ECE 464 Project: Single Stage Binary Convolutional Neural Network

Grant Edwards

Name: **Grant Edwards**
Unityid: **gnedward**
StudentID:. **200192179**

Delay (ns to run provided provided example).
Clock period: **2.87ns**
cycles": **1**
1* 2.87 = **2.87ns delay**

Logic Area: (um^2)

351.917998um^2

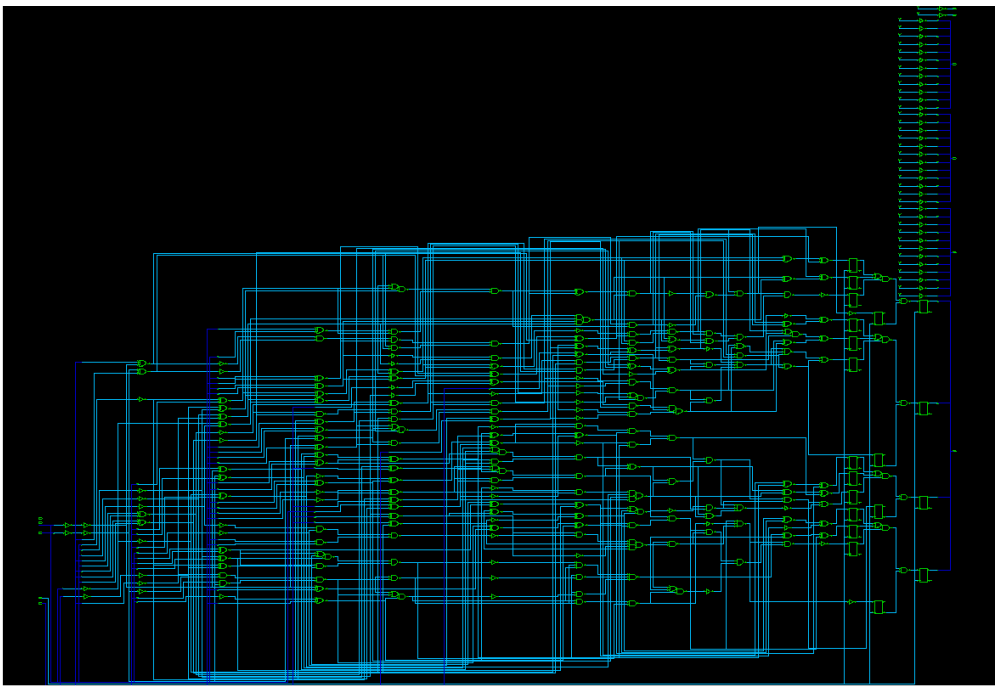
Memory: N/A

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

0.00099009 ns^{-1}.um^{-2}

Delay (TA provided example. TA to complete)

1/(delay.area) (TA)



Abstract

The goal of this project was to create a fixed size, single-stage binary convolutional neural network. This takes in two input matrices, that being weight (3x3) and input (4x4), then returns an accumulated result value within 1 clock cycle and is written into data. My implementation uses XNORs input and weight values, accumulates the value, and then checks to see if the result is above 5. This method utilizes roughly 351 area at a period of 2.87ns, has a calculation cycle of 1, and meets slack requirements at 0.0012. This is the lowest area and period found possible with the configuration of hardware that is necessary for the convolution.

ECE 464 Project: Single Stage Binary Convolutional Neural Network
Grant Edwards

Introduction:

The rest of this report will outline my implementation in my module that calculates the resultant convolution vector and stores the value.

The initial approach I tried was a state machine that read, computer, and outputted the result, but I realized that I could create a far simpler design that would run efficiently, reducing the area and the period to provide a single clock cycle design. I was able to hard-design the XNOR segment and cut out all semblance of a state machine in order to reduce cycle and area. In later sections, you will see some diagrams of the actual logic I use to determine a convolution.

Convolution Explained:

Below, you can see a rather visual demonstration of what the circuit does, specifically with an input 4x4 matrix, and a weight 3x3 matrix. The resultant convolution is stored in a 2x2 result matrix, then stored in the SRAM unit. You can see below a step-by-step visual of the actual convolution, the shadow being the weight, the background being the input, and the caster being the output matrix.

A	**B**	**C**	**D**
E	**F**	**G**	**H**
I	**J**	**K**	**L**
M	**N**	**O**	**P**

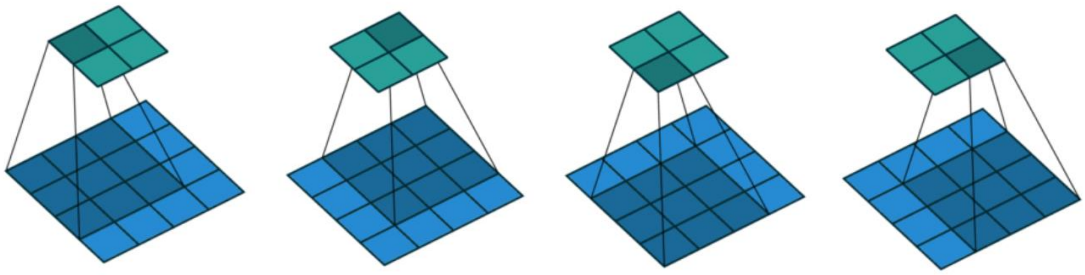
*

W1	**W2**	**W3**
W4	**W5**	**W6**
W7	**W8**	**W9**

=

R1	**R2**
R3	**R4**

W1*A + W2*B + W3*C + W4*E + W5*F + W6*G + W7*I + W8*J + W9*K = **R1**
W1*A + W2*B + W3*C + W4*E + W5*F + W6*G + W7*I + W8*J + W9*K = **R2**
W1*A + W2*B + W3*C + W4*E + W5*F + W6*G + W7*I + W8*J + W9*K = **R3**
W1*A + W2*B + W3*C + W4*E + W5*F + W6*G + W7*I + W8*J + W9*K = **R4**



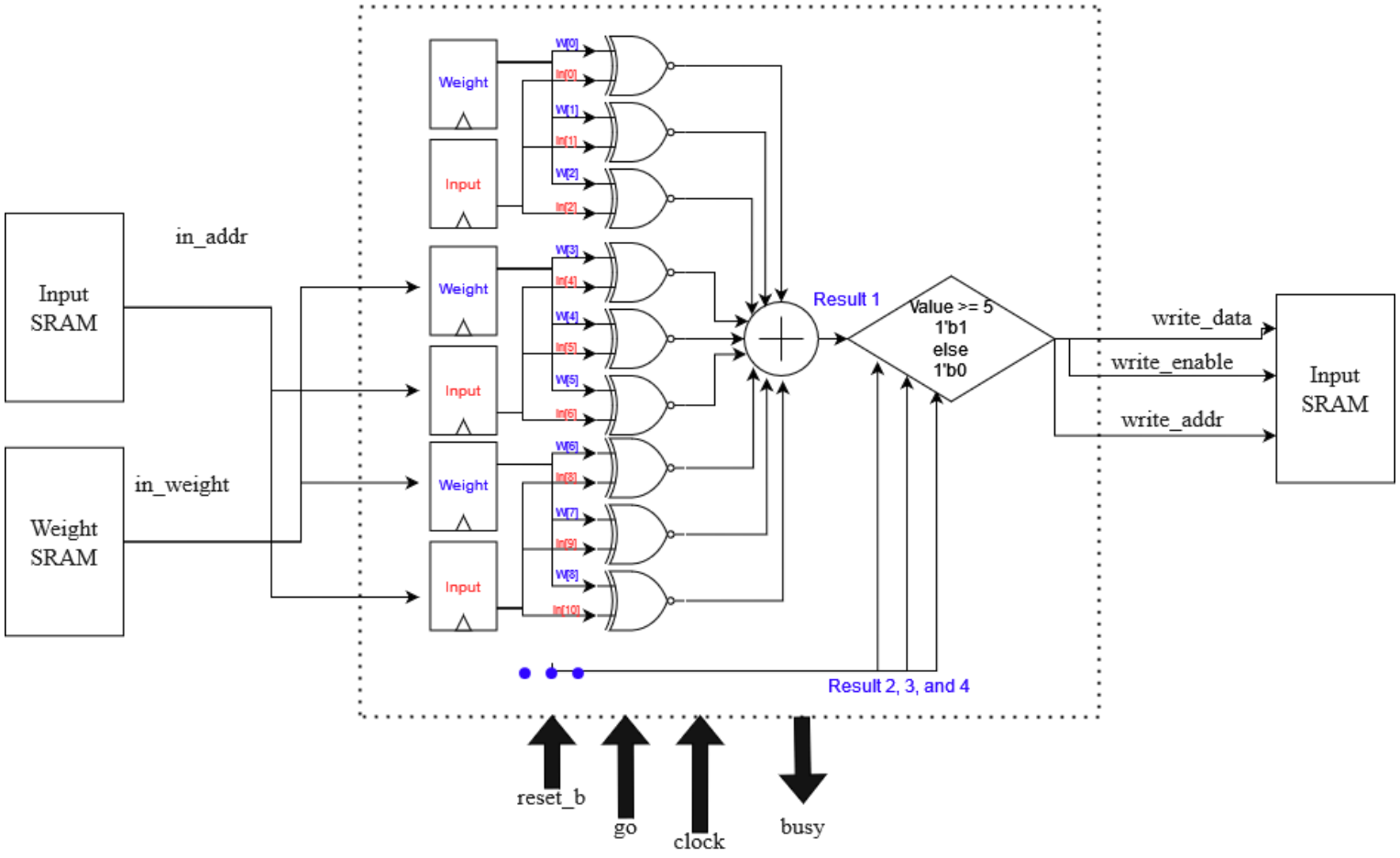
ECE 464 Project: Single Stage Binary Convolutional Neural Network

Grant Edwards

Overview:

The logic design of this project is based around the use of three SRAM accesses, that being the Input matrix from memory, the weight matrix from memory, then the SRAM access to the output. Below, you can see the individual segments of at least 9 different flip-flops pulling information and being used within the XNOR logic into the accumulator.

You can also see a triple blue dot indicating that this structure repeats and is wired in a way in which it can achieve all the four results that are needed. This includes three more sets of nine XNORs being fed into the separate accumulators that are checked with the logic of determining whether the result is positive or negative, then writing it to memory.



ECE 464 Project: Single Stage Binary Convolutional Neural Network
Grant Edwards

Micro-Architecture:

This module does not use a finite state machine, and simply creates a logical, single-cycle convolutional computation that is fed to the output memory. So, no state machine can be demonstrated here, it simply receives information, performs a convolution, and outputs. You can consult the diagram above, as well as the code comments.

XNOR-Accumulator Convolution Algorithm:

Within the project I use a set of XNORs in parallel to perform convolution on the weight and input data, and release it into an accumulator, and then a control block. This works because XNOR resembles the functionality of adding -1s and +1s.

A	B	D
0	0	1
0	1	0
1	0	0
1	1	1

Since this project focuses on getting inputs of negative or positive numbers, the data in binary; therefore, it offers more ease of use, delegating negative numbers to 0 and positive numbers to 1.
You can see from the truth table if we have a situation of positive * positive, the result is a positive number, and in a situation of a negative * negative, the result is positive. For situations where we have a negative and a positive – or vice versa – we will always have a negative number.

So, in this instance, I treated these values as single bit values within a 4-bit accumulator. So that if we received a value of 5 or greater, then that would indicate we had over half of the values being positive ones. If we had 4, then that would indicate we had a negative, and if we had 5 it would indicate we had a positive.

```
if(result2>=5)
    dut_sram_write_data[1] <= 1'b1;
else
    dut_sram_write_data[1] <= 1'b0;
```

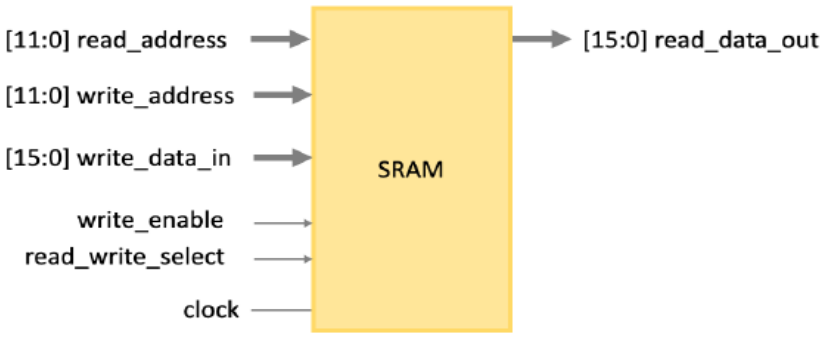
You can see here with this snippet of code that each stored result from the section of the XNOR is evaluated and then directly written to the output into the SRAM for completion of the convolution.

ECE 464 Project: Single Stage Binary Convolutional Neural Network

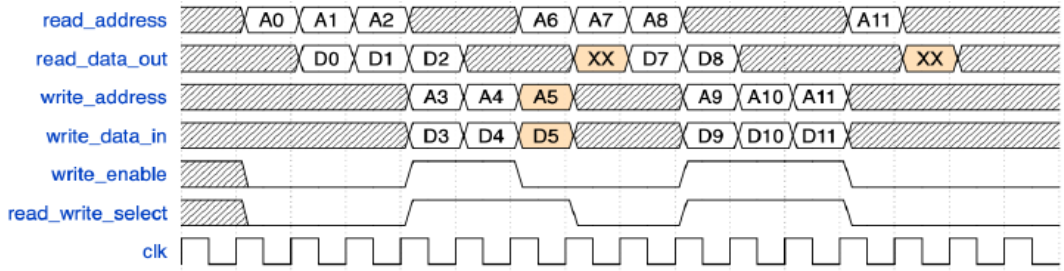
Grant Edwards

Interface Specifications:

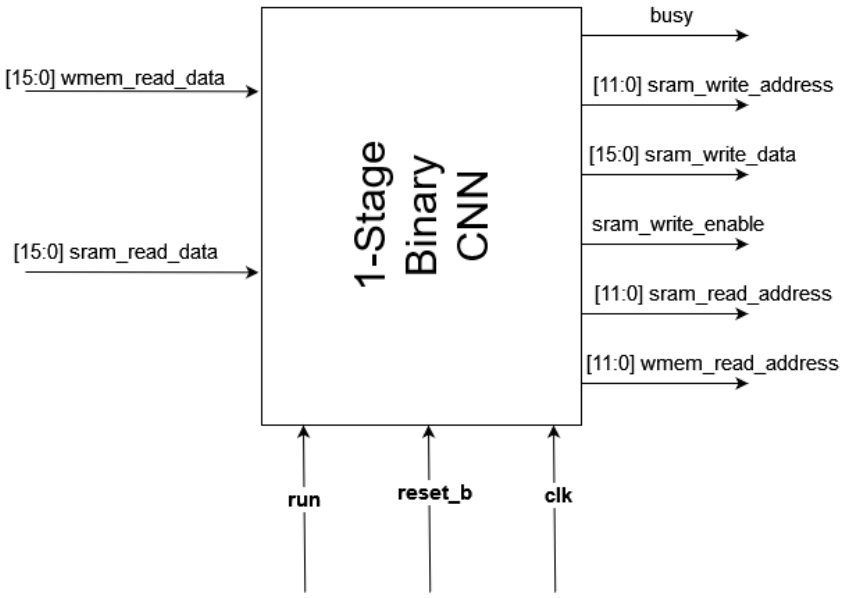
SRAM interface



The SRAM is word addressable and has a one cycle delay between address and data. When writing to the SRAM, you would have to set the "write_enable" to high. The SRAM will write the data in the next cycle.



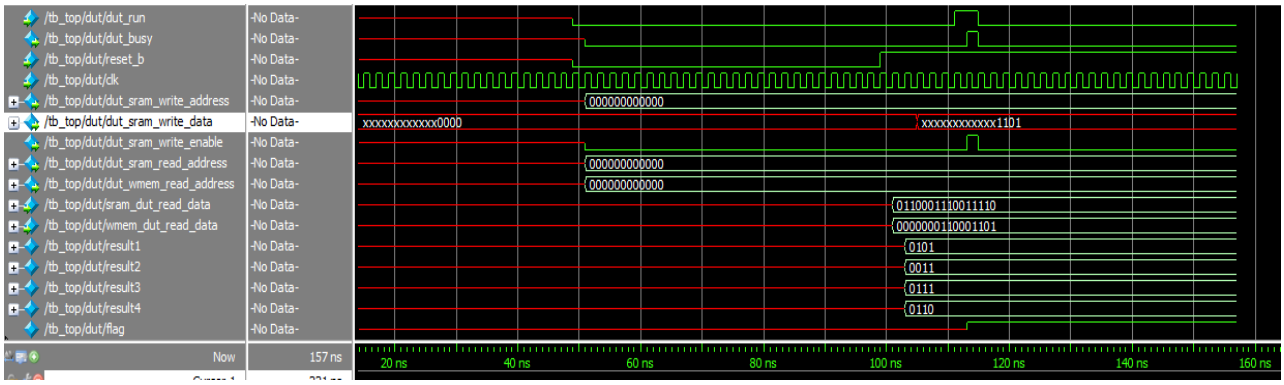
Single Stage Binary Convolutional Interface:



Inputs	Description
[15:0] wmem_read_data	Stores incoming weight data
[15:0]sram_read_data	Stores incoming input data
run	Sends signal for calculation to start
reset_b	Set module to a waiting state where logic is reset
clk	Master clock signal used

Outputs	Description
busy	Gets set high to signal the whole module that it's functional
[11:0] sram_write_address	The address is sent to the output for the SRAM to write
[15:0] sram_write_data	This holds the data sent through output
sram_write_enable	Set high to signal the SRAM to capture the write data
[11:0] sram_read_address	Request address for input matrix
[11:0]wmem_read_address	Request address for weight matrix

Cycle-by Cycle Functionality on Matrices:



As you can see functionality is determined through a combination of inputs and outputs surrounding the run, busy, reset_b, flag, and write_enable segments. write_enable determines the ability to write into the SRAM memory, and we can directly see the reset_b being used to trigger the calculations of the rest of the module. After calculation is complete, the flag is set, the write is high, then low, and the result is directly pushed to the SRAM with dut_run completing the save and write.

The data is taken directly from the data matrices, and is pushed into the convolution within one cycle, those outputs are then measured with the rest of the design to determine their activity in negative or positive ranges associated with the convolutional result.

ECE 464 Project: Single Stage Binary Convolutional Neural Network

Grant Edwards

Verification:

Many of the steps I took to debug this system have been adding checks within the code to print results, as well as checking the individual waveform generation in simulation. I also ensured that I was able to test different computations of convolution with the dataset, and even changed the weight matrix to assess output.

I used a testbench that took a dat file that had information pulled from addresses specified by the design and compared the particular result values to a value that was given to be true. So, when I was testing with my own inputs in order to check for issues, I was able to calculate the results that were needed by hand, as well as simulate a random dataset with pre-calculated golden_outputs. I tried changing or bit flipping segments of my code to try and put a stop to certain pieces of functionality in the computation, but it doesn't seem to have a huge issue, unless it's a drastic flip of a significant bit in the calculation itself.

Once the whole testbench reports all the values are correct, I ensure that I'm getting the correct output files, and see that the correct waveform is outputted, as well as everything along every stage is computed and set correctly.

I think some things that could be added would be automatic testing with random input files and have some autogenerated sequences built into the testbench with golden_outputs pre-calculated to check the circuit with.

Results Achieved:

Using Synopsys 2019, I was able to synthesize designs that meet slack, and hold a relatively limited amount of area; that being an area of 351, a period of 2.87, calculations done in 1 cycle, and power of roughly 97uW. Below are some segments of the synthesized analysis data from synopsys.

ECE 464 Project: Single Stage Binary Convolutional Neural Network

Grant Edwards

Area:

```
dc_shell> report_area

****************************************
Report : area
Design : MyDesign
Version: P-2019.03-SP1
Date   : Tue Nov 16 20:47:34 2021
****************************************

Library(s) Used:

    NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm (File: /afs
llLibrary_PDKv1_2_v2008_10/liberty/520/NangateOpenCellLibrary_PDK

Number of ports:                         89
Number of nets:                         344
Number of cells:                        318
Number of combinational cells:          259
Number of sequential cells:              20
Number of macros/black boxes:             0
Number of buf/inv:                       87
Number of references:                    24

Combinational area:               254.827996
Buf/Inv area:                      48.678000
Noncombinational area:             97.090002
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                  351.917998
Total area:                 undefined
1
```

Timing Analysis:

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : MyDesign
Version: P-2019.03-SP1
Date   : Tue Nov 16 20:48:37 2021
****************************************

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

  Startpoint: wmem_dut_read_data[3]
              (input port clocked by clk)
  Endpoint: result4_reg_2_
            (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
  input external delay                   0.6580     0.6580 r
  wmem_dut_read_data[3] (in)             0.0973     0.7553 r
  U92/Z (BUF_X4)                         0.1863     0.9416 r
  U226/ZN (XNOR2_X2)                     0.2958     1.2374 r
  U311/ZN (XNOR2_X2)                     0.3184     1.5558 r
  U240/ZN (XNOR2_X2)                     0.3426     1.8984 r
  U241/ZN (NAND2_X1)                     0.1289     2.0273 f
  U246/ZN (NAND2_X2)                     0.1733     2.2006 r
  U244/ZN (NAND2_X2)                     0.0748     2.2754 f
  U88/ZN (NOR2_X2)                       0.1517     2.4271 r
  U91/ZN (NOR2_X2)                       0.0805     2.5075 f
  result4_reg_2_/D (DFF_X1)              0.0000     2.5075 f
  data arrival time                                 2.5075

  clock clk (rise edge)                  2.8700     2.8700
  clock network delay (ideal)            0.0000     2.8700
  clock uncertainty                     -0.0500     2.8200
  result4_reg_2_/CK (DFF_X1)             0.0000     2.8200 r
  library setup time                    -0.3110     2.5090
  data required time                                2.5090
  -----------------------------------------------------------
  data required time                                2.5090
  data arrival time                                -2.5075
  -----------------------------------------------------------
  slack (MET)                                       0.0014
```

ECE 464 Project: Single Stage Binary Convolutional Neural Network

Grant Edwards

Power:

```
****************************************
Report : power
        -analysis_effort low
Design : MyDesign
Version: P-2019.03-SP1
Date   : Tue Nov 16 20:49:02 2021
****************************************


Library(s) Used:

    NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm (File: /afs/eos.ncsu.edu/lockers/research/ece/wda
llLibrary_PDKv1_2_v2008_10/liberty/520/NangateOpenCellLibrary_PDKv1_2_v2008_10_slow_nldm.db)


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


Global Operating Voltage = 0.95
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =  75.3829 uW   (78%)
  Net Switching Power  =  20.9618 uW   (22%)
                         ---------
Total Dynamic Power    =  96.3447 uW  (100%)

Cell Leakage Power     =   1.9718 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000             0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000             0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000             0.0000  (   0.00%)
clock_network      0.0000           0.0000            0.0000             0.0000  (   0.00%)
register       3.3111e-02       3.0749e-03        1.5548e+05         3.6341e-02  (  36.96%)
sequential         0.0000           0.0000            0.0000             0.0000  (   0.00%)
combinational  4.2272e-02       1.7887e-02        1.8163e+06         6.1975e-02  (  63.04%)
--------------------------------------------------------------------------------------------------
Total          7.5383e-02 mW     2.0962e-02 mW     1.9718e+06 pW      9.8316e-02 mW
1
```

ECE 464 Project: Single Stage Binary Convolutional Neural Network

Grant Edwards

Conclusions:

With the synthesis and verification complete on the module. It is confirmed to have the correct functionality and is a finished design for the functionality intended. There may be ways in which the design can be edited to allow for multiple sets of input but would likely need an overhaul of specified ports and segments.

This is about as efficient as I could drive my design without compromising total efficiency with a greater period or a greater area. The design completes a proper 1 cycle for calculations, and that is what matters the most with this design – besides area. I think there could be some improvements I haven't gotten to yet for power efficiency, and I can improve the power performance, but may have to sacrifice clocking time for calculations.

After all the time it took me to fix and verify this module, I would say that overall, I am pleased with the throughput and the capability of my module, and hope to make improvements on this design even after this assignment is turned in.