Final Project Report First Page. Must match this format (Title)

Name: **Cheng-Yu(Rick) Lin**
Unityid: **clin33@ncsu.edu**
StudentID:. **200607561**

Delay (ns to run provided provided example).
Clock period: **5.25 ns**
cycles: **2820 cycles**

Logic Area:
9802.631883 um^2

Memory: N/A

1/(delay.area) ($ns^{-1}.um^{-2}$)
1.3780942e-8

Delay (TA provided example. TA to complete)

1/(delay.area) (TA)

ECE 564 Final Project

Name: Cheng-Yu(Rick) Lin
Unity ID: clin33@ncsu.edu
ID: 200607561

Abstract

This project focuses on implementing key components of the Transformer architecture in hardware, specifically the self-attention mechanism,

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

which is crucial for processing sequential data in natural language processing and other AI applications.

Introduction

In this project, the main goal is to implement the "Scaled Dot-Product Attention" computation in transformer architecture in RTL code, with an emphasis on optimizing performance measured area * total computation delay. The implementation is divided into three main phases:

1. Computation of Query, Key, and Value matrices by multiplying the Input matrix with respective Weight matrices (W^{Query}, W^{Key}, W^{Value}).
2. Calculation of the Score matrix, obtained by Query * Key.
3. Evaluation of the final Result matrix, computed by Score * Value

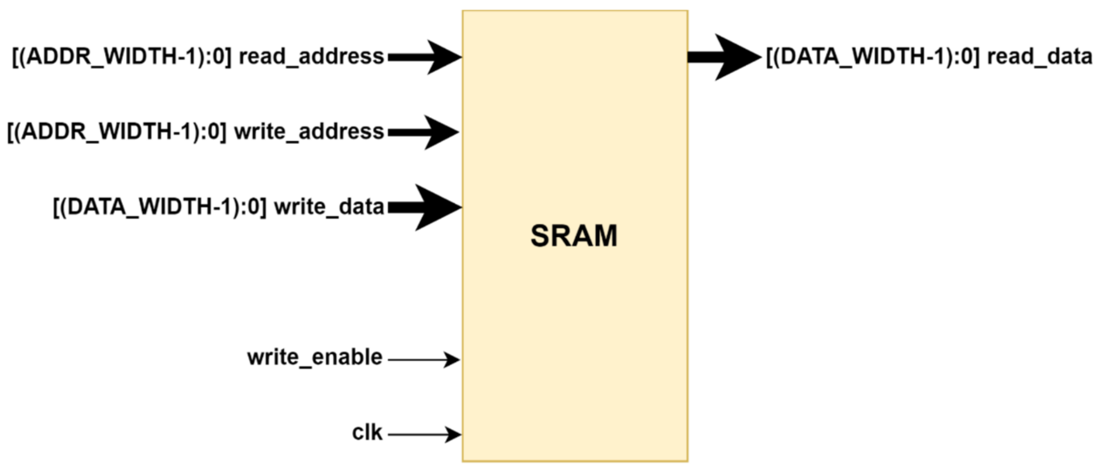
Interface Specification

1. I/O Signal Table

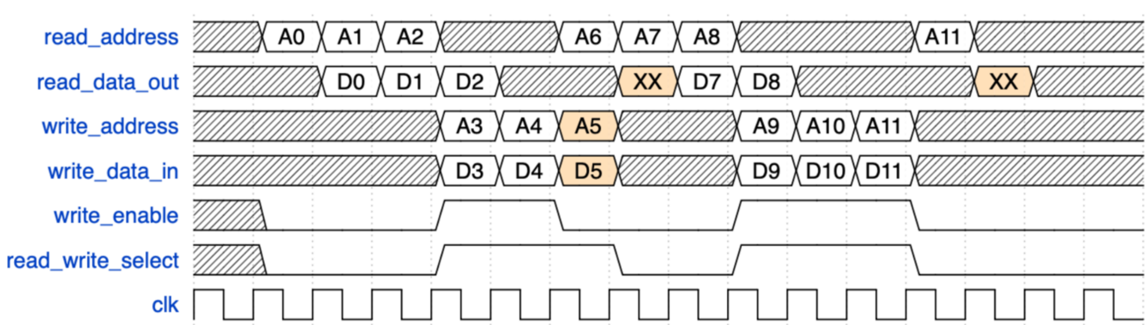
Name	I/O	Width	Description
reset_n	I	1	Asynchrounous Negative edge reset signal
clk	I	1	Universal clock signal
dut_valid	I	1	Used to signal that a valid input can be computed from the SRAM
dut_ready	O	1	Indicate dut is ready to receive new input from the SRAM
Input_write_enable	O	1	0 for Read, 1 for write to Input SRAM
Input_write_address	O	16	Not used
Input_write_data	O	32	Not used
Input_read_address	O	16	The address to read from Input SRAM
Input_read_data	O	32	The corresponding data read from SRAM according to input read address
Weight_write_enable	O	1	0 for Read, 1 for write to Input SRAM
Weight_write_address	O	16	Not used
Weight_write_data	O	32	Not used
Weight_read_address	O	16	The address to read from Weight SRAM

Weight_read_data	O	32	The corresponding data read from SRAM according to Weight read address
Result_write_enable	O	1	0 for Read, 1 for write Result to SRAM
Result_write_address	O	16	The address to write to Result SRAM
Result_write_data	O	32	The corresponding data to write to SRAM according to Result Write address
Result_read_address	O	16	The address to read from Result SRAM
Result_read_data	O	32	The corresponding data read from SRAM according to Result read address
Scratch_write_enable	O	1	0 for Read, 1 for write Scratch to SRAM
Scratch_write_address	O	16	The address to write to Scratch SRAM
Scratch_write_data	O	32	The corresponding data to write to SRAM according to Scratch Write address
Scratch_read_address	O	16	The address to read from Scratch SRAM
Scratch_read_data	O	32	The corresponding data read from SRAM according to Scratch read address

2. SRAM Interface



3. SRAM Timing Diagram



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

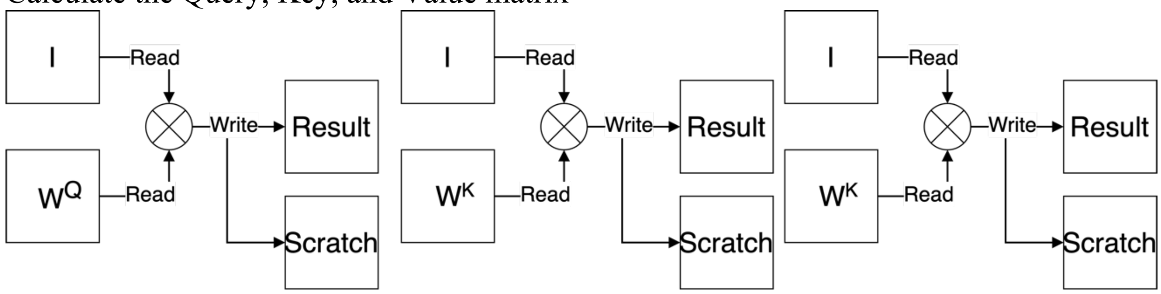
SRAM will write the data in the next cycle. Additionally, write and read in the same cycle is prohibited, read to the same address after write is forbidden as well.

Micro-Architecture

1. Flow Chart

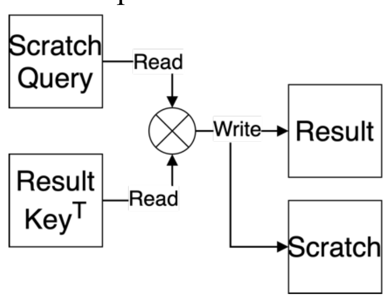
Phase 1:

Calculate the Query, Key, and Value matrix



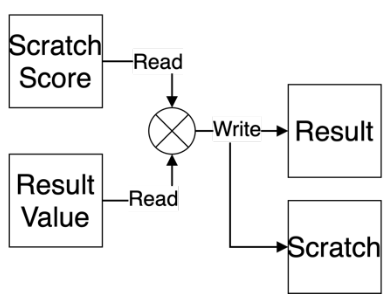
Phase 2:

Calculate Score matrix, where it read data from Scratch Pad and Result SRAM to accelerate computation process in this phase.

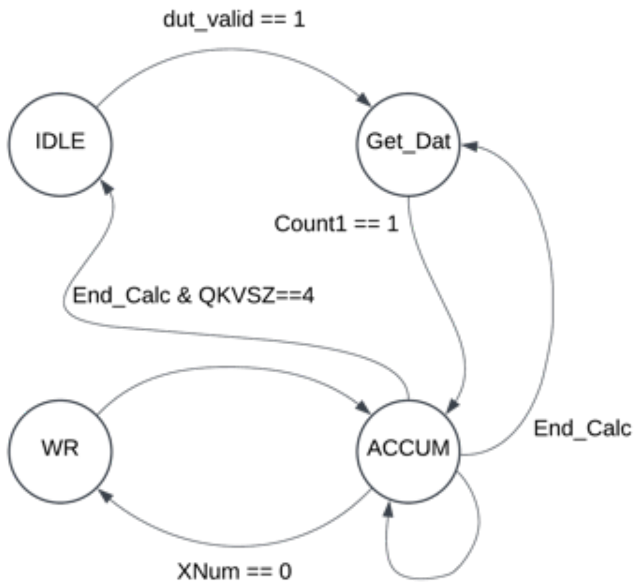


Phase 3:

Same process with phase 2.



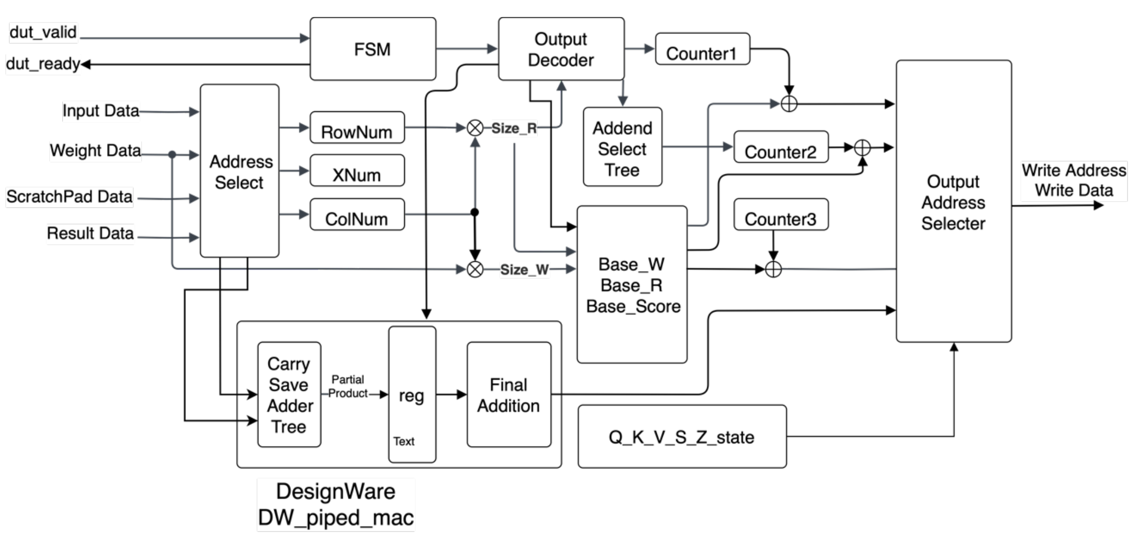
2. State Diagram



IDLE: wait for testbench to send dut_valid signal to start computation
Get_Dat: get and calculate each matrices size
ACCUM: multiplication and accumulation
WR: write state

In this project, I utilized a state register named QKVSZ to record which phase is the whole computation process at, hence reduced the complexity and the size of finite state machine.

3. Architecture



Detailed Implementation and Innovation

1. Invoke DesignWare pipelined-MAC module

To enhance performance and reduce critical path delay in multiplication and addition operations, the design utilizes the DesignWare DW_piped_mac module. This component employs multi-level carry-save adders to calculate partial products in the first cycle, followed by the final addition in the subsequent cycle. This approach significantly reduces the clock period with more than 1.5ns by distributing the critical path across two cycles.

2. SRAM address computation,

The real SRAM address equation is,

$$read/write\ Address\ =\ Base\ Address\ +\ Relative\ Address(Counter)$$

The implementation uses two counters and an addend select tree for addressing the first-read and second-read matrices, along with a separate counter for the write matrix. Additionally, three registers are used to track the current memory positions of each matrix.

Results Achieved

Through the critical path distribution of DW_piped_mac, the clock period and area is improved by 0.65 ns and approximately 400 um^2 compared to original design.