# Object Identification and Tracking on an FPGA using a camcorder and a Robotic Arm

Kooresh Akhbari (1003870180)
Pranshu Malik (1004138916)

## 1. Introduction

The goal of our project was to make a robot that detects a cup and puts a coin into that cup. This was a practical use for demonstrating object detection and tracking its location in space — all using an FPGA (DE1 SoC) that also coordinates with an Arduino.

We started with the intention to make a robot that is controlled by the FPGA. After consulting our Teaching assistant and iterating on our idea a few times, we finally decided to make a robot that has a camera mounted on it, rotates in a certain radius to find an object, in our case a cup, then drops an item into the cup. All of the video processing is done on the FPGA and we also used an Arduino with a motor controller to interface the motors, since they can't communicate with FPGAs directly. Our system has the flexibility to detect objects of variable sizes and different colors by allowing calibration of the corresponding thresholds. Another use of this robotic system would be for it to track a moving object in its field, while the structure itself can be adapted for other uses such one with an end-effector for drawing all kinds of sketches on a paper.

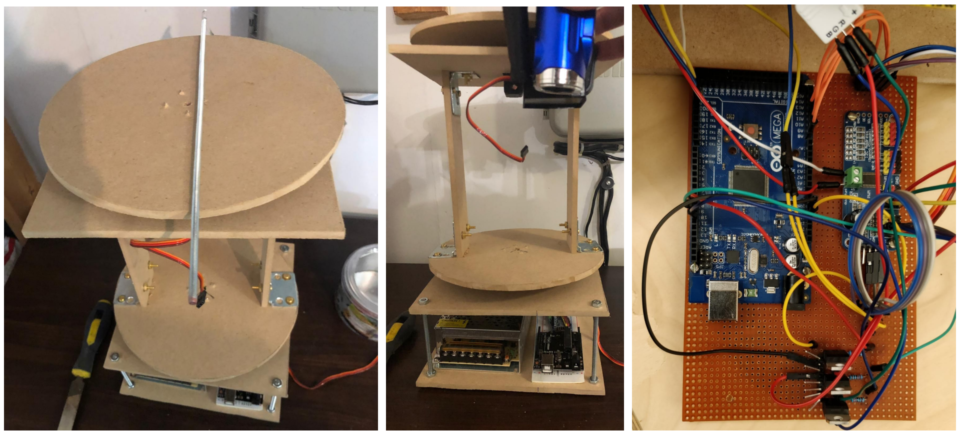
## 2. The Design

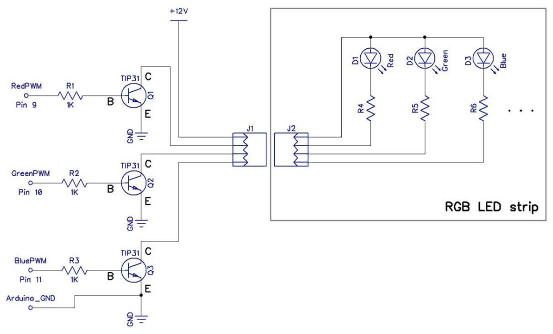
Electromechanical: The robot's frame was cut out of MDF (Medium Density Fiber). We first made a rough estimate of the size and the height of the robot, then we cut out the main parts of the robot, such as the circular plates and the two vertical stands. Once we had the basic parts of the robot ready, we cut out the top horizontal stick that was used to mount the cup and the camera. This was a critical step, because if it was too short then the camera would have the base of the robot in its video frame and if it was too long then it would exert an excess moment on the top motor's shaft.

All separate parts of the robot were then assembled together using angle brackets along with screws, nuts, and washers to ensure a sturdy connection to the mainframe of the robot.

Then it came to designing the circuitry for some parts of the robot. We took a protoboard and mounted the Arduino, the motor controller, soldered the transistors and the resistors that were used for the LED strips, and lastly, provided a common ground to all individual circuits. This vastly reduced the time that it took us to set up the robot in the future. After having all the electronics together we placed them all under the robot, to package the power supply, the circuit board, and all the wires.

In all, other than the FPGA, we had five other devices which were connected to the robot, a camera, three servo motors, an Arduino, a servo motor controller, LED strip, and a power supply. The camera provided video as input to the FPGA. We used two high torque servo motors, to control the rotation of our robot, and we used a smaller servo motor to control the container for the coin. We also installed an LED strip at the base of the robot to change colors based on the current state of the robot. This served as a hardware/physical debugging platform and an aesthetic addition to the robot. All the devices were powered by a 5v 12A power supply.





The circuit for LED state control was inspired from https://cdn.makezine.com/uploads/2013/07/image04.jpg

Software and controls:

*Top-level module:* The diagram below depicts the entire process of our system: how all our main modules connect, communicate, and work together. The camera's video-in goes into the video decoder module which decodes the video, converts it from YCbCr format to RGB and streams information, pixel by pixel to our color filter. The color filter binarized the pixels based on the R, G, and B thresholds that are set by the user, which can also be calibrated during runtime. The binarized pixels are the input for our pixel sequence detector module, which detects the presence of an object meeting the minimum dimensions, that can again be changed by the user during runtime. Once an object is found, the GPIO-Arduino interface is sent a signal from the detector module. This change is communicated to the Arduino in a specific way, which then controls the motors and LEDs in a particular routine. The Audio module also plays a specific audio file given the same state signals as the Arduino.

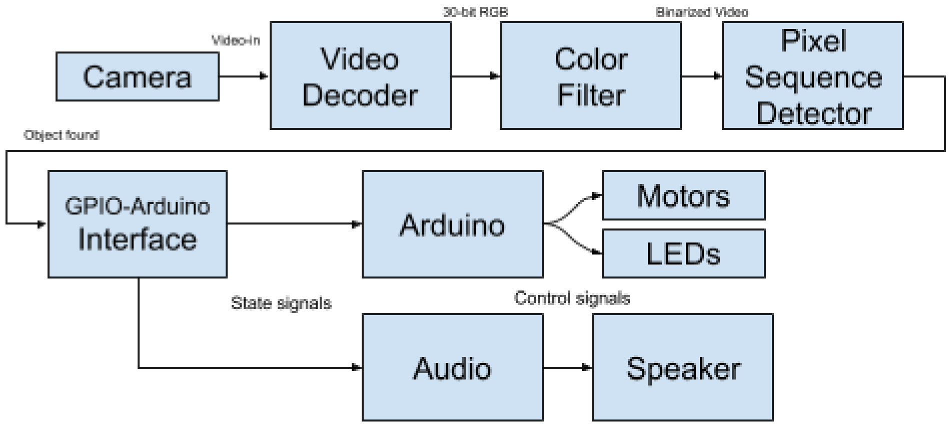


Diagram 1: Top-level overview of the project

*Calibrate module:* It lets the user change either the minimum dimension of the object (cup) or the color threshold. The particular threshold can be selected using the onboard switches and incremented or decremented using two keys.

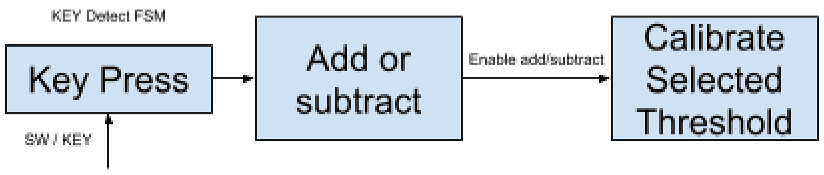


Diagram 2: Overview of the calibration module

*Color filter module:* It takes in the converted pixels in RGB (30 bits) format from the decoder and then binarizes it based on the threshold, which can be changed during runtime through the calibrate module, allowing the user to optimize the detector for that environment that the robot is in.

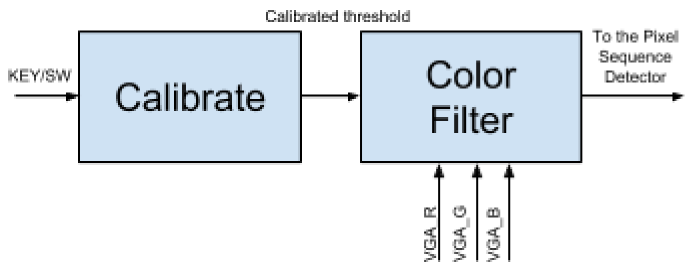


Diagram 3: Overview of the color filter module

*Pixel Sequence Detector:* For every frame, determined by vertical sync, this module goes through all the rows of pixels, one-by-one in real-time, and counts the maximum number of consecutive white pixels only in the central region of the frame. If that number exceeds or matches the threshold, then that means two things: first, that the object has been found and, second, that the centroid can be calculated. All the start and end positions for such lines of white pixels are recorded and the centroid is calculated by bisecting the line corresponding to the maximum count. The pixels reaching over the threshold are displayed as red, implying that the object has been found. The centroid as a single dot (pixel) displayed in the next frame once the corresponding pixel of that coordinate is read by the FPGA. This module also sends the `object_found` signal to the GPIO-Arduino interface.

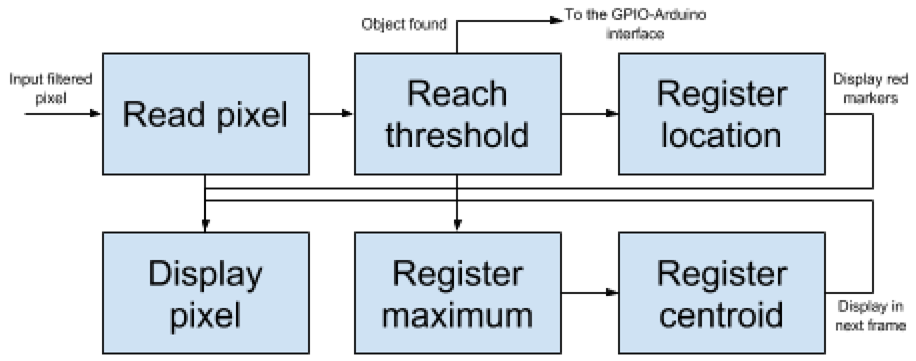


Diagram 4: Overview of the pixel sequence detector module

*GPIO-Arduino interface module:* This module communicates what the robot should do, to the Arduino. If the object is found while searching or if the reset key is pressed at any time during its operation, this module sends the corresponding changes to the Arduino through the GPIO pins. The output signal is changed in a very controlled manner where, first, it signals the Arduino to anticipate a change and then allows it enough time to register the change. This was done to make sure no "garbage" signals are interpreted by the robot that disrupts its routine. If there is no change in state, then the FPGA maintains its previous signal and the robot remains in its previous routine.

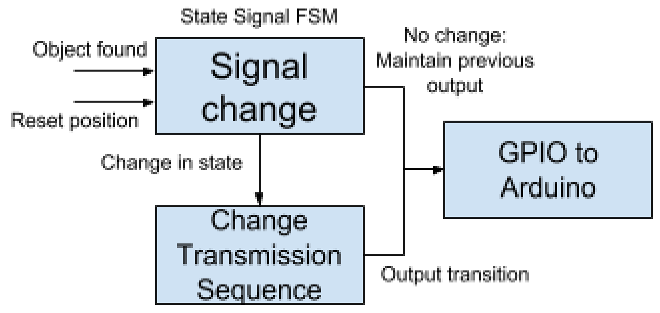


Diagram 5: Overview of the GPIO-Arduino interface module

*Arduino sketch:* This contains the runtime instructions for the robot, based on the input signals from the FPGA. They are routines set for searching, placing, resetting position, and stopping the robotic arm. It first checks for a valid signal, since the transmission medium can be lossy, unpredictable, or even damaged. Once, it detects a signal change request from the FPGA and records a valid input signal (routine), the routine is switched from the current one to the one requested by the FPGA. If there were no changes detected then it would follow its previous routine.

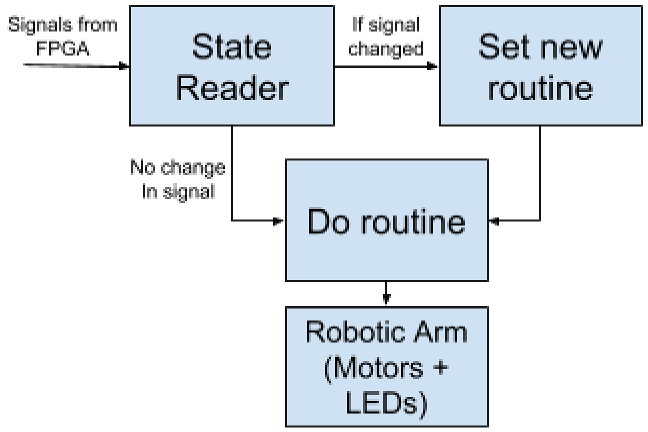


Diagram 6: Overview of the Arduino sketch (runtime instructions)

*Audio control module:* This module has a ram in it, which stores three different tunes in it. It takes in three enable signals from the GPIO-Arduino interface module which represent which state the robot is it, for example searching for the cup, dropping the coin, or if it has found the cup. Accordingly, the corresponding tune is accessed from the RAM which makes the FPGA play the sound clip that was specified for that state.

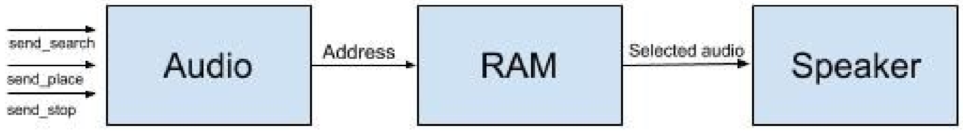


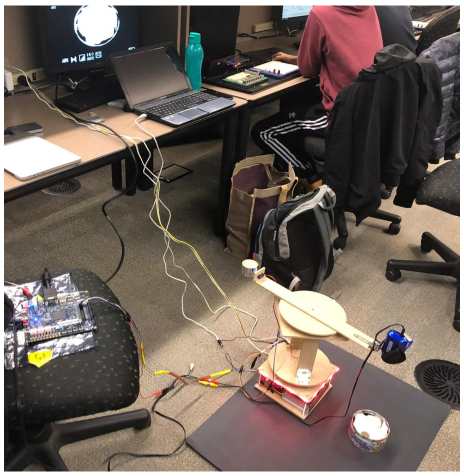
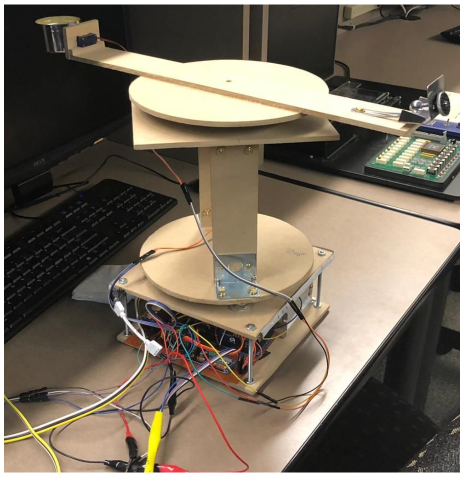
Diagram 7: Overview of the audio control module

## 3. Report on Success

We met all the milestones for our project to make the robot detect a cup object then drop an object into it, and in the final week of the project, we decided to add more features to our robot and smoothen all its operations.

We first added an LED strip whose color signified the different states that the robot was in. Then we added audio that was supposed to change according to the to the current state of the robot. It worked fine with switches or a single signal wire, but unfortunately, we could not figure out how to have it work autonomously based on multiple signals for the state of the robot. That was because the circuit had multiple clocks, potentially asynchronous (out of phase), for both audio and video which were not providing proper synchronization signals.

One fix we had for this was to have to two FPGAs communicating via GPIO pins, one handling the audio side and one handling the video side. We did not implement this, because we thought to have two FPGAs might have been against the rules of the project, therefore we just controlled them with switches on the final day.

Video explaining our project: YouTube

## 4. What we would do differently

Initially, when we designed and built the robot we did not have an efficient way to set it up and connect all the wires. Therefore it always took us a significant amount of time (~20 minutes) to connect everything together. As a result, we had to sacrifice a day to make an efficient system that would basically make the robot “plug and play”. We decided to get a protoboard and solder all the main wires onto it which helped us reduce our set-up time to ~5 minutes. Therefore if we were to do this project again we would design the robot to hold the wires in a neater way from the start and also have the PCB to manage all the wiring and centrally connect all the hardware.

We would have also spent more time and effort into experimenting with the color filter, to make it more sophisticated. At the current moment, the color filter only binarizes the pixels of the video, but it would be better to use a filtered input for the binarization itself so that noise can be reduced. Also, if time allowed, adding dynamic calibration would have definitely been a major focus of the project, so that we would not have to calibrate it ourselves every time. Also, as mentioned in the previous sections, we would have had two FPGAs talk to each other for controlling the audio and video separately and also try to add the motion tracking functionality in our system.

# Appendices: Verilog Code and Schematics

**Appendix A: Verilog Code**

Code-block 1: Pixel Detector Module

```
`timescale 1ns / 1ns // `timescale time_unit/time_precision

module pixel_sequence_detector (
                          input [29:0] input_pixel,
                          input        vsync,
                                       hsync,
                                       clock,
                          input [10:0] x_curr,
                                       y_curr,
                          input [8:6]SW,
                          input [2:0]  KEY,
                          output [10:0] x_start,
                                       y_start,
                                       x_end,
                                       y_end,
                                x_center,
                                       y_center,
                                x_out,
                                       y_out,
                          output [8:0] count,
                          output [7:0] counter_wire,
                          output [29:0] output_pixel,
                          output       object_found,
                          output [2:0] current_state_pixel);

wire                   ld_max_count,
                   ld_x_s_max,
                   ld_y_s_max,
                   ld_x_e_max,
                   ld_y_e_max,
                   ld_x_s,
                   ld_y_s,
                   ld_x_e,
                   ld_y_e,
                   ld_x_curr,
                   ld_y_curr,
                   ld_count;

control C1 ( .input_pixel (input_pixel),
                          .vsync (vsync),
                          .hsync (hsync),
                          .clock (clock),
                          .ld_max_count (ld_max_count),
                          .ld_x_s_max (ld_x_s_max),
                          .ld_y_s_max (ld_y_s_max),
                          .ld_x_e_max (ld_x_e_max),
                          .ld_y_e_max (ld_y_e_max),
                          .ld_x_s (ld_x_s),
                          .ld_y_s (ld_y_s),
                          .ld_x_e (ld_x_e),
                          .ld_y_e (ld_y_e),
                          .ld_x_curr (ld_x_curr),
                          .ld_y_curr (ld_y_curr),
                          .ld_count (ld_count));

datapath D1 (
                          .input_pixel (input_pixel),
                          .vsync (vsync),
                          .hsync (hsync),
                          .clock (clock),
                          .ld_max_count (ld_max_count),
                          .ld_x_s_max (ld_x_s_max),
                          .ld_y_s_max (ld_y_s_max),
                          .ld_x_e_max (ld_x_e_max),
                          .ld_y_e_max (ld_y_e_max),
                          .ld_x_s (ld_x_s),
                          .ld_y_s (ld_y_s),
                          .ld_x_e (ld_x_e),
                          .ld_y_e (ld_y_e),
                          .ld_count (ld_count),
                          .ld_x_curr (ld_x_curr),
                          .ld_y_curr (ld_y_curr),
                          .x_curr (x_curr),
                          .y_curr (y_curr),
                          .count (count),
                          .x_start (x_start),
                          .y_start (y_start),
                          .x_end (x_end),
                          .y_end (y_end),
                          .x_center (x_center),
                          .y_center (y_center),
                          .x_out (x_out),
                          .y_out (y_out),
                          .output_pixel (output_pixel),
                          .object_found (object_found),
                          .SW(SW),
                          .KEY(KEY),
                          .counter_threshold(counter_wire),
                          .current_state(current_state_pixel));

endmodule // pixel_sequence_detector

module control (
                          input [29:0] input_pixel,
                          input              vsync,
                                             hsync,
                                              clock,
                          output reg          ld_max_count,
                                              ld_x_s_max,
                                              ld_y_s_max,
                                              ld_x_e_max,
                                              ld_y_e_max,
                                              ld_x_s,
                                        ld_y_s,
ld_x_e,
                                              ld_y_e,
                                              ld_x_curr,
                                              ld_y_curr,
                                              ld_count);

                          reg [3:0] current_state, next_state;
                          localparam    READ_PIXEL= 3'd0,
                                                    REG_START = 3'd1,
                                                    COUNT_PIXEL = 3'd2,
                                                    NEW_PIXEL = 3'd3,
                                                    REG_END = 3'd4;

                          // state table
                          always @(*)
                          begin

                          case (current_state)
                                 //
                                 READ_PIXEL:next_state = (input_pixel > 0)?REG_START:READ_PIXEL;
                                 //
                                 REG_START:   next_state = COUNT_PIXEL;
                                 //
                                 COUNT_PIXEL:next_state = (input_pixel > 0) ? NEW_PIXEL : REG_END;
                                 //
                                 NEW_PIXEL:next_state = COUNT_PIXEL;
                                 //
                                 REG_END:           next_state = READ_PIXEL;
                                 //
                                 default: next_state = READ_PIXEL;
                          endcase

                          end // state_table

                          // output logic for datapath control signals
                          always @(*)
                          begin
                          // all signals are 0 by default, to avoid latches.

                                 ld_max_count = 1'b0;
                                 ld_x_s_max   = 1'b0;
                                 ld_y_s_max   = 1'b0;
                                 ld_x_e_max   = 1'b0;
                                 ld_y_e_max   = 1'b0;
                                 ld_x_s       = 1'b0;
                                 ld_y_s       = 1'b0;
                                 ld_x_e       = 1'b0;
                                 ld_y_e       = 1'b0;
                                 ld_x_curr    = 1'b0;
                                 ld_y_curr    = 1'b0;
                                 ld_count     = 1'b0;

                                 case (current_state)
                                        READ_PIXEL:  begin
                                                                         // nothing
                                                                  end

                                        REG_START:   begin
                                                                         ld_x_s       = 1'b1;
                                                                         ld_y_s       = 1'b1;
                                                                  end
                                        COUNT_PIXEL:begin
                                                                         ld_count     = 1'b1;
                                                                  end
                                        NEW_PIXEL:begin
                                                                         ld_x_curr = 1'b1;
                                                                         ld_y_curr = 1'b1;
                                                                  end

                                        REG_END:            begin
                                                                         ld_x_e       = 1'b1;
                                                                         ld_y_e       = 1'b1;
                                                                         Ld_max_count = 1'b1;
                                                                         ld_x_s_max   = 1'b1;
                                                                         ld_y_s_max   = 1'b1;
                                                                         ld_x_e_max   = 1'b1;
                                                                         ld_y_e_max   = 1'b1;
                                                                  end
                                 // no default needed; all of our outputs were assigned a value
                                 endcase

                          end // enable_signals

                          // current_state registers
                          always @(posedge clock)
                          begin
                                 if(vsync == 1'b1)
                                        current_state <= READ_PIXEL;
                                 else
                                        current_state <= next_state;
                          end // state_FFS

endmodule //control
```

```verilog
module datapath (
                          input         [29:0]input_pixel,
                          input                      vsync,
                                                     hsync,
                                                     clock,
                                                     ld_max_count,
                                                     ld_x_s_max,
                                                     ld_y_s_max,
                                                     ld_x_e_max,
                          ld_y_e_max,

                                                     ld_x_s,
                                                     ld_y_s,
                                                     ld_x_e,
                                                     ld_y_e,
                                              ld_x_curr,
                                                     ld_y_curr,
                                                     ld_count,
                          input         [10:0]x_curr,
                                                     y_curr,
                          input [8:6] SW,
                          input               [2:0] KEY,
                          output reg    [8:0] count,
                          output reg    [10:0] x_start,
                                                     y_start,
                                                     x_end,
                                                     y_end,
                                                     x_center,
                                                     y_center,
                                        x_out,
                                                     y_out,
                          output reg [29:0] output_pixel,
                          output reg                 object_found,
                          output reg[7:0] counter_threshold,
                          output[2:0] current_state
);

                          // input registers
                          reg [10:0]    x_s, x_e, x_s_max, x_e_max,
                                        y_s, y_e, y_s_max, y_e_max,
                                        x_curr_reg, y_curr_reg;
                          reg [8:0]     max_count, vsync_count;


                          localparam    //counter_threshold = 9'd25,
                                                     X_LIM_START         = 9'd200,
                                                     X_LIM_END           = 9'd300,
                                                     Y_LIM_START         = 9'd250,
                                                     Y_LIM_END           = 9'd300;

                          always @(posedge clock)
```

```
                            begin
                                  if ((counter_threshold <= 8'd5))
                                        begin
                                              counter_threshold <= 8'd30;
                                        end

                                  else if((counter_threshold >= 8'd100))
                                        begin
                                              counter_threshold <= 8'd30;
                                        end

                                  else if (enable_pixel_add)
                                        begin
                                        counter_threshold <= counter_threshold + 8'd5;
                                        end

                                  else if (enable_pixel_sub)
                                        begin
                                        counter_threshold <= counter_threshold - 8'd5;
                                        end

                                  else if (reset_pixel)
                                        begin
                                        counter_threshold <= 8'd30;
                                        end

                                  else
                                        counter_threshold = counter_threshold;
                            end

wire enable_pixel_add,
      enable_filter_add,
      enable_pixel_sub,
      enable_filter_sub,
      reset_pixel,
      reset_filter;

calibrate pixel ( .SW(SW),
                            .KEY(KEY),
                            .clock(clock),
                            .enable_pixel_add(enable_pixel_add),
                            .enable_filter_add(enable_filter_add),
                            .enable_pixel_sub(enable_pixel_sub),
                            .enable_filter_sub(enable_filter_sub),
                            .reset_pixel(reset_pixel),
                            .reset_filter(reset_filter),
                            .current_state(current_state));


                            always @(posedge clock)
                            begin
```

```verilog
if(vsync == 1'b1)
begin
             x_s          <= 11'd0;
             x_e          <= 11'd0;
             x_s_max      <= 11'd0;
             x_e_max      <= 11'd0;
             y_s          <= 11'd0;
             y_e          <= 11'd0;
             y_s_max      <= 11'd0;
             y_e_max      <= 11'd0;
             x_curr_reg <= 11'd0;
             y_curr_reg <= 11'd0;
             count  <= 9'd0;
             max_count<= 9'd0;
             vsync_count <= vsync_count + 1;
end
else if(hsync == 1'b1)
begin
             count <= 1'b0;
end
else
begin
             //
             if ((ld_x_s == 1'b1) & (ld_y_s == 1'b1))
             begin
                     x_s         <= x_curr;
                     y_s         <= y_curr;
                     x_curr_reg <= x_curr;
                     y_curr_reg <= y_curr;
                     count       <= 9'd0;
                     max_count <= 9'd0;
                     if (vsync_count > 500)
                     begin
                          object_found <= 1'b0;
                          vsync_count <= 0;
                     end
             end
             //
             if (ld_count == 1'b1)
             begin
                     if ((x_s > X_LIM_START) & (x_s < X_LIM_END) &
                           (y_s > Y_LIM_START) & (y_s < Y_LIM_END))
                          count <= count + 1;
             end
             //
             if ((ld_x_curr == 1'b1) & (ld_y_curr == 1'b1))
             begin
                    x_curr_reg <= x_curr;
                    y_curr_reg <= y_curr;
             end
             //
```

```
    if (    (ld_max_count     == 1'b1) &
                (ld_x_s_max   == 1'b1) &
                (ld_y_s_max   == 1'b1) &
                (ld_x_e_max   == 1'b1) &
                (ld_y_e_max   == 1'b1) &
                (ld_x_e       == 1'b1) &
                (ld_y_e       == 1'b1)
        )
    begin
        if (count >= max_count)
        begin
            max_count<= count;
            x_s_max <= x_s;
            y_s_max <= y_s;
            x_e <= x_curr;
            y_e <= y_curr;
            x_e_max <= x_e;
            y_e_max <= y_e;
        end
        if(max_count > threshold)
        begin
            x_center <= (x_s_max + ((x_e_max+x_s_max)/2));
            y_center <= y_e_max;
            object_found <= 1'b1;
            output_pixel <= {10'b1111111100, 10'd0,10'd0};
        end
    end
end
if (object_found == 1'b1)
begin
    x_out <= x_center;
    y_out <= y_center;
    output_pixel <= {10'b1111111100, 10'd0, 10'd0};
    // drawn COM
    object_found <= 1'b0;
end
else
begin
    x_out <= x_curr;
    y_out <= y_curr;
    output_pixel <= input_pixel;
end
end // for the reset else's

if ((count > counter_threshold) & (input_pixel != 0))
begin
    output_pixel <= {10'b1111111100, 10'd0, 10'd0};
    object_found <= 1'b1;
end
else
begin
```

```
                                                output_pixel <= input_pixel;
                                        end
                                        x_out <= x_curr;
                                        y_out <= y_curr;
                                end

endmodule // datapath
```

Code-block 2: Color filter

```
module Color_Filter (
input clk,
input [9:0] oVGA_Red,
input [9:0] oVGA_Green,
input [9:0] oVGA_Blue,
input [8:6] SW,
input [2:0] KEY,
output reg [29:0] filtered_color,
output reg [9:0] counter_threshold,
output [2:0] current_state);

wire    enable_pixel_add,
        enable_filter_add,
        enable_pixel_sub,
        enable_filter_sub,
        reset_pixel,
        reset_filter;

calibrate filter                (.SW(SW),
                                .KEY(KEY),
                                .clock(clk),
                                .enable_pixel_add(enable_pixel_add),
                                .enable_filter_add(enable_filter_add),
                                .enable_pixel_sub(enable_pixel_sub),
                                .enable_filter_sub(enable_filter_sub),
                                .reset_pixel(reset_pixel),
                                .reset_filter(reset_filter),
                                .current_state(current_state));

                                always @(posedge clk)
                                begin
                                        if ((counter_threshold <= 10'd0))
                                        begin
                                                        counter_threshold <= 10'd600;
                                        end

                                        else if((counter_threshold >= 10'd1020))
                                        begin
                                                        counter_threshold <= 10'd600;
                                        end

                                        else if (enable_filter_add)
                                        begin
                                              counter_threshold <= counter_threshold + 10'd10;
                                        end

                                        else if (enable_filter_sub)
                                        begin
                                              counter_threshold <= counter_threshold - 10'd10;
                                        end

                                        else if (reset_filter)
                                        begin
                                              counter_threshold <= 10'd600;
                                        end

                                        else
                                              counter_threshold <= counter_threshold;
                                end

always @ (posedge clk)
begin
        if ((oVGA_Red < counter_threshold) & (oVGA_Green < counter_threshold) & (oVGA_Blue <
counter_threshold))
                  filtered_color <= 30'd0; // black
        else filtered_color <= 30'b111111110011111111001111111100; // white
end

endmodule // Color_Filter
```

Code-block 3:Calibrate module

```
`timescale 1ns / 1ns // `timescale time_unit/time_precision

module calibrate (input [8:6] SW,
                                       input [2:0]   KEY,
                                       input         clock,
                                       outputenable_pixel_add,
                                                     enable_filter_add,
                                                     enable_pixel_sub,
                                                     enable_filter_sub,
                                                     reset_pixel,
                                                     reset_filter,
                                       output [2:0]  current_state);

control_calibrationC1 (.SW(SW), .KEY(KEY), .enable_pixel_add(enable_pixel_add),
                      .enable_filter_add(enable_filter_add), .enable_pixel_sub(enable_pixel_sub),
                      .enable_filter_sub(enable_filter_sub), .reset_pixel(reset_pixel),
              .reset_filter(reset_filter), .clock(clock), .current_state(current_state));

endmodule // calibrate


module control_calibration (input [8:6] SW,
                           input [2:0]KEY,
                           inputclock,
                           output reg   enable_pixel_add,
                                        enable_filter_add,
                                               enable_pixel_sub,
                                               enable_filter_sub,
                                               reset_pixel,
                                               reset_filter,
                          output reg [2:0] current_state);

                          reg [2:0] next_state;

                          localparam                KEY_WAIT           = 3'd0,
                                                    CALIBRATE          = 3'd1,
                                                    CALIBRATE_WAIT     = 3'd2,
                                                    RESET              = 3'd3,
                                                    RESET_WAIT         = 3'd4;

                          // state table
                          always @(*)
                          begin
                                 //
                                 case (current_state)
                                        //
                                        KEY_WAIT: if ((KEY[2] ^ KEY[1]) & (SW[6] == 0))
                                               begin
                                                      next_state = CALIBRATE;
                                                end

                                                      else if (SW[6] == 1)
                                               begin
                                                             next_state = RESET;
                                                      end

                                                      else
                                                      begin
                                                      next_state = KEY_WAIT;
                                               end
                                        //
                                        CALIBRATE:next_state = CALIBRATE_WAIT;
                                        //
                                        CALIBRATE_WAIT: next_state = ((KEY[2] == 1) & (KEY[1] ==
1)) ? KEY_WAIT : CALIBRATE_WAIT;
                                        //
                                        RESET:        next_state = RESET_WAIT;
```

```
            //
            RESET_WAIT: next_state = (SW[6]==0) ? KEY_WAIT: RESET_WAIT;
            //
            default: next_state = KEY_WAIT;
        endcase
        //

    end // state_table

    // output logic for datapath control signals
    always @(*)
    begin
    // all signals are 0 by default, to avoid latches.

        enable_pixel_add    = 1'd0;
        enable_filter_add   = 1'd0;
        enable_pixel_sub    = 1'd0;
        enable_filter_sub = 1'd0;
        reset_pixel         = 1'd0;
        reset_filter        = 1'd0;

        case (current_state)
            //
            KEY_WAIT:           ; // nothing
            //
            CALIBRATE: if ((SW[8] == 1) & (SW[7] == 0) & (KEY[2] == 0))
                            enable_filter_add = 1'd1;

                    else if ((SW[8] == 1) & (SW[7] == 0) & (KEY[1] == 0))
                            enable_filter_sub = 1'd1;

                    else if ((SW[8] == 0) & (SW[7] == 1) & (KEY[2] == 0))
                            enable_pixel_add = 1'd1;

                    else if ((SW[8] == 0) & (SW[7] == 1) & (KEY[1] == 0))
                            enable_pixel_sub = 1'd1;
            //
            CALIBRATE_WAIT: ; // nothing
            //
            RESET:                  // global reset happens if sw 7 is up
                                    // else only the selected is reset
                                    if ((SW[8] == 1) & (SW[7] == 0))
                                        reset_filter = 1'd1;

                                    else if ((SW[8] == 0) & (SW[7] == 1))
                                        reset_pixel = 1'd1;

                                    else if ((SW[8] == 0) & (SW[7] == 0))
                                    begin
                                        reset_pixel = 1'd1;
                                        reset_filter = 1'd1;
```

```
                                                        end
                                    //
                                    RESET_WAIT:         ; // nothing

                                    // no default needed; all of our outputs were assigned a
value
                              endcase

                        end // enable_signals

                        // current_state registers
                        always @(posedge clock)
                        begin
                              if(~KEY[0] == 1)
                                    current_state <= KEY_WAIT;
                              else
                                    current_state <= next_state;
                        end // state_FFS

endmodule // control_calibration
```

Code-block 4: GPIO-Arduino interface

```
`timescale 1ns / 1ns // `timescale time_unit/time_precision

module GPIO_Arduino (input      reset,
                                    object_found,
                                    clock_50,
                                    SW,
                  output [4:0]      sig_out,
                  output            counted,
                              implement,
                                    start_ctr);


control_ard C0 (.reset(reset),
                               .object_found(object_found),
                               .counted(counted),
                               .clock(clock_50),
                               .send_search(send_search),
                               .send_place(send_place),
                               .send_stop(send_stop),
                               .send_reset_position(send_reset_position),
                               .start_ctr(start_ctr),
                               .SW(SW)
                               );


datapath_ard D0 (.send_search(send_search),
                              .send_place(send_place),
                                .send_stop(send_stop),
                                .send_reset_position(send_reset_position),
                                .implement(implement),
                                .clock(clock_50),
                                .signal_out(sig_out));

counter CTR1 (.clock(clock_50),
                           .start_ctr(start_ctr),
                           .counted(counted),
                           .implement(implement));

endmodule


module control_ard  (input              reset,
                                        object_found,
                                        counted,
                                        clock,
                                        SW,
                    output reg          send_search,
                                        send_place,
                                        send_stop,
                                        send_reset_position,
                                        start_ctr);

                          reg [3:0] current_state, next_state;

                          // state_FF assignments
                          localparam    START_ROUTINE       = 4'd0,
                                        SET_RESET           = 4'd1,
                                        RESET               = 4'd2,
                                        SET_SEARCH          = 4'd3,
                                        SEARCH              = 4'd4,
                                        SEARCH_META         = 4'd5,
                                        SET_PLACE           = 4'd6,
                                        PLACE               = 4'd7,
                                        SET_STOP            = 4'd8,
                                        STOP                = 4'd9;

                          // state table
                          always @(*)
                          begin
                          //
                                case (current_state)
                                        //
                                        START_ROUTINE:            begin
                                                                  next_state = SET_RESET;
                                                            end

                                        SET_RESET:          begin
                                                                  if ((!counted) & (!reset))
```

```
                                 next_state = SET_RESET;
                          else if (reset)
                                 next_state =START_ROUTINE;
                          else
                                 next_state = RESET;
                    end

RESET:              begin
                    next_state=(SW==0)? RESET : SET_SEARCH;
                    end

SET_SEARCH:         begin
                          if ((!counted) & (!reset))
                                 next_state = SET_SEARCH;
                          else if (reset)
                                 next_state =START_ROUTINE;
                          else
                                 next_state = SEARCH;
                    end

SEARCH:             begin
                          if ((!object_found) & (!reset))
                                 next_state = SEARCH;
                          else if (reset)
                                 next_state =START_ROUTINE;
                          else
                                 next_state = SEARCH_META;
                    end

SEARCH_META:        begin
                          next_state = SET_PLACE;
                    end

SET_PLACE:          begin
                          if ((!counted) & (!reset))
                                 next_state = SET_PLACE;
                          else if (reset)
                                 next_state =START_ROUTINE;
                          else
                                 next_state = PLACE;
                    end

PLACE:        begin
                          next_state = SET_STOP;
                    end

SET_STOP:           begin
                          if ((!counted) & (!reset))
                                 next_state = STOP;
                          else if (reset)
                                 next_state =START_ROUTINE;
```

```
                                                        else
                                                                next_state = STOP;
                                                end

                        STOP:                   begin
                                                        if (reset)
                                                                next_state =START_ROUTINE;
                                                        else
                                                                next_state = STOP;
                                                end

                        default:                next_state = START_ROUTINE;
                        //
                endcase
        end

        // output logic
        always @(*)
        begin

                send_search                             = 1'b0;
                send_place                              = 1'b0;
                send_stop                               = 1'b0;
                send_reset_position                     = 1'b0;
                start_ctr                               = 1'b0;


                case (current_state)
                        //
                        START_ROUTINE:          begin
                                                        send_reset_position = 1;
                                                        start_ctr = 1;
                                                end

                        SET_RESET:              begin
                                                        send_reset_position = 1;
                                                end

RESET:                  begin
                                                        send_search = 1;
                                                        start_ctr = 1;
                                                end

                        SET_SEARCH:             begin
                                                        send_search = 1;
                                                end

                        SEARCH:                 begin
                                                        send_search = 1;
                                                end
```

```
                                        SEARCH_META:        begin
                                                                  send_place = 1;
                                                                  start_ctr = 1;
                                                            end

                                        SET_PLACE:          begin
                                                                  send_place = 1;

                                                            end

                                        PLACE:       begin
                                                                  send_stop = 1;
                                                                  start_ctr = 1;

                                                            end

                                        SET_STOP:           begin
                                                                  send_stop = 1;

                                                            end

                                        STOP:               begin
                                                                  send_reset_position = 1;
                                                            end
                                        // no defaults
                                 endcase

                          end // output logic

                          // current_state registers
                          always @(posedge clock)
                          begin
                                 if(reset)
                                        current_state <= START_ROUTINE;
                                 else
                                        current_state <= next_state;
                          end // state_FFs transition

endmodule // control

/* datapath module */
module datapath_ard       (input              send_search,
                                              send_place,
                                              send_stop,
                                              send_reset_position,
                                              implement,
                                              clock,
                     output reg [4:0] signal_out);

                          //
                          localparam    SEARCH                    = 4'b1000,
                        PLACE                       = 4'b0100,
                        RESET_POSITION              = 4'b0010,
                        STOP                        = 4'b0001;

                //
                always @(posedge clock)
                begin
                        if(send_reset_position)
                        begin
                                signal_out [4:1] <= RESET_POSITION;
                        end

                        else if (send_search)
                        begin
                                signal_out [4:1] <= SEARCH;
                        end

                        else if (send_place)
                        begin
                                signal_out [4:1] <= PLACE;
                        end

                        else if(send_stop)
                        begin
                                signal_out [4:1] <= STOP;
                        end

                        signal_out[0] <= implement;
                end

endmodule // datapath


module counter (input clock, start_ctr, output reg counted, implement);

                reg [26:0] cycleCount;

                always @(posedge clock) // triggered on edges of clock
                begin

                        if (start_ctr == 1'b1) // synchronous active-high
                        begin
                                cycleCount <= 27'd50000000;
                                counted <= 1'b0;
                                implement <= 1'b0;
                        end
                        else if (cycleCount > 27'd25000000)
                        begin
                                cycleCount <= cycleCount - 1'b1; // decrement state
                                counted <= 1'b0;
                                implement <= 1'b0;
                        end
                        else if (cycleCount == 0)
                        begin
                                counted <= 1'b1;
                                implement <= 1'b1;
                        end

                        if ((cycleCount <= 27'd25000000) & (cycleCount > 27'd0))
                        begin
                                implement <= 1'b1;
                                cycleCount <= cycleCount - 1'b1; // decrement state
                                counted <= 1'b0;
                        end
                end
endmodule // counter
```

Code-block 5: Arduino code

```
/*
 * Arm Routines v_2.0
 * Last edit: 23 November 2018
 * - Added support for LED strips to show the current state
*/

/*
 * Error Log:
 * For INVALID SIGNAL INPUT, ROUTINE IS SET TO PLACE FOR SOME REASON
 */

#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

// input signal threshold
#define IN_THRESH 600

// routine constants
#define SEARCH 1
#define PLACE 2
#define RESET_POSITION 3
#define STOP 4

// servo constants
#define MIN_PULSE_WIDTH 650
#define MAX_PULSE_WIDTH 2350
#define DEFAULT_PULSE_WIDTH 1500
#define FREQUENCY 50

// motor pin# on PCA9685 servo controller
#define BASE_MOTOR 0
```

```
#define TOP_MOTOR 1
#define CUP_MOTOR 4

// motor min and max angles
#define BASE_MIN 10
#define BASE_MAX 170
#define TOP_MIN 0
#define TOP_MAX 172
#define CUP_MIN 180
#define CUP_MAX 0

// delays to achieve desired rotation speeds
#define BASE_DELAY 50
#define CUP_DELAY 3
#define TOP_DELAY 14

// led strip output pins
#define RED_PIN 8
#define GREEN_PIN 9
#define BLUE_PIN 10

// intitialize runtime global variables:

// signal input from FPGA's GPIO
int signal_in[4] = {0, 0, 1, 0}; // start with reset
// starting arm state.
int arm_state[3] = {BASE_MIN, TOP_MIN, CUP_MIN}; // {BASE, TOP, CUP}
// arm routine
int arm_routine = RESET_POSITION; // start with reset
// signal change flag
bool signal_changed = false; // no change from start

// setup servo motor object
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

/* run at first, during setup */
void setup() {
Serial.begin(9600);
pwm.begin();
pwm.setPWMFreq (FREQUENCY);
pinMode (10, INPUT);
}

/* moves motor to the specified angle */
void moveMotor (int angle, int motor_pin) {
int pulse_wide, pulse_width;

// convert to pulse width
pulse_wide = map (angle, 0, 180, MIN_PULSE_WIDTH, MAX_PULSE_WIDTH);
pulse_width = int (float (pulse_wide)/1000000 * FREQUENCY * 4096);
```

```
// control Motor
pwm.setPWM (motor_pin, 0, pulse_width);
}

/* detects change in input signal */
void signal_in_change () {
// FPGA's GPIO set A8 = 0 when changing
// make sure change is of a valid type, eliminate stray spikes
if ((analogRead(A8)/IN_THRESH == 0) && (analogRead(A9)/IN_THRESH + analogRead(A10)/
 IN_THRESH + analogRead(A11)/IN_THRESH + analogRead(A12)/IN_THRESH) == 1) {

// see if there is a change detected
if (signal_in[0] != (analogRead (A9))/IN_THRESH ||
signal_in[1] != (analogRead (A10))/IN_THRESH ||
signal_in[2] != (analogRead (A11))/IN_THRESH ||
signal_in[3] != (analogRead (A12))/IN_THRESH) {

// set new values
signal_in[0] = (analogRead (A9))/IN_THRESH;
signal_in[1] = (analogRead (A10))/IN_THRESH;
signal_in[2] = (analogRead (A11))/IN_THRESH;
signal_in[3] = (analogRead (A12))/IN_THRESH;

Serial.println("NEW SIGNAL REGISTERED");
signal_changed = true; // input signal changed
}
}
// signal change flag is raised with invalid signal
else if (analogRead(A8)/IN_THRESH == 0) {
Serial.println("INVALID ROUTINE REQUESTED");
signal_changed = true;
}
// signal change flag is not raised
else if (analogRead(A8)/IN_THRESH == 1) {
Serial.println("NO CHANGE IN INPUT SIGNAL DETECTED");
signal_changed = false;
}
else { // error
Serial.println("INVALID SIGNAL CHANGE FLAG STATE");
signal_changed = false;
}
}

/* interprets routine from input signal combination */
void set_routine () {

/*
 * The routines are as follows:
 * signal_in 4:1 = 1 0 0 0 --> SEARCH (1)
 * signal_in 4:1 = 0 1 0 0 --> PLACE (2)
 * signal_in 4:1 = 0 0 1 0 --> RESET_POSITION (3)
 * signal_in 4:1 = 0 0 0 1 --> STOP (4)
 * default: STOP (4)
*/

if ((signal_in[3] == 1) && (signal_in[2] == 0) &&
 (signal_in[1] == 0) && (signal_in[0] == 0)) {
// set routine to SEARCH
Serial.println("ROUTINE SET TO SEARCH");
arm_routine = SEARCH;
}

else if ((signal_in[3] == 0) && (signal_in[2] == 1)
&& (signal_in[1] == 0) && (signal_in[0] == 0)) {
// set routine to PLACE
Serial.println("ROUTINE SET TO PLACE");
arm_routine = PLACE;
}

else if ((signal_in[3] == 0) && (signal_in[2] == 0)
&& (signal_in[1] == 1) && (signal_in[0] == 0)) {
// set routine to RESET_POSITION
Serial.println("ROUTINE SET TO RESET_POSITION");
arm_routine = RESET_POSITION;
}

else if ((signal_in[3] == 0) && (signal_in[2] == 0)
&& (signal_in[1] == 0) && (signal_in[0] == 1)) {
// set routine to STOP
Serial.println("ROUTINE SET TO STOP");
arm_routine = STOP;
}
else { // default case is STOP
arm_routine = STOP;
Serial.println ("ARM STOPPED: DEFAULT STATE");
}
}

/* arm routines */
void do_routine () {
// implement corresponding actions to the routine
switch (arm_routine) {
case SEARCH:
Serial.println("SEARCH ROUTINE INITIATED");

// yellow while searching
analogWrite(RED_PIN, 15);
analogWrite(GREEN_PIN, 15);
analogWrite(BLUE_PIN, 0);

// rotate base from current position to BASE_MAX degrees
for (int angle = arm_state[0]; (angle < BASE_MAX) &&
(analogRead(A8)/600 == 1); angle++) {
moveMotor ((arm_state[0]), BASE_MOTOR);
arm_state[0]++; // update arm_state[0] for base motor

// yellow while searching
analogWrite(RED_PIN, 15);
analogWrite(GREEN_PIN, 15);
analogWrite(BLUE_PIN, 0);

delay(BASE_DELAY);
}
// rotate base from BASE_MAX to BASE_MIN degrees
for (int angle = arm_state[0]; (angle > BASE_MIN)
&& (analogRead(A8)/600 == 1); angle--) {
moveMotor ((arm_state[0]), BASE_MOTOR);
arm_state[0]--; // update arm_state[0] for base motor

// yellow while searching
analogWrite(RED_PIN, 15);
analogWrite(GREEN_PIN, 15);
analogWrite(BLUE_PIN, 0);

delay(BASE_DELAY);
}
signal_in_change();
break;

case PLACE:
Serial.println("PLACE ROUTINE INITIATED");

// rotate top from 0 to TOP_MAX degrees
for (int angle = arm_state[1]; angle < TOP_MAX; angle++) {
moveMotor ((arm_state[1]), TOP_MOTOR);
arm_state[1]++; // update arm_state[1] for top motor

// green while placing
analogWrite(RED_PIN, 0);
analogWrite(GREEN_PIN, 255);
analogWrite(BLUE_PIN, 0);

delay(TOP_DELAY);
}

// rotate cup from CUP_MIN to CUP_MAX degrees
for (int angle = arm_state[2]; angle > CUP_MAX; angle--) {
moveMotor ((arm_state[2]), CUP_MOTOR);
arm_state[2]--; // update arm_state[2] for cup motor

// green while placing
analogWrite(RED_PIN, 0);
analogWrite(GREEN_PIN, 255);
analogWrite(BLUE_PIN, 0);

delay(CUP_DELAY);
}

// rotate cup from CUP_MAX to CUP_MIN degrees
for (int angle = arm_state[2]; angle < CUP_MIN; angle++) {
moveMotor ((arm_state[2]), CUP_MOTOR);
arm_state[2]++; // update arm_state[2] for cup motor

// blue after placing
analogWrite(RED_PIN, 0);
analogWrite(GREEN_PIN, 0);
analogWrite(BLUE_PIN, 255);

delay(CUP_DELAY);
}

// rotate top from TOP_MAX to TOP_MIN degrees
for (int angle = arm_state[1]; angle > TOP_MIN; angle--) {
moveMotor ((arm_state[1]), TOP_MOTOR);
arm_state[1]--; // update arm_state[1] for top motor

// blue after placing
analogWrite(RED_PIN, 0);
analogWrite(GREEN_PIN, 0);
analogWrite(BLUE_PIN, 255);

delay(TOP_DELAY);
}

// blue after placing
analogWrite(RED_PIN, 0);
analogWrite(GREEN_PIN, 0);
analogWrite(BLUE_PIN, 255);

// force stop after place
signal_in[2] = 0;
signal_in[0] = 1;
arm_routine = STOP;
break;

case RESET_POSITION:
Serial.println("RESET POSITION ROUTINE INITIATED");

// set all motors to starting state

moveMotor (BASE_MIN, BASE_MOTOR);
// white while reset
analogWrite(RED_PIN, 15);
analogWrite(GREEN_PIN, 15);
analogWrite(BLUE_PIN, 15);
delay(500);

moveMotor (TOP_MIN, TOP_MOTOR);
// white while reset
analogWrite(RED_PIN, 15);
analogWrite(GREEN_PIN, 15);
analogWrite(BLUE_PIN, 15);
delay(500);

moveMotor (CUP_MIN, CUP_MOTOR);
// white while reset
analogWrite(RED_PIN, 15);
analogWrite(GREEN_PIN, 15);
analogWrite(BLUE_PIN, 15);
delay(500);

// reset states for all motors
arm_state[0] = BASE_MIN;
arm_state[1] = TOP_MIN;
arm_state[2] = CUP_MIN;
break;

case STOP:
// remain in this state unless input signal changes
while (!signal_changed) {
Serial.println("STOP ROUTINE INITIATED");

// red while in stop
analogWrite(RED_PIN, 255);
analogWrite(GREEN_PIN, 0);
analogWrite(BLUE_PIN, 0);

// check for signal change
signal_in_change();
delay(200);
}
// no motors are moved
break;

default: // error: no motors are moved
Serial.println ("INVALID ROUTINE PROCESSED");
}
}


/* instructions followed during runtime */
void loop() {
// manage robot state through signals from FPGA's GPIO
if (!signal_changed) {
 do_routine();
 Serial.println("ROUTINE GRACEFULLY IMPLEMENTED");
 signal_in_change();
 delay(5);
}
else if (signal_changed) {
set_routine ();
signal_in_change ();
Serial.println("SIGNAL GRACEFULLY CHANGED");
delay(5);
}
else { // error
Serial.println("ERROR: INVALID STATE"); // should not reach here
delay(500);
}
// delay 1ms to make stable transition to start of loop
delay (1);
}
```

Code-block 6: Audio

```
module Audio_main   (input       CLOCK_50,
                    input [0:0]   KEY,
                    input  send_search,
 send_place,
 send_stop,
                    inout        AUD_BCLK,
                                 AUD_ADCLRCK,
                                 AUD_DACLRCK,

                    input        AUD_ADCDAT,
                    output       AUD_XCK,
                                 AUD_DACDAT
                    inout        FPGA_I2C_SDAT,
                    output       FPGA_I2C_SCLK);

/*****************************************************************************
 * Internal Wires and Registers Declarations *
 *****************************************************************************/

// Internal Wires
wire                audio_in_available;
wire          [31:0] left_channel_audio_in;
wire          [31:0] right_channel_audio_in;
wire                read_audio_in;

wire                audio_out_allowed;
wire          [31:0] left_channel_audio_out;
wire          [31:0] right_channel_audio_out;
wire                write_audio_out;
```

```verilog
wire         [5:0] audio_out_ram;
wire         [15:0] address_count;

assign read_audio_in = audio_in_available & audio_out_allowed;
assign write_audio_out = audio_in_available & audio_out_allowed;

reg [15:0] address;
reg [15:0] cycleCount;
reg [15:0] address_start, address_end;

always @(posedge CLOCK_50) // triggered on edges of clock
begin
       if (~KEY[0])
       begin
              address_start <= 16'd0;
              address_end <= 16'd0;
              cycleCount <= 16'd0;
              address <= address_start;
       end

       if (send_search)
       begin
              address_start <= 16'd0;
              address_end <= 16'd27100;
              cycleCount <= 16'd0;
              address <= address_start;
       end

       else if (send_place)
       begin
              address_start <= 16'd27101;
              address_end <= 16'd43830;
              cycleCount <= 16'd0;
              address <= address_start;
       end

       else if (send_stop)
       begin
              address_start <= 16'd43831;
              address_end <= 16'd54300;
              cycleCount <= 16'd0;
              address <= address_start;
       end

       // when reach depth, reset
       if (cycleCount== 16'd2000) // synchronous active-high
       begin
              address <= address + 16'd1;
              cycleCount <= 16'd0;
       end
```

```verilog
        else
        begin
                cycleCount <= cycleCount + 16'd1;
        end

        if (address == address_end)
        begin
                cycleCount <= 16'd0;
                address <= address_start;
        end

end


assign address_count = address;


audio_ram2 ar (
        .address(address_count),
        .clock(CLOCK_50),
        .data(),
        .wren(1'b0),
        .q(audio_out_ram));


Audio_ControllerAC  (// Inputs
                    .CLOCK_50(CLOCK_50),
                    .reset(~KEY[0]),

                    .clear_audio_in_memory(),
                    .read_audio_in(read_audio_in),

                    .clear_audio_out_memory(),
                    .left_channel_audio_out({audio_out_ram, 26'b0}),
                    .right_channel_audio_out(32'b0),
                    .write_audio_out(1'b1),

                    .AUD_ADCDAT(AUD_ADCDAT),

                    // Bidirectionals
                    .AUD_BCLK(AUD_BCLK),
                    .AUD_ADCLRCK(AUD_ADCLRCK),
                    .AUD_DACLRCK(AUD_DACLRCK),

                    // Outputs
                    .left_channel_audio_in(left_channel_audio_in),
                    .right_channel_audio_in(right_channel_audio_in),
                    .audio_in_available(audio_in_available),

                    .audio_out_allowed(audio_out_allowed),
                    .AUD_DACDAT(AUD_DACDAT),
                    .AUD_XCK(AUD_XCK));
endmodule // Audio_main
```

Code-block 7: Top-level module

```
// ============================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ============================================================================
//
// Permission:
//
// Terasic grants permission to use and modify this code for use
// in synthesis for all Terasic Development Boards and Altera Development
// Kits made by Terasic.Other use of this code, including the selling
// ,duplication, or modification of any portion is strictly prohibited.
//
// Disclaimer:
//
// This VHDL/Verilog or C/C++ source code is intended as a design reference
// which illustrates how these types of functions can be implemented.
// It is the user's responsibility to verify their design for
// consistency and functionality through the use of formal
// verification methods.Terasic provides no warranty regarding the use
// or functionality of this code.
//
// ============================================================================
//
//Terasic Technologies Inc
//9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
// web: http://www.terasic.com/
// email: support@terasic.com
//
// ============================================================================
//Date:Mon Jun 17 20:35:29 2013
// ============================================================================

//`define ENABLE_HPS

module DE1_SoC_TV (

///////// ADC /////////
inoutADC_CS_N,
output ADC_DIN,
inputADC_DOUT,
output ADC_SCLK,

///////// AUD /////////
```

```verilog
inputAUD_ADCDAT,
inoutAUD_ADCLRCK,
inoutAUD_BCLK,
output AUD_DACDAT,
inoutAUD_DACLRCK,
output AUD_XCK,

///////// CLOCK2 /////////
inputCLOCK2_50,

///////// CLOCK3 /////////
inputCLOCK3_50,

///////// CLOCK4 /////////
inputCLOCK4_50,

///////// CLOCK /////////
inputCLOCK_50,

///////// DRAM /////////
output[12:0] DRAM_ADDR,
output[1:0]DRAM_BA,
output DRAM_CAS_N,
output DRAM_CKE,
output DRAM_CLK,
output DRAM_CS_N,
inout [15:0] DRAM_DQ,
output DRAM_LDQM,
output DRAM_RAS_N,
output DRAM_UDQM,
output DRAM_WE_N,

///////// FAN /////////
output FAN_CTRL,

///////// FPGA /////////
output FPGA_I2C_SCLK,
inoutFPGA_I2C_SDAT,

///////// GPIO /////////
output [8:4] GPIO_0,

///////// HEX0 /////////
output[6:0]HEX0,

///////// HEX1 /////////
output[6:0]HEX1,

///////// HEX2 /////////
output[6:0]HEX2,
///////// HEX3 /////////
output[6:0]HEX3,

///////// HEX4 /////////
output[6:0]HEX4,

///////// HEX5 /////////
output[6:0]HEX5,

`ifdef ENABLE_HPS
///////// HPS /////////
inoutHPS_CONV_USB_N,
output[14:0] HPS_DDR3_ADDR,
output[2:0]HPS_DDR3_BA,
output HPS_DDR3_CAS_N,
output HPS_DDR3_CKE,
output HPS_DDR3_CK_N,
output HPS_DDR3_CK_P,
output HPS_DDR3_CS_N,
output[3:0]HPS_DDR3_DM,
inout [31:0] HPS_DDR3_DQ,
inout [3:0]HPS_DDR3_DQS_N,
inout [3:0]HPS_DDR3_DQS_P,
output HPS_DDR3_ODT,
output HPS_DDR3_RAS_N,
output HPS_DDR3_RESET_N,
inputHPS_DDR3_RZQ,
output HPS_DDR3_WE_N,
output HPS_ENET_GTX_CLK,
inoutHPS_ENET_INT_N,
output HPS_ENET_MDC,
inoutHPS_ENET_MDIO,
inputHPS_ENET_RX_CLK,
input [3:0]HPS_ENET_RX_DATA,
inputHPS_ENET_RX_DV,
output[3:0]HPS_ENET_TX_DATA,
output HPS_ENET_TX_EN,
inout [3:0]HPS_FLASH_DATA,
output HPS_FLASH_DCLK,
output HPS_FLASH_NCSO,
inoutHPS_GSENSOR_INT,
inoutHPS_I2C1_SCLK,
inoutHPS_I2C1_SDAT,
inoutHPS_I2C2_SCLK,
inoutHPS_I2C2_SDAT,
inoutHPS_I2C_CONTROL,
inoutHPS_KEY,
inoutHPS_LED,
inoutHPS_LTC_GPIO,
output HPS_SD_CLK,
inoutHPS_SD_CMD,
inout [3:0]HPS_SD_DATA,
output HPS_SPIM_CLK,
inputHPS_SPIM_MISO,
output HPS_SPIM_MOSI,
inoutHPS_SPIM_SS,
inputHPS_UART_RX,
output HPS_UART_TX,
inputHPS_USB_CLKOUT,
inout [7:0]HPS_USB_DATA,
inputHPS_USB_DIR,
inputHPS_USB_NXT,
output HPS_USB_STP,
`endif /*ENABLE_HPS*/

///////// IRDA /////////
inputIRDA_RXD,
output IRDA_TXD,

///////// KEY /////////
input [3:0]KEY,

///////// LEDR /////////
output[9:0]LEDR,

///////// PS2 /////////
inoutPS2_CLK,
inoutPS2_CLK2,
inoutPS2_DAT,
inoutPS2_DAT2,

///////// SW /////////
input [9:0]SW,

///////// TD /////////
input TD_CLK27,
input[7:0]TD_DATA,
input TD_HS,
outputTD_RESET_N,
input TD_VS,

///////// VGA /////////
output[7:0]VGA_B,
output VGA_BLANK_N,
output VGA_CLK,
output[7:0]VGA_G,
output VGA_HS,
output[7:0]VGA_R,
output VGA_SYNC_N,
output VGA_VS);
```

```
//=======================================================
//REG/WIRE declarations
//=======================================================

wire   CLK_18_4;
wire   CLK_25;

//     For Audio CODEC
wire          AUD_CTRL_CLK;//     For Audio Controller

//     For ITU-R 656 Decoder
wire   [15:0] YCbCr;
wire   [9:0]  TV_X;
wire          TV_DVAL;

//     For VGA Controller
wire   [9:0]  mRed;
wire   [9:0]  mGreen;
wire   [9:0]  mBlue;
wire   [10:0] VGA_X;
wire   [10:0] VGA_Y;
wire   VGA_Read;    //     VGA data request
wire   m1VGA_Read;  //     Read odd field
wire   m2VGA_Read;  //     Read even field

//     For YUV 4:2:2 to YUV 4:4:4
wire   [7:0]  mY;
wire   [7:0]  mCb;
wire   [7:0]  mCr;

//     For field select
wire   [15:0] mYCbCr;
wire   [15:0] mYCbCr_d;
wire   [15:0] m1YCbCr;
wire   [15:0] m2YCbCr;
wire   [15:0] m3YCbCr;

//     For Delay Timer
wire   TD_Stable;
Wire DLY0;
wire   DLY1;
wire   DLY2;

//     For Down Sample
wire   [3:0]  Remain;
wire   [9:0]  Quotient;

wire          mDVAL;

wire   [15:0] m4YCbCr;
wire   [15:0] m5YCbCr;
wire   [8:0]  Tmp1,Tmp2;
wire   [7:0]  Tmp3,Tmp4;

wireNTSC;
wirePAL;
//=============================================================================
// Structural coding
//=============================================================================

//      Turn On TV Decoder
assign TD_RESET_N    = 1'b1;
assign LED           = VGA_Y;


assign m1VGA_Read    =      VGA_Y[0]            ?      1'b0          :      VGA_Read     ;
assign m2VGA_Read    =      VGA_Y[0]            ?      VGA_Read      :      1'b0         ;
assign mYCbCr_d      =      !VGA_Y[0]           ?      m1YCbCr              :      m2YCbCr
;
assign mYCbCr        =      m5YCbCr;

assign Tmp1   =      m4YCbCr[7:0]+mYCbCr_d[7:0];
assign Tmp2   =      m4YCbCr[15:8]+mYCbCr_d[15:8];
assign Tmp3   =      Tmp1[8:2]+m3YCbCr[7:1];
assign Tmp4   =      Tmp2[8:2]+m3YCbCr[15:9];
assign m5YCbCr       =      {Tmp4,Tmp3};

//      TV Decoder Stable Check
TD_Detect                  u2     (      .oTD_Stable(TD_Stable),
                                         .oNTSC(NTSC),
                                         .oPAL(PAL),
                                         .iTD_VS(TD_VS),
                                         .iTD_HS(TD_HS),
                                         .iRST_N(KEY[0]));

//      Reset Delay Timer
Reset_Delay                u3     (      .iCLK(CLOCK_50),
                                         .iRST(TD_Stable),
                                         .oRST_0(DLY0),
                                         .oRST_1(DLY1),
                                         .oRST_2(DLY2));

//      ITU-R 656 to YUV 4:2:2
ITU_656_Decoder            u4     (      //      TV Decoder Input
                                  .iTD_DATA(TD_DATA),
                                  //      Position Output
                                  .oTV_X(TV_X),
                                  //      YUV 4:2:2 Output
                                  .oYCbCr(YCbCr),
                                  .oDVAL(TV_DVAL),
                                  //      Control Signals
```

```
                                 .iSwap_CbCr(Quotient[0]),
                                 .iSkip(Remain==4'h0),
                                 .iRST_N(DLY1),
                                 .iCLK_27(TD_CLK27));

//      For Down Sample 720 to 640
DIV                       u5     (      .aclr(!DLY0),
                                        .clock(TD_CLK27),
                                        .denom(4'h9),
                                        .numer(TV_X),
                                        .quotient(Quotient),
                                        .remain(Remain));

//      SDRAM frame buffer
Sdram_Control_4Port u6     (//    HOST Side
                          .REF_CLK(TD_CLK27),
                          .CLK_18(AUD_CTRL_CLK),
                          .RESET_N(DLY0),
                          //     FIFO Write Side 1
                          .WR1_DATA(YCbCr),
                          .WR1(TV_DVAL),
                          .WR1_FULL(WR1_FULL),
                          .WR1_ADDR(0),
                          .WR1_MAX_ADDR(NTSC ? 640*507 : 640*576),//    525-18
                          .WR1_LENGTH(9'h80),
                          .WR1_LOAD(!DLY0),
                          .WR1_CLK(TD_CLK27),
                          //     FIFO Read Side 1
                    .RD1_DATA(m1YCbCr),
                    .RD1(m1VGA_Read),
                    .RD1_ADDR(NTSC ? 640*13 : 640*22), //   Read odd field and bypess blanking
                          .RD1_MAX_ADDR(NTSC ? 640*253 : 640*262),
                          .RD1_LENGTH(9'h80),
                    .RD1_LOAD(!DLY0),
                          .RD1_CLK(TD_CLK27),
                          //     FIFO Read Side 2
                          .RD2_DATA(m2YCbCr),
                    .RD2(m2VGA_Read),
                    .RD2_ADDR(NTSC ? 640*267 : 640*310),// Read even field and bypess blanking
                          .RD2_MAX_ADDR(NTSC ? 640*507 : 640*550),
                          .RD2_LENGTH(9'h80),
                    .RD2_LOAD(!DLY0),
                          .RD2_CLK(TD_CLK27),
                          //     SDRAM Side
                          .SA(DRAM_ADDR),
              .BA(DRAM_BA),
                     .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_UDQM,DRAM_LDQM}),
                           .SDR_CLK(DRAM_CLK));

//      YUV 4:2:2 to YUV 4:4:4
YUV422_to_444       u7      (      //      YUV 4:2:2 Input
                                              .iYCbCr(mYCbCr),
                                              //      YUV    4:4:4 Output
                                              .oY(mY),
                                              .oCb(mCb),
                                              .oCr(mCr),
                                              //      Control Signals
                                              .iX(VGA_X-160),
                                              .iCLK(TD_CLK27),
                                              .iRST_N(DLY0));

//      YCbCr 8-bit to RGB-10 bit
YCbCr2RGB           u8            (//    Output Side
                                 .Red(mRed),
                                 .Green(mGreen),
                                 .Blue(mBlue),
                                 .oDVAL(mDVAL),
                                 //      Input Side
                                 .iY(mY),
                                 .iCb(mCb),
                                 .iCr(mCr),
                                 .iDVAL(VGA_Read),
                                 //      Control Signal
                                 .iRESET(!DLY2),
                                 .iCLK(TD_CLK27));

//      VGA Controller
wire [9:0] vga_r10;
wire [9:0] vga_g10;
wire [9:0] vga_b10;
assign VGA_R = vga_r10[9:2];
assign VGA_G = vga_g10[9:2];
assign VGA_B = vga_b10[9:2];

//      color filter
wire [29:0] filter_out;
wire [9:0] counter_filter;

Color_Filter CF1    (.clk(TD_CLK27),
                     .oVGA_Red(mRed),
                     .oVGA_Green(mGreen),
                     .oVGA_Blue(mBlue),
                     .filtered_color(filter_out),
                     .SW(SW[8:6]),
                     .KEY(KEY[2:0]),
                     .counter_threshold(counter_filter),
                    .current_state(current_state_filter));

wire [10:0] x_curr, y_curr, x_start, y_start, x_end, y_end, x_center, y_center, x_out, y_out;
wire [8:0] count;
wire [7:0] counter_wire;
wire [29:0] output_filtered;
assign x_curr = VGA_X;
assign y_curr = VGA_Y;
wire [2:0] current_state_pixel, current_state_filter;

pixel_sequence_detector   P1     (.input_pixel (filter_out),
                                  .vsync (TD_VS),
                                  .hsync (TD_HS),
                                  .clock (TD_CLK27),
                                  .x_curr (x_curr),
                                  .y_curr (y_curr),
                                  .x_start (x_start),
                                  .y_start (y_start),
                                  .x_end (x_end),
                                  .y_end (y_end),
                                  .x_center (x_center),
                                  .y_center (y_center),
                                  .x_out (x_out),
                                  .y_out (y_out),
                                  .count (count),
                                  .counter_wire(counter_wire),
                                  .KEY    (KEY [2:0]),
                                  .object_found (object_found),
                                  .output_pixel (output_filtered),
                                  .SW(SW[8:6]),
                                  .current_state_pixel(current_state_pixel));

wire          object_found, counted, implement, reset_ctr;
wire [4:0]    signal_out;

assign GPIO_0[8:4] = signal_out;

GPIO_Arduino ARD1   (.reset(~KEY[3]),
                    .object_found(object_found),
                    .clock_50(CLOCK_50),
                    .sig_out(signal_out),
                    .counted(counted),
                    .implement(implement),
                    .start_ctr(reset_ctr),
                    .SW(SW[9]));

reg [23:0] Hex_wire;

always @ (posedge CLOCK_50)
begin
      if ((SW[1] == 0) & (SW[2] == 0))
```

```
       begin
       Hex_wire <= {signal_out[4:1], 3'd0, reset_ctr, 3'd0, counted, 3'd0, implement, 3'd0,
signal_out[0], 3'd0, object_found};

       end
       else if ((SW[1] == 1) & (SW[2] == 0))
       begin
       Hex_wire <= {1'd0, current_state_pixel, 4'd0, 8'd0, counter_wire};
       end
       else if ((SW[2] == 1) & (SW[1] == 0))
       begin
       Hex_wire <= {1'd0, current_state_filter, 8'd0, 2'd0, counter_filter};
       end
end

SEG7_LUT_6    u0     (.oSEG0(HEX0),
                      .oSEG1(HEX1),
                      .oSEG2(HEX2),
                      .oSEG3(HEX3),
                      .oSEG4(HEX4),
                      .oSEG5(HEX5),
                      .iDIG(Hex_wire));

// display selector
reg [29:0] output_pixel2;
always @ (posedge CLOCK_50)
       begin
              if (SW[0] == 1) output_pixel2 <= output_filtered;
              else output_pixel2 <= {mRed, mBlue, mGreen};
       end


VGA_Ctrl      u9     (//    Host Side
                      .iRed(output_pixel2[29:20]),
                      .iGreen(output_pixel2[9:0]),
                      .iBlue(output_pixel2[19:10]),
                      .oCurrent_X(VGA_X),
                      .oCurrent_Y(VGA_Y),
                      .oRequest(VGA_Read),
                      //     VGA Side
                      .oVGA_R(vga_r10),
                      .oVGA_G(vga_g10),
                      .oVGA_B(vga_b10),
                      .oVGA_HS(VGA_HS),
                      .oVGA_VS(VGA_VS),
                      .oVGA_SYNC(VGA_SYNC_N),
                      .oVGA_BLANK(VGA_BLANK_N),
                      .oVGA_CLOCK(VGA_CLK),
                      //     Control Signal
                      .iCLK(TD_CLK27),
                      .iRST_N(DLY2));
```

```
//     Line buffer, delay one line
Line_Buffer  u10    (.aclr(!DLY0),
                    .clken(VGA_Read),
                    .clock(TD_CLK27),
                    .shiftin(mYCbCr_d),
                    .shiftout(m3YCbCr));

Line_Buffer u11     (.aclr(!DLY0),
                    .clken(VGA_Read),
                    .clock(TD_CLK27),
                    .shiftin(m3YCbCr),
                    .shiftout(m4YCbCr));

//     Audio CODEC and video decoder setting
I2C_AV_Config       u1     (//    Host Side
                    .iCLK(CLOCK_50),
                    .iRST_N(KEY[0]),
                    //     I2C Side
                    .I2C_SCLK(FPGA_I2C_SCLK),
                    .I2C_SDAT(FPGA_I2C_SDAT));

Audio_main   audio  (.CLOCK_50(CLOCK_50),
                    .KEY(KEY[0]),
                    .object_found(object_found),
                    .AUD_BCLK(AUD_BCLK),
                    .AUD_ADCLRCK(AUD_ADCLRCK),
                    .AUD_DACLRCK(AUD_DACLRCK),
                    .AUD_ADCDAT(AUD_ADCDAT),
                    .AUD_XCK(AUD_XCK),
                    .AUD_DACDAT(AUD_DACDAT));
endmodule // DE1_SoC_TV
```

**Appendix B: Selected schematics**

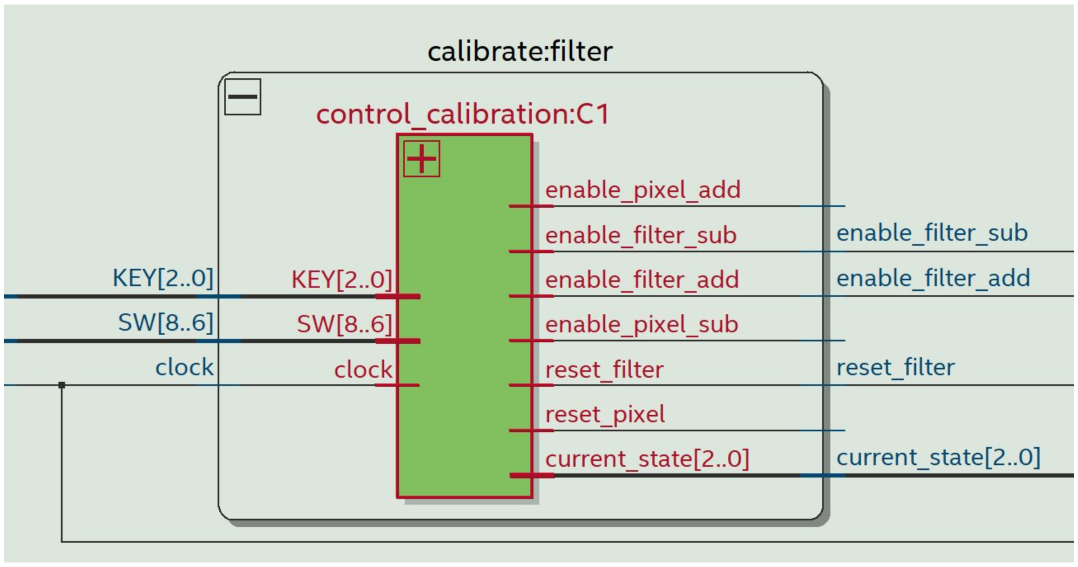


Figure 1: Top-level schematic of the calibration module

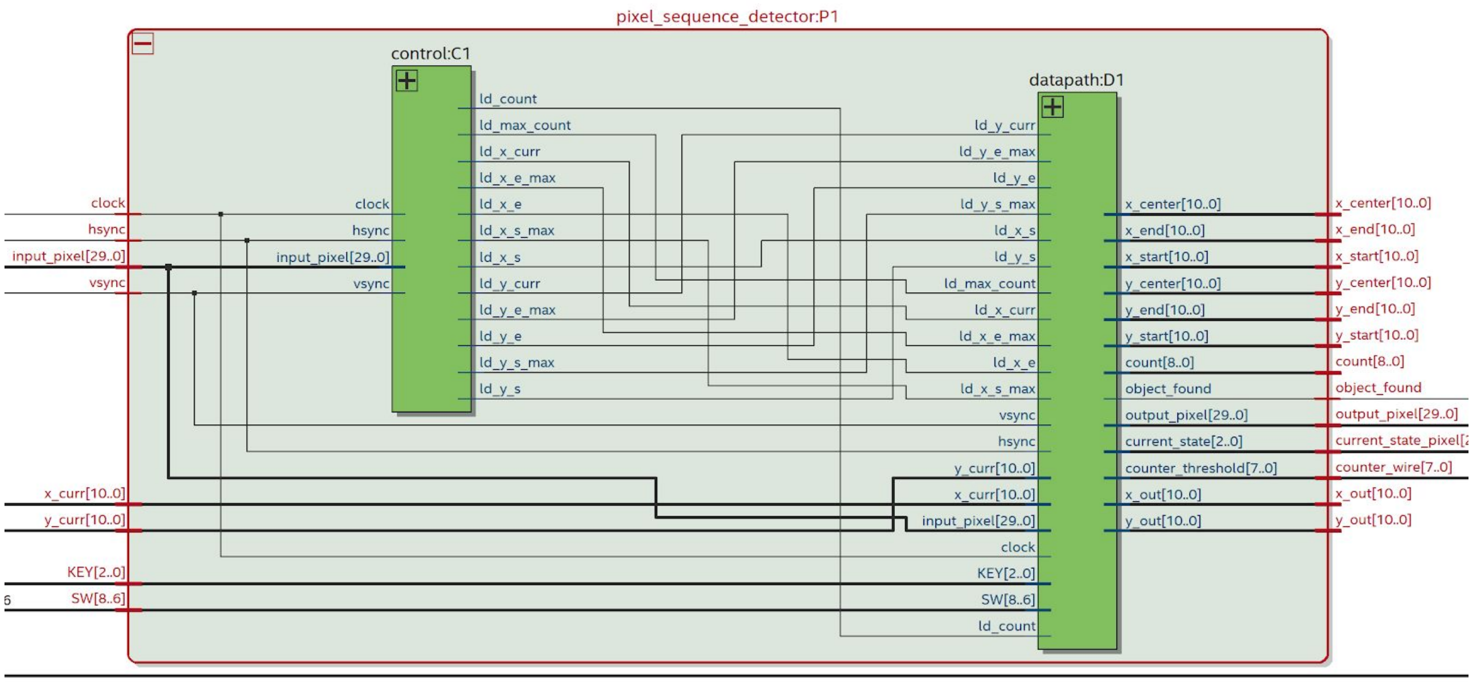


Figure 2: Top-level schematic of Pixel Sequence Detector module

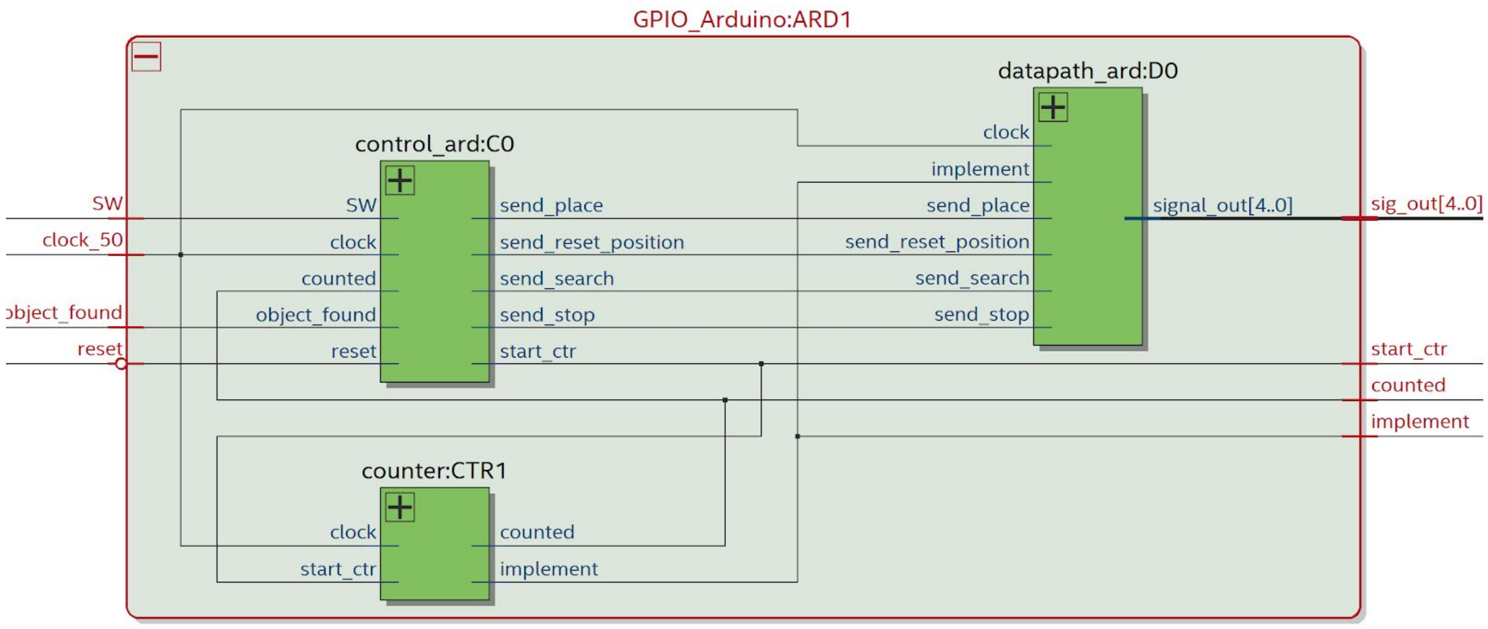


Figure 3: Top-level schematic of the GPIO-Arduino Interface module