**NPU**

module top\_npu #(

parameter WIDTH = 32,

parameter ADDR = 8

)(

// Clock and reset

input logic clk,

input logic rst,

// AXI-like interface

input logic valid,

input logic [ADDR-1:0] local\_addr,

input logic [WIDTH-1:0] din,

input logic ready,

input logic we,

// Control signals

input logic start, // npu start

input logic read\_opt\_top,

input logic load\_ip\_mem,

input logic load\_ip\_top,

input logic [ADDR-1:0] load\_mem\_addr,

// Outputs

output logic [WIDTH-1:0] final\_opt,

output logic done,

output logic normal

);

logic [ADDR-1:0] load\_mem\_addr\_w;

assign load\_mem\_addr\_w =load\_mem\_addr;

// Internal connections

logic [WIDTH-1:0] input\_data\_ext\_w [0:8];

logic [WIDTH-1:0] weights1\_ext\_w [0:8];

logic [WIDTH-1:0] weights2\_ext\_w [0:8];

logic [WIDTH-1:0] weights3\_ext\_w [0:8];

logic [WIDTH-1:0] matmul\_weights\_ext\_w [0:2][0:2];

logic [WIDTH-1:0] final\_output\_w;

// Instantiate AXI interface

sram #(

.ADDR\_WIDTH(ADDR),

.DATA\_WIDTH(WIDTH)

) axi\_inst (

.clk(clk),

.rst(rst),

.load\_mem\_addr(load\_mem\_addr\_w),

.valid(valid),

.local\_addr(local\_addr),

.din(din),

.final\_opt\_top(final\_output\_w),

.ready(ready),

.we(we),

.input\_data\_ext(input\_data\_ext\_w),

.weights1\_ext(weights1\_ext\_w),

.weights2\_ext(weights2\_ext\_w),

.weights3\_ext(weights3\_ext\_w),

.matmul\_weights\_ext(matmul\_weights\_ext\_w),

.final\_opt(final\_opt)

);

// Instantiate top module

core #(

.WIDTH(WIDTH),

.ADDR(ADDR)

) npu\_top (

.clk(clk),

.rst(rst),

.start(start),

.read\_opt\_top(read\_opt\_top),

.load\_ip\_mem(load\_ip\_mem),

.load\_ip\_top(load\_ip\_top),

.addr(load\_mem\_addr), // Connected to local\_addr from AXI

.input\_data\_ext(input\_data\_ext\_w),

.weights1\_ext(weights1\_ext\_w),

.weights2\_ext(weights2\_ext\_w),

.weights3\_ext(weights3\_ext\_w),

.matmul\_weights\_ext(matmul\_weights\_ext\_w),

.final\_output(final\_output\_w),

.done(done)

);

npu\_classifier #(

.WIDTH(WIDTH),

.THRESHOLD(32'd1000000)

) classifier (

.final\_output(final\_opt),

.is\_normal(normal)

);

endmodule

///

module sram # (

parameter ADDR\_WIDTH = 8,

parameter DATA\_WIDTH = 32

)(

input wire clk,

input wire rst,

input logic [ADDR\_WIDTH-1:0] load\_mem\_addr,

input logic valid,

input logic [ADDR\_WIDTH-1:0] local\_addr,

input logic [DATA\_WIDTH-1:0] din,

input logic [DATA\_WIDTH-1:0] final\_opt\_top,

input logic ready,

input logic we,

output logic [DATA\_WIDTH-1:0] input\_data\_ext [0:8],

output logic [DATA\_WIDTH-1:0] weights1\_ext [0:8],

output logic [DATA\_WIDTH-1:0] weights2\_ext [0:8],

output logic [DATA\_WIDTH-1:0] weights3\_ext [0:8],

output logic [DATA\_WIDTH-1:0] matmul\_weights\_ext [0:2][0:2],

output logic [DATA\_WIDTH-1:0] final\_opt

);

reg [ADDR\_WIDTH-1:0] addr;

(\* ram\_style = "block", keep = "true" \*) reg [DATA\_WIDTH-1:0] mem [0:(1<<ADDR\_WIDTH)-1];

always\_ff @(posedge clk)begin

if (we) begin

addr =addr+1;

end

else begin

addr =local\_addr;

end

end

always\_ff @(posedge clk)begin

if (rst)begin

for (int i = 0; i < (1<<ADDR\_WIDTH); i = i + 1)

mem[i] <= '0;

end

if (we)begin

mem[addr] <= din;

$display("[%0t] SRAM Write: addr=%h, data=%h",

$time, addr, din);

end

else if (ready)begin

final\_opt <= final\_opt\_top;

// mem[local\_addr + 45]<=final\_opt\_top;

end

else if (valid) begin

input\_data\_ext[0] <= mem[load\_mem\_addr +0];

input\_data\_ext[1] <= mem[load\_mem\_addr +1];

input\_data\_ext [2] <= mem[load\_mem\_addr +2];

input\_data\_ext [3] <= mem[load\_mem\_addr +3];

input\_data\_ext [4] <= mem[load\_mem\_addr +4];

input\_data\_ext [5] <= mem[load\_mem\_addr +5];

input\_data\_ext [6] <= mem[load\_mem\_addr +6];

input\_data\_ext [7] <= mem[load\_mem\_addr +7];

input\_data\_ext [8] <= mem[load\_mem\_addr +8];

// Load Weights 1

weights1\_ext[0] <= mem[load\_mem\_addr +9];

weights1\_ext[1] <= mem[load\_mem\_addr +10];

weights1\_ext[2] <= mem[load\_mem\_addr +11];

weights1\_ext[3] <= mem[load\_mem\_addr +12];

weights1\_ext[4] <= mem[load\_mem\_addr +13];

weights1\_ext[5] <= mem[load\_mem\_addr +14];

weights1\_ext[6] <= mem[load\_mem\_addr +15];

weights1\_ext[7] <= mem[load\_mem\_addr +16];

weights1\_ext[8] <= mem[load\_mem\_addr +17];

// Load Weights 2

weights2\_ext[0] <= mem[load\_mem\_addr +18];

weights2\_ext[1] <= mem[load\_mem\_addr +19];

weights2\_ext[2] <= mem[load\_mem\_addr +20];

weights2\_ext[3] <= mem[load\_mem\_addr +21];

weights2\_ext[4] <= mem[load\_mem\_addr +22];

weights2\_ext[5] <= mem[load\_mem\_addr +23];

weights2\_ext[6] <= mem[load\_mem\_addr +24];

weights2\_ext[7] <= mem[load\_mem\_addr +25];

weights2\_ext[8] <= mem[load\_mem\_addr +26];

// Load Weights 3

weights3\_ext[0] <= mem[load\_mem\_addr +27];

weights3\_ext[1] <= mem[load\_mem\_addr +28];

weights3\_ext[2] <= mem[load\_mem\_addr +29];

weights3\_ext[3] <= mem[load\_mem\_addr +30];

weights3\_ext[4] <= mem[load\_mem\_addr +31];

weights3\_ext[5] <= mem[load\_mem\_addr +32];

weights3\_ext[6] <= mem[load\_mem\_addr +33];

weights3\_ext[7] <= mem[load\_mem\_addr +34];

weights3\_ext[8] <= mem[load\_mem\_addr +35];

// Load MatMul Weights (2D Array)

matmul\_weights\_ext[0][0] <= mem[load\_mem\_addr + 36];

matmul\_weights\_ext[0][1] <= mem[load\_mem\_addr + 37];

matmul\_weights\_ext[0][2] <= mem[load\_mem\_addr + 38];

matmul\_weights\_ext[1][0] <= mem[load\_mem\_addr + 39];

matmul\_weights\_ext[1][1] <= mem[load\_mem\_addr + 40];

matmul\_weights\_ext[1][2] <= mem[load\_mem\_addr + 41];

matmul\_weights\_ext[2][0] <= mem[load\_mem\_addr + 42];

matmul\_weights\_ext[2][1] <= mem[load\_mem\_addr + 43];

matmul\_weights\_ext[2][2] <= mem[load\_mem\_addr + 44];

end

end

// for Initialize output arrays (example - you should provide real values)

endmodule

//

module core#(parameter WIDTH = 32,ADDR=8)(

input logic clk,

input logic rst ,

input logic start,// npu start

input logic read\_opt\_top,

input logic load\_ip\_mem,

input logic load\_ip\_top,

input logic [7:0] addr,

input logic [WIDTH-1:0] input\_data\_ext [0:8],

input logic [WIDTH-1:0] weights1\_ext [0:8],

input logic [WIDTH-1:0] weights2\_ext [0:8],

input logic [WIDTH-1:0] weights3\_ext [0:8],

input logic [WIDTH-1:0] matmul\_weights\_ext [0:2][0:2],

output logic [WIDTH-1:0] final\_output,

output logic done

);

// wires from memory to top npu

logic [WIDTH-1:0] input\_data\_to\_top\_w [0:8];

logic [WIDTH-1:0] weights1\_to\_top\_w [0:8];

logic [WIDTH-1:0] weights2\_to\_top\_w [0:8];

logic [WIDTH-1:0] weights3\_to\_top\_w [0:8];

logic [WIDTH-1:0] matmul\_weights\_to\_top\_w [0:2][0:2];

///wires from top nup to memory

logic [WIDTH-1:0] conv\_out1\_w;

logic [WIDTH-1:0] conv\_out2\_w;

logic [WIDTH-1:0] conv\_out3\_w;

logic [WIDTH-1:0] matmul\_out1\_w;

logic [WIDTH-1:0] matmul\_out2\_w;

logic [WIDTH-1:0] matmul\_out3\_w;

logic [WIDTH-1:0] final\_output\_w;

/// npu module

memory\_npu\_system\_top #(WIDTH) npu\_sys(

.clk(clk),

.rst(rst),

.start(start),

.done(done),

// External Memory Interfaces

.input\_data\_ext(input\_data\_to\_top\_w) ,

.weights1\_ext (weights1\_to\_top\_w) ,

.weights2\_ext (weights2\_to\_top\_w) ,

.weights3\_ext (weights3\_to\_top\_w) ,

.matmul\_weights\_ext(matmul\_weights\_to\_top\_w) ,

// NPU Outputs

.conv\_out1(conv\_out1\_w),

.conv\_out2(conv\_out2\_w),

. conv\_out3(conv\_out3\_w),

. matmul\_out1(matmul\_out1\_w),

.matmul\_out2(matmul\_out2\_w),

.matmul\_out3(matmul\_out3\_w),

.final\_output(final\_output\_w)

);

//// memory module

memory#( WIDTH,ADDR)memory (

.clk(clk),

.rst(rst),

.load\_ip\_mem(load\_ip\_mem),

. load\_ip\_top(load\_ip\_top),

. read\_opt\_top(read\_opt\_top),

.load\_mem\_addr(addr),

/\*

if the starting is 0 then the output will be automaticaly stored in the 48 postion

if we have to load another either rest evry thng or give the load\_mem\_addr as 49

\*/

// External Memory Interfaces

// External input data and weights (from testbench)

.input\_data\_ext(input\_data\_ext) ,

.weights1\_ext(weights1\_ext),

.weights2\_ext(weights2\_ext ) ,

.weights3\_ext(weights3\_ext) ,

.matmul\_weights\_ext(matmul\_weights\_ext) ,

// Data outputs to Top Module

.input\_data\_to\_top(input\_data\_to\_top\_w) ,

.weights1\_to\_top(weights1\_to\_top\_w),

.weights2\_to\_top (weights2\_to\_top\_w) ,

.weights3\_to\_top (weights3\_to\_top\_w) ,

.matmul\_weights\_to\_top(matmul\_weights\_to\_top\_w) ,

// NPU Outputs

.conv\_out1(conv\_out1\_w),

.conv\_out2(conv\_out2\_w),

.conv\_out3(conv\_out3\_w),

.matmul\_out1(matmul\_out1\_w),

.matmul\_out2(matmul\_out2\_w),

.matmul\_out3(matmul\_out3\_w),

.final\_output(final\_output\_w),

. final\_output\_d(final\_output)

);

endmodule

//

module memory\_npu\_system\_top #(parameter WIDTH = 32)(

input logic clk,

input logic rst,

input logic start,

output logic done,

// External Memory Interfaces

input logic [WIDTH-1:0] input\_data\_ext [0:8],

input logic [WIDTH-1:0] weights1\_ext [0:8],

input logic [WIDTH-1:0] weights2\_ext [0:8],

input logic [WIDTH-1:0] weights3\_ext [0:8],

input logic [WIDTH-1:0] matmul\_weights\_ext [0:2][0:2],

// NPU Outputs

output logic [WIDTH-1:0] conv\_out1,

output logic [WIDTH-1:0] conv\_out2,

output logic [WIDTH-1:0] conv\_out3,

output logic [WIDTH-1:0] matmul\_out1,

output logic [WIDTH-1:0] matmul\_out2,

output logic [WIDTH-1:0] matmul\_out3,

output logic [WIDTH-1:0] final\_output

);

// FSM

typedef enum logic [1:0] {IDLE, RUN} state\_t;

state\_t state;

logic npu\_start, npu\_done;

always\_ff @(posedge clk or posedge rst) begin

if (rst) begin

state <= IDLE;

npu\_start <= 0;

end else begin

case (state)

IDLE: begin

if (start) begin

npu\_start <= 1;

state <= RUN;

end

end

RUN: begin

if (npu\_done) begin

npu\_start <= 0;

state <= IDLE;

end

end

endcase

end

end

assign done = (state == IDLE && start == 0 && npu\_done);

// Instantiate NPU Core with external SRAM-like memory

npu\_pipeline\_top #(WIDTH) core (

.clk(clk),

.rst(rst),

.start(npu\_start),

.input\_data(input\_data\_ext),

.weights1(weights1\_ext),

.weights2(weights2\_ext),

.weights3(weights3\_ext),

.matmul\_weights(matmul\_weights\_ext),

.done(npu\_done),

.conv\_out1(conv\_out1),

.conv\_out2(conv\_out2),

.conv\_out3(conv\_out3),

.matmul\_out1(matmul\_out1),

.matmul\_out2(matmul\_out2),

.matmul\_out3(matmul\_out3),

.final\_output(final\_output)

);

endmodule

//

module npu\_pipeline\_top #(parameter WIDTH = 32)(

input logic clk,

input logic rst,

input logic start,

input logic [WIDTH-1:0] input\_data[0:8],

input logic [WIDTH-1:0] weights1[0:8],

input logic [WIDTH-1:0] weights2[0:8],

input logic [WIDTH-1:0] weights3[0:8],

input logic [WIDTH-1:0] matmul\_weights[0:2][0:2],

output logic done,

output logic [WIDTH-1:0] conv\_out1,

output logic [WIDTH-1:0] conv\_out2,

output logic [WIDTH-1:0] conv\_out3,

output logic [WIDTH-1:0] matmul\_out1,

output logic [WIDTH-1:0] matmul\_out2,

output logic [WIDTH-1:0] matmul\_out3,

output logic [WIDTH-1:0] final\_output

);

// Internal registers

logic [WIDTH-1:0] l1\_input\_data[0:8];

logic [WIDTH-1:0] l1\_weights[0:8][0:2];

logic [WIDTH-1:0] l1\_conv\_outputs[0:2];

logic [WIDTH-1:0] l1\_matmul\_weights[0:2][0:2];

logic [WIDTH-1:0] l1\_matmul\_outputs[0:2];

logic compute\_conv, compute\_matmul, compute\_pool;

// FSM States

typedef enum logic [2:0] {

IDLE, LOAD, CONVOLVE, MATMUL, POOL, DONE

} state\_t;

state\_t state, next\_state;

// FSM

always\_ff @(posedge clk or posedge rst) begin

if (rst)

state <= IDLE;

else

state <= next\_state;

end

always\_comb begin

next\_state = state;

compute\_conv = 0;

compute\_matmul = 0;

compute\_pool = 0;

done = 0;

case (state)

IDLE: if (start) next\_state = LOAD;

LOAD: next\_state = CONVOLVE;

CONVOLVE: begin

compute\_conv = 1;

next\_state = MATMUL;

end

MATMUL: begin

compute\_matmul = 1;

next\_state = POOL;

end

POOL: begin

compute\_pool = 1;

next\_state = DONE;

end

DONE: done = 1;

endcase

end

// Load Inputs

always\_ff @(posedge clk) begin

if (state == LOAD) begin

for (int i = 0; i < 9; i++) begin

l1\_input\_data[i] <= input\_data[i];

l1\_weights[i][0] <= weights1[i];

l1\_weights[i][1] <= weights2[i];

l1\_weights[i][2] <= weights3[i];

end

for (int i = 0; i < 3; i++) begin

for (int j = 0; j < 3; j++) begin

l1\_matmul\_weights[i][j] <= matmul\_weights[i][j];

end

end

end

end

// Convolution + ReLU

always\_ff @(posedge clk) begin

if (compute\_conv) begin

for (int f = 0; f < 3; f++) begin

l1\_conv\_outputs[f] = 0;

for (int i = 0; i < 9; i++) begin

l1\_conv\_outputs[f] += l1\_input\_data[i] \* l1\_weights[i][f];

end

if (l1\_conv\_outputs[f][WIDTH-1])

l1\_conv\_outputs[f] = 0;

end

end

end

// Matrix Multiplication

always\_ff @(posedge clk) begin

if (compute\_matmul) begin

for (int i = 0; i < 3; i++) begin

l1\_matmul\_outputs[i] = 0;

for (int j = 0; j < 3; j++) begin

l1\_matmul\_outputs[i] += l1\_conv\_outputs[j] \* l1\_matmul\_weights[i][j];

end

end

end

end

// Max Pooling

always\_ff @(posedge clk) begin

if (compute\_pool) begin

final\_output = l1\_matmul\_outputs[0];

if (l1\_matmul\_outputs[1] > final\_output)

final\_output = l1\_matmul\_outputs[1];

if (l1\_matmul\_outputs[2] > final\_output)

final\_output = l1\_matmul\_outputs[2];

end

end

// Outputs

assign conv\_out1 = l1\_conv\_outputs[0];

assign conv\_out2 = l1\_conv\_outputs[1];

assign conv\_out3 = l1\_conv\_outputs[2];

assign matmul\_out1 = l1\_matmul\_outputs[0];

assign matmul\_out2 = l1\_matmul\_outputs[1];

assign matmul\_out3 = l1\_matmul\_outputs[2];

endmodule

///

module memory#(parameter WIDTH = 32,ADDR=8)(

input logic clk,

input logic rst,

input logic load\_ip\_mem,

input logic load\_ip\_top,

input logic read\_opt\_top,

input logic [ADDR-1:0]load\_mem\_addr,

/\*

if the starting is 0 then the output will be automaticaly stored in the 48 postion

if we have to load another either rest evry thng or give the load\_mem\_addr as 49

\*/

// External Memory Interfaces

// External input data and weights (from testbench)

input logic [WIDTH-1:0] input\_data\_ext [0:8],

input logic [WIDTH-1:0] weights1\_ext [0:8],

input logic [WIDTH-1:0] weights2\_ext [0:8],

input logic [WIDTH-1:0] weights3\_ext [0:8],

input logic [WIDTH-1:0] matmul\_weights\_ext [0:2][0:2],

// Data outputs to Top Module

output reg [WIDTH-1:0] input\_data\_to\_top [0:8],

output reg [WIDTH-1:0] weights1\_to\_top [0:8],

output reg [WIDTH-1:0] weights2\_to\_top [0:8],

output reg [WIDTH-1:0] weights3\_to\_top [0:8],

output reg [WIDTH-1:0] matmul\_weights\_to\_top [0:2][0:2],

// NPU Outputs

input logic [WIDTH-1:0] conv\_out1,

input logic [WIDTH-1:0] conv\_out2,

input logic [WIDTH-1:0] conv\_out3,

input logic [WIDTH-1:0] matmul\_out1,

input logic [WIDTH-1:0] matmul\_out2,

input logic [WIDTH-1:0] matmul\_out3,

input logic [WIDTH-1:0] final\_output,

output logic [WIDTH-1:0] final\_output\_d

);

logic rst\_sig;

logic done;

// assign load\_mem\_addr=0;

// event write\_mem;

// event load\_top;

// event write\_top;

reg [WIDTH-1:0] mem [0:59];

int i;

always\_ff @(posedge clk)begin

if (rst || rst\_sig)begin

for (int i = 0; i < 60; i = i + 1)

mem[i] <= '0;

end

else if (load\_ip\_mem)begin

mem[0] = input\_data\_ext[0];

mem[1] = input\_data\_ext[1];

mem[2] = input\_data\_ext[2];

mem[3] = input\_data\_ext[3];

mem[4] = input\_data\_ext[4];

mem[5] = input\_data\_ext[5];

mem[6] = input\_data\_ext[6];

mem[7] = input\_data\_ext[7];

mem[8] = input\_data\_ext[8];

//weights 1

mem[9] = weights1\_ext[0];

mem[10] = weights1\_ext[1];

mem[11] = weights1\_ext[2];

mem[12] = weights1\_ext[3];

mem[13] = weights1\_ext[4];

mem[14] = weights1\_ext[5];

mem[15] = weights1\_ext[6];

mem[16] = weights1\_ext[7];

mem[17] = weights1\_ext[8];

// weights 2

mem[18] = weights2\_ext[0];

mem[19] = weights2\_ext[1];

mem[20] = weights2\_ext[2];

mem[21] = weights2\_ext[3];

mem[22] = weights2\_ext[4];

mem[23] = weights2\_ext[5];

mem[24] = weights2\_ext[6];

mem[25] = weights2\_ext[7];

mem[26] = weights2\_ext[8];

// weights 3

mem[27] = weights3\_ext[0];

mem[28] = weights3\_ext[1];

mem[29] = weights3\_ext[2];

mem[30] = weights3\_ext[3];

mem[31] = weights3\_ext[4];

mem[32] = weights3\_ext[5];

mem[33] = weights3\_ext[6];

mem[34] = weights3\_ext[7];

mem[35] = weights3\_ext[8];

// matmul weights

mem[ 36] = matmul\_weights\_ext[0][0];

mem[ 37] = matmul\_weights\_ext[0][1];

mem[ 38] = matmul\_weights\_ext[0][2];

mem[ 39] = matmul\_weights\_ext[1][0];

mem[ 40] = matmul\_weights\_ext[1][1];

mem[ 41] = matmul\_weights\_ext[1][2];

mem[ 42] = matmul\_weights\_ext[2][0];

mem[ 43] = matmul\_weights\_ext[2][1];

mem[ 44] = matmul\_weights\_ext[2][2];

end

else if (load\_ip\_top) begin

// Load Input Data

input\_data\_to\_top [0] = mem[0];

input\_data\_to\_top [1] = mem[1];

input\_data\_to\_top [2] = mem[2];

input\_data\_to\_top [3] = mem[3];

input\_data\_to\_top [4] = mem[4];

input\_data\_to\_top [5] = mem[5];

input\_data\_to\_top [6] = mem[6];

input\_data\_to\_top [7] = mem[7];

input\_data\_to\_top [8] = mem[8];

// Load Weights 1

weights1\_to\_top[0] = mem[9];

weights1\_to\_top[1] = mem[10];

weights1\_to\_top[2] = mem[11];

weights1\_to\_top[3] = mem[12];

weights1\_to\_top[4] = mem[13];

weights1\_to\_top[5] = mem[14];

weights1\_to\_top[6] = mem[15];

weights1\_to\_top[7] = mem[16];

weights1\_to\_top[8] = mem[17];

// Load Weights 2

weights2\_to\_top[0] = mem[18];

weights2\_to\_top[1] = mem[19];

weights2\_to\_top[2] = mem[20];

weights2\_to\_top[3] = mem[21];

weights2\_to\_top[4] = mem[22];

weights2\_to\_top[5] = mem[23];

weights2\_to\_top[6] = mem[24];

weights2\_to\_top[7] = mem[25];

weights2\_to\_top[8] = mem[26];

// Load Weights 3

weights3\_to\_top[0] = mem[27];

weights3\_to\_top[1] = mem[28];

weights3\_to\_top[2] = mem[29];

weights3\_to\_top[3] = mem[30];

weights3\_to\_top[4] = mem[31];

weights3\_to\_top[5] = mem[32];

weights3\_to\_top[6] = mem[33];

weights3\_to\_top[7] = mem[34];

weights3\_to\_top[8] = mem[35];

// Load MatMul Weights (2D Array)

matmul\_weights\_to\_top[0][0] = mem[ 36];

matmul\_weights\_to\_top[0][1] = mem[ 37];

matmul\_weights\_to\_top[0][2] = mem[ 38];

matmul\_weights\_to\_top[1][0] = mem[ 39];

matmul\_weights\_to\_top[1][1] = mem[ 40];

matmul\_weights\_to\_top[1][2] = mem[ 41];

matmul\_weights\_to\_top[2][0] = mem[ 42];

matmul\_weights\_to\_top[2][1] = mem[ 43];

matmul\_weights\_to\_top[2][2] = mem[ 44];

end

else if (read\_opt\_top)begin

$display("final output :%0d",final\_output);

mem[ 45]=conv\_out1;

mem[ 46]=conv\_out2;

mem[ 47]=conv\_out3;

mem[ 48]=matmul\_out1;

mem[ 49]=matmul\_out2;

mem[ 50]=matmul\_out3;

mem[ 51]=final\_output;

done=1;

end

else if(done)begin

final\_output\_d= mem[51];

rst\_sig=1;

end

end

endmodule

module npu\_classifier#(

parameter WIDTH = 32,

parameter THRESHOLD = 32'd1000000

)(

input logic [WIDTH-1:0] final\_output,

output logic is\_normal

);

always\_comb begin

if (final\_output < THRESHOLD)

is\_normal = 1'b1; // Normal

else

is\_normal = 1'b0; // Abnormal

end

endmodule

**AXI WRAPPER**

module myAXIL\_IP #(

// {{{

//

// Size of the AXI-lite bus. These are fixed, since 1) AXI-lite

// is fixed at a width of 32-bits by Xilinx def'n, and 2) since

// we only ever have 4 configuration words.

parameter C\_AXI\_ADDR\_WIDTH = 8,

parameter C\_AXI\_DATA\_WIDTH = 32,

parameter [0:0] OPT\_SKIDBUFFER = 1'b0,

parameter [0:0] OPT\_LOWPOWER = 0,

// }}}

parameter WIDTH = 8

) (

// {{{

input wire S\_AXI\_ACLK,

input wire S\_AXI\_ARESETN,

//

input wire S\_AXI\_AWVALID,

output wire S\_AXI\_AWREADY,

input wire [C\_AXI\_ADDR\_WIDTH-1:0] S\_AXI\_AWADDR,

input wire [2:0] S\_AXI\_AWPROT,

//

input wire S\_AXI\_WVALID,

output wire S\_AXI\_WREADY,

input wire [C\_AXI\_DATA\_WIDTH-1:0] S\_AXI\_WDATA,

input wire [C\_AXI\_DATA\_WIDTH/8-1:0] S\_AXI\_WSTRB,

//

output wire S\_AXI\_BVALID,

input wire S\_AXI\_BREADY,

output wire [1:0] S\_AXI\_BRESP,

//

input wire S\_AXI\_ARVALID,

output wire S\_AXI\_ARREADY,

input wire [C\_AXI\_ADDR\_WIDTH-1:0] S\_AXI\_ARADDR,

input wire [2:0] S\_AXI\_ARPROT,

//

output wire S\_AXI\_RVALID,

input wire S\_AXI\_RREADY,

output wire [C\_AXI\_DATA\_WIDTH-1:0] S\_AXI\_RDATA,

output wire [1:0] S\_AXI\_RRESP,

// }}}

input wire [C\_AXI\_DATA\_WIDTH-1:0] s\_axis\_tdata,

input wire s\_axis\_tvalid,

output wire s\_axis\_tready,

input wire s\_axis\_tlast,

// AXI-Stream master (to host)

output wire [C\_AXI\_DATA\_WIDTH-1:0] m\_axis\_tdata,

output wire m\_axis\_tvalid,

input wire m\_axis\_tready,

output wire m\_axis\_tlast

// Add custom ports here

);

// These are the AXI Lite registers (4 times 32-bit). The names can be changed to something more meaningful.

wire rst = ~S\_AXI\_ARESETN; // active-high reset

Wire valid\_i1,done\_i1,ready\_i1,start\_i1,read\_opt\_top\_i1,load\_ip\_mem\_i1,load\_ip\_top\_i1, write\_en\_i1;

wire[WIDTH+23:0] din\_i1;

wire [WIDTH-1:0] local\_addr\_i1,load\_mem\_addr\_i1;

wire[WIDTH+23:0] final\_opt\_i1;

easyaxil #(

.C\_AXI\_ADDR\_WIDTH( C\_AXI\_ADDR\_WIDTH ),

.C\_AXI\_DATA\_WIDTH( C\_AXI\_DATA\_WIDTH ),

.OPT\_SKIDBUFFER(OPT\_SKIDBUFFER),

.OPT\_LOWPOWER(OPT\_LOWPOWER),

.WIDTH(WIDTH))

control\_easyaxil\_U(

.S\_AXI\_ACLK(S\_AXI\_ACLK),

.S\_AXI\_ARESETN(S\_AXI\_ARESETN),

.S\_AXI\_AWVALID(S\_AXI\_AWVALID),

.S\_AXI\_AWREADY(S\_AXI\_AWREADY),

.S\_AXI\_AWADDR(S\_AXI\_AWADDR),

.S\_AXI\_AWPROT(S\_AXI\_AWPROT),

.S\_AXI\_WVALID(S\_AXI\_WVALID),

.S\_AXI\_WREADY(S\_AXI\_WREADY),

.S\_AXI\_WDATA(S\_AXI\_WDATA),

.S\_AXI\_WSTRB(S\_AXI\_WSTRB),

.S\_AXI\_BVALID(S\_AXI\_BVALID),

.S\_AXI\_BREADY(S\_AXI\_BREADY),

.S\_AXI\_BRESP(S\_AXI\_BRESP),

.S\_AXI\_ARVALID(S\_AXI\_ARVALID),

.S\_AXI\_ARREADY(S\_AXI\_ARREADY),

.S\_AXI\_ARADDR(S\_AXI\_ARADDR),

.S\_AXI\_ARPROT(S\_AXI\_ARPROT),

.S\_AXI\_RVALID(S\_AXI\_RVALID),

.S\_AXI\_RREADY(S\_AXI\_RREADY),

.S\_AXI\_RDATA(S\_AXI\_RDATA),

.S\_AXI\_RRESP(S\_AXI\_RRESP),

.valid(valid\_i1),

.local\_addr(local\_addr\_i1),

.ready(ready\_i1),

.start(start\_i1), // controlled via AXI-Lite

.read\_opt\_top(read\_opt\_top\_i1), // FIXED

.load\_ip\_mem(load\_ip\_mem\_i1), // FIXED

.load\_ip\_top(load\_ip\_top\_i1), // FIXED

.load\_mem\_addr(load\_mem\_addr\_i1),

.final\_opt(final\_opt\_i1),

.done(done\_i1)

);

wire [C\_AXI\_DATA\_WIDTH-1:0] din\_wire;

wire write\_en\_wire;

axistream #(

.ADDR\_WIDTH(WIDTH),

.DATA\_WIDTH(C\_AXI\_DATA\_WIDTH)

) axistream\_inst (

.clk(S\_AXI\_ACLK),

.rst(rst),

.s\_axis\_tdata(s\_axis\_tdata),

.s\_axis\_tvalid(s\_axis\_tvalid),

.s\_axis\_tready(s\_axis\_tready),

.s\_axis\_tlast(s\_axis\_tlast),

.m\_axis\_tdata(m\_axis\_tdata),

.m\_axis\_tvalid(m\_axis\_tvalid),

.m\_axis\_tready(m\_axis\_tready),

.m\_axis\_tlast(m\_axis\_tlast),

.din(din\_i1),

.write\_en(write\_en\_i1),

.final\_opt(final\_opt\_i1),

.done(done\_i1),

.ready(ready\_i1)

);

// Then connect din\_wire and write\_en\_wire as inputs to your NPU or other logic.

top\_npu #(

.WIDTH(C\_AXI\_DATA\_WIDTH),

.ADDR(WIDTH)

) top\_npu\_inst (

.clk(S\_AXI\_ACLK),

.rst(S\_AXI\_ARESETN),

.valid(valid\_i1),

.local\_addr(local\_addr\_i1),

.ready(ready\_i1),

.start(start\_i1), // controlled via AXI-Lite

.read\_opt\_top(read\_opt\_top\_i1), // FIXED

.load\_ip\_mem(load\_ip\_mem\_i1), // FIXED

.load\_ip\_top(load\_ip\_top\_i1), // FIXED

.load\_mem\_addr(load\_mem\_addr\_i1),

.final\_opt(final\_opt\_i1),

.done(done\_i1),

.din(din\_i1),

.we(write\_en\_i1)

);

endmodule

//

`default\_nettype none

// }}}

module easyaxil #(

// {{{

//

// Size of the AXI-lite bus. These are fixed, since 1) AXI-lite

// is fixed at a width of 32-bits by Xilinx def'n, and 2) since

// we only ever have 4 configuration words.

parameter C\_AXI\_ADDR\_WIDTH = 8,

parameter C\_AXI\_DATA\_WIDTH = 32,

parameter [0:0] OPT\_SKIDBUFFER = 1'b0,

parameter [0:0] OPT\_LOWPOWER = 0,

//User Parameters

parameter WIDTH = 8

// }}}

) (

//User Ports

output wire start,

output wire[WIDTH-1:0] load\_mem\_addr,

output wire[WIDTH-1:0] local\_addr,

output wire load\_ip\_mem,

output wire load\_ip\_top,

output wire read\_opt\_top,

output wire valid,

output wire ready,

input wire[WIDTH+23:0] final\_opt,

input wire done,

// {{{

input wire S\_AXI\_ACLK,

input wire S\_AXI\_ARESETN,

//

input wire S\_AXI\_AWVALID,

output wire S\_AXI\_AWREADY,

input wire [C\_AXI\_ADDR\_WIDTH-1:0] S\_AXI\_AWADDR,

input wire [2:0] S\_AXI\_AWPROT,

//

input wire S\_AXI\_WVALID,

output wire S\_AXI\_WREADY,

input wire [C\_AXI\_DATA\_WIDTH-1:0] S\_AXI\_WDATA,

input wire [C\_AXI\_DATA\_WIDTH/8-1:0] S\_AXI\_WSTRB,

//

output wire S\_AXI\_BVALID,

input wire S\_AXI\_BREADY,

output wire [1:0] S\_AXI\_BRESP,

//

input wire S\_AXI\_ARVALID,

output wire S\_AXI\_ARREADY,

input wire [C\_AXI\_ADDR\_WIDTH-1:0] S\_AXI\_ARADDR,

input wire [2:0] S\_AXI\_ARPROT,

//

output wire S\_AXI\_RVALID,

input wire S\_AXI\_RREADY,

output wire [C\_AXI\_DATA\_WIDTH-1:0] S\_AXI\_RDATA,

output wire [1:0] S\_AXI\_RRESP

// }}

);

////////////////////////////////////////////////////////////////////////

//

// Register/wire signal declarations

// {{{

////////////////////////////////////////////////////////////////////////

//

localparam ADDRLSB = $clog2(C\_AXI\_DATA\_WIDTH)-3;

wire i\_reset = !S\_AXI\_ARESETN;

wire axil\_write\_ready;

wire [C\_AXI\_ADDR\_WIDTH-ADDRLSB-4:0] awskd\_addr;

//

wire [C\_AXI\_DATA\_WIDTH-1:0] wskd\_data;

wire [C\_AXI\_DATA\_WIDTH/8-1:0] wskd\_strb;

reg axil\_bvalid;

//

wire axil\_read\_ready;

wire [C\_AXI\_ADDR\_WIDTH-ADDRLSB-4:0] arskd\_addr;

reg [C\_AXI\_DATA\_WIDTH-1:0] axil\_read\_data;

reg axil\_read\_valid;

reg [31:0] r0, r1, r2, r3,r4,r5,r6,r7;

wire [31:0] wskd\_r0, wskd\_r1, wskd\_r2, wskd\_r3,wskd\_r4,wskd\_r5,wskd\_r6,wskd\_r7;

wire[C\_AXI\_DATA\_WIDTH - 1: 0] valid\_i,done\_i,ready\_i,start\_i,read\_opt\_top\_i,load\_ip\_mem\_i,load\_ip\_top\_i;

wire[C\_AXI\_DATA\_WIDTH - 1: 0] local\_addr\_i,load\_mem\_addr\_i;

wire[C\_AXI\_DATA\_WIDTH - 1: 0] final\_opt\_i;

// }}}

////////////////////////////////////////////////////////////////////////

//

// AXI-lite signaling

//

////////////////////////////////////////////////////////////////////////

//

// {{{

//

// Write signaling

//

// {{{

generate if (OPT\_SKIDBUFFER)

begin : SKIDBUFFER\_WRITE

// {{{

wire awskd\_valid, wskd\_valid;

skidbuffer #(.OPT\_OUTREG(0),

.OPT\_LOWPOWER(OPT\_LOWPOWER),

.DW(C\_AXI\_ADDR\_WIDTH-ADDRLSB))

axilawskid(//

.i\_clk(S\_AXI\_ACLK), .i\_reset(i\_reset),

.i\_valid(S\_AXI\_AWVALID), .o\_ready(S\_AXI\_AWREADY),

.i\_data(S\_AXI\_AWADDR[C\_AXI\_ADDR\_WIDTH-4:ADDRLSB]),

.o\_valid(awskd\_valid), .i\_ready(axil\_write\_ready),

.o\_data(awskd\_addr));

skidbuffer #(.OPT\_OUTREG(0),

.OPT\_LOWPOWER(OPT\_LOWPOWER),

.DW(C\_AXI\_DATA\_WIDTH+C\_AXI\_DATA\_WIDTH/8))

axilwskid(//

.i\_clk(S\_AXI\_ACLK), .i\_reset(i\_reset),

.i\_valid(S\_AXI\_WVALID), .o\_ready(S\_AXI\_WREADY),

.i\_data({ S\_AXI\_WDATA, S\_AXI\_WSTRB }),

.o\_valid(wskd\_valid), .i\_ready(axil\_write\_ready),

.o\_data({ wskd\_data, wskd\_strb }));

assign axil\_write\_ready = awskd\_valid && wskd\_valid

&& (!S\_AXI\_BVALID || S\_AXI\_BREADY);

// }}}

end else begin : SIMPLE\_WRITES

// {{{

reg axil\_awready;

initial axil\_awready = 1'b0;

always @(posedge S\_AXI\_ACLK)

if (!S\_AXI\_ARESETN)

axil\_awready <= 1'b0;

else

axil\_awready <= !axil\_awready

&& (S\_AXI\_AWVALID && S\_AXI\_WVALID)

&& (!S\_AXI\_BVALID || S\_AXI\_BREADY);

assign S\_AXI\_AWREADY = axil\_awready;

assign S\_AXI\_WREADY = axil\_awready;

assign awskd\_addr = S\_AXI\_AWADDR[C\_AXI\_ADDR\_WIDTH-4:ADDRLSB];

assign wskd\_data = S\_AXI\_WDATA;

assign wskd\_strb = S\_AXI\_WSTRB;

assign axil\_write\_ready = axil\_awready;

// }}}

end endgenerate

initial axil\_bvalid = 0;

always @(posedge S\_AXI\_ACLK)

if (i\_reset)

axil\_bvalid <= 0;

else if (axil\_write\_ready)

axil\_bvalid <= 1;

else if (S\_AXI\_BREADY)

axil\_bvalid <= 0;

assign S\_AXI\_BVALID = axil\_bvalid;

assign S\_AXI\_BRESP = 2'b00;

// }}}

//

// Read signaling

//

// {{{

generate if (OPT\_SKIDBUFFER)

begin : SKIDBUFFER\_READ

// {{{

wire arskd\_valid;

skidbuffer #(.OPT\_OUTREG(0),

.OPT\_LOWPOWER(OPT\_LOWPOWER),

.DW(C\_AXI\_ADDR\_WIDTH-ADDRLSB))

axilarskid(//

.i\_clk(S\_AXI\_ACLK), .i\_reset(i\_reset),

.i\_valid(S\_AXI\_ARVALID), .o\_ready(S\_AXI\_ARREADY),

.i\_data(S\_AXI\_ARADDR[C\_AXI\_ADDR\_WIDTH-4:ADDRLSB]),

.o\_valid(arskd\_valid), .i\_ready(axil\_read\_ready),

.o\_data(arskd\_addr));

assign axil\_read\_ready = arskd\_valid

&& (!axil\_read\_valid || S\_AXI\_RREADY);

// }}}

end else begin : SIMPLE\_READS

// {{{

reg axil\_arready;

always @(\*)

axil\_arready = !S\_AXI\_RVALID;

assign arskd\_addr = S\_AXI\_ARADDR[C\_AXI\_ADDR\_WIDTH-4:ADDRLSB];

assign S\_AXI\_ARREADY = axil\_arready;

assign axil\_read\_ready = (S\_AXI\_ARVALID && S\_AXI\_ARREADY);

// }}}

end endgenerate

initial axil\_read\_valid = 1'b0;

always @(posedge S\_AXI\_ACLK)

if (i\_reset)

axil\_read\_valid <= 1'b0;

else if (axil\_read\_ready)

axil\_read\_valid <= 1'b1;

else if (S\_AXI\_RREADY)

axil\_read\_valid <= 1'b0;

assign S\_AXI\_RVALID = axil\_read\_valid;

assign S\_AXI\_RDATA = axil\_read\_data;

assign S\_AXI\_RRESP = 2'b00;

// }}}

// }}}

////////////////////////////////////////////////////////////////////////

//

// AXI-lite register logic

//

////////////////////////////////////////////////////////////////////////

//

// {{{

// apply\_wstrb(old\_data, new\_data, write\_strobes)

assign wskd\_r0 = apply\_wstrb(r0, wskd\_data, wskd\_strb);

assign wskd\_r1 = apply\_wstrb(r1, wskd\_data, wskd\_strb);

assign wskd\_r2 = apply\_wstrb(r2, wskd\_data, wskd\_strb);

assign wskd\_r3 = apply\_wstrb(r3, wskd\_data, wskd\_strb);

assign wskd\_r4 = apply\_wstrb(r4, wskd\_data, wskd\_strb);

assign wskd\_r5 = apply\_wstrb(r5, wskd\_data, wskd\_strb);

assign wskd\_r6 = apply\_wstrb(r6, wskd\_data, wskd\_strb);

assign wskd\_r7 = apply\_wstrb(r7, wskd\_data, wskd\_strb);

initial r0 = 0;

initial r1 = 0;

initial r2 = 0;

initial r3 = 0;

initial r4 = 0;

initial r5 = 0;

initial r6 = 0;

initial r7 = 0;

always @(posedge S\_AXI\_ACLK)

if (i\_reset)

begin

r0 <= 0;

r1 <= 0;

r2 <= 0;

r3 <= 0;

r4 <= 0;

r5 <= 0;

r6 <= 0;

r7 <= 0;

end else if (axil\_write\_ready)

begin

case(awskd\_addr)

3'b000: r0 <= wskd\_r0;

3'b001: r1 <= wskd\_r1;

3'b010: r2 <= wskd\_r2;

3'b011: r3 <= wskd\_r3;

3'b100: r4 <= wskd\_r4;

3'b101: r5 <= wskd\_r5;

3'b110: r6 <= wskd\_r6;

3'b111: r7 <= wskd\_r7;

endcase

end

initial axil\_read\_data = 0;

always @(posedge S\_AXI\_ACLK)

if (OPT\_LOWPOWER && !S\_AXI\_ARESETN)

axil\_read\_data <= 0;

else if (!S\_AXI\_RVALID || S\_AXI\_RREADY)

begin

case(arskd\_addr)

3'b000: axil\_read\_data <= r0;

3'b001: axil\_read\_data <= r1;

3'b010: axil\_read\_data <= r2;

3'b011: axil\_read\_data <= r3;

3'b100: axil\_read\_data <= done\_i;

3'b101: axil\_read\_data <= final\_opt\_i;

//3'b010: axil\_read\_data <= c\_wire;

//2'b11: axil\_read\_data <= r3;

endcase

if (OPT\_LOWPOWER && !axil\_read\_ready)

axil\_read\_data <= 0;

end

function [C\_AXI\_DATA\_WIDTH-1:0] apply\_wstrb;

input [C\_AXI\_DATA\_WIDTH-1:0] prior\_data;

input [C\_AXI\_DATA\_WIDTH-1:0] new\_data;

input [C\_AXI\_DATA\_WIDTH/8-1:0] wstrb;

integer k;

for(k=0; k<C\_AXI\_DATA\_WIDTH/8; k=k+1)

begin

apply\_wstrb[k\*8 +: 8]

= wstrb[k] ? new\_data[k\*8 +: 8] : prior\_data[k\*8 +: 8];

end

endfunction

// }}}

// User Logic

// assign a\_wire = r0;

// assign b\_wire = r1;

// assign c\_wire[WIDTH : 0] = c;

// assign c\_wire[C\_AXI\_DATA\_WIDTH-1:WIDTH] = 0;

// assign a = a\_wire[WIDTH - 1 : 0];

// assign b = b\_wire[WIDTH - 1 : 0];

assign local\_addr\_i = r0;

assign load\_mem\_addr\_i = r1;

assign valid\_i = r2;

assign load\_ip\_mem\_i = r3;

assign load\_ip\_top\_i = r4;

assign start\_i = r5;

assign read\_opt\_top\_i = r6;

assign ready\_i= r7;

assign done\_i = { {(C\_AXI\_DATA\_WIDTH-1){1'b0}}, done };

// assign done\_i=done;

// assign done\_i[C\_AXI\_DATA\_WIDTH-1:1]= 0;

assign final\_opt\_i[WIDTH+23:0] =final\_opt;

// User Logic

assign valid = valid\_i[0];

assign ready = ready\_i[0];

assign start = start\_i[0];

assign read\_opt\_top =read\_opt\_top\_i[0];

assign load\_ip\_mem = load\_ip\_mem\_i[0];

assign load\_ip\_top = load\_ip\_top\_i[0];

assign local\_addr = local\_addr\_i[WIDTH-1: 0];

assign load\_mem\_addr = load\_mem\_addr\_i[WIDTH-1: 0];

// Make Verilator happy

// {{{

// Verilator lint\_off UNUSED

wire unused;

assign unused = &{ 1'b0, S\_AXI\_AWPROT, S\_AXI\_ARPROT,

S\_AXI\_ARADDR[ADDRLSB-1:0],

S\_AXI\_AWADDR[ADDRLSB-1:0] };

// Verilator lint\_on UNUSED

// }}}

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

//

// Formal properties

// {{{

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

`ifdef FORMAL

////////////////////////////////////////////////////////////////////////

//

// The AXI-lite control interface

//

////////////////////////////////////////////////////////////////////////

//

// {{{

localparam F\_AXIL\_LGDEPTH = 4;

wire [F\_AXIL\_LGDEPTH-1:0] faxil\_rd\_outstanding,

faxil\_wr\_outstanding,

faxil\_awr\_outstanding;

faxil\_slave #(

// {{{

.C\_AXI\_DATA\_WIDTH(C\_AXI\_DATA\_WIDTH),

.C\_AXI\_ADDR\_WIDTH(C\_AXI\_ADDR\_WIDTH),

.F\_LGDEPTH(F\_AXIL\_LGDEPTH),

.F\_AXI\_MAXWAIT(3),

.F\_AXI\_MAXDELAY(3),

.F\_AXI\_MAXRSTALL(5),

.F\_OPT\_COVER\_BURST(4)

// }}}

) faxil(

// {{{

.i\_clk(S\_AXI\_ACLK), .i\_axi\_reset\_n(S\_AXI\_ARESETN),

//

.i\_axi\_awvalid(S\_AXI\_AWVALID),

.i\_axi\_awready(S\_AXI\_AWREADY),

.i\_axi\_awaddr( S\_AXI\_AWADDR),

.i\_axi\_awprot( S\_AXI\_AWPROT),

//

.i\_axi\_wvalid(S\_AXI\_WVALID),

.i\_axi\_wready(S\_AXI\_WREADY),

.i\_axi\_wdata( S\_AXI\_WDATA),

.i\_axi\_wstrb( S\_AXI\_WSTRB),

//

.i\_axi\_bvalid(S\_AXI\_BVALID),

.i\_axi\_bready(S\_AXI\_BREADY),

.i\_axi\_bresp( S\_AXI\_BRESP),

//

.i\_axi\_arvalid(S\_AXI\_ARVALID),

.i\_axi\_arready(S\_AXI\_ARREADY),

.i\_axi\_araddr( S\_AXI\_ARADDR),

.i\_axi\_arprot( S\_AXI\_ARPROT),

//

.i\_axi\_rvalid(S\_AXI\_RVALID),

.i\_axi\_rready(S\_AXI\_RREADY),

.i\_axi\_rdata( S\_AXI\_RDATA),

.i\_axi\_rresp( S\_AXI\_RRESP),

//

.f\_axi\_rd\_outstanding(faxil\_rd\_outstanding),

.f\_axi\_wr\_outstanding(faxil\_wr\_outstanding),

.f\_axi\_awr\_outstanding(faxil\_awr\_outstanding)

// }}}

);

always @(\*)

if (OPT\_SKIDBUFFER)

begin

assert(faxil\_awr\_outstanding== (S\_AXI\_BVALID ? 1:0)

+(S\_AXI\_AWREADY ? 0:1));

assert(faxil\_wr\_outstanding == (S\_AXI\_BVALID ? 1:0)

+(S\_AXI\_WREADY ? 0:1));

assert(faxil\_rd\_outstanding == (S\_AXI\_RVALID ? 1:0)

+(S\_AXI\_ARREADY ? 0:1));

end else begin

assert(faxil\_wr\_outstanding == (S\_AXI\_BVALID ? 1:0));

assert(faxil\_awr\_outstanding == faxil\_wr\_outstanding);

assert(faxil\_rd\_outstanding == (S\_AXI\_RVALID ? 1:0));

end

//

// Check that our low-power only logic works by verifying that anytime

// S\_AXI\_RVALID is inactive, then the outgoing data is also zero.

//

always @(\*)

if (OPT\_LOWPOWER && !S\_AXI\_RVALID)

assert(S\_AXI\_RDATA == 0);

// }}}

////////////////////////////////////////////////////////////////////////

//

// Register return checking

// {{{

////////////////////////////////////////////////////////////////////////

//

//

`define CHECK\_REGISTERS

`ifdef CHECK\_REGISTERS

faxil\_register #(

// {{{

.AW(C\_AXI\_ADDR\_WIDTH),

.DW(C\_AXI\_DATA\_WIDTH),

.ADDR(0)

// }}}

) fr0 (

// {{{

.S\_AXI\_ACLK(S\_AXI\_ACLK),

.S\_AXI\_ARESETN(S\_AXI\_ARESETN),

.S\_AXIL\_AWW(axil\_write\_ready),

.S\_AXIL\_AWADDR({ awskd\_addr, {(ADDRLSB){1'b0}} }),

.S\_AXIL\_WDATA(wskd\_data),

.S\_AXIL\_WSTRB(wskd\_strb),

.S\_AXIL\_BVALID(S\_AXI\_BVALID),

.S\_AXIL\_AR(axil\_read\_ready),

.S\_AXIL\_ARADDR({ arskd\_addr, {(ADDRLSB){1'b0}} }),

.S\_AXIL\_RVALID(S\_AXI\_RVALID),

.S\_AXIL\_RDATA(S\_AXI\_RDATA),

.i\_register(r0)

// }}}

);

faxil\_register #(

// {{{

.AW(C\_AXI\_ADDR\_WIDTH),

.DW(C\_AXI\_DATA\_WIDTH),

.ADDR(4)

// }}}

) fr1 (

// {{{

.S\_AXI\_ACLK(S\_AXI\_ACLK),

.S\_AXI\_ARESETN(S\_AXI\_ARESETN),

.S\_AXIL\_AWW(axil\_write\_ready),

.S\_AXIL\_AWADDR({ awskd\_addr, {(ADDRLSB){1'b0}} }),

.S\_AXIL\_WDATA(wskd\_data),

.S\_AXIL\_WSTRB(wskd\_strb),

.S\_AXIL\_BVALID(S\_AXI\_BVALID),

.S\_AXIL\_AR(axil\_read\_ready),

.S\_AXIL\_ARADDR({ arskd\_addr, {(ADDRLSB){1'b0}} }),

.S\_AXIL\_RVALID(S\_AXI\_RVALID),

.S\_AXIL\_RDATA(S\_AXI\_RDATA),

.i\_register(r1)

// }}}

);

faxil\_register #(

// {{{

.AW(C\_AXI\_ADDR\_WIDTH),

.DW(C\_AXI\_DATA\_WIDTH),

.ADDR(8)

// }}}

) fr2 (

// {{{

.S\_AXI\_ACLK(S\_AXI\_ACLK),

.S\_AXI\_ARESETN(S\_AXI\_ARESETN),

.S\_AXIL\_AWW(axil\_write\_ready),

.S\_AXIL\_AWADDR({ awskd\_addr, {(ADDRLSB){1'b0}} }),

.S\_AXIL\_WDATA(wskd\_data),

.S\_AXIL\_WSTRB(wskd\_strb),

.S\_AXIL\_BVALID(S\_AXI\_BVALID),

.S\_AXIL\_AR(axil\_read\_ready),

.S\_AXIL\_ARADDR({ arskd\_addr, {(ADDRLSB){1'b0}} }),

.S\_AXIL\_RVALID(S\_AXI\_RVALID),

.S\_AXIL\_RDATA(S\_AXI\_RDATA),

.i\_register(r2)

// }}}

);

faxil\_register #(

// {{{

.AW(C\_AXI\_ADDR\_WIDTH),

.DW(C\_AXI\_DATA\_WIDTH),

.ADDR(12)

// }}}

) fr3 (

// {{{

.S\_AXI\_ACLK(S\_AXI\_ACLK),

.S\_AXI\_ARESETN(S\_AXI\_ARESETN),

.S\_AXIL\_AWW(axil\_write\_ready),

.S\_AXIL\_AWADDR({ awskd\_addr, {(ADDRLSB){1'b0}} }),

.S\_AXIL\_WDATA(wskd\_data),

.S\_AXIL\_WSTRB(wskd\_strb),

.S\_AXIL\_BVALID(S\_AXI\_BVALID),

.S\_AXIL\_AR(axil\_read\_ready),

.S\_AXIL\_ARADDR({ arskd\_addr, {(ADDRLSB){1'b0}} }),

.S\_AXIL\_RVALID(S\_AXI\_RVALID),

.S\_AXIL\_RDATA(S\_AXI\_RDATA),

.i\_register(r3)

// }}}

);

`endif

// }}}

////////////////////////////////////////////////////////////////////////

//

// Cover checks

//

////////////////////////////////////////////////////////////////////////

//

// {{{

// While there are already cover properties in the formal property

// set above, you'll probably still want to cover something

// application specific here

// }}}

`endif

// }}}

endmodule

//

`default\_nettype none

// }}}

module skidbuffer #(

// {{{

parameter [0:0] OPT\_LOWPOWER = 0,

parameter [0:0] OPT\_OUTREG = 1,

//

parameter [0:0] OPT\_PASSTHROUGH = 0,

parameter DW = 8,

parameter [0:0] OPT\_INITIAL = 1'b1

// }}}

) (

// {{{

input wire i\_clk, i\_reset,

input wire i\_valid,

output wire o\_ready,

input wire [DW-1:0] i\_data,

output wire o\_valid,

input wire i\_ready,

output reg [DW-1:0] o\_data

// }}}

);

wire [DW-1:0] w\_data;

generate if (OPT\_PASSTHROUGH)

begin : PASSTHROUGH

// {{{

assign { o\_valid, o\_ready } = { i\_valid, i\_ready };

always @(\*)

if (!i\_valid && OPT\_LOWPOWER)

o\_data = 0;

else

o\_data = i\_data;

assign w\_data = 0;

// Keep Verilator happy

// Verilator lint\_off UNUSED

// {{{

wire unused\_passthrough;

assign unused\_passthrough = &{ 1'b0, i\_clk, i\_reset };

// }}}

// Verilator lint\_on UNUSED

// }}}

end else begin : LOGIC

// We'll start with skid buffer itself

// {{{

reg r\_valid;

reg [DW-1:0] r\_data;

// r\_valid

// {{{

initial if (OPT\_INITIAL) r\_valid = 0;

always @(posedge i\_clk)

if (i\_reset)

r\_valid <= 0;

else if ((i\_valid && o\_ready) && (o\_valid && !i\_ready))

// We have incoming data, but the output is stalled

r\_valid <= 1;

else if (i\_ready)

r\_valid <= 0;

// }}}

// r\_data

// {{{

initial if (OPT\_INITIAL) r\_data = 0;

always @(posedge i\_clk)

if (OPT\_LOWPOWER && i\_reset)

r\_data <= 0;

else if (OPT\_LOWPOWER && (!o\_valid || i\_ready))

r\_data <= 0;

else if ((!OPT\_LOWPOWER || !OPT\_OUTREG || i\_valid) && o\_ready)

r\_data <= i\_data;

assign w\_data = r\_data;

// }}}

// o\_ready

// {{{

assign o\_ready = !r\_valid;

// }}}

//

// And then move on to the output port

//

if (!OPT\_OUTREG)

begin : NET\_OUTPUT

// Outputs are combinatorially determined from inputs

// {{{

// o\_valid

// {{{

assign o\_valid = !i\_reset && (i\_valid || r\_valid);

// }}}

// o\_data

// {{{

always @(\*)

if (r\_valid)

o\_data = r\_data;

else if (!OPT\_LOWPOWER || i\_valid)

o\_data = i\_data;

else

o\_data = 0;

// }}}

// }}}

end else begin : REG\_OUTPUT

// Register our outputs

// {{{

// o\_valid

// {{{

reg ro\_valid;

initial if (OPT\_INITIAL) ro\_valid = 0;

always @(posedge i\_clk)

if (i\_reset)

ro\_valid <= 0;

else if (!o\_valid || i\_ready)

ro\_valid <= (i\_valid || r\_valid);

assign o\_valid = ro\_valid;

// }}}

// o\_data

// {{{

initial if (OPT\_INITIAL) o\_data = 0;

always @(posedge i\_clk)

if (OPT\_LOWPOWER && i\_reset)

o\_data <= 0;

else if (!o\_valid || i\_ready)

begin

if (r\_valid)

o\_data <= r\_data;

else if (!OPT\_LOWPOWER || i\_valid)

o\_data <= i\_data;

else

o\_data <= 0;

end

// }}}

// }}}

end

// }}}

end endgenerate

// Keep Verilator happy

// {{{

// verilator coverage\_off

// Verilator lint\_off UNUSED

wire unused;

assign unused = &{ 1'b0, w\_data };

// Verilator lint\_on UNUSED

// verilator coverage\_on

// }}}

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

//

// Formal properties

// {{{

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

`ifdef FORMAL

`ifdef SKIDBUFFER

`define ASSUME assume

`else

`define ASSUME assert

`endif

reg f\_past\_valid;

initial f\_past\_valid = 0;

always @(posedge i\_clk)

f\_past\_valid <= 1;

always @(\*)

if (!f\_past\_valid)

assume(i\_reset);

////////////////////////////////////////////////////////////////////////

//

// Incoming stream properties / assumptions

// {{{

////////////////////////////////////////////////////////////////////////

//

always @(posedge i\_clk)

if (!f\_past\_valid)

begin

`ASSUME(!i\_valid || !OPT\_INITIAL);

end else if ($past(i\_valid && !o\_ready && !i\_reset) && !i\_reset)

`ASSUME(i\_valid && $stable(i\_data));

`ifdef VERIFIC

`define FORMAL\_VERIFIC

// Reset properties

property RESET\_CLEARS\_IVALID;

@(posedge i\_clk) i\_reset |=> !i\_valid;

endproperty

property IDATA\_HELD\_WHEN\_NOT\_READY;

@(posedge i\_clk) disable iff (i\_reset)

i\_valid && !o\_ready |=> i\_valid && $stable(i\_data);

endproperty

`ifdef SKIDBUFFER

assume property (IDATA\_HELD\_WHEN\_NOT\_READY);

`else

assert property (IDATA\_HELD\_WHEN\_NOT\_READY);

`endif

`endif

// }}}

////////////////////////////////////////////////////////////////////////

//

// Outgoing stream properties / assumptions

// {{{

////////////////////////////////////////////////////////////////////////

//

generate if (!OPT\_PASSTHROUGH)

begin

always @(posedge i\_clk)

if (!f\_past\_valid) // || $past(i\_reset))

begin

// Following any reset, valid must be deasserted

assert(!o\_valid || !OPT\_INITIAL);

end else if ($past(o\_valid && !i\_ready && !i\_reset) && !i\_reset)

// Following any stall, valid must remain high and

// data must be preserved

assert(o\_valid && $stable(o\_data));

end endgenerate

// }}}

////////////////////////////////////////////////////////////////////////

//

// Other properties

// {{{

////////////////////////////////////////////////////////////////////////

//

//

generate if (!OPT\_PASSTHROUGH)

begin

// Rule #1:

// If registered, then following any reset we should be

// ready for a new request

// {{{

always @(posedge i\_clk)

if (f\_past\_valid && $past(OPT\_OUTREG && i\_reset))

assert(o\_ready);

// }}}

// Rule #2:

// All incoming data must either go directly to the

// output port, or into the skid buffer

// {{{

`ifndef VERIFIC

always @(posedge i\_clk)

if (f\_past\_valid && !$past(i\_reset) && $past(i\_valid && o\_ready

&& (!OPT\_OUTREG || o\_valid) && !i\_ready))

assert(!o\_ready && w\_data == $past(i\_data));

`else

assert property (@(posedge i\_clk)

disable iff (i\_reset)

(i\_valid && o\_ready

&& (!OPT\_OUTREG || o\_valid) && !i\_ready)

|=> (!o\_ready && w\_data == $past(i\_data)));

`endif

// }}}

// Rule #3:

// After the last transaction, o\_valid should become idle

// {{{

if (!OPT\_OUTREG)

begin

// {{{

always @(posedge i\_clk)

if (f\_past\_valid && !$past(i\_reset) && !i\_reset

&& $past(i\_ready))

begin

assert(o\_valid == i\_valid);

assert(!i\_valid || (o\_data == i\_data));

end

// }}}

end else begin

// {{{

always @(posedge i\_clk)

if (f\_past\_valid && !$past(i\_reset))

begin

if ($past(i\_valid && o\_ready))

assert(o\_valid);

if ($past(!i\_valid && o\_ready && i\_ready))

assert(!o\_valid);

end

// }}}

end

// }}}

// Rule #4

// Same thing, but this time for o\_ready

// {{{

always @(posedge i\_clk)

if (f\_past\_valid && $past(!o\_ready && i\_ready))

assert(o\_ready);

// }}}

// If OPT\_LOWPOWER is set, o\_data and w\_data both need to be

// zero any time !o\_valid or !r\_valid respectively

// {{{

if (OPT\_LOWPOWER)

begin

always @(\*)

if ((OPT\_OUTREG || !i\_reset) && !o\_valid)

assert(o\_data == 0);

always @(\*)

if (o\_ready)

assert(w\_data == 0);

end

// }}}

end endgenerate

// }}}

always @(posedge i\_clk)

if (!OPT\_PASSTHROUGH && !i\_reset && !o\_ready)

assert(o\_valid);

////////////////////////////////////////////////////////////////////////

//

// Cover checks

// {{{

////////////////////////////////////////////////////////////////////////

//

//

`ifdef SKIDBUFFER

generate if (!OPT\_PASSTHROUGH)

begin

reg f\_changed\_data;

initial f\_changed\_data = 0;

always @(posedge i\_clk)

if (i\_reset)

f\_changed\_data <= 1;

else if (i\_valid && $past(!i\_valid || o\_ready))

begin

if (i\_data != $past(i\_data + 1))

f\_changed\_data <= 0;

end else if (!i\_valid && i\_data != 0)

f\_changed\_data <= 0;

`ifndef VERIFIC

reg [3:0] cvr\_steps, cvr\_hold;

always @(posedge i\_clk)

if (i\_reset)

begin

cvr\_steps <= 0;

cvr\_hold <= 0;

end else begin

cvr\_steps <= cvr\_steps + 1;

cvr\_hold <= cvr\_hold + 1;

case(cvr\_steps)

0: if (o\_valid || i\_valid)

cvr\_steps <= 0;

1: if (!i\_valid || !i\_ready)

cvr\_steps <= 0;

2: if (!i\_valid || !i\_ready)

cvr\_steps <= 0;

3: if (!i\_valid || !i\_ready)

cvr\_steps <= 0;

4: if (!i\_valid || i\_ready)

cvr\_steps <= 0;

5: if (!i\_valid || !i\_ready)

cvr\_steps <= 0;

6: if (!i\_valid || !i\_ready)

cvr\_steps <= 0;

7: if (!i\_valid || i\_ready)

cvr\_steps <= 0;

8: if (!i\_valid || i\_ready)

cvr\_steps <= 0;

9: if (!i\_valid || !i\_ready)

cvr\_steps <= 0;

10: if (!i\_valid || !i\_ready)

cvr\_steps <= 0;

11: if (!i\_valid || !i\_ready)

cvr\_steps <= 0;

12: begin

cvr\_steps <= cvr\_steps;

cover(!o\_valid && !i\_valid && f\_changed\_data);

if (!o\_valid || !i\_ready)

cvr\_steps <= 0;

else

cvr\_hold <= cvr\_hold + 1;

end

default: assert(0);

endcase

end

`else

// Cover test

cover property (@(posedge i\_clk)

disable iff (i\_reset)

(!o\_valid && !i\_valid)

##1 i\_valid && i\_ready [\*3]

##1 i\_valid && !i\_ready

##1 i\_valid && i\_ready [\*2]

##1 i\_valid && !i\_ready [\*2]

##1 i\_valid && i\_ready [\*3]

// Wait for the design to clear

##1 o\_valid && i\_ready [\*0:5]

##1 (!o\_valid && !i\_valid && f\_changed\_data));

`endif

end endgenerate

`endif // SKIDBUFFER

// }}}

`endif

// }}}

endmodule

module axistream #(

parameter ADDR\_WIDTH = 8,

parameter DATA\_WIDTH = 32

)(

input wire clk,

input wire rst,

// AXI-Stream Slave (input)

input wire [DATA\_WIDTH-1:0] s\_axis\_tdata,

input wire s\_axis\_tvalid,

output reg s\_axis\_tready,

input reg s\_axis\_tlast,

// AXI-Stream Master (output)

output reg [DATA\_WIDTH-1:0] m\_axis\_tdata,

output reg m\_axis\_tvalid,

input wire m\_axis\_tready,

output reg m\_axis\_tlast,

output wire write\_en,

output wire [DATA\_WIDTH-1:0] final\_opt,

output wire done,

input wire ready,

input wire [DATA\_WIDTH-1:0] din

);

// Internal control registers

reg [DATA\_WIDTH-1:0] din2;

reg write\_en2;

reg end\_of\_stream;

reg ready2;

wire [DATA\_WIDTH-1:0] final\_opt2;

wire done2;

// ---------------------------

// AXI-Stream Input (to NPU)

// ---------------------------

assign din2=din;

always @(posedge clk) begin

if (rst) begin

s\_axis\_tready <= 0;

din2 <= 0;

write\_en2 <= 0;

end\_of\_stream <= 0; // Optional flag to track tlast

end else begin

s\_axis\_tready <= 1;

if (s\_axis\_tvalid && s\_axis\_tready) begin

din2 <= s\_axis\_tdata;

write\_en2 <= 1;

if (s\_axis\_tlast) begin

end\_of\_stream <= 1; // You can use this to trigger downstream logic

end else begin

end\_of\_stream <= 0;

end

end else begin

write\_en2 <= 0;

end\_of\_stream <= 0;

end

end

end

// ---------------------------

// AXI-Stream Output (from NPU)

// ---------------------------

reg [3:0] send\_count;

reg sending;

reg done\_latched;

always @(posedge clk) begin

if (rst) begin

m\_axis\_tvalid <= 0;

m\_axis\_tlast <= 0;

m\_axis\_tdata <= 0;

send\_count <= 0;

sending <= 0;

done\_latched <= 0;

end

else begin

// Latch done so it is not missed

if (done2)

done\_latched <= 1;

// Start sending only when done\_latched AND read\_opt\_top are true

if (done\_latched && ready2 && !sending) begin

sending <= 1;

send\_count <= 0;

m\_axis\_tdata <= final\_opt2;

m\_axis\_tvalid <= 1;

m\_axis\_tlast <= 1;

$display("[%0t] Output sending started. final\_opt = %h", $time, final\_opt2);

end

// Wait for handshake

if (sending && m\_axis\_tvalid && m\_axis\_tready) begin

m\_axis\_tvalid <= 0;

m\_axis\_tlast <= 0;

sending <= 0;

done\_latched <= 0;

$display("[%0t] Output sent on AXI-Stream: %h", $time, final\_opt2);

end

end

end

assign din=din2[DATA\_WIDTH-1:0] ;

assign ready = ready2;

assign done = done2;

assign final\_opt = final\_opt2[DATA\_WIDTH-1:0];

assign write\_en =write\_en2;

endmodule