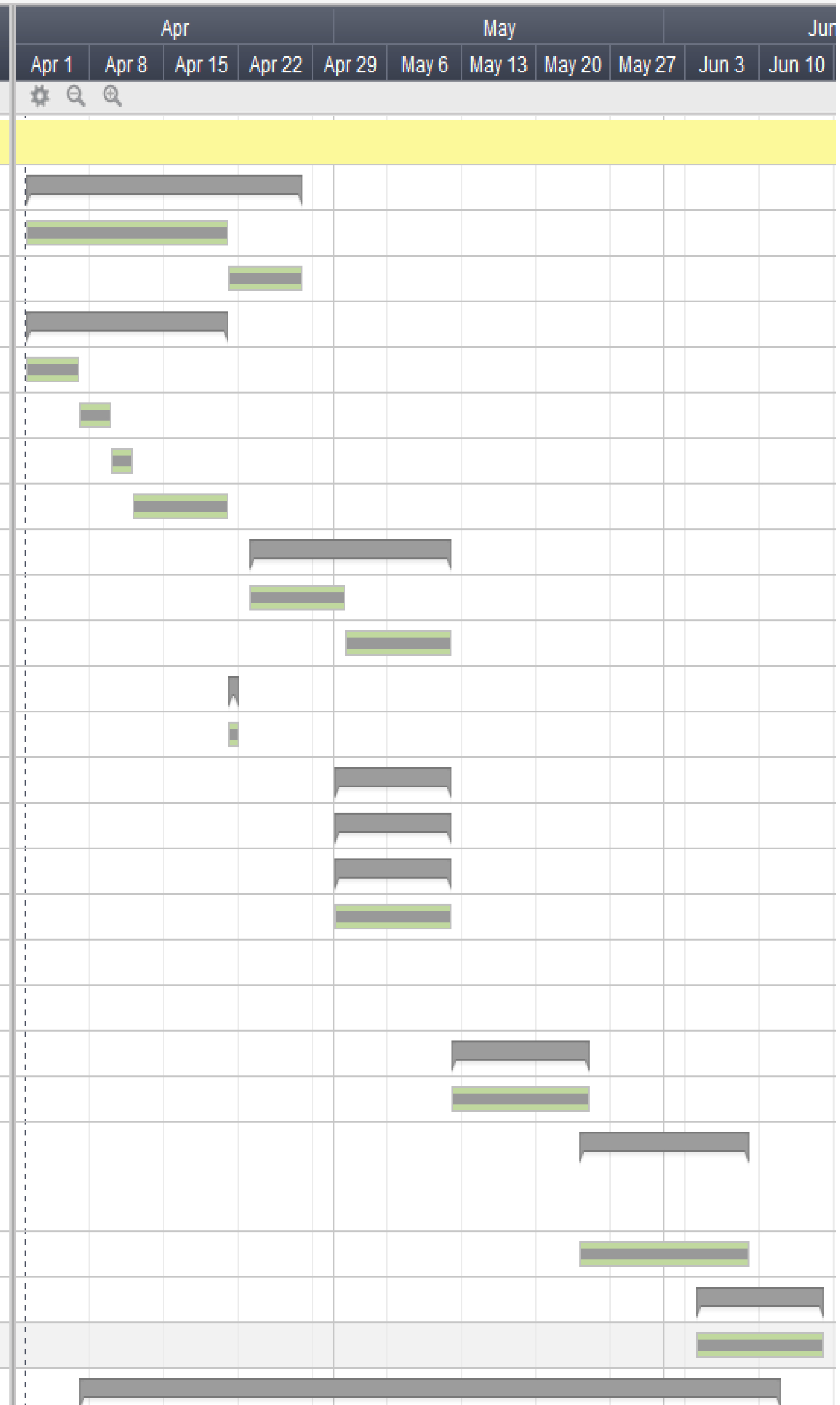
PUPILS DETECTION

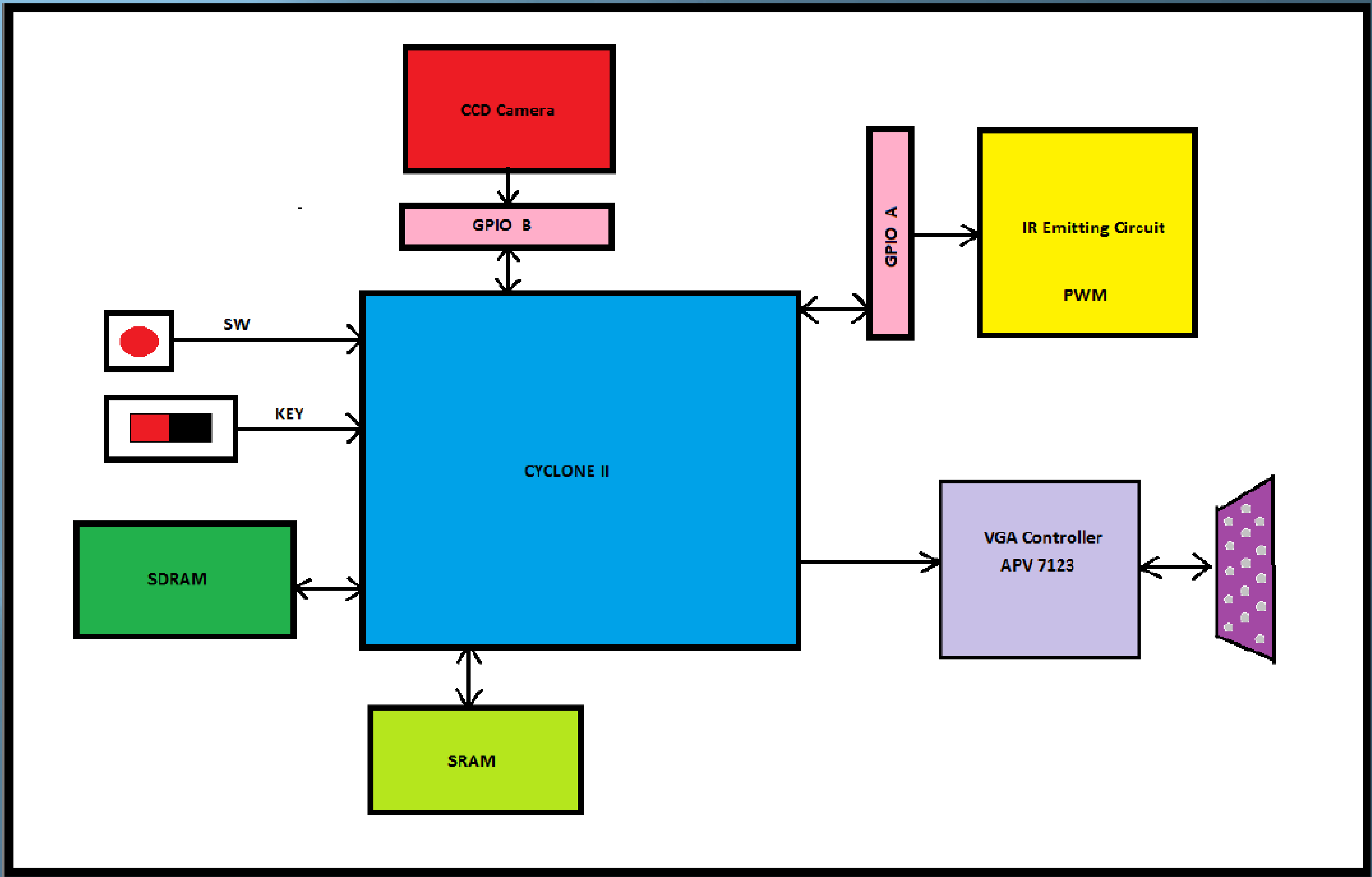
Christopher Upton
Yohannes Kassa
Pak Chan

Table of Contents

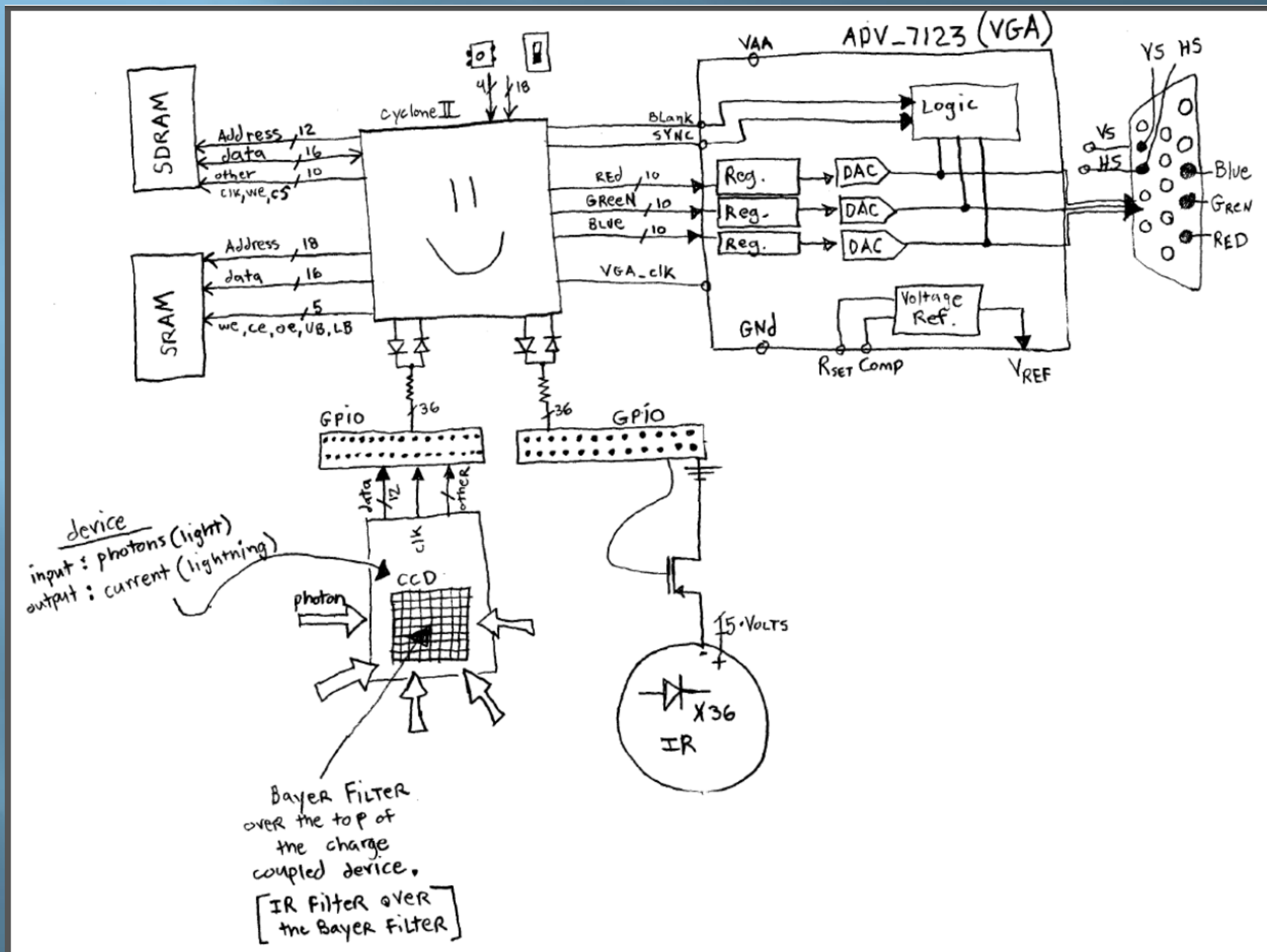
Task Name	Start Date	End Date
<< Tips on Using this Template		
Fine-tune Hadware	**04/02/12**	**04/27/12**
Find good yet inexpensive IR Filter	04/02/12	04/20/12
Trim the new IR Filter to size	04/21/12	04/27/12
Breadboard attachment -- Mount IR LEDs further from camera's lens	04/02/12	04/20/12
Find more breadboard	04/02/12	04/06/12
Trim breadboard to size	04/07/12	04/09/12
Attach breadboard attachments (off-axis)	04/10/12	04/11/12
Solder and wire-wrap IR LEDs	04/12/12	04/20/12
Crosshairs	04/23/12	05/11/12
Write a Verilog module that will pin-point the center of a circle	04/23/12	05/01/12
Write another module drawing crosshairs across specific pixel rows/columns	05/02/12	05/11/12
Frame Subtraction	**04/21/12**	**04/21/12**
SRAM-SDRAM --> SDRAM-SRAM	04/21/12	04/21/12
RGB -> USB -> Windows Application	05/01/12	05/11/12
Figure out what sequence of data needs to be sent through the USB	05/01/12	05/11/12
Find and read the USB video specification documents	05/01/12	05/11/12
USB.com	05/01/12	05/11/12
Find and read the USB audio specification documents		
USB.com		
Write Verilog code that converts VGA RGB signals to USB video signals	05/12/12	05/24/12
Send appropriate video-data through USB cable	05/12/12	05/24/12
Implement the USB video class driver into a MSVS C++ program (Usbvideo.sys). Goto this websaite for more information: http://msdn.microsoft.com/en-us/library/windows/hardware/ff568649(v=vs.85).aspx	05/24/12	06/08/12
See also: http://msdn.microsoft.com/en-us/library/ms697062(v=vs.85).aspx	05/24/12	06/08/12
Do as many projects as possible	06/04/12	06/15/12
ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf	06/04/12	06/15/12
Documentation	**04/07/12**	**06/11/12**



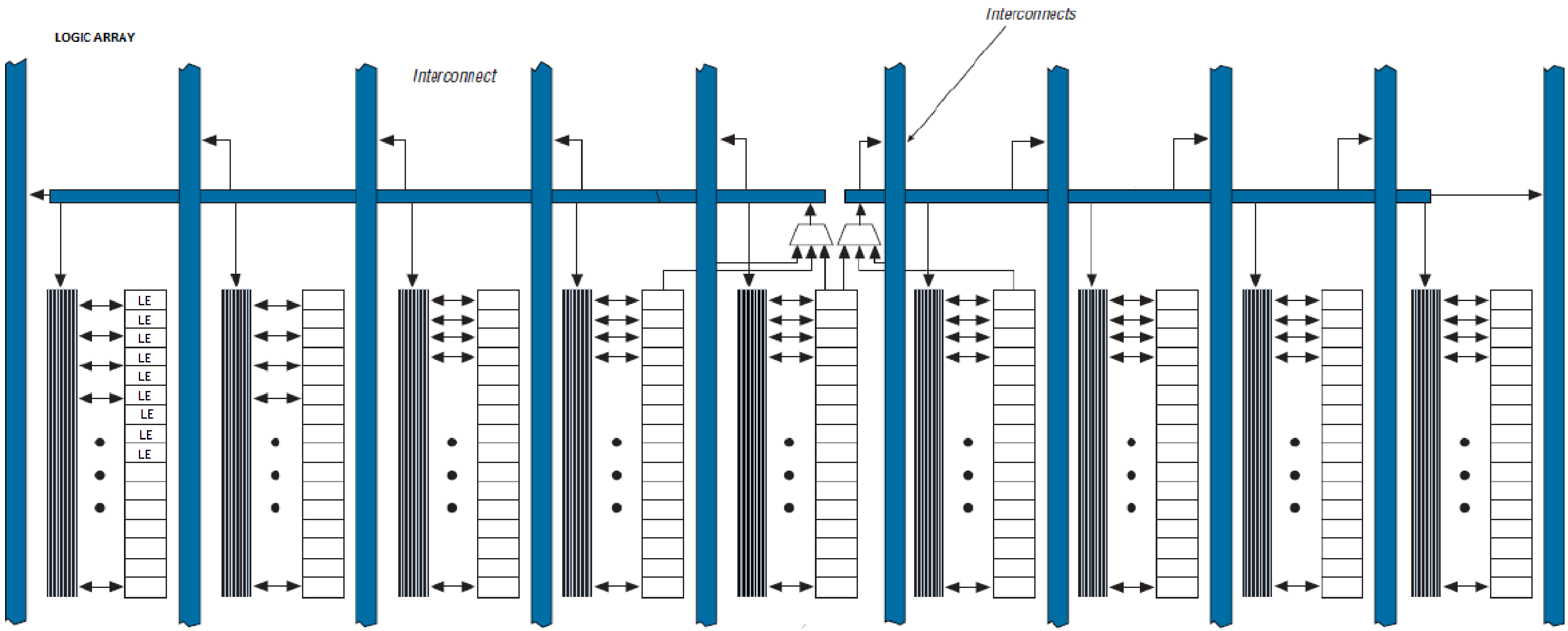
Block Diagram



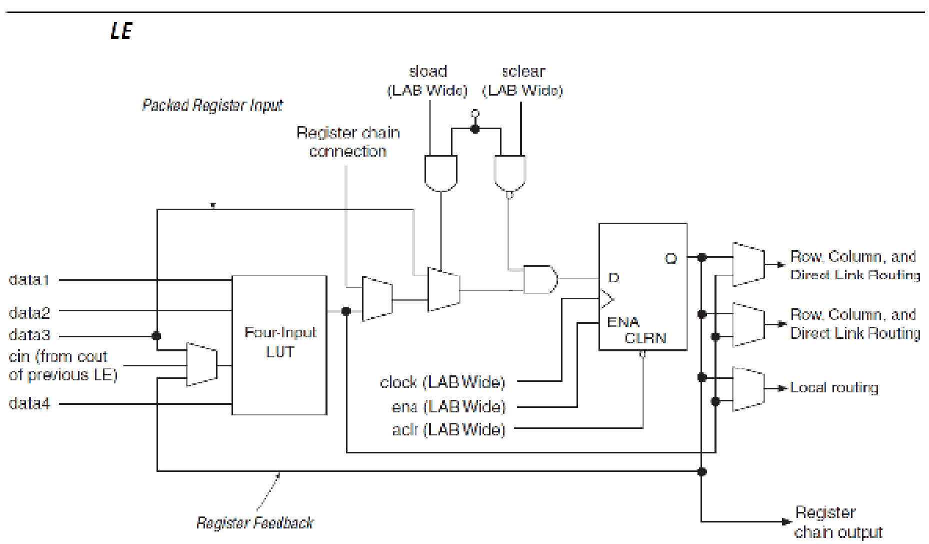
Block Diagram



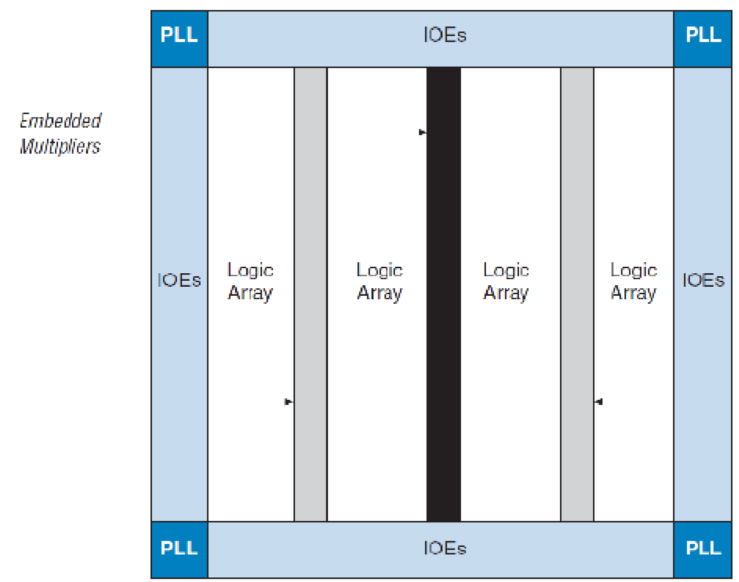
CYCLONE II



LOGIC ARRAY
Interconnect
Interconnects
LE
LE
LE
LE
LE
LE
LE
LE
LE



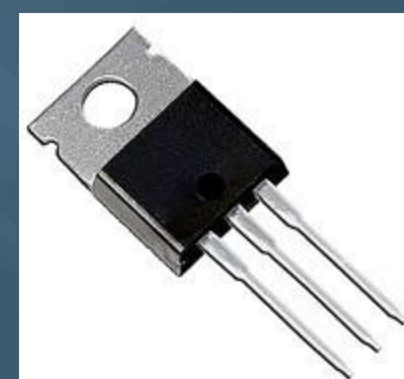
LE
Packed Register Input
sload
(LAB Wide)
sclear
(LAB Wide)
Register chain
connection
data1
data2
data3
cin (from cout
of previous LE)
data4
Four-Input
LUT
D
Q
ENA
CLRN
clock (LAB Wide)
ena (LAB Wide)
aclr (LAB Wide)
Row, Column, and
Direct Link Routing
Row, Column, and
Direct Link Routing
Local routing
Register
chain output
Register Feedback



Embedded
Multipliers
PLL
IOEs
PLL
IOEs
Logic
Array
Logic
Array
Logic
Array
Logic
Array
IOEs
PLL
IOEs
PLL

Materials

- MOSFET Transistor
- 2″ Round Board 36 IR LEDs
- 850nm Infrared IR Pass Filter
- BreadBoard + Headers + Jumper Cables
- ~2 Watts

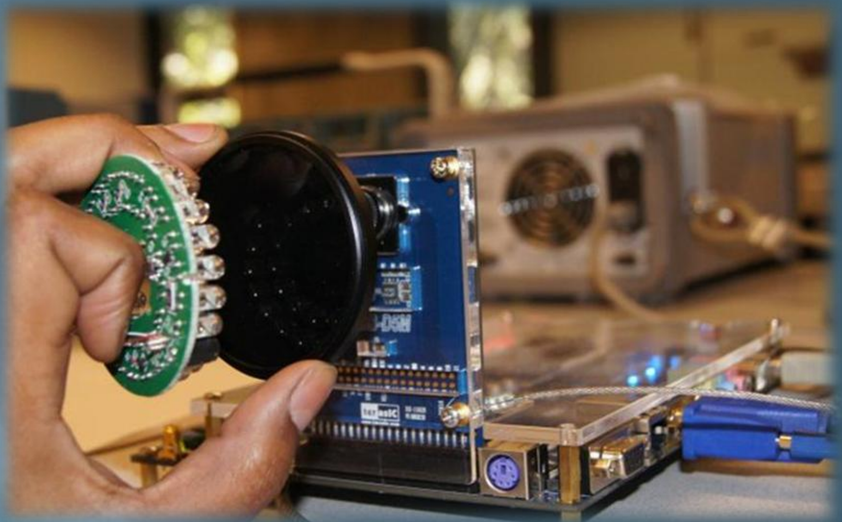


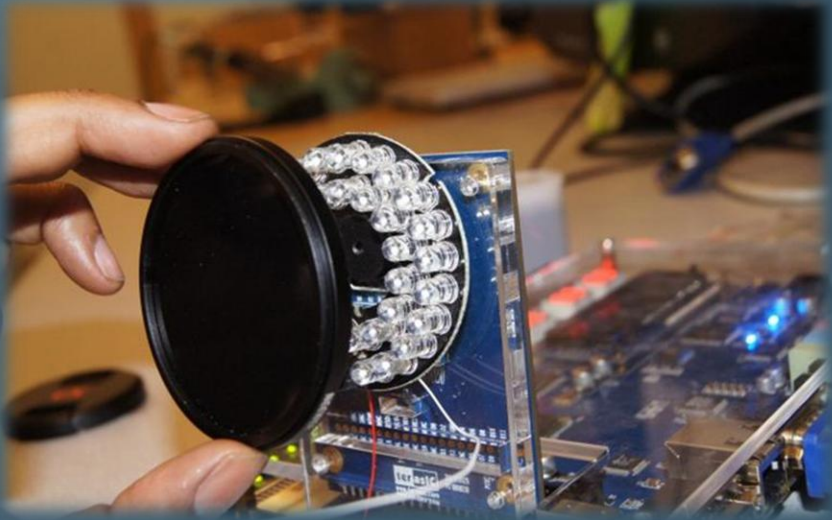
Materials

- Altera DE2 Starter Kit
  - Terasic D5M Camera
  - Quartus II
  - Cyclone II
  - PC + USB-Blaster

Implementation

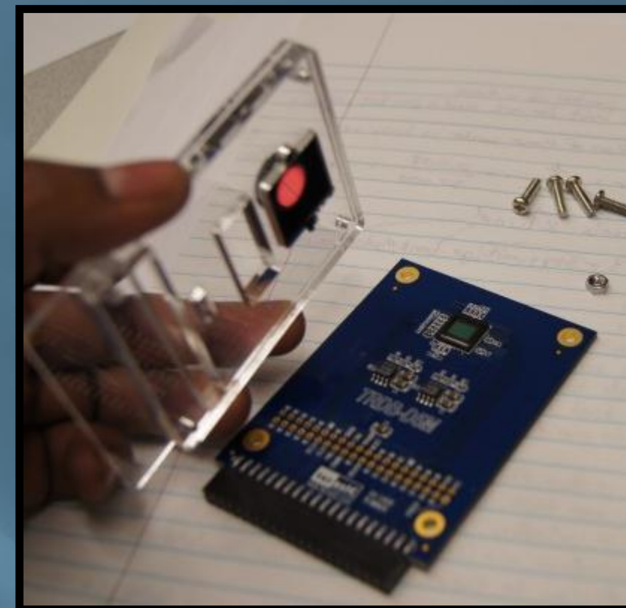
- Step 1: Bring the system up.
  - Upload Quartus II Video Camera Project.
    - Install Altera-Usb Driver.
  - Supply the IR Circuit w/ 12 Volts
  - Use the IR filter to filter IR light.

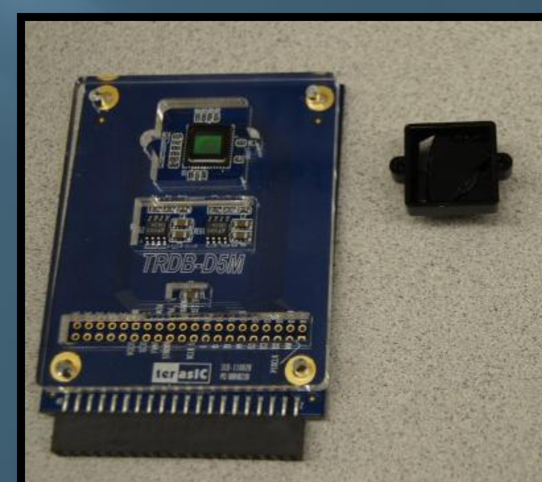




Implementation

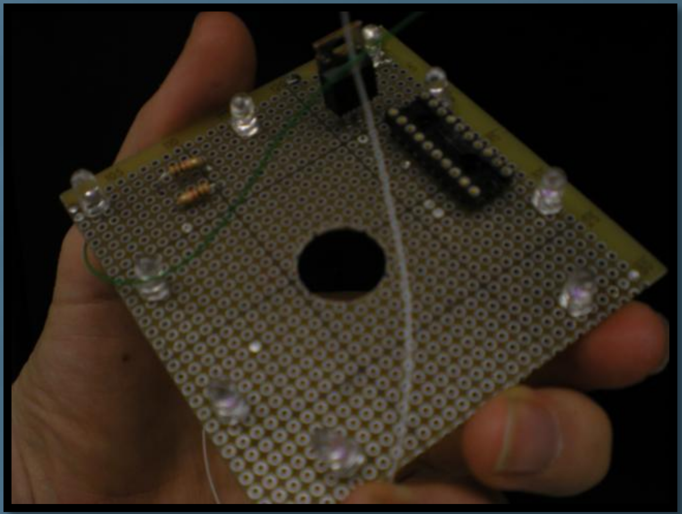
- Step 2: Remove the IR-Blocking Filter
  - The D5M comes with a color filter
  - This color filter is blocking IR light
    - Crack the color filter
    - Remove the color filter
- Step 3: Fit the IR filter into the Camera
  - Trim the IR filter to size.
  - Fit the IR filter into the Camera.

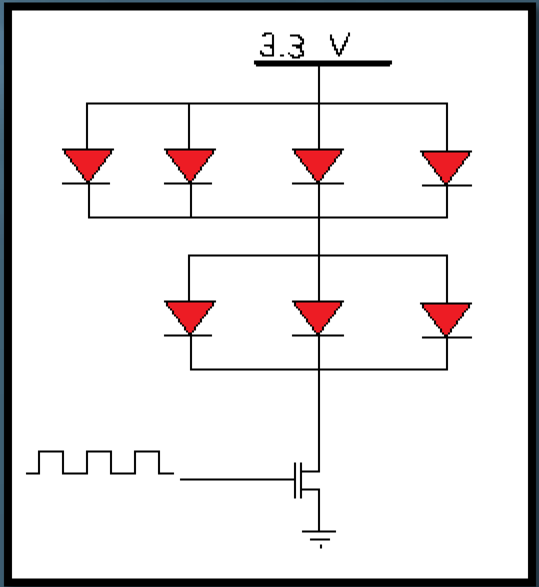




Implementation

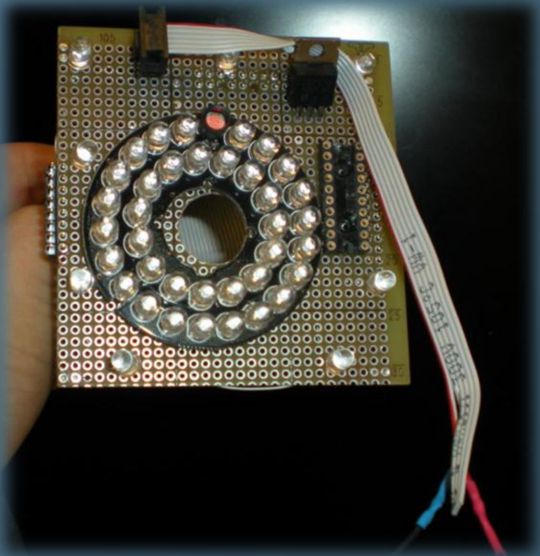
- Step 4: Board Drilling
  - Drill a hole to fit around the D5M camera lens.
  - Machine room in Jack Baskin Basement has a drill.
- Step 5: Design a Circuit
  - Wire-wrap, solder
  - Wire-unwrap, unsolder
  - Derive a circuit meeting our Voltage/Current requirements
    - ~3.3 Volts
    - 100-125 mAmps
    - Series/Parallel Combination



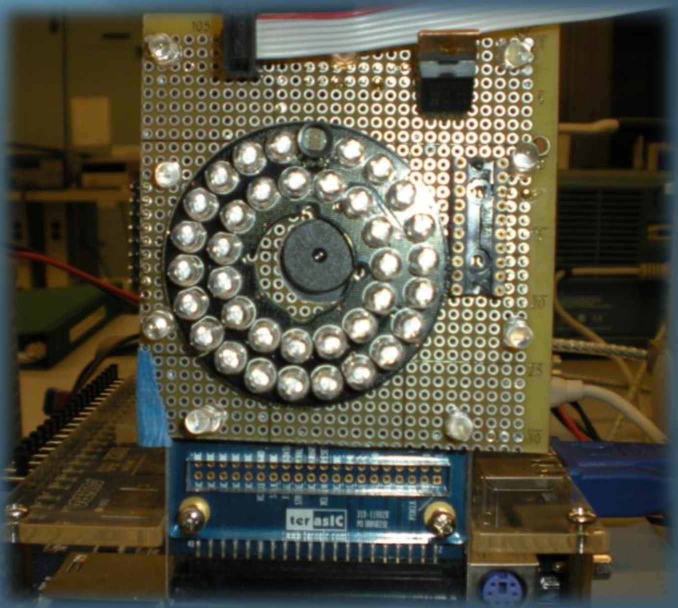


Implementation

- Step 6: Width Modulation
  - Output Header GPIO
  - 40 Pins @ 3.3 Volts
  - Write Verilog code to control GPIO.
  - Flash the GPIO signal every unit time.
  - Flash the LEDs every unit time.
  - Period = 1 second
  - Duty Cycle = 50%

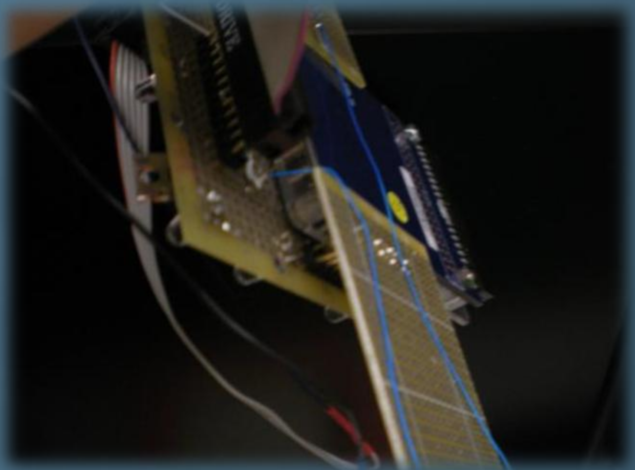


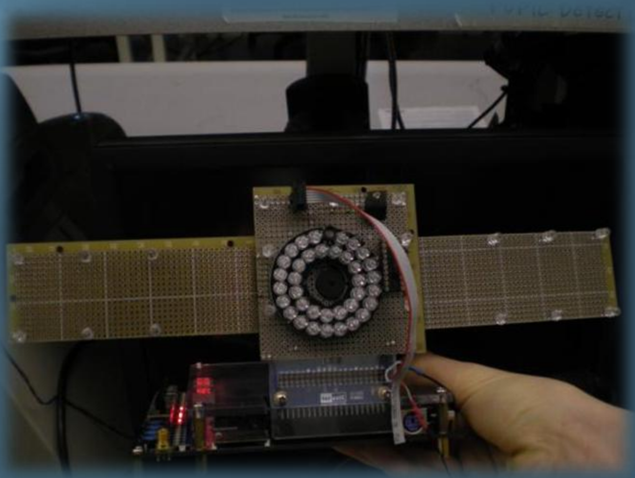
- Step 7: Tune Up
  - Minimize wire length
  - Use the 36 LED board
  - Use Headers and Jumpers
  - Solder Pins in place
  - Focus the lens



Implementation

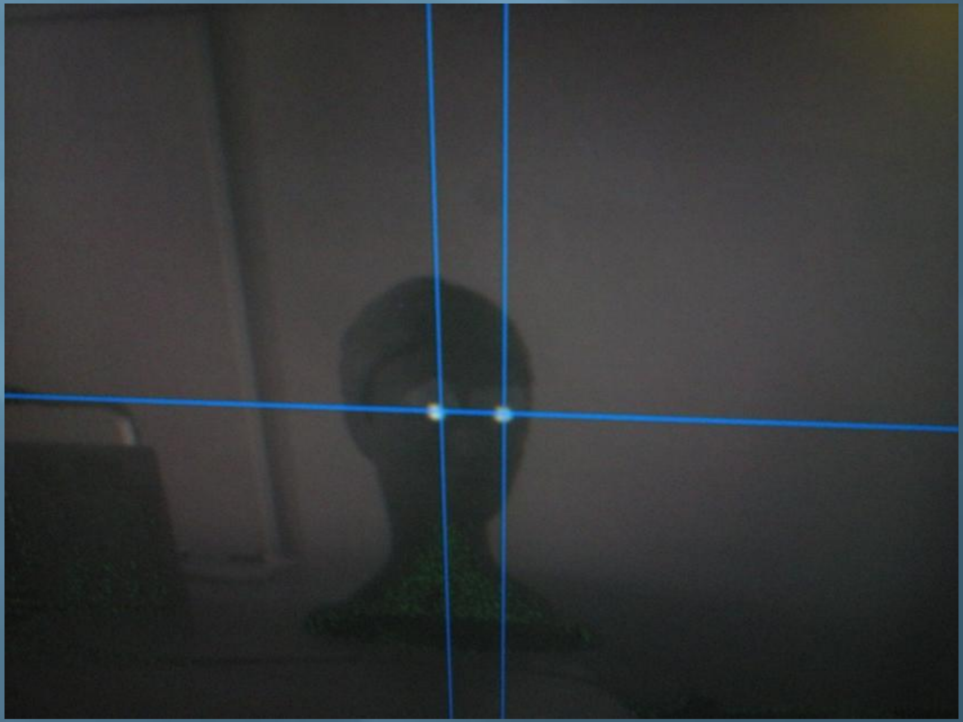
- Step 8: More LEDs
  - Increase the range of detection
    - Decreases the amount of reflected light.
    - Reflection proportional to
      - (amount of light) / ((distance)^2)
- Step 9: Secure IR-emitter-board to the camera.
    - Solder IR LEDs to the breadboard.
    - Using headers and solder, brace the camera between breadboards.



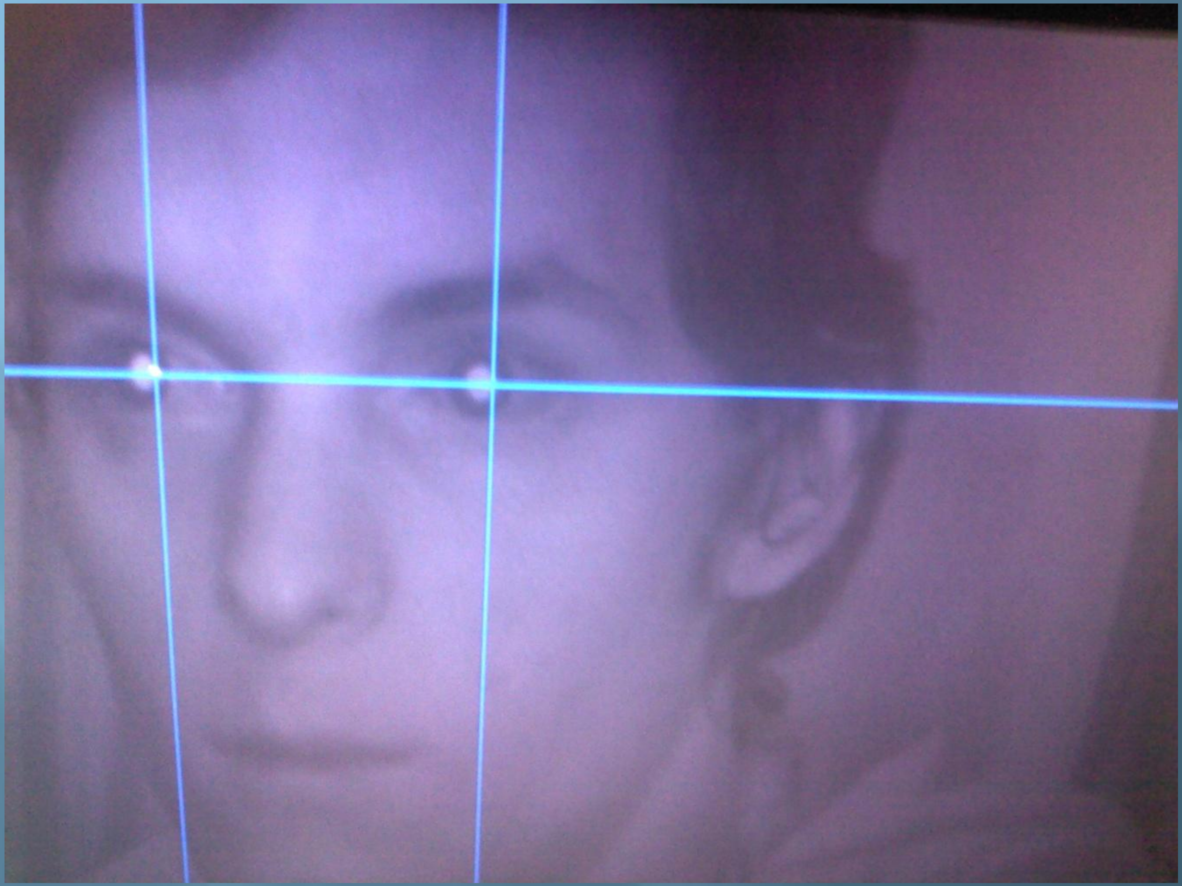


Implementation

- Step 10: Cross Hairs
  - Only if the detected IR light is within ±10 pixels of the average (horizontal and vertical) will its data be included in the average.
  - average[t] = average[t-1] + (data[t] – average[t-1]) / (sample_size++)



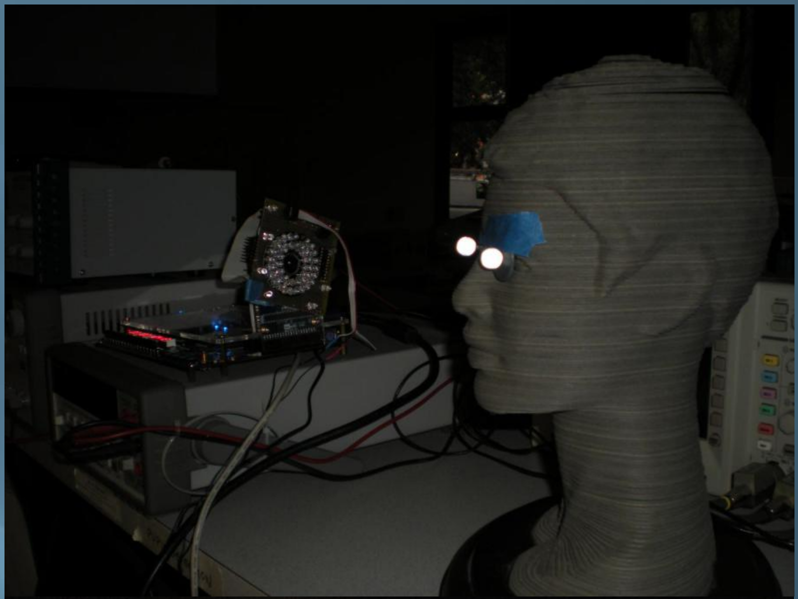
Implementation

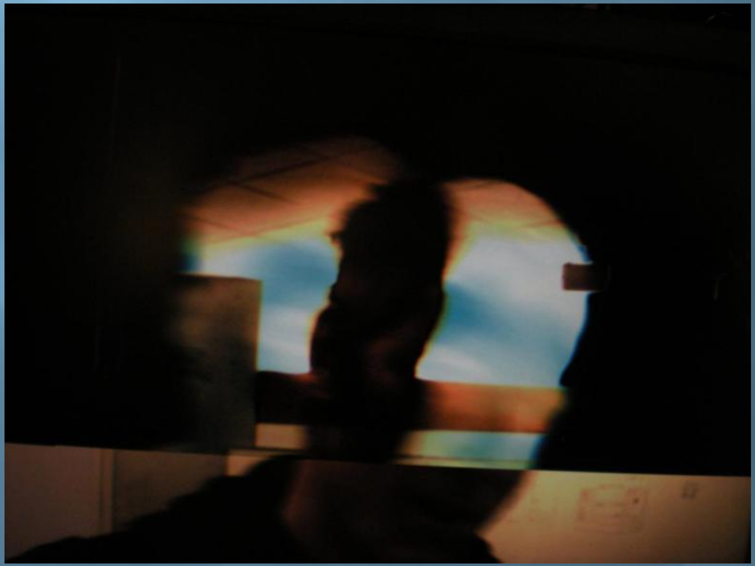


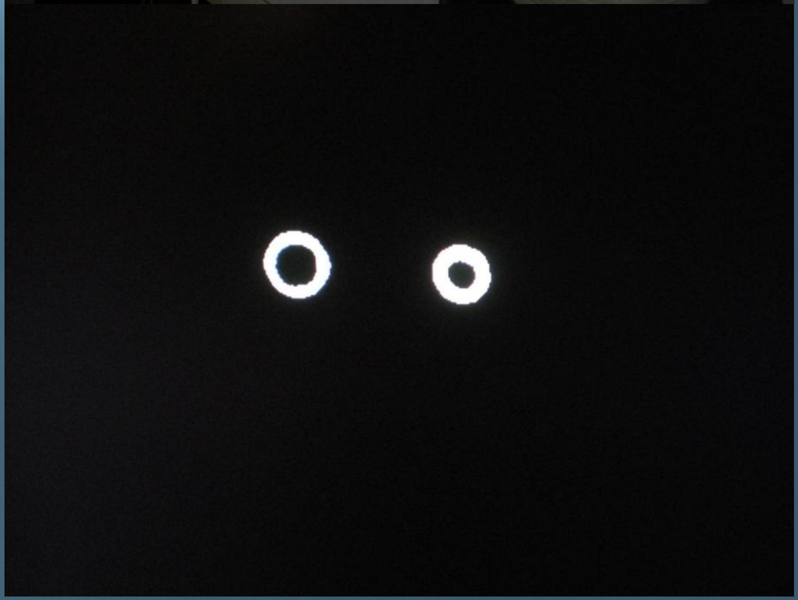
Step 11: That's ***<u>not</u>*** a Manikin

- Change the hardware
  - Remove the less-desirable LEDs.
- Change the software
  - Edit the cross hairs module.
  - PWM duty cycle = %100
  - Find the RGB thresh holds.

The End







```
module cross_hairs    (
                    sys_clk,
                    areset_n,
                    H_Cont,
                    V_Cont,
                    MRead_DATA1,
                    average_H1,
                    average_H2,
                    average_V1,
                    average_V2,
                    slave_WRITE
                 );

input sys_clk;
input areset_n;
input [12:0] H_Cont;
input [12:0] V_Cont;
input MRead_DATA1;
input slave_WRITE;

reg          [12:0]sample_size1;
output reg  [12:0]average_H1;
output reg  [12:0]average_V1;

reg        [12:0]sample_size2;
output reg  [12:0]average_H2;
output reg  [12:0]average_V2;

reg new_frame;
reg newnew_frame;
reg [5:0]biggest_diameter;
reg [5:0]temp_diameter;

always @(posedge sys_clk, negedge areset_n)
begin
   if (!areset_n)
   begin
      sample_size1<=0;
      average_H1<=0;
      average_V1<=0;
      sample_size2<=0;
      average_H2<=0;
      average_V2<=0;
      new_frame<=0;
      newnew_frame<=0;
      biggest_diameter<=0;
      temp_diameter<=0;
   end
   else
   begin
      if (H_Cont<=1 && V_Cont<=1)
      begin
         biggest_diameter <= 10;
         new_frame <= 1;
      end

      if (MRead_DATA1 && slave_WRITE)
      begin
         if (new_frame==1)
         begin
            sample_size1<=0;
            average_H1<=0;
            average_V1<=0;
            sample_size2<=0;
            average_H2<=0;
            average_V2<=0;
```

```
            new_frame<=0;
            newnew_frame<=1;
          end
          temp_diameter <= temp_diameter+1;
          if ((H_Cont<average_H1+(10) && H_Cont>average_H1-(10) && V_Cont<average_V1+(10) && V_Cont>average_V1-(10)) ||
(newnew_frame==1))
          begin
            sample_size1 <= sample_size1 + 1;
            average_H1   <= average_H1 + ((H_Cont)/(sample_size1+1)) - ((average_H1)/(sample_size1+1));
            average_V1   <= average_V1 + ((V_Cont)/(sample_size1+1)) - ((average_V1)/(sample_size1+1));
            newnew_frame <= 0;
          end
          else if ((H_Cont<average_H2+(10) && H_Cont>average_H2-(10)  && V_Cont<average_V2+(10) && V_Cont>average_V2-(10)) ||
(average_H2==0))
          begin
            sample_size2 <= sample_size2 + 1;
            average_H2   <= average_H2 + ((H_Cont)/(sample_size2+1)) - ((average_H2)/(sample_size2+1));
            average_V2   <= average_V2 + ((V_Cont)/(sample_size2+1)) - ((average_V2)/(sample_size2+1));
          end
        end
        else
        begin
          if (temp_diameter > biggest_diameter) biggest_diameter <= temp_diameter;
          temp_diameter <= 0;
        end
    end
end
endmodule
```

```
module frame_subtraction (
                sys_clk,
                areset_n,
                SRAM_DQ,
                Read_DATA1,
                Read_DATA2,
                WRead_DATA1,
                WRead_DATA2
            );

input sys_clk;
input areset_n;
input [15:0] SRAM_DQ;
input [14:0] Read_DATA1;
input [14:0] Read_DATA2;

output reg [14:0] WRead_DATA1;
output reg [14:0] WRead_DATA2;

//SRAM_DQ[15:11] = red[9:5];
//SRAM_DQ[10:6]  = blue[9:5];
//SRAM_DQ[5:0]   = green{[14:10],[14:14]};

//Read_DATA2[9:0] = red;
//Read_DATA1[9:0] = blue;
//{NRead_DATA1[14:10],NRead_DATA2[14:10]} = green;

always @(*)
begin
   if (SRAM_DQ[15:11]>Read_DATA2[9:5])
      WRead_DATA2[9:0]={SRAM_DQ[15:11],5'b0}-{Read_DATA2[9:5],5'b0};
   else
      WRead_DATA2[9:0]={Read_DATA2[9:5],5'b0}-{SRAM_DQ[15:11],5'b0};

   if (SRAM_DQ[10:6]>Read_DATA1[9:5])
      WRead_DATA1[9:0]={SRAM_DQ[10:6],5'b0}-{Read_DATA1[9:5],5'b0};
   else
      WRead_DATA1[9:0]={Read_DATA1[9:5],5'b0}-{SRAM_DQ[10:6],5'b0};

   if (SRAM_DQ[5:0]>{Read_DATA1[14:10],Read_DATA2[14]})
      {WRead_DATA1[14:10],WRead_DATA2[14:10]}={SRAM_DQ[5:0],4'b0}-{Read_DATA1[14:10],Read_DATA2[14],4'b0};
   else
      {WRead_DATA1[14:10],WRead_DATA2[14:10]}={Read_DATA1[14:10],Read_DATA2[14],4'b0}-{SRAM_DQ[5:0],4'b0};
end
endmodule
```

```
module gpio_1_19 (sys_clk, areset_n, GPIOO, Cont);

input sys_clk;
input areset_n;
output reg GPIOO;
output reg [31:0] Cont;

always @ (posedge sys_clk, negedge areset_n)
begin
   if (!areset_n) begin Cont<=0; GPIOO<=0;end
   else
   begin
      if(Cont!=32'h02faf080)
      begin
         Cont<=Cont+1;
         if (Cont<32'h0180_0000)
            GPIOO<=0;
         else
            GPIOO<=1;
      end
      else
      begin
         Cont<=0;
         GPIOO<=1;
      end
   end
end

endmodule
```

```
module USB_slave (sys_clk, areset_n, OTG_CS_N,
OTG_RD_N, OTG_WR_N, OTG_RST_N,
          USB_slave_DONE_read, USB_slave_DONE_write,
USB_slave_READ,
          USB_slave_WRITE);

input sys_clk;
input areset_n;

input USB_slave_READ;
input USB_slave_WRITE;

output reg OTG_CS_N;
output reg OTG_RD_N;
output reg OTG_WR_N;
output reg OTG_RST_N;
output reg USB_slave_DONE_read;
output reg USB_slave_DONE_write;

reg [7:0] counter;
reg [4:0] Sstate, Snext;

parameter    S0  = 5'b00000,
            S1  = 5'b00001,
            S2  = 5'b00010,
            S3  = 5'b00011,
            S4  = 5'b00100,
            S5  = 5'b00101,
            S6  = 5'b00110,
            S7  = 5'b00111,
            S8  = 5'b01000,
            S9  = 5'b01001,
            S10 = 5'b01010,
            S11 = 5'b01011,
            S12 = 5'b01100,
            S13 = 5'b01101,
            S14 = 5'b01110,
            S15 = 5'b01111,
            S16 = 5'b10000,
            S17 = 5'b10001,
            S18 = 5'b10010,
            S19 = 5'b10011,
            S20 = 5'b10100,
            S21 = 5'b10101,
            S23 = 5'b10110,
            S24 = 5'b10111;


always @(*)
begin
   case (Sstate)
      S0:   begin
            OTG_CS_N = 1;
            OTG_RD_N = 1;
            OTG_WR_N = 1;
            USB_slave_DONE_read = 1;
            USB_slave_DONE_write = 1;
            if (USB_slave_WRITE)
               Snext = S6;
            else if (USB_slave_READ)
               Snext = S1;
            else
               Snext = S0;
            end

// READ READ READ READ READ READ READ READ READ
READ READ READ
// READ READ READ READ READ READ READ READ READ
READ READ READ
// READ READ READ READ READ READ READ READ READ
READ READ READ

      S1:      begin
            OTG_CS_N = 0;
            OTG_RD_N = 1;
            OTG_WR_N = 1;
            USB_slave_DONE_read = 0;
            USB_slave_DONE_write = 0;
            Snext = S2;
            end

      S2:      begin
            OTG_CS_N = 0;
            OTG_RD_N = 0;
            OTG_WR_N = 1;
            USB_slave_DONE_read = 0;
            USB_slave_DONE_write = 0;
            Snext = S3;
            end

      S3:      begin
            OTG_CS_N = 0;
            OTG_RD_N = 0;
            OTG_WR_N = 1;
            USB_slave_DONE_read = 0;
            USB_slave_DONE_write = 0;
            Snext = S4;
            end

      S4:      begin
            OTG_CS_N = 0;
            OTG_RD_N = 1;
            OTG_WR_N = 1;
            USB_slave_DONE_read = 0;
            USB_slave_DONE_write = 0;
            Snext = S5;
            end

      S5:      begin
            OTG_CS_N = 1;
            OTG_RD_N = 1;
            OTG_WR_N = 1;
            USB_slave_DONE_read = 0;
            USB_slave_DONE_write = 0;
            if (counter == 8'b0000_1010)
               Snext = S0;
```

```
        else
           Snext = S5;
        end

// WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE
WRITE WRITE
// WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE
WRITE WRITE
// WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE
WRITE WRITE

    S6:     begin
        OTG_CS_N = 0;
        OTG_RD_N = 1;
        OTG_WR_N = 1;
        USB_slave_DONE_read = 0;
        USB_slave_DONE_write = 0;
        Snext = S7;
        end

    S7:     begin
        OTG_CS_N = 0;
        OTG_RD_N = 1;
        OTG_WR_N = 0;
        USB_slave_DONE_read = 0;
        USB_slave_DONE_write = 0;
        Snext = S8;
        end

    S8:     begin
        OTG_CS_N = 0;
        OTG_RD_N = 1;
        OTG_WR_N = 0;
        USB_slave_DONE_read = 0;
        USB_slave_DONE_write = 0;
        Snext = S9;
        end

    S9:     begin
        OTG_CS_N = 0;
        OTG_RD_N = 1;
        OTG_WR_N = 1;
        USB_slave_DONE_read = 0;
        USB_slave_DONE_write = 0;
        Snext = S10;
        end

    S10:     begin
        OTG_CS_N = 1;
        OTG_RD_N = 1;
        OTG_WR_N = 1;
        USB_slave_DONE_read = 0;
        USB_slave_DONE_write = 0;
        if (counter == 8'b0000_1010)
           Snext = S0;
        else
           Snext = S10;
        end

    default:    begin
            OTG_CS_N = 1;
            OTG_RD_N = 1;
            OTG_WR_N = 1;
            USB_slave_DONE_read = 0;
            USB_slave_DONE_write = 0;
            Snext = S0;
            end
    endcase
end

always @ (posedge sys_clk, negedge areset_n)
begin
  if (!areset_n) begin Sstate <= S0; end
  else
  begin
    Sstate <= Snext;
    if(Snext==S5 || Snext==S10) counter<=counter+1;
    else counter<=0;
  end
end

endmodule
```

```
module USB_master (sys_clk, areset_n,
USB_slave_DONE_read, USB_slave_DONE_write,
USB_slave_READ,
           USB_slave_WRITE, OTG_DATA, OTG_ADDR, SW);

// first step is writing the first 4BYtes of the 32Byte AV
header to memory addresses.

// the first 4Bytes == 1234_001f (just a sample) (not
actuallly the first 4 bytes of av header))

input sys_clk;
input areset_n;

input [17:0] SW;
input USB_slave_DONE_read;
input USB_slave_DONE_write;

output reg USB_slave_READ;
output reg USB_slave_WRITE;
output reg [15:0] OTG_DATA;
output reg [1:0]  OTG_ADDR;

reg   [32:0] counter;
reg    [2:0]  Mstate, Mnext;

parameter
         M1 = 3'b001,
         M2 = 3'b010,
         M0 = 3'b000;

always @(*)
begin
   OTG_DATA = 16'h1234;
   OTG_ADDR = 2'b10;
   case (Mstate)
   M0:    begin
            USB_slave_READ = 0;
            USB_slave_WRITE = 0;
            if (SW[10])
               Mnext = M1;
            else if (SW[9])
               Mnext = M2;
            else
               Mnext = M0;
         end

   M1:    begin
            USB_slave_READ = 1;
            USB_slave_WRITE = 0;
            if (SW[10])
               Mnext = M1;
            else if (SW[9])
               Mnext = M2;
            else
               Mnext = M0;
         end

   M2:    begin
            USB_slave_READ = 0;
            USB_slave_WRITE = 1;
            if (SW[10])
               Mnext = M1;
            else if (SW[9])
               Mnext = M2;
            else
               Mnext = M0;
         end

   default:
         begin
            USB_slave_READ = 0;
            USB_slave_WRITE = 0;
            Mnext = M0;
         end
   endcase
end

always @ (posedge sys_clk, negedge areset_n)
begin
   if (!areset_n) Mstate <= M0;
   else
   begin
      Mstate <= Mnext;
      //if (Mstate==M1 || Mstate==M2) OTG_ADDR <=
OTG_ADDR+1;
      //else OTG_ADDR <= 0;
   end
end

endmodule
```

```
module SRAM_master (sys_clk, areset_n, slave_READ, slave_WRITE, SRAM_ADDR, SW, pulse_time,
        slave_DONE_read, slave_DONE_write, KEY_2, KEY_1, H_Cont, V_Cont);

`include "VGA_Param.h"      // from VGA_Controller.v
wire    [12:0]      v_mask;   // from VGA_Controller.v
assign v_mask = 13'd0;      // from VGA_Controller.v

input [17:0] SW;
input sys_clk;
input areset_n;
input slave_DONE_read;
input slave_DONE_write;
input KEY_1;
input KEY_2;
input [12:0]H_Cont;
input [12:0]V_Cont;
input [32:0]pulse_time;

output reg slave_READ;
output reg slave_WRITE;
output reg [17:0] SRAM_ADDR;
   reg [32:0] PRE_SRAM_ADDR;
   reg [2:0] Mstate, Mnext;

parameter
      M1 = 3'b001,
      M2 = 3'b010,
      M0 = 3'b000;

always @(*)
begin
  case (Mstate)
    M0:     begin
        slave_READ = 0;
        slave_WRITE = 0;
        if (!KEY_1)
          Mnext = M2;
        else
          Mnext = M1;
        end

    M1:     begin
        slave_READ = 1;
        slave_WRITE = 0;
        if (SW[5] && slave_DONE_read)
          Mnext = M2;
        else if (pulse_time<=32'h02faf080 && pulse_time>=32'h002c0f080)
          Mnext = M2;
        else if (!KEY_2)
          Mnext = M1;
        else if (!KEY_1)
          Mnext = M2;
        else
          Mnext = M1;
        end

    M2:     begin
        slave_READ = 0;
        slave_WRITE = 1;
        if (pulse_time>32'h002c0f080 && pulse_time<32'h02faf080)
          Mnext = M2;
        else if (SW[5] && slave_DONE_write)
          Mnext = M1;
        else if (!KEY_1)
          Mnext = M2;
        else
```

```
                Mnext = M1;
            end

        default:    begin
                slave_READ = 0;
                slave_WRITE = 0;
                Mnext = M0;
                end
    endcase
end

always @ (posedge sys_clk, negedge areset_n)
begin
    if (!areset_n) SRAM_ADDR <= 0;
    else
        begin
        if (SW[5] && slave_DONE_write)
            begin
                if ((H_Cont == 13'b0) && (V_Cont == 13'b0))
                    SRAM_ADDR <= 0;
                else
                    SRAM_ADDR <= SRAM_ADDR + 1;
            end
        else if (slave_DONE_read || slave_DONE_write)
            if ((H_Cont == 13'b0) && (V_Cont == 13'b0))
                begin
                SRAM_ADDR <= 0;
                PRE_SRAM_ADDR <= 0;
                end
            else if (PRE_SRAM_ADDR > 32'h03ffff)
                begin
                PRE_SRAM_ADDR <= PRE_SRAM_ADDR + 1;
                SRAM_ADDR <= 0;
                end
            else if (    H_Cont>=X_START     && H_Cont<X_START+H_SYNC_ACT &&
                    V_Cont>=Y_START+v_mask     && V_Cont<Y_START+V_SYNC_ACT )
                begin
                PRE_SRAM_ADDR <= PRE_SRAM_ADDR + 1;
                SRAM_ADDR <= SRAM_ADDR + 1;
                end
        end
end

always @ (posedge sys_clk, negedge areset_n)

    if (!areset_n) Mstate <= M0;
    else Mstate <= Mnext;

endmodule
```

```
module SRAM_slave (sys_clk, areset_n, CE_N, OE_N, WE_N, data_drive,
            slave_DONE_read, slave_DONE_write, slave_READ,
            slave_WRITE);

input sys_clk;
input areset_n;

input slave_READ;
input slave_WRITE;

output reg CE_N;
output reg OE_N;
output reg WE_N;
output reg data_drive;
output reg slave_DONE_read;
output reg slave_DONE_write;

reg [7:0] wait_here;

reg [4:0] Sstate, Snext;

parameter    S0 = 5'b00000,
          S1 = 5'b00001,
          S2 = 5'b00010,
          S3 = 5'b00011,
          S4 = 5'b00100,
          S5 = 5'b00101,
          S6 = 5'b00110,
          S7 = 5'b00111,
          S8 = 5'b01000,
          S9 = 5'b10000;

always @(*)
begin
   case (Sstate)
      S0:   begin
            CE_N = 1;
            OE_N = 1;
            WE_N = 1;
            data_drive = 0;
            slave_DONE_read = 0;
            slave_DONE_write = 0;
            if (slave_WRITE)
               Snext = S6;
            else if (slave_READ)
               Snext = S1;
            else
               Snext = S0;
            end


// READ READ READ READ READ READ READ READ READ READ READ READ
// READ READ READ READ READ READ READ READ READ READ READ READ
// READ READ READ READ READ READ READ READ READ READ READ READ

      S1:      begin
            CE_N = 0;
            OE_N = 1;
            WE_N = 1;
            data_drive = 0;
            slave_DONE_read = 0;
            slave_DONE_write = 0;
            Snext = S2;
            end

      S2:      begin
```

```
            CE_N = 0;
            OE_N = 0;
            WE_N = 1;
            data_drive = 0;
            slave_DONE_read = 1;
            slave_DONE_write = 0;
            Snext = S3;
            end

        S3:     begin
            CE_N = 0;
            OE_N = 1;
            WE_N = 1;
            data_drive = 0;
            slave_DONE_read = 0;
            slave_DONE_write = 0;
            if (slave_WRITE)
              Snext = S6;
            else if (slave_READ)
              Snext = S1;
            else
              Snext = S0;
            end

    // WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE
    // WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE
    // WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE WRITE

        S6:     begin
            CE_N = 0;
            OE_N = 1;
            WE_N = 1;
            data_drive = 1;
            slave_DONE_read = 0;
            slave_DONE_write = 0;
            Snext = S7;
            end

        S7:     begin
            CE_N = 0;
            OE_N = 1;
            WE_N = 0;
            data_drive = 1;
            slave_DONE_read = 0;
            slave_DONE_write = 1;
            Snext = S8;
            end

        S8:     begin
            CE_N = 0;
            OE_N = 1;
            WE_N = 1;
            data_drive = 1;
            slave_DONE_read = 0;
            slave_DONE_write = 0;
            if (slave_WRITE)
              Snext = S6;
            else if (slave_READ)
              Snext = S1;
            else
              Snext = S0;
            end

        default:    begin
              CE_N = 1;
              OE_N = 1;
```

```
            WE_N = 1;
            data_drive = 0;
            slave_DONE_read = 0;
            slave_DONE_write = 0;
            Snext = S0;
            end
     endcase
end

always @ (posedge sys_clk, negedge areset_n)
begin
   if (!areset_n) begin Sstate <= S0; end
   else Sstate <= Snext;
end

endmodule
```