Final Project Report First Page. Must match this format (Single Stage Binary Convolutional Network Hardware Accelerator)

Name:Nathan Carels
Unityid:nlcarels
StudentID:200233416

Delay (ns to run provided example).
Clock period: 4.75ns
cycles: 49

Logic Area: (um^2)
1600

Memory: N/A

1/(delay.area) ($ns^{-1}.um^{-2}$)

2.685e-6

Delay (TA provided example. TA to complete)

1/(delay.area) (TA)

Abstract
This piece of hardware is a single stage binary convolution accelerator that can accept 3 different sized input matrixes. The input, output and weights are all stored in separate SRAMs and the outputs are left zero padded. Convolution is achieved by calculating the XNOR of the weight and input and bit counting the result to determine the sign of the desired output bit. Doing this in parallel allows the module to pipeline inputs and operate with a new memory input every clock cycle. A FSM controls the inputs and outputs and allows the module to process multiple input matrixes until it encounters a matrix terminated with 0x00FF. Each matrix is preempted with an xdim and ydim indicating how many columns and rows the matrix has respectively. The weights are stored in the second address of the weight SRAM and are a 3x3 matrix. Using this approach allowed memory bandwidth to be well utilized.

Single Stage Binary Convolutional Network Hardware Accelerator

Nathan Carels

Abstract

This piece of hardware is a single stage binary convolutional network accelerator that can accept 3 different sized input matrixes. The input, output and weights are all stored in separate SRAMs. The outputs are left zero padded. Convolution is achieved by calculating the XNOR of the weight and input and bit counting the result to determine the sign of the desired output bit. Doing this in parallel allows the module to pipeline inputs and operate with a new memory input every clock cycle. A FSM controls the inputs and outputs and allows the module to process multiple input matrixes until it encounters a matrix terminated with 0x00FF. Each matrix is preempted with an xdim and ydim indicating how many columns and rows the matrix has respectively. The weights are stored in the second address of the weight SRAM and are a 3x3 matrix. Using this approach allowed memory bandwidth to be well utilized.

Report Structure

1. Introduction

1.1 Summary

The hardware designed in this module implements a kernel convolution. The kernel convolution is designed to be used with neural networks that use sign rounding to replace the traditional multiplication and addition with XNORs and Bit counting to determine the output. Utilizing this technique allows more efficient hardware to be created as it avoids large multiplication circuits.

1.2 Challenges

Designing this piece of hardware had a few challenges, particularly with bit counting and controller design. Bit counting was a challenge to find a gate efficient method in computing the sign. The solution that was decided upon was to add the individual bits and compare in order to determine the sign instead of adding 1 for set bits and subtracting 1 for unset bits as that would result in extra multiplexers and gates within the adders. Controlling the outputs properly was challenging due to the pipelined inputs. This was overcame by introducing starting states to get the inputs into a place so the core processing loop to handle multiple matrixes.

1.3 Results Summary

Using this design I was able to achieve a high throughput and maximize using memory bandwidth on the input memory. The final design used 1600 um^2 of space and worked at a clock period of 4.75 ns.

2. Micro-Architecture

2.1 Sign Bit Hardware Algorithm

In order to process a whole row of input data with a 3x3 convolutional matrix, 14 sign bit calculators are need for a 16 bit wide input. The sign was calculated by summing the set bits of the inputs from each row XNOR with the weight row as shown below. Where InMatrix is a 3x3 subset of the input matrix.

$$\sim(\{InMatrix\} \oplus \{WeightMatrix1\}) = IntOut$$

$$\sum_i IntOut[i] > 4 = BitSignOutput$$

By shifting the inputs up and using multiple calculation in parallel for each output bit the output row can be computed once a clock cycle.

2.2 Controller FSM

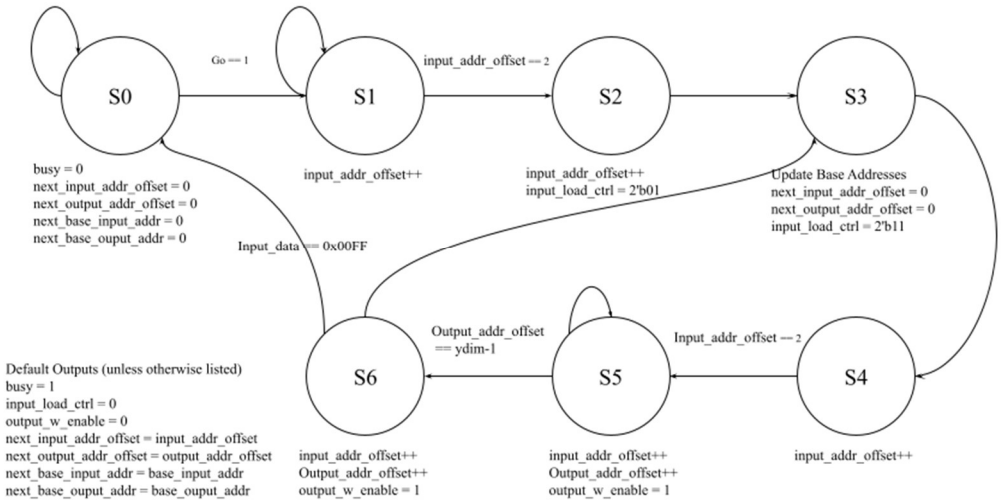


Figure 1: Controller FSM

2.3 Architecture Diagram and Data Flow

The controller generates the address for the weights, inputs and outputs SRAM interfaces which introduce flip flops to the SRAM connections. Data is then received in the interfaces and is piped into the convolution calculator and the dimension registers. Loading of the dimensions and weights is signaled by the controller. The convolution calculator then pipelines the inputs and properly pads the outputs according the x-dimension of the input matrix. Which is then loaded into the output SRAM interface. The controller will then send a write signal when valid outputs are being calculated.

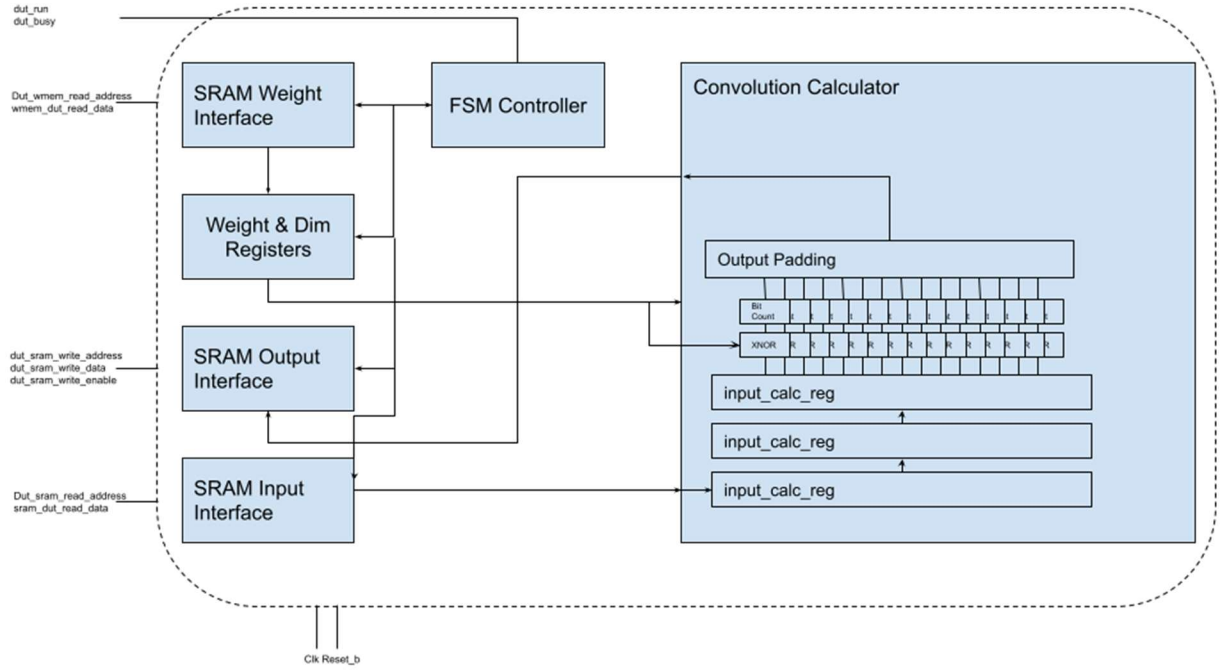


Figure 2: Architecture Drawing

3. Interface Specification

3.1 Description

Interfacing with this hardware consists of 2 input and 1 output SRAMS that require addresses and data lines as well as a write enable line for the output SRAM. It requires a clock and for a reset to be asserted before operation begins. Afterwards the go input will trigger the module to start calculations on the matrix at address 0 while asserting a busy signal high. After it encounters a matrix terminated in 0x00FF it will resume waiting for the next assertion of go.

3.2 Interface Signal Table

Signal (in module order)	Width (Bits)	Signal Description
dut_run	1	Signals module matrix is ready to run.
dut_busy	1	Signals device module is processing.

reset_b	1	Resets the module into wait state and initializes registers to 0.
clk	1	Clock input for module flip flops.
dut_sram_write_address	12	The address in the output SRAM that dut_sram_write_data is stored in.
dut_sram_write_data	16	Data send to output SRAM.
dut_sram_write_enable	1	Must be high for data to be written.
dut_sram_read_address	12	Input SRAM read address.
sram_dut_read_data	16	Input line for data from SRAM. Delayed from address by 1 clock cycle.
dut_wmem_read_address	12	Weight SRAM read address.
wmem_dut_read_data	16	Input line for weight from SRAM. Delayed from address by 1 clock cycle.

3.3 Sample Timing Diagram

The following image is a sample timing diagram for the interface. It shows 3 separate parts of a run with multiple matrixes. First occurs the loading where the module starts reading in data and starting the output. This occurs until the next matrix is encountered which is shown in the second part of the timing diagram and finally followed by the end of processing signaled by a matrix terminated in 0x00FF. F's in the names below are used to indicate the final data or final address that contains an input or output.

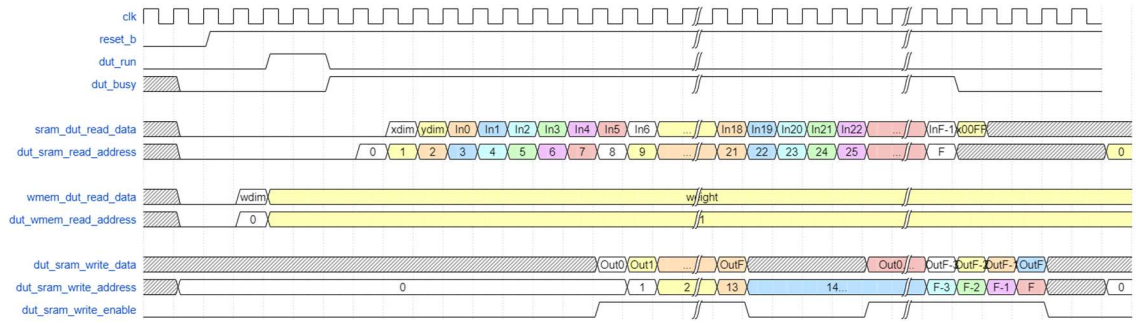


Figure 3: Interface Timing Diagram

4. Technical Implementation

The whole module is contained within the top_without_mem.v

4.1 Hierarchy

1. MyDesign – Top Module
    - 1.1.1. Controller – FSM controller
    - 1.1.2. Convolution Calculator – Pipelines inputs and has combination output logic
        - 1.1.2.1.1. Select output Padding – Left 0 pads output
        - 1.1.2.1.2. Input Registers – Pipelines the inputs
        - 1.1.2.1.3. XNOR Bitcount Add Compare (x14) – Calculates sign of output

1.1.3. Weight Reg – Holds the weight matrix
1.1.4. Input Dim Reg – Holds the Input dimensions
1.1.5. Read Input Interface – Flip Flop interfaces for input SRAM
1.1.6. Weight Input Interface – Flip Flop interfaces for weight SRAM
1.1.7. Output Interface – Flip Flop interface for output SRAM

This hierarchy allowed the bulk of Datapath to be within its own module while the submodules simplified the amount of HDL that had to be written.

5. Verification

Verification was done through the provided test bench. The inputs and the outputs were placed in folders for each run along with the expected output vectors. The test bench asserts the run signal and then compares the output to the expected output vectors for the given input.

6. Results Achieved

The following table is a list of the results achieved.

Metric	Results
Area	1600 um^2
Clock Speed	210526315 hz
Delay for 3 matrixes (12.667 avg ydim)	49 Cycles
Delay	232.75ns
1/(delay.area)	2.685e-6 ($ns^{-1}um^{-2}$)

7. Conclusions

The hardware is able to operate with a high throughput utilizing the maximum possible bandwidth of the input memory. By doing this in combination with a high clock rate allows the module to convolute the input matrixes efficiently with an average of 77.583ns per matrix in testing with a total cell area of 1600um^2.

8. Appendix

8.1 Module Interface

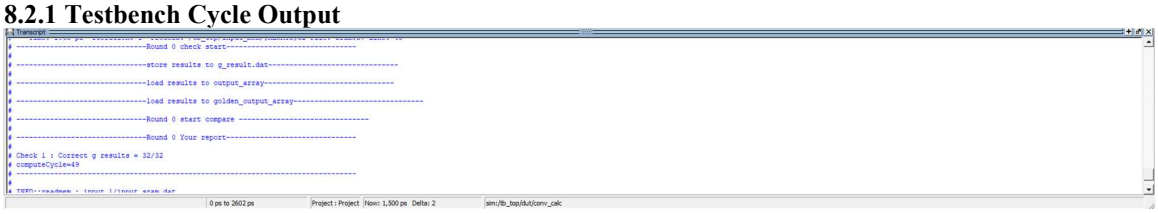
```
module MyDesign(
  input dut_run,
  output dut_busy,
  input reset_b,
  input clk,

  output [11:0]dut_sram_write_address,
  output [15:0]dut_sram_write_data,
  output dut_sram_write_enable,

  output [11:0]dut_sram_read_address,
  input [15:0]sram_dut_read_data,

  output [11:0]dut_wmem_read_address,
  input [15:0]wmem_dut_read_data);
```

8.2.1 Testbench Cycle Output



8.3 Timing Reports

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : MyDesign
Version: P-2019.03-SP1
Date   : Thu Oct 28 14:40:09 2021
****************************************

Operating Conditions: slow   Library:
NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm
Wire Load Model Mode: top

  Startpoint: controller/input_addr_offset_reg[1]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: controller/base_input_addr_reg[5]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Point                                                   Incr       Path
  ---------------------------------------------------------------------------
  clock clk (rise edge)                                 0.0000     0.0000
  clock network delay (ideal)                           0.0000     0.0000
  controller/input_addr_offset_reg[1]/CK (DFF_X2)       0.0000     0.0000 r
  controller/input_addr_offset_reg[1]/QN (DFF_X2)       0.4840     0.4840 r
  U1033/ZN (NOR2_X2)                                    0.0880     0.5720 f
  U648/ZN (INV_X1)                                      0.1081     0.6801 r
  U423/ZN (OAI21_X1)                                    0.1472     0.8273 f
  U1003/ZN (AOI222_X1)                                  1.1085     1.9358 r
  U1004/ZN (AOI222_X1)                                  0.4037     2.3395 f
  U1005/ZN (AOI222_X2)                                  0.9236     3.2631 r
  U1006/ZN (AOI222_X1)                                  0.3176     3.5806 f
  U655/ZN (XNOR2_X1)                                    0.3713     3.9519 f
  U428/ZN (OAI22_X1)                                    0.4462     4.3982 r
  controller/base_input_addr_reg[5]/D (DFF_X1)          0.0000     4.3982 r
  data arrival time                                                4.3982

  clock clk (rise edge)                                 4.7500     4.7500
  clock network delay (ideal)                           0.0000     4.7500
  clock uncertainty                                    -0.0500     4.7000
  controller/base_input_addr_reg[5]/CK (DFF_X1)         0.0000     4.7000 r
  library setup time                                   -0.2964     4.4036
  data required time                                               4.4036
  ---------------------------------------------------------------------------
  data required time                                               4.4036
  data arrival time                                               -4.3982
  ---------------------------------------------------------------------------
  slack (MET)                                                      0.0054
```

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay min
        -max_paths 1
Design : MyDesign
Version: P-2019.03-SP1
Date   : Thu Oct 28 14:40:09 2021
****************************************

Operating Conditions: fast   Library:
NangateOpenCellLibrary_PDKv1_2_v2008_10_fast_nldm
Wire Load Model Mode: top

  Startpoint: controller/output_addr_offset_reg[0]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: controller/output_addr_offset_reg[0]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: min

  Point                                                   Incr       Path
  --------------------------------------------------------------------------
  clock clk (rise edge)                                 0.0000     0.0000
  clock network delay (ideal)                           0.0000     0.0000
  controller/output_addr_offset_reg[0]/CK (DFF_X1)      0.0000     0.0000 r
  controller/output_addr_offset_reg[0]/Q (DFF_X1)       0.0574     0.0574 r
  U491/ZN (AOI22_X1)                                    0.0184     0.0758 f
  controller/output_addr_offset_reg[0]/D (DFF_X1)       0.0000     0.0758 f
  data arrival time                                                0.0758

  clock clk (rise edge)                                 0.0000     0.0000
  clock network delay (ideal)                           0.0000     0.0000
  clock uncertainty                                     0.0500     0.0500
  controller/output_addr_offset_reg[0]/CK (DFF_X1)      0.0000     0.0500 r
  library hold time                                     0.0004     0.0504
  data required time                                               0.0504
  --------------------------------------------------------------------------
  data required time                                               0.0504
  data arrival time                                               -0.0758
  --------------------------------------------------------------------------
  slack (MET)                                                      0.0255
```

8.4 Area Report

```
 ****************************************
Report : cell
Design : MyDesign
Version: P-2019.03-SP1
Date   : Thu Oct 28 14:40:09 2021
****************************************

Attributes:
    b - black box (unknown)
    h - hierarchical
   mo - map_only
```

```
    n - noncombinational
    r - removable
    u - contains unmapped logic

Cell                      Reference       Library             Area Attributes
… (Omitted for conciseness)
--------------------------------------------------------------------------------
-
Total 745 cells                                                1600.7880
```

8.5 Timing Diagram Script

//Generated Using https://wavedrom.com/

```
{signal: [
  {name: 'clk', wave: 'p.................|......|......'},
  {name:'reset_b', wave:'0.1.............|......|......'},
  {name:'dut_run', wave:'0...1.0.........|......|......'},
  {name:'dut_busy', wave:'x0....1.........|......|.0....'},
  {},
  {name:'sram_dut_read_data', wave:'x0......2345678923|456789|23x....',
data:['xdim', 'ydim', 'In0', 'In1', 'In2', 'In3', 'In4', 'In5','In6',
'...','In18','In19','In20','In21','In22','...','InF-1','x00FF']},
  {name:'dut_sram_read_address', wave:'x0.....23456789234|456789|2x....3',
data:['0','1', '2', '3', '4', '5', '6', '7','8','9',
'...','21','22','23','24','25','...', 'F','0']},
  {},
  {name:'wmem_dut_read_data', wave:'x0.23............|......|.......',
data:['wdim', 'weight']},
  {name:'dut_wmem_read_address', wave:'x0.23............|......|.......',
data:['0','1']},
  {},
  {name:'dut_sram_write_data', wave:'x.............234|4x...9|2345x..',
data:['Out0', 'Out1', '...','OutF', 'Out0....','OutF-3','OutF-2', 'OutF-1',
'OutF']},
  {name:'dut_sram_write_address',
wave:'x2.............23|45....|6789x.2',data:['0', '1', '2','13','14...','F-
3', 'F-2', 'F-1', 'F','0']},
  {name:'dut_sram_write_enable', wave:'0.............1..|.0...1|....0..'},
]}
```

8.6 Design Code

```
module MyDesign(
  input dut_run,
  output dut_busy,
  input reset_b,
  input clk,

  output [11:0]dut_sram_write_address,
  output [15:0]dut_sram_write_data,
  output dut_sram_write_enable,

  output [11:0]dut_sram_read_address,
  input [15:0]sram_dut_read_data,

  output [11:0]dut_wmem_read_address,
  input [15:0]wmem_dut_read_data);

  //dimension values and weight matrix
  wire [4:0] xdim, ydim;
  wire [15:0]weight;
  //loading dimensions from the controller
  wire [1:0]input_dim_load_ctrl;
```

```
  //addr and write enable from controller for sram interfaces
  wire [11:0]fsm_weight_addr, fsm_input_addr, fsm_output_addr;
  wire fsm_w_enable;
  //Data from interface outputs to device
  wire [15:0]weight_input_data, input_input_data;
  //Whole row output calculation to be sent to sram interface
  wire [15:0]calculation_output;


  //CONTROLLER
  //FSM controller sets addresses and load controls
  fms_controller controller(.clock(clk), .reset(reset_b), .go(dut_run),
    .ydim(ydim),
    .input_data(input_input_data),
    .busy(dut_busy),
    .input_load_ctrl(input_dim_load_ctrl),
    .input_addr(fsm_input_addr),
    .weight_addr(fsm_weight_addr),
    .output_addr(fsm_output_addr),
    .output_w_enable(fsm_w_enable));

//-------------------------------------------------------------------------------
-
  //CONVOLUTION module
  //does the calculations from the previous 3 inputs and left padds the output
  convolution conv_calc(.clock(clk),
    .reset(reset_b),
    .xdim(xdim),
    .weight(weight),
    .input_data(input_input_data),
    .padded_out(calculation_output));

//-------------------------------------------------------------------------------
-
  //WEIGHT AND DIM REGISTERS
  //holds the weight and dimensions after they have been loaded from SRAM
  input_weight_registers weight_reg(.clock(clk),
    .reset(reset_b),
    .input_sram_data(weight_input_data),
    .weight_out(weight));
  input_dim_registers dim_reg(.clock(clk),
    .reset(reset_b),
    .input_sram_data(input_input_data),
    .input_load_ctrl(input_dim_load_ctrl),
    .xdim_reg(xdim),
    .ydim_reg(ydim));


//-------------------------------------------------------------------------------
-
  //READ INTERFACE
  //pipelines inputs and outputs to the SRAM
  sram_read_interface read_input(.clock(clk),
    .reset(reset_b),
    .sram_interface_data(sram_dut_read_data),
    .dev_interface_read_addr(fsm_input_addr),
    .interface_sram_read_addr(dut_sram_read_address),
    .interface_dev_data_out(input_input_data));
  sram_read_interface read_weight(.clock(clk),
    .reset(reset_b),
    .sram_interface_data(wmem_dut_read_data),
    .dev_interface_read_addr(fsm_weight_addr),
    .interface_sram_read_addr(dut_wmem_read_address),
```

```
    .interface_dev_data_out(weight_input_data));

//-------------------------------------------------------------------------------
-
  //WRITE INTERFACE
  //Pipelines the outputs to the SRAM
  sram_write_interface write_output(.clock(clk), .reset(reset_b),
    .dev_interface_write_addr(fsm_output_addr),
    .dev_interface_write_data(calculation_output),
    .dev_interface_r_enable(fsm_w_enable),
    .interface_sram_write_addr(dut_sram_write_address),
    .interface_sram_write_data(dut_sram_write_data),
    .interface_sram_r_enable(dut_sram_write_enable)
    );

endmodule

//stores the weight
module input_weight_registers(
  input clock,
  input reset,
  input [15:0]input_sram_data,
  output [15:0]weight_out
  );

  reg [8:0]weight;
  assign weight_out = {6'b0,weight};


  always @ ( posedge clock ) begin
    if(~reset)
      weight<=0;
    else
      weight<=input_sram_data;
  end

endmodule

// This module contains the registers for x and y dim
// It handels when to update them from input_select_ctrl
// 00,10 -> dont update
// 01 -> save input to xdim_reg
// 11 -> save input to ydim_reg
module input_dim_registers(
  input clock,
  input reset,
  input [15:0]input_sram_data,
  input [1:0]input_load_ctrl,
  output reg [4:0] xdim_reg,
  output reg [4:0] ydim_reg
  );
  //next values to be loaded in
  reg [4:0] xdim_reg_next;
  reg [4:0] ydim_reg_next;


  //clear registers on sync reset
  always@(posedge clock)begin
    if(~reset)begin
      xdim_reg <= 0;
      ydim_reg <= 0;
    end
    else begin
```

```
      xdim_reg <= xdim_reg_next;
      ydim_reg <= ydim_reg_next;
    end
  end

  always@(*)begin
    //hold value by default
    xdim_reg_next = xdim_reg;
    ydim_reg_next = ydim_reg;

    //otherwise load new value
    casex(input_load_ctrl)
      2'b01:  xdim_reg_next = input_sram_data[4:0];
      2'b11:  ydim_reg_next = input_sram_data[4:0];
    endcase
  end

endmodule


//the inputs to the ram through a flip flop
module sram_read_interface(
  input clock,
  input reset,
  input [15:0]sram_interface_data,           //from mem
  input [11:0]dev_interface_read_addr,      //from ctrl
  output reg [11:0]interface_sram_read_addr, //to mem
  output reg [15:0]interface_dev_data_out  //to inputs
  );

  //inputs and ouputs pass through a flip flop
  always @ ( posedge clock ) begin
    if(~reset)begin
      interface_sram_read_addr <= 0;
      interface_dev_data_out <=0;
    end
    else begin
      interface_sram_read_addr <= dev_interface_read_addr; //data from mem into
logic
      interface_dev_data_out <= sram_interface_data; //addr from ctrl into mem
    end

  end


endmodule

//the output to the ram through a flip flop
module sram_write_interface(
  input clock,
  input reset,

  input [11:0]dev_interface_write_addr,//from ctrl
  input [15:0]dev_interface_write_data,      //from ctrl
  input dev_interface_r_enable,             //from ctrl

  output reg [11:0]interface_sram_write_addr, //to mem
  output reg [15:0]interface_sram_write_data, //to mem
  output reg interface_sram_r_enable);

  //inputs and ouputs pass through a flip flop
  always @ ( posedge clock ) begin
    if(~reset)begin
```

```
      interface_sram_write_addr <= 0;
      interface_sram_write_data <=0;
      interface_sram_r_enable <=0;
    end
    else begin
      interface_sram_write_addr <= dev_interface_write_addr;
      interface_sram_write_data <=dev_interface_write_data;
      interface_sram_r_enable <=dev_interface_r_enable;
    end

  end


endmodule

//Captures the inputs and piplines them to compute the convolution with the
weight
module convolution(
  input clock,
  input reset,
  input [4:0]xdim,
  input [15:0]weight,
  input [15:0]input_data,
  output [15:0]padded_out);


wire [15:0]reg_1, reg_2, reg_3;
wire [13:0]output_raw;

//padd the ouput
output_select_padding
padded_output(.xnor_bitcnt_add_cmp_output(output_raw),.xdim(xdim),.padded_outpu
t(padded_out));

//clocks the data through the registers to shift them to get the proper outputs
input_registers input_storage_regs(.clock(clock),
  .reset(reset),
  .data_input(input_data),
  .reg_1(reg_1),
  .reg_2(reg_2),
  .reg_3(reg_3));

//output sign determination in parallel for each row
xnor_bitcnt_add_cmp output_bit_0(.weight_row_1(weight[2:0]),
.weight_row_2(weight[5:3]),
.weight_row_3(weight[8:6]),.input_reg_1_bits(reg_1[2:0]),.input_reg_2_bits(reg_
2[2:0]),.input_reg_3_bits(reg_3[2:0]),.sign_output(output_raw[0]));
xnor_bitcnt_add_cmp
output_bit_1(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),
.weight_row_3(weight[8:6]),.input_reg_1_bits(reg_1[3:1]),.input_reg_2_bits(reg_
2[3:1]),.input_reg_3_bits(reg_3[3:1]),.sign_output(output_raw[1]));
xnor_bitcnt_add_cmp
output_bit_2(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),
.weight_row_3(weight[8:6]),.input_reg_1_bits(reg_1[4:2]),.input_reg_2_bits(reg_
2[4:2]),.input_reg_3_bits(reg_3[4:2]),.sign_output(output_raw[2]));
xnor_bitcnt_add_cmp
output_bit_3(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),
.weight_row_3(weight[8:6]),.input_reg_1_bits(reg_1[5:3]),.input_reg_2_bits(reg_
2[5:3]),.input_reg_3_bits(reg_3[5:3]),.sign_output(output_raw[3]));
xnor_bitcnt_add_cmp
output_bit_4(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),
.weight_row_3(weight[8:6]),.input_reg_1_bits(reg_1[6:4]),.input_reg_2_bits(reg_
2[6:4]),.input_reg_3_bits(reg_3[6:4]),.sign_output(output_raw[4]));
```

```
xnor_bitcnt_add_cmp
output_bit_5(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),
.weight_row_3(weight[8:6]),.input_reg_1_bits(reg_1[7:5]),.input_reg_2_bits(reg_
2[7:5]),.input_reg_3_bits(reg_3[7:5]),.sign_output(output_raw[5]));
xnor_bitcnt_add_cmp
output_bit_6(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),
.weight_row_3(weight[8:6]),.input_reg_1_bits(reg_1[8:6]),.input_reg_2_bits(reg_
2[8:6]),.input_reg_3_bits(reg_3[8:6]),.sign_output(output_raw[6]));
xnor_bitcnt_add_cmp
output_bit_7(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),
.weight_row_3(weight[8:6]),.input_reg_1_bits(reg_1[9:7]),.input_reg_2_bits(reg_
2[9:7]),.input_reg_3_bits(reg_3[9:7]),.sign_output(output_raw[7]));
xnor_bitcnt_add_cmp
output_bit_8(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),
.weight_row_3(weight[8:6]),.input_reg_1_bits(reg_1[10:8]),.input_reg_2_bits(reg
_2[10:8]),.input_reg_3_bits(reg_3[10:8]),.sign_output(output_raw[8]));
xnor_bitcnt_add_cmp
output_bit_9(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),
.weight_row_3(weight[8:6]),.input_reg_1_bits(reg_1[11:9]),.input_reg_2_bits(reg
_2[11:9]),.input_reg_3_bits(reg_3[11:9]),.sign_output(output_raw[9]));
xnor_bitcnt_add_cmp
output_bit_10(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),.weight_row
_3(weight[8:6]),.input_reg_1_bits(reg_1[12:10]),.input_reg_2_bits(reg_2[12:10])
,.input_reg_3_bits(reg_3[12:10]),.sign_output(output_raw[10]));
xnor_bitcnt_add_cmp
output_bit_11(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),.weight_row
_3(weight[8:6]),.input_reg_1_bits(reg_1[13:11]),.input_reg_2_bits(reg_2[13:11])
,.input_reg_3_bits(reg_3[13:11]),.sign_output(output_raw[11]));
xnor_bitcnt_add_cmp
output_bit_12(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),.weight_row
_3(weight[8:6]),.input_reg_1_bits(reg_1[14:12]),.input_reg_2_bits(reg_2[14:12])
,.input_reg_3_bits(reg_3[14:12]),.sign_output(output_raw[12]));
xnor_bitcnt_add_cmp
output_bit_13(.weight_row_1(weight[2:0]),.weight_row_2(weight[5:3]),.weight_row
_3(weight[8:6]),.input_reg_1_bits(reg_1[15:13]),.input_reg_2_bits(reg_2[15:13])
,.input_reg_3_bits(reg_3[15:13]),.sign_output(output_raw[13]));

endmodule



//shifts the input from reg_3 -> reg_1
module input_registers(
  input clock,
  input reset,
  input [15:0]data_input,
  output reg [15:0]reg_1,
  output reg [15:0]reg_2,
  output reg [15:0]reg_3);

  //shifting the registers from top to bottom handles the vertical shifting
  always @ ( posedge clock ) begin
    if(~reset)begin
      reg_1 <= 0;
      reg_2 <= 0;
      reg_3 <= 0;
    end
    else begin
      reg_1 <= reg_2;
      reg_2 <= reg_3;
      reg_3 <= data_input;
    end
```

```
  end

endmodule

//selects the output padding for each weight
module output_select_padding(
  input [13:0]xnor_bitcnt_add_cmp_output,
  input [4:0]xdim,
  output reg [15:0] padded_output
  );

  //combinational logic to properly left padd the output with zeros
  always @ ( * ) begin
  padded_output = 0;
    casex(xdim)
      5'd16: padded_output[15:0] = {2'b0,xnor_bitcnt_add_cmp_output[13:0]};
      5'd12: padded_output[15:0] = {6'b0,xnor_bitcnt_add_cmp_output[9:0]};
      5'd10: padded_output[15:0] = {8'b0,xnor_bitcnt_add_cmp_output[7:0]};
      default: padded_output = 16'bx;
    endcase
  end

endmodule

// takes 4 x 4bit inputs and calculates the sign bit from xnor and weights for
variable inputs
module xnor_bitcnt_add_cmp(
  input [2:0]weight_row_1,
  input [2:0]weight_row_2,
  input [2:0]weight_row_3,
  input [2:0]input_reg_1_bits,
  input [2:0]input_reg_2_bits,
  input [2:0]input_reg_3_bits,
  output sign_output);

  wire [3:0] xnor_out_1, xnor_out_2,xnor_out_3;
  wire [3:0] bitcount_out;




  //compute the xnors
  assign xnor_out_1 = ~(weight_row_1^input_reg_1_bits);
  assign xnor_out_2 = ~(weight_row_2^input_reg_2_bits);
  assign xnor_out_3 = ~(weight_row_3^input_reg_3_bits);

  //use the xnor_out_3[2] as the cary in
  assign bitcount_out = ((xnor_out_1[0] + xnor_out_1[1]) + (xnor_out_1[2] +
xnor_out_2[0]))
                        + ((xnor_out_2[1] + xnor_out_2[2])+ (xnor_out_3[0] +
xnor_out_3[1]))
                        + xnor_out_3[2];

  //calculate the sign of the bitcount
  assign sign_output = bitcount_out > 4'b0100;


endmodule

module fms_controller(
  input clock,
  input reset,
  input go,
```

```
  input [4:0]ydim,
  input [15:0]input_data,
  output reg busy,
  output reg [1:0]input_load_ctrl,
  output [11:0]input_addr,
  output [11:0]weight_addr,
  output [11:0]output_addr,
  output reg output_w_enable);

parameter S0 = 3'd0,
          S1 = 3'd1,
          S2 = 3'd2,
          S3 = 3'd3,
          S4 = 3'd4,
          S5 = 3'd5,
          S6 = 3'd6;

reg [3:0] state, next_state;

reg [4:0] input_addr_offset, output_addr_offset;
reg [4:0] next_input_addr_offset, next_output_addr_offset;

reg [6:0]base_input_addr, base_ouput_addr;
reg [6:0]next_base_input_addr, next_base_ouput_addr;

// assigns the next output as current base + offset
assign input_addr = base_input_addr + input_addr_offset;
assign output_addr = base_ouput_addr + output_addr_offset;
assign weight_addr = 1;

//create the flip flops for stored values
always @ ( posedge clock ) begin
  if(~reset)begin
    state <= S0;
    input_addr_offset <= 0;
    output_addr_offset <= 0;
    base_input_addr <= 0;
    base_ouput_addr <= 0;
  end
  else begin
    state <= next_state;
    input_addr_offset <= next_input_addr_offset;
    output_addr_offset <= next_output_addr_offset;
    base_input_addr <= next_base_input_addr;
    base_ouput_addr <= next_base_ouput_addr;
  end
end

always @ ( * ) begin
  //default state
  next_state = S0;
  //busy unless in wait state
  busy = 1;
  //do not load x or y dim
  input_load_ctrl = 0;
  //output only in certain state
  output_w_enable = 0;
  //hold values unless otherwise set
  next_input_addr_offset = input_addr_offset;
  next_output_addr_offset = output_addr_offset;
  next_base_input_addr = base_input_addr;
  next_base_ouput_addr = base_ouput_addr;
```

```
  case(state)
    S0: begin
      //reset the offset registers
      next_input_addr_offset = 0;
      next_output_addr_offset = 0;
      next_base_input_addr = 0;
      next_base_ouput_addr = 0;
      busy = 0;
      //change state
      if(go)
        next_state= S1;
      else
        next_state= S0;
    end

    S1:begin
      //inc offset
      next_input_addr_offset = input_addr_offset + 1'b1;

      //detect cycle before dimensions will appear in input
      if(input_addr_offset == 2)
        next_state = S2;
      else
        next_state = S1;

    end

    S2: begin
      //inc offset
      next_input_addr_offset = input_addr_offset + 1'b1;

      //load the xdim on next clock
      input_load_ctrl = 2'b01;
      next_state = S3;

    end

    S3: begin
      //inc input offset update the bases and reset next offsets to zero
      //this allows mutiple inputs to be processed as the cycle is a loop from
this point
      next_base_input_addr = next_base_input_addr + input_addr_offset + 1'b1;
      next_base_ouput_addr = next_base_ouput_addr + output_addr_offset;
      next_input_addr_offset = 0;
      next_output_addr_offset = 0;
      //load the ydim on next clock
      input_load_ctrl = 2'b11;
      next_state = S4;
    end

    S4:begin
      //inc offset
      next_input_addr_offset = input_addr_offset + 1'b1;
      //detect that outputs will be ready on next clock cycle
      if(input_addr_offset == 2)
        next_state = S5;
      else
        next_state = S4;
    end

    S5:begin
      //incr input and outputs adress and enable the output
      next_input_addr_offset = input_addr_offset + 1'b1;
```

```
      next_output_addr_offset = output_addr_offset + 1'b1;
      output_w_enable = 1;

      //detect when next dim are ready to be loaded
      if(input_addr_offset == ydim-1'd1)
        next_state = S6;
      else
        next_state = S5;

    end

    S6:begin
      //incr inputs and outputs are still processing
      next_input_addr_offset = input_addr_offset + 1'b1;
      next_output_addr_offset = output_addr_offset + 1'b1;
      output_w_enable = 1;
      //load the next dimension
      input_load_ctrl = 2'b01;
      next_state = S3;

      //detect end of data stream
      if(input_data == 16'h00ff)
        next_state = S0;

    end


  endcase
end

endmodule
```

8.7 SRAM Code

```
//`timescale 1ns/10ps

module sram     #(parameter  ADDR_WIDTH      = 32  ,
                  parameter  DATA_WIDTH      = 16  ,
                  parameter  MEM_INIT_FILE   = ""  )
                (
                //------------------------------------------------------------
--
                //
                input  wire [ADDR_WIDTH-1:0  ]  write_address ,
                input  wire [DATA_WIDTH-1:0  ]  write_data    ,
                input  wire [ADDR_WIDTH-1:0  ]  read_address  ,
                output reg  [DATA_WIDTH-1:0  ]  read_data     ,
                input  wire                                   write_enable
,
                           input  reset ,
                input  clock
                );


    //-------------------------------------------------------
    // Associative memory

    bit  [DATA_WIDTH-1 :0  ]     mem     [int]  ;

    //-------------------------------------------------------
    // RAW and X condition
```

```
    reg  [ADDR_WIDTH-1:0   ]     last_write_addr;
    reg                          last_write_en;
    //-------------------------------------------------------
    // Read
    always @(posedge clock)
      begin
         if(!reset) // Active low reset
              begin
                     for(int i = 0 ; i<mem.size(); i++)
                           mem[i] = 'hx;
              end
        else if ( ^read_address === 1'bx || ( (last_write_addr) ==
(read_address) && last_write_en == 1'b1) )
          read_data <= 16'bx;

        else
          read_data <= mem [read_address];
      end

    //-------------------------------------------------------
    // Write

    always @(posedge clock)
      begin
         if(reset) // Active low reset
              begin
                     last_write_addr <= write_address;
                     last_write_en <= write_enable;

                     if (write_enable)
                           mem [write_address] = write_data;
        end
      end
    //-------------------------------------------------------


    //-------------------------------------------------------
       //Loading inputs and weights to sram
    string entry  ;
    int fileDesc ;
    bit [ADDR_WIDTH-1 :0 ]  memory_address ;
    bit [DATA_WIDTH-1 :0 ]  memory_data    ;



    task  loadInitFile(input string memFile);
      if (memFile != "")
        begin
          fileDesc = $fopen (memFile, "r");
          if (fileDesc == 0)
            begin
              $display("ERROR::readmem file error : %s ", memFile);
              $finish;
            end
          $display("INFO::readmem : %s ", memFile);
          while (!$feof(fileDesc))
            begin
              void'($fgets(entry, fileDesc));
              void'($sscanf(entry, "@%x %b", memory_address, memory_data));
              //$display("INFO::readmem file contents : %s  : Addr:%h,
Data:%h", memFile, memory_address, memory_data);
              mem[memory_address] = memory_data ;
```

```
            end
          $fclose(fileDesc);
        end
      endtask

endmodule
```

8.8 Test bench Code

```
`include "sram.sv"
module tb_top();


  parameter CLK_PHASE=5;
  parameter ROUND=1;
  //parameter ADDR_464=12'h0d;
  parameter ADDR_564=12'h01f;
  //parameter num_results_464=14;
  parameter num_results_564=32;

  time computeCycle[ROUND];
  event computeStart[ROUND];
  event computeEnd[ROUND];
  event checkFinish[ROUND];
  time startTime[ROUND];
  time endTime[ROUND];

  int correctResult[ROUND];
  reg [15:0] result_array[int];
  reg [15:0] golden_result_array[int];
  int i;
  int j;
  int k;
  int q;
  int p;
  //------------------------------------------------------------------------
  // General
  //
  reg                                   clk            ;
  reg                                   reset_b        ;
  reg                                   dut_run        ;
  wire                                  dut_busy       ;

  //------------------------------------------------------------------------
  //----------------------output sram-------------------------------------------
---
  wire                                     dut_sram_write_enable  ;
  wire [11:0]                              dut_sram_write_address ;
  wire [15:0]                              dut_sram_write_data    ;
  //------------------------input sram----------------------------------------
--
  wire [11:0]                              dut_sram_read_address  ;
  wire [15:0]                              sram_dut_read_data     ;
  //------------------------weights ----------------------------------------
  wire [11:0]                              dut_wmem_read_address    ;
  wire [15:0]                              wmem_dut_read_data       ;  // read
data

  //------------------------------------------------------------------------
  //------------------------------------------------------------------------
  //------------------------------------------------------------------------
  //SRAM
```

```
  //sram for inputs
  sram  #(.ADDR_WIDTH    (12),
          .DATA_WIDTH    (16),
          .MEM_INIT_FILE ("input_sram.dat"        ))
          input_mem  (
          .write_enable ( 1'b0  ),
          .write_address( 12'b0 ),
          .write_data   ( 16'b0    ),
          .read_address ( dut_sram_read_address  ),
          .read_data    ( sram_dut_read_data     ),
         .reset        ( reset_b                     ),
          .clock        ( clk                    )
         );

  //sram for weights
  sram  #(.ADDR_WIDTH    (12),
          .DATA_WIDTH    (16),
          .MEM_INIT_FILE ("weight_sram.dat"           ))
          weight_mem  (
          .write_enable ( 1'b0                     ),
          .write_address( 12'b0                    ),
          .write_data   ( 16'b0                    ),
          .read_address ( dut_wmem_read_address  ),
          .read_data    ( wmem_dut_read_data     ),
         .reset        ( reset_b                     ),
          .clock        ( clk                    )
         );

  //sram for outputs
  sram  #(.ADDR_WIDTH    (12),
          .DATA_WIDTH    (16),
          .MEM_INIT_FILE ("output_sram.dat"        ))
          output_mem  (
          .write_enable ( dut_sram_write_enable  ),
          .write_address( dut_sram_write_address ),
          .write_data   ( dut_sram_write_data    ),
          .read_address ( 12'b0  ),
          .read_data    (      ),
         .reset        ( reset_b                     ),
          .clock        ( clk                    )
         );


//--------------------------------------------------------------------------
// DUT
//--------------------------------------------------------------------------
  MyDesign dut(
//--------------------------------------------------------------------------
//Control signals
          .dut_run             ( dut_run              ),
          .dut_busy            ( dut_busy             ),
          .reset_b             ( reset_b                    ),
          .clk                 ( clk                      ),

//--------------------------------------------------------------------------
//input and output SRAM interface
          .dut_sram_write_address  ( dut_sram_write_address    ),
          .dut_sram_write_data     ( dut_sram_write_data       ),
          .dut_sram_write_enable   ( dut_sram_write_enable     ),
          .dut_sram_read_address   ( dut_sram_read_address     ),
          .sram_dut_read_data      ( sram_dut_read_data        ),
//--------------------------------------------------------------------------
//weights SRAM interface
```

```
          .dut_wmem_read_address    ( dut_wmem_read_address    ),
          .wmem_dut_read_data       ( wmem_dut_read_data       )
         );


  //--------------------------------------------------------------------------
  //  clk
  initial
    begin
        clk                     = 1'b0;
        forever # CLK_PHASE clk = ~clk;
    end


  //--------------------------------------------------------------------------
  //--------------------------------------------------------------------------
  // Stimulus
  initial begin
       $display("--------------------------------start_simulation----------------
---------------\n");
      repeat(25) @(posedge clk);
      reset_b=0;
      dut_run=0;
      repeat(25) @(posedge clk);
      reset_b=1;
      for(j=0;j<ROUND;j=j+1) begin
         if(j!=0) wait(checkFinish[j-1]);
         input_mem.loadInitFile($sformatf("input_%0d/input_sram.dat",j));
         weight_mem.loadInitFile($sformatf("input_%0d/weight_sram.dat",j));

             repeat(5) @(posedge clk);
             wait(dut_busy==0);
         @(posedge clk);
         dut_run=1; // DUT starts computing
         ->computeStart[j];
         $display("--------------------------------Round %0d start--------------
-----------------\n",j);
         wait(dut_busy==1);
         @(posedge clk);
         dut_run=0;
         wait(dut_busy==0);
         ->computeEnd[j];
      end
  end

  //--------------------------------------------------------------------------
  //--------------------------------------------------------------------------
  // Timer
  //
  initial begin
    for(k=0;k<ROUND;k=k+1) begin
       wait(computeStart[k]);
       startTime[k]=$time;
       wait(computeEnd[k]);
       endTime[k]=$time;
       computeCycle[k]=endTime[k]-startTime[k];
     end
  end

  //--------------------------------------------------------------------------
  // Result collector
  // Collect your compute results
```

```
  initial begin
    for(q=0;q<ROUND;q=q+1) begin
       wait(computeEnd[q]);
       repeat(10) @(posedge clk);
       $display("--------------------------------Round %0d check start----------
---------------------\n",q);
       $display("--------------------------------store results to g_result.dat--
----------------------------\n");

       if (q==0)

       //$writememb($sformatf("input_%0d/result.dat",q),output_mem.mem,12'h000,A
DDR_464);

       $writememb($sformatf("input_%0d/result.dat",q),output_mem.mem,12'h000,ADD
R_564);
       //else
       //
       $writememb($sformatf("input_%0d/result.dat",q),output_mem.mem,12'h000,ADD
R_ONE);


//$writememb($sformatf("input_%0d/result.dat",q),output_mem.mem,12'h000,12'h08f
);
          repeat(10) @(posedge clk);

       //-------------------------------------------------------------------------
----
       //-------------------------------------------------------------------------
----
       // Result comparator
       // Compare your compute results with the results computed by Python
script
       $display("--------------------------------load results to output_array---
----------------------------\n");
       $readmemb($sformatf("input_%0d/result.dat",q),result_array);

       $display("--------------------------------load results to
golden_output_array--------------------------------\n");

$readmemb($sformatf("input_%0d/golden_outputs.dat",q),golden_result_array);

          $display("--------------------------------Round %0d start compare -----
-------------------------\n",q);
            //if(q==0)begin
              //     for(i=0;i<num_results_464;i=i+1) begin
              //           if(result_array[i]==golden_result_array[i])
correctResult[q]=correctResult[q]+1;
              //     end
            //end

       if(q==0)begin
                     for(i=0;i<num_results_564;i=i+1) begin
                     if(result_array[i]==golden_result_array[i])
correctResult[q]=correctResult[q]+1;
              end
       end
       //else begin
       //     for(i=0;i<num_results_one;i=i+1) begin
         //         if(result_array[i]==golden_result_array[i])
correctResult[q]=correctResult[q]+1;
        //    end
       //end
```

```
         $display("------------------------------Round %0d Your report--------
----------------------\n",q);
       if(q==0)
                    //$display("Check 1 : Correct g results =
%0d/%0d",correctResult[q],num_results_464);
             $display("Check 1 : Correct g results =
%0d/%0d",correctResult[q],num_results_564);

       //else
       //     $display("Check 1 : Correct g results =
%0d/%0d",correctResult[q],num_results_one);

       $display("computeCycle=%0d",computeCycle[q]/(2*CLK_PHASE));
       $display("------------------------------------------------------------
-------------------\n");
       @(posedge clk);
       ->checkFinish[q];
     end
     $finish;
        end

endmodule
```

8.9 Synthesis Script

```
# setup name of the clock in your design.
set clkname clk

# set variable "modname" to the name of topmost module in design
set modname MyDesign

# set variable "RTL_DIR" to the HDL directory w.r.t synthesis directory
set RTL_DIR    ../v

# set variable "type" to a name that distinguishes this synthesis run
set type tut1

#set the number of digits to be used for delay results
set report_default_significant_digits 4

set CLK_PER 10
#-------------------------------------------------------------
#
# Basic Synthesis Script (TCL format)
#
# Revision History
#   1/15/03  : Author Shane T. Gehring - from class example
#   2/09/07  : Author Zhengtao Yu      - from class example
#   12/14/07 : Author Ravi Jenkal      - updated to 180 nm & tcl
#   10/7/20  : P Franzon - Project specific script
#
#-------------------------------------------------------------
#-----------------------------------------------------------
# Read in Verilog file and map (synthesize) onto a generic
# library.
# MAKE SURE THAT YOU CORRECT ALL WARNINGS THAT APPEAR
# during the execution of the read command are fixed
# or understood to have no impact.
# ALSO CHECK your latch/flip-flop list for unintended
# latches
#-----------------------------------------------------------
```

```
read_verilog $RTL_DIR/top_without_mem.v


#-------------------------------------------------------
# Our first Optimization 'compile' is intended to
# produce a design that will meet hold-time
# under worst-case conditions:
#            - slowest process corner
#            - highest operating temperature and lowest Vcc
#            - expected worst case clock skew
#-------------------------------------------------------
#-------------------------------------------------------
# Set the current design to the top level instance name
# to make sure that you are working on the right design
# at the time of constraint setting and compilation
#-------------------------------------------------------

 current_design $modname

#-------------------------------------------------------
# Set the synthetic library variable to enable use of
# desigware blocks
#-------------------------------------------------------
 set synthetic_library [list dw_foundation.sldb]

#-------------------------------------------------------
# Specify the worst case (slowest) libraries and
# slowest temperature/Vcc conditions
# This would involve setting up the slow library as the
# target and setting the link library to the conctenation
# of the target and the synthetic library
#-------------------------------------------------------
 set target_library NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm.db
 set link_library   [concat  $target_library $synthetic_library]

#-------------------------------------------------------
# Specify a 5000ps clock period with 50% duty cycle
# and a skew of 50ps
#-------------------------------------------------------
 #set CLK_PER  10
 set CLK_SKEW 0.05
 create_clock -name $clkname -period $CLK_PER -waveform "0 [expr $CLK_PER / 2]"
$clkname
 set_clock_uncertainty $CLK_SKEW $clkname

#-------------------------------------------------------
# Now set up the 'CONSTRAINTS' on the design:
# 1.  How much of the clock period is lost in the
#     modules connected to it
# 2.  What type of cells are driving the inputs
# 3.  What type of cells and how many (fanout) must it
#     be able to drive
#-------------------------------------------------------

# Following parameters have been modified based on Nangate 45nm library (slow
conditional):
# DFF_CKQ, IP_DELAY, DFF_SETUP, OP_DELAY, WIRE_LOAD_EST
# These values are based on simulation. Credited to: Christopher Mineo

#-------------------------------------------------------
# ASSUME being driven by a slowest D-flip-flop
# The DFF cell has a clock-Q delay of 638 ps
```

```
# EX: 50um M3 has R of 178.57 Ohms and C of 12.5585fF. 0.69RC = 1.55ps, and
wire load
# of 50um M3 is 13fF. Therefore, roughly 20ps wire delay is assumed.
# NOTE: THESE ARE INITIAL ASSUMPTIONS ONLY
#-------------------------------------------------------
#
 set DFF_CKQ 0.638
 set IP_DELAY [expr 0.02 + $DFF_CKQ]
 set_input_delay $IP_DELAY -clock $clkname [remove_from_collection [all_inputs]
$clkname]

#-------------------------------------------------------
# ASSUME this module is driving a D-flip-flip
# The DFF cell has a set-up time of 546 ps
# Same wire delay as mentioned above
# NOTE: THESE ARE INITIAL ASSUMPTIONS ONLY
#-------------------------------------------------------
 set DFF_SETUP 0.546
 set OP_DELAY [expr 0.02 + $DFF_SETUP]
 set_output_delay $OP_DELAY -clock $clkname [all_outputs]

#-------------------------------------------------------
# ASSUME being driven by a D-flip-flop
#-------------------------------------------------------

 set DR_CELL_NAME DFFR_X1
 set DR_CELL_PIN  Q

 set_driving_cell -lib_cell "$DR_CELL_NAME" -pin "$DR_CELL_PIN"
[remove_from_collection [all_inputs] $clkname]

#-------------------------------------------------------
# ASSUME the worst case output load is
# 4 D-flip-flop (D-inputs) and
# 0.013 units of wiring capacitance
#-------------------------------------------------------
 set PORT_LOAD_CELL
NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm/DFFR_X1/D
 set WIRE_LOAD_EST   0.013
 set FANOUT          4
 set PORT_LOAD [expr $WIRE_LOAD_EST + $FANOUT * [load_of $PORT_LOAD_CELL]]
 set_load $PORT_LOAD [all_outputs]

#-------------------------------------------------------
# Now set the GOALS for the compile
# In most cases you want minimum area, so set the
# goal for maximum area to be 0
#-------------------------------------------------------
 set_max_area 0
#-------------------------------------------------------
# This command prevents feedthroughs from input to output and avoids assign
statements
#-------------------------------------------------------
 set_fix_multiple_port_nets -all -buffer_constants [get_designs]

#-------------------------------------------------------
# For logic reduction, I want to make one flat design
# This is commented out.  You can add it back in if you are having trouble with
hierarchy
#-------------------------------------------------------

# ungroup -flatten -all
```

```
#-------------------------------------------------------
# check the design before optimization
#-------------------------------------------------------

#-------------------------------------------------------
# check_design checks for consistency of design and issues
# warnings and errors. An error would imply the design is
# not compilable. See > man check_design for more information.
# HOWEVER, often problems pointed out before compile are ignorable
# Only fix things at this point if the fix is obvious
#-------------------------------------------------------
check_design

#-------------------------------------------------------
# link performs check for presence of the design components
# instantiated within the design. It makes sure that all the
# components (either library unit or other designs within the
# heirarchy) are present in the search path and connects all
# of the disparate components logically to the present design
#-------------------------------------------------------
link
#-------------------------------------------------------
# Now resynthesize the design to meet constraints,
# and try to best achieve the goal, and using the
# library cells.  In large designs, compile can take
# a lllooonnnnggg time!
#
# Note you might need to modify this design
#
# See "Coding Guidelines for Datapath Synthesis" on pipelines and retiming
# This is in the synopsys documentation
# Additional commands are needed to support retiming
#
#-------------------------------------------------------


compile_ultra

#-------------------------------------------------------
#clock period optimization (Verified to work for this design)
#-------------------------------------------------------
create_clock -period 8 -waveform {0 4} clk
compile -incremental

create_clock -period 7 -waveform {0 3.5} clk
compile -incremental

create_clock -period 6 -waveform {0 3} clk
compile -incremental

create_clock -period 5.5 -waveform {0 2.75} clk
compile -incremental

create_clock -period 5.25 -waveform {0 2.625} clk
compile -incremental

create_clock -period 5 -waveform {0 2.5} clk
compile -incremental

create_clock -period 4.75 -waveform {0 2.375} clk
compile -incremental

#-------------------------------------------------------
```

```
#
# Run check_design again.  THIS ONE MATTERS.  ANY PROBLEMS MOST LIKELY NEED
FIXING
#
#-------------------------------------------------------

check_design


#-------------------------------------------------------
# This is just a safety item: Write out the design before
# fixing hold violations
#-------------------------------------------------------
 write -hierarchy -f verilog -o ${modname}_init.v

#-------------------------------------------------------
# Now trace the critical (slowest) path and see if
# the timing works.
# If the slack is NOT met, you HAVE A PROBLEM and
# need to redesign or try some other minimization
# tricks that Synopsys can do
#-------------------------------------------------------

 report_timing  > timing_max_slow.rpt

#-------------------------------------------------------
# This is your section to do different things to
# improve timing or area - RTFM (Read The Manual) :)
# also this is where you can squeeze down the clock period if you want
#-------------------------------------------------------



#-------------------------------------------------------
# Now resynthesize the design for the fastest corner
# making sure that hold time conditions are met
#-------------------------------------------------------

#-------------------------------------------------------
# Specify the fastest process corner and lowest temp
# and highest (fastest) Vcc
#-------------------------------------------------------

 set target_library NangateOpenCellLibrary_PDKv1_2_v2008_10_fast_nldm.db
 set link_library   NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm.db
 set link_library   [concat  $link_library dw_foundation.sldb]
 translate

#-------------------------------------------------------
# Set the design rule to 'fix hold time violations'
# Then compile the design again, telling Synopsys to
# only change the design if there are hold time
# violations.
#-------------------------------------------------------

 set_fix_hold $clkname
 compile -only_design_rule -incremental
 #compile -prioritize_min_paths -only_hold_time
# report_timing -delay min -nworst 30 >
timing_report_${modname}_min_postfix.rpt
# report_timing -delay min -nworst 30 >
timing_report_${modname}_min_postfix.rpt
```

```
#-------------------------------------------------------
# Report the fastest path.  Make sure the hold
# is actually met.
#-------------------------------------------------------
# report_timing  > timing_max_fast_${type}.rpt
 report_timing -delay min  > timing_min_fast_holdcheck_${type}.rpt

#-------------------------------------------------------
# Write out the 'fastest' (minimum) timing file
# in Standard Delay Format.  We might use this in
# later verification.
#-------------------------------------------------------

 write_sdf counter_min.sdf

#-------------------------------------------------------
# Since Synopsys has to insert logic to meet hold
# violations, we might find that we have setup
# violations now.  So lets recheck with the slowest
# corner, etc.
#  YOU have problems if the slack is NOT MET
# 'translate' means 'translate to new library'
#-------------------------------------------------------

 set target_library NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm.db
 set link_library   NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm.db
 set link_library   [concat  $link_library dw_foundation.sldb]
 translate
 report_timing  > timing_max_slow_holdfixed_${type}.rpt
# report_timing -delay min  > timing_min_slow_holdfixed_${type}.rpt

#-------------------------------------------------------
# Sanity checks to see if the libraries are characterized
# correctly
#-------------------------------------------------------
# set target_library NangateOpenCellLibrary_PDKv1_2_v2008_10_fast_nldm.db
# set link_library   NangateOpenCellLibrary_PDKv1_2_v2008_10_fast_nldm.db
# set link_library   [concat  $link_library dw_foundation.sldb]
# translate
# report_timing  > timing_max_fast_holdfixed_${type}.rpt
# report_timing -delay min  > timing_min_fast_holdfixed_${type}.rpt

# set target_library NangateOpenCellLibrary_PDKv1_2_v2008_10_typical_nldm.db
# set link_library   NangateOpenCellLibrary_PDKv1_2_v2008_10_typical_nldm.db
# set link_library   [concat  $link_library dw_foundation.sldb]
# translate
# report_timing  > timing_max_typ_holdfixed_${type}.rpt
# report_timing -delay min  > timing_min_typ_holdfixed_${type}.rpt


#-------------------------------------------------------
# Write out area distribution for the final design
#-------------------------------------------------------
 report_cell > cell_report_final.rpt

#-------------------------------------------------------
# Write out the resulting netlist in Verliog format
#-------------------------------------------------------
 change_names -rules verilog -hierarchy > fixed_names_init
 write -hierarchy -f verilog -o ${modname}_final.v
# write -hierarchy -format verilog -output
${modname}_netlist_holdfixed_${type}.v #RAVI
```

```
#-------------------------------------------------------
# Write out the 'slowest' (maximum) timing file
# in Standard Delay Format.  We might use this in
# later verification.
#-------------------------------------------------------

 write_sdf counter_max.sdf


write -f ddc -output synth_final.ddc
```

8.10 Final Design