Base	End	I...	Tags	Opcode Name
☑		**clk_0**	Clock Source							
		clk_in	Clock Input	**clk**	*exported*					
		clk_in_reset	Reset Input	**reset**						
		clk	Clock Output	*Double-click to*	clk_0					
		clk_reset	Reset Output	*Double-click to*						
☑		**nios2_gen2_0**	Nios II Processor							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		data_master	Avalon Memory Mapped ...	*Double-click to*	[clk]					
		instruction_master	Avalon Memory Mapped ...	*Double-click to*	[clk]					
		irq	Interrupt Receiver	*Double-click to*	[clk]	IRQ 0	IRQ 31			
		debug_reset_request	Reset Output	*Double-click to*	[clk]					
		debug_mem_slave	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000_1000	0x0000_17ff			
		custom_instruction_master	Custom Instruction Master	*Double-click to*						
☑		**onchip_memory2_0**	On-Chip Memory (RAM o...							
		clk1	Clock Input	*Double-click to*	**clk_0**					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk1]	0x0000_0000	0x0000_000f			
		reset1	Reset Input	*Double-click to*	[clk1]					
☑		**led**	PIO (Parallel I/O)							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000_00c0	0x0000_00cf			
		external_connection	Conduit	**led_wire**						
☑		**sdram**	SDRAM Controller							
		clk	Clock Input	*Double-click to*	**sdram...**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x1000_0000	0x17ff_ffff			
		wire	Conduit	**sdram_wire**						
☑		**sdram_pll**	Avalon ALTPLL							
		inclk_interface	Clock Input	*Double-click to*	**clk_0**					
		inclk_interface_reset	Reset Input	*Double-click to*	[inclk_i...					
		pll_slave	Avalon Memory Mapped ...	*Double-click to*	[inclk_i...	0x0000_00d0	0x0000_00df			
		c0	Clock Output	*Double-click to*	sdram_...					
		c1	Clock Output	**sdram_clk**	sdram_...					
		areset_conduit	Conduit	*Double-click to*						
		locked_conduit	Conduit	*Double-click to*						
		phasedone_conduit	Conduit	*Double-click to*						
☑		**sysid_qsys_0**	System ID Peripheral							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		control_slave	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000_00e8	0x0000_00ef			
☑		**accumulate**	PIO (Parallel I/O)							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000_00b0	0x0000_00bf			
		external_connection	Conduit	**accumulate**						
☑		**switch**	PIO (Parallel I/O)							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000_00a0	0x0000_00af			
		external_connection	Conduit	**switch**						
☑		**clear**	PIO (Parallel I/O)							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000_0090	0x0000_009f			
		external_connection	Conduit	**clear**						
☑		**jtag_uart_0**	JTAG UART							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		avalon_jtag_slave	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000_00f0	0x0000_00f7			
		irq	Interrupt Sender	*Double-click to*	[clk]			0		
☑		**to_hw_port**	PIO (Parallel I/O)							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000_0080	0x0000_008f			
		external_connection	Conduit	**to_hw_port**						
☑		**to_sw_port**	PIO (Parallel I/O)							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000_0070	0x0000_007f			
		external_connection	Conduit	**to_sw_port**						
☑		**to_hw_sig**	PIO (Parallel I/O)							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000_0060	0x0000_006f			
		external_connection	Conduit	**to_hw_sig**						
☑		**to_sw_sig**	PIO (Parallel I/O)							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0000_0050	0x0000_005f			
		external_connection	Conduit	**to_sw_sig**						

Challenges Faced:

We got some parts of the project working fine and while some worked in simulation, we couldn't get it to work together. But we are sure that it was a small bug that went unnoticed that caused the glitch The color detection module and the io module handshakes worked just

fine in simulation but failed to work when were put together and were flashed to hardware. There were a lot of other design issues that we faced along the course of the project. Getting that camera to work was itself a major task by itself. We also spent a lot of time trying to figure out the pixel values that the camera received as those would be required for color detection purposes but that was an extremely difficult task to do considering how the camera outputted data to the VGA with funky colors because of bit shifts and data width restrictions.

Because of the modular requirements of the cube, we also faced design challenges while deciding color averaging and filtering choices. Moreover making sure that all the signals reached the required modules at the right moment without clock skews was another important challenge that we managed to tackle but couldn't manage to display in physical demo. We did sort the issues in simulation. The FSM for this was massive.

The task of mapping to VGA of the 54 blocks was another challenge as all the pixel values has to be taken care of as printing one block could result in clearing of the previous blocks otherwise. Mapping of the non-deterministic number of moves implied not knowing how many characters to take care of. We built a FSM to tackle this issue that would work with user input but it didn't print beyond a character without glitch.

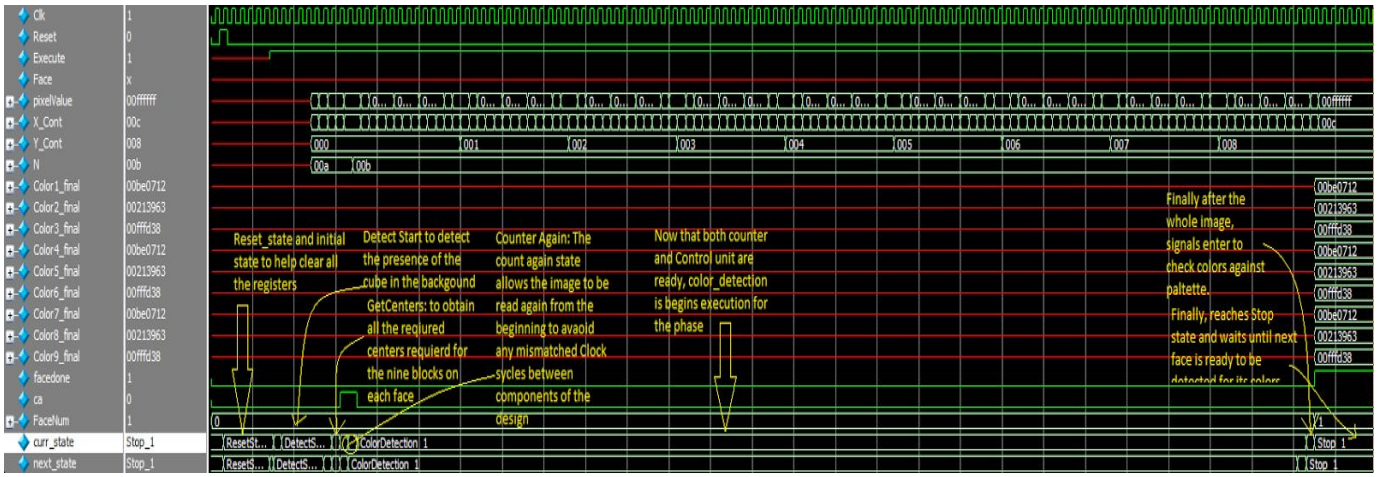
The algorithm that we modified and the input system that we created in C worked well otherwise and displayed the simulation in software just fine.

Conclusion

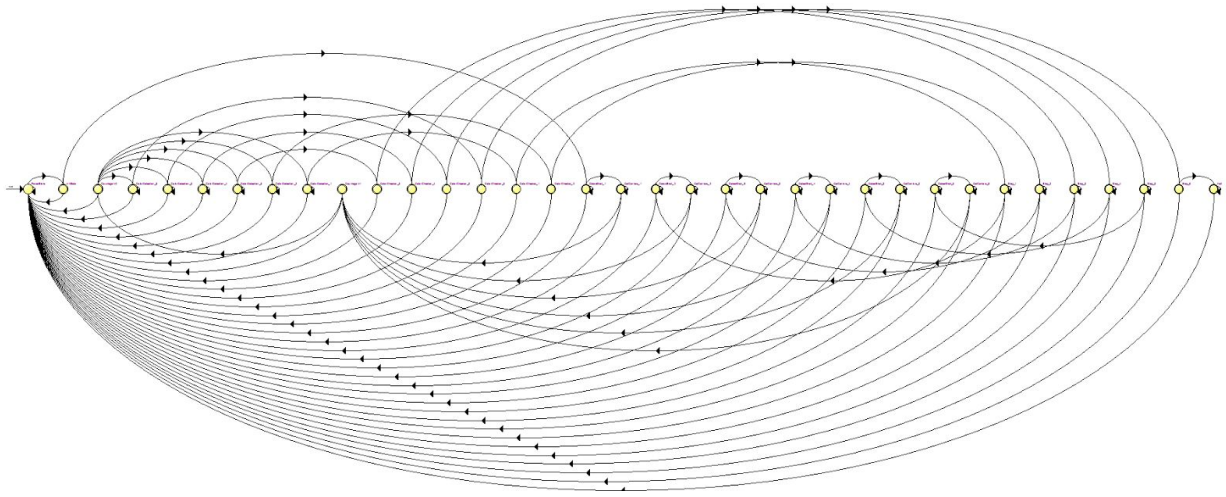
We enjoyed working on the project even if we did not manage to achieve all the goals that we had set out with. We believe that had we managed time slightly better, we surely could have fixed the bugs but it was an extremely good learning experience overall. We learnt a lot of skills and got the chance to put our skills to use.

Appendix

LUT	2678
DSP	-
Memory (BRAM)	165,888
Flip-Flops	2233
Frequency	165.02 MHz
Static Power	102.01 mW
Dynamic Power	0.92 mW
Total Power	181.74 mW



Annotated Simulation of color detection Testbench



FSM for color detection

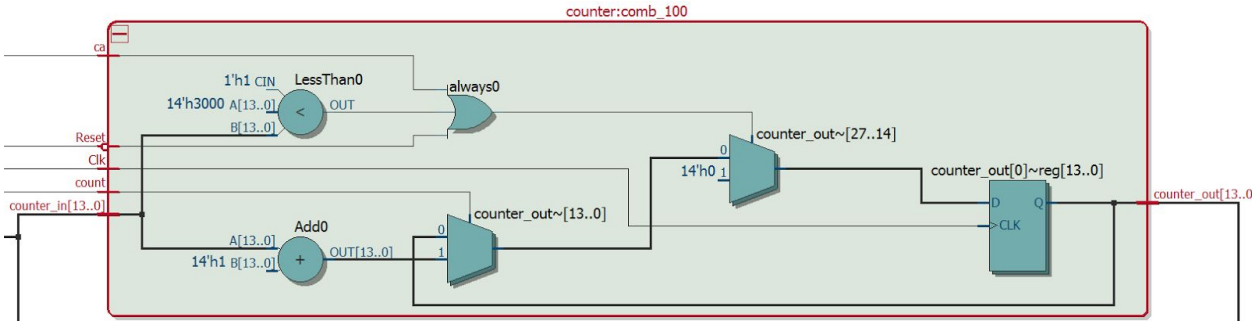


Figure1: Counter (for addresses)

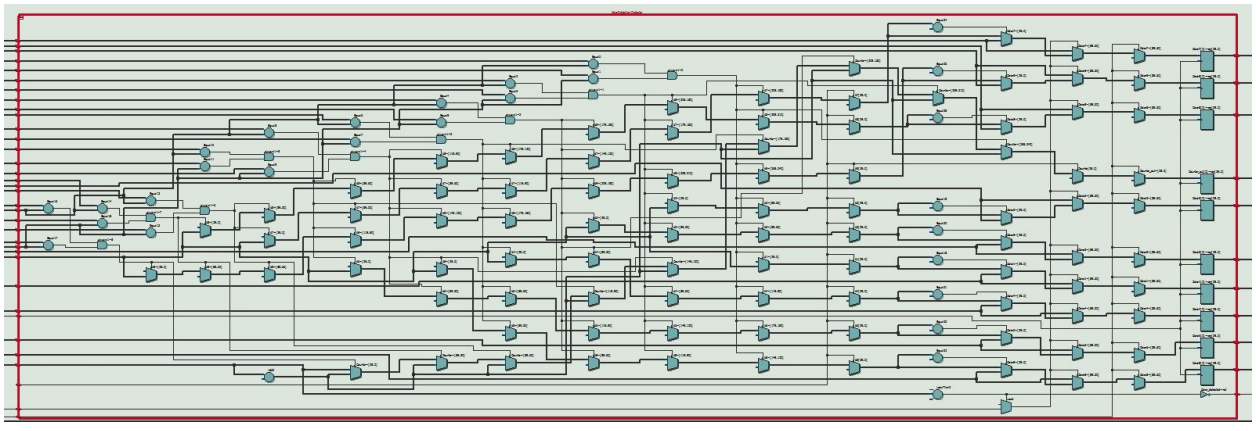


Figure2: Color Detection

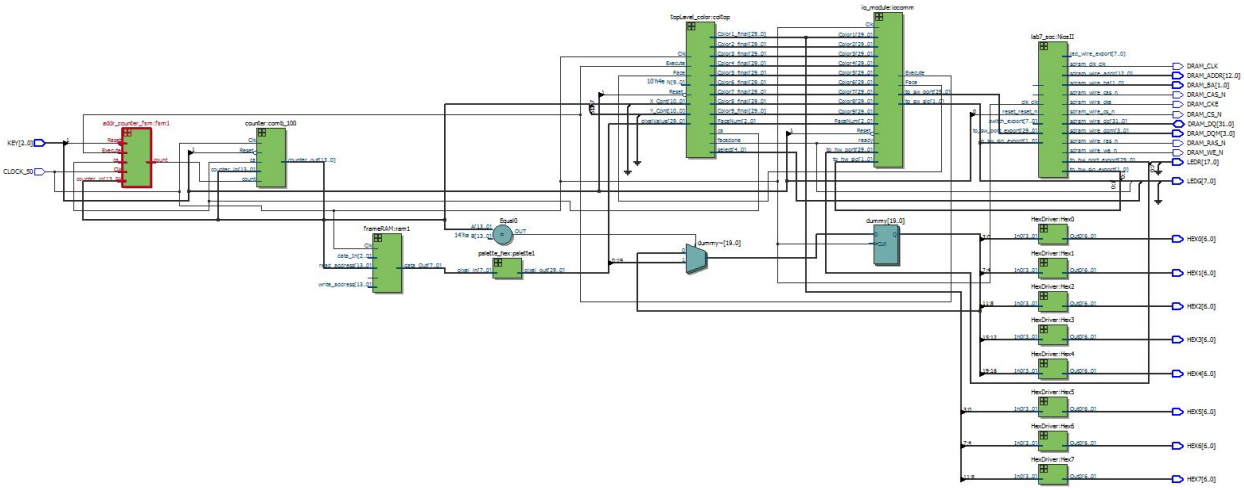


Figure3: TopLevel (for color detection)

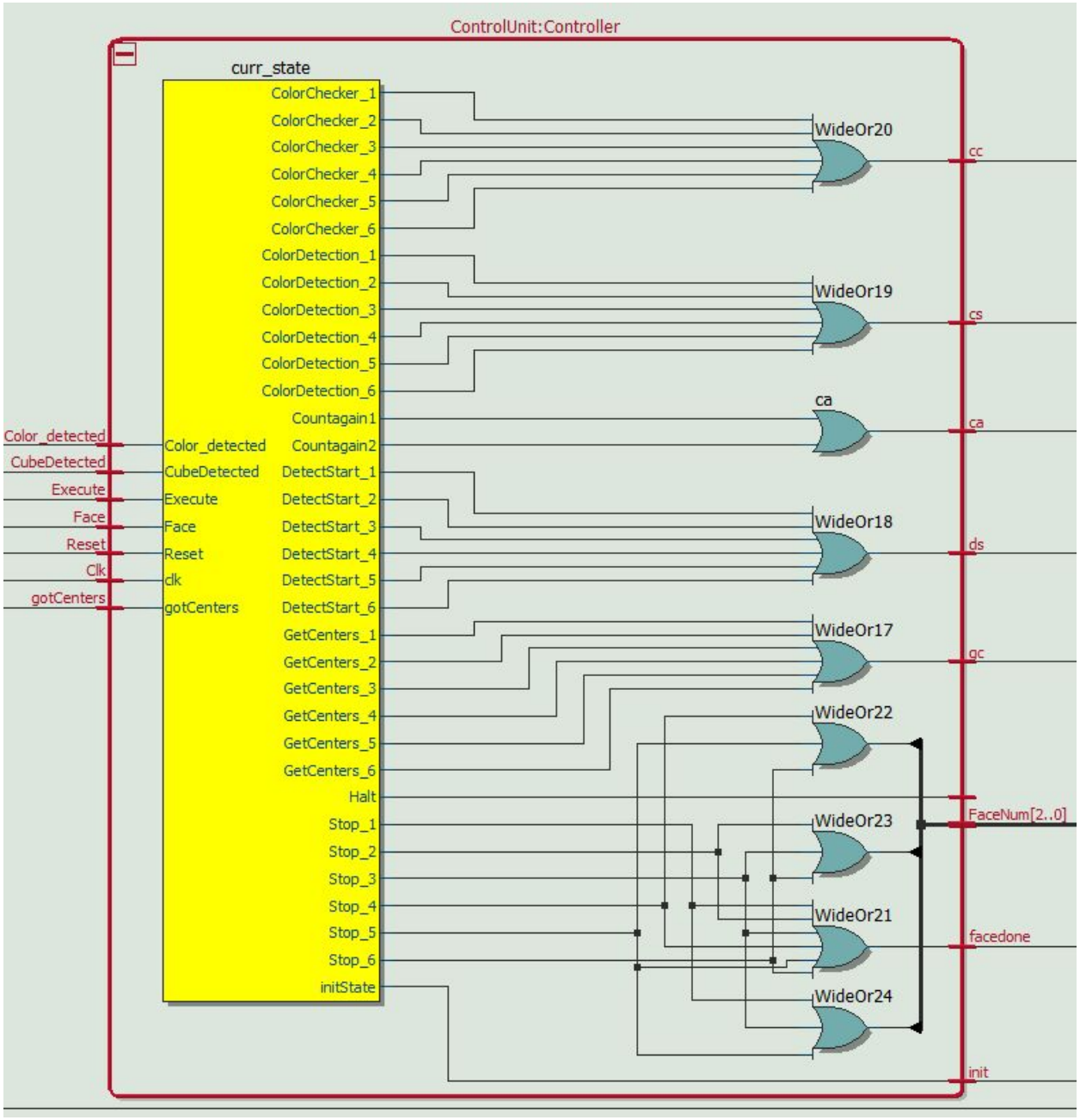


Figure4: Control Unit

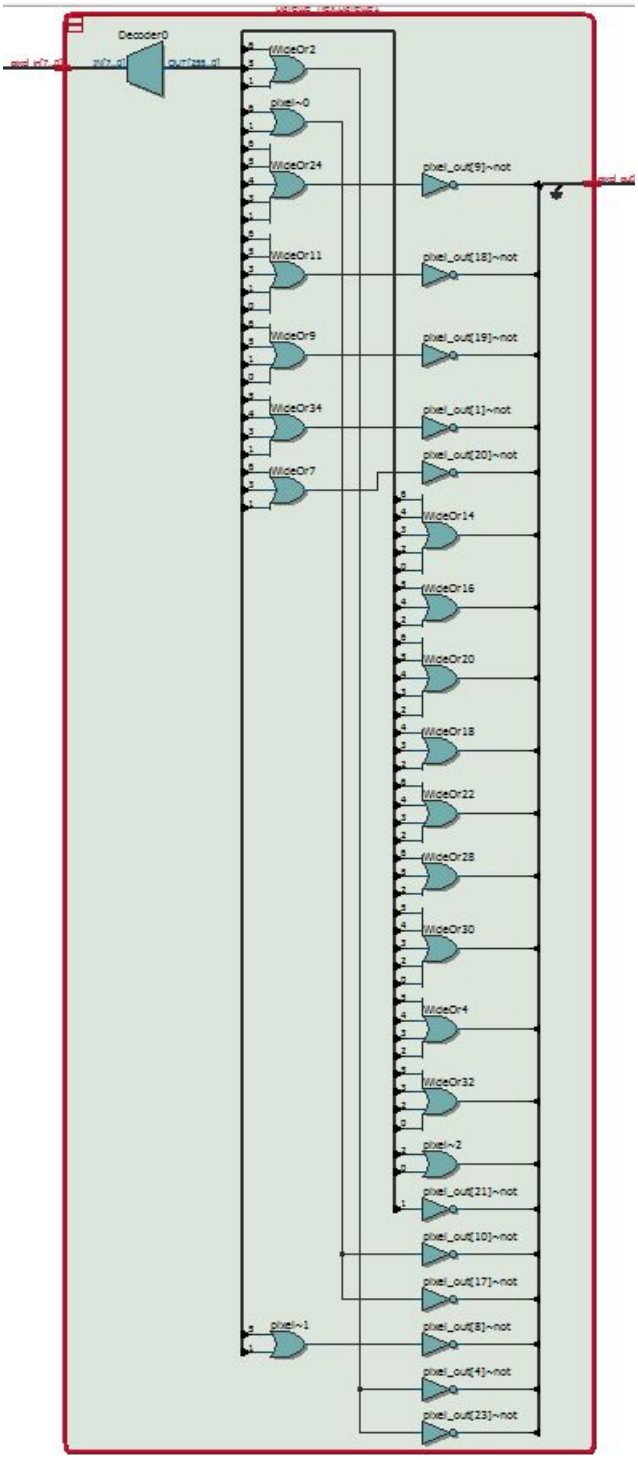


Figure5: palette hex

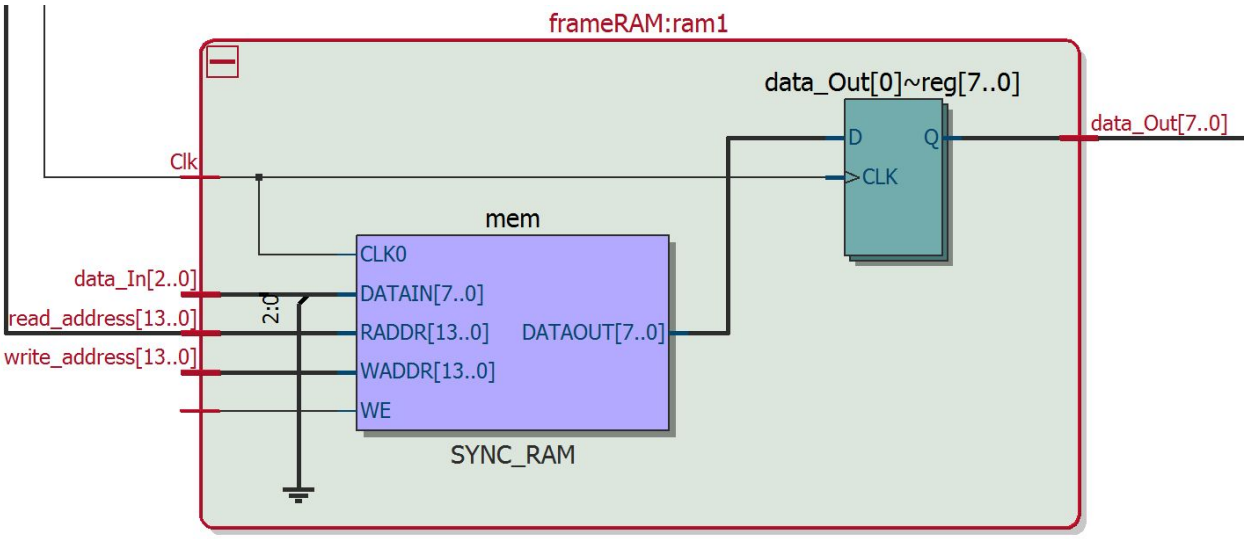


Figure6: frame RAM

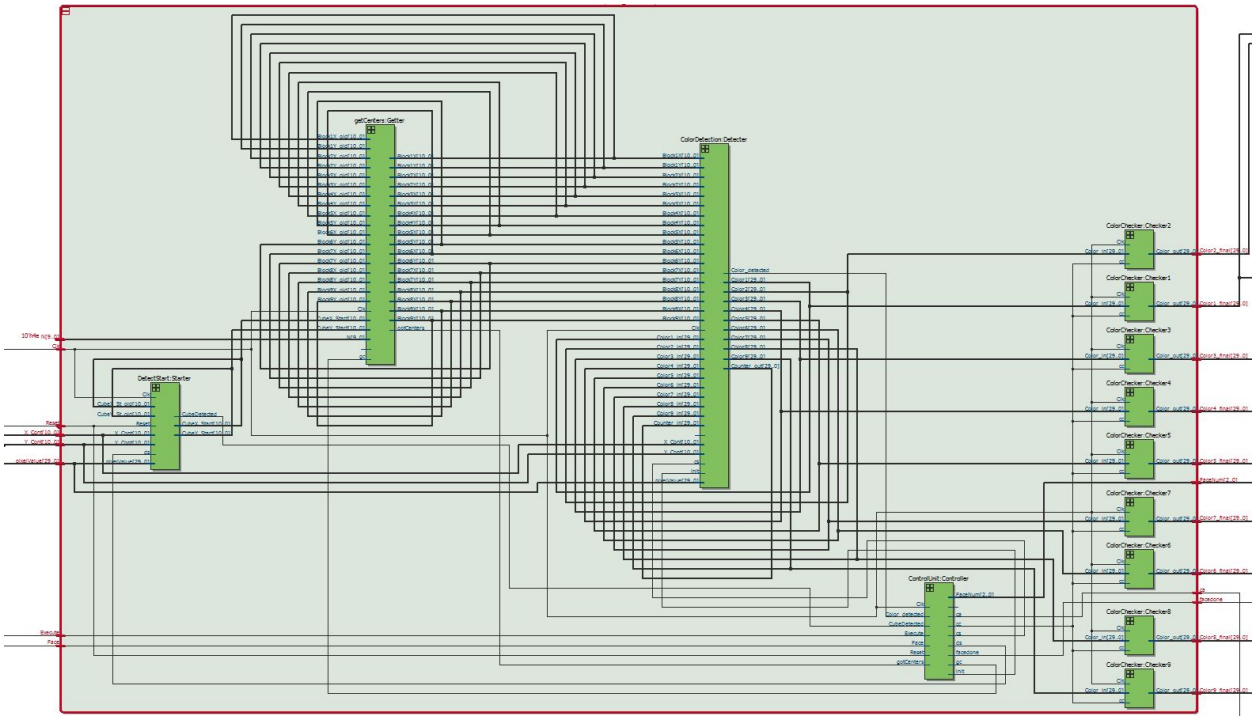


Figure7: Top Level (color)

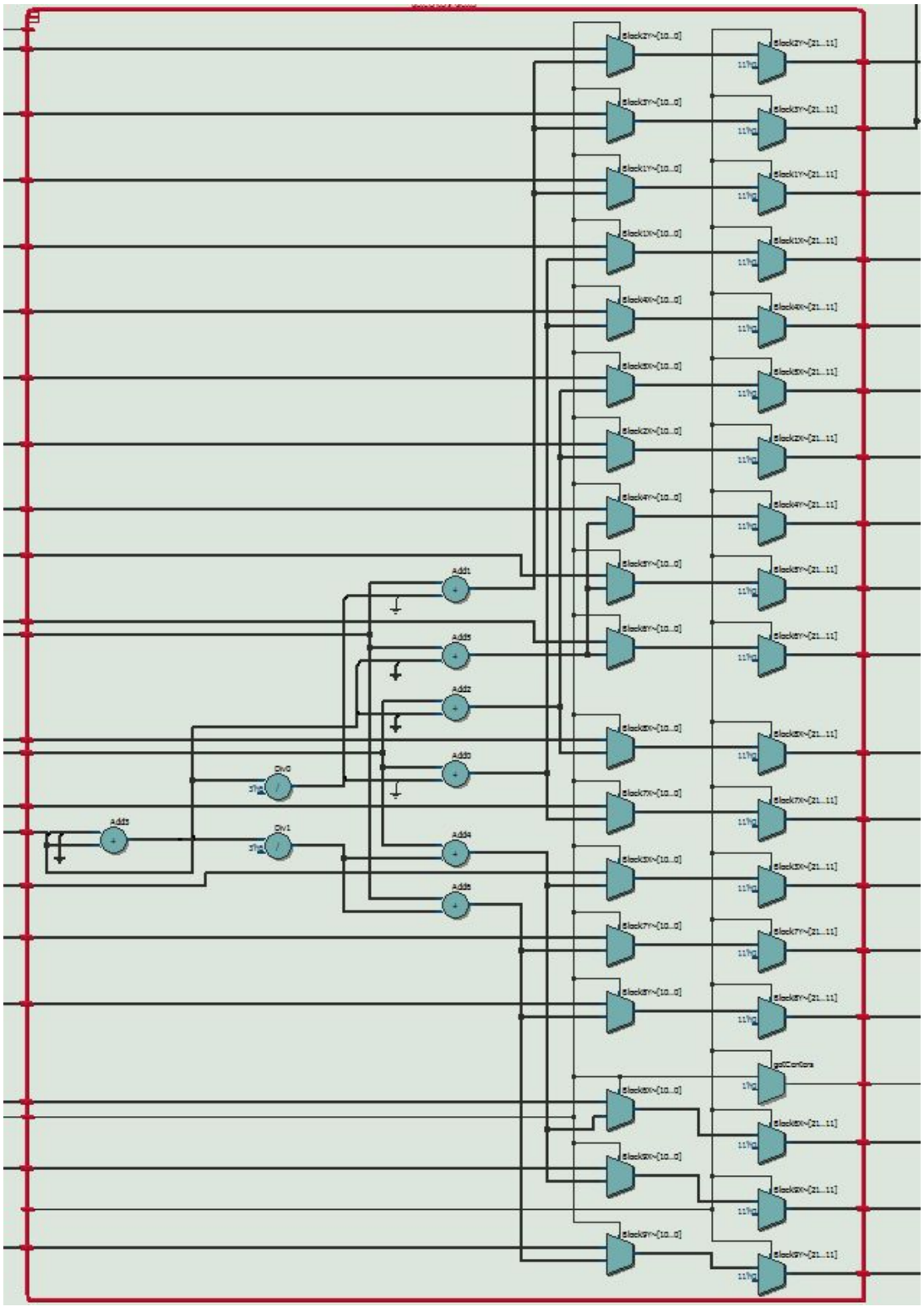


Figure8: GetCenters

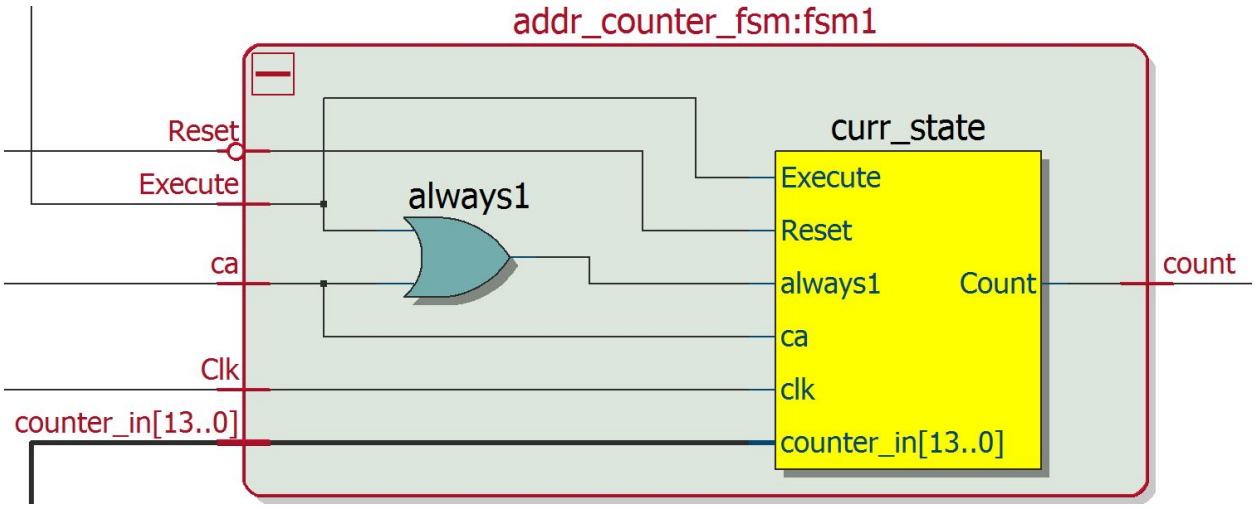


Figure9: address counter FSM

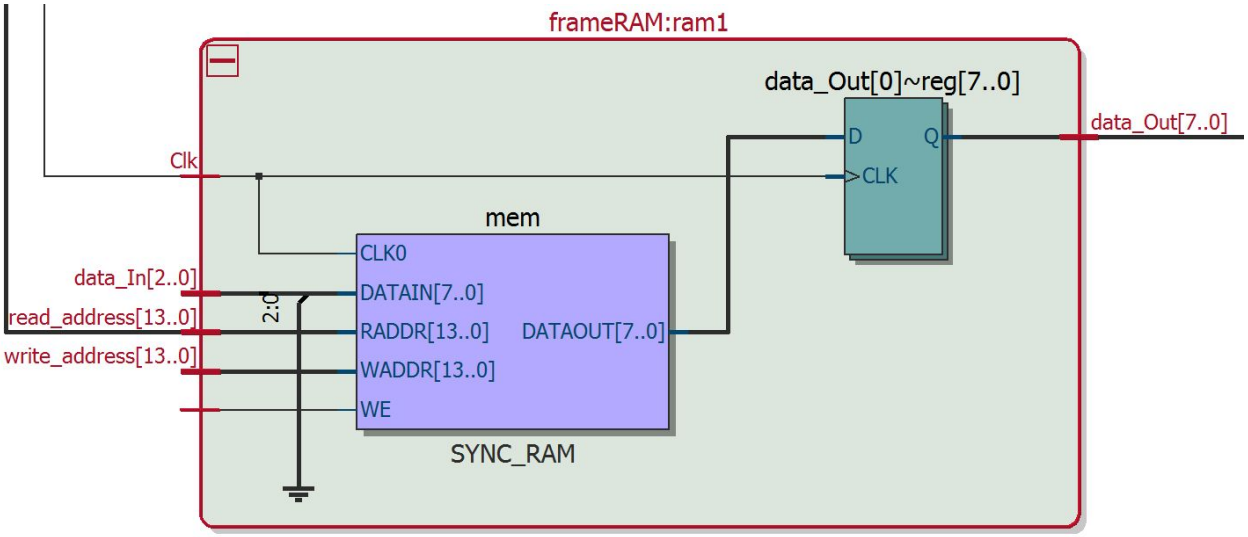


Figure10: RAM