Image Feature Detection and Matching

Matthew Conover

6.111 Final Project Report

Introduction

For my final project I implemented the corner detection method oFAST (Oriented Features from Accelerated Segment Test) on an FPGA, along with UART communication to allow the transfer of data between the FPGA and a PC. This was done on the Nexys4 DDR, using a Teensy 3.2 for serial communication.

Goals

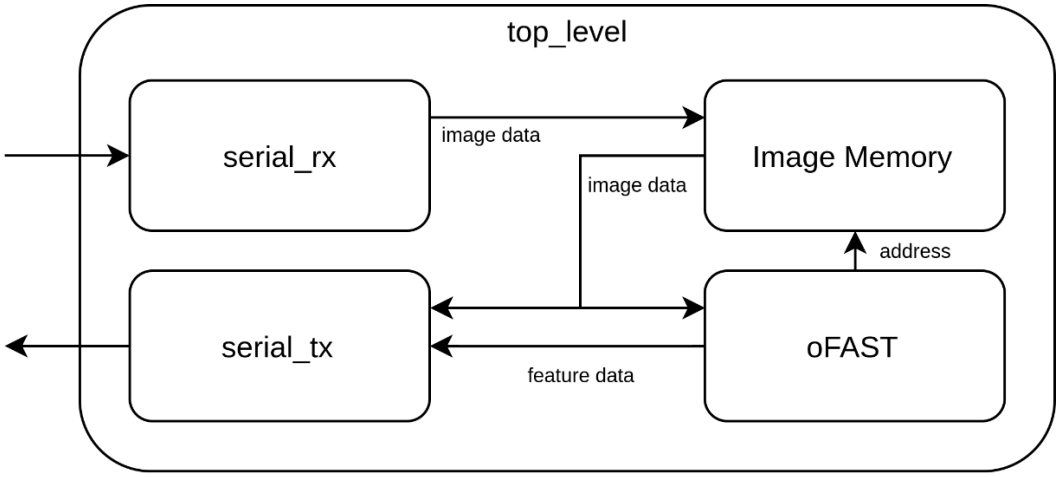
Commitment: UART, oFAST, rBRIEF (Rotated Binary Robust Independent Elementary Features)

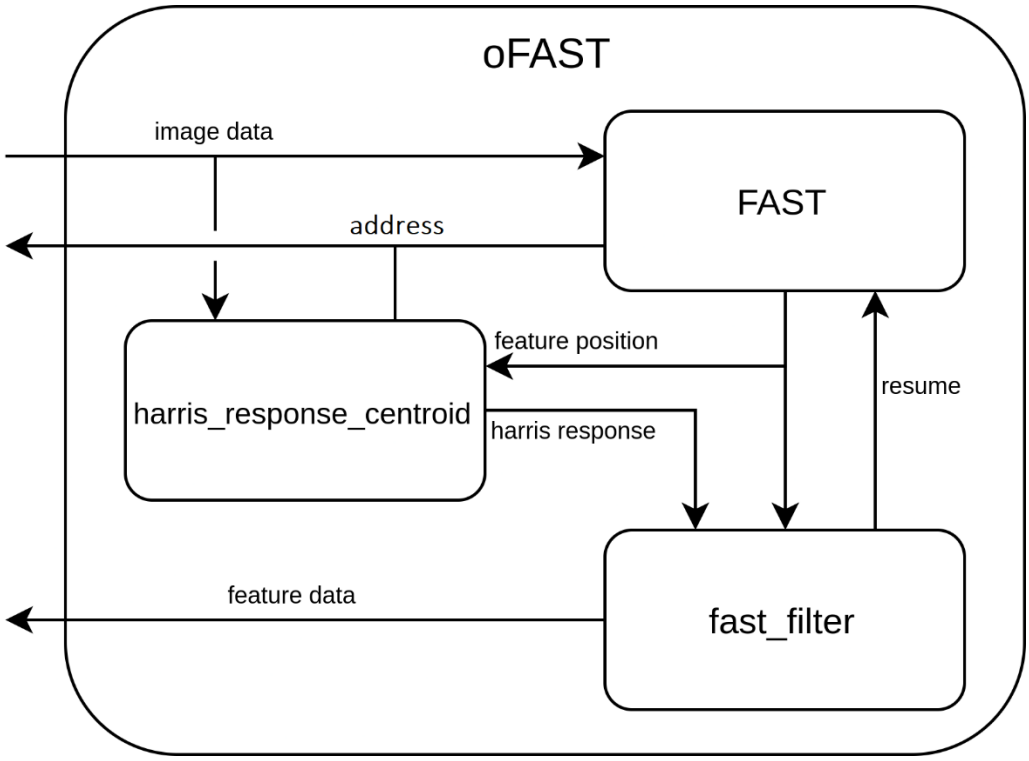
Expected: Feature matching, full communication between oFAST, rBRIEF and memory

Stretch: Features detected and matched between frames of live video

I initially underestimated the complexity of oFAST, so I was not able achieve all of the goals I had set out for this project. Despite this, I am content with the progress and level of complexity I was able to accomplish.

Block Diagrams





Subsystems

top_level

This top module controls the flow of data to/from UART, memory, and oFAST. When idling it waits for a new command to come over UART. It has three major modes: “Get image” which controls saving a new image into memory, “Send image” which sends the current image back to the user, and “oFAST” which directs the output of the oFAST module to the UART transceiver. Once a task is completed it returns to waiting for a new command.

serial_rx

This module handles UART input, it operates at 64 MHz, and is designed to expect one stop bit with no parity bits. The port itself operates at a 4 MHz baud rate, this allows for oversampling by 16, which provides resilience against noise, and distortions in the signal caused by its high frequency.

The receiver takes samples from the port, which idles at high. Upon measuring a low it checks to see whether the next two samples are also low, to ensure it is not simply noise. If it passes this, it delays by a few cycles until it would be in the center of the start bit being sent, and takes 3 samples. If all of these samples are low this is considered a valid start bit, otherwise it goes back to idling.

With the start bit detected, the receiver delays several cycles waiting until it would be in the center of the next bit, takes 5 consecutive measurements centered there, and determines the value of the bit based on whether these measurements meet the cutoff provided to the module. After this the bit value is saved and this process is repeated for the rest of the data bits and the stop bit. At the end, it checks whether the stop bit is high, if it is the data is sent to the rest of the system, otherwise it is discarded, after this the receiver returns to idle.

I had many issues with this module over the course of this project. The first issue was the actual methodology, which was resolved the easiest. Very early on I had tried operating it at 4MHz, taking one sample per bit, and quickly realized this was not practical. Next, I tried oversampling, and taking 16 samples over the entire bit, starting at the initial low signal sustaining itself for a couple samples. This also failed, likely due to the actual width of the signal varying between bits biasing the result. After doing a little research, it seemed the best way was to take multiple samples in what was expected to be the center of the bit. This has worked well.

This did not resolve all of the issues however, and the rest were a bit harder to identify. What made it especially difficult was that the issues did not appear in simulations and would randomly appear between different bitstream generations. The issues that compounded together was the metastability and data loss of moving the done signal and received data between the main and serial clock domains, as well as metastability of the incoming port signal. I attribute a lot of the challenge with this to the issue of identification of the problems, which came down to my inexperience dealing with this type of issue. I certainly will not forget this moving forward.

serial_tx

This module handles UART output, it operates at 4 MHz and includes one stop bit and no parity bit. Its operation is more straightforward than the receiver, a byte is provided, and it sets the port's value accordingly each clock cycle. I did however also have similar issues with it as I did the receiver, with metastability between the clock domains. I worked to resolve these issues in parallel with those of the receiver, however they appeared far less frequently and did not pose as large an issue overall.

oFAST

This module controls the flow of data between FAST, harris_response_centroid, and fast_filter. When FAST detects a feature, it is stalled as the feature gets passed to harris_response_centroid to obtain information about the feature, after which it is passed to the fast_filter. The module then waits for a response from fast_filter, either communicating it has finished, or that it has outputted new features from the filter to send to top_level. Once this has been completed, FAST detection is resumed, and the process repeats until FAST has completed iterating over the entire image. The module then checks the fast_filter for any remaining features to output, then finishes and communicates it is done to top_level.

The pipelining of these modules could be improved, with a separate module directing the output of the fast_filter to top_level, FAST would not have to be stalled for as long.

FAST

This module iterates over the pixels in an image, applying the FAST test to each, and if it is a valid feature its position is outputted, then the module stalls until instructed to resume.

For each test it loads a small patch of surrounding pixel data from memory and checks whether the intensity of the pixels in a circle around the center pixel are above or below that of the center pixel by some threshold intensity. The results of these tests are stored in two bit-strings. There is considered a feature at this location if there are twelve consecutive pixels in the circle that pass together one of the two intensity tests. First, a quick test is then done on the bit-strings to determine whether twelve consecutive passes is possible, if it passes it continues to the more involved test, otherwise it moves on to the next location in the image to test for a feature. The longer test involves checking each twelve-bit window in the bit-strings.

The throughput of this module could be further improved in a couple ways. First, as of now it loads the image patch from scratch each time, however in general there is significant overlap between patches each iteration. Instead of reloading, a circular buffer could be used, and only new data reloaded for a row when it is necessary. Second, parallelizing the loading of the

rows would also increase throughput, instead of waiting for it to be received and saving it to the patch buffer sequentially.

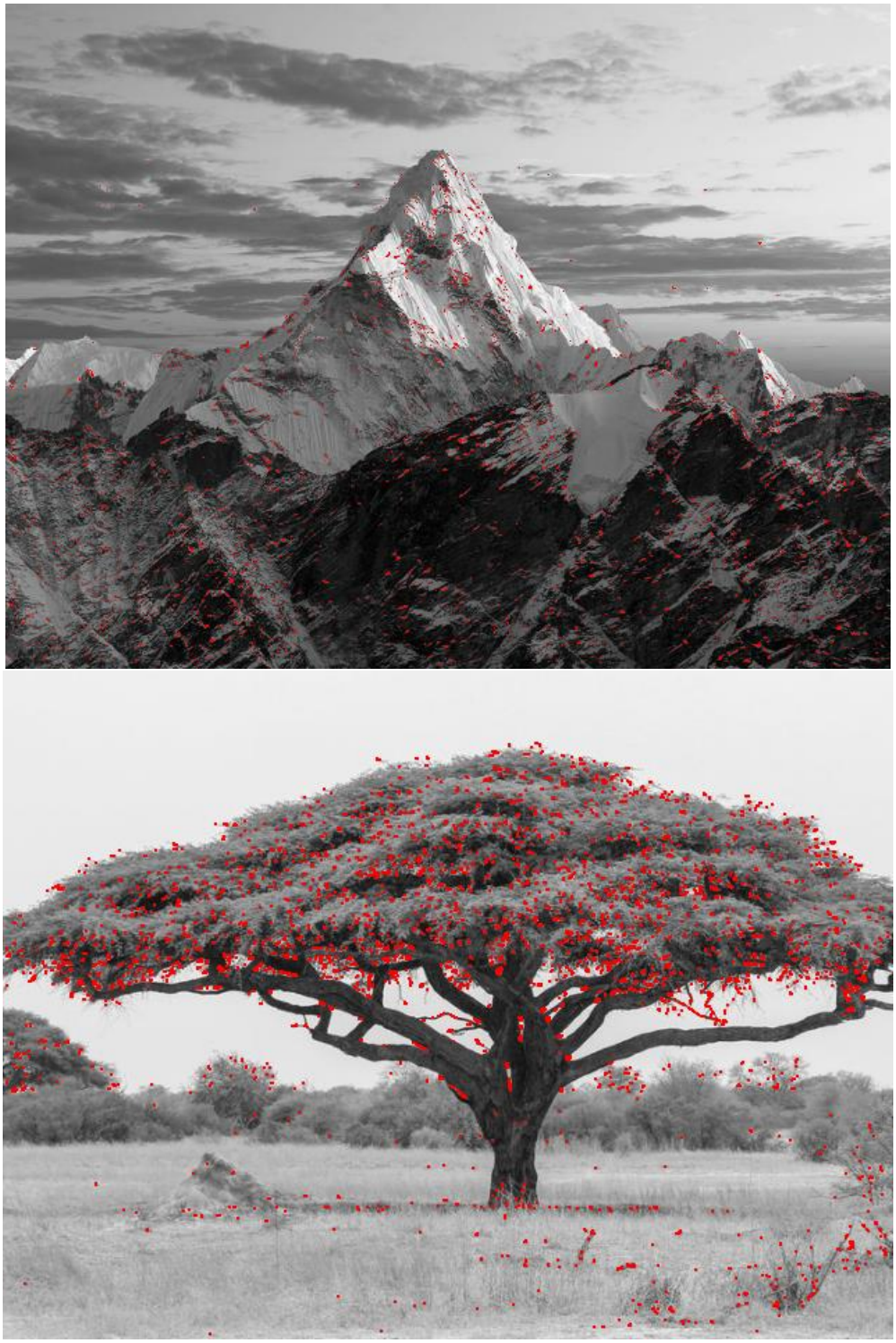


Fig 1. FAST features detected, threshold = 28, no filtering.

harris_response_centroid

The role of this module is to calculate the Harris response of a 21x21 patch centered at the position of a feature, as well as calculate its centroid and use that centroid to get a measure for the orientation of the feature. It stores three rows of the patch at a time in a circular buffer, allowing it to calculate the derivatives necessary for the Harris response. The Harris response itself is computed from the determinant and trace of the structure matrix, the contributions to the elements of this matrix are calculated in parallel with that component to the centroid over multiple stages. The final value for the Harris response is stored as a 32-bit unsigned integer, with a negative value for the response overwritten to zero. The x and y components of the centroid are used as indexes in a 2d array to acquire a discrete measure for the angle, ranging from 0-31. The values in this array were calculated with arctan2 using Python.

The latency of this module could be improved by increasing the size of the circular buffer by a row, and not pausing the loading of the rows for the calculation. Throughput is not as important, as this module will only be used once a feature has been located, but as this requires memory access FAST must stall until it is finished, making latency important. This stalling of FAST could also be avoided if the BRAM used to store the image was dual port.

fast_filter

The job of this module is to apply Adaptive Non-Maximal Suppression (ANMS) to the features detected by FAST, using the Harris response calculated by the harris_response_centroid module. It checks to see if each feature has a Harris response that is at least 12.5% higher than all of the other features in some radius around itself. Those that pass this are outputted from the filter, the rest are eventually discarded.

To accomplish this, I created a linked-list structure in BRAM, with each element containing feature data and the index of the next element. Along with this, each feature has a "valid" bit associated with it, which refers to whether the feature has yet to be disqualified as a feature by not meeting the criteria defined above.

When a new feature is sent to the filter, the filter compares the new feature to every other feature stored in the list. If the list's feature is within some radius of the new feature their Harris responses are compared. If the Harris response of the list's feature is not more than 12.5% higher than the new feature its valid bit set to zero, and if the Harris response of the new feature is not 12.5% higher than that of the list's feature its valid bit will be set to zero. At this end of this process, regardless of the value of the valid bit, the new feature is appended to the linked list. This is because even if the feature has been disqualified, it could still disqualify future incoming features, so it must be stored in the list.

The FAST detection goes one row at a time down the image, so once the y position difference between a new feature and one in the list is larger than the radius of influence, no future features will be within that radius. So, once this happens the feature in the list can be removed from this list and outputted by the filter if it is valid, freeing up space.

The linked list structure itself operates by having a stack of available addresses in memory where a new feature can be located. When a new feature is added its address is popped off this stack and used, and when a feature is removed its address is added to the stack, while the next-element address contained in the previous element in the linked list is updated to be that of the following element.

By using this strategy, the total number of features in the image do not matter, but rather the maximum number of features over a range of rows the size of the radius. If the radius was ten, as long as the largest number of features over a stretch of ten rows was below the capacity of the filter, then there would be no issues. This also means that the radius can be adjusted to deal with images with too many features for the current capacity.

Once all the features have been detected for a given image, the filter “prints” any remaining elements in the list which are valid, outputting them, then cleans the address stack back to its original state.

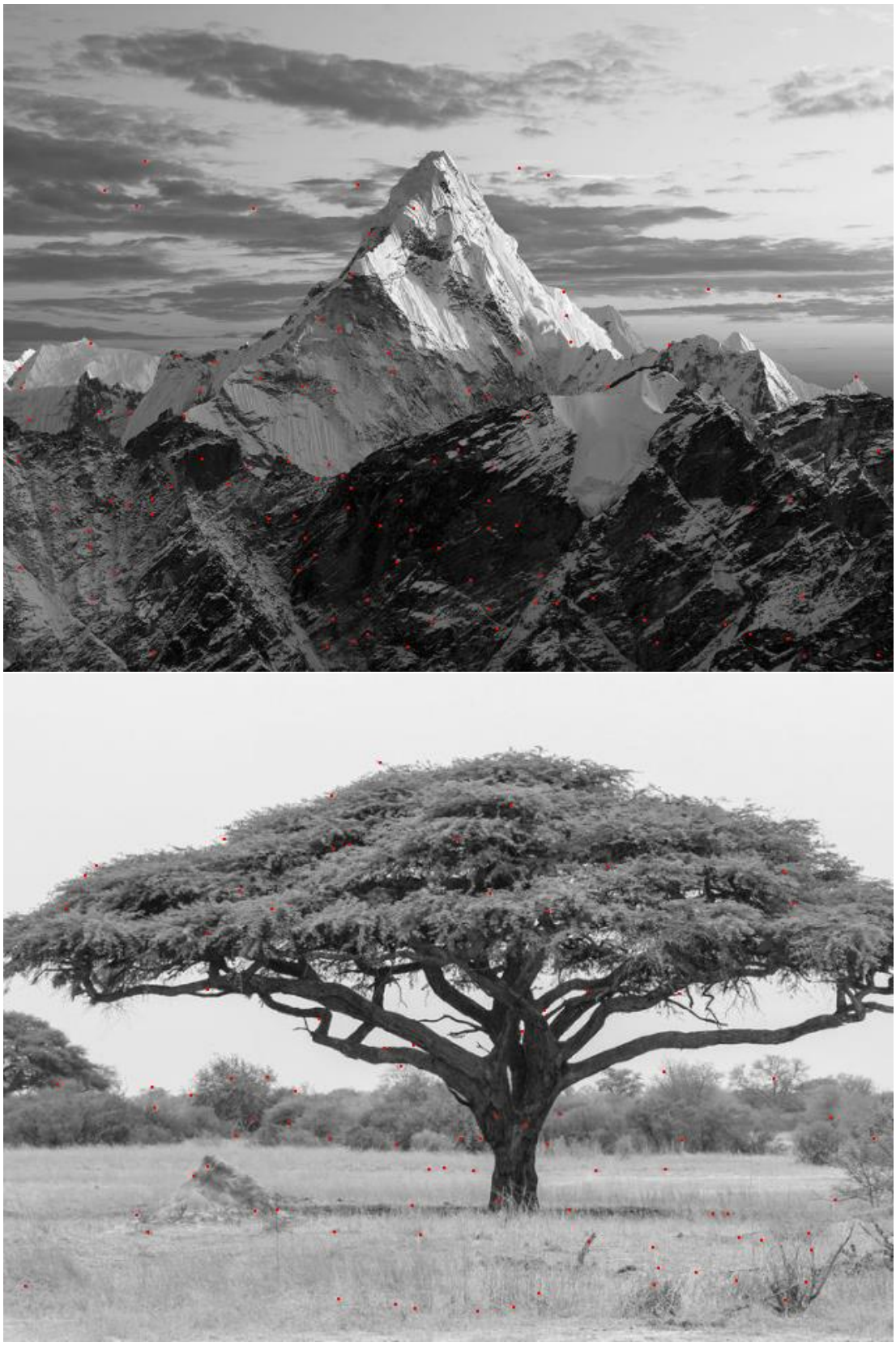


Fig 2. oFAST features detected, threshold = 28, filter radius = 10.

Conclusion

With the functioning oFAST module completed, the next steps would be to implement the improvements mentioned earlier, then rBRIEF and feature matching for fully operational ORB (Oriented FAST Rotated BRIEF) feature detection, description, and matching.

From this project I have learned specific lessons, such as how to identify issues arising from clock domain crossing and metastability, along with broader lessons like how to handle projects with a higher level of complexity than I had been exposed to up to this point, and the importance of making effective simulations for debugging purposes. This experience will be useful for future projects.

Appendix

```
`timescale 1ns / 1ps
module top_level(
    input clk_100mhz,
    input btnr,
    input uart_txd_in,
    input ja[1:0],
    input [15:0] sw,
    output logic jb[1:0],
    output logic [15:0] led
);
    assign jb[1] = 0;

    //STATES
    parameter AWAITING_CMD = 8'b0000_0000;
    parameter TEST_SERIAL = 8'b1000_0000;
    logic [31:0] data_index = 0;
    logic img_data_type = 0; //0: size, 1: pixels
    parameter UART_GET_IMG_DATA = 8'b0001_0000;
    parameter UART_GET_IMG_DATA_AWAITING = 8'b0001_0001;
    parameter UART_GET_IMG_DATA_STORE_SIZE = 8'b0001_0010;
    parameter UART_GET_IMG_DATA_STORE_PIXEL = 8'b0001_0011;
    parameter UART_GET_IMG_DATA_STORE_PIXEL_BLOCK = 8'b0001_0100;
    parameter UART_GET_IMG_DATA_DONE = 8'b0001_0101;

    parameter UART_SEND_IMG_DATA = 8'b0010_0000;
    parameter UART_SEND_IMG_DATA_AWAITING = 8'b0010_0001;
    parameter UART_SEND_IMG_DATA_WRITE_SIZE = 8'b0010_0010;
    parameter UART_SEND_IMG_DATA_WRITE_PIXEL = 8'b0010_0011;
    parameter UART_SEND_IMG_DATA_DONE = 8'b0010_0100;

    parameter OFAST = 8'b0101_0000;
    parameter OFAST_MAIN = 8'b0101_0001;
    parameter OFAST_AWAITING = 8'b0101_0010;
    parameter OFAST_SEND = 8'b0101_0011;
    parameter OFAST_DONE = 8'b0101_0100;

    logic [7:0] state = AWAITING_CMD;
    logic [7:0] ret_state = AWAITING_CMD;
    logic [71:0] data_buff = 72'b0;
    logic [3:0] data_buff_size = 0;

    //CLOCKS

    logic clk_64mhz;
```

```
    logic clk_4mhz = 0;
    logic clk_1hz = 0;
    logic [21:0] clk_1hz_counter = 0;
    logic [3:0] clk_4mhz_counter = 0;
    clk_wiz_64mhz clk_gen_64mhz (.clk_in1(clk_100mhz),.clk_out1(clk_64mhz));
    logic clk_8mhz;
    clk_wiz_8mhz clk_gen_8mhz (.clk_in1(clk_100mhz),.clk_out1(clk_8mhz));
    always_ff @(posedge clk_8mhz) begin
        clk_4mhz <= ~clk_4mhz;
    end

    //UART
    logic [7:0] serial_rx_data;
    logic serial_read;
    logic [2:0] uart_cutoff;
    assign uart_cutoff = 5;
    serial_rx serial_rx1
(.clk_main(clk_100mhz),.clk_serial(clk_64mhz),.raw_port(ja[0]),.recieved(serial_read),.data(serial_rx_data),.cutoff(uart_cutoff));

    logic [7:0] serial_tx_data;
    logic serial_tx_ready;
    logic serial_write = 0;
    serial_tx serial_tx1
(.clk_main(clk_100mhz),.clk_serial(clk_4mhz),.write(serial_write),.data(serial_tx_data),.ready(serial_tx_ready),.port(jb[0]));

    //MEMORY
    logic [16:0] img_mem_addr;
    logic [31:0] img_mem_read_data;
    logic [31:0] img_mem_write_data;
    logic img_mem_write = 0;

    logic [15:0] img_dim_x = 0;
    logic [15:0] img_dim_y = 0;
    logic [31:0] img_size;
    logic [31:0] img_block_buff = 32'h0000_0000;
    assign img_size = img_dim_x*img_dim_y;

    img_mem_gen_640x480_32 img_mem (.addra(img_mem_addr),
                                .clka(clk_100mhz),
                                .dina(img_mem_write_data),
                                .douta(img_mem_read_data),
                                .wea(img_mem_write),
                                .ena(1));

    logic [5:0] ofast_mode;
    logic [31:0] ofast_img_data;
    logic [16:0] ofast_img_addr;
    logic ofast_img_write;

    logic ofast_stream_ready = 0;
    logic ofast_stream_write;
    logic [71:0] ofast_stream_data;
    logic [3:0] ofast_stream_data_size;

    logic [7:0] ofast_threshold;
    logic ofast_start;
    logic ofast_reset;
    logic ofast_done;
    oFAST ofast (.clk(clk_100mhz),.mode(ofast_mode),.img_size_x(img_dim_x),.img_size_y(img_dim_y),
            .img_data(img_mem_read_data),.img_addr(ofast_img_addr),.stream_ready(ofast_stream_ready),
            .stream_write(ofast_stream_write),.stream_data(ofast_stream_data),.stream_data_size(ofast_stream_data_size),
            .threshold(sw[9:2]),.radius(sw[15:10]),.start(ofast_start),.reset(ofast_reset),.done(ofast_done));


    assign led[15:8] = state;
```

```
always_ff @(posedge clk_100mhz) begin
    if(btnr) begin
        state <= AWAITING_CMD;
    end else begin
        case(state)
            AWAITING_CMD: begin
                img_mem_write <= 0;
                serial_write <= 0;
                if(serial_read) begin
                    state <= {serial_rx_data[7:4],4'b0};
                end
            end

            TEST_SERIAL: begin
                if(serial_read&(~serial_write)) begin
                    serial_tx_data <= serial_rx_data;
                    serial_write <= 1;
                end else begin
                    serial_write <= 0;
                end
            end

            //IMAGE FROM UART
            UART_GET_IMG_DATA: begin
                state <= UART_GET_IMG_DATA_AWAITING;
                img_data_type <= 0;
                data_index <= 0;
                img_mem_write <= 0;
                img_mem_addr <= 0;;
            end
            UART_GET_IMG_DATA_AWAITING: begin
                if(serial_read) begin
                    if(img_data_type) begin
                        state <= UART_GET_IMG_DATA_STORE_PIXEL;
                        img_mem_write <= 0;
                    end else begin
                        img_data_type <= (data_index[1:0] == 2'b11);
                        state <= UART_GET_IMG_DATA_STORE_SIZE;
                    end
                end
            end
            UART_GET_IMG_DATA_STORE_SIZE: begin
                data_index[1:0] <= data_index[1:0]+1;
                state <= UART_GET_IMG_DATA_AWAITING;
                case(data_index[1:0])
                    0: img_dim_x[7:0] <= serial_rx_data;
                    1: img_dim_x[15:8] <= serial_rx_data;
                    2: img_dim_y[7:0] <= serial_rx_data;
                    3: img_dim_y[15:8] <= serial_rx_data;
                endcase
            end
            UART_GET_IMG_DATA_STORE_PIXEL: begin
                img_block_buff <= {serial_rx_data,img_block_buff[31:8]};
                if(data_index+1 < img_size) begin
                    data_index <= data_index+1;
                    if(data_index[1:0]==2'b11) begin
                        state <= UART_GET_IMG_DATA_STORE_PIXEL_BLOCK;
                    end else begin
                        state <= UART_GET_IMG_DATA_AWAITING;
                    end
                end else begin
                    state <= UART_GET_IMG_DATA_DONE;
                end
            end
            UART_GET_IMG_DATA_STORE_PIXEL_BLOCK: begin
```

```
    state <= UART_GET_IMG_DATA_AWAITING;
    img_mem_write_data <= img_block_buff;
    img_mem_addr <= data_index[18:2]-1;
    img_mem_write <= 1;
end
UART_GET_IMG_DATA_DONE: begin
    state <= AWAITING_CMD;
    case(data_index[1:0])
        3: img_mem_write_data <= img_block_buff;
        2: img_mem_write_data <= img_block_buff>>8;
        1: img_mem_write_data <= img_block_buff>>16;
        0: img_mem_write_data <= img_block_buff>>24;
        default: serial_tx_data <= 8'h00;
    endcase
    img_mem_addr <= data_index[18:2];
    img_mem_write <= 1;
end

//IMAGE TO UART
UART_SEND_IMG_DATA: begin
    state <= UART_SEND_IMG_DATA_AWAITING;
    img_mem_write <= 0;
    img_mem_addr <= 0;
    img_data_type <= 0;
    data_index <= 0;
    serial_write <= 0;
end
UART_SEND_IMG_DATA_AWAITING: begin
    if(serial_tx_ready&(~serial_write)) begin
        if(img_data_type) begin
            if(data_index[1:0] == 2'b00) begin
                img_block_buff <= img_mem_read_data;
                img_mem_addr <= img_mem_addr+1;
            end
            state <= UART_SEND_IMG_DATA_WRITE_PIXEL;
        end else begin
            img_data_type <= (data_index[1:0] == 2'b11);
            state <= UART_SEND_IMG_DATA_WRITE_SIZE;
        end
    end else begin
        serial_write <= 0;
    end
end
UART_SEND_IMG_DATA_WRITE_SIZE: begin
    data_index[1:0] <= data_index[1:0]+1;
    state <= UART_SEND_IMG_DATA_AWAITING;
    serial_write <= 1;
    case(data_index[1:0])
        0: serial_tx_data <= img_dim_x[7:0];
        1: serial_tx_data <= img_dim_x[15:8];
        2: serial_tx_data <= img_dim_y[7:0];
        3: serial_tx_data <= img_dim_y[15:8];
        default: serial_tx_data <= 8'h00;
    endcase
end
UART_SEND_IMG_DATA_WRITE_PIXEL: begin
    data_index <= data_index+1;
    serial_write <= 1;
    serial_tx_data <= img_block_buff[7:0];
    img_block_buff <= img_block_buff>>8;
    if(data_index+1 < img_size) begin
        state <= UART_SEND_IMG_DATA_AWAITING;
    end else begin
        state <= AWAITING_CMD;
    end
```

```
    end



    OFAST: begin
        img_mem_write <= 0;
        ofast_mode <= {sw[1:0],4'b0};
        ofast_start <= 1;
        ofast_stream_ready <= 0;
        state <= OFAST_MAIN;
    end
    OFAST_MAIN: begin
        ofast_start <= 0;
        img_mem_addr <= ofast_img_addr;
        serial_write <= 0;

        if(ofast_done) begin
            state <= OFAST_AWAITING;
            ret_state <= OFAST_DONE;
            data_index[3:0] <= 15;
            ofast_stream_ready <= 0;
            ofast_reset <= 1;
        end else if(ofast_stream_write) begin
            state <= OFAST_AWAITING;
            ret_state <= OFAST_MAIN;
            ofast_stream_ready <= 0;
            data_buff <= ofast_stream_data;
            data_buff_size <= ofast_stream_data_size;
            data_index[3:0] <= 0;
        end else if(serial_tx_ready) begin
            ofast_stream_ready <= 1;
        end
    end
    OFAST_AWAITING: begin
        img_mem_addr <= ofast_img_addr;
        if(serial_tx_ready&(~serial_write)) begin
            state <= OFAST_SEND;
        end else begin
            serial_write <= 0;
        end
    end
    OFAST_SEND: begin
        img_mem_addr <= ofast_img_addr;
        data_index[3:0] <= data_index[3:0]+1;
        case(data_index[3:0])
            0: serial_tx_data <= data_buff_size;
            15: serial_tx_data <= 8'h00;
            default: serial_tx_data <= data_buff[7:0];
        endcase
        serial_write <= 1;
        if(data_index[3:0] < data_buff_size) begin
            data_buff <= (data_index[3:0] != 0) ? data_buff>>8 : data_buff;
            state <= OFAST_AWAITING;
        end else begin
            state <= ret_state;
        end
    end
    OFAST_DONE: begin
        state <= AWAITING_CMD;
        ofast_stream_ready <= 0;
        serial_write <= 0;
        ofast_reset <= 0;
    end
    default: state <= AWAITING_CMD;
endcase
```

```
        end
    end
endmodule


module serial_rx(
    input clk_main,
    input clk_serial,
    input raw_port,
    input [2:0] cutoff,
    output logic recieved,
    output logic [7:0] data
);

    parameter CUTOFF = 5;
    parameter IDLE = 3'b000;
    parameter NEXT = 3'b001;
    parameter HALF_DELAY = 3'b010;
    parameter START_SAMPLE = 3'b011;
    parameter FULL_DELAY = 3'b100;
    parameter BIT_SAMPLE = 3'b101;
    parameter DONE1 = 3'b110;
    parameter DONE2 = 3'b111;
    logic [2:0] state = IDLE;
    logic [3:0] index;
    logic [3:0] counter;
    logic [2:0] sample_buff = 0;
    logic [7:0] data_buff = 0;
    logic [3:0] port_buff = 0;
    logic prev_port;
    logic port;

    logic done_serial = 0;
    logic done_main = 0;
    logic [4:0] done_buff = 0;
    logic [2:0] done_xfer = 0;
    logic prev_done_main = 0;

    always_ff @(posedge clk_main) begin
        done_xfer <= {done_xfer[1:0],done_serial};
        done_buff <= {done_buff[3:0],done_xfer[2]};
        done_main <= (done_buff[0]+done_buff[1]+done_buff[2]+done_buff[3]+done_buff[4])>=3;
        prev_done_main <= done_main;

        if(done_main&(~prev_done_main)&(~recieved)) begin
            recieved <= 1;
            data <= data_buff;
        end else begin
            recieved <= 0;
        end
    end

    always_ff @(posedge clk_serial) begin
        port_buff <= {port_buff[2:0],raw_port};
        port <= port_buff[3];
        case(state)
            IDLE: begin
                done_serial <= 0;
                if(~port) begin
                    state <= NEXT;
                    index <= 0;
                    counter <= 1;
                end
            end
            NEXT: begin
```

```
            counter <= counter+1;
            if(port) begin
               state <= IDLE;
            end else if(counter == 2) begin
               state <= HALF_DELAY;
            end
         end
         HALF_DELAY: begin
            counter <= counter+1;
            if(counter == 6) begin
               state <= START_SAMPLE;
               sample_buff <= 0;
            end
         end
         START_SAMPLE: begin
            counter <= counter+1;
            sample_buff <= {1'b0,port,sample_buff[1]};
            if(counter == 9) begin
               state <= (port&sample_buff[1]&sample_buff[0]) ? IDLE : FULL_DELAY;
            end
         end
         FULL_DELAY: begin
            counter <= counter+1;
            if(counter == 5) begin
               state <= BIT_SAMPLE;
               sample_buff <= 0;
            end
         end
         BIT_SAMPLE: begin
            counter <= counter+1;
            sample_buff <= sample_buff+port;
            if(counter == 10) begin
               if(index == 8) begin
                  if(sample_buff+port >= 3) begin
                     state <= DONE1;
                  end else begin
                     state <= IDLE;
                  end
               end else begin
                  index <= index+1;
                  data_buff <= {(sample_buff+port >= cutoff),data_buff[7:1]};
                  state <= FULL_DELAY;
               end
            end
         end
         DONE1: begin
            done_serial <= 1;
            state <= DONE2;
         end
         DONE2: begin
            state <= IDLE;
         end
      endcase
   end
endmodule

module serial_tx(
   input clk_main,
   input clk_serial,
   input [7:0] data,
   input write,
   output logic ready,
   output logic port
);
   parameter IDLE = 3'b000;
```

```
parameter START = 3'b010;
parameter BUSY = 3'b110;

parameter READ = 3'b011;
parameter WRITE = 3'b100;
parameter PAUSE = 3'b101;


logic [2:0] main_state = IDLE;
//logic [2:0] serial_state_buff = IDLE;
logic [2:0] serial_state = IDLE;
logic [7:0] main_buff = 0;
wire [7:0] main_buff_xfer;
assign main_buff_xfer = main_buff;
logic [7:0] serial_buff;
logic [3:0] index;
logic done = 0;
logic done_main = 0;
logic done_main_prev = 0;
logic [4:0] done_buff = 0;
logic start_sending = 0;
logic [2:0] start_sending_buff = 0;
logic start_sending_serial;
assign start_sending_serial = start_sending_buff[0]&start_sending_buff[1]&start_sending_buff[2];

always_ff @(posedge clk_main) begin
   done_buff <= {done_buff[3:0],done};
   done_main <= (done_buff[0]+done_buff[1]+done_buff[2]+done_buff[3]+done_buff[4]) >= 3;
   done_main_prev <= done_main;
   start_sending_buff <= {start_sending_buff[1:0],start_sending};

   case(main_state)
      IDLE: begin
         if(write) begin
            main_state <= START;
            main_buff <= data;
            ready <= 0;
         end else begin
            ready <= 1;
         end
      end
      START: begin
         start_sending <= 1;
         main_state <= BUSY;
      end
      BUSY: begin
         if(serial_state == READ) begin
            start_sending <= 0;
         end
         if(done_main&(~done_main_prev)) begin
            main_state <= IDLE;
            ready <= 1;
         end
      end
   endcase
end
always_ff @(posedge clk_serial) begin
   case(serial_state)
      IDLE: begin
         if(start_sending_serial) begin
            port <= 0;
            serial_state <= READ;
         end else begin
            port <= 1;
         end
```

```
            done <= 0;
        end
        READ: begin
            serial_state <= WRITE;
            port <= main_buff_xfer[0];
            serial_buff <= {1'b0,main_buff_xfer[7:1]};
            index <= 1;
        end
        WRITE: begin
            index <= index+1;
            case(index)
                8: begin
                    serial_state <= PAUSE;
                    port <= 1;
                    index <= 0;
                end
                default: begin
                    port <= serial_buff[0];
                    serial_buff <= serial_buff>>1;
                end
            endcase
        end
        PAUSE: begin
            index <= index+1;
            if(index == 15) begin
                serial_state <= IDLE;
                done <= 1;
            end
        end
      endcase
  end

endmodule


module oFAST(
    input clk,
    input [5:0] mode,
    input [15:0] img_size_x,
    input [15:0] img_size_y,
    input [31:0] img_data,
    output logic [16:0] img_addr,

    input stream_ready,
    output logic stream_write,
    output logic [71:0] stream_data,
    output logic [3:0] stream_data_size,

    input [7:0] threshold,
    input [5:0] radius,
    input start,
    input reset,
    output logic done
);
    parameter IDLE = 6'b00_0000;
    parameter DONE = 6'b00_0001;

    //FOR DEBUGGING
    parameter FAST = 6'b01_0000;
    parameter FAST_SETUP = 6'b01_0001;
    parameter FAST_TESTING = 6'b01_0010;
    parameter FAST_AWAITING = 6'b01_0011;
    parameter FAST_SEND = 6'b01_0100;
    parameter HARRIS = 6'b10_0000;
    parameter HARRIS_SETUP = 6'b10_0001;
```

```
    parameter HARRIS_TESTING = 6'b10_0010;
    parameter HARRIS_SEND = 6'b10_0011;

    //MAIN STATES
    parameter OFAST = 6'b11_0000;
    parameter OFAST_FAST_SETUP = 6'b11_0001;
    parameter OFAST_AWAITING_FAST = 6'b11_0010;
    parameter OFAST_HARRIS_SETUP = 6'b11_0011;
    parameter OFAST_AWAITING_HARRIS = 6'b11_0100;
    parameter OFAST_AWAITING_STREAM = 6'b11_0101;
    parameter OFAST_SEND = 6'b11_0110;
    parameter OFAST_AWAITING_FILTER = 6'b11_0111;
    parameter OFAST_FILTER = 6'b11_1000;
    parameter OFAST_FILTER_SETUP = 6'b11_1001;
    parameter OFAST_DONE = 6'b11_1010;
    parameter OFAST_DONE_SETUP = 6'b11_1011;
    parameter OFAST_SEND_DONE = 6'b11_1100;

    parameter FAST_MEM_CONTROL = 2'b00;
    parameter HARRIS_MEM_CONTROL = 2'b01;

    logic [5:0] state = IDLE;
    assign sub_state = state;
    logic [5:0] ret_state;
    logic [1:0] mem_control = FAST_MEM_CONTROL;

    logic [31:0] fast_img_data;
    logic [16:0] fast_img_addr;
    logic fast_start = 0;
    logic fast_resume = 0;
    logic fast_debug = 0;
    logic fast_reset = 0;
    logic fast_feat_found;
    logic fast_done;
    logic [15:0] fast_feat_x;
    logic [15:0] fast_feat_y;
    FAST fast (.clk(clk),.img_size_x(img_size_x),.img_size_y(img_size_y),.img_data(img_data),.img_addr(fast_img_addr),
.threshold(threshold),.start(fast_start),.reset(fast_reset),.resume(fast_resume),.debug(fast_debug),.done(fast_done),
.feat_found(fast_feat_found),.feat_x(fast_feat_x),.feat_y(fast_feat_y));

    logic [15:0] hr_c_feat_x;
    logic [15:0] hr_c_feat_y;
    logic hr_c_start = 0;
    logic [16:0] hr_c_img_addr;
    logic hr_c_done;
    logic [31:0] h_resp;
    logic hr_c_ready;
    logic [4:0] centroid;
    harris_response_centroid hr_c
(.clk(clk),.centroid(centroid),.img_size_x(img_size_x),.img_size_y(img_size_y),.img_data(img_data),.img_addr(hr_c_img_addr),.start(hr_c_start
),.feat_x(hr_c_feat_x),.feat_y(hr_c_feat_y),.done(hr_c_done),.h_resp(h_resp),.ready(hr_c_ready));

    logic [68:0] filter_feat_data = 64'b0;
    logic filter_new_feat = 0;
    logic prev_new_filtered = 0;
    logic filter_print = 0;
    logic filter_resume = 0;
    logic filter_ready;
    logic filter_done;
    logic filter_new_filtered;
    logic [68:0] filter_filtered_feat;
    logic [15:0] filter_state = 0;
    logic [3:0] filter_curr_state;
```

```
    fast_filter filter
(.clk(clk),.feat_data(filter_feat_data),.new_feat(filter_new_feat),.print(filter_print),.resume(filter_resume),.ready(filter_ready),.done(filter_done),.
new_filtered(filter_new_filtered),.filtered_feat(filter_filtered_feat),.radius(radius));


    always_comb begin
        case(mem_control)
            FAST_MEM_CONTROL: img_addr = fast_img_addr;
            HARRIS_MEM_CONTROL: img_addr = hr_c_img_addr;
            default: img_addr = 0;
        endcase
    end

    logic [7:0] data_index = 0;
    always_ff @(posedge clk) begin
        case(state)
            IDLE: begin
                done <= 0;
                stream_write <= 0;
                if(start) begin
                    state <= mode;
                end
            end
            DONE: begin
                stream_write <= 0;
                if(reset) begin
                    done <= 0;
                    state <= IDLE;
                end else begin
                    done <= 1;
                end
            end
            FAST: begin
                fast_debug <= 1;
                mem_control <= FAST_MEM_CONTROL;
                state <= FAST_SETUP;
                fast_reset <= 1;
            end
            FAST_SETUP: begin
                state <= FAST_TESTING;
                fast_reset <= 0;
                fast_start <= 1;
            end
            FAST_TESTING: begin
                fast_start <= 0;
                fast_resume <= 0;
                stream_write <= 0;
                if(fast_feat_found) begin
                    state <= FAST_AWAITING;
                end else if(fast_done) begin
                    state <= DONE;
                end
            end
            FAST_AWAITING: begin
                if(stream_ready&(~stream_write)) begin
                    state <= FAST_SEND;
                end else begin
                    stream_write <= 0;
                end
            end
            FAST_SEND: begin
                stream_write <= 1;
                stream_data <= {40'b0,fast_feat_x,fast_feat_y};
                stream_data_size <= 4;
                fast_resume <= 1;
```

```
    state <= FAST_TESTING;
end

HARRIS: begin
    mem_control <= HARRIS_MEM_CONTROL;
    if(stream_ready) begin
        hr_c_start <= 1;
        hr_c_feat_x <= 10;
        hr_c_feat_y <= 10;
        state <= HARRIS_TESTING;
    end
end
HARRIS_TESTING: begin
    hr_c_start <= 0;
    if(hr_c_done) begin
        stream_write <= 1;
        stream_data <= {35'b0,centroid,h_resp};
        stream_data_size <= 5;
        state <= DONE;
    end
end

OFAST: begin
    fast_debug <= 0;
    mem_control <= FAST_MEM_CONTROL;
    state <= OFAST_FAST_SETUP;
    fast_reset <= 1;
    filter_print <= 0;
    filter_new_feat <= 0;
    filter_resume <= 0;
end
OFAST_FAST_SETUP: begin
    state <= OFAST_AWAITING_FAST;
    fast_reset <= 0;
    fast_start <= 1;
end
OFAST_AWAITING_FAST: begin
    fast_start <= 0;
    fast_resume <= 0;
    stream_write <= 0;
    if(fast_feat_found) begin
        state <= OFAST_HARRIS_SETUP;
        ret_state <= OFAST_AWAITING_FAST;
    end else if(fast_done) begin
        state <= OFAST_AWAITING_FILTER;
        filter_print <= 1;
        ret_state <= DONE;
    end
end
OFAST_HARRIS_SETUP: begin
    mem_control <= HARRIS_MEM_CONTROL;
    hr_c_start <= 1;
    hr_c_feat_x <= fast_feat_x;
    hr_c_feat_y <= fast_feat_y;
    state <= OFAST_AWAITING_HARRIS;
end
OFAST_AWAITING_HARRIS: begin
    hr_c_start <= 0;
    if(hr_c_done) begin
        state <= OFAST_FILTER_SETUP;
    end
end
OFAST_FILTER_SETUP: begin
    if(filter_ready) begin
        state <= OFAST_AWAITING_FILTER;
```

```
                filter_feat_data <= {centroid,h_resp,fast_feat_x,fast_feat_y};
                filter_new_feat <= 1;
            end
        end
        OFAST_AWAITING_FILTER: begin
            filter_new_feat <= 0;
            filter_resume <= 0;
            filter_print <= 0;
            stream_write <= 0;
            if((~filter_new_feat)&(~filter_resume)&(~filter_print)) begin
                if(filter_new_filtered) begin
                    state <= OFAST_AWAITING_STREAM;
                end else if(filter_ready|filter_done) begin
                    fast_resume <= 1;
                    state <= ret_state;
                    mem_control <= FAST_MEM_CONTROL;
                end
            end
        end
        OFAST_AWAITING_STREAM: begin
            if(stream_ready&(~stream_write)) begin
                state <= OFAST_SEND;
            end else begin
                stream_write <= 0;
            end
        end
        OFAST_SEND: begin
            stream_write <= 1;
            stream_data <= filter_filtered_feat;
            stream_data_size <= 9;
            state <= OFAST_AWAITING_FILTER;
            mem_control <= FAST_MEM_CONTROL;
            filter_resume <= 1;
        end



        default: state <= IDLE;
      endcase
   end
endmodule

module FAST(
   input clk,
   input [15:0] img_size_x,
   input [15:0] img_size_y,
   input [31:0] img_data,
   output logic [16:0] img_addr,

   input [7:0] threshold,
   input start,
   input reset,
   input resume,
   input debug,
   output logic detected,
   output logic done,
   output logic feat_found,
   output logic [15:0] feat_x,
   output logic [15:0] feat_y,

   input [2:0] test_index, //temp
   output logic [7:0] test_row [6:0]
);
   parameter IDLE = 4'b0000;
   parameter SCAN = 4'b0001;
```

```
parameter FAST_TEST = 4'b0010;
parameter INIT_LOAD = 4'b0011;
parameter INIT_LOAD_ROW = 4'b0100;
parameter QUICK_CHECK = 4'b0101;
parameter LONG_CHECK = 4'b0110;
parameter FOUND_FEAT = 4'b0111;
parameter DONE = 4'b1000;
parameter AWAITING = 4'b1001;
logic [3:0] state = IDLE;
logic [3:0] ret_state = IDLE;

parameter DEBUG_BOUND_X = 3;
parameter DEBUG_BOUND_Y = 3;
parameter BOUND_X = 10;
parameter BOUND_Y = 10;
logic [15:0] bound_x;
assign bound_x = debug ? DEBUG_BOUND_X : BOUND_X;
logic [15:0] bound_y;
assign bound_y = debug ? DEBUG_BOUND_Y : BOUND_Y;

logic [15:0] x_index;
logic [15:0] y_index;
logic [2:0] row_index = 0;
logic [31:0] row_center_addr;

logic [7:0] patch [6:0][6:0];
logic [7:0] buff_row_blocks [11:0];

logic [2:0] delay;
logic load_block_back;
logic load_block_front;



logic [7:0] center;
logic [7:0] add_center_threshold;
logic [7:0] sub_center_threshold;

logic [15:0] fast_circle_high;
logic [15:0] fast_circle_high_buff;
logic valid_high;
logic [15:0] fast_circle_low;
logic [15:0] fast_circle_low_buff;
logic valid_low;
logic [3:0] window_counter;
logic high_feat;
logic low_feat;

always_comb begin
    row_center_addr = img_size_x*((y_index+row_index)-3)+x_index;
    load_block_back = (row_center_addr[1:0] != 2'b11);
    load_block_front = (row_center_addr[1:0] != 2'b00);
    center = patch[3][3];

    test_row = patch[(test_index < 6) ? test_index : 6][6:0];

    valid_high = (center < 255 - threshold);
    valid_low = (center > threshold);
    add_center_threshold = center+threshold;
    sub_center_threshold = center-threshold;
    if(valid_low) begin
        fast_circle_low[0] = (patch[0][3] < sub_center_threshold);
        fast_circle_low[1] = (patch[0][4] < sub_center_threshold);
        fast_circle_low[2] = (patch[1][5] < sub_center_threshold);
        fast_circle_low[3] = (patch[2][6] < sub_center_threshold);
```

```
        fast_circle_low[4] = (patch[3][6] < sub_center_threshold);
        fast_circle_low[5] = (patch[4][6] < sub_center_threshold);
        fast_circle_low[6] = (patch[5][5] < sub_center_threshold);
        fast_circle_low[7] = (patch[6][4] < sub_center_threshold);
        fast_circle_low[8] = (patch[6][3] < sub_center_threshold);
        fast_circle_low[9] = (patch[6][2] < sub_center_threshold);
        fast_circle_low[10] =(patch[5][1] < sub_center_threshold);
        fast_circle_low[11] =(patch[4][0] < sub_center_threshold);
        fast_circle_low[12] =(patch[3][0] < sub_center_threshold);
        fast_circle_low[13] =(patch[2][0] < sub_center_threshold);
        fast_circle_low[14] =(patch[1][1] < sub_center_threshold);
        fast_circle_low[15] =(patch[0][2] < sub_center_threshold);
    end else begin
        fast_circle_low = 0;
    end
    if(valid_high) begin
        fast_circle_high[0] = (patch[0][3] > add_center_threshold);
        fast_circle_high[1] = (patch[0][4] > add_center_threshold);
        fast_circle_high[2] = (patch[1][5] > add_center_threshold);
        fast_circle_high[3] = (patch[2][6] > add_center_threshold);
        fast_circle_high[4] = (patch[3][6] > add_center_threshold);
        fast_circle_high[5] = (patch[4][6] > add_center_threshold);
        fast_circle_high[6] = (patch[5][5] > add_center_threshold);
        fast_circle_high[7] = (patch[6][4] > add_center_threshold);
        fast_circle_high[8] = (patch[6][3] > add_center_threshold);
        fast_circle_high[9] = (patch[6][2] > add_center_threshold);
        fast_circle_high[10] =(patch[5][1] > add_center_threshold);
        fast_circle_high[11] =(patch[4][0] > add_center_threshold);
        fast_circle_high[12] =(patch[3][0] > add_center_threshold);
        fast_circle_high[13] =(patch[2][0] > add_center_threshold);
        fast_circle_high[14] =(patch[1][1] > add_center_threshold);
        fast_circle_high[15] =(patch[0][2] > add_center_threshold);
    end else begin
        fast_circle_high = 0;
    end
end

always_ff @(posedge clk) begin
    if(reset) begin
        state <= IDLE;
        done <= 0;
        feat_found <= 0;
    end else begin
        case(state)
            IDLE: begin
                done <= 0;
                feat_found <= 0;
                if(start) begin
                    feat_found <= 0;
                    state <= FAST_TEST;
                    x_index <= bound_x;
                    y_index <= bound_y;
                end
            end
            SCAN: begin
                feat_found <= 0;
                if(x_index+bound_x+1 < img_size_x) begin
                    x_index <= x_index+1;
                    state <= FAST_TEST;
                end else if(y_index+bound_y+1 < img_size_y) begin
                    x_index <= bound_x;
                    y_index <= y_index+1;
                    state <= FAST_TEST;
                end else begin
                    state <= DONE;
```

```
    end
end
FAST_TEST: begin
    row_index <= 0;
    state <= INIT_LOAD;
    ret_state <= SCAN;
end
INIT_LOAD: begin
    delay <= 0;
    if(row_index != 7) begin
        state <= INIT_LOAD_ROW;
        img_addr <= 0;
    end else begin
        state <= QUICK_CHECK;
    end
end
INIT_LOAD_ROW: begin
    delay <= delay+1;
    case(delay)
        0: img_addr <= load_block_back ? (row_center_addr>>2)-1 : img_addr;
        1: img_addr <= row_center_addr>>2;
        2: img_addr <= load_block_front ? (row_center_addr>>2)+1: img_addr;
        4: begin
            if(load_block_back) begin
                buff_row_blocks[0] <= img_data[7:0];
                buff_row_blocks[1] <= img_data[15:8];
                buff_row_blocks[2] <= img_data[23:16];
                buff_row_blocks[3] <= img_data[31:24];
            end
        end
        5: begin
            buff_row_blocks[4] <= img_data[7:0];
            buff_row_blocks[5] <= img_data[15:8];
            buff_row_blocks[6] <= img_data[23:16];
            buff_row_blocks[7] <= img_data[31:24];
        end
        6: begin
            if(load_block_front) begin
                buff_row_blocks[8] <= img_data[7:0];
                buff_row_blocks[9] <= img_data[15:8];
                buff_row_blocks[10] <= img_data[23:16];
                buff_row_blocks[11] <= img_data[31:24];
            end
        end
        7: begin
            case(row_center_addr[1:0])
                0: patch[row_index] <= buff_row_blocks[7:1];
                1: patch[row_index] <= buff_row_blocks[8:2];
                2: patch[row_index] <= buff_row_blocks[9:3];
                3: patch[row_index] <= buff_row_blocks[10:4];
            endcase
            row_index <= row_index+1;
            state <= INIT_LOAD;
        end
    endcase
end
QUICK_CHECK: begin
    delay <= delay+1;
    if(delay == 3) begin
        if((fast_circle_low[0]+fast_circle_low[4]+fast_circle_low[8]+fast_circle_low[12] >= 3)|
          (fast_circle_high[0]+fast_circle_high[4]+fast_circle_high[8]+fast_circle_high[12] >= 3)) begin
            state <= LONG_CHECK;
            window_counter <= 0;
            fast_circle_high_buff <= fast_circle_high;
            fast_circle_low_buff <= fast_circle_low;
```

```
                end else begin
                    state <= ret_state;
                end
            end
        end
        LONG_CHECK: begin
            window_counter <= window_counter+1;
            fast_circle_high_buff <= {fast_circle_high_buff[0],fast_circle_high_buff[15:1]};
            fast_circle_low_buff <= {fast_circle_low_buff[0],fast_circle_low_buff[15:1]};
            if((fast_circle_high_buff[11:0] == 12'b1111_1111_1111)|(fast_circle_low_buff[11:0] == 12'b1111_1111_1111)) begin
                state <= FOUND_FEAT;
            end else if(window_counter == 15) begin
                state <= ret_state;
            end
        end
        FOUND_FEAT: begin
            feat_x <= x_index;
            feat_y <= y_index;
            feat_found <= 1;
            state <= AWAITING;
        end
        AWAITING: begin
            feat_found <= 0;
            if(resume) begin
                state <= ret_state;
            end
        end
        DONE: begin
            feat_found <= 0;
            done <= 1;
        end
      endcase
    end
  end
endmodule


module harris_response_centroid(
    input clk,
    input [15:0] img_size_x,
    input [15:0] img_size_y,
    input [31:0] img_data,
    output logic [16:0] img_addr,

    input start,
    input [15:0] feat_x,
    input [15:0] feat_y,

    output logic done,
    output logic ready,
    output logic [31:0] h_resp,
    output logic [4:0] centroid
);
    //For 21x21 image patches
    parameter IDLE = 3'b000;
    parameter LOAD_INIT = 3'b001;
    parameter LOAD_ROW = 3'b010;
    parameter LOAD_NEXT = 3'b011;
    parameter CALC = 3'b100;
    parameter DONE = 3'b101;
    logic [2:0] state = IDLE;
    logic [2:0] ret_state;
    logic [4:0] cycle;
    logic [4:0] calc_cycle;
```

```
logic [4:0] row_index = 0;
logic [1:0] prev_row;
logic [1:0] curr_row;
logic [1:0] next_row;

logic [31:0] row_center_addr;
logic [4:0] offset;

logic [7:0] patch_buff [2:0][20:0];
logic [7:0] row_buff [23:0] = '{0,0,0,0,0,0,0,
                                0,0,0,0,0,0,0,
                                0,0,0,0,0,0,0,
                                0,0,0};

wire signed [8:0] patch [2:0][20:0];
genvar i;
integer j;
generate
   for(i=0;i<21;i=i+1) begin
      assign patch[0][i] = {1'b0,patch_buff[0][i]};
      assign patch[1][i] = {1'b0,patch_buff[1][i]};
      assign patch[2][i] = {1'b0,patch_buff[2][i]};
   end
endgenerate

logic load_block_back;
logic load_block_front;

logic h_enable;

logic signed [15:0] h_buff_row_xx [18:0];
logic signed [15:0] h_buff_row_yy [18:0];
logic signed [15:0] h_buff_row_xy [18:0];

logic signed [20:0] h_sum_row_xx = 0;
logic signed [16:0] h_sum_row_xx_s1 [9:0] = '{0,0,0,0,0,0,0,0,0,0};
logic signed [17:0] h_sum_row_xx_s2 [4:0] = '{0,0,0,0,0};
logic signed [18:0] h_sum_row_xx_s3 [2:0] = '{0,0,0};
logic signed [19:0] h_sum_row_xx_s4 [1:0] = '{0,0};

logic signed [20:0] h_sum_row_yy = 0;
logic signed [16:0] h_sum_row_yy_s1 [9:0] = '{0,0,0,0,0,0,0,0,0,0};
logic signed [17:0] h_sum_row_yy_s2 [4:0] = '{0,0,0,0,0};
logic signed [18:0] h_sum_row_yy_s3 [2:0] = '{0,0,0};
logic signed [19:0] h_sum_row_yy_s4 [1:0] = '{0,0};

logic signed [20:0] h_sum_row_xy = 0;
logic signed [16:0] h_sum_row_xy_s1 [9:0] = '{0,0,0,0,0,0,0,0,0,0};
logic signed [17:0] h_sum_row_xy_s2 [4:0] = '{0,0,0,0,0};
logic signed [18:0] h_sum_row_xy_s3 [2:0] = '{0,0,0};
logic signed [19:0] h_sum_row_xy_s4 [1:0] = '{0,0};

logic signed [25:0] h_sum_xx;
logic signed [25:0] h_sum_yy;
logic signed [25:0] h_sum_xy;

logic signed [25:0] hr_var_xx;
logic signed [25:0] hr_var_yy;
logic signed [25:0] hr_var_xy;
logic signed [53:0] hr_var_tr2;
logic signed [52:0] hr_var_det;
logic signed [53:0] hr_var_resp;

always_comb begin
   load_block_back = (row_center_addr[1:0] < 2'b10);
```

```
    load_block_front = (row_center_addr[1:0] > 2'b01);
    row_center_addr = img_size_x*((feat_y+row_index)-10)+feat_x;
    hr_var_tr2 = (hr_var_xx+hr_var_yy)*(hr_var_xx+hr_var_yy);
    hr_var_det = (hr_var_xx*hr_var_yy-hr_var_xy*hr_var_xy);
    hr_var_resp = (hr_var_det-((hr_var_tr2>>>5)+(hr_var_tr2>>>6)));
end

logic [8:0] c_sum_row_s1 [10:0];
logic [9:0] c_sum_row_s2 [5:0];
logic [10:0] c_sum_row_s3 [2:0];
logic [11:0] c_sum_row_s4 [1:0];
logic [12:0] c_sum_row;

logic [12:0] c_buff_row_x [20:0];
logic [13:0] c_sum_row_x_s1 [10:0];
logic [14:0] c_sum_row_x_s2 [5:0];
logic [15:0] c_sum_row_x_s3 [2:0];
logic [16:0] c_sum_row_x_s4 [1:0];
logic [17:0] c_sum_row_x;

logic [12:0] c_buff_row_y [20:0];
logic [13:0] c_sum_row_y_s1 [10:0];
logic [14:0] c_sum_row_y_s2 [5:0];
logic [15:0] c_sum_row_y_s3 [2:0];
logic [16:0] c_sum_row_y_s4 [1:0];
logic [17:0] c_sum_row_y;

logic [22:0] c_sum_x;
logic [22:0] c_sum_y;
logic [17:0] c_sum;

logic [4:0] quo_x;
logic [17:0] rem_x;
logic [4:0] quo_y;
logic [17:0] rem_y;
logic [22:0] div_buff;
logic div_zero;

logic [4:0] angles [20:0][20:0] = (
'{'{4,  4,  5,  5,  5,  6,  6,  7,  7,  7,  8,  9,  9,  9, 10, 10, 11, 11, 11, 12, 12},
  '{4,  4,  4,  5,  5,  5,  6,  6,  7,  7,  8,  9,  9, 10, 10, 11, 11, 11, 12, 12, 12},
  '{3,  4,  4,  4,  5,  5,  6,  6,  7,  7,  8,  9,  9, 10, 10, 11, 11, 12, 12, 12, 13},
  '{3,  3,  4,  4,  4,  5,  5,  6,  7,  7,  8,  9,  9, 10, 11, 11, 12, 12, 12, 13, 13},
  '{3,  3,  3,  4,  4,  4,  5,  6,  6,  7,  8,  9, 10, 10, 11, 12, 12, 12, 13, 13, 13},
  '{2,  3,  3,  3,  4,  4,  5,  5,  6,  7,  8,  9, 10, 11, 11, 12, 12, 13, 13, 13, 14},
  '{2,  2,  2,  3,  3,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 13, 13, 14, 14, 14},
  '{1,  2,  2,  2,  2,  3,  3,  4,  5,  6,  8, 10, 11, 12, 13, 13, 14, 14, 14, 14, 15},
  '{1,  1,  1,  1,  2,  2,  2,  3,  4,  6,  8, 10, 12, 13, 14, 14, 14, 15, 15, 15, 15},
  '{1,  1,  1,  1,  1,  1,  1,  2,  2,  4,  8, 12, 14, 14, 15, 15, 15, 15, 15, 15, 15},
  '{0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16},
  '{31, 31, 31, 31, 31, 31, 31, 30, 30, 28, 24, 20, 18, 18, 17, 17, 17, 17, 17, 17, 17},
  '{31, 31, 31, 31, 30, 30, 30, 29, 28, 26, 24, 22, 20, 19, 18, 18, 18, 17, 17, 17, 17},
  '{31, 30, 30, 30, 30, 29, 29, 28, 27, 26, 24, 22, 21, 20, 19, 19, 18, 18, 18, 18, 17},
  '{30, 30, 30, 29, 29, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 19, 19, 18, 18, 18},
  '{30, 29, 29, 29, 28, 28, 27, 27, 26, 25, 24, 23, 22, 21, 21, 20, 20, 19, 19, 19, 18},
  '{29, 29, 29, 28, 28, 28, 27, 26, 26, 25, 24, 23, 22, 22, 21, 20, 20, 20, 19, 19, 19},
  '{29, 29, 28, 28, 28, 27, 27, 26, 25, 25, 24, 23, 23, 22, 21, 21, 20, 20, 20, 19, 19},
  '{29, 28, 28, 28, 27, 27, 26, 26, 25, 25, 24, 23, 23, 22, 22, 21, 21, 20, 20, 20, 19},
  '{28, 28, 28, 27, 27, 27, 26, 26, 25, 25, 24, 23, 23, 22, 22, 21, 21, 21, 20, 20, 20},
  '{28, 28, 27, 27, 27, 26, 26, 25, 25, 25, 24, 23, 23, 23, 22, 22, 21, 21, 21, 20, 20}}
  );

always_ff @(posedge clk) begin
    case(state)
        IDLE: begin
```

```
    done <= 0;
    if(start) begin
        h_sum_xx <= 0;
        h_sum_yy <= 0;
        h_sum_xy <= 0;
        c_sum <= 0;
        c_sum_x <= 0;
        c_sum_y <= 0;
        state <= LOAD_INIT;
        row_index <= 0;
        ready <= 0;
    end else begin
        ready <= 1;
    end
end
LOAD_INIT: begin
    cycle <= 0;
    offset <= 2;
    img_addr <= 0;
    calc_cycle <= 0;
    case(row_index)
        0: begin
            next_row <= row_index;
            state <= LOAD_ROW;
            ret_state <= CALC;
            h_enable <= 0;
        end
        1: begin
            next_row <= row_index;
            state <= LOAD_ROW;
            ret_state <= CALC;
            h_enable <= 0;
        end
        2: begin
            next_row <= row_index;
            curr_row <= 1;
            prev_row <= 0;
            state <= LOAD_ROW;
            ret_state <= CALC;
            h_enable <= 1;
        end
        default: begin
            if(row_index < 21) begin
                next_row <= prev_row;
                curr_row <= next_row;
                prev_row <= curr_row;
                state <= LOAD_ROW;
                ret_state <= CALC;
                h_enable <= 1;
            end else begin
                hr_var_xx <= h_sum_xx;
                hr_var_yy <= h_sum_yy;
                hr_var_xy <= h_sum_xy;
                state <= DONE;
            end
        end
    endcase
end
LOAD_ROW: begin
    cycle <= cycle+1;
    case(cycle)
        0: img_addr <= load_block_back ? (row_center_addr>>2)-3 : img_addr;
        1: img_addr <= (row_center_addr>>2)-2;
        2: img_addr <= (row_center_addr>>2)-1;
        3: img_addr <= (row_center_addr>>2);
```

```
        4: begin
            img_addr <= (row_center_addr>>2)+1;
            if(load_block_back) begin
                row_buff[0] <= img_data[23:16];
                row_buff[1] <= img_data[31:24];
            end
        end
        5: img_addr <= (row_center_addr>>2)+2;
        6: img_addr <= load_block_front ? (row_center_addr>>2)+3 : img_addr;
        10: begin
            if(load_block_front) begin
                row_buff[22] <= img_data[7:0];
                row_buff[23] <= img_data[15:8];
            end
        end
        11: begin
            case(row_center_addr[1:0])
                0: patch_buff[next_row] <= row_buff[20:0];
                1: patch_buff[next_row] <= row_buff[21:1];
                2: patch_buff[next_row] <= row_buff[22:2];
                3: patch_buff[next_row] <= row_buff[23:3];
            endcase
            state <= ret_state;
            row_index <= row_index+1;
        end
    endcase
    //3 cycle latency, 2 from mem, 1 from img_addr
    if((cycle > 4) & (cycle < 10)) begin
        row_buff[offset] <= img_data[7:0];
        row_buff[offset+1] <= img_data[15:8];
        row_buff[offset+2] <= img_data[23:16];
        row_buff[offset+3] <= img_data[31:24];
        offset <= offset+4;
    end
end
CALC: begin
    calc_cycle <= calc_cycle+1;

    case(calc_cycle)
        0: begin
            if(h_enable) begin
                for(j=0;j<19;j=j+1) begin
                    h_buff_row_xx[j] <= (patch[curr_row][j+2]-patch[curr_row][j])>>>1;
                    h_buff_row_yy[j] <= (patch[next_row][j+1]-patch[prev_row][j+1])>>>1;
                end
            end
            for(j=0;j<21;j=j+1) begin
                c_buff_row_y[j] = (row_index-1)*patch_buff[next_row][j];
                c_buff_row_x[j] = j*patch_buff[next_row][j];
            end

            for(j=0;j<10;j=j+1) begin
                c_sum_row_s1[j] <= patch_buff[next_row][2*j]+patch_buff[next_row][2*j+1];
            end
            c_sum_row_s1[10] <= patch_buff[next_row][20];
        end
        1: begin
            if(h_enable) begin
                for(j=0;j<19;j=j+1) begin
                    h_buff_row_xx[j] <= h_buff_row_xx[j]*h_buff_row_xx[j];
                    h_buff_row_yy[j] <= h_buff_row_yy[j]*h_buff_row_yy[j];
                    h_buff_row_xy[j] <= h_buff_row_xx[j]*h_buff_row_yy[j];
                end
            end
```

```
    for(j=0;j<5;j=j+1) begin
        c_sum_row_s2[j] <= c_sum_row_s1[2*j]+c_sum_row_s1[2*j+1];
    end
    c_sum_row_s2[5] <= c_sum_row_s1[10];

    for(j=0;j<10;j=j+1) begin
        c_sum_row_y_s1[j] <= c_buff_row_y[2*j]+c_buff_row_y[2*j+1];
        c_sum_row_x_s1[j] <= c_buff_row_x[2*j]+c_buff_row_x[2*j+1];
    end
    c_sum_row_y_s1[10] <= c_buff_row_y[20];
    c_sum_row_x_s1[10] <= c_buff_row_x[20];
end
2: begin
    if(h_enable) begin
        for(j=0;j<9;j=j+1) begin
            h_sum_row_xx_s1[j] <= h_buff_row_xx[2*j]+h_buff_row_xx[2*j+1];
            h_sum_row_yy_s1[j] <= h_buff_row_yy[2*j]+h_buff_row_yy[2*j+1];
            h_sum_row_xy_s1[j] <= h_buff_row_xy[2*j]+h_buff_row_xy[2*j+1];
        end
        h_sum_row_xx_s1[9] <= h_buff_row_xx[18];
        h_sum_row_yy_s1[9] <= h_buff_row_yy[18];
        h_sum_row_xy_s1[9] <= h_buff_row_xy[18];
    end

    for(j=0;j<3;j=j+1) begin
        c_sum_row_s3[j] <= c_sum_row_s2[2*j]+c_sum_row_s2[2*j+1];
    end

    for(j=0;j<5;j=j+1) begin
        c_sum_row_y_s2[j] <= c_sum_row_y_s1[2*j]+c_sum_row_y_s1[2*j+1];
        c_sum_row_x_s2[j] <= c_sum_row_x_s1[2*j]+c_sum_row_x_s1[2*j+1];
    end
    c_sum_row_y_s2[5] <= c_sum_row_y_s1[10];
    c_sum_row_x_s2[5] <= c_sum_row_x_s1[10];
end
3: begin
    if(h_enable) begin
        for(j=0;j<5;j=j+1) begin
            h_sum_row_xx_s2[j] <= h_sum_row_xx_s1[2*j]+h_sum_row_xx_s1[2*j+1];
            h_sum_row_yy_s2[j] <= h_sum_row_yy_s1[2*j]+h_sum_row_yy_s1[2*j+1];
            h_sum_row_xy_s2[j] <= h_sum_row_xy_s1[2*j]+h_sum_row_xy_s1[2*j+1];
        end
    end
    c_sum_row_s4[0] <= c_sum_row_s3[0]+c_sum_row_s3[1];
    c_sum_row_s4[1] <= c_sum_row_s3[2];

    for(j=0;j<3;j=j+1) begin
        c_sum_row_y_s3[j] <= c_sum_row_y_s2[2*j]+c_sum_row_y_s2[2*j+1];
        c_sum_row_x_s3[j] <= c_sum_row_x_s2[2*j]+c_sum_row_x_s2[2*j+1];
    end
end
4: begin
    if(h_enable) begin
        for(j=0;j<2;j=j+1) begin
            h_sum_row_xx_s3[j] <= h_sum_row_xx_s2[2*j]+h_sum_row_xx_s2[2*j+1];
            h_sum_row_yy_s3[j] <= h_sum_row_yy_s2[2*j]+h_sum_row_yy_s2[2*j+1];
            h_sum_row_xy_s3[j] <= h_sum_row_xy_s2[2*j]+h_sum_row_xy_s2[2*j+1];
        end
        h_sum_row_xx_s3[2] <= h_sum_row_xx_s2[4];
        h_sum_row_yy_s3[2] <= h_sum_row_yy_s2[4];
        h_sum_row_xy_s3[2] <= h_sum_row_xy_s2[4];
    end
    c_sum_row <= c_sum_row_s4[0]+c_sum_row_s4[1];

    c_sum_row_y_s4[0] <= c_sum_row_y_s3[0]+c_sum_row_y_s3[1];
```

```
            c_sum_row_y_s4[1] <= c_sum_row_y_s3[2];

            c_sum_row_x_s4[0] <= c_sum_row_x_s3[0]+c_sum_row_x_s3[1];
            c_sum_row_x_s4[1] <= c_sum_row_x_s3[2];
        end

        5: begin
            if(h_enable) begin
                h_sum_row_xx_s4[0] <= h_sum_row_xx_s3[0]+h_sum_row_xx_s3[1];
                h_sum_row_xx_s4[1] <= h_sum_row_xx_s3[2];
                h_sum_row_yy_s4[0] <= h_sum_row_yy_s3[0]+h_sum_row_yy_s3[1];
                h_sum_row_yy_s4[1] <= h_sum_row_yy_s3[2];
                h_sum_row_xy_s4[0] <= h_sum_row_xy_s3[0]+h_sum_row_xy_s3[1];
                h_sum_row_xy_s4[1] <= h_sum_row_xy_s3[2];
            end
            c_sum <= c_sum+c_sum_row;

            c_sum_row_y <= c_sum_row_y_s4[0]+c_sum_row_y_s4[1];
            c_sum_row_x <= c_sum_row_x_s4[0]+c_sum_row_x_s4[1];
        end
        6: begin
            if(h_enable) begin
                h_sum_row_xx <= h_sum_row_xx_s4[0]+h_sum_row_xx_s4[1];
                h_sum_row_yy <= h_sum_row_yy_s4[0]+h_sum_row_yy_s4[1];
                h_sum_row_xy <= h_sum_row_xy_s4[0]+h_sum_row_xy_s4[1];
            end else begin
                state <= LOAD_INIT;
            end
            c_sum_y <= c_sum_y+c_sum_row_y;
            c_sum_x <= c_sum_x+c_sum_row_x;
        end
        7: begin
            h_sum_xx <= h_sum_xx+h_sum_row_xx;
            h_sum_yy <= h_sum_yy+h_sum_row_yy;
            h_sum_xy <= h_sum_xy+h_sum_row_xy;
            state <= LOAD_INIT;
        end
    endcase
end
DONE: begin
    cycle <= cycle+1;
    case(cycle)
        0: begin
            div_buff <= c_sum<<4;
            quo_y <= 0;
            quo_x <= 0;
            div_zero <= (c_sum==0);
        end
        6: begin
            if((c_sum_y<<1) != c_sum) begin
                quo_y <= quo_y + ((c_sum_y<<1) >= c_sum);
            end else if(quo_y[0] == 1'b1) begin
                quo_y <= quo_y+1;
            end
            if((c_sum_x<<1) != c_sum) begin
                quo_x <= quo_x + ((c_sum_x<<1) >= c_sum);
            end else if(quo_x[0] == 1'b1) begin
                quo_x <= quo_x+1;
            end
        end
        7: begin
            h_resp <= hr_var_resp[53] ? 0 : hr_var_resp[52:21];
            centroid <= angles[quo_y][quo_x];
            done <= 1;
            state <= IDLE;
```

```
                end
                default: begin
                    if(~div_zero) begin
                        if(c_sum_y >= div_buff) begin
                            c_sum_y <= c_sum_y-div_buff;
                            quo_y <= {quo_y[3:0],1'b1};
                        end else begin
                            quo_y <= {quo_y[3:0],1'b0};
                        end
                        if(c_sum_x >= div_buff) begin
                            c_sum_x <= c_sum_x-div_buff;
                            quo_x <= {quo_x[3:0],1'b1};
                        end else begin
                            quo_x <= {quo_x[3:0],1'b0};
                        end
                        div_buff <= div_buff>>1;
                    end else begin
                        quo_x <= 10;
                        quo_y <= 10;
                    end
                end
            endcase
        end
    endcase
  end
endmodule

module fast_filter(
  input clk,
  input [68:0] feat_data,
  input new_feat,
  input print,
  input resume,
  output logic ready,
  output logic new_filtered,
  output logic [68:0] filtered_feat,
  output logic done,
  input logic [5:0] radius,

  output logic [69:0] debug_out,
  output logic [3:0] debug_curr_state,
  output logic [9:0] debug_curr_index,
  output logic [9:0] debug_stack
);
  parameter IDLE = 4'b0000;
  parameter INIT = 4'b0001;
  parameter AWAITING = 4'b0010;
  parameter SCAN = 4'b0011;
  parameter COMPARE = 4'b0100;
  parameter REMOVE = 4'b0101;
  parameter ADD = 4'b0110;
  parameter SEND = 4'b0111;
  parameter PRINT = 4'b1000;
  parameter PAUSE = 4'b1001;
  parameter CLEAN = 4'b1010;

  logic [80:0] curr_feat_in = 81'b0;
  logic [80:0] curr_feat_out;
  logic curr_feat_write = 1'b0;
  logic [80:0] var_feat_in = 81'b0;
  logic [80:0] var_feat_out;
  logic var_feat_write = 1'b0;
  logic [11:0] var_feat_addr = 0;
  logic [11:0] stack_out;
  logic [11:0] stack_in = 0;
```

```
logic stack_write = 0;

logic [68:0] new_feat_data;

logic [11:0] curr_index = 0;
logic [11:0] prev_index;
logic [11:0] stack_index = 0;

logic [31:0] curr_harris;
logic [15:0] curr_feat_x;
logic [15:0] curr_feat_y;
logic [4:0] curr_centroid;
logic [11:0] next_list_addr;

filter_feat_mem feat_mem (.clka(clk),.clkb(clk),
                    .addra(curr_index),.dina(curr_feat_in),.douta(curr_feat_out),.wea(curr_feat_write),
                    .addrb(var_feat_addr),.dinb(var_feat_in),.doutb(var_feat_out),.web(var_feat_write));

filter_stack_mem stack_mem(.clka(clk),.addra(stack_index),.dina(stack_in),.douta(stack_out),.wea(stack_write));

assign curr_centroid = curr_feat_out[80:76];
assign curr_harris = curr_feat_out[75:44];
assign curr_feat_y = curr_feat_out[27:12];
assign curr_feat_x = curr_feat_out[43:28];
assign next_list_addr = curr_feat_out[11:0];
logic [15:0] dist_x;
logic [15:0] dist_y;

assign dist_y = new_feat_data[15:0]>curr_feat_y ? new_feat_data[15:0]-curr_feat_y : curr_feat_y-new_feat_data[15:0];
assign dist_x = new_feat_data[31:16]>curr_feat_x ? new_feat_data[31:16]-curr_feat_x : curr_feat_x-new_feat_data[31:16];

logic [3:0] state = INIT;
logic [3:0] ret_state = INIT;
logic [3:0] cycle;
logic [31:0] valid_buff [127:0] = '{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

logic [31:0] harris_bound1;
logic [31:0] harris_bound2;
assign harris_bound1 = new_feat_data[63:32]+(new_feat_data[63:32]>>3);
assign harris_bound2 = curr_harris+(curr_harris>>3);
logic harris_compare1;
logic harris_compare2;
assign harris_compare1 = harris_bound1 >= curr_harris;
assign harris_compare2 = harris_bound2 >= new_feat_data[63:32];

logic [15:0] dist_tot = 0;
logic valid;


assign debug_stack = stack_index;
assign debug_curr_state = state;
assign debug_curr_index = curr_index;
assign debug_out = curr_feat_out;

always_ff @(posedge clk) begin
   case(state)
      INIT: begin
         stack_index <= 0;
```

```
    stack_write <= 0;
    curr_index <= 0;
    prev_index <= 0;
    done <= 0;
    curr_feat_write <= 1;
    curr_feat_in <= {5'b0,32'b0,32'b0,12'b0};
    new_filtered <= 0;
    state <= AWAITING;
    ready <= 0;
    cycle <= 0;
  end
  AWAITING: begin
    case(cycle)
      0: begin
        curr_feat_write <= 0;
        var_feat_write <= 0;
        stack_write <= 0;
        curr_index <= 0;
        new_filtered <= 0;
        cycle <= cycle+1;
        ready <= 0;
      end
      3: begin
        if(new_feat) begin
          new_feat_data <= feat_data;
          valid <= 1;
          ready <= 0;
          state <= SCAN;
          cycle <= 0;
        end else if(print) begin
          state <= PRINT;
          ready <= 0;
          cycle <= 0;
        end else begin
          ready <= 1;
        end
      end
      default: cycle <= cycle+1;
    endcase
  end
  SCAN: begin
    curr_feat_write <= 0;
    var_feat_write <= 0;
    if(next_list_addr != 0) begin
      curr_index <= next_list_addr;
      prev_index <= curr_index;
      state <= COMPARE;
    end else begin
      state <= ADD;
    end
    cycle <= 0;
  end
  COMPARE: begin
    case(cycle)
      3: begin
        if(dist_y > radius) begin
          state <= valid_buff[curr_index[11:5]][curr_index[4:0]] ? SEND : REMOVE;
          ret_state <= REMOVE;
          cycle <= 0;
        end else if(dist_x > radius) begin
          state <= SCAN;
          cycle <= 0;
        end else begin
          dist_tot <= dist_x[5:0]*dist_x[5:0]+dist_y[5:0]*dist_y[5:0];
          cycle <= cycle+1;
```

```
            end
        end
        4: begin
            state <= SCAN;
            cycle <= 0;
            if(radius*radius >= dist_tot) begin
                if(harris_compare1) begin
                    valid_buff[curr_index[11:5]][curr_index[4:0]] <= 1'b0;
                end
                if(harris_compare2) begin
                    valid <= 0;
                end
            end
        end
        default: cycle <= cycle+1;
    endcase
end
ADD: begin
    case(cycle)
        4: begin
            curr_feat_in <= {curr_feat_out[80:12],stack_out};
            curr_feat_write <= 1;
            var_feat_addr <= stack_out;
            var_feat_in <= {new_feat_data,12'b0};
            var_feat_write <= 1;
            valid_buff[stack_out[11:5]][stack_out[4:0]] <= valid;
            stack_index <= stack_index+1;
            state <= AWAITING;
            cycle <= 0;
        end
        default: cycle <= cycle+1;
    endcase
end
REMOVE: begin
    case(cycle)
        0: begin
            var_feat_addr <= prev_index;
            cycle <= cycle+1;
        end
        3: begin
            var_feat_in <= {var_feat_out[80:12],next_list_addr};
            stack_index <= stack_index-1;
            stack_in <= curr_index;
            stack_write <= 1;
            curr_index <= prev_index;
            var_feat_write <= 1;
            cycle <= cycle+1;
        end
        4: begin
            stack_write <= 0;
            var_feat_write <= 0;
            cycle <= cycle+1;
        end
        6: begin
            state <= SCAN;
            cycle <= 0;
        end
        default: cycle <= cycle+1;
    endcase
end
SEND: begin
    case(cycle)
        3: begin
            filtered_feat <= curr_feat_out[80:12];
            state <= PAUSE;
```

```
                cycle <= 0;
            end
            default: cycle <= cycle+1;
        endcase
    end
    PAUSE: begin
        if(resume) begin
            new_filtered <= 0;
            state <= ret_state;
            cycle <= 0;
        end else begin
            new_filtered <= 1;
        end
    end
    PRINT: begin
        case(cycle)
            5: begin
                if(next_list_addr != 0) begin
                    curr_index <= next_list_addr;
                    prev_index <= curr_index;
                    state <= valid_buff[next_list_addr[11:5]][next_list_addr[4:0]] ? SEND : PRINT;
                    ret_state <= PRINT;
                    cycle <= 0;
                end else begin
                    done <= 1;
                    state <= CLEAN;
                    stack_write <= 1;
                    stack_index <= 0;
                    stack_in <= 1;
                    cycle <= 0;
                end
            end
            default: cycle <= cycle+1;
        endcase
    end
    CLEAN: begin
        if(stack_index < 4095) begin
            stack_index <= stack_index+1;
            stack_in <= stack_in+1;
        end else begin
            stack_write <= 0;
            stack_index <= 0;
            state <= INIT;
        end
    end
  endcase
 end
endmodule
```