Final Project Report

Name: Cristian Hellmer
UnityID: crhellme
StudentID: 200190107

Delay:
Clock period: 5 ns
cycles for example 0: 3382
cycles for example 1: 19200
Delay for example 0: 16910 ns
Delay for example 1: 96000 ns

Logic Area: (um^2)
25493

1/(delay.area)($ns^{-1}.um^{-2}$):
example 0:
$1/(16910ns*25493um^2) = 2.320e\text{-}9\ ns^{-1}um^{-2}$
example 1:
$1/(96000ns*25493um^2) = 4.086e\text{-}10\ ns^{-1}um^{-2}$

Delay (TA):

1/(delay.area)(TA):

Abstract
The task of this project is to implement a multi-precision artificial neural network. The hardware being designed will be able to receive a selection of inputs and weights from two separate SRAMs and perform matrix multiplication in order to generate outputs. These outputs are then placed in an output SRAM. The hardware should be able to handle inputs and weights of varying precision. My approach to this project was to read the inputs and weights from the SRAMs and store them in an array of registers big enough to handle the largest precision possible. Once finished reading the inputs and weights, they would be multiplied and stored into the output SRAM. Using the approach, I achieved a clock period of 5 ns and a logic area of 25493 um^2.

Multi-precision Artificial Neural Network

Cristian Hellmer

Abstract

The task of this project is to implement a multi-precision artificial neural network. The hardware being designed will be able to receive a selection of inputs and weights from two separate SRAMs and perform matrix multiplication in order to generate outputs. These outputs are then placed in an output SRAM. The hardware should be able to handle inputs and weights of varying precision. My approach to this project was to read the inputs and weights from the SRAMs and store them in an array of registers big enough to handle the largest precision possible. Once finished reading the inputs and weights, they would be multiplied and stored into the output SRAM. Using the approach, I achieved a clock period of 5 ns and a logic area of 25493 um^2.

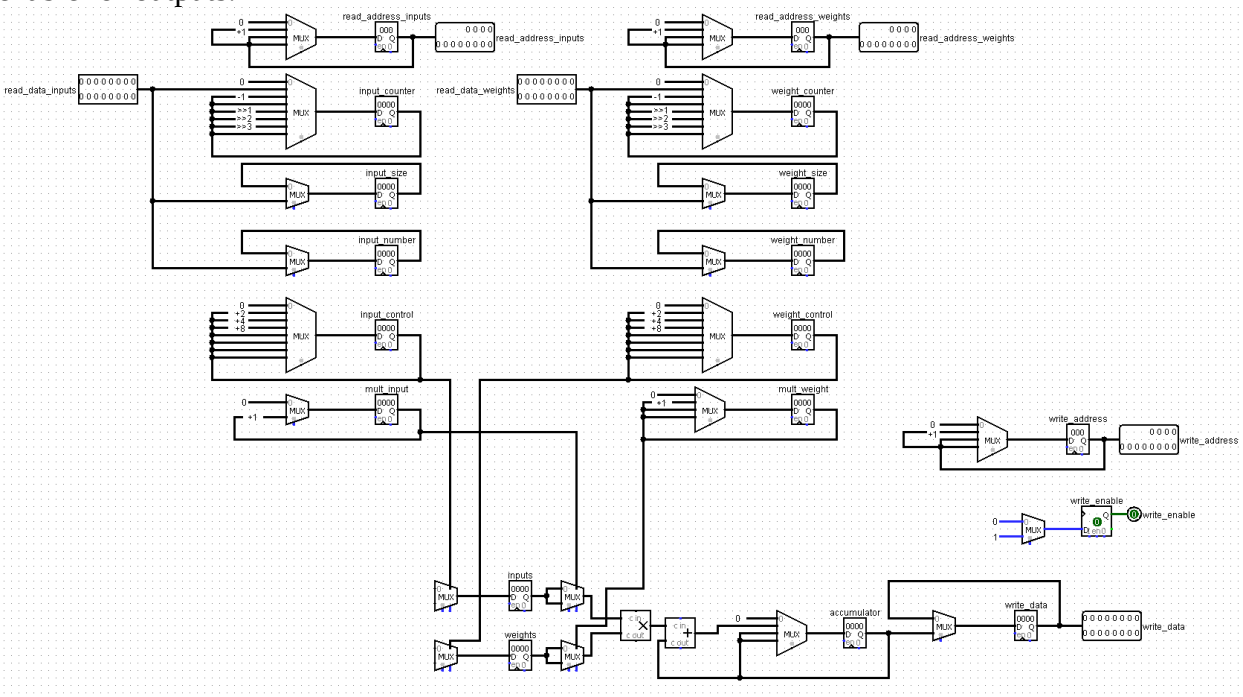
1. Introduction

The hardware being designed here is a neural network which receives a selection of inputs and weights from two separate SRAMs and perform matrix multiplication in order to generate outputs. The inputs and weights can be 2, 4, or 8 bits wide. The outputs will be stored in an SRAM connected to the design. I achieved a clock period of 5 ns, a logic area of 25493 um^2, and a throughput of 200,000,000 MACS/s. The rest of the report discussed the micro-architecture of the design, interface specification, technical implementation, verification methods, and the results achieved.

2. Micro-Architecture

Upon receiving a run signal, the design will begin by obtaining the number of inputs and size of inputs. Using this information, the inputs will be stored in order for use later during multiplication. The number of weights and size of weights will then be accessed, and one row of weights will also be stored. To execute the matrix multiplication, a counter for the inputs and a counter for the weights are used. After each multiply and accumulate (MAC), the input counter will decrement, and the next input and weight will be used for the next MAC. Every time the input counter completes, the system will decrement the weight counter if there are more rows of weights to multiply and obtain the weights from the SRAM. If there are no more rows of weighs, the system will end the current problem and look for a new one. If the SRAM signals that there are no more problems, the system will enter the reset state.

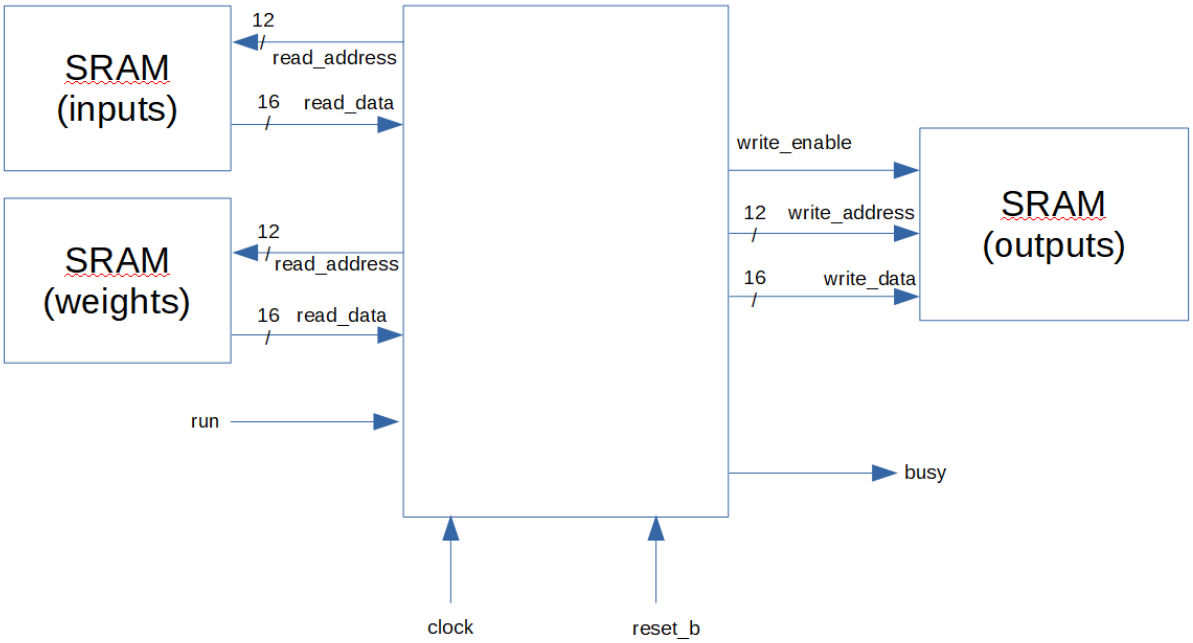
Below is the high-level architecture drawing of the dataflow in the system. read_address_inputs moves through the input SRAM, input_counter is keeps track of the number of inputs needed to be accessed in SRAM for a specific problem, input_size holds the current size of inputs, input_number holds the current number of inputs, input_control controls local input storage, mult_input controls multiplication of inputs, read_address_weights moves through the weight SRAM, weight_counter is keeps track of the number of weights needed to be accessed in SRAM for a specific problem, weight_size holds the current size of weights, weight_number holds the current number of weights, weight_control controls local weight storage, mult_weight controls multiplication of weights, inputs holds the inputs used for multiplication, weights holds the weights used for multiplication, accumulator keeps track of output value during multiplications, write_data holds SRAM write data for outputs, write_address holds SRAM write address for outputs, and write_enable holds SRAM write enable for outputs.



3. Interface Specification

Signal	Width	Function	Input/Output
clock	1	Control memory elements	Input
reset_b	1	Bring the system to a known state	Input
run	1	Begin multiplication of inputs and weights	Input
read_data_inputs	16	Brings input SRAM data into system	Input
read_data_weights	16	Brings weight SRAM data into system	Input
busy	1	Notifies when the system is running	Output
read_address_inputs	12	Determines which input to access in SRAM	Output
read_address_weights	12	Determines which weight to access in SRAM	Output
write_enable	1	Enables writing to the output SRAM	Output
write_address	12	Determines which output address to write to	Output
write_data	16	Holds the data to be write in the output SRAM	Output

The top-level interface to my design allows for 3 SRAMs, one for inputs, weights, and outputs, to be connected to my design. My design can be controlled using run, clock, and reset_b, and be monitored using busy.



4. Technical Implementation

This design does not feature any hierarchy. Instead, the entire design is housed within a single module. This was done due to the relatively small size of the code for the design.

5. Verification

To verify correctness, SRAMs filled with known inputs and weights were given to my design. My design was then ran and the output SRAM was compared to an SRAM pre-filled with the known answers to the problems in the input and weight SRAM. A variety of sizes, signs, and numbers were used to test the functionality of the design. Also, I monitored the signals in the design during operation using ModelSim and verified that the signals matched what I was expecting from my design.

6. Results Achieved

Throughput: 200,000,000 MACS/s
Clock period: 5 ns
Area: 25493 um^2
Setup Slack: 0.0011 ns
Hold Slack: 0.0434 ns

7. Conclusions

In conclusion, this project allowed me to develop multiple skills related to digital logic design. Laying out the dataflow gave me a better understanding of how the system works and how the different signals interact with each other. I believe an area of improvement is the area of the design. There is possibility to eliminate the use of registers to store the inputs and weights locally, however, implementing such a design would take considerable effort. Otherwise, I believe the design to be fairly optimized.