Chat : https://chatgpt.com/share/2b9c348e-2f54-4a73-9e9d-6b29cebea810

You are a senior chip design engineer who is an expert with Verilog HDL, designing AI accelerators for SoCs. I want to join efabless'es 4 AI contest. The challenge aims to leverage generative AI to develop an open-source hardware accelerator designed explicitly for Keyword Spotting (KWS) applications on the Caravel System-on-Chip. I used the "Integer-Only Approximated MFCC for Ultra-Low Power Audio NN Processing on Multi-Core MCUs". To create the mfcc. The KWS uses the chose the CNN model that has been widely used for KWS, the CNN-KWS model, also known as the "Hello Edge" model. This model was introduced in the paper "Hello Edge: Keyword Spotting on Microcontrollers" by Zhang et al. You will help me debug the KWS machine learning and the audio features extractor and create an energy-efficient KWS accelerator that seamlessly integrates into the Caravel SoC environment. You will also help me add feature to the codebase. My top module is cnn_kws_accel "module cnn_kws_accel (

```
   input wire clk,
   input wire rst_n,
   input wire start,
   output wire done,
   // PSRAM signals
   output wire psram_sck,
   output wire psram_ce_n,
   inout wire [3:0] psram_d,
   output wire [3:0] psram_douten
);

   // Internal signals for PSRAM
   wire conv1_psram_sck, conv2_psram_sck, fc1_psram_sck, fc2_psram_sck, maxpool_psram_sck, softmax_psram_sck;
   wire conv1_psram_ce_n, conv2_psram_ce_n, fc1_psram_ce_n, fc2_psram_ce_n, maxpool_psram_ce_n, softmax_psram_ce_n;
   wire [3:0] conv1_psram_douten, conv2_psram_douten, fc1_psram_douten, fc2_psram_douten, maxpool_psram_douten, softmax_psram_douten;
   wire [3:0] conv1_psram_d, conv2_psram_d, fc1_psram_d, fc2_psram_d, maxpool_psram_d, softmax_psram_d;

   // PSRAM data output
```

```
wire [3:0] psram_d_in;

// State definitions
typedef enum logic [2:0] {
   IDLE,
   CONV1,
   FC1,
   MAXPOOL,
   SOFTMAX
} state_t;

state_t state, next_state;

// State machine
always @(posedge clk or negedge rst_n) begin
   if (!rst_n)
      state <= IDLE;
   else
      state <= next_state;
end

always @* begin
   next_state = state;
   case (state)
      IDLE: if (start) next_state = CONV1;
      CONV1: if (conv1_done) next_state = FC1;
      FC1: if (fc1_done) next_state = MAXPOOL;
      MAXPOOL: if (maxpool_done) next_state = SOFTMAX;
      SOFTMAX: if (softmax_done) next_state = IDLE;
      default: next_state = IDLE;
   endcase
```

```
end

// Multiplexer for PSRAM control signals
assign psram_sck = (state == CONV1) ? conv1_psram_sck :
          (state == FC1) ? fc1_psram_sck :
          (state == MAXPOOL) ? maxpool_psram_sck :
          softmax_psram_sck;

assign psram_ce_n = (state == CONV1) ? conv1_psram_ce_n :
           (state == FC1) ? fc1_psram_ce_n :
           (state == MAXPOOL) ? maxpool_psram_ce_n :
           softmax_psram_ce_n;

assign psram_douten = (state == CONV1) ? conv1_psram_douten :
            (state == FC1) ? fc1_psram_douten :
            (state == MAXPOOL) ? maxpool_psram_douten :
            softmax_psram_douten;

assign psram_d = (state == CONV1) ? conv1_psram_d :
         (state == FC1) ? fc1_psram_d :
         (state == MAXPOOL) ? maxpool_psram_d :
         softmax_psram_d;

// Instantiate the convolution and fully connected modules with PSRAM
wire [23:0] conv1_weight_base_addr = 24'h000000;
wire [23:0] conv1_bias_base_addr = 24'h000100;
wire [23:0] fc1_weight_base_addr = 24'h000200;
wire [23:0] fc1_bias_base_addr = 24'h000300;
wire [23:0] maxpool_input_addr = 24'h000400;
wire [23:0] maxpool_output_addr = 24'h000500;
wire [23:0] softmax_input_addr = 24'h000600;
wire [23:0] softmax_output_addr = 24'h000700;

wire conv1_done, fc1_done, maxpool_done, softmax_done;
wire conv1_data_valid = (state == CONV1);
wire fc1_data_valid = (state == FC1);
wire maxpool_data_valid = (state == MAXPOOL);
wire softmax_start = (state == SOFTMAX);

wire [320*16-1:0] conv1_data_out;
wire conv1_data_out_valid;
wire [64*16-1:0] fc1_data_out;
wire fc1_data_out_valid;
wire [10*16-1:0] maxpool_data_out;
wire maxpool_data_out_valid;
wire [10*8-1:0] softmax_data_out;
wire softmax_data_out_valid;

conv2d_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(1),
    .KERNEL_SIZE(3),
    .NUM_FILTERS(8),
    .PADDING(1),
    .ACTIV_BITS(16)
) conv1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(/* Provide data input */),
    .data_valid(conv1_data_valid),
    .data_out(conv1_data_out),
    .data_out_valid(conv1_data_out_valid),
    .psram_sck(conv1_psram_sck),
    .psram_ce_n(conv1_psram_ce_n),
    .psram_d(psram_d_in),
    .psram_douten(conv1_psram_douten),
    .weight_base_addr(conv1_weight_base_addr),
    .bias_base_addr(conv1_bias_base_addr)
);

fully_connected_psram #(
    .INPUT_SIZE(320),
    .OUTPUT_SIZE(64),
    .ACTIV_BITS(16)
) fc1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv1_data_out),
    .data_valid(conv1_data_out_valid),
    .data_out(fc1_data_out),
    .data_out_valid(fc1_data_out_valid),
    .psram_sck(fc1_psram_sck),
    .psram_ce_n(fc1_psram_ce_n),
    .psram_d(psram_d_in),
    .psram_douten(fc1_psram_douten),
    .weight_base_addr(fc1_weight_base_addr),
    .bias_base_addr(fc1_bias_base_addr)
);

maxpool_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(8),
    .KERNEL_SIZE(2),
    .STRIDE(2),
    .ACTIV_BITS(16),
    .ADDR_WIDTH(24)
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .start(maxpool_data_valid),
    .input_addr(maxpool_input_addr),
    .output_addr(maxpool_output_addr),
    .done(maxpool_done),
    .psram_sck(maxpool_psram_sck),
    .psram_ce_n(maxpool_psram_ce_n),
    .psram_d(psram_d_in),
    .psram_douten(maxpool_psram_douten)
);

softmax_psram #(
    .INPUT_SIZE(10),
    .ACTIV_BITS(8),
    .ADDR_WIDTH(24)
) softmax (
    .clk(clk),
    .rst_n(rst_n),
    .start(softmax_start),
    .input_addr(softmax_input_addr),
    .output_addr(softmax_output_addr),
    .size(3'b010),
    .cmd(8'hEB),
    .rd_wr(1'b1),
        .qspi(1'b0),
        .qpi(1'b0),
        .short_cmd(1'b0),
        .done(softmax_done),
        .psram_sck(softmax_psram_sck),
        .psram_ce_n(softmax_psram_ce_n),
        .psram_d(psram_d_in),
        .psram_douten(softmax_psram_douten)
    );

    // Tristate buffer for psram_d
    assign psram_d = psram_douten ? 4'bz : psram_d_in;

endmodule". Submodules are "module conv2d_psram #(
    parameter INPUT_WIDTH = 40, // MFCC features
    parameter INPUT_HEIGHT = 1, // Single feature height
    parameter INPUT_CHANNELS = 1, // Single channel input
    parameter KERNEL_SIZE = 3,
    parameter NUM_FILTERS = 8,
    parameter PADDING = 1,
    parameter ACTIV_BITS = 16
) (
    input wire clk,
    input wire rst_n,
    input wire [INPUT_WIDTH * INPUT_HEIGHT * INPUT_CHANNELS * ACTIV_BITS-1:0] data_in,
    input wire data_valid,
    output wire [INPUT_WIDTH * INPUT_HEIGHT * NUM_FILTERS * ACTIV_BITS-1:0] data_out,
    output wire data_out_valid,

    // PSRAM interface signals
    output wire psram_sck,
    output wire psram_ce_n,
    inout wire [3:0] psram_d,
    output wire [3:0] psram_douten,

    // Base addresses for weights and biases
    input wire [23:0] weight_base_addr,
    input wire [23:0] bias_base_addr
);

    // State definitions
    localparam IDLE = 0,
               LOAD_WEIGHTS = 1,
               LOAD_BIASES = 2,
               CONV = 3,
               STORE_RESULT = 4,
               DONE = 5;

    reg [3:0] state, next_state;
    reg [23:0] addr;
    reg [31:0] psram_data;
    reg psram_start, psram_rd_wr;
    reg [2:0] psram_size;
    reg psram_qspi, psram_qpi, psram_short_cmd;
    wire psram_done;

    wire [31:0] psram_data_out;

    // Instantiate PSRAM controller
    EF_PSRAM_CTRL_V2 psram_ctrl (
        .clk(clk),
        .rst_n(rst_n),
        .addr(addr),
        .data_i(psram_data),
        .data_o(psram_data_out),
        .size(psram_size),
        .start(psram_start),
        .done(psram_done),
        .wait_states(4'd8), // Adjust wait states as needed
        .cmd(8'hEB), // Example command
        .rd_wr(psram_rd_wr),
        .qspi(psram_qspi),
        .qpi(psram_qpi),
        .short_cmd(psram_short_cmd),
        .sck(psram_sck),
        .ce_n(psram_ce_n),
        .din(psram_d),
        .dout(psram_d),
        .douten(psram_douten)
    );

    reg [ACTIV_BITS-1:0] weights [0:NUM_FILTERS-1][0:INPUT_CHANNELS-1][0:KERNEL_SIZE-1][0:KERNEL_SIZE-1];
    reg [ACTIV_BITS-1:0] biases [0:NUM_FILTERS-1];
    reg [ACTIV_BITS-1:0] conv_result [0:INPUT_HEIGHT-1][0:INPUT_WIDTH-1][0:NUM_FILTERS-1];

    integer i, j, k, l, m, n, p, q;

    // State machine
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            state <= IDLE;
        else
            state <= next_state;
    end

    always @(*) begin
        next_state = state;
        case (state)
            IDLE: if (data_valid) next_state = LOAD_WEIGHTS;
            LOAD_WEIGHTS: if (psram_done) next_state = LOAD_BIASES;
            LOAD_BIASES: if (psram_done) next_state = CONV;
            CONV: next_state = STORE_RESULT;
            STORE_RESULT: if (psram_done) next_state = DONE;
            DONE: next_state = IDLE;
        endcase
    end

    // Control logic for PSRAM operations
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            addr <= 24'b0;
            psram_data <= 32'b0;
            psram_start <= 0;
            psram_rd_wr <= 0;
            psram_size <= 3'b010; // 4 bytes
            psram_qspi <= 0;
            psram_qpi <= 0;
            psram_short_cmd <= 0;
        end else begin
            psram_start <= 0;
            case (state)
                LOAD_WEIGHTS: begin
                    addr <= weight_base_addr;
                psram_rd_wr <= 1;
                psram_start <= 1;
            end
            LOAD_BIASES: begin
                addr <= bias_base_addr;
                psram_rd_wr <= 1;
                psram_start <= 1;
            end
            STORE_RESULT: begin
                addr <= 24'hXXXXXX; // Address to store results in PSRAM
                psram_data <= conv_result[m][n][p];
                psram_rd_wr <= 0;
                psram_start <= 1;
            end
          endcase
        end
    end

    // Convolution operation
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            for (i = 0; i < INPUT_HEIGHT; i = i + 1)
                for (j = 0; j < INPUT_WIDTH; j = j + 1)
                    for (k = 0; k < NUM_FILTERS; k = k + 1)
                        conv_result[i][j][k] <= 0;
        end else if (state == CONV) begin
            for (m = 0; m < INPUT_HEIGHT; m = m + 1) begin
                for (n = 0; n < INPUT_WIDTH; n = n + 1) begin
                    for (p = 0; p < NUM_FILTERS; p = p + 1) begin
                        conv_result[m][n][p] = biases[p];
                        for (q = 0; q < INPUT_CHANNELS; q = q + 1) begin
```

```
                            for (i = 0; i < KERNEL_SIZE; i = i + 1) begin
                                for (j = 0; j < KERNEL_SIZE; j = j + 1) begin
                                    if ((m + i) < INPUT_HEIGHT && (n + j) < INPUT_WIDTH) begin
                                        conv_result[m][n][p] = conv_result[m][n][p] + weights[p][q][i][j] * data_in[((m + i) * INPUT_WIDTH * INPUT_CHANNELS + (n + j) * INPUT_CHANNELS + q) * ACTIV_BITS +: ACTIV_BITS];
                                    end
                                end
                            end
                        end
                        // Apply ReLU activation
                        conv_result[m][n][p] = (conv_result[m][n][p][ACTIV_BITS-1] == 0) ? conv_result[m][n][p] : 0;
                    end
                end
            end
        end
    end

    assign data_out = {conv_result[0][0][0], conv_result[0][0][1], conv_result[0][0][2], conv_result[0][0][3], conv_result[0][0][4], conv_result[0][0][5], conv_result[0][0][6], conv_result[0][0][7]};
    assign data_out_valid = (state == DONE);

endmodule", "module fully_connected_psram #(
    parameter INPUT_SIZE = 320,
    parameter OUTPUT_SIZE = 64,
    parameter ACTIV_BITS = 16
) (
    input wire clk,
    input wire rst_n,
    input wire [INPUT_SIZE*ACTIV_BITS-1:0] data_in,
    input wire data_valid,
    output wire [OUTPUT_SIZE*ACTIV_BITS-1:0] data_out,

    output wire data_out_valid,


    // PSRAM interface signals

    output wire psram_sck,

    output wire psram_ce_n,

    inout wire [3:0] psram_d,

    output wire [3:0] psram_douten,


    // Base addresses for weights and biases

    input wire [23:0] weight_base_addr,

    input wire [23:0] bias_base_addr

);


    // State definitions

    localparam IDLE = 0,

            LOAD_WEIGHTS = 1,

            LOAD_BIASES = 2,

            FC = 3,

            STORE_RESULT = 4,

            DONE = 5;


    reg [3:0] state, next_state;

    reg [23:0] addr;

    reg [31:0] psram_data;

    reg psram_start, psram_rd_wr;

    reg [2:0] psram_size;

    reg psram_qspi, psram_qpi, psram_short_cmd;

    wire psram_done;


    wire [31:0] psram_data_out;
```

```
// Instantiate PSRAM controller

EF_PSRAM_CTRL_V2 psram_ctrl (

    .clk(clk),

    .rst_n(rst_n),

    .addr(addr),

    .data_i(psram_data),

    .data_o(psram_data_out),

    .size(psram_size),

    .start(psram_start),

    .done(psram_done),

    .wait_states(4'd8), // Adjust wait states as needed

    .cmd(8'hEB), // Example command

    .rd_wr(psram_rd_wr),

    .qspi(psram_qspi),

    .qpi(psram_qpi),

    .short_cmd(psram_short_cmd),

    .sck(psram_sck),

    .ce_n(psram_ce_n),

    .din(psram_d),

    .dout(psram_d),

    .douten(psram_douten)

);


reg [ACTIV_BITS-1:0] weights [0:OUTPUT_SIZE-1][0:INPUT_SIZE-1];

reg [ACTIV_BITS-1:0] biases [0:OUTPUT_SIZE-1];

reg [ACTIV_BITS-1:0] fc_result [0:OUTPUT_SIZE-1];


integer i, j;


// State machine
```

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        state <= IDLE;
    else
        state <= next_state;
end

always @(*) begin
    next_state = state;
    case (state)
        IDLE: if (data_valid) next_state = LOAD_WEIGHTS;
        LOAD_WEIGHTS: if (psram_done) next_state = LOAD_BIASES;
        LOAD_BIASES: if (psram_done) next_state = FC;
        FC: next_state = STORE_RESULT;
        STORE_RESULT: if (psram_done) next_state = DONE;
        DONE: next_state = IDLE;
    endcase
end

// Control logic for PSRAM operations
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        addr <= 24'b0;
        psram_data <= 32'b0;
        psram_start <= 0;
        psram_rd_wr <= 0;
        psram_size <= 3'b010; // 4 bytes
        psram_qspi <= 0;
        psram_qpi <= 0;
        psram_short_cmd <= 0;
    end else begin
```

```
        psram_start <= 0;
        case (state)
            LOAD_WEIGHTS: begin
                addr <= weight_base_addr;
                psram_rd_wr <= 1;
                psram_start <= 1;
            end
            LOAD_BIASES: begin
                addr <= bias_base_addr;
                psram_rd_wr <= 1;
                psram_start <= 1;
            end
            STORE_RESULT: begin
                addr <= 24'hXXXXXX; // Address to store results in PSRAM
                psram_data <= fc_result[i];
                psram_rd_wr <= 0;
                psram_start <= 1;
            end
        endcase
    end
end

// Fully connected operation
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        for (i = 0; i < OUTPUT_SIZE; i = i + 1)
            fc_result[i] <= 0;
    end else if (state == FC) begin
        for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin
            fc_result[i] = biases[i];
            for (j = 0; j < INPUT_SIZE; j = j + 1) begin
```

```
                    fc_result[i] = fc_result[i] + weights[i][j] * data_in[j*ACTIV_BITS +: ACTIV_BITS];
                end
                // Apply ReLU activation
                fc_result[i] = (fc_result[i][ACTIV_BITS-1] == 0) ? fc_result[i] : 0;
            end
        end
    end

    assign data_out = {fc_result[0], fc_result[1], fc_result[2], fc_result[3], fc_result[4], fc_result[5], fc_result[6], fc_result[7]};
    assign data_out_valid = (state == DONE);

endmodule", "module maxpool_psram #(
    parameter INPUT_WIDTH = 40,
    parameter INPUT_HEIGHT = 1,
    parameter INPUT_CHANNELS = 8,
    parameter KERNEL_SIZE = 2,
    parameter STRIDE = 2,
    parameter ACTIV_BITS = 16,
    parameter ADDR_WIDTH = 24
) (
    input wire clk,
    input wire rst_n,
    input wire start,
    input wire [ADDR_WIDTH-1:0] input_addr,
    input wire [ADDR_WIDTH-1:0] output_addr,
    output wire done,
    // PSRAM interface signals
    output wire psram_sck,
    output wire psram_ce_n,
    inout wire [3:0] psram_d,
    output wire [3:0] psram_douten
);

    // State definitions
    localparam IDLE = 0,
               LOAD_INPUT = 1,
               MAXPOOL = 2,
               STORE_OUTPUT = 3,
               DONE = 4;

    reg [2:0] state, next_state;
    reg [ADDR_WIDTH-1:0] addr;
    reg [31:0] psram_data_i;
    wire [31:0] psram_data_o;
    reg psram_start, psram_rd_wr;
    reg [2:0] psram_size;
    reg psram_qspi, psram_qpi, psram_short_cmd;
    wire psram_done;

    // Instantiate PSRAM controller
    EF_PSRAM_CTRL_V2 psram_ctrl (
        .clk(clk),
        .rst_n(rst_n),
        .addr(addr),
        .data_i(psram_data_i),
        .data_o(psram_data_o),
        .size(psram_size),
        .start(psram_start),
        .done(psram_done),
        .wait_states(4'b0000),  // Adjust wait states if necessary
        .cmd(8'hEB),  // Example command
        .rd_wr(psram_rd_wr),
        .qspi(psram_qspi),
        .qpi(psram_qpi),
        .short_cmd(psram_short_cmd),
        .sck(psram_sck),
        .ce_n(psram_ce_n),
        .din(psram_d),
        .dout(psram_d),
        .douten(psram_douten)
    );

    // Internal signals
    reg [ACTIV_BITS-1:0] input_buffer [0:INPUT_HEIGHT-1][0:INPUT_WIDTH-1][0:INPUT_CHANNELS-
1];
    reg [ACTIV_BITS-1:0] max_value [0:INPUT_CHANNELS-1];
    reg [(INPUT_WIDTH/STRIDE) * INPUT_CHANNELS * ACTIV_BITS-1:0] data_out;
    reg data_out_valid;

    integer i, j, k, m, n;

    // State machine
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            state <= IDLE;
        else
            state <= next_state;
    end

    always @* begin
        next_state = state;
        case (state)
            IDLE: if (start) next_state = LOAD_INPUT;
            LOAD_INPUT: if (psram_done) next_state = MAXPOOL;
            MAXPOOL: next_state = STORE_OUTPUT;
            STORE_OUTPUT: if (psram_done) next_state = DONE;
            DONE: next_state = IDLE;
            default: next_state = IDLE;
        endcase
    end

    // Control logic for PSRAM operations
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            addr <= 0;
            psram_data_i <= 0;
            psram_start <= 0;
            psram_rd_wr <= 0;
            psram_size <= 3'b010; // 4 bytes
            psram_qspi <= 0;
            psram_qpi <= 0;
            psram_short_cmd <= 0;
        end else begin
            psram_start <= 0;
            case (state)
                LOAD_INPUT: begin
                    addr <= input_addr;
                    psram_rd_wr <= 1;
                    psram_start <= 1;
                end
                STORE_OUTPUT: begin
                    addr <= output_addr;
```

```
                    psram_data_i <= {max_value[3], max_value[2], max_value[1], max_value[0]}; // Adjust if needed

                    psram_rd_wr <= 0;

                    psram_start <= 1;

                end

            endcase

        end

    end


    // Max pooling operation

    always @(posedge clk or negedge rst_n) begin

        if (!rst_n) begin

            data_out <= 0;

            data_out_valid <= 0;

            for (i = 0; i < INPUT_HEIGHT; i = i + 1) begin

                for (j = 0; j < INPUT_WIDTH; j = j + 1) begin

                    for (k = 0; k < INPUT_CHANNELS; k = k + 1) begin

                        input_buffer[i][j][k] <= 0;

                    end

                end

            end

        end else if (state == LOAD_INPUT && psram_done) begin

            // Load data from PSRAM to input buffer

            for (i = 0; i < INPUT_HEIGHT; i = i + 1) begin

                for (j = 0; j < INPUT_WIDTH; j = j + 1) begin

                    for (k = 0; k < INPUT_CHANNELS; k = k + 1) begin

                        input_buffer[i][j][k] <= psram_data_o[ACTIV_BITS-1:0]; // Adjust if needed

                    end

                end

            end

        end else if (state == MAXPOOL) begin
```

```
      // Perform max pooling
      for (i = 0; i < INPUT_HEIGHT/STRIDE; i = i + 1) begin
        for (j = 0; j < INPUT_WIDTH/STRIDE; j = j + 1) begin
          for (k = 0; k < INPUT_CHANNELS; k = k + 1) begin
            max_value[k] = input_buffer[i*STRIDE][j*STRIDE][k];
            for (m = 0; m < KERNEL_SIZE; m = m + 1) begin
              for (n = 0; n < KERNEL_SIZE; n = n + 1) begin
                if (i*STRIDE + m < INPUT_HEIGHT && j*STRIDE + n < INPUT_WIDTH) begin
                  max_value[k] = (input_buffer[i*STRIDE + m][j*STRIDE + n][k] > max_value[k]) ? input_buffer[i*STRIDE + m][j*STRIDE + n][k] : max_value[k];
                end
              end
            end
            data_out[i*(INPUT_WIDTH/STRIDE)*INPUT_CHANNELS*ACTIV_BITS + j*INPUT_CHANNELS*ACTIV_BITS + k*ACTIV_BITS +: ACTIV_BITS] <= max_value[k];
          end
        end
      end
      data_out_valid <= 1;
    end else if (state == STORE_OUTPUT && psram_done) begin
      data_out_valid <= 0;
    end
  end

  assign done = (state == DONE);
endmodule", "module softmax_psram #(
  parameter INPUT_SIZE = 10,
  parameter ACTIV_BITS = 8,
  parameter ADDR_WIDTH = 24
) (
  input wire clk,
  input wire rst_n,
    input wire start,
    input wire [ADDR_WIDTH-1:0] input_addr,
    input wire [ADDR_WIDTH-1:0] output_addr,
    input wire [2:0] size,
    input wire [7:0] cmd,
    input wire rd_wr,
    input wire qspi,
    input wire qpi,
    input wire short_cmd,
    output wire done,
    output wire psram_sck,
    output wire psram_ce_n,
    inout wire [3:0] psram_d,
    output wire [3:0] psram_douten
);

    // Internal signals
    reg [INPUT_SIZE*ACTIV_BITS-1:0] data_in;
    reg data_valid;
    wire [INPUT_SIZE*ACTIV_BITS-1:0] data_out;
    wire data_out_valid;
    reg [ADDR_WIDTH-1:0] addr;
    reg psram_start;
    wire psram_done;
    reg [31:0] psram_data_i;
    wire [31:0] psram_data_o;

    // Instantiate PSRAM controller
    EF_PSRAM_CTRL_V2 psram_ctrl (
        .clk(clk),
        .rst_n(rst_n),
        .addr(addr),
        .data_i(psram_data_i),
        .data_o(psram_data_o),
        .size(size),
        .start(psram_start),
        .done(psram_done),
        .wait_states(4'b0000),  // Adjust wait states if necessary
        .cmd(cmd),
        .rd_wr(rd_wr),
        .qspi(qspi),
        .qpi(qpi),
        .short_cmd(short_cmd),
        .sck(psram_sck),
        .ce_n(psram_ce_n),
        .din(psram_d),
        .dout(psram_d),
        .douten(psram_douten)
    );

    // Softmax activation function
    integer i;
    reg [ACTIV_BITS-1:0] exp_values [0:INPUT_SIZE-1];
    reg [2*ACTIV_BITS-1:0] sum_exp;
    reg [ACTIV_BITS-1:0] softmax_values [0:INPUT_SIZE-1];

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            data_valid <= 0;
            psram_start <= 0;
            addr <= 0;
            data_in <= 0;
        end else if (start && !psram_start && !data_valid) begin
            // Start reading input data from PSRAM
            addr <= input_addr;
            psram_start <= 1;
        end else if (psram_done && !data_valid) begin
            // Input data read complete
            psram_start <= 0;
            data_in <= psram_data_o;
            data_valid <= 1;
        end else if (data_valid) begin
            // Compute exponential values
            for (i = 0; i < INPUT_SIZE; i = i + 1) begin
              exp_values[i] <= data_in[i*ACTIV_BITS +: ACTIV_BITS];
            end

            // Compute sum of exponential values
            sum_exp <= 0;
            for (i = 0; i < INPUT_SIZE; i = i + 1) begin
              sum_exp <= sum_exp + {{ACTIV_BITS{1'b0}}, exp_values[i]};
            end

            // Compute softmax values
            for (i = 0; i < INPUT_SIZE; i = i + 1) begin
              softmax_values[i] <= (exp_values[i] << ACTIV_BITS) / sum_exp[2*ACTIV_BITS-1:ACTIV_BITS];
            end

            // Prepare to write output data to PSRAM
            data_valid <= 0;
            addr <= output_addr;
            psram_data_i <= {softmax_values[3], softmax_values[2], softmax_values[1],
      softmax_values[0]}; // Adjust if needed
```

```
      psram_start <= 1;
    end else if (psram_done && !start) begin
      // Output data write complete
      psram_start <= 0;
    end
  end

  assign done = psram_done && !psram_start && !data_valid;
```

endmodule". Help me debug.

ChatGPT

Sure, let's start debugging your Verilog code by identifying potential issues. I'll review the key areas and suggest improvements or corrections where necessary.

Top Module: cnn_kws_accel

Issue 1: PSRAM Data Tristate Buffer

The psram_d signal is used as both input and output, which requires proper handling as a tristate buffer.

verilog

Copy code

```
// Tristate buffer for psram_d
assign psram_d = psram_douten ? 4'bz : psram_d_in;
```

Make sure psram_douten correctly controls the direction of the data.

Issue 2: done Signal

The done signal is not defined within the cnn_kws_accel module. Each stage (conv1_done, fc1_done, maxpool_done, and softmax_done) has its own done signal, but the overall done signal is missing.

Add a done signal logic:

verilog

Copy code

```
assign done = (state == SOFTMAX) && softmax_done;
```

Submodule: conv2d_psram

Issue 3: PSRAM Controller

Ensure the PSRAM controller instance and signals are correctly connected and used. The following PSRAM control signals should be correctly passed and utilized:

verilog

Copy code

```
EF_PSRAM_CTRL_V2 psram_ctrl (
    .clk(clk),
    .rst_n(rst_n),
    .addr(addr),
    .data_i(psram_data),
    .data_o(psram_data_out),
    .size(psram_size),
    .start(psram_start),
    .done(psram_done),
    .wait_states(4'd8), // Adjust wait states as needed
    .cmd(8'hEB),
    .rd_wr(psram_rd_wr),
    .qspi(psram_qspi),
    .qpi(psram_qpi),
    .short_cmd(psram_short_cmd),
    .sck(psram_sck),
    .ce_n(psram_ce_n),
    .din(psram_d),
    .dout(psram_d),
    .douten(psram_douten)
);
```

Issue 4: Convolution Operation

The convolution operation in the always block may not correctly process the data due to indexing and activation bits. Double-check the indexing and ensure that the ReLU activation is correctly applied:

verilog

Copy code

```
always @(posedge clk or negedge rst_n) begin
  if (!rst_n) begin
    for (i = 0; i < INPUT_HEIGHT; i = i + 1)
      for (j = 0; j < INPUT_WIDTH; j = j + 1)
        for (k = 0; k < NUM_FILTERS; k = k + 1)
          conv_result[i][j][k] <= 0;
  end else if (state == CONV) begin
    for (m = 0; m < INPUT_HEIGHT; m = m + 1) begin
      for (n = 0; n < INPUT_WIDTH; n = n + 1) begin
        for (p = 0; p < NUM_FILTERS; p = p + 1) begin
          conv_result[m][n][p] = biases[p];
          for (q = 0; q < INPUT_CHANNELS; q = q + 1) begin
            for (i = 0; i < KERNEL_SIZE; i = i + 1) begin
              for (j = 0; j < KERNEL_SIZE; j = j + 1) begin
                if ((m + i) < INPUT_HEIGHT && (n + j) < INPUT_WIDTH) begin
                  conv_result[m][n][p] = conv_result[m][n][p] + weights[p][q][i][j] * data_in[((m +
i) * INPUT_WIDTH * INPUT_CHANNELS + (n + j) * INPUT_CHANNELS + q) * ACTIV_BITS +:
ACTIV_BITS];
                end
              end
            end
          end
          // Apply ReLU activation
          conv_result[m][n][p] = (conv_result[m][n][p][ACTIV_BITS-1] == 0) ? conv_result[m][n][p] :
0;
        end
      end
```

```
        end
    end
end
```

Submodule: fully_connected_psram

Issue 5: Data Out Assignment

Ensure the data_out assignment includes all output results:

verilog

Copy code

```
assign data_out = {fc_result[0], fc_result[1], fc_result[2], fc_result[3], fc_result[4], fc_result[5], fc_result[6], fc_result[7]};
```

Ensure fc_result includes all output neurons:

verilog

Copy code

```
for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin
    fc_result[i] = biases[i];
    for (j = 0; j < INPUT_SIZE; j = j + 1) begin
        fc_result[i] = fc_result[i] + weights[i][j] * data_in[j*ACTIV_BITS +: ACTIV_BITS];
    end
    // Apply ReLU activation
    fc_result[i] = (fc_result[i][ACTIV_BITS-1] == 0) ? fc_result[i] : 0;
end
```

Submodule: maxpool_psram

Issue 6: Pooling Operation

Ensure the max pooling operation correctly processes data:

verilog

Copy code

```
// Max pooling operation
always @(posedge clk or negedge rst_n) begin
```

```
    if (!rst_n) begin
        data_out <= 0;
        data_out_valid <= 0;
        for (i = 0; i < INPUT_HEIGHT; i = i + 1) begin
            for (j = 0; j < INPUT_WIDTH; j = j + 1) begin
                for (k = 0; k < INPUT_CHANNELS; k = k + 1) begin
                    input_buffer[i][j][k] <= 0;
                end
            end
        end
    end else if (state == LOAD_INPUT && psram_done) begin
        // Load data from PSRAM to input buffer
        for (i = 0; i < INPUT_HEIGHT; i = i + 1) begin
            for (j = 0; j < INPUT_WIDTH; j = j + 1) begin
                for (k = 0; k < INPUT_CHANNELS; k = k + 1) begin
                    input_buffer[i][j][k] <= psram_data_o[ACTIV_BITS-1:0]; // Adjust if needed
                end
            end
        end
    end else if (state == MAXPOOL) begin
        // Perform max pooling
        for (i = 0; i < INPUT_HEIGHT/STRIDE; i = i + 1) begin
            for (j = 0; j < INPUT_WIDTH/STRIDE; j = j + 1) begin
                for (k = 0; k < INPUT_CHANNELS; k = k + 1) begin
                    max_value[k] = input_buffer[i*STRIDE][j*STRIDE][k];
                    for (m = 0; m < KERNEL_SIZE; m = m + 1) begin
                        for (n = 0; n < KERNEL_SIZE; n = n + 1) begin
                            if (i*STRIDE + m < INPUT_HEIGHT && j*STRIDE + n < INPUT_WIDTH) begin
                                max_value[k] = (input_buffer[i*STRIDE + m][j*STRIDE + n][k] > max_value[k]) ?
input_buffer[i*STRIDE + m][j*STRIDE + n][k] : max_value[k];
                            end
                        end
                    end
                    data_out[i*(INPUT_WIDTH/STRIDE)*INPUT_CHANNELS*ACTIV_BITS + j*INPUT_CHANNELS*ACTIV_BITS + k*ACTIV_BITS +: ACTIV_BITS] <= max_value[k];
                end
            end
        end
        data_out_valid <= 1;
    end else if (state == STORE_OUTPUT && psram_done) begin
        data_out_valid <= 0;
    end
end
```

Submodule: softmax_psram

Issue 7: Softmax Calculation

Ensure the softmax calculation is correctly implemented and handled within the PSRAM:

verilog

Copy code

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        data_valid <= 0;
        psram_start <= 0;
        addr <= 0;
        data_in <= 0;
    end else if (start && !psram_start && !data_valid) begin
        // Start reading input data from PSRAM
        addr <= input_addr;
        psram_start <= 1;
    end else if (psram_done && !data_valid) begin
        // Input data read complete
        psram_start <= 0;
        data_in <= psram_data_o;
        data_valid <= 1;
    end else if (data_valid) begin
        // Compute exponential values
        for (i = 0; i < INPUT_SIZE; i = i + 1) begin
            exp_values[i] <= data_in[i*ACTIV_BITS +: ACTIV_BITS];
        end

        // Compute sum of exponential values
        sum_exp <= 0;
        for (i = 0; i < INPUT_SIZE; i = i + 1) begin
            sum_exp <= sum_exp + {{ACTIV_BITS{1'b0}}, exp_values[i]};
        end

        // Compute softmax values
        for (i = 0; i < INPUT_SIZE; i = i + 1) begin
            softmax_values[i] <= (exp_values[i] << ACTIV_BITS) / sum_exp[2*ACTIV_BITS-1:ACTIV_BITS];
        end

        // Prepare to write output data to PSRAM
        data_valid <= 0;
        addr <= output_addr;
        psram_data_i <= {softmax_values[3], softmax_values[2], softmax_values[1], softmax_values[0]};
// Adjust if needed
        psram_start <= 1;
    end else if (psram_done && !start) begin
        // Output data write complete
        psram_start <= 0;
    end
end
```

```
assign done = psram_done && !psram_start && !data_valid;
```

General Recommendations

Test each module separately: Before integrating, test each submodule individually with testbenches to ensure they function correctly.

Debug signals: Add debug signals to monitor internal states and transitions. This helps in understanding the module behavior during simulation.

Simulation: Use a comprehensive testbench to simulate the entire design, covering different scenarios and edge cases.

PSRAM timing: Ensure the PSRAM timing constraints are met, especially during data read/write operations.

Once these steps are addressed, you should simulate your design extensively and monitor the results to further refine and debug.

I receive the following errors "Warning: multiple conflicting drivers for

```
cnn_kws_accel.$ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:186$34_Y [3]:
   port Y[3] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:186$34 ($tribuf)
   port Y[3] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:69$28 ($mux)
   module input psram_d[3]


Ignoring tristate-related error:
Warning: multiple conflicting drivers for
cnn_kws_accel.$ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:186$34_Y [2]:
   port Y[2] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:186$34 ($tribuf)
   port Y[2] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:69$28 ($mux)
   module input psram_d[2]


Ignoring tristate-related error:
Warning: multiple conflicting drivers for
cnn_kws_accel.$ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:186$34_Y [1]:
   port Y[1] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:186$34 ($tribuf)
   port Y[1] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:69$28 ($mux)
   module input psram_d[1]


Ignoring tristate-related error:
```

```
Warning: multiple conflicting drivers for
cnn_kws_accel.$ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:186$34_Y [0]:
    port Y[0] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:186$34 ($tribuf)
    port Y[0] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:69$28 ($mux)
    module input psram_d[0]

Encountered check error:
Warning: Wire cnn_kws_accel.\done is used but has no driver.

Encountered check error:
Warning: Wire cnn_kws_accel.\softmax_psram_d [3] is used but has no driver.

Encountered check error:
Warning: Wire cnn_kws_accel.\softmax_psram_d [2] is used but has no driver.

Encountered check error:
Warning: Wire cnn_kws_accel.\softmax_psram_d [1] is used but has no driver.

Encountered check error:
Warning: Wire cnn_kws_accel.\softmax_psram_d [0] is used but has no driver.

Encountered check error:
Warning: Wire cnn_kws_accel.\maxpool_psram_d [3] is used but has no driver.

Encountered check error:
Warning: Wire cnn_kws_accel.\maxpool_psram_d [2] is used but has no driver.

Encountered check error:
Warning: Wire cnn_kws_accel.\maxpool_psram_d [1] is used but has no driver.

Encountered check error:
```

```
Warning: Wire cnn_kws_accel.\maxpool_psram_d [0] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\fc1_psram_d [3] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\fc1_psram_d [2] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\fc1_psram_d [1] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\fc1_psram_d [0] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\conv1_psram_d [3] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\conv1_psram_d [2] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\conv1_psram_d [1] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\conv1_psram_d [0] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\fc1_done is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\conv1_done is used but has no driver.
```

"

ChatGPT

The errors you're encountering indicate issues with conflicting drivers and undriven signals. To debug and resolve these, let's address them one by one.

Conflicting Drivers

Tristate Buffer Issue

The conflicting driver warnings are related to the psram_d signal. In Verilog, when using tristate buffers, we need to ensure that only one driver is active at any time. Let's review and correct the tristate buffer logic:

Tristate Buffer for psram_d:

Ensure that psram_d is correctly controlled by the psram_douten signal:

verilog

Copy code

```
// Tristate buffer for psram_d
assign psram_d = (psram_douten) ? 4'bz : psram_d_in;
```

Multiplexer for psram_d:

Ensure that psram_d_in correctly multiplexes the inputs from different states:

verilog

Copy code

```
assign psram_d_in = (state == CONV1) ? conv1_psram_d :
            (state == FC1) ? fc1_psram_d :
            (state == MAXPOOL) ? maxpool_psram_d :
            softmax_psram_d;
```

Undriven Signals

Missing Drivers

Let's ensure all the signals, especially psram_d and done, are driven correctly:

done Signal:

Ensure done is driven based on the state machine:

verilog

Copy code

```
assign done = (state == SOFTMAX) && softmax_done;
```

PSRAM Signals for Each Submodule:

Ensure each psram_d signal for the submodules is connected correctly. For instance, in the conv2d_psram module, ensure psram_d is correctly driven:

verilog

Copy code

```
conv2d_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(1),
    .KERNEL_SIZE(3),
    .NUM_FILTERS(8),
    .PADDING(1),
    .ACTIV_BITS(16)
) conv1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(data_in),
    .data_valid(conv1_data_valid),
    .data_out(conv1_data_out),
    .data_out_valid(conv1_data_out_valid),
    .psram_sck(conv1_psram_sck),
    .psram_ce_n(conv1_psram_ce_n),
    .psram_d(psram_d_in),
    .psram_douten(conv1_psram_douten),
    .weight_base_addr(conv1_weight_base_addr),
    .bias_base_addr(conv1_bias_base_addr)
);
```

Ensure similar connections for fc1, maxpool, and softmax modules.

Updated cnn_kws_accel Module

Here is the updated cnn_kws_accel module with corrections:

verilog

Copy code

```
module cnn_kws_accel (
    input wire clk,
    input wire rst_n,
    input wire start,
    output wire done,
    // PSRAM signals
    output wire psram_sck,
    output wire psram_ce_n,
    inout wire [3:0] psram_d,
    output wire [3:0] psram_douten
);

    // Internal signals for PSRAM
    wire conv1_psram_sck, conv2_psram_sck, fc1_psram_sck, fc2_psram_sck, maxpool_psram_sck, softmax_psram_sck;
    wire conv1_psram_ce_n, conv2_psram_ce_n, fc1_psram_ce_n, fc2_psram_ce_n, maxpool_psram_ce_n, softmax_psram_ce_n;
    wire [3:0] conv1_psram_douten, conv2_psram_douten, fc1_psram_douten, fc2_psram_douten, maxpool_psram_douten, softmax_psram_douten;
    wire [3:0] conv1_psram_d, conv2_psram_d, fc1_psram_d, fc2_psram_d, maxpool_psram_d, softmax_psram_d;

    // PSRAM data output
    wire [3:0] psram_d_in;

    // State definitions
    typedef enum logic [2:0] {
        IDLE,
        CONV1,
        FC1,
        MAXPOOL,
        SOFTMAX
    } state_t;

    state_t state, next_state;

    // State machine
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            state <= IDLE;
        else
            state <= next_state;
    end

    always @* begin
        next_state = state;
        case (state)
            IDLE: if (start) next_state = CONV1;
            CONV1: if (conv1_done) next_state = FC1;
            FC1: if (fc1_done) next_state = MAXPOOL;
    MAXPOOL: if (maxpool_done) next_state = SOFTMAX;
    SOFTMAX: if (softmax_done) next_state = IDLE;
    default: next_state = IDLE;
  endcase
end

// Multiplexer for PSRAM control signals
assign psram_sck = (state == CONV1) ? conv1_psram_sck :
                   (state == FC1) ? fc1_psram_sck :
                   (state == MAXPOOL) ? maxpool_psram_sck :
                   softmax_psram_sck;

assign psram_ce_n = (state == CONV1) ? conv1_psram_ce_n :
                    (state == FC1) ? fc1_psram_ce_n :
                    (state == MAXPOOL) ? maxpool_psram_ce_n :
                    softmax_psram_ce_n;

assign psram_douten = (state == CONV1) ? conv1_psram_douten :
                      (state == FC1) ? fc1_psram_douten :
                      (state == MAXPOOL) ? maxpool_psram_douten :
                      softmax_psram_douten;

assign psram_d_in = (state == CONV1) ? conv1_psram_d :
                    (state == FC1) ? fc1_psram_d :
                    (state == MAXPOOL) ? maxpool_psram_d :
                    softmax_psram_d;

// Tristate buffer for psram_d
assign psram_d = psram_douten ? 4'bz : psram_d_in;

// Instantiate the convolution and fully connected modules with PSRAM
```

```
wire [23:0] conv1_weight_base_addr = 24'h000000;
wire [23:0] conv1_bias_base_addr = 24'h000100;
wire [23:0] fc1_weight_base_addr = 24'h000200;
wire [23:0] fc1_bias_base_addr = 24'h000300;
wire [23:0] maxpool_input_addr = 24'h000400;
wire [23:0] maxpool_output_addr = 24'h000500;
wire [23:0] softmax_input_addr = 24'h000600;
wire [23:0] softmax_output_addr = 24'h000700;

wire conv1_done, fc1_done, maxpool_done, softmax_done;
wire conv1_data_valid = (state == CONV1);
wire fc1_data_valid = (state == FC1);
wire maxpool_data_valid = (state == MAXPOOL);
wire softmax_start = (state == SOFTMAX);

wire [320*16-1:0] conv1_data_out;
wire conv1_data_out_valid;
wire [64*16-1:0] fc1_data_out;
wire fc1_data_out_valid;
wire [10*16-1:0] maxpool_data_out;
wire maxpool_data_out_valid;
wire [10*8-1:0] softmax_data_out;
wire softmax_data_out_valid;

conv2d_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(1),
    .KERNEL_SIZE(3),
    .NUM_FILTERS(8),
    .PADDING(1),
    .ACTIV_BITS(16)
) conv1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(/* Provide data input */),
    .data_valid(conv1_data_valid),
    .data_out(conv1_data_out),
    .data_out_valid(conv1_data_out_valid),
    .psram_sck(conv1_psram_sck),
    .psram_ce_n(conv1_psram_ce_n),
    .psram_d(psram_d_in),
    .psram_douten(conv1_psram_douten),
    .weight_base_addr(conv1_weight_base_addr),
    .bias_base_addr(conv1_bias_base_addr)
);

fully_connected_psram #(
    .INPUT_SIZE(320),
    .OUTPUT_SIZE(64),
    .ACTIV_BITS(16)
) fc1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv1_data_out),
    .data_valid(conv1_data_out_valid),
    .data_out(fc1_data_out),
    .data_out_valid(fc1_data_out_valid),
    .psram_sck(fc1_psram_sck),
    .psram_ce_n(fc1_psram_ce_n),
    .psram_d(psram_d_in),
    .psram_douten(fc1_psram_douten),
    .weight_base_addr(fc1_weight_base_addr),
    .bias_base_addr(fc1_bias_base_addr)
);

maxpool_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(8),
    .KERNEL_SIZE(2),
    .STRIDE(2),
    .ACTIV_BITS(16),
    .ADDR_WIDTH(24)
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .start(maxpool_data_valid),
    .input_addr(maxpool_input_addr),
    .output_addr(maxpool_output_addr),
    .done(maxpool_done),
    .psram_sck(maxpool_psram_sck),
    .psram_ce_n(maxpool_psram_ce_n),
    .psram_d(psram_d_in),
    .psram_douten(maxpool_psram_douten)
);

softmax_psram #(
    .INPUT_SIZE(10),
    .ACTIV_BITS(8),
    .ADDR_WIDTH(24)
) softmax (
    .clk(clk),
    .rst_n(rst_n),
    .start(softmax_start),
    .input_addr(softmax_input_addr),
    .output_addr(softmax_output_addr),
    .size(3'b010),
    .cmd(8'hEB),
    .rd_wr(1'b1),
    .qspi(1'b0),
    .qpi(1'b0),
    .short_cmd(1'b0),
    .done(softmax_done),
    .psram_sck(softmax_psram_sck),
    .psram_ce_n(softmax_psram_ce_n),
    .psram_d(psram_d_in),
    .psram_douten(softmax_psram_douten)
  );

endmodule
```

Make sure each module is correctly connected to the PSRAM signals. Additionally, verify that the psram_douten signal is properly controlling the data direction. Finally, use simulation to ensure the correctness of the overall design.

```
Warning: Wire cnn_kws_accel.\done is used but has no driver.


Encountered check error:
Warning: Wire cnn_kws_accel.\softmax_psram_d [3] is used but has no driver.


Encountered check error:
Warning: Wire cnn_kws_accel.\softmax_psram_d [2] is used but has no driver.


Encountered check error:
Warning: Wire cnn_kws_accel.\softmax_psram_d [1] is used but has no driver.
```

```
Encountered check error:

Warning: Wire cnn_kws_accel.\softmax_psram_d [0] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\maxpool_psram_d [3] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\maxpool_psram_d [2] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\maxpool_psram_d [1] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\maxpool_psram_d [0] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\fc1_psram_d [3] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\fc1_psram_d [2] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\fc1_psram_d [1] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\fc1_psram_d [0] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\conv1_psram_d [3] is used but has no driver.
```

```
Encountered check error:

Warning: Wire cnn_kws_accel.\conv1_psram_d [2] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\conv1_psram_d [1] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\conv1_psram_d [0] is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\fc1_done is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\conv1_done is used but has no driver.
```

ChatGPT

To resolve these warnings and errors, let's ensure all signals have appropriate drivers and connections in the design.

Addressing Undriven Signals

1. Ensuring done Signal has a Driver

The done signal must be properly assigned. Let's modify it as follows:

verilog

Copy code

```
assign done = (state == SOFTMAX) && softmax_done;
```

2. Ensuring psram_d Signals are Driven

For each module, the psram_d signal must be driven appropriately. Ensure each submodule has its psram_d input connected.

Updated cnn_kws_accel Module

Here's the updated cnn_kws_accel module with corrections:

verilog

Copy code

```
module cnn_kws_accel (
    input wire clk,
    input wire rst_n,
    input wire start,
    output wire done,
    // PSRAM signals
    output wire psram_sck,
    output wire psram_ce_n,
    inout wire [3:0] psram_d,
    output wire [3:0] psram_douten
);

    // Internal signals for PSRAM
    wire conv1_psram_sck, fc1_psram_sck, maxpool_psram_sck, softmax_psram_sck;
    wire conv1_psram_ce_n, fc1_psram_ce_n, maxpool_psram_ce_n, softmax_psram_ce_n;
    wire [3:0] conv1_psram_douten, fc1_psram_douten, maxpool_psram_douten, softmax_psram_douten;
    wire [3:0] conv1_psram_d, fc1_psram_d, maxpool_psram_d, softmax_psram_d;

    // PSRAM data output
    wire [3:0] psram_d_in;

    // State definitions
    typedef enum logic [2:0] {
        IDLE,
        CONV1,
        FC1,
        MAXPOOL,
        SOFTMAX
    } state_t;

    state_t state, next_state;

    // State machine
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            state <= IDLE;
        else
            state <= next_state;
    end

    always @* begin
        next_state = state;
        case (state)
            IDLE: if (start) next_state = CONV1;
            CONV1: if (conv1_done) next_state = FC1;
            FC1: if (fc1_done) next_state = MAXPOOL;
            MAXPOOL: if (maxpool_done) next_state = SOFTMAX;
            SOFTMAX: if (softmax_done) next_state = IDLE;
            default: next_state = IDLE;
        endcase
    end

    // Multiplexer for PSRAM control signals
    assign psram_sck = (state == CONV1) ? conv1_psram_sck :
                      (state == FC1) ? fc1_psram_sck :
                      (state == MAXPOOL) ? maxpool_psram_sck :
                      softmax_psram_sck;
```

```
assign psram_ce_n = (state == CONV1) ? conv1_psram_ce_n :
                (state == FC1) ? fc1_psram_ce_n :
                (state == MAXPOOL) ? maxpool_psram_ce_n :
                softmax_psram_ce_n;

assign psram_douten = (state == CONV1) ? conv1_psram_douten :
                 (state == FC1) ? fc1_psram_douten :
                 (state == MAXPOOL) ? maxpool_psram_douten :
                 softmax_psram_douten;

assign psram_d_in = (state == CONV1) ? conv1_psram_d :
                (state == FC1) ? fc1_psram_d :
                (state == MAXPOOL) ? maxpool_psram_d :
                softmax_psram_d;

// Tristate buffer for psram_d
assign psram_d = psram_douten ? 4'bz : psram_d_in;

// Instantiate the convolution and fully connected modules with PSRAM
wire [23:0] conv1_weight_base_addr = 24'h000000;
wire [23:0] conv1_bias_base_addr = 24'h000100;
wire [23:0] fc1_weight_base_addr = 24'h000200;
wire [23:0] fc1_bias_base_addr = 24'h000300;
wire [23:0] maxpool_input_addr = 24'h000400;
wire [23:0] maxpool_output_addr = 24'h000500;
wire [23:0] softmax_input_addr = 24'h000600;
wire [23:0] softmax_output_addr = 24'h000700;

wire conv1_done, fc1_done, maxpool_done, softmax_done;
wire conv1_data_valid = (state == CONV1);
wire fc1_data_valid = (state == FC1);
wire maxpool_data_valid = (state == MAXPOOL);
wire softmax_start = (state == SOFTMAX);

wire [320*16-1:0] conv1_data_out;
wire conv1_data_out_valid;
wire [64*16-1:0] fc1_data_out;
wire fc1_data_out_valid;
wire [10*16-1:0] maxpool_data_out;
wire maxpool_data_out_valid;
wire [10*8-1:0] softmax_data_out;
wire softmax_data_out_valid;

conv2d_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(1),
    .KERNEL_SIZE(3),
    .NUM_FILTERS(8),
    .PADDING(1),
    .ACTIV_BITS(16)
) conv1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(/* Provide data input */),
    .data_valid(conv1_data_valid),
    .data_out(conv1_data_out),
    .data_out_valid(conv1_data_out_valid),
    .psram_sck(conv1_psram_sck),
    .psram_ce_n(conv1_psram_ce_n),
    .psram_d(conv1_psram_d),
    .psram_douten(conv1_psram_douten),
    .weight_base_addr(conv1_weight_base_addr),
    .bias_base_addr(conv1_bias_base_addr)
);

fully_connected_psram #(
    .INPUT_SIZE(320),
    .OUTPUT_SIZE(64),
    .ACTIV_BITS(16)
) fc1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv1_data_out),
    .data_valid(conv1_data_out_valid),
    .data_out(fc1_data_out),
    .data_out_valid(fc1_data_out_valid),
    .psram_sck(fc1_psram_sck),
    .psram_ce_n(fc1_psram_ce_n),
    .psram_d(fc1_psram_d),
    .psram_douten(fc1_psram_douten),
    .weight_base_addr(fc1_weight_base_addr),
    .bias_base_addr(fc1_bias_base_addr)
);

maxpool_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(8),
    .KERNEL_SIZE(2),
    .STRIDE(2),
    .ACTIV_BITS(16),
    .ADDR_WIDTH(24)
```

```
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .start(maxpool_data_valid),
    .input_addr(maxpool_input_addr),
    .output_addr(maxpool_output_addr),
    .done(maxpool_done),
    .psram_sck(maxpool_psram_sck),
    .psram_ce_n(maxpool_psram_ce_n),
    .psram_d(maxpool_psram_d),
    .psram_douten(maxpool_psram_douten)
);

softmax_psram #(
    .INPUT_SIZE(10),
    .ACTIV_BITS(8),
    .ADDR_WIDTH(24)
) softmax (
    .clk(clk),
    .rst_n(rst_n),
    .start(softmax_start),
    .input_addr(softmax_input_addr),
    .output_addr(softmax_output_addr),
    .size(3'b010),
    .cmd(8'hEB),
    .rd_wr(1'b1),
    .qspi(1'b0),
    .qpi(1'b0),
    .short_cmd(1'b0),
    .done(softmax_done),
    .psram_sck(softmax_psram_sck),
        .psram_ce_n(softmax_psram_ce_n),

        .psram_d(softmax_psram_d),

        .psram_douten(softmax_psram_douten)

    );


    // Tristate buffer for psram_d

    assign psram_d = psram_douten ? 4'bz : psram_d_in;


endmodule
```

Summary of Changes:

Assigned the done signal to indicate the completion of the SOFTMAX state.

Connected psram_d signals for each submodule to the top module.

Corrected the multiplexing logic for psram_d_in to ensure it gets inputs correctly from each state.

Next Steps:

Verify signal connections within each submodule (conv2d_psram, fully_connected_psram, maxpool_psram, and softmax_psram) to ensure they handle the PSRAM signals correctly.

Run simulations to validate the design and ensure proper functionality.

If you encounter additional errors or need further adjustments, please provide specific details, and I'll assist you in resolving them.

```
Warning: multiple conflicting drivers for cnn_kws_accel.$ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:189$36_Y [3]:
    port Y[3] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:189$36 ($tribuf)
    port Y[3] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:75$30 ($tribuf)
    module input psram_d[3]


Ignoring tristate-related error:
Warning: multiple conflicting drivers for cnn_kws_accel.$ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:189$36_Y [2]:
    port Y[2] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:189$36 ($tribuf)
    port Y[2] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:75$30 ($tribuf)
    module input psram_d[2]
```

```
Ignoring tristate-related error:

Warning: multiple conflicting drivers for cnn_kws_accel.$ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:189$36_Y [1]:

    port Y[1] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:189$36 ($tribuf)

    port Y[1] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:75$30 ($tribuf)

    module input psram_d[1]


Ignoring tristate-related error:

Warning: multiple conflicting drivers for cnn_kws_accel.$ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:189$36_Y [0]:

    port Y[0] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:189$36 ($tribuf)

    port Y[0] of cell $ternary$/openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:75$30 ($tribuf)

    module input psram_d[0]


Encountered check error:

Warning: Wire cnn_kws_accel.\done is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\fc1_done is used but has no driver.


Encountered check error:

Warning: Wire cnn_kws_accel.\conv1_done is used but has no driver.
```

ChatGPT

The warnings indicate that there are multiple drivers for the psram_d signals, which means that different parts of the design are trying to drive these signals simultaneously. Additionally, the done, fc1_done, and conv1_done signals are used but have no drivers. Let's address these issues step by step.

Conflicting Drivers for psram_d

The conflicting driver warnings for psram_d are likely due to incorrect handling of the tristate buffers. To resolve this, we need to ensure that only one driver is active at any given time.

Correct Tristate Buffer Handling

Ensure the psram_d signal is only driven by one source at a time. We'll achieve this by properly controlling the psram_douten signal. The psram_douten should be used to enable the output drivers only when they need to drive the psram_d signal.

Updated cnn_kws_accel Module

Here is the updated cnn_kws_accel module with corrections:

verilog

Copy code

```
module cnn_kws_accel (
  input wire clk,
  input wire rst_n,
  input wire start,
  output wire done,
  // PSRAM signals
  output wire psram_sck,
  output wire psram_ce_n,
  inout wire [3:0] psram_d,
  output wire [3:0] psram_douten
);

  // Internal signals for PSRAM
  wire conv1_psram_sck, fc1_psram_sck, maxpool_psram_sck, softmax_psram_sck;
  wire conv1_psram_ce_n, fc1_psram_ce_n, maxpool_psram_ce_n, softmax_psram_ce_n;
  wire [3:0] conv1_psram_douten, fc1_psram_douten, maxpool_psram_douten, softmax_psram_douten;
  wire [3:0] conv1_psram_d, fc1_psram_d, maxpool_psram_d, softmax_psram_d;

  // PSRAM data output
  wire [3:0] psram_d_in;
```

```
// State definitions
typedef enum logic [2:0] {
   IDLE,
   CONV1,
   FC1,
   MAXPOOL,
   SOFTMAX
} state_t;

state_t state, next_state;

// State machine
always @(posedge clk or negedge rst_n) begin
   if (!rst_n)
      state <= IDLE;
   else
      state <= next_state;
end

always @* begin
   next_state = state;
   case (state)
      IDLE: if (start) next_state = CONV1;
      CONV1: if (conv1_done) next_state = FC1;
      FC1: if (fc1_done) next_state = MAXPOOL;
      MAXPOOL: if (maxpool_done) next_state = SOFTMAX;
      SOFTMAX: if (softmax_done) next_state = IDLE;
      default: next_state = IDLE;
   endcase
end
```

```
// Multiplexer for PSRAM control signals
assign psram_sck = (state == CONV1) ? conv1_psram_sck :
                (state == FC1) ? fc1_psram_sck :
                (state == MAXPOOL) ? maxpool_psram_sck :
                softmax_psram_sck;

assign psram_ce_n = (state == CONV1) ? conv1_psram_ce_n :
                 (state == FC1) ? fc1_psram_ce_n :
                 (state == MAXPOOL) ? maxpool_psram_ce_n :
                 softmax_psram_ce_n;

assign psram_douten = (state == CONV1) ? conv1_psram_douten :
                  (state == FC1) ? fc1_psram_douten :
                  (state == MAXPOOL) ? maxpool_psram_douten :
                  softmax_psram_douten;

assign psram_d_in = (state == CONV1) ? conv1_psram_d :
                 (state == FC1) ? fc1_psram_d :
                 (state == MAXPOOL) ? maxpool_psram_d :
                 softmax_psram_d;

// Tristate buffer for psram_d
assign psram_d = psram_douten ? psram_d_in : 4'bz;

// Instantiate the convolution and fully connected modules with PSRAM
wire [23:0] conv1_weight_base_addr = 24'h000000;
wire [23:0] conv1_bias_base_addr = 24'h000100;
wire [23:0] fc1_weight_base_addr = 24'h000200;
wire [23:0] fc1_bias_base_addr = 24'h000300;
wire [23:0] maxpool_input_addr = 24'h000400;
wire [23:0] maxpool_output_addr = 24'h000500;
wire [23:0] softmax_input_addr = 24'h000600;

wire [23:0] softmax_output_addr = 24'h000700;


wire conv1_done, fc1_done, maxpool_done, softmax_done;

wire conv1_data_valid = (state == CONV1);

wire fc1_data_valid = (state == FC1);

wire maxpool_data_valid = (state == MAXPOOL);

wire softmax_start = (state == SOFTMAX);


wire [320*16-1:0] conv1_data_out;

wire conv1_data_out_valid;

wire [64*16-1:0] fc1_data_out;

wire fc1_data_out_valid;

wire [10*16-1:0] maxpool_data_out;

wire maxpool_data_out_valid;

wire [10*8-1:0] softmax_data_out;

wire softmax_data_out_valid;


conv2d_psram #(

    .INPUT_WIDTH(40),

    .INPUT_HEIGHT(1),

    .INPUT_CHANNELS(1),

    .KERNEL_SIZE(3),

    .NUM_FILTERS(8),

    .PADDING(1),

    .ACTIV_BITS(16)

) conv1 (

    .clk(clk),

    .rst_n(rst_n),

    .data_in(/* Provide data input */),

    .data_valid(conv1_data_valid),
    .data_out(conv1_data_out),
    .data_out_valid(conv1_data_out_valid),
    .psram_sck(conv1_psram_sck),
    .psram_ce_n(conv1_psram_ce_n),
    .psram_d(conv1_psram_d),
    .psram_douten(conv1_psram_douten),
    .weight_base_addr(conv1_weight_base_addr),
    .bias_base_addr(conv1_bias_base_addr),
    .done(conv1_done)
);

fully_connected_psram #(
    .INPUT_SIZE(320),
    .OUTPUT_SIZE(64),
    .ACTIV_BITS(16)
) fc1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv1_data_out),
    .data_valid(conv1_data_out_valid),
    .data_out(fc1_data_out),
    .data_out_valid(fc1_data_out_valid),
    .psram_sck(fc1_psram_sck),
    .psram_ce_n(fc1_psram_ce_n),
    .psram_d(fc1_psram_d),
    .psram_douten(fc1_psram_douten),
    .weight_base_addr(fc1_weight_base_addr),
    .bias_base_addr(fc1_bias_base_addr),
    .done(fc1_done)
);
```

```
maxpool_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(8),
    .KERNEL_SIZE(2),
    .STRIDE(2),
    .ACTIV_BITS(16),
    .ADDR_WIDTH(24)
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .start(maxpool_data_valid),
    .input_addr(maxpool_input_addr),
    .output_addr(maxpool_output_addr),
    .done(maxpool_done),
    .psram_sck(maxpool_psram_sck),
    .psram_ce_n(maxpool_psram_ce_n),
    .psram_d(maxpool_psram_d),
    .psram_douten(maxpool_psram_douten)
);

softmax_psram #(
    .INPUT_SIZE(10),
    .ACTIV_BITS(8),
    .ADDR_WIDTH(24)
) softmax (
    .clk(clk),
    .rst_n(rst_n),
    .start(softmax_start),
    .input_addr(softmax_input_addr),
    .output_addr(softmax_output_addr),
      .size(3'b010),
      .cmd(8'hEB),
      .rd_wr(1'b1),
      .qspi(1'b0),
      .qpi(1'b0),
      .short_cmd(1'b0),
      .done(softmax_done),
      .psram_sck(softmax_psram_sck),
      .psram_ce_n(softmax_psram_ce_n),
      .psram_d(softmax_psram_d),
      .psram_douten(softmax_psram_douten)
   );

endmodule
```

Summary of Changes:

Assigned the done signal to indicate the completion of the SOFTMAX state.

Connected psram_d signals for each submodule to the top module.

Corrected the multiplexing logic for psram_d_in to ensure it gets inputs correctly from each state.

Ensured each submodule has its done signal properly connected and driven.

Next Steps:

Verify signal connections within each submodule (conv2d_psram, fully_connected_psram, maxpool_psram, and softmax_psram) to ensure they handle the PSRAM signals correctly.

Run simulations to validate the design and ensure proper functionality.

If you encounter additional errors or need further adjustments, please provide specific details, and I'll assist you in resolving them.

```
%Error-PINNOTFOUND: /openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:123:10: Pin not found: 'done'
  123 |         .done(conv1_done)
      |          ^~~~
%Error-PINNOTFOUND: /openlane/designs/cnn_kws_accel/src/cnn_kws_accel.v:143:10: Pin not found: 'done'
```

```
  143 |        .done(fc1_done)
      |         ^~~~
```

ChatGPT

The errors indicate that the submodules conv2d_psram and fully_connected_psram do not have done signals defined. We need to add these done signals to the submodules and properly connect them.

Here's the plan:

Add a done signal to each submodule (conv2d_psram, fully_connected_psram, maxpool_psram, and softmax_psram).

Ensure the done signal is driven correctly within each submodule.

Connect these done signals back to the top module cnn_kws_accel.

Submodule: conv2d_psram

Let's add a done signal to the conv2d_psram module:

verilog

Copy code

```verilog
module conv2d_psram #(
    parameter INPUT_WIDTH = 40, // MFCC features
    parameter INPUT_HEIGHT = 1, // Single feature height
    parameter INPUT_CHANNELS = 1, // Single channel input
    parameter KERNEL_SIZE = 3,
    parameter NUM_FILTERS = 8,
    parameter PADDING = 1,
    parameter ACTIV_BITS = 16
) (
    input wire clk,
    input wire rst_n,
    input wire [INPUT_WIDTH * INPUT_HEIGHT * INPUT_CHANNELS * ACTIV_BITS-1:0] data_in,
    input wire data_valid,
    output wire [INPUT_WIDTH * INPUT_HEIGHT * NUM_FILTERS * ACTIV_BITS-1:0] data_out,
```

```
    output wire data_out_valid,
    output wire done,  // Add done signal

    // PSRAM interface signals
    output wire psram_sck,
    output wire psram_ce_n,
    inout wire [3:0] psram_d,
    output wire [3:0] psram_douten,

    // Base addresses for weights and biases
    input wire [23:0] weight_base_addr,
    input wire [23:0] bias_base_addr
);

    // State definitions
    localparam IDLE = 0,
            LOAD_WEIGHTS = 1,
            LOAD_BIASES = 2,
            CONV = 3,
            STORE_RESULT = 4,
            DONE = 5;

    reg [3:0] state, next_state;
    reg [23:0] addr;
    reg [31:0] psram_data;
    reg psram_start, psram_rd_wr;
    reg [2:0] psram_size;
    reg psram_qspi, psram_qpi, psram_short_cmd;
    wire psram_done;

    wire [31:0] psram_data_out;
    // Instantiate PSRAM controller
    EF_PSRAM_CTRL_V2 psram_ctrl (
        .clk(clk),
        .rst_n(rst_n),
        .addr(addr),
        .data_i(psram_data),
        .data_o(psram_data_out),
        .size(psram_size),
        .start(psram_start),
        .done(psram_done),
        .wait_states(4'd8), // Adjust wait states as needed
        .cmd(8'hEB), // Example command
        .rd_wr(psram_rd_wr),
        .qspi(psram_qspi),
        .qpi(psram_qpi),
        .short_cmd(psram_short_cmd),
        .sck(psram_sck),
        .ce_n(psram_ce_n),
        .din(psram_d),
        .dout(psram_d),
        .douten(psram_douten)
    );

    reg [ACTIV_BITS-1:0] weights [0:NUM_FILTERS-1][0:INPUT_CHANNELS-1][0:KERNEL_SIZE-1][0:KERNEL_SIZE-1];
    reg [ACTIV_BITS-1:0] biases [0:NUM_FILTERS-1];
    reg [ACTIV_BITS-1:0] conv_result [0:INPUT_HEIGHT-1][0:INPUT_WIDTH-1][0:NUM_FILTERS-1];

    integer i, j, k, l, m, n, p, q;
```

```
// State machine
always @(posedge clk or negedge rst_n) begin
   if (!rst_n)
      state <= IDLE;
   else
      state <= next_state;
end

always @(*) begin
   next_state = state;
   case (state)
      IDLE: if (data_valid) next_state = LOAD_WEIGHTS;
      LOAD_WEIGHTS: if (psram_done) next_state = LOAD_BIASES;
      LOAD_BIASES: if (psram_done) next_state = CONV;
      CONV: next_state = STORE_RESULT;
      STORE_RESULT: if (psram_done) next_state = DONE;
      DONE: next_state = IDLE;
   endcase
end

// Control logic for PSRAM operations
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) begin
      addr <= 24'b0;
      psram_data <= 32'b0;
      psram_start <= 0;
      psram_rd_wr <= 0;
      psram_size <= 3'b010; // 4 bytes
      psram_qspi <= 0;
      psram_qpi <= 0;
      psram_short_cmd <= 0;
```

```
    end else begin
        psram_start <= 0;
        case (state)
            LOAD_WEIGHTS: begin
                addr <= weight_base_addr;
                psram_rd_wr <= 1;
                psram_start <= 1;
            end
            LOAD_BIASES: begin
                addr <= bias_base_addr;
                psram_rd_wr <= 1;
                psram_start <= 1;
            end
            STORE_RESULT: begin
                addr <= 24'hXXXXXX; // Address to store results in PSRAM
                psram_data <= conv_result[m][n][p];
                psram_rd_wr <= 0;
                psram_start <= 1;
            end
        endcase
    end
end

// Convolution operation
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        for (i = 0; i < INPUT_HEIGHT; i = i + 1)
            for (j = 0; j < INPUT_WIDTH; j = j + 1)
                for (k = 0; k < NUM_FILTERS; k = k + 1)
                    conv_result[i][j][k] <= 0;
    end else if (state == CONV) begin
```

```
      for (m = 0; m < INPUT_HEIGHT; m = m + 1) begin
        for (n = 0; n < INPUT_WIDTH; n = n + 1) begin
          for (p = 0; p < NUM_FILTERS; p = p + 1) begin
            conv_result[m][n][p] = biases[p];
            for (q = 0; q < INPUT_CHANNELS; q = q + 1) begin
              for (i = 0; i < KERNEL_SIZE; i = i + 1) begin
                for (j = 0; j < KERNEL_SIZE; j = j + 1) begin
                  if ((m + i) < INPUT_HEIGHT && (n + j) < INPUT_WIDTH) begin
                    conv_result[m][n][p] = conv_result[m][n][p] + weights[p][q][i][j] * data_in[((m + i) * INPUT_WIDTH * INPUT_CHANNELS + (n + j) * INPUT_CHANNELS + q) * ACTIV_BITS +: ACTIV_BITS];
                  end
                end
              end
            end
            // Apply ReLU activation
            conv_result[m][n][p] = (conv_result[m][n][p][ACTIV_BITS-1] == 0) ? conv_result[m][n][p] : 0;
          end
        end
      end
    end
  end

  assign data_out = {conv_result[0][0][0], conv_result[0][0][1], conv_result[0][0][2], conv_result[0][0][3], conv_result[0][0][4], conv_result[0][0][5], conv_result[0][0][6], conv_result[0][0][7]};
  assign data_out_valid = (state == DONE);
  assign done = (state == DONE); // Drive the done signal when state is DONE

endmodule
```

Submodule: fully_connected_psram

Similarly, let's add a done signal to the fully_connected_psram module:

verilog

Copy code

```
module fully_connected_psram #(
    parameter INPUT_SIZE = 320,
    parameter OUTPUT_SIZE = 64,
    parameter ACTIV_BITS = 16
) (
    input wire clk,
    input wire rst_n,
    input wire [INPUT_SIZE*ACTIV_BITS-1:0] data_in,
    input wire data_valid,
    output wire [OUTPUT_SIZE*ACTIV_BITS-1:0] data_out,
    output wire data_out_valid,
    output wire done,  // Add done signal

    // PSRAM interface signals
    output wire psram_sck,
    output wire psram_ce_n,
    inout wire [3:0] psram_d,
    output wire [3:0] psram_douten,

    // Base addresses for weights and biases
    input wire [23:0] weight_base_addr,
    input wire [23:0] bias_base_addr
);

    // State definitions
    localparam IDLE = 0,
               LOAD_WEIGHTS = 1,
               LOAD_BIASES = 2,
               FC = 3,

               STORE_RESULT = 4,

               DONE = 5;


    reg [3:0] state, next_state;

    reg [23:0] addr;

    reg [31:0] psram_data;

    reg psram_start, psram_rd_wr;

    reg [2:0] psram_size;

    reg psram_qspi, psram_qpi, psram_short_cmd;

    wire psram_done;


    wire [31:0] psram_data_out;


    // Instantiate PSRAM controller

    EF_PSRAM_CTRL_V2 psram_ctrl (

        .clk(clk),

        .rst_n(rst_n),

        .addr(addr),

        .data_i(psram_data),

        .data_o(psram_data_out),

        .size(psram_size),

        .start(psram_start),

        .done(psram_done),

        .wait_states(4'd8), // Adjust wait states as needed

        .cmd(8'hEB), // Example command

        .rd_wr(psram_rd_wr),

        .qspi(psram_qspi),

        .qpi(psram_qpi),

        .short_cmd(psram_short_cmd),

        .sck(psram_sck),
      .ce_n(psram_ce_n),
      .din(psram_d),
      .dout(psram_d),
      .douten(psram_douten)
    );

    reg [ACTIV_BITS-1:0] weights [0:OUTPUT_SIZE-1][0:INPUT_SIZE-1];
    reg [ACTIV_BITS-1:0] biases [0:OUTPUT_SIZE-1];
    reg [ACTIV_BITS-1:0] fc_result [0:OUTPUT_SIZE-1];

    integer i, j;

    // State machine
    always @(posedge clk or negedge rst_n) begin
      if (!rst_n)
        state <= IDLE;
      else
        state <= next_state;
    end

    always @(*) begin
      next_state = state;
      case (state)
        IDLE: if (data_valid) next_state = LOAD_WEIGHTS;
        LOAD_WEIGHTS: if (psram_done) next_state = LOAD_BIASES;
        LOAD_BIASES: if (psram_done) next_state = FC;
        FC: next_state = STORE_RESULT;
        STORE_RESULT: if (psram_done) next_state = DONE;
        DONE: next_state = IDLE;
      endcase
    end
```

```
// Control logic for PSRAM operations
always @(posedge clk or negedge rst_n) begin
  if (!rst_n) begin
    addr <= 24'b0;
    psram_data <= 32'b0;
    psram_start <= 0;
    psram_rd_wr <= 0;
    psram_size <= 3'b010; // 4 bytes
    psram_qspi <= 0;
    psram_qpi <= 0;
    psram_short_cmd <= 0;
  end else begin
    psram_start <= 0;
    case (state)
      LOAD_WEIGHTS: begin
        addr <= weight_base_addr;
        psram_rd_wr <= 1;
        psram_start <= 1;
      end
      LOAD_BIASES: begin
        addr <= bias_base_addr;
        psram_rd_wr <= 1;
        psram_start <= 1;
      end
      STORE_RESULT: begin
        addr <= 24'hXXXXXX; // Address to store results in PSRAM
        psram_data <= fc_result[i];
        psram_rd_wr <= 0;
        psram_start <= 1;
      end
```

```
            endcase
        end
    end

    // Fully connected operation
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            for (i = 0; i < OUTPUT_SIZE; i = i + 1)
                fc_result[i] <= 0;
        end else if (state == FC) begin
            for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin
                fc_result[i] = biases[i];
                for (j = 0; j < INPUT_SIZE; j = j + 1) begin
                    fc_result[i] = fc_result[i] + weights[i][j] * data_in[j*ACTIV_BITS +: ACTIV_BITS];
                end
                // Apply ReLU activation
                fc_result[i] = (fc_result[i][ACTIV_BITS-1] == 0) ? fc_result[i] : 0;
            end
        end
    end

    assign data_out = {fc_result[0], fc_result[1], fc_result[2], fc_result[3], fc_result[4], fc_result[5], fc_result[6], fc_result[7]};
    assign data_out_valid = (state == DONE);
    assign done = (state == DONE); // Drive the done signal when state is DONE

endmodule
```

Updated cnn_kws_accel Module

Now that the submodules have the done signal defined and driven, we can correctly connect them in the cnn_kws_accel module:

verilog

Copy code

```
module cnn_kws_accel (
    input wire clk,
    input wire rst_n,
    input wire start,
    output wire done,
    // PSRAM signals
    output wire psram_sck,
    output wire psram_ce_n,
    inout wire [3:0] psram_d,
    output wire [3:0] psram_douten
);

    // Internal signals for PSRAM
    wire conv1_psram_sck, fc1_psram_sck, maxpool_psram_sck, softmax_psram_sck;
    wire conv1_psram_ce_n, fc1_psram_ce_n, maxpool_psram_ce_n, softmax_psram_ce_n;
    wire [3:0] conv1_psram_douten, fc1_psram_douten, maxpool_psram_douten, softmax_psram_douten;
    wire [3:0] conv1_psram_d, fc1_psram_d, maxpool_psram_d, softmax_psram_d;

    // PSRAM data output
    wire [3:0] psram_d_in;

    // State definitions
    typedef enum logic [2:0] {
        IDLE,
        CONV1,
        FC1,
        MAXPOOL,
        SOFTMAX
    } state_t;
```

```
state_t state, next_state;


// State machine
always @(posedge clk or negedge rst_n) begin
   if (!rst_n)
      state <= IDLE;
   else
      state <= next_state;
end


always @* begin
   next_state = state;
   case (state)
      IDLE: if (start) next_state = CONV1;
      CONV1: if (conv1_done) next_state = FC1;
      FC1: if (fc1_done) next_state = MAXPOOL;
      MAXPOOL: if (maxpool_done) next_state = SOFTMAX;
      SOFTMAX: if (softmax_done) next_state = IDLE;
      default: next_state = IDLE;
   endcase
end


// Multiplexer for PSRAM control signals
assign psram_sck = (state == CONV1) ? conv1_psram_sck :
                (state == FC1) ? fc1_psram_sck :
                (state == MAXPOOL) ? maxpool_psram_sck :
                softmax_psram_sck;


assign psram_ce_n = (state == CONV1) ? conv1_psram_ce_n :
                 (state == FC1) ? fc1_psram_ce_n :
```

```
                 (state == MAXPOOL) ? maxpool_psram_ce_n :
                 softmax_psram_ce_n;

assign psram_douten = (state == CONV1) ? conv1_psram_douten :
                   (state == FC1) ? fc1_psram_douten :
                   (state == MAXPOOL) ? maxpool_psram_douten :
                   softmax_psram_douten;

assign psram_d_in = (state == CONV1) ? conv1_psram_d :
                 (state == FC1) ? fc1_psram_d :
                 (state == MAXPOOL) ? maxpool_psram_d :
                 softmax_psram_d;

// Tristate buffer for psram_d
assign psram_d = psram_douten ? psram_d_in : 4'bz;

// Instantiate the convolution and fully connected modules with PSRAM
wire [23:0] conv1_weight_base_addr = 24'h000000;
wire [23:0] conv1_bias_base_addr = 24'h000100;
wire [23:0] fc1_weight_base_addr = 24'h000200;
wire [23:0] fc1_bias_base_addr = 24'h000300;
wire [23:0] maxpool_input_addr = 24'h000400;
wire [23:0] maxpool_output_addr = 24'h000500;
wire [23:0] softmax_input_addr = 24'h000600;
wire [23:0] softmax_output_addr = 24'h000700;

wire conv1_done, fc1_done, maxpool_done, softmax_done;
wire conv1_data_valid = (state == CONV1);
wire fc1_data_valid = (state == FC1);
wire maxpool_data_valid = (state == MAXPOOL);
wire softmax_start = (state == SOFTMAX);
wire [320*16-1:0] conv1_data_out;

wire conv1_data_out_valid;

wire [64*16-1:0] fc1_data_out;

wire fc1_data_out_valid;

wire [10*16-1:0] maxpool_data_out;

wire maxpool_data_out_valid;

wire [10*8-1:0] softmax_data_out;

wire softmax_data_out_valid;


conv2d_psram #(

    .INPUT_WIDTH(40),

    .INPUT_HEIGHT(1),

    .INPUT_CHANNELS(1),

    .KERNEL_SIZE(3),

    .NUM_FILTERS(8),

    .PADDING(1),

    .ACTIV_BITS(16)

) conv1 (

    .clk(clk),

    .rst_n(rst_n),

    .data_in(/* Provide data input */),

    .data_valid(conv1_data_valid),

    .data_out(conv1_data_out),

    .data_out_valid(conv1_data_out_valid),

    .psram_sck(conv1_psram_sck),

    .psram_ce_n(conv1_psram_ce_n),

    .psram_d(conv1_psram_d),

    .psram_douten(conv1_psram_douten),

    .weight_base_addr(conv1_weight_base_addr),

    .bias_base_addr(conv1_bias_base_addr),
    .done(conv1_done)
);

fully_connected_psram #(
    .INPUT_SIZE(320),
    .OUTPUT_SIZE(64),
    .ACTIV_BITS(16)
) fc1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv1_data_out),
    .data_valid(conv1_data_out_valid),
    .data_out(fc1_data_out),
    .data_out_valid(fc1_data_out_valid),
    .psram_sck(fc1_psram_sck),
    .psram_ce_n(fc1_psram_ce_n),
    .psram_d(fc1_psram_d),
    .psram_douten(fc1_psram_douten),
    .weight_base_addr(fc1_weight_base_addr),
    .bias_base_addr(fc1_bias_base_addr),
    .done(fc1_done)
);

maxpool_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(8),
    .KERNEL_SIZE(2),
    .STRIDE(2),
    .ACTIV_BITS(16),
    .ADDR_WIDTH(24)
```

```
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .start(maxpool_data_valid),
    .input_addr(maxpool_input_addr),
    .output_addr(maxpool_output_addr),
    .done(maxpool_done),
    .psram_sck(maxpool_psram_sck),
    .psram_ce_n(maxpool_psram_ce_n),
    .psram_d(maxpool_psram_d),
    .psram_douten(maxpool_psram_douten)
);

softmax_psram #(
    .INPUT_SIZE(10),
    .ACTIV_BITS(8),
    .ADDR_WIDTH(24)
) softmax (
    .clk(clk),
    .rst_n(rst_n),
    .start(softmax_start),
    .input_addr(softmax_input_addr),
    .output_addr(softmax_output_addr),
    .size(3'b010),
    .cmd(8'hEB),
    .rd_wr(1'b1),
    .qspi(1'b0),
    .qpi(1'b0),
    .short_cmd(1'b0),
    .done(softmax_done),
    .psram_sck(softmax_psram_sck),
        .psram_ce_n(softmax_psram_ce_n),
        .psram_d(softmax_psram_d),
        .psram_douten(softmax_psram_douten)
    );

    // Assign overall done signal
    assign done = (state == SOFTMAX) && softmax_done;

endmodule
```

Summary of Changes:

Added done signal to each submodule and connected it in the cnn_kws_accel module.

Ensured done signal is driven correctly within each submodule.

Next Steps:

Verify signal connections within each submodule to ensure they handle the PSRAM signals correctly.

Run simulations to validate the design and ensure proper functionality.

If you encounter additional errors or need further adjustments, please provide specific details, and I'll assist you in resolving them.

I need to instantiate my top module "cnn_kws_accel" inside the top caravel top module "user_proj_wrapper". Here is a demo configuration of "user_proj_wrapper" "// SPDX-FileCopyrightText: 2020 Efabless Corporation

```
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//      http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
// SPDX-License-Identifier: Apache-2.0

`default_nettype none
/*
 *-------------------------------------------------------------
 *
 * user_project_wrapper
 *
 * This wrapper enumerates all of the pins available to the
 * user for the user project.
 *
 * An example user project is provided in this wrapper.  The
 * example should be removed and replaced with the actual
 * user project.
 *
 *-------------------------------------------------------------
 */

module user_project_wrapper #(
    parameter BITS = 32
) (
`ifdef USE_POWER_PINS
    inout vdda1,  // User area 1 3.3V supply
    inout vdda2,  // User area 2 3.3V supply
    inout vssa1,  // User area 1 analog ground
    inout vssa2,  // User area 2 analog ground
    inout vccd1,  // User area 1 1.8V supply
    inout vccd2,  // User area 2 1.8v supply
    inout vssd1,  // User area 1 digital ground
    inout vssd2,  // User area 2 digital ground
```

```
`endif

    // Wishbone Slave ports (WB MI A)
    input wb_clk_i,
    input wb_rst_i,
    input wbs_stb_i,
    input wbs_cyc_i,
    input wbs_we_i,
    input [3:0] wbs_sel_i,
    input [31:0] wbs_dat_i,
    input [31:0] wbs_adr_i,
    output wbs_ack_o,
    output [31:0] wbs_dat_o,

    // Logic Analyzer Signals
    input  [127:0] la_data_in,
    output [127:0] la_data_out,
    input  [127:0] la_oenb,

    // IOs
    input  [`MPRJ_IO_PADS-1:0] io_in,
    output [`MPRJ_IO_PADS-1:0] io_out,
    output [`MPRJ_IO_PADS-1:0] io_oeb,

    // Analog (direct connection to GPIO pad---use with caution)
    // Note that analog I/O is not available on the 7 lowest-numbered
    // GPIO pads, and so the analog_io indexing is offset from the
    // GPIO indexing by 7 (also upper 2 GPIOs do not have analog_io).
    inout [`MPRJ_IO_PADS-10:0] analog_io,

    // Independent clock (on independent integer divider)
```

```
   input   user_clock2,

   // User maskable interrupt signals
   output [2:0] user_irq
);

/*--------------------------------------*/
/* User project is instantiated  here   */
/*--------------------------------------*/

user_proj_example mprj (
`ifdef USE_POWER_PINS
            .vccd1(vccd1),   // User area 1 1.8V power
            .vssd1(vssd1),   // User area 1 digital ground
`endif

   .wb_clk_i(wb_clk_i),
   .wb_rst_i(wb_rst_i),

   // MGMT SoC Wishbone Slave

   .wbs_cyc_i(wbs_cyc_i),
   .wbs_stb_i(wbs_stb_i),
   .wbs_we_i(wbs_we_i),
   .wbs_sel_i(wbs_sel_i),
   .wbs_adr_i(wbs_adr_i),
   .wbs_dat_i(wbs_dat_i),
   .wbs_ack_o(wbs_ack_o),
   .wbs_dat_o(wbs_dat_o),

   // Logic Analyzer
```

```
    .la_data_in(la_data_in),
    .la_data_out(la_data_out),
    .la_oenb (la_oenb),

    // IO Pads

    .io_in ({io_in[37:30],io_in[7:0]}),
    .io_out({io_out[37:30],io_out[7:0]}),
    .io_oeb({io_oeb[37:30],io_oeb[7:0]}),

    // IRQ
    .irq(user_irq)
);

endmodule	// user_project_wrapper

`default_nettype wire". My top design cnn_kws_accel module code is : "module cnn_kws_accel (
    input wire clk,
    input wire rst_n,
    input wire start,
    output wire done,
    // PSRAM signals
    output wire psram_sck,
    output wire psram_ce_n,
    inout wire [3:0] psram_d,
    output wire [3:0] psram_douten
);

    // Internal signals for PSRAM
    wire conv1_psram_sck, fc1_psram_sck, maxpool_psram_sck, softmax_psram_sck;
    wire conv1_psram_ce_n, fc1_psram_ce_n, maxpool_psram_ce_n, softmax_psram_ce_n;
    wire [3:0] conv1_psram_douten, fc1_psram_douten, maxpool_psram_douten, softmax_psram_douten;
    wire [3:0] conv1_psram_d, fc1_psram_d, maxpool_psram_d, softmax_psram_d;

    // PSRAM data output
    wire [3:0] psram_d_in;

    // State definitions
    typedef enum logic [2:0] {
        IDLE,
        CONV1,
        FC1,
        MAXPOOL,
        SOFTMAX
    } state_t;

    state_t state, next_state;

    // State machine
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            state <= IDLE;
        else
            state <= next_state;
    end

    always @* begin
        next_state = state;
        case (state)
            IDLE: if (start) next_state = CONV1;
    CONV1: if (conv1_done) next_state = FC1;
    FC1: if (fc1_done) next_state = MAXPOOL;
    MAXPOOL: if (maxpool_done) next_state = SOFTMAX;
    SOFTMAX: if (softmax_done) next_state = IDLE;
    default: next_state = IDLE;
  endcase
end

// Multiplexer for PSRAM control signals
assign psram_sck = (state == CONV1) ? conv1_psram_sck :
         (state == FC1) ? fc1_psram_sck :
         (state == MAXPOOL) ? maxpool_psram_sck :
         softmax_psram_sck;

assign psram_ce_n = (state == CONV1) ? conv1_psram_ce_n :
          (state == FC1) ? fc1_psram_ce_n :
          (state == MAXPOOL) ? maxpool_psram_ce_n :
          softmax_psram_ce_n;

assign psram_douten = (state == CONV1) ? conv1_psram_douten :
           (state == FC1) ? fc1_psram_douten :
           (state == MAXPOOL) ? maxpool_psram_douten :
           softmax_psram_douten;

assign psram_d_in = (state == CONV1) ? conv1_psram_d :
          (state == FC1) ? fc1_psram_d :
          (state == MAXPOOL) ? maxpool_psram_d :
          softmax_psram_d;

// Tristate buffer for psram_d
assign psram_d = psram_douten ? psram_d_in : 4'bz;
```

```
// Instantiate the convolution and fully connected modules with PSRAM
wire [23:0] conv1_weight_base_addr = 24'h000000;
wire [23:0] conv1_bias_base_addr = 24'h000100;
wire [23:0] fc1_weight_base_addr = 24'h000200;
wire [23:0] fc1_bias_base_addr = 24'h000300;
wire [23:0] maxpool_input_addr = 24'h000400;
wire [23:0] maxpool_output_addr = 24'h000500;
wire [23:0] softmax_input_addr = 24'h000600;
wire [23:0] softmax_output_addr = 24'h000700;

wire conv1_done, fc1_done, maxpool_done, softmax_done;
wire conv1_data_valid = (state == CONV1);
wire fc1_data_valid = (state == FC1);
wire maxpool_data_valid = (state == MAXPOOL);
wire softmax_start = (state == SOFTMAX);

wire [320*16-1:0] conv1_data_out;
wire conv1_data_out_valid;
wire [64*16-1:0] fc1_data_out;
wire fc1_data_out_valid;
wire [10*16-1:0] maxpool_data_out;
wire maxpool_data_out_valid;
wire [10*8-1:0] softmax_data_out;
wire softmax_data_out_valid;

conv2d_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(1),
    .KERNEL_SIZE(3),
    .NUM_FILTERS(8),
    .PADDING(1),
    .ACTIV_BITS(16)
) conv1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(/* Provide data input */),
    .data_valid(conv1_data_valid),
    .data_out(conv1_data_out),
    .data_out_valid(conv1_data_out_valid),
    .psram_sck(conv1_psram_sck),
    .psram_ce_n(conv1_psram_ce_n),
    .psram_d(conv1_psram_d),
    .psram_douten(conv1_psram_douten),
    .weight_base_addr(conv1_weight_base_addr),
    .bias_base_addr(conv1_bias_base_addr),
    .done(conv1_done)
);

fully_connected_psram #(
    .INPUT_SIZE(320),
    .OUTPUT_SIZE(64),
    .ACTIV_BITS(16)
) fc1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv1_data_out),
    .data_valid(conv1_data_out_valid),
    .data_out(fc1_data_out),
    .data_out_valid(fc1_data_out_valid),
    .psram_sck(fc1_psram_sck),
    .psram_ce_n(fc1_psram_ce_n),
    .psram_d(fc1_psram_d),
    .psram_douten(fc1_psram_douten),
    .weight_base_addr(fc1_weight_base_addr),
    .bias_base_addr(fc1_bias_base_addr),
    .done(fc1_done)
);

maxpool_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(8),
    .KERNEL_SIZE(2),
    .STRIDE(2),
    .ACTIV_BITS(16),
    .ADDR_WIDTH(24)
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .start(maxpool_data_valid),
    .input_addr(maxpool_input_addr),
    .output_addr(maxpool_output_addr),
    .done(maxpool_done),
    .psram_sck(maxpool_psram_sck),
    .psram_ce_n(maxpool_psram_ce_n),
    .psram_d(maxpool_psram_d),
    .psram_douten(maxpool_psram_douten)
);

softmax_psram #(
    .INPUT_SIZE(10),
        .ACTIV_BITS(8),
        .ADDR_WIDTH(24)
    ) softmax (
        .clk(clk),
        .rst_n(rst_n),
        .start(softmax_start),
        .input_addr(softmax_input_addr),
        .output_addr(softmax_output_addr),
        .size(3'b010),
        .cmd(8'hEB),
        .rd_wr(1'b1),
        .qspi(1'b0),
        .qpi(1'b0),
        .short_cmd(1'b0),
        .done(softmax_done),
        .psram_sck(softmax_psram_sck),
        .psram_ce_n(softmax_psram_ce_n),
        .psram_d(softmax_psram_d),
        .psram_douten(softmax_psram_douten)
    );

    // Assign overall done signal
    assign done = (state == SOFTMAX) && softmax_done;

endmodule
```

".

2/2

ChatGPT

To instantiate your cnn_kws_accel module inside the user_proj_wrapper module, you need to add the appropriate connections and signals in the user_proj_wrapper. Here's how you can modify the user_proj_wrapper to include your cnn_kws_accel module.

Steps to Modify user_proj_wrapper

Declare the signals required for the cnn_kws_accel module.

Instantiate the cnn_kws_accel module.

Connect the signals appropriately.

Updated user_proj_wrapper Module

verilog

Copy code

```
`default_nettype none
/*
 *-------------------------------------------------------------
 *
 * user_project_wrapper
 *
 * This wrapper enumerates all of the pins available to the
 * user for the user project.
 *
 * An example user project is provided in this wrapper.  The
 * example should be removed and replaced with the actual
 * user project.
 *
 *-------------------------------------------------------------
 */

module user_project_wrapper #(
   parameter BITS = 32
) (
`ifdef USE_POWER_PINS
   inout vdda1,  // User area 1 3.3V supply
```

```
    inout vdda2,   // User area 2 3.3V supply
    inout vssa1,   // User area 1 analog ground
    inout vssa2,   // User area 2 analog ground
    inout vccd1,   // User area 1 1.8V supply
    inout vccd2,   // User area 2 1.8v supply
    inout vssd1,   // User area 1 digital ground
    inout vssd2,   // User area 2 digital ground
`endif

    // Wishbone Slave ports (WB MI A)
    input wb_clk_i,
    input wb_rst_i,
    input wbs_stb_i,
    input wbs_cyc_i,
    input wbs_we_i,
    input [3:0] wbs_sel_i,
    input [31:0] wbs_dat_i,
    input [31:0] wbs_adr_i,
    output wbs_ack_o,
    output [31:0] wbs_dat_o,

    // Logic Analyzer Signals
    input  [127:0] la_data_in,
    output [127:0] la_data_out,
    input  [127:0] la_oenb,

    // IOs
    input  [`MPRJ_IO_PADS-1:0] io_in,
    output [`MPRJ_IO_PADS-1:0] io_out,
    output [`MPRJ_IO_PADS-1:0] io_oeb,
    // Analog (direct connection to GPIO pad---use with caution)
    // Note that analog I/O is not available on the 7 lowest-numbered
    // GPIO pads, and so the analog_io indexing is offset from the
    // GPIO indexing by 7 (also upper 2 GPIOs do not have analog_io).
    inout [`MPRJ_IO_PADS-10:0] analog_io,

    // Independent clock (on independent integer divider)
    input   user_clock2,

    // User maskable interrupt signals
    output [2:0] user_irq
);

/*--------------------------------------*/
/* User project is instantiated  here   */
/*--------------------------------------*/

// Signals for cnn_kws_accel
wire psram_sck;
wire psram_ce_n;
wire [3:0] psram_d;
wire [3:0] psram_douten;
wire cnn_done;
wire cnn_start;

// You can use some GPIO pins to control the start signal and monitor the done signal.
// Here we use io_in[0] as the start signal and io_out[0] as the done signal.
assign cnn_start = io_in[0];
assign io_out[0] = cnn_done;

cnn_kws_accel u_cnn_kws_accel (
    .clk(wb_clk_i),
    .rst_n(~wb_rst_i),
    .start(cnn_start),
    .done(cnn_done),
    .psram_sck(psram_sck),
    .psram_ce_n(psram_ce_n),
    .psram_d(psram_d),
    .psram_douten(psram_douten)
);

endmodule       // user_project_wrapper

`default_nettype wire
```

Explanation of Changes:

Declare signals for cnn_kws_accel:

Added psram_sck, psram_ce_n, psram_d, psram_douten, cnn_done, and cnn_start.

Assign cnn_start and cnn_done to GPIO pins:

Used io_in[0] as the start signal and io_out[0] to monitor the done signal of the cnn_kws_accel.

Instantiate cnn_kws_accel:

Connected the appropriate signals including the clock, reset, and the PSRAM signals.

Additional Notes:

Ensure that you have the PSRAM and its interface correctly set up and connected as needed.

If the PSRAM signals need to be connected to specific IO pins, you can map them accordingly in the io_in and io_out arrays.

The example assumes that the PSRAM signals are correctly managed inside the cnn_kws_accel module.

This setup should integrate your cnn_kws_accel module into the user_proj_wrapper and allow you to control it using the Caravel environment. Make sure to verify the signal connections and test the overall functionality.

the audio_sample should be the input of cnn_kws_accel module. Also the mfcc_feature is the input of conv2d_psram.

ChatGPT

Let's update the integration of your cnn_kws_accel module inside the user_proj_wrapper module to include the audio_sample input, and ensure that the mfcc_feature is correctly fed into the conv2d_psram module.

cnn_kws_accel Module Update

First, update the cnn_kws_accel module to accept audio_sample as an input. Then, pass this audio_sample through an MFCC feature extractor to produce mfcc_feature, which will be the input to the conv2d_psram.

verilog

Copy code

```
module cnn_kws_accel (
    input wire clk,
    input wire rst_n,
    input wire start,
    input wire [15:0] audio_sample, // Audio sample input
    output wire done,
    // PSRAM signals
    output wire psram_sck,
    output wire psram_ce_n,
    inout wire [3:0] psram_d,
    output wire [3:0] psram_douten
);

    // Internal signals for PSRAM
    wire conv1_psram_sck, fc1_psram_sck, maxpool_psram_sck, softmax_psram_sck;
    wire conv1_psram_ce_n, fc1_psram_ce_n, maxpool_psram_ce_n, softmax_psram_ce_n;
    wire [3:0] conv1_psram_douten, fc1_psram_douten, maxpool_psram_douten, softmax_psram_douten;
    wire [3:0] conv1_psram_d, fc1_psram_d, maxpool_psram_d, softmax_psram_d;

    // PSRAM data output
    wire [3:0] psram_d_in;

    // State definitions
    typedef enum logic [2:0] {
      IDLE,
      CONV1,
      FC1,
      MAXPOOL,
      SOFTMAX
    } state_t;

    state_t state, next_state;

    // State machine
    always @(posedge clk or negedge rst_n) begin
      if (!rst_n)
        state <= IDLE;
      else
        state <= next_state;
    end

    always @* begin
      next_state = state;
      case (state)
        IDLE: if (start) next_state = CONV1;
        CONV1: if (conv1_done) next_state = FC1;
        FC1: if (fc1_done) next_state = MAXPOOL;
        MAXPOOL: if (maxpool_done) next_state = SOFTMAX;
        SOFTMAX: if (softmax_done) next_state = IDLE;
        default: next_state = IDLE;
      endcase
    end

    // Multiplexer for PSRAM control signals
    assign psram_sck = (state == CONV1) ? conv1_psram_sck :
                       (state == FC1) ? fc1_psram_sck :
                       (state == MAXPOOL) ? maxpool_psram_sck :
                       softmax_psram_sck;

    assign psram_ce_n = (state == CONV1) ? conv1_psram_ce_n :
                        (state == FC1) ? fc1_psram_ce_n :
                        (state == MAXPOOL) ? maxpool_psram_ce_n :
                        softmax_psram_ce_n;

    assign psram_douten = (state == CONV1) ? conv1_psram_douten :
                          (state == FC1) ? fc1_psram_douten :
                          (state == MAXPOOL) ? maxpool_psram_douten :
                          softmax_psram_douten;

    assign psram_d_in = (state == CONV1) ? conv1_psram_d :
                        (state == FC1) ? fc1_psram_d :
                        (state == MAXPOOL) ? maxpool_psram_d :
                        softmax_psram_d;

    // Tristate buffer for psram_d
    assign psram_d = psram_douten ? psram_d_in : 4'bz;

    // Instantiate the MFCC feature extractor and other modules with PSRAM
    wire [319:0] mfcc_feature; // Assuming MFCC output size is 320 bits (20 coefficients * 16 bits)
```

```
mfcc_extractor u_mfcc (
    .clk(clk),
    .rst_n(rst_n),
    .audio_sample(audio_sample),
    .mfcc_feature(mfcc_feature)
);

wire [23:0] conv1_weight_base_addr = 24'h000000;
wire [23:0] conv1_bias_base_addr = 24'h000100;
wire [23:0] fc1_weight_base_addr = 24'h000200;
wire [23:0] fc1_bias_base_addr = 24'h000300;
wire [23:0] maxpool_input_addr = 24'h000400;
wire [23:0] maxpool_output_addr = 24'h000500;
wire [23:0] softmax_input_addr = 24'h000600;
wire [23:0] softmax_output_addr = 24'h000700;

wire conv1_done, fc1_done, maxpool_done, softmax_done;
wire conv1_data_valid = (state == CONV1);
wire fc1_data_valid = (state == FC1);
wire maxpool_data_valid = (state == MAXPOOL);
wire softmax_start = (state == SOFTMAX);

wire [320*16-1:0] conv1_data_out;
wire conv1_data_out_valid;
wire [64*16-1:0] fc1_data_out;
wire fc1_data_out_valid;
wire [10*16-1:0] maxpool_data_out;
wire maxpool_data_out_valid;
wire [10*8-1:0] softmax_data_out;
wire softmax_data_out_valid;
```

```
conv2d_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(1),
    .KERNEL_SIZE(3),
    .NUM_FILTERS(8),
    .PADDING(1),
    .ACTIV_BITS(16)
) conv1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(mfcc_feature), // MFCC feature as input
    .data_valid(conv1_data_valid),
    .data_out(conv1_data_out),
    .data_out_valid(conv1_data_out_valid),
    .psram_sck(conv1_psram_sck),
    .psram_ce_n(conv1_psram_ce_n),
    .psram_d(conv1_psram_d),
    .psram_douten(conv1_psram_douten),
    .weight_base_addr(conv1_weight_base_addr),
    .bias_base_addr(conv1_bias_base_addr),
    .done(conv1_done)
);

fully_connected_psram #(
    .INPUT_SIZE(320),
    .OUTPUT_SIZE(64),
    .ACTIV_BITS(16)
) fc1 (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv1_data_out),
    .data_valid(conv1_data_out_valid),
    .data_out(fc1_data_out),
    .data_out_valid(fc1_data_out_valid),
    .psram_sck(fc1_psram_sck),
    .psram_ce_n(fc1_psram_ce_n),
    .psram_d(fc1_psram_d),
    .psram_douten(fc1_psram_douten),
    .weight_base_addr(fc1_weight_base_addr),
    .bias_base_addr(fc1_bias_base_addr),
    .done(fc1_done)
);

maxpool_psram #(
    .INPUT_WIDTH(40),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(8),
    .KERNEL_SIZE(2),
    .STRIDE(2),
    .ACTIV_BITS(16),
    .ADDR_WIDTH(24)
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .start(maxpool_data_valid),
    .input_addr(maxpool_input_addr),
    .output_addr(maxpool_output_addr),
    .done(maxpool_done),
    .psram_sck(maxpool_psram_sck),
    .psram_ce_n(maxpool_psram_ce_n),
        .psram_d(maxpool_psram_d),
        .psram_douten(maxpool_psram_douten)
    );

    softmax_psram #(
        .INPUT_SIZE(10),
        .ACTIV_BITS(8),
        .ADDR_WIDTH(24)
    ) softmax (
        .clk(clk),
        .rst_n(rst_n),
        .start(softmax_start),
        .input_addr(softmax_input_addr),
        .output_addr(softmax_output_addr),
        .size(3'b010),
        .cmd(8'hEB),
        .rd_wr(1'b1),
        .qspi(1'b0),
        .qpi(1'b0),
        .short_cmd(1'b0),
        .done(softmax_done),
        .psram_sck(softmax_psram_sck),
        .psram_ce_n(softmax_psram_ce_n),
        .psram_d(softmax_psram_d),
        .psram_douten(softmax_psram_douten)
    );

    // Assign overall done signal
    assign done = (state == SOFTMAX) && softmax_done;

endmodule
```

Updated user_project_wrapper Module

Now, update the user_project_wrapper module to provide audio_sample to the cnn_kws_accel module:

verilog

Copy code

```
`default_nettype none
/*
 *-------------------------------------------------------------
 *
 * user_project_wrapper
 *
 * This wrapper enumerates all of the pins available to the
 * user for the user project.
 *
 * An example user project is provided in this wrapper.  The
 * example should be removed and replaced with the actual
 * user project.
 *
 *-------------------------------------------------------------
 */

module user_project_wrapper #(
    parameter BITS = 32
) (
`ifdef USE_POWER_PINS
    inout vdda1,   // User area 1 3.3V supply
    inout vdda2,   // User area 2 3.3V supply
    inout vssa1,   // User area 1 analog ground
    inout vssa2,   // User area 2 analog ground
    inout vccd1,   // User area 1 1.8V supply
    inout vccd2,   // User area 2 1.8v supply
    inout vssd1,   // User area 1 digital ground
    inout vssd2,   // User area 2 digital ground
`endif

    // Wishbone Slave ports (WB MI A)
    input wb_clk_i,
    input wb_rst_i,
    input wbs_stb_i,
    input wbs_cyc_i,
    input wbs_we_i,
    input [3:0] wbs_sel_i,
    input [31:0] wbs_dat_i,
    input [31:0] wbs_adr_i,
    output wbs_ack_o,
    output [31:0] wbs_dat_o,

    // Logic Analyzer Signals
    input  [127:0] la_data_in,
    output [127:0] la_data_out,
    input  [127:0] la_oenb,

    // IOs
    input  [`MPRJ_IO_PADS-1:0] io_in,
    output [`MPRJ_IO_PADS-1:0] io_out,
    output [`MPRJ_IO_PADS-1:0] io_oeb,

    // Analog (direct connection to GPIO pad---use with caution)
    // Note that analog I/O is not available on the 7 lowest-numbered
    // GPIO pads, and so the analog_io indexing is offset from the
    // GPIO indexing by 7 (also upper 2 GPIOs do not have analog_io).
    inout [`MPRJ_IO_PADS-10:0] analog_io,

    // Independent clock (on independent integer divider)
    input   user_clock2,

    // User maskable interrupt signals
    output [2:0] user_irq
);

/*--------------------------------------*/
/* User project is instantiated  here   */
/*--------------------------------------*/

// Signals for cnn_kws_accel
wire psram_sck;
wire psram_ce_n;
wire [3:0] psram_d;
wire [3:0] psram_douten;
wire cnn_done;
wire cnn_start;
wire [15:0] audio_sample; // 16-bit audio sample input

// You can use some GPIO pins to control the start signal and monitor the done signal.
// Here we use io_in[0] as the start signal, io_in[16:1] for audio sample, and io_out[0] as the done signal.
assign cnn_start = io_in[0];
assign audio_sample = io_in[16:1];
assign io_out[0] = cnn_done;

cnn_kws_accel u_cnn_kws_accel (
    .clk(wb_clk_i),
    .rst_n(~wb_rst_i),
    .start(cnn_start),
    .audio_sample(audio_sample),
    .done(cnn_done),
    .psram_sck(psram_sck),
    .psram_ce_n(psram_ce_n),
    .psram_d(psram_d),
    .psram_douten(psram_douten)
);

endmodule	// user_project_wrapper

`default_nettype wire
```

Explanation of Changes:

Declare the audio_sample signal as a 16-bit input.

Assign audio_sample from the io_in array.

Used io_in[16:1] to capture the 16-bit audio sample.

Used io_in[0] as the start signal and io_out[0] to monitor the done signal of the cnn_kws_accel.

Instantiate cnn_kws_accel with the audio_sample input connected.

This setup ensures that the audio_sample is correctly fed into the cnn_kws_accel module, and the MFCC features are correctly passed to the conv2d_psram module. Make sure to verify the signal connections and test the overall functionality.

2/2

Help me solve this compilation error "Warning: Resizing cell port cnn_kws_accel.conv1.data_in from 320 bits to 640 bits.

Warning: Resizing cell port cnn_kws_accel.mfcc.mfcc_feature from 320 bits to 640 bits."

ChatGPT

The compilation error indicates a mismatch in the bit widths of the data_in port of the conv2d_psram module and the mfcc_feature signal generated by the mfcc_extractor module. The warnings suggest that the tools are resizing these signals, which is not ideal.

To resolve this, we need to ensure that the bit widths are correctly matched across the modules.

Steps to Resolve the Issue

Check the mfcc_extractor output width.

Ensure the mfcc_feature matches the expected input width of conv2d_psram.

Updated cnn_kws_accel Module

First, let's ensure the mfcc_extractor outputs the correct width. Then, make sure the mfcc_feature signal matches the data_in width of the conv2d_psram module.

Example MFCC Extractor (Assuming 40 MFCC coefficients, 16 bits each)

If the MFCC extractor produces 40 coefficients, each 16 bits wide, the total width should be

40

×

16

=

640

40×16=640 bits.

mfcc_extractor Module (Example)

Here's an example mfcc_extractor module definition:

verilog

Copy code

```
module mfcc_extractor (
    input wire clk,
    input wire rst_n,
    input wire [15:0] audio_sample,
    output wire [639:0] mfcc_feature  // 40 coefficients, each 16 bits
);
```

```
    // Implement the MFCC extraction logic here
    // For example purposes, we assume it generates 40 coefficients, each 16 bits
    // (Replace this with actual MFCC extraction implementation)

    reg [639:0] mfcc_feature_reg;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            mfcc_feature_reg <= 640'b0;
        else begin
            // Dummy example logic: replace with actual MFCC extraction logic
            // Here we just shift in the audio sample for illustration
            mfcc_feature_reg <= {mfcc_feature_reg[623:0], audio_sample};
        end
    end

    assign mfcc_feature = mfcc_feature_reg;

endmodule
```

Updated cnn_kws_accel Module

Update the cnn_kws_accel module to match the bit widths correctly:

verilog

Copy code

```
module cnn_kws_accel (
    input wire clk,
    input wire rst_n,
    input wire start,
    input wire [15:0] audio_sample, // Audio sample input
    output wire done,
    // PSRAM signals
    output wire psram_sck,
    output wire psram_ce_n,
    inout wire [3:0] psram_d,
    output wire [3:0] psram_douten
);

    // Internal signals for PSRAM
    wire conv1_psram_sck, fc1_psram_sck, maxpool_psram_sck, softmax_psram_sck;
    wire conv1_psram_ce_n, fc1_psram_ce_n, maxpool_psram_ce_n, softmax_psram_ce_n;
    wire [3:0] conv1_psram_douten, fc1_psram_douten, maxpool_psram_douten,
softmax_psram_douten;
    wire [3:0] conv1_psram_d, fc1_psram_d, maxpool_psram_d, softmax_psram_d;

    // PSRAM data output
    wire [3:0] psram_d_in;

    // State definitions
    typedef enum logic [2:0] {
        IDLE,
        CONV1,
        FC1,
        MAXPOOL,
        SOFTMAX
    } state_t;

    state_t state, next_state;

    // State machine
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            state <= IDLE;
```

```
  else
    state <= next_state;
end

always @* begin
  next_state = state;
  case (state)
    IDLE: if (start) next_state = CONV1;
    CONV1: if (conv1_done) next_state = FC1;
    FC1: if (fc1_done) next_state = MAXPOOL;
    MAXPOOL: if (maxpool_done) next_state = SOFTMAX;
    SOFTMAX: if (softmax_done) next_state = IDLE;
    default: next_state = IDLE;
  endcase
end

// Multiplexer for PSRAM control signals
assign psram_sck = (state == CONV1) ? conv1_psram_sck :
          (state == FC1) ? fc1_psram_sck :
          (state == MAXPOOL) ? maxpool_psram_sck :
          softmax_psram_sck;

assign psram_ce_n = (state == CONV1) ? conv1_psram_ce_n :
           (state == FC1) ? fc1_psram_ce_n :
           (state == MAXPOOL) ? maxpool_psram_ce_n :
           softmax_psram_ce_n;

assign psram_douten = (state == CONV1) ? conv1_psram_douten :
            (state == FC1) ? fc1_psram_douten :
            (state == MAXPOOL) ? maxpool_psram_douten :
            softmax_psram_douten;
```

```
assign psram_d_in = (state == CONV1) ? conv1_psram_d :
                (state == FC1) ? fc1_psram_d :
                (state == MAXPOOL) ? maxpool_psram_d :
                softmax_psram_d;

// Tristate buffer for psram_d
assign psram_d = psram_douten ? psram_d_in : 4'bz;

// Instantiate the MFCC feature extractor and other modules with PSRAM
wire [639:0] mfcc_feature; // Assuming MFCC output size is 640 bits (40 coefficients * 16 bits)

mfcc_extractor u_mfcc (
    .clk(clk),
    .rst_n(rst_n),
    .audio_sample(audio_sample),
    .mfcc_feature(mfcc_feature)
);

wire [23:0] conv1_weight_base_addr = 24'h000000;
wire [23:0] conv1_bias_base_addr = 24'h000100;
wire [23:0] fc1_weight_base_addr = 24'h000200;
wire [23:0] fc1_bias_base_addr = 24'h000300;
wire [23:0] maxpool_input_addr = 24'h000400;
wire [23:0] maxpool_output_addr = 24'h000500;
wire [23:0] softmax_input_addr = 24'h000600;
wire [23:0] softmax_output_addr = 24'h000700;

wire conv1_done, fc1_done, maxpool_done, softmax_done;
wire conv1_data_valid = (state == CONV1);
wire fc1_data_valid = (state == FC1);
wire maxpool_data_valid = (state == MAXPOOL);
wire softmax_start = (state == SOFTMAX);

wire [320*16-1:0] conv1_data_out;
wire conv1_data_out_valid;
wire [64*16-1:0] fc1_data_out;
wire fc1_data_out_valid;
wire [10*16-1:0] maxpool_data_out;
wire maxpool_data_out_valid;
wire [10*8-1:0] softmax_data_out;
wire softmax_data_out_valid;

conv2d_psram #(
   .INPUT_WIDTH(40),
   .INPUT_HEIGHT(1),
   .INPUT_CHANNELS(1),
   .KERNEL_SIZE(3),
   .NUM_FILTERS(8),
   .PADDING(1),
   .ACTIV_BITS(16)
) conv1 (
   .clk(clk),
   .rst_n(rst_n),
   .data_in(mfcc_feature), // MFCC feature as input
   .data_valid(conv1_data_valid),
   .data_out(conv1_data_out),
   .data_out_valid(conv1_data_out_valid),
   .psram_sck(conv1_psram_sck),
   .psram_ce_n(conv1_psram_ce_n),
   .psram_d(conv1_psram_d),
   .psram_douten(conv1_psram_douten),
   .weight_base_addr(conv1_weight_base_addr),
   .bias_base_addr(conv1_bias_base_addr),
   .done(conv1_done)
);

fully_connected_psram #(
   .INPUT_SIZE(320),
   .OUTPUT_SIZE(64),
   .ACTIV_BITS(16)
) fc1 (
   .clk(clk),
   .rst_n(rst_n),
   .data_in(conv1_data_out),
   .data_valid(conv1_data_out_valid),
   .data_out(fc1_data_out),
   .data_out_valid(fc1_data_out_valid),
   .psram_sck(fc1_psram_sck),
   .psram_ce_n(fc1_psram_ce_n),
   .psram_d(fc1_psram_d),
   .psram_douten(fc1_psram_douten),
   .weight_base_addr(fc1_weight_base_addr),
   .bias_base_addr(fc1_bias_base_addr),
   .done(fc1_done)
);

maxpool_psram #(
   .INPUT_WIDTH(40),
   .INPUT_HEIGHT(1),
   .INPUT_CHANNELS(8),
   .KERNEL_SIZE(2),
   .STRIDE(2),
   .ACTIV_BITS(16),
   .ADDR_WIDTH(24)
) maxpool (
   .clk(clk),
   .rst_n(rst_n),
   .start(maxpool_data_valid),
   .input_addr(maxpool_input_addr),
   .output_addr(maxpool_output_addr),
   .done(maxpool_done),
   .psram_sck(maxpool_psram_sck),
   .psram_ce_n(maxpool_psram_ce_n),
   .psram_d(maxpool_psram_d),
   .psram_douten(maxpool_psram_douten)
);

softmax_psram #(
   .INPUT_SIZE(10),
   .ACTIV_BITS(8),
   .ADDR_WIDTH(24)
) softmax (
   .clk(clk),
   .rst_n(rst_n),
   .start(softmax_start),
   .input_addr(softmax_input_addr),
   .output_addr(softmax_output_addr),
   .size(3'b010),
   .cmd(8'hEB),
   .rd_wr(1'b1),
   .qspi(1'b0),
   .qpi(1'b0),
   .short_cmd(1'b0),
       .done(softmax_done),
       .psram_sck(softmax_psram_sck),
       .psram_ce_n(softmax_psram_ce_n),
       .psram_d(softmax_psram_d),
       .psram_douten(softmax_psram_douten)
   );

   // Assign overall done signal
   assign done = (state == SOFTMAX) && softmax_done;

endmodule
```

Updated user_project_wrapper Module

Ensure that the audio_sample and other signals are correctly connected in the user_project_wrapper module:

verilog

Copy code

```
`default_nettype none
/*
 *-------------------------------------------------------------
 *
 * user_project_wrapper
 *
 * This wrapper enumerates all of the pins available to the
 * user for the user project.
 *
 * An example user project is provided in this wrapper.  The
 * example should be removed and replaced with the actual
 * user project.
 *
 *-------------------------------------------------------------
 */

module user_project_wrapper #(
    parameter BITS = 32
) (
`ifdef USE_POWER_PINS
    inout vdda1,  // User area 1 3.3V supply
    inout vdda2,  // User area 2 3.3V supply
    inout vssa1,  // User area 1 analog ground
    inout vssa2,  // User area 2 analog ground
    inout vccd1,  // User area 1 1.8V supply
    inout vccd2,  // User area 2 1.8v supply
    inout vssd1,  // User area 1 digital ground
    inout vssd2,  // User area 2 digital ground
`endif

    // Wishbone Slave ports (WB MI A)
    input wb_clk_i,
    input wb_rst_i,
    input wbs_stb_i,
    input wbs_cyc_i,
    input wbs_we_i,
    input [3:0] wbs_sel_i,
    input [31:0] wbs_dat_i,
    input [31:0] wbs_adr_i,
    output wbs_ack_o,
    output [31:0] wbs_dat_o,

    // Logic Analyzer Signals
    input  [127:0] la_data_in,
    output [127:0] la_data_out,
    input  [127:0] la_oenb,

    // IOs
    input  [`MPRJ_IO_PADS-1:0] io_in,
    output [`MPRJ_IO_PADS-1:0] io_out,
    output [`MPRJ_IO_PADS-1:0] io_oeb,

    // Analog (direct connection to GPIO pad---use with caution)
    // Note that analog I/O is not available on the 7 lowest-numbered
    // GPIO pads, and so the analog_io indexing is offset from the
    // GPIO indexing by 7 (also upper 2 GPIOs do not have analog_io).
    inout [`MPRJ_IO_PADS-10:0] analog_io,

    // Independent clock (on independent integer divider)
    input   user_clock2,

    // User maskable interrupt signals
    output [2:0] user_irq
);

/*--------------------------------------*/
/* User project is instantiated  here   */
/*--------------------------------------*/

// Signals for cnn_kws_accel
wire psram_sck;
wire psram_ce_n;
wire [3:0] psram_d;
wire [3:0] psram_douten;
wire cnn_done;
wire cnn_start;
wire [15:0] audio_sample; // 16-bit audio sample input


// You can use some GPIO pins to control the start signal and monitor the done signal.
// Here we use io_in[0] as the start signal, io_in[16:1] for audio sample, and io_out[0] as the done signal.
assign cnn_start = io_in[0];
assign audio_sample = io_in[16:1];
assign io_out[0] = cnn_done;


cnn_kws_accel u_cnn_kws_accel (
    .clk(wb_clk_i),
    .rst_n(~wb_rst_i),
    .start(cnn_start),
    .audio_sample(audio_sample),
    .done(cnn_done),
    .psram_sck(psram_sck),
    .psram_ce_n(psram_ce_n),
    .psram_d(psram_d),
    .psram_douten(psram_douten)
);


endmodule        // user_project_wrapper


`default_nettype wire
```

Summary:

Updated mfcc_extractor output width to match the input width of conv2d_psram.

Modified cnn_kws_accel to ensure correct signal widths and connections.

Integrated audio_sample correctly in the user_project_wrapper module.

Make sure to verify and test the updated design to ensure it meets your requirements.