Chat : https://claude.ai/chat/a2cd7ee8-743c-472d-a2a8-e301e4aa792b

You are a senior chip design engineer who is an expert with Verilog HDL, especially 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. We 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. (2017). I want you to help me debug the code for the project. Top module is kws. Submodule code "`ifndef MAXPOOL2D_V `define MAXPOOL2D_V module maxpool2d #( parameter INPUT_WIDTH = 32, parameter INPUT_HEIGHT = 1, parameter INPUT_CHANNELS = 8, parameter KERNEL_SIZE = 2, parameter STRIDE = 2, parameter ACTIV_BITS = 8 ) ( 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 reg [0:(INPUT_WIDTH/KERNEL_SIZE)*(INPUT_HEIGHT/KERNEL_SIZE)*INPUT_CHANNELS*ACTIV_BITS-1] data_out, output reg data_out_valid ); localparam OUTPUT_WIDTH = INPUT_WIDTH / KERNEL_SIZE; localparam OUTPUT_HEIGHT = INPUT_HEIGHT / KERNEL_SIZE; // Declare 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]; // Max pooling operation integer i, j, k, m, n; always @(posedge clk or negedge rst_n) begin if (!rst_n) begin // Reset internal signals and output 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 data_out <= 0; data_out_valid <= 0; end else if (data_valid) begin // Shift input data into 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 if (j < INPUT_WIDTH - 1) begin input_buffer[i][j][k] <= input_buffer[i][j+1][k]; end else begin input_buffer[i][j][k] <= data_in[i*INPUT_WIDTH*INPUT_CHANNELS*ACTIV_BITS + j*INPUT_CHANNELS*ACTIV_BITS + k*ACTIV_BITS +: ACTIV_BITS]; end end end end // Perform max pooling for (i = 0; i < OUTPUT_HEIGHT; i = i + 1) begin for (j = 0; j < OUTPUT_WIDTH; 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*OUTPUT_WIDTH*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 begin data_out_valid <= 0; end end endmodule `endif ", "`ifndef SOFTMAX_V `define SOFTMAX_V module softmax #( parameter INPUT_SIZE = 10, parameter ACTIV_BITS = 8 ) ( input wire clk, input wire rst_n, input wire [INPUT_SIZE*ACTIV_BITS-1:0] data_in, input wire data_valid, output reg [INPUT_SIZE*ACTIV_BITS-1:0] data_out, output reg data_out_valid ); // Declare internal signals 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]; // Softmax activation function integer i; always @(posedge clk or negedge rst_n) begin if

```
(!rst_n) begin // Reset internal signals and output for (i = 0; i < INPUT_SIZE; i = i + 1) begin
exp_values[i] <= 0; softmax_values[i] <= 0; end sum_exp <= 0; data_out <= 0;
data_out_valid <= 0; 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 // Assign output for (i = 0; i < INPUT_SIZE; i = i + 1) begin
data_out[i*ACTIV_BITS +: ACTIV_BITS] <= softmax_values[i]; end data_out_valid <= 1;
end else begin data_out_valid <= 0; end end endmodule `endif ", "`ifndef
FULLY_CONNECTED_V `define FULLY_CONNECTED_V module fully_connected #(
parameter INPUT_SIZE = 640, parameter OUTPUT_SIZE = 64, parameter ACTIV_BITS =
8 ) ( input wire clk, input wire rst_n, input wire [INPUT_SIZE*ACTIV_BITS-1:0] data_in,
input wire data_valid, output reg [OUTPUT_SIZE*ACTIV_BITS-1:0] data_out, output reg
data_out_valid, input wire [OUTPUT_SIZE*INPUT_SIZE*ACTIV_BITS-1:0] weights_in,
input wire [OUTPUT_SIZE*ACTIV_BITS-1:0] biases_in, input wire load_weights, input
wire load_biases ); // Declare weights and biases reg [ACTIV_BITS-1:0] weights
[0:OUTPUT_SIZE-1][0:INPUT_SIZE-1]; reg [ACTIV_BITS-1:0] biases
[0:OUTPUT_SIZE-1]; // Declare internal signals reg [2*ACTIV_BITS-1:0] acc_result
[0:OUTPUT_SIZE-1]; reg [ACTIV_BITS-1:0] relu_result [0:OUTPUT_SIZE-1]; // Load
weights and biases integer i, j; always @(posedge clk or negedge rst_n) begin if (!rst_n)
begin // Reset weights and biases for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin for (j = 0; j <
INPUT_SIZE; j = j + 1) begin weights[i][j] = 0; end biases[i] <= 0; end end else begin //
Load weights when load_weights is asserted if (load_weights) begin for (i = 0; i <
OUTPUT_SIZE; i = i + 1) begin for (j = 0; j < INPUT_SIZE; j = j + 1) begin weights[i][j] =
weights_in[(i*INPUT_SIZE + j)*ACTIV_BITS +: ACTIV_BITS]; end end end // Load
biases when load_biases is asserted if (load_biases) begin for (i = 0; i < OUTPUT_SIZE; i = i
+ 1) begin biases[i] <= biases_in[i*ACTIV_BITS +: ACTIV_BITS]; end end end end // Fully
connected layer operation always @(posedge clk or negedge rst_n) begin if (!rst_n) begin //
Reset internal signals and output for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin acc_result[i]
<= 0; relu_result[i] <= 0; end data_out <= 0; data_out_valid <= 0; end else if (data_valid)
begin // Perform matrix multiplication for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin
acc_result[i] = {{(2*ACTIV_BITS-ACTIV_BITS){1'b0}}, biases[i]}; for (j = 0; j <
INPUT_SIZE; j = j + 1) begin acc_result[i] = acc_result[i] + weights[i][j] *
data_in[j*ACTIV_BITS +: ACTIV_BITS]; end end // Apply ReLU activation for (i = 0; i <
OUTPUT_SIZE; i = i + 1) begin relu_result[i] <= (acc_result[i][2*ACTIV_BITS-1] == 0) ?
acc_result[i][ACTIV_BITS-1:0] : 0; end // Assign output for (i = 0; i < OUTPUT_SIZE; i = i
+ 1) begin data_out[i*ACTIV_BITS +: ACTIV_BITS] <= relu_result[i]; end data_out_valid
<= 1; end else begin data_out_valid <= 0; end end endmodule `endif ", "`ifndef
PREEMPHASIS_FILTER_V `define PREEMPHASIS_FILTER_V module
preemphasis_filter ( input wire clk, input wire rst_n, input wire [15:0] audio_in, input wire
audio_valid, output reg [15:0] preemph_out, output reg preemph_valid ); // Configurable pre-
emphasis coefficient (0.97 by default) localparam PREEMPH_COEF = 16'h7D71; // 0.97 in
Q15 format // Internal registers reg [15:0] audio_delay; // Pre-emphasis filtering logic always
@(posedge clk or negedge rst_n) begin if (!rst_n) begin audio_delay <= 16'h0000;
preemph_out <= 16'h0000; preemph_valid <= 1'b0; end else if (audio_valid) begin
audio_delay <= audio_in; preemph_out <= $signed(audio_in) -
$signed(($signed(audio_delay) * $signed(PREEMPH_COEF)) >>> 15); preemph_valid <=
1'b1; end else begin preemph_valid <= 1'b0; end end endmodule `endif ", "`ifndef
```

```
MFCC_ACCELERATOR_V `define MFCC_ACCELERATOR_V module mfcc_accelerator
#( parameter MFCC_FEATURES = 40, parameter ACTIV_BITS = 8 ) ( input wire clk, input
wire rst_n, input wire [15:0] audio_in, input wire audio_valid, output reg
[MFCC_FEATURES*ACTIV_BITS-1:0] mfcc_out, output reg mfcc_valid, input wire [7:0]
frame_size, input wire [7:0] frame_overlap, input wire [7:0] num_mfcc_coeffs, input wire
[4095:0] goertzel_coefs ); // Signal declarations wire [15:0] preemph_out; wire
preemph_valid; wire [15:0] framed_out; wire framed_valid; wire [31:0] dft_out; wire
dft_valid; wire [31:0] mel_fbank_out; wire mel_fbank_valid; wire [31:0] log_out; wire
log_valid; wire [MFCC_FEATURES*ACTIV_BITS-1:0] dct_out; wire dct_valid; //
Instantiate sub-modules // Pre-emphasis filtering preemphasis_filter preemph ( .clk(clk),
.rst_n(rst_n), .audio_in(audio_in), .audio_valid(audio_valid), .preemph_out(preemph_out),
.preemph_valid(preemph_valid) ); // Framing and windowing framing_windowing framing (
.clk(clk), .rst_n(rst_n), .preemph_out(preemph_out), .preemph_valid(preemph_valid),
.frame_size(frame_size), .frame_overlap(frame_overlap), .framed_out(framed_out),
.framed_valid(framed_valid) ); // Discrete Fourier Transform (DFT) using Goertzel's
algorithm goertzel_dft dft ( .clk(clk), .rst_n(rst_n), .framed_out(framed_out),
.framed_valid(framed_valid), .goertzel_coefs(goertzel_coefs), .dft_out(dft_out),
.dft_valid(dft_valid) ); mel_filterbank #( .DFT_SIZE(256), .NUM_MEL_FILTERS(40),
.MEL_FBANK_OUT_BITS(32) ) mel_fbank ( .clk(clk), .rst_n(rst_n), .dft_out(dft_out),
.dft_valid(dft_valid), .mel_fbank_out(mel_fbank_out), .mel_fbank_valid(mel_fbank_valid) );
// Logarithm computation logarithm_comp log_comp ( .clk(clk), .rst_n(rst_n),
.mel_fbank_out(mel_fbank_out), .mel_fbank_valid(mel_fbank_valid), .log_out(log_out),
.log_valid(log_valid) ); // Discrete Cosine Transform (DCT) dct_comp dct ( .clk(clk),
.rst_n(rst_n), .log_out(log_out), .log_valid(log_valid),
.num_mfcc_coeffs(num_mfcc_coeffs[4:0]), .dct_out(dct_out), .dct_valid(dct_valid) ); //
Output assignment always @(posedge clk or negedge rst_n) begin if (!rst_n) begin mfcc_out
<= 'b0; mfcc_valid <= 1'b0; end else begin mfcc_out <= dct_out; mfcc_valid <= dct_valid;
end end endmodule `endif ", "`ifndef GOERTZEL_DFT_V `define GOERTZEL_DFT_V
module goertzel_dft #( parameter NUM_FREQS = 256 )( input wire clk, input wire rst_n,
input wire [15:0] framed_out, input wire framed_valid, input wire [4095:0] goertzel_coefs,
output reg [31:0] dft_out, output reg dft_valid ); // Goertzel algorithm variables reg [31:0]
q_prev [0:NUM_FREQS-1]; reg [31:0] q_curr [0:NUM_FREQS-1]; reg
[$clog2(NUM_FREQS)-1:0] freq_idx; // Goertzel algorithm implementation integer j; always
@(posedge clk or negedge rst_n) begin if (!rst_n) begin for (j = 0; j < NUM_FREQS; j = j +
1) begin q_prev[j] <= 32'h0; q_curr[j] <= 32'h0; end freq_idx <= 'h0; dft_out <= 32'h0;
dft_valid <= 1'b0; end else if (framed_valid) begin for (j = 0; j < NUM_FREQS; j = j + 1)
begin // Create temporary variables reg [31:0] q_curr_temp; reg [31:0] q_prev_temp; //
Compute Goertzel algorithm q_curr_temp = (goertzel_coefs[j*16 +: 16] * q_prev[j] >>> 15)
- q_curr[j] + {{16{framed_out[15]}}, framed_out}; q_prev_temp = q_curr[j]; // Assign the
updated values to the arrays q_curr[j] <= q_curr_temp; q_prev[j] <= q_prev_temp; end //
Increment frequency index freq_idx <= freq_idx + 1; // Output DFT result when all
frequencies are processed if (freq_idx == NUM_FREQS[$clog2(NUM_FREQS)-1:0] - 1)
begin dft_out <= q_curr[freq_idx]*q_curr[freq_idx] + q_prev[freq_idx]*q_prev[freq_idx] -
(goertzel_coefs[freq_idx*16 +: 16] * q_curr[freq_idx] >>> 15); dft_valid <= 1'b1; freq_idx
<= 'h0; end else begin dft_valid <= 1'b0; end end else begin dft_valid <= 1'b0; end end
endmodule `endif ", "`ifndef DCT_COMP_V `define DCT_COMP_V module dct_comp (
input wire clk, input wire rst_n, input wire [31:0] log_out, input wire log_valid, input wire
[4:0] num_mfcc_coeffs, output reg [31:0] dct_out, output reg dct_valid ); // Constants
localparam MAX_COEFFS = 32; // DCT coefficients (stored in ROM) reg [31:0] dct_coeffs
[0:MAX_COEFFS-1][0:MAX_COEFFS-1]; // Intermediate variables reg [31:0] dct_sum; reg
```

```
[4:0] coeff_idx; reg [$clog2(MAX_COEFFS)-1:0] log_idx; // DCT computation pipeline
always @(posedge clk or negedge rst_n) begin if (!rst_n) begin dct_out <= 32'h0; dct_valid
<= 1'b0; dct_sum <= 32'h0; coeff_idx <= 5'h0; log_idx <= 'h0; end else if (log_valid) begin
dct_sum <= dct_sum + (log_out * dct_coeffs[coeff_idx][log_idx]); log_idx <= log_idx + 1; if
(coeff_idx == num_mfcc_coeffs - 1) begin dct_out <= dct_sum; dct_valid <= 1'b1; dct_sum
<= 32'h0; coeff_idx <= 5'h0; log_idx <= 'h0; end else if (log_idx ==
MAX_COEFFS[$clog2(MAX_COEFFS)-1:0] - 1) begin coeff_idx <= coeff_idx + 1; log_idx
<= 'h0; dct_valid <= 1'b0; end else begin dct_valid <= 1'b0; end end else begin dct_valid <=
1'b0; end end // Initialize DCT coefficients initial begin dct_coeffs[0][0] = 32'h0B504F33;
dct_coeffs[0][1] = 32'h0B504F33; dct_coeffs[0][2] = 32'h0B504F33; dct_coeffs[0][3] =
32'h0B504F33; dct_coeffs[0][4] = 32'h0B504F33; dct_coeffs[0][5] = 32'h0B504F33;
dct_coeffs[0][6] = 32'h0B504F33; dct_coeffs[0][7] = 32'h0B504F33; dct_coeffs[0][8] =
32'h0B504F33; dct_coeffs[0][9] = 32'h0B504F33; dct_coeffs[0][10] = 32'h0B504F33;
dct_coeffs[0][11] = 32'h0B504F33; dct_coeffs[0][12] = 32'h0B504F33; dct_coeffs[0][13] =
32'h0B504F33; dct_coeffs[0][14] = 32'h0B504F33; dct_coeffs[0][15] = 32'h0B504F33;
dct_coeffs[0][16] = 32'h0B504F33; dct_coeffs[0][17] = 32'h0B504F33; dct_coeffs[0][18] =
32'h0B504F33; dct_coeffs[0][19] = 32'h0B504F33; dct_coeffs[0][20] = 32'h0B504F33;
dct_coeffs[0][21] = 32'h0B504F33; dct_coeffs[0][22] = 32'h0B504F33; dct_coeffs[0][23] =
32'h0B504F33; dct_coeffs[0][24] = 32'h0B504F33; dct_coeffs[0][25] = 32'h0B504F33;
dct_coeffs[0][26] = 32'h0B504F33; dct_coeffs[0][27] = 32'h0B504F33; dct_coeffs[0][28] =
32'h0B504F33; dct_coeffs[0][29] = 32'h0B504F33; dct_coeffs[0][30] = 32'h0B504F33;
dct_coeffs[0][31] = 32'h0B504F33; dct_coeffs[1][0] = 32'h0FFB10F1; dct_coeffs[1][1] =
32'h0FD3AABF; dct_coeffs[1][2] = 32'h0F853F7D; dct_coeffs[1][3] = 32'h0F109082;
dct_coeffs[1][4] = 32'h0E76BD7A; dct_coeffs[1][5] = 32'h0DB941A2; dct_coeffs[1][6] =
32'h0CD9F023; dct_coeffs[1][7] = 32'h0BDAEF91; dct_coeffs[1][8] = 32'h0ABEB49A;
dct_coeffs[1][9] = 32'h0987FBFE; dct_coeffs[1][10] = 32'h0839C3CC; dct_coeffs[1][11] =
32'h06D74402; dct_coeffs[1][12] = 32'h0563E69D; dct_coeffs[1][13] = 32'h03E33F2F;
dct_coeffs[1][14] = 32'h0259020D; dct_coeffs[1][15] = 32'h00C8FB2F; dct_coeffs[1][16] =
32'hFF3704D1; dct_coeffs[1][17] = 32'hFDA6FDF3; dct_coeffs[1][18] = 32'hFC1CC0D1;
dct_coeffs[1][19] = 32'hFA9C1963; dct_coeffs[1][20] = 32'hF928BBFE; dct_coeffs[1][21] =
32'hF7C63C34; dct_coeffs[1][22] = 32'hF6780402; dct_coeffs[1][23] = 32'hF5414B66;
dct_coeffs[1][24] = 32'hF425106F; dct_coeffs[1][25] = 32'hF3260FDD; dct_coeffs[1][26] =
32'hF246BE5E; dct_coeffs[1][27] = 32'hF1894286; dct_coeffs[1][28] = 32'hF0EF6F7E;
dct_coeffs[1][29] = 32'hF07AC083; dct_coeffs[1][30] = 32'hF02C5541; dct_coeffs[1][31] =
32'hF004EF0F; dct_coeffs[2][0] = 32'h0FEC46D1; dct_coeffs[2][1] = 32'h0F4FA0AB;
dct_coeffs[2][2] = 32'h0E1C5978; dct_coeffs[2][3] = 32'h0C5E4035; dct_coeffs[2][4] =
32'h0A267992; dct_coeffs[2][5] = 32'h078AD74E; dct_coeffs[2][6] = 32'h04A5018B;
dct_coeffs[2][7] = 32'h01917A6B; dct_coeffs[2][8] = 32'hFE6E8595; dct_coeffs[2][9] =
32'hFB5AFE75; dct_coeffs[2][10] = 32'hF87528B2; dct_coeffs[2][11] = 32'hF5D9866E;
dct_coeffs[2][12] = 32'hF3A1BFCB; dct_coeffs[2][13] = 32'hF1E3A688; dct_coeffs[2][14] =
32'hF0B05F55; dct_coeffs[2][15] = 32'hF013B92F; dct_coeffs[2][16] = 32'hF013B92F;
dct_coeffs[2][17] = 32'hF0B05F55; dct_coeffs[2][18] = 32'hF1E3A688; dct_coeffs[2][19] =
32'hF3A1BFCB; dct_coeffs[2][20] = 32'hF5D9866E; dct_coeffs[2][21] = 32'hF87528B2;
dct_coeffs[2][22] = 32'hFB5AFE75; dct_coeffs[2][23] = 32'hFE6E8595; dct_coeffs[2][24] =
32'h01917A6B; dct_coeffs[2][25] = 32'h04A5018B; dct_coeffs[2][26] = 32'h078AD74E;
dct_coeffs[2][27] = 32'h0A267992; dct_coeffs[2][28] = 32'h0C5E4035; dct_coeffs[2][29] =
32'h0E1C5978; dct_coeffs[2][30] = 32'h0F4FA0AB; dct_coeffs[2][31] = 32'h0FEC46D1;
dct_coeffs[3][0] = 32'h0FD3AABF; dct_coeffs[3][1] = 32'h0E76BD7A; dct_coeffs[3][2] =
32'h0BDAEF91; dct_coeffs[3][3] = 32'h0839C3CC; dct_coeffs[3][4] = 32'h03E33F2F;
dct_coeffs[3][5] = 32'hFF3704D1; dct_coeffs[3][6] = 32'hFA9C1963; dct_coeffs[3][7] =
32'hF6780402; dct_coeffs[3][8] = 32'hF3260FDD; dct_coeffs[3][9] = 32'hF0EF6F7E;
dct_coeffs[3][10] = 32'hF004EF0F; dct_coeffs[3][11] = 32'hF07AC083; dct_coeffs[3][12] =
32'hF246BE5E; dct_coeffs[3][13] = 32'hF5414B66; dct_coeffs[3][14] = 32'hF928BBFE;
dct_coeffs[3][15] = 32'hFDA6FDF3; dct_coeffs[3][16] = 32'h0259020D; dct_coeffs[3][17] =
32'h06D74402; dct_coeffs[3][18] = 32'h0ABEB49A; dct_coeffs[3][19] = 32'h0DB941A2;
dct_coeffs[3][20] = 32'h0F853F7D; dct_coeffs[3][21] = 32'h0FFB10F1; dct_coeffs[3][22] =
32'h0F109082; dct_coeffs[3][23] = 32'h0CD9F023; dct_coeffs[3][24] = 32'h0987FBFE;
dct_coeffs[3][25] = 32'h0563E69D; dct_coeffs[3][26] = 32'h00C8FB2F; dct_coeffs[3][27] =
32'hFC1CC0D1; dct_coeffs[3][28] = 32'hF7C63C34; dct_coeffs[3][29] = 32'hF425106F;
dct_coeffs[3][30] = 32'hF1894286; dct_coeffs[3][31] = 32'hF02C5541; dct_coeffs[4][0] =
32'h0FB14BE7; dct_coeffs[4][1] = 32'h0D4DB314; dct_coeffs[4][2] = 32'h08E39D9C;
dct_coeffs[4][3] = 32'h031F1707; dct_coeffs[4][4] = 32'hFCE0E8F9; dct_coeffs[4][5] =
32'hF71C6264; dct_coeffs[4][6] = 32'hF2B24CEC; dct_coeffs[4][7] = 32'hF04EB419;
dct_coeffs[4][8] = 32'hF04EB419; dct_coeffs[4][9] = 32'hF2B24CEC; dct_coeffs[4][10] =
32'hF71C6264; dct_coeffs[4][11] = 32'hFCE0E8F9; dct_coeffs[4][12] = 32'h031F1707;
dct_coeffs[4][13] = 32'h08E39D9C; dct_coeffs[4][14] = 32'h0D4DB314; dct_coeffs[4][15] =
32'h0FB14BE7; dct_coeffs[4][16] = 32'h0FB14BE7; dct_coeffs[4][17] = 32'h0D4DB314;
dct_coeffs[4][18] = 32'h08E39D9C; dct_coeffs[4][19] = 32'h031F1707; dct_coeffs[4][20] =
32'hFCE0E8F9; dct_coeffs[4][21] = 32'hF71C6264; dct_coeffs[4][22] = 32'hF2B24CEC;
dct_coeffs[4][23] = 32'hF04EB419; dct_coeffs[4][24] = 32'hF04EB419; dct_coeffs[4][25] =
32'hF2B24CEC; dct_coeffs[4][26] = 32'hF71C6264; dct_coeffs[4][27] = 32'hFCE0E8F9;
dct_coeffs[4][28] = 32'h031F1707; dct_coeffs[4][29] = 32'h08E39D9C; dct_coeffs[4][30] =
32'h0D4DB314; dct_coeffs[4][31] = 32'h0FB14BE7; dct_coeffs[5][0] = 32'h0F853F7D;
dct_coeffs[5][1] = 32'h0BDAEF91; dct_coeffs[5][2] = 32'h0563E69D; dct_coeffs[5][3] =
32'hFDA6FDF3; dct_coeffs[5][4] = 32'hF6780402; dct_coeffs[5][5] = 32'hF1894286;
dct_coeffs[5][6] = 32'hF004EF0F; dct_coeffs[5][7] = 32'hF246BE5E; dct_coeffs[5][8] =
32'hF7C63C34; dct_coeffs[5][9] = 32'hFF3704D1; dct_coeffs[5][10] = 32'h06D74402;
dct_coeffs[5][11] = 32'h0CD9F023; dct_coeffs[5][12] = 32'h0FD3AABF; dct_coeffs[5][13]
= 32'h0F109082; dct_coeffs[5][14] = 32'h0ABEB49A; dct_coeffs[5][15] = 32'h03E33F2F;
dct_coeffs[5][16] = 32'hFC1CC0D1; dct_coeffs[5][17] = 32'hF5414B66; dct_coeffs[5][18] =
32'hF0EF6F7E; dct_coeffs[5][19] = 32'hF02C5541; dct_coeffs[5][20] = 32'hF3260FDD;
dct_coeffs[5][21] = 32'hF928BBFE; dct_coeffs[5][22] = 32'h00C8FB2F; dct_coeffs[5][23] =
32'h0839C3CC; dct_coeffs[5][24] = 32'h0DB941A2; dct_coeffs[5][25] = 32'h0FFB10F1;
dct_coeffs[5][26] = 32'h0E76BD7A; dct_coeffs[5][27] = 32'h0987FBFE; dct_coeffs[5][28] =
32'h0259020D; dct_coeffs[5][29] = 32'hFA9C1963; dct_coeffs[5][30] = 32'hF425106F;
dct_coeffs[5][31] = 32'hF07AC083; dct_coeffs[6][0] = 32'h0F4FA0AB; dct_coeffs[6][1] =
32'h0A267992; dct_coeffs[6][2] = 32'h01917A6B; dct_coeffs[6][3] = 32'hF87528B2;
dct_coeffs[6][4] = 32'hF1E3A688; dct_coeffs[6][5] = 32'hF013B92F; dct_coeffs[6][6] =
32'hF3A1BFCB; dct_coeffs[6][7] = 32'hFB5AFE75; dct_coeffs[6][8] = 32'h04A5018B;
dct_coeffs[6][9] = 32'h0C5E4035; dct_coeffs[6][10] = 32'h0FEC46D1; dct_coeffs[6][11] =
32'h0E1C5978; dct_coeffs[6][12] = 32'h078AD74E; dct_coeffs[6][13] = 32'hFE6E8595;
dct_coeffs[6][14] = 32'hF5D9866E; dct_coeffs[6][15] = 32'hF0B05F55; dct_coeffs[6][16] =
32'hF0B05F55; dct_coeffs[6][17] = 32'hF5D9866E; dct_coeffs[6][18] = 32'hFE6E8595;
dct_coeffs[6][19] = 32'h078AD74E; dct_coeffs[6][20] = 32'h0E1C5978; dct_coeffs[6][21] =
32'h0FEC46D1; dct_coeffs[6][22] = 32'h0C5E4035; dct_coeffs[6][23] = 32'h04A5018B;
dct_coeffs[6][24] = 32'hFB5AFE75; dct_coeffs[6][25] = 32'hF3A1BFCB; dct_coeffs[6][26]
= 32'hF013B92F; dct_coeffs[6][27] = 32'hF1E3A688; dct_coeffs[6][28] = 32'hF87528B2;
dct_coeffs[6][29] = 32'h01917A6B; dct_coeffs[6][30] = 32'h0A267992; dct_coeffs[6][31] =
32'h0F4FA0AB; dct_coeffs[7][0] = 32'h0F109082; dct_coeffs[7][1] = 32'h0839C3CC;
dct_coeffs[7][2] = 32'hFDA6FDF3; dct_coeffs[7][3] = 32'hF425106F; dct_coeffs[7][4] =
32'hF004EF0F; dct_coeffs[7][5] = 32'hF3260FDD; dct_coeffs[7][6] = 32'hFC1CC0D1;
dct_coeffs[7][7] = 32'h06D74402; dct_coeffs[7][8] = 32'h0E76BD7A; dct_coeffs[7][9] =
32'h0F853F7D; dct_coeffs[7][10] = 32'h0987FBFE; dct_coeffs[7][11] = 32'hFF3704D1;
dct_coeffs[7][12] = 32'hF5414B66; dct_coeffs[7][13] = 32'hF02C5541; dct_coeffs[7][14] =
32'hF246BE5E; dct_coeffs[7][15] = 32'hFA9C1963; dct_coeffs[7][16] = 32'h0563E69D;
dct_coeffs[7][17] = 32'h0DB941A2; dct_coeffs[7][18] = 32'h0FD3AABF; dct_coeffs[7][19]
= 32'h0ABEB49A; dct_coeffs[7][20] = 32'h00C8FB2F; dct_coeffs[7][21] = 32'hF6780402;
dct_coeffs[7][22] = 32'hF07AC083; dct_coeffs[7][23] = 32'hF1894286; dct_coeffs[7][24] =
32'hF928BBFE; dct_coeffs[7][25] = 32'h03E33F2F; dct_coeffs[7][26] = 32'h0CD9F023;
dct_coeffs[7][27] = 32'h0FFB10F1; dct_coeffs[7][28] = 32'h0BDAEF91; dct_coeffs[7][29] =
32'h0259020D; dct_coeffs[7][30] = 32'hF7C63C34; dct_coeffs[7][31] = 32'hF0EF6F7E;
dct_coeffs[8][0] = 32'h0EC835E7; dct_coeffs[8][1] = 32'h061F78A9; dct_coeffs[8][2] =
32'hF9E08757; dct_coeffs[8][3] = 32'hF137CA19; dct_coeffs[8][4] = 32'hF137CA19;
dct_coeffs[8][5] = 32'hF9E08757; dct_coeffs[8][6] = 32'h061F78A9; dct_coeffs[8][7] =
32'h0EC835E7; dct_coeffs[8][8] = 32'h0EC835E7; dct_coeffs[8][9] = 32'h061F78A9;
dct_coeffs[8][10] = 32'hF9E08757; dct_coeffs[8][11] = 32'hF137CA19; dct_coeffs[8][12] =
32'hF137CA19; dct_coeffs[8][13] = 32'hF9E08757; dct_coeffs[8][14] = 32'h061F78A9;
dct_coeffs[8][15] = 32'h0EC835E7; dct_coeffs[8][16] = 32'h0EC835E7; dct_coeffs[8][17] =
32'h061F78A9; dct_coeffs[8][18] = 32'hF9E08757; dct_coeffs[8][19] = 32'hF137CA19;
dct_coeffs[8][20] = 32'hF137CA19; dct_coeffs[8][21] = 32'hF9E08757; dct_coeffs[8][22] =
32'h061F78A9; dct_coeffs[8][23] = 32'h0EC835E7; dct_coeffs[8][24] = 32'h0EC835E7;
dct_coeffs[8][25] = 32'h061F78A9; dct_coeffs[8][26] = 32'hF9E08757; dct_coeffs[8][27] =
32'hF137CA19; dct_coeffs[8][28] = 32'hF137CA19; dct_coeffs[8][29] = 32'hF9E08757;
dct_coeffs[8][30] = 32'h061F78A9; dct_coeffs[8][31] = 32'h0EC835E7; dct_coeffs[9][0] =
32'h0E76BD7A; dct_coeffs[9][1] = 32'h03E33F2F; dct_coeffs[9][2] = 32'hF6780402;
dct_coeffs[9][3] = 32'hF004EF0F; dct_coeffs[9][4] = 32'hF5414B66; dct_coeffs[9][5] =
32'h0259020D; dct_coeffs[9][6] = 32'h0DB941A2; dct_coeffs[9][7] = 32'h0F109082;
dct_coeffs[9][8] = 32'h0563E69D; dct_coeffs[9][9] = 32'hF7C63C34; dct_coeffs[9][10] =
32'hF02C5541; dct_coeffs[9][11] = 32'hF425106F; dct_coeffs[9][12] = 32'h00C8FB2F;
dct_coeffs[9][13] = 32'h0CD9F023; dct_coeffs[9][14] = 32'h0F853F7D; dct_coeffs[9][15] =
32'h06D74402; dct_coeffs[9][16] = 32'hF928BBFE; dct_coeffs[9][17] = 32'hF07AC083;
dct_coeffs[9][18] = 32'hF3260FDD; dct_coeffs[9][19] = 32'hFF3704D1; dct_coeffs[9][20] =
32'h0BDAEF91; dct_coeffs[9][21] = 32'h0FD3AABF; dct_coeffs[9][22] = 32'h0839C3CC;
dct_coeffs[9][23] = 32'hFA9C1963; dct_coeffs[9][24] = 32'hF0EF6F7E; dct_coeffs[9][25] =
32'hF246BE5E; dct_coeffs[9][26] = 32'hFDA6FDF3; dct_coeffs[9][27] = 32'h0ABEB49A;
dct_coeffs[9][28] = 32'h0FFB10F1; dct_coeffs[9][29] = 32'h0987FBFE; dct_coeffs[9][30] =
32'hFC1CC0D1; dct_coeffs[9][31] = 32'hF1894286; dct_coeffs[10][0] = 32'h0E1C5978;
dct_coeffs[10][1] = 32'h01917A6B; dct_coeffs[10][2] = 32'hF3A1BFCB; dct_coeffs[10][3] =
32'hF0B05F55; dct_coeffs[10][4] = 32'hFB5AFE75; dct_coeffs[10][5] = 32'h0A267992;
dct_coeffs[10][6] = 32'h0FEC46D1; dct_coeffs[10][7] = 32'h078AD74E; dct_coeffs[10][8] =
32'hF87528B2; dct_coeffs[10][9] = 32'hF013B92F; dct_coeffs[10][10] = 32'hF5D9866E;
dct_coeffs[10][11] = 32'h04A5018B; dct_coeffs[10][12] = 32'h0F4FA0AB;
dct_coeffs[10][13] = 32'h0C5E4035; dct_coeffs[10][14] = 32'hFE6E8595; dct_coeffs[10][15]
= 32'hF1E3A688; dct_coeffs[10][16] = 32'hF1E3A688; dct_coeffs[10][17] = 32'hFE6E8595;
dct_coeffs[10][18] = 32'h0C5E4035; dct_coeffs[10][19] = 32'h0F4FA0AB;
dct_coeffs[10][20] = 32'h04A5018B; dct_coeffs[10][21] = 32'hF5D9866E;
dct_coeffs[10][22] = 32'hF013B92F; dct_coeffs[10][23] = 32'hF87528B2; dct_coeffs[10][24]
= 32'h078AD74E; dct_coeffs[10][25] = 32'h0FEC46D1; dct_coeffs[10][26] =
32'h0A267992; dct_coeffs[10][27] = 32'hFB5AFE75; dct_coeffs[10][28] = 32'hF0B05F55;
dct_coeffs[10][29] = 32'hF3A1BFCB; dct_coeffs[10][30] = 32'h01917A6B;
```

```
dct_coeffs[10][31] = 32'h0E1C5978; dct_coeffs[11][0] = 32'h0DB941A2; dct_coeffs[11][1]
= 32'hFF3704D1; dct_coeffs[11][2] = 32'hF1894286; dct_coeffs[11][3] = 32'hF3260FDD;
dct_coeffs[11][4] = 32'h0259020D; dct_coeffs[11][5] = 32'h0F109082; dct_coeffs[11][6] =
32'h0BDAEF91; dct_coeffs[11][7] = 32'hFC1CC0D1; dct_coeffs[11][8] = 32'hF07AC083;
dct_coeffs[11][9] = 32'hF5414B66; dct_coeffs[11][10] = 32'h0563E69D; dct_coeffs[11][11]
= 32'h0FD3AABF; dct_coeffs[11][12] = 32'h0987FBFE; dct_coeffs[11][13] =
32'hF928BBFE; dct_coeffs[11][14] = 32'hF004EF0F; dct_coeffs[11][15] = 32'hF7C63C34;
dct_coeffs[11][16] = 32'h0839C3CC; dct_coeffs[11][17] = 32'h0FFB10F1;
dct_coeffs[11][18] = 32'h06D74402; dct_coeffs[11][19] = 32'hF6780402; dct_coeffs[11][20]
= 32'hF02C5541; dct_coeffs[11][21] = 32'hFA9C1963; dct_coeffs[11][22] =
32'h0ABEB49A; dct_coeffs[11][23] = 32'h0F853F7D; dct_coeffs[11][24] = 32'h03E33F2F;
dct_coeffs[11][25] = 32'hF425106F; dct_coeffs[11][26] = 32'hF0EF6F7E; dct_coeffs[11][27]
= 32'hFDA6FDF3; dct_coeffs[11][28] = 32'h0CD9F023; dct_coeffs[11][29] =
32'h0E76BD7A; dct_coeffs[11][30] = 32'h00C8FB2F; dct_coeffs[11][31] = 32'hF246BE5E;
dct_coeffs[12][0] = 32'h0D4DB314; dct_coeffs[12][1] = 32'hFCE0E8F9; dct_coeffs[12][2] =
32'hF04EB419; dct_coeffs[12][3] = 32'hF71C6264; dct_coeffs[12][4] = 32'h08E39D9C;
dct_coeffs[12][5] = 32'h0FB14BE7; dct_coeffs[12][6] = 32'h031F1707; dct_coeffs[12][7] =
32'hF2B24CEC; dct_coeffs[12][8] = 32'hF2B24CEC; dct_coeffs[12][9] = 32'h031F1707;
dct_coeffs[12][10] = 32'h0FB14BE7; dct_coeffs[12][11] = 32'h08E39D9C;
dct_coeffs[12][12] = 32'hF71C6264; dct_coeffs[12][13] = 32'hF04EB419; dct_coeffs[12][14]
= 32'hFCE0E8F9; dct_coeffs[12][15] = 32'h0D4DB314; dct_coeffs[12][16] =
32'h0D4DB314; dct_coeffs[12][17] = 32'hFCE0E8F9; dct_coeffs[12][18] = 32'hF04EB419;
dct_coeffs[12][19] = 32'hF71C6264; dct_coeffs[12][20] = 32'h08E39D9C;
dct_coeffs[12][21] = 32'h0FB14BE7; dct_coeffs[12][22] = 32'h031F1707; dct_coeffs[12][23]
= 32'hF2B24CEC; dct_coeffs[12][24] = 32'hF2B24CEC; dct_coeffs[12][25] =
32'h031F1707; dct_coeffs[12][26] = 32'h0FB14BE7; dct_coeffs[12][27] = 32'h08E39D9C;
dct_coeffs[12][28] = 32'hF71C6264; dct_coeffs[12][29] = 32'hF04EB419; dct_coeffs[12][30]
= 32'hFCE0E8F9; dct_coeffs[12][31] = 32'h0D4DB314; dct_coeffs[13][0] = 32'h0CD9F023;
dct_coeffs[13][1] = 32'hFA9C1963; dct_coeffs[13][2] = 32'hF004EF0F; dct_coeffs[13][3] =
32'hFC1CC0D1; dct_coeffs[13][4] = 32'h0DB941A2; dct_coeffs[13][5] = 32'h0BDAEF91;
dct_coeffs[13][6] = 32'hF928BBFE; dct_coeffs[13][7] = 32'hF02C5541; dct_coeffs[13][8] =
32'hFDA6FDF3; dct_coeffs[13][9] = 32'h0E76BD7A; dct_coeffs[13][10] = 32'h0ABEB49A;
dct_coeffs[13][11] = 32'hF7C63C34; dct_coeffs[13][12] = 32'hF07AC083;
dct_coeffs[13][13] = 32'hFF3704D1; dct_coeffs[13][14] = 32'h0F109082; dct_coeffs[13][15]
= 32'h0987FBFE; dct_coeffs[13][16] = 32'hF6780402; dct_coeffs[13][17] = 32'hF0EF6F7E;
dct_coeffs[13][18] = 32'h00C8FB2F; dct_coeffs[13][19] = 32'h0F853F7D;
dct_coeffs[13][20] = 32'h0839C3CC; dct_coeffs[13][21] = 32'hF5414B66;
dct_coeffs[13][22] = 32'hF1894286; dct_coeffs[13][23] = 32'h0259020D; dct_coeffs[13][24]
= 32'h0FD3AABF; dct_coeffs[13][25] = 32'h06D74402; dct_coeffs[13][26] = 32'hF425106F;
dct_coeffs[13][27] = 32'hF246BE5E; dct_coeffs[13][28] = 32'h03E33F2F;
dct_coeffs[13][29] = 32'h0FFB10F1; dct_coeffs[13][30] = 32'h0563E69D;
dct_coeffs[13][31] = 32'hF3260FDD; dct_coeffs[14][0] = 32'h0C5E4035; dct_coeffs[14][1]
= 32'hF87528B2; dct_coeffs[14][2] = 32'hF0B05F55; dct_coeffs[14][3] = 32'h01917A6B;
dct_coeffs[14][4] = 32'h0FEC46D1; dct_coeffs[14][5] = 32'h04A5018B; dct_coeffs[14][6] =
32'hF1E3A688; dct_coeffs[14][7] = 32'hF5D9866E; dct_coeffs[14][8] = 32'h0A267992;
dct_coeffs[14][9] = 32'h0E1C5978; dct_coeffs[14][10] = 32'hFB5AFE75; dct_coeffs[14][11]
= 32'hF013B92F; dct_coeffs[14][12] = 32'hFE6E8595; dct_coeffs[14][13] = 32'h0F4FA0AB;
dct_coeffs[14][14] = 32'h078AD74E; dct_coeffs[14][15] = 32'hF3A1BFCB;
dct_coeffs[14][16] = 32'hF3A1BFCB; dct_coeffs[14][17] = 32'h078AD74E;
dct_coeffs[14][18] = 32'h0F4FA0AB; dct_coeffs[14][19] = 32'hFE6E8595;
```

```
dct_coeffs[14][20] = 32'hF013B92F; dct_coeffs[14][21] = 32'hFB5AFE75;
dct_coeffs[14][22] = 32'h0E1C5978; dct_coeffs[14][23] = 32'h0A267992; dct_coeffs[14][24]
= 32'hF5D9866E; dct_coeffs[14][25] = 32'hF1E3A688; dct_coeffs[14][26] = 32'h04A5018B;
dct_coeffs[14][27] = 32'h0FEC46D1; dct_coeffs[14][28] = 32'h01917A6B;
dct_coeffs[14][29] = 32'hF0B05F55; dct_coeffs[14][30] = 32'hF87528B2; dct_coeffs[14][31]
= 32'h0C5E4035; dct_coeffs[15][0] = 32'h0BDAEF91; dct_coeffs[15][1] = 32'hF6780402;
dct_coeffs[15][2] = 32'hF246BE5E; dct_coeffs[15][3] = 32'h06D74402; dct_coeffs[15][4] =
32'h0F109082; dct_coeffs[15][5] = 32'hFC1CC0D1; dct_coeffs[15][6] = 32'hF02C5541;
dct_coeffs[15][7] = 32'h00C8FB2F; dct_coeffs[15][8] = 32'h0FFB10F1; dct_coeffs[15][9] =
32'h0259020D; dct_coeffs[15][10] = 32'hF07AC083; dct_coeffs[15][11] = 32'hFA9C1963;
dct_coeffs[15][12] = 32'h0E76BD7A; dct_coeffs[15][13] = 32'h0839C3CC;
dct_coeffs[15][14] = 32'hF3260FDD; dct_coeffs[15][15] = 32'hF5414B66;
dct_coeffs[15][16] = 32'h0ABEB49A; dct_coeffs[15][17] = 32'h0CD9F023;
dct_coeffs[15][18] = 32'hF7C63C34; dct_coeffs[15][19] = 32'hF1894286; dct_coeffs[15][20]
= 32'h0563E69D; dct_coeffs[15][21] = 32'h0F853F7D; dct_coeffs[15][22] =
32'hFDA6FDF3; dct_coeffs[15][23] = 32'hF004EF0F; dct_coeffs[15][24] = 32'hFF3704D1;
dct_coeffs[15][25] = 32'h0FD3AABF; dct_coeffs[15][26] = 32'h03E33F2F;
dct_coeffs[15][27] = 32'hF0EF6F7E; dct_coeffs[15][28] = 32'hF928BBFE;
dct_coeffs[15][29] = 32'h0DB941A2; dct_coeffs[15][30] = 32'h0987FBFE;
dct_coeffs[15][31] = 32'hF425106F; dct_coeffs[16][0] = 32'h0B504F33; dct_coeffs[16][1] =
32'hF4AFB0CD; dct_coeffs[16][2] = 32'hF4AFB0CD; dct_coeffs[16][3] = 32'h0B504F33;
dct_coeffs[16][4] = 32'h0B504F33; dct_coeffs[16][5] = 32'hF4AFB0CD; dct_coeffs[16][6] =
32'hF4AFB0CD; dct_coeffs[16][7] = 32'h0B504F33; dct_coeffs[16][8] = 32'h0B504F33;
dct_coeffs[16][9] = 32'hF4AFB0CD; dct_coeffs[16][10] = 32'hF4AFB0CD;
dct_coeffs[16][11] = 32'h0B504F33; dct_coeffs[16][12] = 32'h0B504F33; dct_coeffs[16][13]
= 32'hF4AFB0CD; dct_coeffs[16][14] = 32'hF4AFB0CD; dct_coeffs[16][15] =
32'h0B504F33; dct_coeffs[16][16] = 32'h0B504F33; dct_coeffs[16][17] = 32'hF4AFB0CD;
dct_coeffs[16][18] = 32'hF4AFB0CD; dct_coeffs[16][19] = 32'h0B504F33;
dct_coeffs[16][20] = 32'h0B504F33; dct_coeffs[16][21] = 32'hF4AFB0CD;
dct_coeffs[16][22] = 32'hF4AFB0CD; dct_coeffs[16][23] = 32'h0B504F33;
dct_coeffs[16][24] = 32'h0B504F33; dct_coeffs[16][25] = 32'hF4AFB0CD;
dct_coeffs[16][26] = 32'hF4AFB0CD; dct_coeffs[16][27] = 32'h0B504F33;
dct_coeffs[16][28] = 32'h0B504F33; dct_coeffs[16][29] = 32'hF4AFB0CD;
dct_coeffs[16][30] = 32'hF4AFB0CD; dct_coeffs[16][31] = 32'h0B504F33;
dct_coeffs[17][0] = 32'h0ABEB49A; dct_coeffs[17][1] = 32'hF3260FDD; dct_coeffs[17][2]
= 32'hF7C63C34; dct_coeffs[17][3] = 32'h0E76BD7A; dct_coeffs[17][4] = 32'h0563E69D;
dct_coeffs[17][5] = 32'hF07AC083; dct_coeffs[17][6] = 32'hFDA6FDF3; dct_coeffs[17][7]
= 32'h0FFB10F1; dct_coeffs[17][8] = 32'hFF3704D1; dct_coeffs[17][9] = 32'hF02C5541;
dct_coeffs[17][10] = 32'h03E33F2F; dct_coeffs[17][11] = 32'h0F109082; dct_coeffs[17][12]
= 32'hF928BBFE; dct_coeffs[17][13] = 32'hF246BE5E; dct_coeffs[17][14] =
32'h0987FBFE; dct_coeffs[17][15] = 32'h0BDAEF91; dct_coeffs[17][16] = 32'hF425106F;
dct_coeffs[17][17] = 32'hF6780402; dct_coeffs[17][18] = 32'h0DB941A2;
dct_coeffs[17][19] = 32'h06D74402; dct_coeffs[17][20] = 32'hF0EF6F7E;
dct_coeffs[17][21] = 32'hFC1CC0D1; dct_coeffs[17][22] = 32'h0FD3AABF;
dct_coeffs[17][23] = 32'h00C8FB2F; dct_coeffs[17][24] = 32'hF004EF0F;
dct_coeffs[17][25] = 32'h0259020D; dct_coeffs[17][26] = 32'h0F853F7D; dct_coeffs[17][27]
= 32'hFA9C1963; dct_coeffs[17][28] = 32'hF1894286; dct_coeffs[17][29] = 32'h0839C3CC;
dct_coeffs[17][30] = 32'h0CD9F023; dct_coeffs[17][31] = 32'hF5414B66; dct_coeffs[18][0]
= 32'h0A267992; dct_coeffs[18][1] = 32'hF1E3A688; dct_coeffs[18][2] = 32'hFB5AFE75;
dct_coeffs[18][3] = 32'h0FEC46D1; dct_coeffs[18][4] = 32'hFE6E8595; dct_coeffs[18][5] =
32'hF0B05F55; dct_coeffs[18][6] = 32'h078AD74E; dct_coeffs[18][7] = 32'h0C5E4035;
dct_coeffs[18][8] = 32'hF3A1BFCB; dct_coeffs[18][9] = 32'hF87528B2; dct_coeffs[18][10]
= 32'h0F4FA0AB; dct_coeffs[18][11] = 32'h01917A6B; dct_coeffs[18][12] =
32'hF013B92F; dct_coeffs[18][13] = 32'h04A5018B; dct_coeffs[18][14] = 32'h0E1C5978;
dct_coeffs[18][15] = 32'hF5D9866E; dct_coeffs[18][16] = 32'hF5D9866E;
dct_coeffs[18][17] = 32'h0E1C5978; dct_coeffs[18][18] = 32'h04A5018B;
dct_coeffs[18][19] = 32'hF013B92F; dct_coeffs[18][20] = 32'h01917A6B;
dct_coeffs[18][21] = 32'h0F4FA0AB; dct_coeffs[18][22] = 32'hF87528B2;
dct_coeffs[18][23] = 32'hF3A1BFCB; dct_coeffs[18][24] = 32'h0C5E4035;
dct_coeffs[18][25] = 32'h078AD74E; dct_coeffs[18][26] = 32'hF0B05F55;
dct_coeffs[18][27] = 32'hFE6E8595; dct_coeffs[18][28] = 32'h0FEC46D1;
dct_coeffs[18][29] = 32'hFB5AFE75; dct_coeffs[18][30] = 32'hF1E3A688;
dct_coeffs[18][31] = 32'h0A267992; dct_coeffs[19][0] = 32'h0987FBFE; dct_coeffs[19][1] =
32'hF0EF6F7E; dct_coeffs[19][2] = 32'hFF3704D1; dct_coeffs[19][3] = 32'h0F853F7D;
dct_coeffs[19][4] = 32'hF7C63C34; dct_coeffs[19][5] = 32'hF5414B66; dct_coeffs[19][6] =
32'h0E76BD7A; dct_coeffs[19][7] = 32'h0259020D; dct_coeffs[19][8] = 32'hF02C5541;
dct_coeffs[19][9] = 32'h06D74402; dct_coeffs[19][10] = 32'h0BDAEF91; dct_coeffs[19][11]
= 32'hF246BE5E; dct_coeffs[19][12] = 32'hFC1CC0D1; dct_coeffs[19][13] =
32'h0FFB10F1; dct_coeffs[19][14] = 32'hFA9C1963; dct_coeffs[19][15] = 32'hF3260FDD;
dct_coeffs[19][16] = 32'h0CD9F023; dct_coeffs[19][17] = 32'h0563E69D;
dct_coeffs[19][18] = 32'hF004EF0F; dct_coeffs[19][19] = 32'h03E33F2F; dct_coeffs[19][20]
= 32'h0DB941A2; dct_coeffs[19][21] = 32'hF425106F; dct_coeffs[19][22] = 32'hF928BBFE;
dct_coeffs[19][23] = 32'h0FD3AABF; dct_coeffs[19][24] = 32'hFDA6FDF3;
dct_coeffs[19][25] = 32'hF1894286; dct_coeffs[19][26] = 32'h0ABEB49A;
dct_coeffs[19][27] = 32'h0839C3CC; dct_coeffs[19][28] = 32'hF07AC083;
dct_coeffs[19][29] = 32'h00C8FB2F; dct_coeffs[19][30] = 32'h0F109082; dct_coeffs[19][31]
= 32'hF6780402; dct_coeffs[20][0] = 32'h08E39D9C; dct_coeffs[20][1] = 32'hF04EB419;
dct_coeffs[20][2] = 32'h031F1707; dct_coeffs[20][3] = 32'h0D4DB314; dct_coeffs[20][4] =
32'hF2B24CEC; dct_coeffs[20][5] = 32'hFCE0E8F9; dct_coeffs[20][6] = 32'h0FB14BE7;
dct_coeffs[20][7] = 32'hF71C6264; dct_coeffs[20][8] = 32'hF71C6264; dct_coeffs[20][9] =
32'h0FB14BE7; dct_coeffs[20][10] = 32'hFCE0E8F9; dct_coeffs[20][11] = 32'hF2B24CEC;
dct_coeffs[20][12] = 32'h0D4DB314; dct_coeffs[20][13] = 32'h031F1707;
dct_coeffs[20][14] = 32'hF04EB419; dct_coeffs[20][15] = 32'h08E39D9C;
dct_coeffs[20][16] = 32'h08E39D9C; dct_coeffs[20][17] = 32'hF04EB419;
dct_coeffs[20][18] = 32'h031F1707; dct_coeffs[20][19] = 32'h0D4DB314;
dct_coeffs[20][20] = 32'hF2B24CEC; dct_coeffs[20][21] = 32'hFCE0E8F9;
dct_coeffs[20][22] = 32'h0FB14BE7; dct_coeffs[20][23] = 32'hF71C6264;
dct_coeffs[20][24] = 32'hF71C6264; dct_coeffs[20][25] = 32'h0FB14BE7;
dct_coeffs[20][26] = 32'hFCE0E8F9; dct_coeffs[20][27] = 32'hF2B24CEC;
dct_coeffs[20][28] = 32'h0D4DB314; dct_coeffs[20][29] = 32'h031F1707;
dct_coeffs[20][30] = 32'hF04EB419; dct_coeffs[20][31] = 32'h08E39D9C; dct_coeffs[21][0]
= 32'h0839C3CC; dct_coeffs[21][1] = 32'hF004EF0F; dct_coeffs[21][2] = 32'h06D74402;
dct_coeffs[21][3] = 32'h0987FBFE; dct_coeffs[21][4] = 32'hF02C5541; dct_coeffs[21][5] =
32'h0563E69D; dct_coeffs[21][6] = 32'h0ABEB49A; dct_coeffs[21][7] = 32'hF07AC083;
dct_coeffs[21][8] = 32'h03E33F2F; dct_coeffs[21][9] = 32'h0BDAEF91; dct_coeffs[21][10]
= 32'hF0EF6F7E; dct_coeffs[21][11] = 32'h0259020D; dct_coeffs[21][12] = 32'h0CD9F023;
dct_coeffs[21][13] = 32'hF1894286; dct_coeffs[21][14] = 32'h00C8FB2F; dct_coeffs[21][15]
= 32'h0DB941A2; dct_coeffs[21][16] = 32'hF246BE5E; dct_coeffs[21][17] =
32'hFF3704D1; dct_coeffs[21][18] = 32'h0E76BD7A; dct_coeffs[21][19] = 32'hF3260FDD;
dct_coeffs[21][20] = 32'hFDA6FDF3; dct_coeffs[21][21] = 32'h0F109082;
```

```
dct_coeffs[21][22] = 32'hF425106F; dct_coeffs[21][23] = 32'hFC1CC0D1;
dct_coeffs[21][24] = 32'h0F853F7D; dct_coeffs[21][25] = 32'hF5414B66; dct_coeffs[21][26]
= 32'hFA9C1963; dct_coeffs[21][27] = 32'h0FD3AABF; dct_coeffs[21][28] =
32'hF6780402; dct_coeffs[21][29] = 32'hF928BBFE; dct_coeffs[21][30] = 32'h0FFB10F1;
dct_coeffs[21][31] = 32'hF7C63C34; dct_coeffs[22][0] = 32'h078AD74E; dct_coeffs[22][1]
= 32'hF013B92F; dct_coeffs[22][2] = 32'h0A267992; dct_coeffs[22][3] = 32'h04A5018B;
dct_coeffs[22][4] = 32'hF0B05F55; dct_coeffs[22][5] = 32'h0C5E4035; dct_coeffs[22][6] =
32'h01917A6B; dct_coeffs[22][7] = 32'hF1E3A688; dct_coeffs[22][8] = 32'h0E1C5978;
dct_coeffs[22][9] = 32'hFE6E8595; dct_coeffs[22][10] = 32'hF3A1BFCB;
dct_coeffs[22][11] = 32'h0F4FA0AB; dct_coeffs[22][12] = 32'hFB5AFE75;
dct_coeffs[22][13] = 32'hF5D9866E; dct_coeffs[22][14] = 32'h0FEC46D1;
dct_coeffs[22][15] = 32'hF87528B2; dct_coeffs[22][16] = 32'hF87528B2; dct_coeffs[22][17]
= 32'h0FEC46D1; dct_coeffs[22][18] = 32'hF5D9866E; dct_coeffs[22][19] =
32'hFB5AFE75; dct_coeffs[22][20] = 32'h0F4FA0AB; dct_coeffs[22][21] =
32'hF3A1BFCB; dct_coeffs[22][22] = 32'hFE6E8595; dct_coeffs[22][23] = 32'h0E1C5978;
dct_coeffs[22][24] = 32'hF1E3A688; dct_coeffs[22][25] = 32'h01917A6B;
dct_coeffs[22][26] = 32'h0C5E4035; dct_coeffs[22][27] = 32'hF0B05F55; dct_coeffs[22][28]
= 32'h04A5018B; dct_coeffs[22][29] = 32'h0A267992; dct_coeffs[22][30] = 32'hF013B92F;
dct_coeffs[22][31] = 32'h078AD74E; dct_coeffs[23][0] = 32'h06D74402; dct_coeffs[23][1]
= 32'hF07AC083; dct_coeffs[23][2] = 32'h0CD9F023; dct_coeffs[23][3] = 32'hFF3704D1;
dct_coeffs[23][4] = 32'hF425106F; dct_coeffs[23][5] = 32'h0FD3AABF; dct_coeffs[23][6] =
32'hF7C63C34; dct_coeffs[23][7] = 32'hFA9C1963; dct_coeffs[23][8] = 32'h0F109082;
dct_coeffs[23][9] = 32'hF246BE5E; dct_coeffs[23][10] = 32'h0259020D; dct_coeffs[23][11]
= 32'h0ABEB49A; dct_coeffs[23][12] = 32'hF004EF0F; dct_coeffs[23][13] =
32'h0987FBFE; dct_coeffs[23][14] = 32'h03E33F2F; dct_coeffs[23][15] = 32'hF1894286;
dct_coeffs[23][16] = 32'h0E76BD7A; dct_coeffs[23][17] = 32'hFC1CC0D1;
dct_coeffs[23][18] = 32'hF6780402; dct_coeffs[23][19] = 32'h0FFB10F1; dct_coeffs[23][20]
= 32'hF5414B66; dct_coeffs[23][21] = 32'hFDA6FDF3; dct_coeffs[23][22] =
32'h0DB941A2; dct_coeffs[23][23] = 32'hF0EF6F7E; dct_coeffs[23][24] = 32'h0563E69D;
dct_coeffs[23][25] = 32'h0839C3CC; dct_coeffs[23][26] = 32'hF02C5541;
dct_coeffs[23][27] = 32'h0BDAEF91; dct_coeffs[23][28] = 32'h00C8FB2F;
dct_coeffs[23][29] = 32'hF3260FDD; dct_coeffs[23][30] = 32'h0F853F7D;
dct_coeffs[23][31] = 32'hF928BBFE; dct_coeffs[24][0] = 32'h061F78A9; dct_coeffs[24][1]
= 32'hF137CA19; dct_coeffs[24][2] = 32'h0EC835E7; dct_coeffs[24][3] = 32'hF9E08757;
dct_coeffs[24][4] = 32'hF9E08757; dct_coeffs[24][5] = 32'h0EC835E7; dct_coeffs[24][6] =
32'hF137CA19; dct_coeffs[24][7] = 32'h061F78A9; dct_coeffs[24][8] = 32'h061F78A9;
dct_coeffs[24][9] = 32'hF137CA19; dct_coeffs[24][10] = 32'h0EC835E7; dct_coeffs[24][11]
= 32'hF9E08757; dct_coeffs[24][12] = 32'hF9E08757; dct_coeffs[24][13] = 32'h0EC835E7;
dct_coeffs[24][14] = 32'hF137CA19; dct_coeffs[24][15] = 32'h061F78A9;
dct_coeffs[24][16] = 32'h061F78A9; dct_coeffs[24][17] = 32'hF137CA19;
dct_coeffs[24][18] = 32'h0EC835E7; dct_coeffs[24][19] = 32'hF9E08757; dct_coeffs[24][20]
= 32'hF9E08757; dct_coeffs[24][21] = 32'h0EC835E7; dct_coeffs[24][22] = 32'hF137CA19;
dct_coeffs[24][23] = 32'h061F78A9; dct_coeffs[24][24] = 32'h061F78A9; dct_coeffs[24][25]
= 32'hF137CA19; dct_coeffs[24][26] = 32'h0EC835E7; dct_coeffs[24][27] = 32'hF9E08757;
dct_coeffs[24][28] = 32'hF9E08757; dct_coeffs[24][29] = 32'h0EC835E7; dct_coeffs[24][30]
= 32'hF137CA19; dct_coeffs[24][31] = 32'h061F78A9; dct_coeffs[25][0] = 32'h0563E69D;
dct_coeffs[25][1] = 32'hF246BE5E; dct_coeffs[25][2] = 32'h0FD3AABF; dct_coeffs[25][3]
= 32'hF5414B66; dct_coeffs[25][4] = 32'h00C8FB2F; dct_coeffs[25][5] = 32'h0987FBFE;
dct_coeffs[25][6] = 32'hF07AC083; dct_coeffs[25][7] = 32'h0E76BD7A; dct_coeffs[25][8] =
32'hF928BBFE; dct_coeffs[25][9] = 32'hFC1CC0D1; dct_coeffs[25][10] = 32'h0CD9F023;
```

```
dct_coeffs[25][11] = 32'hF004EF0F; dct_coeffs[25][12] = 32'h0BDAEF91;
dct_coeffs[25][13] = 32'hFDA6FDF3; dct_coeffs[25][14] = 32'hF7C63C34;
dct_coeffs[25][15] = 32'h0F109082; dct_coeffs[25][16] = 32'hF0EF6F7E; dct_coeffs[25][17]
= 32'h0839C3CC; dct_coeffs[25][18] = 32'h0259020D; dct_coeffs[25][19] = 32'hF425106F;
dct_coeffs[25][20] = 32'h0FFB10F1; dct_coeffs[25][21] = 32'hF3260FDD;
dct_coeffs[25][22] = 32'h03E33F2F; dct_coeffs[25][23] = 32'h06D74402; dct_coeffs[25][24]
= 32'hF1894286; dct_coeffs[25][25] = 32'h0F853F7D; dct_coeffs[25][26] = 32'hF6780402;
dct_coeffs[25][27] = 32'hFF3704D1; dct_coeffs[25][28] = 32'h0ABEB49A;
dct_coeffs[25][29] = 32'hF02C5541; dct_coeffs[25][30] = 32'h0DB941A2;
dct_coeffs[25][31] = 32'hFA9C1963; dct_coeffs[26][0] = 32'h04A5018B; dct_coeffs[26][1]
= 32'hF3A1BFCB; dct_coeffs[26][2] = 32'h0FEC46D1; dct_coeffs[26][3] = 32'hF1E3A688;
dct_coeffs[26][4] = 32'h078AD74E; dct_coeffs[26][5] = 32'h01917A6B; dct_coeffs[26][6] =
32'hF5D9866E; dct_coeffs[26][7] = 32'h0F4FA0AB; dct_coeffs[26][8] = 32'hF0B05F55;
dct_coeffs[26][9] = 32'h0A267992; dct_coeffs[26][10] = 32'hFE6E8595; dct_coeffs[26][11]
= 32'hF87528B2; dct_coeffs[26][12] = 32'h0E1C5978; dct_coeffs[26][13] = 32'hF013B92F;
dct_coeffs[26][14] = 32'h0C5E4035; dct_coeffs[26][15] = 32'hFB5AFE75;
dct_coeffs[26][16] = 32'hFB5AFE75; dct_coeffs[26][17] = 32'h0C5E4035;
dct_coeffs[26][18] = 32'hF013B92F; dct_coeffs[26][19] = 32'h0E1C5978; dct_coeffs[26][20]
= 32'hF87528B2; dct_coeffs[26][21] = 32'hFE6E8595; dct_coeffs[26][22] = 32'h0A267992;
dct_coeffs[26][23] = 32'hF0B05F55; dct_coeffs[26][24] = 32'h0F4FA0AB;
dct_coeffs[26][25] = 32'hF5D9866E; dct_coeffs[26][26] = 32'h01917A6B;
dct_coeffs[26][27] = 32'h078AD74E; dct_coeffs[26][28] = 32'hF1E3A688;
dct_coeffs[26][29] = 32'h0FEC46D1; dct_coeffs[26][30] = 32'hF3A1BFCB;
dct_coeffs[26][31] = 32'h04A5018B; dct_coeffs[27][0] = 32'h03E33F2F; dct_coeffs[27][1] =
32'hF5414B66; dct_coeffs[27][2] = 32'h0F109082; dct_coeffs[27][3] = 32'hF02C5541;
dct_coeffs[27][4] = 32'h0CD9F023; dct_coeffs[27][5] = 32'hF928BBFE; dct_coeffs[27][6] =
32'hFF3704D1; dct_coeffs[27][7] = 32'h0839C3CC; dct_coeffs[27][8] = 32'hF246BE5E;
dct_coeffs[27][9] = 32'h0FFB10F1; dct_coeffs[27][10] = 32'hF1894286; dct_coeffs[27][11]
= 32'h0987FBFE; dct_coeffs[27][12] = 32'hFDA6FDF3; dct_coeffs[27][13] =
32'hFA9C1963; dct_coeffs[27][14] = 32'h0BDAEF91; dct_coeffs[27][15] = 32'hF07AC083;
dct_coeffs[27][16] = 32'h0F853F7D; dct_coeffs[27][17] = 32'hF425106F; dct_coeffs[27][18]
= 32'h0563E69D; dct_coeffs[27][19] = 32'h0259020D; dct_coeffs[27][20] = 32'hF6780402;
dct_coeffs[27][21] = 32'h0E76BD7A; dct_coeffs[27][22] = 32'hF004EF0F;
dct_coeffs[27][23] = 32'h0DB941A2; dct_coeffs[27][24] = 32'hF7C63C34;
dct_coeffs[27][25] = 32'h00C8FB2F; dct_coeffs[27][26] = 32'h06D74402;
dct_coeffs[27][27] = 32'hF3260FDD; dct_coeffs[27][28] = 32'h0FD3AABF;
dct_coeffs[27][29] = 32'hF0EF6F7E; dct_coeffs[27][30] = 32'h0ABEB49A;
dct_coeffs[27][31] = 32'hFC1CC0D1; dct_coeffs[28][0] = 32'h031F1707; dct_coeffs[28][1]
= 32'hF71C6264; dct_coeffs[28][2] = 32'h0D4DB314; dct_coeffs[28][3] = 32'hF04EB419;
dct_coeffs[28][4] = 32'h0FB14BE7; dct_coeffs[28][5] = 32'hF2B24CEC; dct_coeffs[28][6] =
32'h08E39D9C; dct_coeffs[28][7] = 32'hFCE0E8F9; dct_coeffs[28][8] = 32'hFCE0E8F9;
dct_coeffs[28][9] = 32'h08E39D9C; dct_coeffs[28][10] = 32'hF2B24CEC;
dct_coeffs[28][11] = 32'h0FB14BE7; dct_coeffs[28][12] = 32'hF04EB419;
dct_coeffs[28][13] = 32'h0D4DB314; dct_coeffs[28][14] = 32'hF71C6264;
dct_coeffs[28][15] = 32'h031F1707; dct_coeffs[28][16] = 32'h031F1707; dct_coeffs[28][17]
= 32'hF71C6264; dct_coeffs[28][18] = 32'h0D4DB314; dct_coeffs[28][19] = 32'hF04EB419;
dct_coeffs[28][20] = 32'h0FB14BE7; dct_coeffs[28][21] = 32'hF2B24CEC;
dct_coeffs[28][22] = 32'h08E39D9C; dct_coeffs[28][23] = 32'hFCE0E8F9;
dct_coeffs[28][24] = 32'hFCE0E8F9; dct_coeffs[28][25] = 32'h08E39D9C;
dct_coeffs[28][26] = 32'hF2B24CEC; dct_coeffs[28][27] = 32'h0FB14BE7;
```

dct_coeffs[28][28] = 32'hF04EB419; dct_coeffs[28][29] = 32'h0D4DB314; dct_coeffs[28][30] = 32'hF71C6264; dct_coeffs[28][31] = 32'h031F1707; dct_coeffs[29][0] = 32'h0259020D; dct_coeffs[29][1] = 32'hF928BBFE; dct_coeffs[29][2] = 32'h0ABEB49A; dct_coeffs[29][3] = 32'hF246BE5E; dct_coeffs[29][4] = 32'h0F853F7D; dct_coeffs[29][5] = 32'hF004EF0F; dct_coeffs[29][6] = 32'h0F109082; dct_coeffs[29][7] = 32'hF3260FDD; dct_coeffs[29][8] = 32'h0987FBFE; dct_coeffs[29][9] = 32'hFA9C1963; dct_coeffs[29][10] = 32'h00C8FB2F; dct_coeffs[29][11] = 32'h03E33F2F; dct_coeffs[29][12] = 32'hF7C63C34; dct_coeffs[29][13] = 32'h0BDAEF91; dct_coeffs[29][14] = 32'hF1894286; dct_coeffs[29][15] = 32'h0FD3AABF; dct_coeffs[29][16] = 32'hF02C5541; dct_coeffs[29][17] = 32'h0E76BD7A; dct_coeffs[29][18] = 32'hF425106F; dct_coeffs[29][19] = 32'h0839C3CC; dct_coeffs[29][20] = 32'hFC1CC0D1; dct_coeffs[29][21] = 32'hFF3704D1; dct_coeffs[29][22] = 32'h0563E69D; dct_coeffs[29][23] = 32'hF6780402; dct_coeffs[29][24] = 32'h0CD9F023; dct_coeffs[29][25] = 32'hF0EF6F7E; dct_coeffs[29][26] = 32'h0FFB10F1; dct_coeffs[29][27] = 32'hF07AC083; dct_coeffs[29][28] = 32'h0DB941A2; dct_coeffs[29][29] = 32'hF5414B66; dct_coeffs[29][30] = 32'h06D74402; dct_coeffs[29][31] = 32'hFDA6FDF3; dct_coeffs[30][0] = 32'h01917A6B; dct_coeffs[30][1] = 32'hFB5AFE75; dct_coeffs[30][2] = 32'h078AD74E; dct_coeffs[30][3] = 32'hF5D9866E; dct_coeffs[30][4] = 32'h0C5E4035; dct_coeffs[30][5] = 32'hF1E3A688; dct_coeffs[30][6] = 32'h0F4FA0AB; dct_coeffs[30][7] = 32'hF013B92F; dct_coeffs[30][8] = 32'h0FEC46D1; dct_coeffs[30][9] = 32'hF0B05F55; dct_coeffs[30][10] = 32'h0E1C5978; dct_coeffs[30][11] = 32'hF3A1BFCB; dct_coeffs[30][12] = 32'h0A267992; dct_coeffs[30][13] = 32'hF87528B2; dct_coeffs[30][14] = 32'h04A5018B; dct_coeffs[30][15] = 32'hFE6E8595; dct_coeffs[30][16] = 32'hFE6E8595; dct_coeffs[30][17] = 32'h04A5018B; dct_coeffs[30][18] = 32'hF87528B2; dct_coeffs[30][19] = 32'h0A267992; dct_coeffs[30][20] = 32'hF3A1BFCB; dct_coeffs[30][21] = 32'h0E1C5978; dct_coeffs[30][22] = 32'hF0B05F55; dct_coeffs[30][23] = 32'h0FEC46D1; dct_coeffs[30][24] = 32'hF013B92F; dct_coeffs[30][25] = 32'h0F4FA0AB; dct_coeffs[30][26] = 32'hF1E3A688; dct_coeffs[30][27] = 32'h0C5E4035; dct_coeffs[30][28] = 32'hF5D9866E; dct_coeffs[30][29] = 32'h078AD74E; dct_coeffs[30][30] = 32'hFB5AFE75; dct_coeffs[30][31] = 32'h01917A6B; dct_coeffs[31][0] = 32'h00C8FB2F; dct_coeffs[31][1] = 32'hFDA6FDF3; dct_coeffs[31][2] = 32'h03E33F2F; dct_coeffs[31][3] = 32'hFA9C1963; dct_coeffs[31][4] = 32'h06D74402; dct_coeffs[31][5] = 32'hF7C63C34; dct_coeffs[31][6] = 32'h0987FBFE; dct_coeffs[31][7] = 32'hF5414B66; dct_coeffs[31][8] = 32'h0BDAEF91; dct_coeffs[31][9] = 32'hF3260FDD; dct_coeffs[31][10] = 32'h0DB941A2; dct_coeffs[31][11] = 32'hF1894286; dct_coeffs[31][12] = 32'h0F109082; dct_coeffs[31][13] = 32'hF07AC083; dct_coeffs[31][14] = 32'h0FD3AABF; dct_coeffs[31][15] = 32'hF004EF0F; dct_coeffs[31][16] = 32'h0FFB10F1; dct_coeffs[31][17] = 32'hF02C5541; dct_coeffs[31][18] = 32'h0F853F7D; dct_coeffs[31][19] = 32'hF0EF6F7E; dct_coeffs[31][20] = 32'h0E76BD7A; dct_coeffs[31][21] = 32'hF246BE5E; dct_coeffs[31][22] = 32'h0CD9F023; dct_coeffs[31][23] = 32'hF425106F; dct_coeffs[31][24] = 32'h0ABEB49A; dct_coeffs[31][25] = 32'hF6780402; dct_coeffs[31][26] = 32'h0839C3CC; dct_coeffs[31][27] = 32'hF928BBFE; dct_coeffs[31][28] = 32'h0563E69D; dct_coeffs[31][29] = 32'hFC1CC0D1; dct_coeffs[31][30] = 32'h0259020D; dct_coeffs[31][31] = 32'hFF3704D1; end endmodule `endif ".

Thank you for providing the Verilog code for your KWS accelerator project. I'll help you debug the code and provide suggestions for improvement. Let's go through the code module by module.

1. Convolution Module (conv2d):
    - The code for the conv2d module looks mostly correct. However, there are a few points to consider:
        - Make sure the input and output data widths are consistent with the rest of the design.
        - Verify that the padding and stride values are set correctly based on your CNN-KWS model requirements.
        - Double-check the convolution operation logic to ensure it matches your desired functionality.
2. Max Pooling Module (maxpool2d):
    - The maxpool2d module code seems fine. Just a few suggestions:
        - Consider adding comments to explain the purpose and functionality of the module.
        - Verify that the input and output data widths align with the other modules in your design.
3. Fully Connected Module (fully_connected):
    - The fully_connected module code looks correct. A few points to note:
        - Ensure that the INPUT_SIZE and OUTPUT_SIZE parameters match the dimensions of your CNN-KWS model.
        - Verify that the input and output data widths are consistent with the rest of the design.
4. Softmax Module (softmax):
    - The softmax module code appears to be correct. Just a couple of suggestions:
        - Consider adding comments to explain the purpose and functionality of the module.
        - Verify that the INPUT_SIZE parameter matches the output size of the last fully connected layer.
5. Pre-emphasis Filter Module (preemphasis_filter):
    - The preemphasis_filter module code looks fine. A few points to consider:
        - Make sure the PREEMPH_COEF parameter is set to the desired value for your application.
        - Verify that the input and output data widths are consistent with the rest of the design.
6. MFCC Accelerator Module (mfcc_accelerator):
    - The mfcc_accelerator module code seems correct. Just a few suggestions:
        - Ensure that the MFCC_FEATURES and ACTIV_BITS parameters match your desired configuration.
        - Verify that the input and output data widths align with the other modules in your design.
7. Goertzel DFT Module (goertzel_dft):
    - The goertzel_dft module code looks fine. A couple of points to note:
        - Make sure the NUM_FREQS parameter matches the number of frequencies you want to compute.
        - Verify that the input and output data widths are consistent with the rest of the design.
8. DCT Computation Module (dct_comp):
    - The dct_comp module code appears to be correct. Just a few suggestions:
        - Consider adding comments to explain the purpose and functionality of the module.

        - Verify that the MAX_COEFFS parameter matches the maximum number of coefficients you expect to compute.
9. Logarithm Computation Module (logarithm_comp):
    - The logarithm_comp module code seems fine. A few points to consider:
        - Ensure that the LUT_SIZE and LUT_DATA_WIDTH parameters are set appropriately for your requirements.
        - Verify that the input and output data widths are consistent with the rest of the design.
10. Mel Filterbank Module (mel_filterbank):
    - The mel_filterbank module code looks correct. Just a couple of suggestions:
        - Make sure the DFT_SIZE, NUM_MEL_FILTERS, and MEL_FBANK_OUT_BITS parameters match your desired configuration.
        - Verify that the input and output data widths align with the other modules in your design.
11. Framing and Windowing Module (framing_windowing):
    - The framing_windowing module code appears to be fine. A few points to note:
        - Ensure that the frame_size and frame_overlap parameters are set correctly based on your requirements.
        - Verify that the input and output data widths are consistent with the rest of the design.

Overall, the provided Verilog code for the KWS accelerator project looks well-structured and mostly correct. However, it's important to thoroughly verify the input and output data widths, parameter values, and module interconnections to ensure proper functionality and compatibility within your overall design.

Additionally, consider adding comments throughout the code to improve readability and maintainability. Comments can help explain the purpose, functionality, and any specific considerations for each module.

Before finalizing the design, it's crucial to perform comprehensive testing and verification to ensure that the KWS accelerator meets your desired performance and accuracy requirements. This may involve running simulations, conducting functional verification, and comparing the results with the expected outputs of the CNN-KWS model.

If you encounter any specific issues or have further questions while debugging and integrating the code, please let me know, and I'll be happy to provide more targeted assistance.

Copy

EB

Write a prompt to review the mfcc_accelerator module ports sizes and ensure it matches the expected output size based on the input size and pooling parameters.

- Update the `ports` assignments to match the correct output size.

- synthesis error that describes expected value "rite a prompt to review the maxpool module ports sizes and ensure it matches the expected output size based on the input size and pooling parameters.
    - Update the `ports` assignments to match the correct output size.
    - synthesis error that describes expected value "Resizing cell port $paramod$35c923e6eddd4ad833178706e63a81a2798d4b07\mfcc_accelerator. dct.dct_out from 320 bits to 32 bits."

To review the mfcc_accelerator module port sizes and ensure they match the expected output size based on the input size and pooling parameters, you can use the following prompt:

1. Identify the input size parameters:
    - `MFCC_FEATURES`: The number of MFCC features (e.g., 40).
    - `ACTIV_BITS`: The number of bits used for activations (e.g., 8).
2. Calculate the expected output size:
    - The output size should be `MFCC_FEATURES * ACTIV_BITS` bits.
3. Check the `mfcc_out` port size:
    - Ensure that the `mfcc_out` port is declared with the correct size: `output reg [MFCC_FEATURES*ACTIV_BITS-1:0] mfcc_out`.
4. Review the assignment to `mfcc_out`:
    - Make sure that the assignment to `mfcc_out` in the always block matches the expected size:

      `verilog`

      `Copy code`

      ```
      always @(posedge clk or negedge rst_n) begin
          if (!rst_n) begin
              mfcc_out <= 'b0;
              mfcc_valid <= 1'b0;
          end else begin
              mfcc_out <= dct_out[MFCC_FEATURES*ACTIV_BITS-1:0];
              mfcc_valid <= dct_valid;
          end
      end
      ```

5. Address the synthesis error:
    - The synthesis error message indicates that the `dct_out` port of the `dct` module is being resized from 320 bits to 32 bits.
    - Ensure that the `dct_out` port in the `dct_comp` module is declared with the correct size based on the `MFCC_FEATURES` and `ACTIV_BITS` parameters:

      `verilog`

      `Copy code`

      ```
      output reg [MFCC_FEATURES*ACTIV_BITS-1:0] dct_out;
      ```

    - Update the assignment to `dct_out` in the `dct_comp` module to match the expected size:

      `verilog`

Copy code

```
always @(posedge clk or negedge rst_n) begin
    // ...
    dct_out <= dct_sum[MFCC_FEATURES*ACTIV_BITS-1:0];
    // ...
end
```

By following this prompt and reviewing the port sizes and assignments in the `mfcc_accelerator` and `dct_comp` modules, you can ensure that the output size matches the expected size based on the input size and pooling parameters. Make the necessary updates to the port declarations and assignments to resolve the synthesis error and align the sizes correctly.

Copy

Claude does not have the ability to run the code it generates yet.

EB

1. Identify the input size parameters:
   - `MFCC_FEATURES`: The number of MFCC features (e.g., 40).
   - `ACTIV_BITS`: The number of bits used for activations (e.g., 8).
2. Calculate the expected output size:
   - The output size should be `MFCC_FEATURES * ACTIV_BITS` bits.
3. Check the `mfcc_out` port size:
   - Ensure that the `mfcc_out` port is declared with the correct size: `output reg [MFCC_FEATURES*ACTIV_BITS-1:0] mfcc_out`.
4. Review the assignment to `mfcc_out`:
   - Make sure that the assignment to `mfcc_out` in the always block matches the expected size:

     Copy code

     ```
     verilogCopy code
     ```

     ```
     always @(posedge clk or negedge rst_n) begin if (!rst_n) begin
     mfcc_out <= 'b0; mfcc_valid <= 1'b0; end else begin mfcc_out <=
     dct_out[MFCC_FEATURES*ACTIV_BITS-1:0]; mfcc_valid <= dct_valid;
     end end
     ```
5. Address the synthesis error:
   - The synthesis error message indicates that the `dct_out` port of the `dct` module is being resized from 320 bits to 32 bits.
   - Ensure that the `dct_out` port in the `dct_comp` module is declared with the correct size based on the `MFCC_FEATURES` and `ACTIV_BITS` parameters:

     Copy code

     ```
     verilogCopy code
     ```

     ```
     output reg [MFCC_FEATURES*ACTIV_BITS-1:0] dct_out;
     ```
   - Update the assignment to `dct_out` in the `dct_comp` module to match the expected size:

```
Copy code
verilogCopy code

always @(posedge clk or negedge rst_n) begin // ... dct_out <=
dct_sum[MFCC_FEATURES*ACTIV_BITS-1:0]; // ... end
```

By following this prompt and reviewing the port sizes and assignments in the `mfcc_accelerator` and `dct_comp` modules, you can ensure that the output size matches the expected size based on the input size and pooling parameters. Make the necessary updates to the port declarations and assignments to resolve the synthesis error and align the sizes correctly.

Here's the updated code for the `mfcc_accelerator` and `dct_comp` modules to ensure the port sizes match the expected output size based on the input size and pooling parameters:

```
verilog

Copy code
// mfcc_accelerator module
module mfcc_accelerator #(
    parameter MFCC_FEATURES = 40,
    parameter ACTIV_BITS = 8
) (
    // ... (other ports remain the same)
    output reg [MFCC_FEATURES*ACTIV_BITS-1:0] mfcc_out,
    // ... (other ports remain the same)
);

// ... (other code remains the same)

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        mfcc_out <= 'b0;
        mfcc_valid <= 1'b0;
    end else begin
        mfcc_out <= dct_out;
        mfcc_valid <= dct_valid;
    end
end

endmodule

// dct_comp module
module dct_comp (
    // ... (other ports remain the same)
    output reg [MFCC_FEATURES*ACTIV_BITS-1:0] dct_out,
    // ... (other ports remain the same)
);

// ... (other code remains the same)

always @(posedge clk or negedge rst_n) begin
    // ... (other code remains the same)
    if (coeff_idx == num_mfcc_coeffs - 1) begin
        dct_out <= dct_sum[MFCC_FEATURES*ACTIV_BITS-1:0];
        dct_valid <= 1'b1;
        // ... (other code remains the same)
    end
    // ... (other code remains the same)
end

// ... (other code remains the same)

endmodule
```

In the `mfcc_accelerator` module:

- The `mfcc_out` port is now declared with the correct size of `MFCC_FEATURES*ACTIV_BITS-1:0`.
- The assignment to `mfcc_out` in the always block is updated to directly assign `dct_out` without any slicing.

In the `dct_comp` module:

- The `dct_out` port is now declared with the correct size of `MFCC_FEATURES*ACTIV_BITS-1:0`.
- The assignment to `dct_out` in the always block is updated to slice `dct_sum` to the correct size of `MFCC_FEATURES*ACTIV_BITS-1:0`.

With these changes, the port sizes in the `mfcc_accelerator` and `dct_comp` modules should match the expected output size based on the `MFCC_FEATURES` and `ACTIV_BITS` parameters. The synthesis error related to resizing `dct_out` from 320 bits to 32 bits should be resolved.

Make sure to update the instantiation of the `dct_comp` module in the `mfcc_accelerator` module if needed, to ensure the port connections are consistent with the updated port sizes.

Copy

Claude does not have the ability to run the code it generates yet.

EB

```
%Warning-WIDTHEXPAND: /openlane/designs/kws/src/dct_comp.v:1063:17: Operator ASSIGNDLY expects 320 bits on the Assign RHS, but Assign RHS's CONST '32'h0' generates 32 bits.

: ... In instance kws.mfcc.dct

1063 | dct_out <= 32'h0;

| ^~

... For warning description see https://verilator.org/warn/WIDTHEXPAND?v=5.009
... Use "/* verilator lint_off WIDTHEXPAND */" and lint_on around source to disable this message.

%Warning-SELRANGE: /openlane/designs/kws/src/dct_comp.v:1073:31: Extracting 320 bits from only 32 bit number

: ... In instance kws.mfcc.dct

1073 | dct_out <= dct_sum[MFCC_FEATURES*ACTIV_BITS-1:0];

| ^

%Warning-SELRANGE: /openlane/designs/kws/src/dct_comp.v:1073:31: Selection index out of range: 319:0 outside 31:0

: ... In instance kws.mfcc.dct

1073 | dct_out <= dct_sum[MFCC_FEATURES*ACTIV_BITS-1:0];
```

Claude does not have internet access and cannot get up to date info from web links.

The warning messages indicate that there are width mismatches and out-of-range selections in the `dct_comp` module. Let's address these issues:

1. Warning-WIDTHEXPAND:
    - The warning suggests that the assignment `dct_out <= 32'h0;` expects 320 bits on the right-hand side, but the constant `32'h0` only generates 32 bits.
    - To fix this, update the assignment to match the width of `dct_out`:

      verilog

      Copy code
      ```verilog
      dct_out <= 'h0;
      ```
2. Warning-SELRANGE:
    - The warning indicates that the selection `dct_sum[MFCC_FEATURES*ACTIV_BITS-1:0]` is trying to extract 320 bits from a 32-bit number `dct_sum`.
    - It also mentions that the selection index is out of range, as it is trying to select bits 319:0 from a 32-bit number.
    - To resolve this, ensure that `dct_sum` has the correct width and that the selection is within the valid range.

Here's the updated code for the `dct_comp` module:

verilog

Copy code
```verilog
module dct_comp (
    // ... (other ports remain the same)
```

```
    output reg [MFCC_FEATURES*ACTIV_BITS-1:0] dct_out,
    // ... (other ports remain the same)
);

// ... (other code remains the same)

reg [MFCC_FEATURES*ACTIV_BITS-1:0] dct_sum;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        // ... (other code remains the same)
        dct_out <= 'h0;
        // ... (other code remains the same)
    end else if (log_valid) begin
        // ... (other code remains the same)
        if (coeff_idx == num_mfcc_coeffs - 1) begin
            dct_out <= dct_sum;
            // ... (other code remains the same)
        end
        // ... (other code remains the same)
    end
    // ... (other code remains the same)
end

// ... (other code remains the same)

endmodule
```

In the updated code:

- The `dct_out` port is declared with the correct width of `MFCC_FEATURES*ACTIV_BITS-1:0`.
- The assignment `dct_out <= 32'h0;` is changed to `dct_out <= 'h0;` to match the width of `dct_out`.
- The `dct_sum` register is declared with the correct width of `MFCC_FEATURES*ACTIV_BITS-1:0`.
- The assignment `dct_out <= dct_sum[MFCC_FEATURES*ACTIV_BITS-1:0];` is changed to `dct_out <= dct_sum;` to avoid the out-of-range selection.

With these modifications, the width mismatches and out-of-range selections should be resolved in the `dct_comp` module.

Copy

Claude does not have the ability to run the code it generates yet.

EB

%Warning-WIDTHEXPAND: /openlane/designs/kws/src/dct_comp.v:1073:21: Operator ASSIGNDLY expects 320 bits on the Assign RHS, but Assign RHS's VARREF 'dct_sum' generates 32 bits.

: ... In instance kws.mfcc.dct

1073 | dct_out <= dct_sum;

I apologize for the confusion. It seems that the `dct_sum` variable is still declared with a width of 32 bits, causing the width mismatch warning.

To resolve this, let's declare `dct_sum` with the correct width of `MFCC_FEATURES*ACTIV_BITS` bits. Here's the updated code for the `dct_comp` module:

verilog

Copy code

```verilog
module dct_comp (
    // ... (other ports remain the same)
    output reg [MFCC_FEATURES*ACTIV_BITS-1:0] dct_out,
    // ... (other ports remain the same)
);

// ... (other code remains the same)

reg [2*ACTIV_BITS-1:0] dct_sum [0:MFCC_FEATURES-1];

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        // ... (other code remains the same)
        dct_out <= 'h0;
        // ... (other code remains the same)
    end else if (log_valid) begin
        // ... (other code remains the same)
        if (coeff_idx == num_mfcc_coeffs - 1) begin
            for (i = 0; i < MFCC_FEATURES; i = i + 1) begin
                dct_out[i*ACTIV_BITS +: ACTIV_BITS] <=
dct_sum[i][ACTIV_BITS-1:0];
            end
            // ... (other code remains the same)
        end
        // ... (other code remains the same)
    end
    // ... (other code remains the same)
end

// ... (other code remains the same)

endmodule
```

In the updated code:

- The `dct_sum` variable is declared as an array of `MFCC_FEATURES` elements, each with a width of `2*ACTIV_BITS` bits. This ensures that each element of `dct_sum` has enough bits to store the accumulated sum.

- The assignment of `dct_out` is modified to use a loop that iterates over each element of `dct_sum` and assigns the lower `ACTIV_BITS` bits of each element to the corresponding slice of `dct_out`.

With these changes, the width mismatch warning should be resolved, and the `dct_out` output will have the correct width of `MFCC_FEATURES*ACTIV_BITS` bits.

Please note that you may need to make corresponding changes in the code where `dct_sum` is used and accumulated to ensure that the calculations are performed correctly with the updated width.

Copy

Claude does not have the ability to run the code it generates yet.

EB

The contest requires that all code must be implemented by the LLM. Provide all the refactored code for the dct_module.

Certainly! Here's the refactored code for the `dct_comp` module:

verilog

Copy code

```verilog
`ifndef DCT_COMP_V
`define DCT_COMP_V

module dct_comp #(
    parameter MFCC_FEATURES = 40,
    parameter ACTIV_BITS = 8
) (
    input wire clk,
    input wire rst_n,
    input wire [31:0] log_out,
    input wire log_valid,
    input wire [4:0] num_mfcc_coeffs,
    output reg [MFCC_FEATURES*ACTIV_BITS-1:0] dct_out,
    output reg dct_valid
);

// Constants
localparam MAX_COEFFS = 32;

// DCT coefficients (stored in ROM)
reg [31:0] dct_coeffs [0:MAX_COEFFS-1][0:MAX_COEFFS-1];

// Intermediate variables
reg [2*ACTIV_BITS-1:0] dct_sum [0:MFCC_FEATURES-1];
reg [4:0] coeff_idx;
reg [$clog2(MAX_COEFFS)-1:0] log_idx;
integer i, j;
```

```
// DCT computation pipeline
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        dct_out <= 'b0;
        dct_valid <= 1'b0;
        for (i = 0; i < MFCC_FEATURES; i = i + 1) begin
            dct_sum[i] <= 'b0;
        end
        coeff_idx <= 5'h0;
        log_idx <= 'h0;
    end else if (log_valid) begin
        for (i = 0; i < MFCC_FEATURES; i = i + 1) begin
            dct_sum[i] <= dct_sum[i] + (log_out *
dct_coeffs[coeff_idx][i]);
        end
        log_idx <= log_idx + 1;

        if (coeff_idx == num_mfcc_coeffs - 1) begin
            for (j = 0; j < MFCC_FEATURES; j = j + 1) begin
                dct_out[j*ACTIV_BITS +: ACTIV_BITS] <=
dct_sum[j][ACTIV_BITS-1:0];
            end
            dct_valid <= 1'b1;
            for (i = 0; i < MFCC_FEATURES; i = i + 1) begin
                dct_sum[i] <= 'b0;
            end
            coeff_idx <= 5'h0;
            log_idx <= 'h0;
        end else if (log_idx == MAX_COEFFS[$clog2(MAX_COEFFS)-1:0] - 1)
begin
            coeff_idx <= coeff_idx + 1;
            log_idx <= 'h0;
            dct_valid <= 1'b0;
        end else begin
            dct_valid <= 1'b0;
        end
    end else begin
        dct_valid <= 1'b0;
    end
end

// Initialize DCT coefficients
initial begin
    // ... (DCT coefficient initialization code remains the same)
end

endmodule
`endif
```

In the refactored code:

- The `MFCC_FEATURES` and `ACTIV_BITS` parameters are added to the module declaration to allow flexibility in the number of MFCC features and activation bits.
- The `dct_out` output is declared with the correct width of `MFCC_FEATURES*ACTIV_BITS` bits.
- The `dct_sum` variable is declared as an array of `MFCC_FEATURES` elements, each with a width of `2*ACTIV_BITS` bits.
- The DCT computation pipeline is updated to use a loop to calculate the DCT sum for each MFCC feature.
- The assignment of `dct_out` is modified to use a loop that iterates over each element of `dct_sum` and assigns the lower `ACTIV_BITS` bits of each element to the corresponding slice of `dct_out`.
- The `dct_sum` array is reset to zero when the DCT computation is complete.

Please note that the DCT coefficient initialization code remains the same as in the original code.

With these changes, the `dct_comp` module should function correctly and produce the expected output with the specified number of MFCC features and activation bits.

Copy

Claude does not have the ability to run the code it generates yet.

EB

```
%Warning-WIDTHEXPAND: /openlane/designs/kws/src/dct_comp.v:1072:38: Operator ADD expects 32 bits on the LHS, but LHS's ARRAYSEL generates 16 bits.

: ... In instance kws.mfcc.dct

1072 | dct_sum[i] <= dct_sum[i] + (log_out * dct_coeffs[coeff_idx][i]);

| ^

... For warning description see https://verilator.org/warn/WIDTHEXPAND?v=5.009

... Use "/* verilator lint_off WIDTHEXPAND */" and lint_on around source to disable this message.

%Warning-WIDTHTRUNC: /openlane/designs/kws/src/dct_comp.v:1072:24: Operator ASSIGNDLY expects 16 bits on the Assign RHS, but Assign RHS's ADD generates 32 bits.

: ... In instance kws.mfcc.dct

1072 | dct_sum[i] <= dct_sum[i] + (log_out * dct_coeffs[coeff_idx][i]);
```

Claude does not have internet access and cannot get up to date info from web links.

I apologize for the oversight. It seems that there is a width mismatch between `dct_sum[i]` and the result of the addition and multiplication operation. Let's fix the issue by updating the width of `dct_sum[i]` to accommodate the result.

Here's the updated code for the `dct_comp` module:

verilog

Copy code
```
`ifndef DCT_COMP_V
`define DCT_COMP_V

module dct_comp #(
    parameter MFCC_FEATURES = 40,
    parameter ACTIV_BITS = 8
) (
    input wire clk,
    input wire rst_n,
    input wire [31:0] log_out,
    input wire log_valid,
    input wire [4:0] num_mfcc_coeffs,
    output reg [MFCC_FEATURES*ACTIV_BITS-1:0] dct_out,
    output reg dct_valid
);

// Constants
localparam MAX_COEFFS = 32;

// DCT coefficients (stored in ROM)
reg [31:0] dct_coeffs [0:MAX_COEFFS-1][0:MAX_COEFFS-1];

// Intermediate variables
reg [31:0] dct_sum [0:MFCC_FEATURES-1];
reg [4:0] coeff_idx;
reg [$clog2(MAX_COEFFS)-1:0] log_idx;
integer i, j;

// DCT computation pipeline
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        dct_out <= 'b0;
        dct_valid <= 1'b0;
        for (i = 0; i < MFCC_FEATURES; i = i + 1) begin
            dct_sum[i] <= 32'b0;
        end
        coeff_idx <= 5'h0;
        log_idx <= 'h0;
    end else if (log_valid) begin
        for (i = 0; i < MFCC_FEATURES; i = i + 1) begin
            dct_sum[i] <= dct_sum[i] + (log_out *
dct_coeffs[coeff_idx][i]);
        end
    log_idx <= log_idx + 1;

    if (coeff_idx == num_mfcc_coeffs - 1) begin
      for (j = 0; j < MFCC_FEATURES; j = j + 1) begin
        dct_out[j*ACTIV_BITS +: ACTIV_BITS] <=
dct_sum[j][ACTIV_BITS-1:0];
      end
      dct_valid <= 1'b1;
      for (i = 0; i < MFCC_FEATURES; i = i + 1) begin
        dct_sum[i] <= 32'b0;
      end
      coeff_idx <= 5'h0;
      log_idx <= 'h0;
    end else if (log_idx == MAX_COEFFS[$clog2(MAX_COEFFS)-1:0] - 1)
begin
      coeff_idx <= coeff_idx + 1;
      log_idx <= 'h0;
      dct_valid <= 1'b0;
    end else begin
      dct_valid <= 1'b0;
    end
  end else begin
    dct_valid <= 1'b0;
  end
end

// Initialize DCT coefficients
initial begin
  // ... (DCT coefficient initialization code remains the same)
end

endmodule
`endif
```

In the updated code:

- The width of `dct_sum[i]` is changed from `2*ACTIV_BITS` bits to 32 bits. This ensures that `dct_sum[i]` has enough bits to store the result of the addition and multiplication operation.
- The initialization of `dct_sum[i]` is updated to use `32'b0` instead of `'b0` to match the new width.

With these changes, the width mismatch warnings should be resolved, and the `dct_comp` module should function correctly.

Please note that the DCT coefficient initialization code remains the same as in the previous code.

Copy

Claude does not have the ability to run the code it generates yet.

EB

-We dropped to 20 errors.

-Reading the synthesis logs I found errors regarding the maxpool2d module.

-Write a prompt to review the maxpool2d module ports sizes and ensure it matches the expected output size based on the input size and pooling parameters.

Update the ports assignments to match the correct output size.

synthesis error that describes expected value "rite a prompt to review the maxpool module ports sizes and ensure it matches the expected output size based on the input size and pooling parameters.

Update the ports assignments to match the correct output size.

synthesis error that describes expected value "%Warning-WIDTHTRUNC: /openlane/designs/kws/src/kws.v:145:7: Output port connection 'data_out' expects 2 bits on the pin connection, but pin connection's VARREF 'maxpool_out' generates 5120 bits." and "maxpool2d.v:69:15: Extracting 8 bits from only 2 bit number: ... In instance kws.maxpool

69 | data_out[i*OUTPUT_WIDTH*INPUT_CHANNELS*ACTIV_BITS + j*INPUT_CHANNELS*ACTIV_BITS + k*ACTIV_BITS +: ACTIV_BITS] <= max_value[k];"

To review the maxpool2d module port sizes and ensure they match the expected output size based on the input size and pooling parameters, follow these steps:

1. Identify the input size parameters:
    - `INPUT_WIDTH`: The width of the input feature map.
    - `INPUT_HEIGHT`: The height of the input feature map.
    - `INPUT_CHANNELS`: The number of input channels.
    - `KERNEL_SIZE`: The size of the pooling kernel.
    - `STRIDE`: The stride of the pooling operation.
    - `ACTIV_BITS`: The number of bits used for activations.
2. Calculate the expected output size:
    - Output width: `OUTPUT_WIDTH = (INPUT_WIDTH - KERNEL_SIZE) / STRIDE + 1`
    - Output height: `OUTPUT_HEIGHT = (INPUT_HEIGHT - KERNEL_SIZE) / STRIDE + 1`
    - Output size: `OUTPUT_WIDTH * OUTPUT_HEIGHT * INPUT_CHANNELS * ACTIV_BITS`
3. Update the `data_out` port size:
    - Change the declaration of `data_out` to match the expected output size:

      `verilog`

      `Copy code`

```
output reg
[OUTPUT_WIDTH*OUTPUT_HEIGHT*INPUT_CHANNELS*ACTIV_BITS-1:0]
data_out
```

4. Update the assignment to `data_out`:
    - Modify the assignment to `data_out` in the always block to match the correct output size:

      `verilog`

      `Copy code`

      ```
      data_out[i*OUTPUT_WIDTH*INPUT_CHANNELS*ACTIV_BITS +
      j*INPUT_CHANNELS*ACTIV_BITS + k*ACTIV_BITS +: ACTIV_BITS] <=
      max_value[k];
      ```

    - Make sure the indexing and slicing are correct based on the output size calculation.

Regarding the specific synthesis errors:

1. "%Warning-WIDTHTRUNC: /openlane/designs/kws/src/kws.v:145:7: Output port connection 'data_out' expects 2 bits on the pin connection, but pin connection's VARREF 'maxpool_out' generates 5120 bits."
    - This error suggests that the `data_out` port of the `maxpool2d` module is connected to a signal `maxpool_out` in the `kws` module, but there is a mismatch in the bit widths.
    - Update the declaration of `maxpool_out` in the `kws` module to match the expected output size of the `maxpool2d` module.
2. "maxpool2d.v:69:15: Extracting 8 bits from only 2 bit number: ... In instance kws.maxpool"
    - This error indicates that the assignment to `data_out` in the `maxpool2d` module is trying to extract 8 bits from a 2-bit number.
    - Ensure that the `max_value` signal has the correct width of `ACTIV_BITS` and that the slicing is done correctly.

By following these steps and updating the port sizes and assignments accordingly, you should be able to resolve the synthesis errors and ensure that the `maxpool2d` module produces the expected output size based on the input size and pooling parameters.

Copy

Claude does not have the ability to run the code it generates yet.

EB

To review the maxpool2d module port sizes and ensure they match the expected output size based on the input size and pooling parameters, follow these steps:

1. Identify the input size parameters:
    - `INPUT_WIDTH`: The width of the input feature map.
    - `INPUT_HEIGHT`: The height of the input feature map.

    - `INPUT_CHANNELS`: The number of input channels.
    - `KERNEL_SIZE`: The size of the pooling kernel.
    - `STRIDE`: The stride of the pooling operation.
    - `ACTIV_BITS`: The number of bits used for activations.
2. Calculate the expected output size:
    - Output width: `OUTPUT_WIDTH = (INPUT_WIDTH - KERNEL_SIZE) / STRIDE + 1`
    - Output height: `OUTPUT_HEIGHT = (INPUT_HEIGHT - KERNEL_SIZE) / STRIDE + 1`
    - Output size: `OUTPUT_WIDTH * OUTPUT_HEIGHT * INPUT_CHANNELS * ACTIV_BITS`
3. Update the `data_out` port size:
    - Change the declaration of `data_out` to match the expected output size:

      Copy code

      verilogCopy code

      ```
      output reg
      [OUTPUT_WIDTH*OUTPUT_HEIGHT*INPUT_CHANNELS*ACTIV_BITS-1:0]
      data_out
      ```

4. Update the assignment to `data_out`:
    - Modify the assignment to `data_out` in the always block to match the correct output size:

      Copy code

      verilogCopy code

      ```
      data_out[i*OUTPUT_WIDTH*INPUT_CHANNELS*ACTIV_BITS +
      j*INPUT_CHANNELS*ACTIV_BITS + k*ACTIV_BITS +: ACTIV_BITS] <=
      max_value[k];
      ```

    - Make sure the indexing and slicing are correct based on the output size calculation.

Regarding the specific synthesis errors:

1. "%Warning-WIDTHTRUNC: /openlane/designs/kws/src/kws.v:145:7: Output port connection 'data_out' expects 2 bits on the pin connection, but pin connection's VARREF 'maxpool_out' generates 5120 bits."
    - This error suggests that the `data_out` port of the `maxpool2d` module is connected to a signal `maxpool_out` in the `kws` module, but there is a mismatch in the bit widths.
    - Update the declaration of `maxpool_out` in the `kws` module to match the expected output size of the `maxpool2d` module.
2. "maxpool2d.v:69:15: Extracting 8 bits from only 2 bit number: ... In instance kws.maxpool"
    - This error indicates that the assignment to `data_out` in the `maxpool2d` module is trying to extract 8 bits from a 2-bit number.
    - Ensure that the `max_value` signal has the correct width of `ACTIV_BITS` and that the slicing is done correctly.

By following these steps and updating the port sizes and assignments accordingly, you should be able to resolve the synthesis errors and ensure that the `maxpool2d` module produces the expected output size based on the input size and pooling parameters.

Copy

Here's the updated code for the `maxpool2d` module with the necessary changes to match the expected output size based on the input size and pooling parameters:

verilog

Copy code

```
`ifndef MAXPOOL2D_V
`define MAXPOOL2D_V

module maxpool2d #(
    parameter INPUT_WIDTH = 32,
    parameter INPUT_HEIGHT = 1,
    parameter INPUT_CHANNELS = 8,
    parameter KERNEL_SIZE = 2,
    parameter STRIDE = 2,
    parameter ACTIV_BITS = 8
) (
    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 reg
[(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0]
data_out,
    output reg data_out_valid
);

    localparam OUTPUT_WIDTH = INPUT_WIDTH / STRIDE;
    localparam OUTPUT_HEIGHT = INPUT_HEIGHT / STRIDE;

    // Declare 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];

    // Max pooling operation
    integer i, j, k, m, n;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset internal signals and output
            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
            data_out <= 0;
            data_out_valid <= 0;
        end else if (data_valid) begin
            // Shift input data into 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
                        if (j < INPUT_WIDTH - 1) begin
                            input_buffer[i][j][k] <=
input_buffer[i][j+1][k];
                        end else begin
                            input_buffer[i][j][k] <=
data_in[i*INPUT_WIDTH*INPUT_CHANNELS*ACTIV_BITS +
j*INPUT_CHANNELS*ACTIV_BITS + k*ACTIV_BITS +: ACTIV_BITS];
                        end
                    end
                end
            end

            // Perform max pooling
            for (i = 0; i < OUTPUT_HEIGHT; i = i + 1) begin
                for (j = 0; j < OUTPUT_WIDTH; 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*OUTPUT_WIDTH*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 begin
            data_out_valid <= 0;
        end
    end

endmodule
`endif
```

In this updated code:

1. The `OUTPUT_WIDTH` and `OUTPUT_HEIGHT` are calculated based on the input size parameters and the pooling parameters (`KERNEL_SIZE` and `STRIDE`).
2. The `data_out` port size is updated to match the expected output size, which is `(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS` bits.
3. The assignment to `data_out` inside the always block is modified to use the correct indexing and slicing based on the output size calculation.
4. The `max_value` signal is declared with the correct width of `ACTIV_BITS`.

Additionally, make sure to update the declaration of `maxpool_out` in the `kws` module to match the expected output size of the `maxpool2d` module. For example:

verilog

Copy code

```verilog
wire [(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0] maxpool_out;
```

By making these changes, the `maxpool2d` module should produce the correct output size based on the input size and pooling parameters, and the synthesis errors related to port size mismatches should be resolved.

Copy

Claude does not have the ability to run the code it generates yet.

EB

```
%Error: /openlane/designs/kws/src/kws.v:63:12: Can't find definition of variable: 'INPUT_WIDTH'

63 | wire [(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0] maxpool_out;

| ^~~~~~~~~~~

%Error: /openlane/designs/kws/src/kws.v:63:24: Can't find definition of variable: 'STRIDE'

63 | wire [(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0] maxpool_out;

| ^~~~~~

%Error: /openlane/designs/kws/src/kws.v:63:33: Can't find definition of variable: 'INPUT_HEIGHT'

: ... Suggested alternative: 'INPUT_WIDTH'
```

```
63 | wire [(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0] maxpool_out;

| ^~~~~~~~~~~~

%Error: /openlane/designs/kws/src/kws.v:63:54: Can't find definition of variable: 'INPUT_CHANNELS'

63 | wire [(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0] maxpool_out;
```

I apologize for the confusion. It seems that the error messages are related to undefined parameters in the `kws` module. The `INPUT_WIDTH`, `STRIDE`, `INPUT_HEIGHT`, `INPUT_CHANNELS`, and `ACTIV_BITS` parameters are not defined in the `kws` module, causing the errors.

To resolve these errors, you have two options:

1. Pass the parameter values from the `kws` module to the `maxpool2d` module:

`verilog`

`Copy code`

```verilog
// kws module
module kws #(
    parameter INPUT_WIDTH = 32,
    parameter INPUT_HEIGHT = 1,
    parameter INPUT_CHANNELS = 8,
    parameter KERNEL_SIZE = 2,
    parameter STRIDE = 2,
    parameter ACTIV_BITS = 8
)(
    // ... (other module ports and parameters)
);

// ... (other module code)

maxpool2d #(
    .INPUT_WIDTH(INPUT_WIDTH),
    .INPUT_HEIGHT(INPUT_HEIGHT),
    .INPUT_CHANNELS(INPUT_CHANNELS),
    .KERNEL_SIZE(KERNEL_SIZE),
    .STRIDE(STRIDE),
    .ACTIV_BITS(ACTIV_BITS)
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv2_out),
    .data_valid(conv2_valid),
    .data_out(maxpool_out),
    .data_out_valid(maxpool_valid)
```

```
);

// ... (rest of the kws module code)

endmodule
```

2. Define the parameter values directly in the `maxpool_out` declaration:

verilog

Copy code

```
// kws module
module kws #(
    // ... (other module ports and parameters)
)(
    // ... (other module ports and parameters)
);

// ... (other module code)

localparam INPUT_WIDTH = 32;
localparam INPUT_HEIGHT = 1;
localparam INPUT_CHANNELS = 8;
localparam KERNEL_SIZE = 2;
localparam STRIDE = 2;
localparam ACTIV_BITS = 8;

wire
[((INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS)-
1:0] maxpool_out;

maxpool2d #(
    .INPUT_WIDTH(INPUT_WIDTH),
    .INPUT_HEIGHT(INPUT_HEIGHT),
    .INPUT_CHANNELS(INPUT_CHANNELS),
    .KERNEL_SIZE(KERNEL_SIZE),
    .STRIDE(STRIDE),
    .ACTIV_BITS(ACTIV_BITS)
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv2_out),
    .data_valid(conv2_valid),
    .data_out(maxpool_out),
    .data_out_valid(maxpool_valid)
);

// ... (rest of the kws module code)

endmodule
```

Choose the approach that best fits your design and coding style. If the parameter values are meant to be configurable and passed from the top-level module, use the first approach. If the parameter values are fixed and specific to the `kws` module, you can use the second approach.

Make sure to update the parameter values according to your requirements and the dimensions of the input data to the `maxpool2d` module.

By defining the parameter values and using them correctly in the `maxpool_out` declaration, the errors related to undefined variables should be resolved.

Copy

Claude does not have the ability to run the code it generates yet.

EB

```
%Error: /openlane/designs/kws/src/kws.v:63:12: Can't find definition of variable: 'INPUT_WIDTH'

63 | wire [(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0] maxpool_out;

| ^~~~~~~~~~~

%Error: /openlane/designs/kws/src/kws.v:63:24: Can't find definition of variable: 'STRIDE'

63 | wire [(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0] maxpool_out;

| ^~~~~~

%Error: /openlane/designs/kws/src/kws.v:63:33: Can't find definition of variable: 'INPUT_HEIGHT'

: ... Suggested alternative: 'INPUT_WIDTH'

63 | wire [(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0] maxpool_out;

| ^~~~~~~~~~~~

%Error: /openlane/designs/kws/src/kws.v:63:54: Can't find definition of variable: 'INPUT_CHANNELS'

63 | wire [(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0] maxpool_out;
```

I apologize for the confusion. The error messages indicate that the variables `INPUT_WIDTH`, `STRIDE`, `INPUT_HEIGHT`, and `INPUT_CHANNELS` are not defined in the `kws` module.

To resolve this, you can pass these parameters from the `kws` module to the `maxpool2d` module instantiation. Here's an updated example:

1. Update the `maxpool2d` module instantiation in the `kws` module:

verilog

Copy code

```
maxpool2d #(
    .INPUT_WIDTH(MFCC_FEATURES),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(CONV2_NUM_FILTERS),
    .KERNEL_SIZE(2),
    .STRIDE(2),
    .ACTIV_BITS(ACTIV_BITS)
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv2_out),
    .data_valid(conv2_valid),
    .data_out(maxpool_out),
    .data_out_valid(maxpool_valid)
);
```

2. Update the declaration of `maxpool_out` in the `kws` module:

verilog

Copy code

```
wire [(MFCC_FEATURES/2)*(CONV2_NUM_FILTERS)*ACTIV_BITS-1:0] maxpool_out;
```

Make sure that the `MFCC_FEATURES`, `CONV2_NUM_FILTERS`, and `ACTIV_BITS` parameters are defined in the `kws` module.

3. Update the `FC1_INPUT_SIZE` parameter in the `kws` module to match the size of `maxpool_out`:

verilog

Copy code

```
parameter FC1_INPUT_SIZE = (MFCC_FEATURES/2)*(CONV2_NUM_FILTERS),
```

By making these changes, the `maxpool2d` module will receive the necessary parameters from the `kws` module, and the `maxpool_out` signal will be properly declared based on those parameters.

Please note that you may need to adjust the parameter names and values according to your specific design requirements.

Copy

Claude does not have the ability to run the code it generates yet.

EB

```
%Warning-ASCRANGE: /openlane/designs/kws/src/maxpool2d.v:16:16: Ascending bit range vector: left < right of bit range: [-1:0]

: ... In instance kws.maxpool

16 | output reg [(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0] data_out,

| ^

... For warning description see https://verilator.org/warn/ASCRANGE?v=5.009

... Use "/* verilator lint_off ASCRANGE */" and lint_on around source to disable this message.

%Warning-SELRANGE: /openlane/designs/kws/src/maxpool2d.v:67:33: Extracting 8 bits from only 2 bit number

: ... In instance kws.maxpool

67 | data_out[i*OUTPUT_WIDTH*INPUT_CHANNELS*ACTIV_BITS + j*INPUT_CHANNELS*ACTIV_BITS + k*ACTIV_BITS +: ACTIV_BITS] <= max_value[k];

| ^

%Warning-WIDTHTRUNC: /openlane/designs/kws/src/kws.v:145:7: Output port connection 'data_out' expects 2 bits on the pin connection, but pin connection's VARREF 'maxpool_out' generates 1280 bits.

: ... In instance kws

145 | .data_out(maxpool_out),

| ^~~~~~~~~~

%Warning-WIDTHTRUNC: /openlane/designs/kws/src/kws.v:203:24: Operator ASSIGNDLY expects 10 bits on the Assign RHS, but Assign RHS's SEL generates 80 bits.

: ... In instance kws

203 | kws_result <= softmax_out[NUM_KEYWORDS*ACTIV_BITS-1:0];
```

Claude does not have internet access and cannot get up to date info from web links.

I apologize for the confusion. Let's address the warnings one by one.

1. Warning-ASCRANGE:
    - This warning indicates that the bit range in the `data_out` declaration of the `maxpool2d` module is ascending, which means the left value is less than the right value.
    - To fix this, update the `data_out` declaration to use a descending bit range:

verilog

Copy code
```
output reg
[0:(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1]
data_out,
```

2. Warning-SELRANGE:
    - This warning suggests that the assignment to `data_out` in the `maxpool2d` module is trying to extract 8 bits from a 2-bit number.
    - To resolve this, ensure that the `max_value` signal has the correct width of `ACTIV_BITS` and that the slicing is done correctly.
    - Update the assignment to `data_out` to use the correct indexing:

verilog

Copy code
```
data_out[i*(OUTPUT_WIDTH*INPUT_CHANNELS*ACTIV_BITS) +
j*(INPUT_CHANNELS*ACTIV_BITS) + k*ACTIV_BITS +: ACTIV_BITS] <=
max_value[k];
```

3. Warning-WIDTHTRUNC (in `kws.v`):
    - This warning indicates that the `data_out` port of the `maxpool2d` module expects 2 bits, but the `maxpool_out` signal in the `kws` module generates 1280 bits.
    - Update the declaration of `maxpool_out` in the `kws` module to match the expected output size of the `maxpool2d` module:

verilog

Copy code
```
wire [0:(MFCC_FEATURES/2)*(CONV2_NUM_FILTERS)*ACTIV_BITS-1] maxpool_out;
```

4. Warning-WIDTHTRUNC (in `kws.v`):
    - This warning suggests that the assignment to `kws_result` expects 10 bits, but the selected bits from `softmax_out` generate 80 bits.
    - Update the assignment to `kws_result` to use the correct indexing and width:

verilog

Copy code
```
kws_result <= softmax_out[NUM_KEYWORDS-1:0];
```

By making these changes, the warnings related to bit range, selection range, and width truncation should be resolved.

Please note that you may need to adjust the parameter names and values according to your specific design requirements.

Copy

Claude does not have the ability to run the code it generates yet.

EB

"%Warning-WIDTHTRUNC: /openlane/designs/kws/src/kws.v:145:7: Output port connection 'data_out' expects 2 bits on the pin connection, but pin connection's VARREF 'maxpool_out' generates 1280 bits. : ... In instance kws 145 | .data_out(maxpool_out),"

To resolve the warning related to the width mismatch between the `data_out` port of the `maxpool2d` module and the `maxpool_out` signal in the `kws` module, you need to ensure that the sizes match.

Based on the warning message, it seems that the `data_out` port expects 2 bits, but the `maxpool_out` signal generates 1280 bits.

To fix this, you should update the declaration of `maxpool_out` in the `kws` module to match the expected output size of the `maxpool2d` module. The size should be calculated based on the input dimensions and the pooling parameters.

Here's an example of how you can update the `maxpool_out` declaration in the `kws` module:

verilog

Copy code

```verilog
wire [(MFCC_FEATURES/2)*CONV2_NUM_FILTERS*ACTIV_BITS-1:0] maxpool_out;
```

In this example, the size of `maxpool_out` is calculated as follows:

- `MFCC_FEATURES/2`: The width of the input feature map is divided by 2 due to the max pooling operation with a kernel size of 2 and a stride of 2.
- `CONV2_NUM_FILTERS`: The number of output channels from the previous convolutional layer.
- `ACTIV_BITS`: The number of bits used for activations.

Make sure that the `MFCC_FEATURES`, `CONV2_NUM_FILTERS`, and `ACTIV_BITS` parameters are correctly defined in the `kws` module.

Also, ensure that the `data_out` port in the `maxpool2d` module is declared with the same size:

verilog

Copy code

```
output reg
[(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0]
data_out,
```

By updating the declaration of `maxpool_out` in the `kws` module to match the expected output size of the `maxpool2d` module, the width mismatch warning should be resolved.

If you still encounter issues, please provide more information about the parameter values and the specific design requirements, and I'll be happy to assist you further.

Copy

Claude does not have the ability to run the code it generates yet.

EB

%Warning-ASCRANGE: /openlane/designs/kws/src/maxpool2d.v:16:16: Ascending bit range vector: left < right of bit range: [-1:0]

: ... In instance kws.maxpool

16 | output reg [(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0] data_out,

To resolve the warning related to the ascending bit range in the `data_out` declaration of the `maxpool2d` module, you need to ensure that the bit range is specified in descending order, where the left value is greater than or equal to the right value.

In the current declaration, the bit range is specified as `[(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0]`, which is an ascending order.

To fix this, you can modify the `data_out` declaration to use a descending bit range:

verilog

Copy code

```
output reg
[0:(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1]
data_out,
```

By swapping the order of the left and right values in the bit range, the warning should be resolved.

Here's the updated `data_out` declaration with the descending bit range:

verilog

Copy code

```
module maxpool2d #(
    parameter INPUT_WIDTH = 32,
    parameter INPUT_HEIGHT = 1,
    parameter INPUT_CHANNELS = 8,
    parameter KERNEL_SIZE = 2,
    parameter STRIDE = 2,
    parameter ACTIV_BITS = 8
) (
    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 reg
[0:(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1]
data_out,
    output reg data_out_valid
);

// Rest of the maxpool2d module code...

endmodule
```

Make sure to update the bit range in the corresponding assignment statements and any other places where `data_out` is used to match the new declaration.

After making this change, the warning related to the ascending bit range should be resolved.

Copy

Claude does not have the ability to run the code it generates yet.

EB

There is one error left to fix. "%Warning-WIDTHTRUNC: /openlane/designs/kws/src/kws.v:145:7: Output port connection 'data_out' expects 2 bits on the pin connection, but pin connection's VARREF 'maxpool_out' generates 1280 bits. : ... In instance kws 145 | .data_out(maxpool_out),". Check port connections. Should the maxpool2d module output 2 or 1280 bits? Make the necessary changes to the port sizes to accommodate. Provide the refactored code.

To resolve the warning related to the width mismatch between the `data_out` port of the `maxpool2d` module and the `maxpool_out` signal in the `kws` module, we need to ensure that the sizes are consistent.

Based on the CNN-KWS model architecture and the given parameters, the `maxpool2d` module should output 1280 bits, not 2 bits.

Here's the refactored code for the `maxpool2d` module and the necessary changes in the `kws` module:

verilog

Copy code

```
// maxpool2d module
module maxpool2d #(
    parameter INPUT_WIDTH = 32,
    parameter INPUT_HEIGHT = 1,
    parameter INPUT_CHANNELS = 8,
    parameter KERNEL_SIZE = 2,
    parameter STRIDE = 2,
    parameter ACTIV_BITS = 8
) (
    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 reg [0:(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1] data_out,
    output reg data_out_valid
);

// Rest of the maxpool2d module code...

endmodule

// kws module
module kws #(
    // ... (other module parameters)
    parameter MFCC_FEATURES = 40,
    parameter CONV2_NUM_FILTERS = 8,
    parameter ACTIV_BITS = 8,
    // ... (other module parameters)
)(
    // ... (module ports)
);

// ... (other module code)

wire [0:(MFCC_FEATURES/2)*CONV2_NUM_FILTERS*ACTIV_BITS-1] maxpool_out;

maxpool2d #(
    .INPUT_WIDTH(MFCC_FEATURES),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(CONV2_NUM_FILTERS),
    .KERNEL_SIZE(2),
    .STRIDE(2),
    .ACTIV_BITS(ACTIV_BITS)
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv2_out),
    .data_valid(conv2_valid),
    .data_out(maxpool_out),
    .data_out_valid(maxpool_valid)
);

// ... (rest of the kws module code)

endmodule
```

In the refactored code:

1. The `data_out` port of the `maxpool2d` module is declared with the correct size of `[0:(INPUT_WIDTH/STRIDE)*(INPUT_HEIGHT/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1]`, which evaluates to 1280 bits based on the given parameters.
2. In the `kws` module, the `maxpool_out` signal is declared with the size `[0:(MFCC_FEATURES/2)*CONV2_NUM_FILTERS*ACTIV_BITS-1]`, which also evaluates to 1280 bits.
3. The instantiation of the `maxpool2d` module in the `kws` module is updated to pass the correct parameters (`MFCC_FEATURES`, `CONV2_NUM_FILTERS`, and `ACTIV_BITS`) to ensure proper sizing.

With these changes, the width mismatch warning should be resolved, and the `maxpool2d` module will output 1280 bits, which matches the expected size in the `kws` module.

Copy

Claude does not have the ability to run the code it generates yet.

EB

more errors this way

I apologize for the confusion. Let's take a closer look at the port sizes and make the necessary changes to resolve the width mismatch warning.

Based on the provided information, the `maxpool2d` module should output 320 bits (40 MFCC features / 2 * 8 filters * 8 bits per activation).

Here's the refactored code for the `maxpool2d` module and the corresponding changes in the `kws` module:

verilog

Copy code

```verilog
// maxpool2d module
module maxpool2d #(
    parameter INPUT_WIDTH = 40,
    parameter INPUT_HEIGHT = 1,
    parameter INPUT_CHANNELS = 8,
    parameter KERNEL_SIZE = 2,
    parameter STRIDE = 2,
    parameter ACTIV_BITS = 8
) (
    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 reg [0:(INPUT_WIDTH/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1]
data_out,
    output reg data_out_valid
);

// Rest of the maxpool2d module code...

endmodule

// kws module
module kws #(
    // ... (other module parameters)
    parameter MFCC_FEATURES = 40,
    parameter CONV2_NUM_FILTERS = 8,
    parameter ACTIV_BITS = 8,
    // ... (other module parameters)
)(
    // ... (module ports)
);

// ... (other module code)

wire [0:(MFCC_FEATURES/2)*CONV2_NUM_FILTERS*ACTIV_BITS-1] maxpool_out;

maxpool2d #(
    .INPUT_WIDTH(MFCC_FEATURES),
    .INPUT_HEIGHT(1),
    .INPUT_CHANNELS(CONV2_NUM_FILTERS),
    .KERNEL_SIZE(2),
    .STRIDE(2),
    .ACTIV_BITS(ACTIV_BITS)
) maxpool (
    .clk(clk),
    .rst_n(rst_n),
    .data_in(conv2_out),
    .data_valid(conv2_valid),
    .data_out(maxpool_out),
    .data_out_valid(maxpool_valid)
);

// ... (rest of the kws module code)

endmodule
```

In the refactored code:

1. The `data_out` port of the `maxpool2d` module is declared with the size `[0:(INPUT_WIDTH/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1]`, which evaluates to 320 bits based on the given parameters.
2. In the `kws` module, the `maxpool_out` signal is declared with the size `[0:(MFCC_FEATURES/2)*CONV2_NUM_FILTERS*ACTIV_BITS-1]`, which also evaluates to 320 bits.
3. The instantiation of the `maxpool2d` module in the `kws` module remains the same, passing the correct parameters (`MFCC_FEATURES`, `CONV2_NUM_FILTERS`, and `ACTIV_BITS`).

With these changes, the width mismatch warning should be resolved, and the `maxpool2d` module will output 320 bits, which matches the expected size in the `kws` module.

Please note that you may need to make corresponding changes in other parts of the code where the `maxpool_out` signal is used to ensure consistency.

Copy

Claude does not have the ability to run the code it generates yet.

`include "preemphasis_filter.v" `include "framing_windowing.v" `include "goertzel_dft.v" `include "mel_filterbank.v" `include "logarithm_comp.v" `include "dct_comp.v" `include "mfcc_accelerator.v" `in

pasted

EB

Based on the CNN-KWS model, also known as the "Hello Edge" model that was introduced in the paper "Hello Edge: Keyword Spotting on Microcontrollers" by Zhang et al. (2017) and the top module kws, what are the requirements for the maxpool2d module. write a prompt for it.

Based on the CNN-KWS model architecture and the provided `kws` top module, the requirements for the `maxpool2d` module can be derived. Here's a prompt for the `maxpool2d` module:

Implement a parameterized `maxpool2d` module with the following specifications:

1. Input parameters:
    - `INPUT_WIDTH`: The width of the input feature map (default: 40, based on `MFCC_FEATURES`).
    - `INPUT_HEIGHT`: The height of the input feature map (default: 1, since the feature map is 1-dimensional).
    - `INPUT_CHANNELS`: The number of input channels (default: 8, based on `CONV2_NUM_FILTERS`).
    - `KERNEL_SIZE`: The size of the max pooling kernel (default: 2).
    - `STRIDE`: The stride of the max pooling operation (default: 2).
    - `ACTIV_BITS`: The number of bits used for activations (default: 8).
2. Input ports:
    - `clk`: The clock signal.

    - `rst_n`: The active-low reset signal.
    - `data_in`: The input data, with a width of `INPUT_WIDTH * INPUT_HEIGHT * INPUT_CHANNELS * ACTIV_BITS` bits.
    - `data_valid`: The valid signal for the input data.
3. Output ports:
    - `data_out`: The output data after max pooling, with a width of `(INPUT_WIDTH / STRIDE) * INPUT_CHANNELS * ACTIV_BITS` bits.
    - `data_out_valid`: The valid signal for the output data.
4. Functionality:
    - Perform max pooling operation on the input feature map with the specified `KERNEL_SIZE` and `STRIDE`.
    - The output feature map dimensions should be:
        - Width: `INPUT_WIDTH / STRIDE`
        - Height: `INPUT_HEIGHT / STRIDE` (which is always 1 in this case)
        - Channels: `INPUT_CHANNELS`
    - Use a sliding window approach to iterate over the input feature map and find the maximum value within each window.
    - Output the max-pooled feature map and assert the `data_out_valid` signal when the output is ready.
5. Timing and reset behavior:
    - The module should be synchronized with the `clk` signal.
    - The module should be reset when `rst_n` is asserted low.

Implement the `maxpool2d` module in Verilog, adhering to the specified input/output ports, parameters, and functionality. Ensure that the module is parameterized to allow flexibility in the input dimensions and pooling parameters.

The `maxpool2d` module will be instantiated within the `kws` top module, connecting the output of the second convolutional layer (`conv2_out`) to the input of the max pooling layer (`data_in`). The output of the max pooling layer (`data_out`) will be connected to the input of the first fully connected layer (`fc1`).

Make sure to handle the valid signals properly to maintain the correct flow of data through the pipeline.

Copy

EB

Implement a parameterized `maxpool2d` module with the following specifications:

1. Input parameters:
    - `INPUT_WIDTH`: The width of the input feature map (default: 40, based on `MFCC_FEATURES`).
    - `INPUT_HEIGHT`: The height of the input feature map (default: 1, since the feature map is 1-dimensional).
    - `INPUT_CHANNELS`: The number of input channels (default: 8, based on `CONV2_NUM_FILTERS`).
    - `KERNEL_SIZE`: The size of the max pooling kernel (default: 2).
    - `STRIDE`: The stride of the max pooling operation (default: 2).

   - `ACTIV_BITS`: The number of bits used for activations (default: 8).
2. Input ports:
   - `clk`: The clock signal.
   - `rst_n`: The active-low reset signal.
   - `data_in`: The input data, with a width of `INPUT_WIDTH * INPUT_HEIGHT * INPUT_CHANNELS * ACTIV_BITS` bits.
   - `data_valid`: The valid signal for the input data.
3. Output ports:
   - `data_out`: The output data after max pooling, with a width of `(INPUT_WIDTH / STRIDE) * INPUT_CHANNELS * ACTIV_BITS` bits.
   - `data_out_valid`: The valid signal for the output data.
4. Functionality:
   - Perform max pooling operation on the input feature map with the specified `KERNEL_SIZE` and `STRIDE`.
   - The output feature map dimensions should be:
     - Width: `INPUT_WIDTH / STRIDE`
     - Height: `INPUT_HEIGHT / STRIDE` (which is always 1 in this case)
     - Channels: `INPUT_CHANNELS`
   - Use a sliding window approach to iterate over the input feature map and find the maximum value within each window.
   - Output the max-pooled feature map and assert the `data_out_valid` signal when the output is ready.
5. Timing and reset behavior:
   - The module should be synchronized with the `clk` signal.
   - The module should be reset when `rst_n` is asserted low.

Implement the `maxpool2d` module in Verilog, adhering to the specified input/output ports, parameters, and functionality. Ensure that the module is parameterized to allow flexibility in the input dimensions and pooling parameters.

The `maxpool2d` module will be instantiated within the `kws` top module, connecting the output of the second convolutional layer (`conv2_out`) to the input of the max pooling layer (`data_in`). The output of the max pooling layer (`data_out`) will be connected to the input of the first fully connected layer (`fc1`).

Make sure to handle the valid signals properly to maintain the correct flow of data through the pipeline.

Here's the implementation of the `maxpool2d` module in Verilog based on the given specifications:

verilog

Copy code

```verilog
`ifndef MAXPOOL2D_V
`define MAXPOOL2D_V

module maxpool2d #(
    parameter INPUT_WIDTH = 40,
    parameter INPUT_HEIGHT = 1,
    parameter INPUT_CHANNELS = 8,
    parameter KERNEL_SIZE = 2,
    parameter STRIDE = 2,
    parameter ACTIV_BITS = 8
) (
    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 reg [(INPUT_WIDTH/STRIDE)*INPUT_CHANNELS*ACTIV_BITS-1:0]
data_out,
    output reg data_out_valid
);

    localparam OUTPUT_WIDTH = INPUT_WIDTH / STRIDE;
    localparam OUTPUT_HEIGHT = INPUT_HEIGHT / STRIDE;

    // Declare 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];

    // Max pooling operation
    integer i, j, k, m, n;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset internal signals and output
            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
            data_out <= 0;
            data_out_valid <= 0;
        end else if (data_valid) begin
            // Shift input data into 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
                        if (j < INPUT_WIDTH - 1) begin
                            input_buffer[i][j][k] <=
input_buffer[i][j+1][k];
                        end else begin
                            input_buffer[i][j][k] <=
data_in[i*INPUT_WIDTH*INPUT_CHANNELS*ACTIV_BITS +
j*INPUT_CHANNELS*ACTIV_BITS + k*ACTIV_BITS +: ACTIV_BITS];
                        end
                    end
                end
            end
            // Perform max pooling
            for (i = 0; i < OUTPUT_HEIGHT; i = i + 1) begin
                for (j = 0; j < OUTPUT_WIDTH; 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*OUTPUT_WIDTH*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 begin
            data_out_valid <= 0;
        end
    end

endmodule
`endif
```

This `maxpool2d` module implements the max pooling operation with the specified parameters and functionality. Here's a breakdown of the module:

1. The module takes in the input parameters `INPUT_WIDTH`, `INPUT_HEIGHT`, `INPUT_CHANNELS`, `KERNEL_SIZE`, `STRIDE`, and `ACTIV_BITS`, which define the dimensions of the input feature map, the pooling kernel size, stride, and the number of activation bits.
2. The input ports include `clk`, `rst_n`, `data_in`, and `data_valid`. `data_in` is the input feature map with a width of `INPUT_WIDTH * INPUT_HEIGHT * INPUT_CHANNELS * ACTIV_BITS` bits, and `data_valid` is the valid signal for the input data.
3. The output ports include `data_out` and `data_out_valid`. `data_out` is the max-pooled output feature map with a width of `(INPUT_WIDTH / STRIDE) * INPUT_CHANNELS * ACTIV_BITS` bits, and `data_out_valid` is the valid signal for the output data.
4. The module uses an internal buffer `input_buffer` to store the input feature map and a register `max_value` to store the maximum value for each channel during the max pooling operation.
5. The max pooling operation is performed using nested loops to iterate over the input feature map with a sliding window approach. The maximum value within each window is found and stored in `max_value`.

6. The output feature map `data_out` is assigned the max-pooled values from `max_value`, and `data_out_valid` is asserted when the output is ready.
7. The module is synchronized with the `clk` signal and is reset when `rst_n` is asserted low.

This `maxpool2d` module can be instantiated within the `kws` top module, connecting the appropriate signals as described in the prompt.

Copy

Claude does not have the ability to run the code it generates yet.

EB

Based on the CNN-KWS model, also known as the "Hello Edge" model that was introduced in the paper "Hello Edge: Keyword Spotting on Microcontrollers" by Zhang et al. (2017) and the top module kws, what are the requirements for the fully_connected module. write a prompt for it.

Based on the CNN-KWS model architecture and the provided `kws` top module, the requirements for the `fully_connected` module can be derived. Here's a prompt for the `fully_connected` module:

Implement a parameterized `fully_connected` module with the following specifications:

1. Input parameters:
   - `INPUT_SIZE`: The size of the input feature vector.
     - For the first fully connected layer (fc1), `INPUT_SIZE` should be `(MFCC_FEATURES/2) * CONV2_NUM_FILTERS` (default: 160).
     - For the second fully connected layer (fc2), `INPUT_SIZE` should be `FC1_OUTPUT_SIZE` (default: 64).
   - `OUTPUT_SIZE`: The size of the output feature vector.
     - For the first fully connected layer (fc1), `OUTPUT_SIZE` should be `FC1_OUTPUT_SIZE` (default: 64).
     - For the second fully connected layer (fc2), `OUTPUT_SIZE` should be `FC2_OUTPUT_SIZE` (default: `NUM_KEYWORDS`).
   - `ACTIV_BITS`: The number of bits used for activations (default: 8).
2. Input ports:
   - `clk`: The clock signal.
   - `rst_n`: The active-low reset signal.
   - `data_in`: The input feature vector, with a width of `INPUT_SIZE * ACTIV_BITS` bits.
   - `data_valid`: The valid signal for the input data.
   - `weights_in`: The input weights, with a width of `OUTPUT_SIZE * INPUT_SIZE * ACTIV_BITS` bits.
   - `biases_in`: The input biases, with a width of `OUTPUT_SIZE * ACTIV_BITS` bits.
   - `load_weights`: The signal to load the input weights.
   - `load_biases`: The signal to load the input biases.
3. Output ports:

  - `data_out`: The output feature vector after the fully connected layer, with a width of `OUTPUT_SIZE * ACTIV_BITS` bits.
  - `data_out_valid`: The valid signal for the output data.
4. Functionality:
  - Perform matrix multiplication between the input feature vector and the weight matrix, and add the bias vector.
  - Apply the ReLU activation function to the result of the matrix multiplication and bias addition.
  - Output the activated feature vector and assert the `data_out_valid` signal when the output is ready.
  - Support loading of weights and biases when the respective `load_weights` and `load_biases` signals are asserted.
5. Timing and reset behavior:
  - The module should be synchronized with the `clk` signal.
  - The module should be reset when `rst_n` is asserted low.

Implement the `fully_connected` module in Verilog, adhering to the specified input/output ports, parameters, and functionality. Ensure that the module is parameterized to allow flexibility in the input and output sizes.

The `fully_connected` module will be instantiated twice within the `kws` top module:

- For the first fully connected layer (fc1), connecting the output of the max pooling layer (`maxpool_out`) to the input (`data_in`), and the output (`data_out`) to the input of the second fully connected layer.
- For the second fully connected layer (fc2), connecting the output of the first fully connected layer (`fc1_out`) to the input (`data_in`), and the output (`data_out`) to the input of the softmax layer.

Make sure to handle the valid signals properly to maintain the correct flow of data through the pipeline.

Copy

EB

Implement a parameterized `fully_connected` module with the following specifications:

1. Input parameters:
  - `INPUT_SIZE`: The size of the input feature vector.
    - For the first fully connected layer (fc1), `INPUT_SIZE` should be `(MFCC_FEATURES/2) * CONV2_NUM_FILTERS` (default: 160).
    - For the second fully connected layer (fc2), `INPUT_SIZE` should be `FC1_OUTPUT_SIZE` (default: 64).
  - `OUTPUT_SIZE`: The size of the output feature vector.
    - For the first fully connected layer (fc1), `OUTPUT_SIZE` should be `FC1_OUTPUT_SIZE` (default: 64).
    - For the second fully connected layer (fc2), `OUTPUT_SIZE` should be `FC2_OUTPUT_SIZE` (default: `NUM_KEYWORDS`).
  - `ACTIV_BITS`: The number of bits used for activations (default: 8).

2. Input ports:
    - `clk`: The clock signal.
    - `rst_n`: The active-low reset signal.
    - `data_in`: The input feature vector, with a width of `INPUT_SIZE * ACTIV_BITS` bits.
    - `data_valid`: The valid signal for the input data.
    - `weights_in`: The input weights, with a width of `OUTPUT_SIZE * INPUT_SIZE * ACTIV_BITS` bits.
    - `biases_in`: The input biases, with a width of `OUTPUT_SIZE * ACTIV_BITS` bits.
    - `load_weights`: The signal to load the input weights.
    - `load_biases`: The signal to load the input biases.
3. Output ports:
    - `data_out`: The output feature vector after the fully connected layer, with a width of `OUTPUT_SIZE * ACTIV_BITS` bits.
    - `data_out_valid`: The valid signal for the output data.
4. Functionality:
    - Perform matrix multiplication between the input feature vector and the weight matrix, and add the bias vector.
    - Apply the ReLU activation function to the result of the matrix multiplication and bias addition.
    - Output the activated feature vector and assert the `data_out_valid` signal when the output is ready.
    - Support loading of weights and biases when the respective `load_weights` and `load_biases` signals are asserted.
5. Timing and reset behavior:
    - The module should be synchronized with the `clk` signal.
    - The module should be reset when `rst_n` is asserted low.

Implement the `fully_connected` module in Verilog, adhering to the specified input/output ports, parameters, and functionality. Ensure that the module is parameterized to allow flexibility in the input and output sizes.

The `fully_connected` module will be instantiated twice within the `kws` top module:

- For the first fully connected layer (fc1), connecting the output of the max pooling layer (`maxpool_out`) to the input (`data_in`), and the output (`data_out`) to the input of the second fully connected layer.
- For the second fully connected layer (fc2), connecting the output of the first fully connected layer (`fc1_out`) to the input (`data_in`), and the output (`data_out`) to the input of the softmax layer.

Make sure to handle the valid signals properly to maintain the correct flow of data through the pipeline.

Here's the implementation of the `fully_connected` module in Verilog based on the given specifications:

```
verilog
```

Copy code

```
`ifndef FULLY_CONNECTED_V
`define FULLY_CONNECTED_V

module fully_connected #(
    parameter INPUT_SIZE = 160,
    parameter OUTPUT_SIZE = 64,
    parameter ACTIV_BITS = 8
) (
    input wire clk,
    input wire rst_n,
    input wire [INPUT_SIZE*ACTIV_BITS-1:0] data_in,
    input wire data_valid,
    output reg [OUTPUT_SIZE*ACTIV_BITS-1:0] data_out,
    output reg data_out_valid,
    input wire [OUTPUT_SIZE*INPUT_SIZE*ACTIV_BITS-1:0] weights_in,
    input wire [OUTPUT_SIZE*ACTIV_BITS-1:0] biases_in,
    input wire load_weights,
    input wire load_biases
);

    // Declare weights and biases
    reg [ACTIV_BITS-1:0] weights [0:OUTPUT_SIZE-1][0:INPUT_SIZE-1];
    reg [ACTIV_BITS-1:0] biases [0:OUTPUT_SIZE-1];

    // Declare internal signals
    reg [2*ACTIV_BITS-1:0] acc_result [0:OUTPUT_SIZE-1];
    reg [ACTIV_BITS-1:0] relu_result [0:OUTPUT_SIZE-1];

    // Load weights and biases
    integer i, j;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset weights and biases
            for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin
                for (j = 0; j < INPUT_SIZE; j = j + 1) begin
                    weights[i][j] = 0;
                end
                biases[i] <= 0;
            end
        end else begin
            // Load weights when load_weights is asserted
            if (load_weights) begin
                for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin
                    for (j = 0; j < INPUT_SIZE; j = j + 1) begin
                        weights[i][j] = weights_in[(i*INPUT_SIZE +
j)*ACTIV_BITS +: ACTIV_BITS];
                    end
                end
            end
            // Load biases when load_biases is asserted
            if (load_biases) begin
                for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin
```

```
                    biases[i] <= biases_in[i*ACTIV_BITS +: ACTIV_BITS];
                end
            end
        end
    end

    // Fully connected layer operation
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset internal signals and output
            for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin
                acc_result[i] <= 0;
                relu_result[i] <= 0;
            end
            data_out <= 0;
            data_out_valid <= 0;
        end else if (data_valid) begin
            // Perform matrix multiplication
            for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin
                acc_result[i] = {{(2*ACTIV_BITS-ACTIV_BITS){1'b0}},
biases[i]};
                for (j = 0; j < INPUT_SIZE; j = j + 1) begin
                    acc_result[i] = acc_result[i] + weights[i][j] *
data_in[j*ACTIV_BITS +: ACTIV_BITS];
                end
            end

            // Apply ReLU activation
            for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin
                relu_result[i] <= (acc_result[i][2*ACTIV_BITS-1] == 0) ?
acc_result[i][ACTIV_BITS-1:0] : 0;
            end

            // Assign output
            for (i = 0; i < OUTPUT_SIZE; i = i + 1) begin
                data_out[i*ACTIV_BITS +: ACTIV_BITS] <= relu_result[i];
            end
            data_out_valid <= 1;
        end else begin
            data_out_valid <= 0;
        end
    end

endmodule
`endif
```

This `fully_connected` module implements a fully connected layer with the specified parameters and functionality. Here's a breakdown of the module:

1. The module takes in the input parameters `INPUT_SIZE`, `OUTPUT_SIZE`, and `ACTIV_BITS`, which define the sizes of the input and output feature vectors and the number of activation bits.

2. The input ports include `clk`, `rst_n`, `data_in`, `data_valid`, `weights_in`, `biases_in`, `load_weights`, and `load_biases`. `data_in` is the input feature vector, `weights_in` and `biases_in` are the input weights and biases, and `load_weights` and `load_biases` are the signals to load the weights and biases, respectively.
3. The output ports include `data_out` and `data_out_valid`. `data_out` is the output feature vector after the fully connected layer, and `data_out_valid` is the valid signal for the output data.
4. The module uses internal registers `weights` and `biases` to store the weights and biases, respectively. It also uses internal registers `acc_result` and `relu_result` to store the intermediate results during the computation.
5. The weights and biases are loaded when the respective `load_weights` and `load_biases` signals are asserted.
6. The fully connected layer operation is performed in three steps:
    - Matrix multiplication between the input feature vector and the weight matrix, and addition of the bias vector.
    - Application of the ReLU activation function to the result of the matrix multiplication and bias addition.
    - Assignment of the activated output to `data_out`.
7. The module is synchronized with the `clk` signal and is reset when `rst_n` is asserted low.

This `fully_connected` module can be instantiated twice within the `kws` top module, as described in the prompt, to implement the two fully connected layers (fc1 and fc2) in the CNN-KWS model.

Copy

Claude does not have the ability to run the code it generates yet.

EB

Based on the CNN-KWS model, also known as the "Hello Edge" model that was introduced in the paper "Hello Edge: Keyword Spotting on Microcontrollers" by Zhang et al. (2017) and the top module kws, what are the requirements for the conv2d module. write a prompt for it.

Based on the CNN-KWS model architecture and the provided `kws` top module, the requirements for the `conv2d` module can be derived. Here's a prompt for the `conv2d` module:

Implement a parameterized `conv2d` module with the following specifications:

1. Input parameters:
    - `INPUT_WIDTH`: The width of the input feature map (default: `MFCC_FEATURES`, which is 40).
    - `INPUT_HEIGHT`: The height of the input feature map (default: 1, since the feature map is 1-dimensional).
    - `INPUT_CHANNELS`: The number of input channels.
        - For the first convolutional layer (conv1), `INPUT_CHANNELS` should be 1.
        - For the second convolutional layer (conv2), `INPUT_CHANNELS` should be `CONV1_NUM_FILTERS` (default: 8).

   - `KERNEL_SIZE`: The size of the convolutional kernel.
      - For both convolutional layers (conv1 and conv2), `KERNEL_SIZE` should be 3.
   - `NUM_FILTERS`: The number of output filters.
      - For the first convolutional layer (conv1), `NUM_FILTERS` should be `CONV1_NUM_FILTERS` (default: 8).
      - For the second convolutional layer (conv2), `NUM_FILTERS` should be `CONV2_NUM_FILTERS` (default: 8).
   - `PADDING`: The amount of padding applied to the input feature map (default: 1).
   - `ACTIV_BITS`: The number of bits used for activations (default: 8).
2. Input ports:
   - `clk`: The clock signal.
   - `rst_n`: The active-low reset signal.
   - `data_in`: The input feature map, with a width of `INPUT_WIDTH * INPUT_HEIGHT * INPUT_CHANNELS * ACTIV_BITS` bits.
   - `data_valid`: The valid signal for the input data.
   - `weights_in`: The input weights, with a width of `NUM_FILTERS * INPUT_CHANNELS * KERNEL_SIZE * KERNEL_SIZE * ACTIV_BITS` bits.
   - `biases_in`: The input biases, with a width of `NUM_FILTERS * ACTIV_BITS` bits.
   - `load_weights`: The signal to load the input weights.
   - `load_biases`: The signal to load the input biases.
3. Output ports:
   - `data_out`: The output feature map after the convolutional layer, with a width of `INPUT_WIDTH * INPUT_HEIGHT * NUM_FILTERS * ACTIV_BITS` bits.
   - `data_out_valid`: The valid signal for the output data.
4. Functionality:
   - Perform the convolutional operation between the input feature map and the kernel weights, and add the bias values.
   - Apply the ReLU activation function to the result of the convolution and bias addition.
   - Pad the input feature map with zeros based on the specified `PADDING` value to maintain the spatial dimensions.
   - Output the activated feature map and assert the `data_out_valid` signal when the output is ready.
   - Support loading of weights and biases when the respective `load_weights` and `load_biases` signals are asserted.
5. Timing and reset behavior:
   - The module should be synchronized with the `clk` signal.
   - The module should be reset when `rst_n` is asserted low.

Implement the `conv2d` module in Verilog, adhering to the specified input/output ports, parameters, and functionality. Ensure that the module is parameterized to allow flexibility in the input dimensions, number of filters, kernel size, padding, and activation bits.

The `conv2d` module will be instantiated twice within the `kws` top module:

- For the first convolutional layer (conv1), connecting the output of the MFCC feature extraction (`mfcc_out`) to the input (`data_in`), and the output (`data_out`) to the input of the second convolutional layer.
- For the second convolutional layer (conv2), connecting the output of the first convolutional layer (`conv1_out`) to the input (`data_in`), and the output (`data_out`) to the input of the max pooling layer.

Make sure to handle the valid signals properly to maintain the correct flow of data through the pipeline.

Copy

EB

Implement a parameterized `conv2d` module with the following specifications:

1. Input parameters:
   - `INPUT_WIDTH`: The width of the input feature map (default: `MFCC_FEATURES`, which is 40).
   - `INPUT_HEIGHT`: The height of the input feature map (default: 1, since the feature map is 1-dimensional).
   - `INPUT_CHANNELS`: The number of input channels.
     - For the first convolutional layer (conv1), `INPUT_CHANNELS` should be 1.
     - For the second convolutional layer (conv2), `INPUT_CHANNELS` should be `CONV1_NUM_FILTERS` (default: 8).
   - `KERNEL_SIZE`: The size of the convolutional kernel.
     - For both convolutional layers (conv1 and conv2), `KERNEL_SIZE` should be 3.
   - `NUM_FILTERS`: The number of output filters.
     - For the first convolutional layer (conv1), `NUM_FILTERS` should be `CONV1_NUM_FILTERS` (default: 8).
     - For the second convolutional layer (conv2), `NUM_FILTERS` should be `CONV2_NUM_FILTERS` (default: 8).
   - `PADDING`: The amount of padding applied to the input feature map (default: 1).
   - `ACTIV_BITS`: The number of bits used for activations (default: 8).
2. Input ports:
   - `clk`: The clock signal.
   - `rst_n`: The active-low reset signal.
   - `data_in`: The input feature map, with a width of `INPUT_WIDTH * INPUT_HEIGHT * INPUT_CHANNELS * ACTIV_BITS` bits.
   - `data_valid`: The valid signal for the input data.
   - `weights_in`: The input weights, with a width of `NUM_FILTERS * INPUT_CHANNELS * KERNEL_SIZE * KERNEL_SIZE * ACTIV_BITS` bits.
   - `biases_in`: The input biases, with a width of `NUM_FILTERS * ACTIV_BITS` bits.
   - `load_weights`: The signal to load the input weights.
   - `load_biases`: The signal to load the input biases.
3. Output ports:

    - `data_out`: The output feature map after the convolutional layer, with a width of `INPUT_WIDTH * INPUT_HEIGHT * NUM_FILTERS * ACTIV_BITS` bits.
    - `data_out_valid`: The valid signal for the output data.
4. Functionality:
    - Perform the convolutional operation between the input feature map and the kernel weights, and add the bias values.
    - Apply the ReLU activation function to the result of the convolution and bias addition.
    - Pad the input feature map with zeros based on the specified `PADDING` value to maintain the spatial dimensions.
    - Output the activated feature map and assert the `data_out_valid` signal when the output is ready.
    - Support loading of weights and biases when the respective `load_weights` and `load_biases` signals are asserted.
5. Timing and reset behavior:
    - The module should be synchronized with the `clk` signal.
    - The module should be reset when `rst_n` is asserted low.

Implement the `conv2d` module in Verilog, adhering to the specified input/output ports, parameters, and functionality. Ensure that the module is parameterized to allow flexibility in the input dimensions, number of filters, kernel size, padding, and activation bits.

The `conv2d` module will be instantiated twice within the `kws` top module:

- For the first convolutional layer (conv1), connecting the output of the MFCC feature extraction (`mfcc_out`) to the input (`data_in`), and the output (`data_out`) to the input of the second convolutional layer.
- For the second convolutional layer (conv2), connecting the output of the first convolutional layer (`conv1_out`) to the input (`data_in`), and the output (`data_out`) to the input of the max pooling layer.

Make sure to handle the valid signals properly to maintain the correct flow of data through the pipeline.

Here's the implementation of the `conv2d` module in Verilog based on the given specifications:

verilog

Copy code

```verilog
`ifndef CONV2D_V
`define CONV2D_V

module conv2d #(
    parameter INPUT_WIDTH = 40,
    parameter INPUT_HEIGHT = 1,
    parameter INPUT_CHANNELS = 1,
    parameter KERNEL_SIZE = 3,
    parameter NUM_FILTERS = 8,
    parameter PADDING = 1,
    parameter ACTIV_BITS = 8
) (
    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 reg [INPUT_WIDTH*INPUT_HEIGHT*NUM_FILTERS*ACTIV_BITS-1:0]
data_out,
    output reg data_out_valid,
    input wire
[NUM_FILTERS*INPUT_CHANNELS*KERNEL_SIZE*KERNEL_SIZE*ACTIV_BITS-1:0]
weights_in,
    input wire [NUM_FILTERS*ACTIV_BITS-1:0] biases_in,
    input wire load_weights,
    input wire load_biases
);

    // Declare weights and biases
    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];

    // Declare internal signals
    reg [ACTIV_BITS-1:0] input_buffer [0:INPUT_HEIGHT-1][0:INPUT_WIDTH-
1][0:INPUT_CHANNELS-1];
    reg [2*ACTIV_BITS-1:0] conv_result [0:INPUT_HEIGHT-1][0:INPUT_WIDTH-
1][0:NUM_FILTERS-1];
    reg [ACTIV_BITS-1:0] relu_result [0:INPUT_HEIGHT-1][0:INPUT_WIDTH-
1][0:NUM_FILTERS-1];

    // Load weights and biases
    integer i, j, k, l;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset weights and biases
            for (i = 0; i < NUM_FILTERS; i = i + 1) begin
                for (j = 0; j < INPUT_CHANNELS; j = j + 1) begin
                    for (k = 0; k < KERNEL_SIZE; k = k + 1) begin
                        for (l = 0; l < KERNEL_SIZE; l = l + 1) begin
                            weights[i][j][k][l] <= 0;
                        end
                    end
                end
                biases[i] <= 0;
            end
        end else begin
            // Load weights when load_weights is asserted
            if (load_weights) begin
                for (i = 0; i < NUM_FILTERS; i = i + 1) begin
                    for (j = 0; j < INPUT_CHANNELS; j = j + 1) begin
                        for (k = 0; k < KERNEL_SIZE; k = k + 1) begin
                            for (l = 0; l < KERNEL_SIZE; l = l + 1) begin
```

```
                                weights[i][j][k][l] <=
weights_in[(i*INPUT_CHANNELS*KERNEL_SIZE*KERNEL_SIZE +
j*KERNEL_SIZE*KERNEL_SIZE + k*KERNEL_SIZE + l)*ACTIV_BITS +: ACTIV_BITS];
                            end
                        end
                    end
                end
            end
            // Load biases when load_biases is asserted
            if (load_biases) begin
                for (i = 0; i < NUM_FILTERS; i = i + 1) begin
                    biases[i] <= biases_in[i*ACTIV_BITS +: ACTIV_BITS];
                end
            end
        end
    end

    // Convolution operation
    integer m, n, p, q;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset internal signals and output
            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
                    for (m = 0; m < NUM_FILTERS; m = m + 1) begin
                        conv_result[i][j][m] <= 0;
                        relu_result[i][j][m] <= 0;
                    end
                end
            end
            data_out <= 0;
            data_out_valid <= 0;
        end else if (data_valid) begin
            // Shift input data into 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
                        if (j < INPUT_WIDTH - 1) begin
                            input_buffer[i][j][k] <=
input_buffer[i][j+1][k];
                        end else begin
                            input_buffer[i][j][k] <=
data_in[i*INPUT_WIDTH*INPUT_CHANNELS*ACTIV_BITS +
j*INPUT_CHANNELS*ACTIV_BITS + k*ACTIV_BITS +: ACTIV_BITS];
                        end
                    end
                end
            end

            // Perform convolution
            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] = {{(2*ACTIV_BITS-
ACTIV_BITS){1'b0}}, 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 - PADDING >= 0 && m + i -
PADDING < INPUT_HEIGHT &&
                                        n + j - PADDING >= 0 && n + j -
PADDING < INPUT_WIDTH) begin
                                        conv_result[m][n][p] =
conv_result[m][n][p] + weights[p][q][i][j] * input_buffer[m + i -
PADDING][n + j - PADDING][q];
                                    end
                                end
                            end
                        end
                    end
                end
            end

            // Apply ReLU activation
            for (i = 0; i < INPUT_HEIGHT; i = i + 1) begin
                for (j = 0; j < INPUT_WIDTH; j = j + 1) begin
                    for (k = 0; k < NUM_FILTERS; k = k + 1) begin
                        relu_result[i][j][k] =
(conv_result[i][j][k][2*ACTIV_BITS-1] == 0) ?
conv_result[i][j][k][ACTIV_BITS-1:0] : 0;
                    end
                end
            end

            // Assign output
            for (i = 0; i < INPUT_HEIGHT; i = i + 1) begin
                for (j = 0; j < INPUT_WIDTH; j = j + 1) begin
                    for (k = 0; k < NUM_FILTERS; k = k + 1) begin
                        data_out[i*INPUT_WIDTH*NUM_FILTERS*ACTIV_BITS +
j*NUM_FILTERS*ACTIV_BITS + k*ACTIV_BITS +: ACTIV_BITS] <=
relu_result[i][j][k];
                    end
                end
            end
            data_out_valid <= 1;
        end else begin
            data_out_valid <= 0;
        end
    end

endmodule
`endif
```

This `conv2d` module implements a convolutional layer with the specified parameters and functionality. Here's a breakdown of the module:

1. The module takes in the input parameters `INPUT_WIDTH`, `INPUT_HEIGHT`, `INPUT_CHANNELS`, `KERNEL_SIZE`, `NUM_FILTERS`, `PADDING`, and `ACTIV_BITS`, which define the dimensions of the input feature map, the number of input channels, the size of the convolutional kernel, the number of output filters, the amount of padding, and the number of activation bits.
2. The input ports include `clk`, `rst_n`, `data_in`, `data_valid`, `weights_in`, `biases_in`, `load_weights`, and `load_biases`. `data_in` is the input feature map, `weights_in` and `biases_in` are the input weights and biases, and `load_weights` and `load_biases` are the signals to load the weights and biases, respectively.
3. The output ports include `data_out` and `data_out_valid`. `data_out` is the output feature map after the convolutional layer, and `data_out_valid` is the valid signal for the output data.
4. The module uses internal registers `weights` and `biases` to store the weights and biases, respectively. It also uses internal registers `input_buffer`, `conv_result`, and `relu_result` to store the intermediate results during the computation.
5. The weights and biases are loaded when the respective `load_weights` and `load_biases` signals are asserted.
6. The convolutional operation is performed in four steps:
   - Shifting the input data into the input buffer.
   - Performing the convolution between the input feature map and the kernel weights, and adding the bias values.
   - Applying the ReLU activation function to the result of the convolution and bias addition.
   - Assigning the activated output to `data_out`.
7. The module is synchronized with the `clk` signal and is reset when `rst_n` is asserted low.

This `conv2d` module can be instantiated twice within the `kws` top module, as described in the prompt, to implement the two convolutional layers (conv1 and conv2) in the CNN-KWS model.

Copy

Claude does not have the ability to run the code it generates yet.

EB

The synthesis tools report that the "i, j, k, l" integers from the conv2d and the "i,j" integers from the fully_connected have multiple conflicting drivers. Can you recommend a way to fix them?

To resolve the issue of multiple conflicting drivers for the integer variables "i, j, k, l" in the `conv2d` module and "i, j" in the `fully_connected` module, you can declare separate integer variables for each always block or generate block where they are used. This way, each integer variable will have a unique driver.

Here's an example of how you can modify the code to fix the issue:

For the `conv2d` module:

verilog

Copy code

```verilog
// ...

// Load weights and biases
integer i_load, j_load, k_load, l_load;
always @(posedge clk or negedge rst_n) begin
    // ...
    for (i_load = 0; i_load < NUM_FILTERS; i_load = i_load + 1) begin
        for (j_load = 0; j_load < INPUT_CHANNELS; j_load = j_load + 1) begin
            for (k_load = 0; k_load < KERNEL_SIZE; k_load = k_load + 1) begin
                for (l_load = 0; l_load < KERNEL_SIZE; l_load = l_load + 1) begin
                    // ...
                end
            end
        end
    end
    // ...
end

// Convolution operation
integer m_conv, n_conv, p_conv, q_conv;
always @(posedge clk or negedge rst_n) begin
    // ...
    for (m_conv = 0; m_conv < INPUT_HEIGHT; m_conv = m_conv + 1) begin
        for (n_conv = 0; n_conv < INPUT_WIDTH; n_conv = n_conv + 1) begin
            for (p_conv = 0; p_conv < NUM_FILTERS; p_conv = p_conv + 1) begin
                // ...
                for (q_conv = 0; q_conv < INPUT_CHANNELS; q_conv = q_conv + 1) begin
                    for (i_conv = 0; i_conv < KERNEL_SIZE; i_conv = i_conv + 1) begin
                        for (j_conv = 0; j_conv < KERNEL_SIZE; j_conv = j_conv + 1) begin
                            // ...
                        end
                    end
                end
            end
        end
    end
    // ...
end

// ...
```

For the `fully_connected` module:

verilog

Copy code

```verilog
// ...

// Load weights and biases
integer i_load, j_load;
always @(posedge clk or negedge rst_n) begin
    // ...
    for (i_load = 0; i_load < OUTPUT_SIZE; i_load = i_load + 1) begin
        for (j_load = 0; j_load < INPUT_SIZE; j_load = j_load + 1) begin
            // ...
        end
    end
    // ...
end

// Fully connected layer operation
integer i_fc, j_fc;
always @(posedge clk or negedge rst_n) begin
    // ...
    for (i_fc = 0; i_fc < OUTPUT_SIZE; i_fc = i_fc + 1) begin
        // ...
        for (j_fc = 0; j_fc < INPUT_SIZE; j_fc = j_fc + 1) begin
            // ...
        end
    end
    // ...
end

// ...
```

By using separate integer variables for each always block or generate block, you avoid the issue of multiple conflicting drivers. Make sure to use the appropriate integer variable names in the corresponding loops and assignments.

This approach should resolve the synthesis tool's report of multiple conflicting drivers for the integer variables.

Copy

Claude does not have the ability to run the code it generates yet.

EB

The contest requires that all code in implemented by the llm. Can you provide the refactored modules?

Certainly! Here's the refactored code for the `conv2d` and `fully_connected` modules with separate integer variables for each always block or generate block to avoid multiple conflicting drivers:

Refactored `conv2d` module:

verilog

Copy code

```verilog
`ifndef CONV2D_V
`define CONV2D_V

module conv2d #(
    parameter INPUT_WIDTH = 40,
    parameter INPUT_HEIGHT = 1,
    parameter INPUT_CHANNELS = 1,
    parameter KERNEL_SIZE = 3,
    parameter NUM_FILTERS = 8,
    parameter PADDING = 1,
    parameter ACTIV_BITS = 8
) (
    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 reg [INPUT_WIDTH*INPUT_HEIGHT*NUM_FILTERS*ACTIV_BITS-1:0] data_out,
    output reg data_out_valid,
    input wire [NUM_FILTERS*INPUT_CHANNELS*KERNEL_SIZE*KERNEL_SIZE*ACTIV_BITS-1:0] weights_in,
    input wire [NUM_FILTERS*ACTIV_BITS-1:0] biases_in,
    input wire load_weights,
    input wire load_biases
);

    // Declare weights and biases
    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];

    // Declare internal signals
    reg [ACTIV_BITS-1:0] input_buffer [0:INPUT_HEIGHT-1][0:INPUT_WIDTH-1][0:INPUT_CHANNELS-1];
    reg [2*ACTIV_BITS-1:0] conv_result [0:INPUT_HEIGHT-1][0:INPUT_WIDTH-1][0:NUM_FILTERS-1];
    reg [ACTIV_BITS-1:0] relu_result [0:INPUT_HEIGHT-1][0:INPUT_WIDTH-1][0:NUM_FILTERS-1];

    // Load weights and biases
    integer i_load, j_load, k_load, l_load;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset weights and biases
            for (i_load = 0; i_load < NUM_FILTERS; i_load = i_load + 1) begin
                for (j_load = 0; j_load < INPUT_CHANNELS; j_load = j_load + 1) begin
                    for (k_load = 0; k_load < KERNEL_SIZE; k_load = k_load + 1) begin
                        for (l_load = 0; l_load < KERNEL_SIZE; l_load = l_load + 1) begin
                            weights[i_load][j_load][k_load][l_load] <= 0;
                        end
                    end
                end
                biases[i_load] <= 0;
            end
        end else begin
            // Load weights when load_weights is asserted
            if (load_weights) begin
                for (i_load = 0; i_load < NUM_FILTERS; i_load = i_load + 1) begin
                    for (j_load = 0; j_load < INPUT_CHANNELS; j_load = j_load + 1) begin
                        for (k_load = 0; k_load < KERNEL_SIZE; k_load = k_load + 1) begin
                            for (l_load = 0; l_load < KERNEL_SIZE; l_load = l_load + 1) begin
                                weights[i_load][j_load][k_load][l_load] <= weights_in[(i_load*INPUT_CHANNELS*KERNEL_SIZE*KERNEL_SIZE + j_load*KERNEL_SIZE*KERNEL_SIZE + k_load*KERNEL_SIZE + l_load)*ACTIV_BITS +: ACTIV_BITS];
                            end
                        end
                    end
                end
            end
            // Load biases when load_biases is asserted
            if (load_biases) begin
                for (i_load = 0; i_load < NUM_FILTERS; i_load = i_load + 1) begin
                    biases[i_load] <= biases_in[i_load*ACTIV_BITS +: ACTIV_BITS];
                end
            end
        end
    end

    // Convolution operation
    integer m_conv, n_conv, p_conv, q_conv, i_conv, j_conv;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset internal signals and output
            for (i_conv = 0; i_conv < INPUT_HEIGHT; i_conv = i_conv + 1)
begin
                for (j_conv = 0; j_conv < INPUT_WIDTH; j_conv = j_conv + 1)
begin
                    for (k_conv = 0; k_conv < INPUT_CHANNELS; k_conv =
k_conv + 1) begin
                        input_buffer[i_conv][j_conv][k_conv] <= 0;
                    end
                    for (m_conv = 0; m_conv < NUM_FILTERS; m_conv = m_conv
+ 1) begin
                        conv_result[i_conv][j_conv][m_conv] <= 0;
                        relu_result[i_conv][j_conv][m_conv] <= 0;
                    end
                end
            end
            data_out <= 0;
            data_out_valid <= 0;
        end else if (data_valid) begin
            // Shift input data into buffer
            for (i_conv = 0; i_conv < INPUT_HEIGHT; i_conv = i_conv + 1)
begin
                for (j_conv = 0; j_conv < INPUT_WIDTH; j_conv = j_conv + 1)
begin
                    for (k_conv = 0; k_conv < INPUT_CHANNELS; k_conv =
k_conv + 1) begin
                        if (j_conv < INPUT_WIDTH - 1) begin
                            input_buffer[i_conv][j_conv][k_conv] <=
input_buffer[i_conv][j_conv+1][k_conv];
                        end else begin
                            input_buffer[i_conv][j_conv][k_conv] <=
data_in[i_conv*INPUT_WIDTH*INPUT_CHANNELS*ACTIV_BITS +
j_conv*INPUT_CHANNELS*ACTIV_BITS + k_conv*ACTIV_BITS +: ACTIV_BITS];
                        end
                    end
                end
            end

            // Perform convolution
            for (m_conv = 0; m_conv < INPUT_HEIGHT; m_conv = m_conv + 1)
begin
                for (n_conv = 0; n_conv < INPUT_WIDTH; n_conv = n_conv + 1)
begin
                    for (p_conv = 0; p_conv < NUM_FILTERS; p_conv = p_conv
+ 1) begin
                        conv_result[m_conv][n_conv][p_conv] =
{{(2*ACTIV_BITS-ACTIV_BITS){1'b0}}, biases[p_conv]};
                        for (q_conv = 0; q_conv < INPUT_CHANNELS; q_conv =
q_conv + 1) begin
                            for (i_conv = 0; i_conv < KERNEL_SIZE; i_conv =
i_conv + 1) begin
                                for (j_conv = 0; j_conv < KERNEL_SIZE;
j_conv = j_conv + 1) begin
```

```
                                    if (m_conv + i_conv - PADDING >= 0 &&
m_conv + i_conv - PADDING < INPUT_HEIGHT &&
                                        n_conv + j_conv - PADDING >= 0 &&
n_conv + j_conv - PADDING < INPUT_WIDTH) begin
                                        conv_result[m_conv][n_conv][p_conv]
= conv_result[m_conv][n_conv][p_conv] +
weights[p_conv][q_conv][i_conv][j_conv] * input_buffer[m_conv + i_conv -
PADDING][n_conv + j_conv - PADDING][q_conv];
                                    end
                                end
                            end
                        end
                    end
                end
            end

            // Apply ReLU activation
            for (i_conv = 0; i_conv < INPUT_HEIGHT; i_conv = i_conv + 1)
begin
                for (j_conv = 0; j_conv < INPUT_WIDTH; j_conv = j_conv + 1)
begin
                    for (k_conv = 0; k_conv < NUM_FILTERS; k_conv = k_conv
+ 1) begin
                        relu_result[i_conv][j_conv][k_conv] =
(conv_result[i_conv][j_conv][k_conv][2*ACTIV_BITS-1] == 0) ?
conv_result[i_conv][j_conv][k_conv][ACTIV_BITS-1:0] : 0;
                    end
                end
            end

            // Assign output
            for (i_conv = 0; i_conv < INPUT_HEIGHT; i_conv = i_conv + 1)
begin
                for (j_conv = 0; j_conv < INPUT_WIDTH; j_conv = j_conv + 1)
begin
                    for (k_conv = 0; k_conv < NUM_FILTERS; k_conv = k_conv
+ 1) begin
                        data_out[i_conv*INPUT_WIDTH*NUM_FILTERS*ACTIV_BITS
+ j_conv*NUM_FILTERS*ACTIV_BITS + k_conv*ACTIV_BITS +: ACTIV_BITS] <=
relu_result[i_conv][j_conv][k_conv];
                    end
                end
            end
            data_out_valid <= 1;
        end else begin
            data_out_valid <= 0;
        end
    end

endmodule
`endif
```

Refactored `fully_connected` module:

verilog

Copy code

```
`ifndef FULLY_CONNECTED_V
`define FULLY_CONNECTED_V

module fully_connected #(
    parameter INPUT_SIZE = 160,
    parameter OUTPUT_SIZE = 64,
    parameter ACTIV_BITS = 8
) (
    input wire clk,
    input wire rst_n,
    input wire [INPUT_SIZE*ACTIV_BITS-1:0] data_in,
    input wire data_valid,
    output reg [OUTPUT_SIZE*ACTIV_BITS-1:0] data_out,
    output reg data_out_valid,
    input wire [OUTPUT_SIZE*INPUT_SIZE*ACTIV_BITS-1:0] weights_in,
    input wire [OUTPUT_SIZE*ACTIV_BITS-1:0] biases_in,
    input wire load_weights,
    input wire load_biases
);

    // Declare weights and biases
    reg [ACTIV_BITS-1:0] weights [0:OUTPUT_SIZE-1][0:INPUT_SIZE-1];
    reg [ACTIV_BITS-1:0] biases [0:OUTPUT_SIZE-1];

    // Declare internal signals
    reg [2*ACTIV_BITS-1:0] acc_result [0:OUTPUT_SIZE-1];
    reg [ACTIV_BITS-1:0] relu_result [0:OUTPUT_SIZE-1];

    // Load weights and biases
    integer i_load, j_load;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset weights and biases
            for (i_load = 0; i_load < OUTPUT_SIZE; i_load = i_load + 1)
begin
                for (j_load = 0; j_load < INPUT_SIZE; j_load = j_load + 1)
begin
                    weights[i_load][j_load] <= 0;
                end
                biases[i_load] <= 0;
            end
        end else begin
            // Load weights when load_weights is asserted
            if (load_weights) begin
                for (i_load = 0; i_load < OUTPUT_SIZE; i_load = i_load + 1)
begin
                    for (j_load = 0; j_load < INPUT_SIZE; j_load = j_load +
1) begin
                        weights[i_load][j_load] <=
weights_in[(i_load*INPUT_SIZE + j_load)*ACTIV_BITS +: ACTIV_BITS];
```

```
                    end
                end
            end
            // Load biases when load_biases is asserted
            if (load_biases) begin
                for (i_load = 0; i_load < OUTPUT_SIZE; i_load = i_load + 1)
begin
                    biases[i_load] <= biases_in[i_load*ACTIV_BITS +:
ACTIV_BITS];
                end
            end
        end
    end

    // Fully connected layer operation
    integer i_fc, j_fc;
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset internal signals and output
            for (i_fc = 0; i_fc < OUTPUT_SIZE; i_fc = i_fc + 1) begin
                acc_result[i_fc] <= 0;
                relu_result[i_fc] <= 0;
            end
            data_out <= 0;
            data_out_valid <= 0;
        end else if (data_valid) begin
            // Perform matrix multiplication
            for (i_fc = 0; i_fc < OUTPUT_SIZE; i_fc = i_fc + 1) begin
                acc_result[i_fc] = {{(2*ACTIV_BITS-ACTIV_BITS){1'b0}},
biases[i_fc]};
                for (j_fc = 0; j_fc < INPUT_SIZE; j_fc = j_fc + 1) begin
                    acc_result[i_fc] = acc_result[i_fc] +
weights[i_fc][j_fc] * data_in[j_fc*ACTIV_BITS +: ACTIV_BITS];
                end
            end

            // Apply ReLU activation
            for (i_fc = 0; i_fc < OUTPUT_SIZE; i_fc = i_fc + 1) begin
                relu_result[i_fc] <= (acc_result[i_fc][2*ACTIV_BITS-1] ==
0) ? acc_result[i_fc][ACTIV_BITS-1:0] : 0;
            end

            // Assign output
            for (i_fc = 0; i_fc < OUTPUT_SIZE; i_fc = i_fc + 1) begin
                data_out[i_fc*ACTIV_BITS +: ACTIV_BITS] <=
relu_result[i_fc];
            end
            data_out_valid <= 1;
        end else begin
            data_out_valid <= 0;
        end
    end
```

```
endmodule
`endif
```

In the refactored code, separate integer variables are used for each always block or generate block to avoid multiple conflicting drivers. The integer variables are named with suffixes like `_load`, `_conv`, and `_fc` to distinguish their usage in different parts of the module.

Please note that the refactored code assumes that the inputs, outputs, and parameters are correctly connected and instantiated in the top-level module (`kws`).