ECE 564 Final Project Report

Erik Seuster

Electrical and Computer Engineering
North Carolina State University

November 15, 2024

Name: Erik Seuster Unityid: eseuste@ncsu.edu StudentID: 200344712		
Delay (ns to run provided example). Clock period: 7.575 ns # cycles": 1020 Delay: 7726.5 ns	Logic Area: 13598.186 (um^2) Memory: N/A	1/(delay.area) $9.5178*10^{-9}$ ($ns^{-1}.um^{-2}$)
Delay (TA provided example. TA to complete)		1/(delay.area) (TA)

Abstract:
This project report presents the design and implementation of a hardware unit for the Transformer Self-Attention mechanism, developed as part of ECE 564 at North Carolina State University. Transformers, as proposed in *Attention is All You Need* by Vaswani et al., are crucial to modern AI applications due to their parallel processing capabilities, which surpass traditional Recurrent Neural Networks (RNNs) and Convolution Neural Networks (CNNs) models in tasks like natural language processing and sentiment analysis. This project implements the Scaled Dot-Product Attention, a core operation in self-attention, by designing a micro-architecture that computes the query, key, and value matrices from input and weight matrices, followed by the scaled dot-product to produce the attention matrix. The architecture uses SRAM for data storage and management, along with a multiply-accumulate matrix multiplier, to efficiently handle large datasets required for Transformer operations. Verification was performed using Questa (ModelSim) and Synopsys, with the final design demonstrating effective clock frequency, area efficiency, and throughput, highlighting the potential of hardware-based approaches for Transformer implementations.

Transformer Self-Attention Hardware Unit Design

North Carolina State University - ECE 564 - ASIC and FPGA Design with Verilog

Erik Seuster

I. Introduction

THIS project focuses on designing a hardware implementation of the key components required in a Transformer Self Attention module for Artificial Intelligence. Transformers use Recurrent Neural Networks (RNNs) and Convolution Neural Networks (CNNs) to process information in parallel instead of sequentially, allowing newer AI models to overcome limitations associated with Long Short Term Memory. Transformers use crucial innovations such as positional encoding and embedding, self-attention, and multi-head-attention to create a sequence of relationships enabling parallel processing.

This project specifically implements scaled dot-product attention, a tool used by transformers to enable AI models to weigh the importance of different words or data points in a sequence. Memory mapping in SRAM is used to efficiently read and write integer values used in matrix multiplication. This report will discuss the hardware micro-architecture, interface specifications, technical implementation, verification, and final results.

II. Micro-Architecture

The hardware approach used in this project relies matrix multiplication to calculate different result matrices by using a multiply-accumulate adder. My design has states for calculating matrices for a query (Q), key (K), value (V), score (S), and ultimately the scaled dot-product attention matrix (Z). The formula used to compute the scaled dot-product attention matrix is as follows:

$$Z = \left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{1}$$

In order to more easily calculate Z, four calculations are done beforehand. First, using input values for the input and weight matrices, Q, K, and V are calculated.

- Query (Q) is obtained by multiplying input embedding (I) with weight matrix W^Q.

$$Q = I * W^Q \tag{2}$$

- Key (K) is obtained by multiplying input embedding (I) with weight matrix W^K.

$$K = I * W^K \tag{3}$$

- Value (V) is obtained by multiplying input embedding (I) with weight matrix W^V.

$$V = I * W^V \tag{4}$$

Afterwards, S can be calculated using these results.

- Transpose the Key matrix from the previous step to obtain K^T.
- Multiply Query (Q) with Key transpose (K^T).

$$S = QK^T \tag{5}$$

Last, Z is calculated and the results are finalized.

- Multiply the score (S) with the value (V).

$$Z = S * V \tag{6}$$

Through meticulous data management, a state machine for control signals such as read and write enables, a data path connecting the high level module to the matrix multiplication module, and handshakes with a test fixture, the data flow and hardware complexity of these calculations are simplified. Pipe lined data flow is used to minimize latency and efficiently read and write to available SRAMs and maximize efficiency.

III. Interface Specification

The top level module of my design interfaces with four SRAMs: one for the input matrix, weight matrix, result matrix, and scratchpad matrix. Reference figure 1 in the appendix for a high level block diagram of the top level module interfacing with the SRAMs and the MyMatrixMultiplier module.

The interface to the design consists of essential control signals, data inputs, and outputs that facilitate communication between the design under test (DUT) and its environment. As shown in the Signal List (Figure 2 in the appendix), control signals, such as reset_n, clk, dut_valid, and dut_ready, manage the initialization and flow of data through the hardware. The reset_n signal resets the system to a known state, while clk provides the clock signal that drives sequential logic. The handshake signals, dut_valid and dut_ready, synchronize data transfers, ensuring that the DUT is ready to accept new inputs when they are valid and signaling when results are ready to be read. Additionally, interfacing with the SRAMs is essential for reading inputs, and writing the resultant output matrices.

The interface timing behavior is depicted in the timing diagram shown in figure 3 of the appendix. Upon reset, the DUT asserts dut_ready, indicating readiness to receive data. When the test fixture asserts dut_valid, the DUT deasserts dut_ready and begins reading from the input and weight SRAMs for computation. After processing, the DUT writes results to both the result and scratchpad SRAMs, reasserting dut_ready to signal that output data is available. This cycle repeats for each input set, with the result and scratchpad write

enables set high during result storage. The scratchpad serves as a duplicate of the result SRAM to enable computations for S and Z, both of which require accessing previous result matrices for computation.

Parallelism ensures maximum efficiency when reading from the result matrix. The detailed interface specification enables clear and efficient communication between the DUT and external components, ensuring smooth data flow and synchronization throughout the computation process.

IV. Technical Implementation

The hardware design is organized hierarchically to facilitate modularization and efficient data flow management. At the top level, the design is structured around three primary modules as shown in figure 1 of the appendix: the top level DUT, the matrix multiplier module, and SRAM interfacing. The DUT module, named MyDesign, interacts directly with both the SRAM interface and the MyMatrixMultiplier module. It serves as a middleman to read from the SRAM, send the correctly indexed values to MyMatrixMultiplier, and then writes the resultant values to the correct indexes in the result and scratchpad SRAMs.

Counters drive the logic within MyMatrixMultiplier to notify the module when a result computation has been completed based on offsets given to it by MyDesign. MyMatrixMultiplier uses an integer multiply accumulate adder to accurately compute the results of the matrix multiplication taking place within the module.

V. Verification

In the preliminary stages of the design, the matrix multiplier and counters associated with it were tested individually prior to the integration of the high level module. This ensured signals were functioning properly when computing the matrix multiplication and writing the results to the correct indexes within the result SRAM. After this was working properly, the higher level module was introduced.

For verification, a test bench using four different input matrix sizes and values was used. Simulation was done using Questa(ModelSim) to show waveforms and register values at different time intervals within the design. Test bench inputs were tested sequentially to verify outputs and proper register resets after obtaining the final Z matrix values. Additionally, Synopsys was used to synthesize the design and ensure there were no errors such as wired-or logic, combinational logic feedback, or unintentional latches among others.

VI. Results

After simulating my design in Questa (ModelSim), and synthesizing my design in Synopsis, the final results are displayed in the table as follows.

Metric Category	**Result**
Max Clock Freq (MHz)	132.013
Logic Area (μm^2)	13598.186
# of cycles achieved	1020
Delay (ns)	7726.5
EDA ($ns^{-1} \cdot \mu m^{-2}$)	9.5178×10^{-9}
Delay (ns)	7726.5
Throughput (GigaBits/sec)	8.5

VII. Conclusion

The project successfully demonstrates the hardware implementation of a Transformer Self-Attention module, highlighting the feasibility of using dedicated hardware for AI-centric matrix computations. The design achieved a maximum clock frequency of 132.013 MHz, a logic area of 13,598.186 μm², and a throughput of 8.5 Gbps. Through efficient pipelined data handling and a modular design approach, this implementation showcases the potential for integrating similar architectures in AI hardware accelerators. Future work may include expanding this architecture to support additional transformer layers or optimizing for lower power consumption, further enhancing the applicability of hardware-accelerated AI in real-world systems.

APPENDIX

Figure 1: High Level Block Diagram

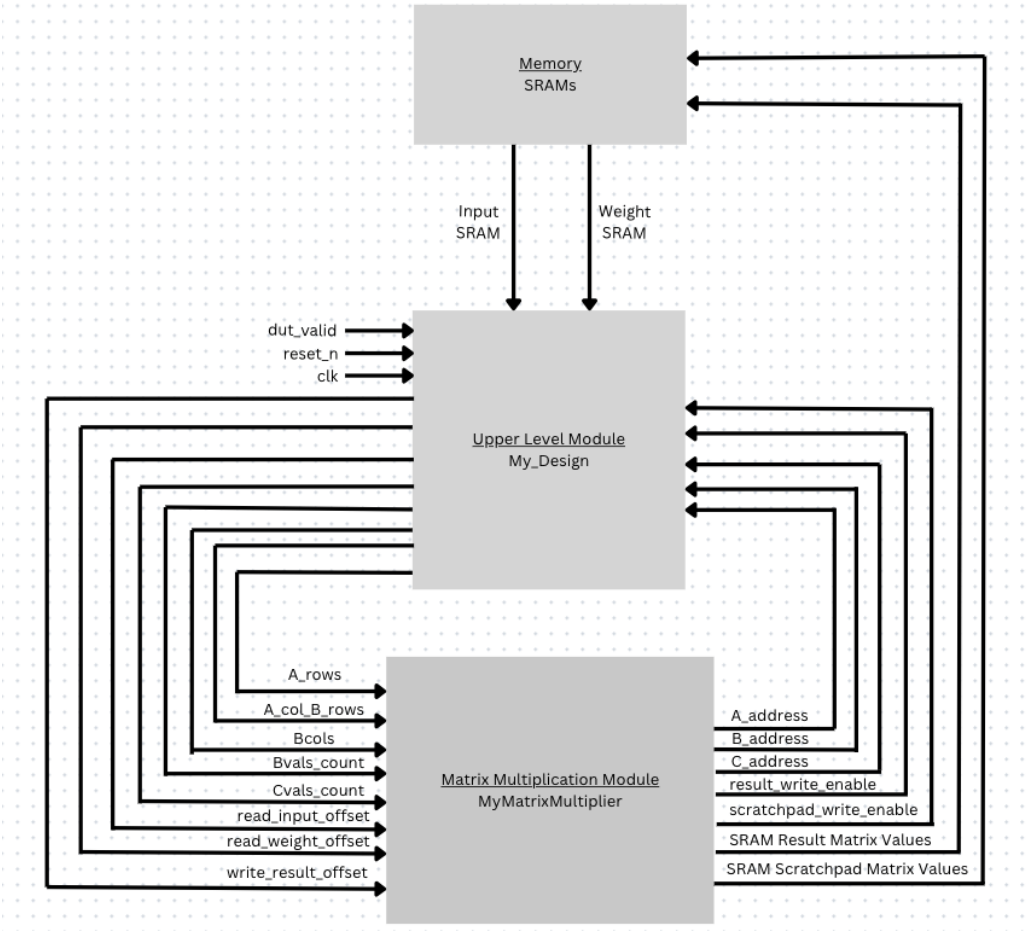

Figure 2: Signal List

Signal Name	Size (bits)	Description
clk	1	Used for timing in procedural blocks.
reset_n	1	Reset to get variables into a known state.
dut_valid_control	1	Used to handshake with `MyMatrixMultiplier` module.
dut_tb_input_write_enable	1	Input matrix write enable.
dut_tb_sram_read_address	16	Input matrix read address index.
tb_dut_sram_input_read_data	32	Input matrix data stored at the read address.
dut_tb_sram_weight_write_enable	1	Weight matrix write enable.
dut_tb_sram_weight_read_address	16	Weight matrix read address index.
tb_dut_sram_weight_read_data	32	Weight matrix data stored at the read address.
dut_tb_sram_scratchpad_write_enable	1	Scratchpad matrix write enable.
dut_tb_sram_scratchpad_write_address	16	Scratchpad matrix write address index.
dut_tb_sram_scratchpad_write_data	32	Scratchpad matrix data to be written to the write address.
dut_tb_sram_scratchpad_read_address	16	Scratchpad matrix read address index.
tb_dut_sram_scratchpad_read_data	32	Scratchpad matrix data stored at the read address.
dut_tb_sram_result_write_enable	1	Result matrix write enable.
dut_tb_sram_result_write_address	16	Result matrix write address index.
dut_tb_sram_result_write_data	32	Result matrix data to be written to the write address.
dut_tb_sram_result_read_address	16	Result matrix read address index.
tb_dut_sram_result_read_data	32	Result matrix data stored at the read address.

Figure 3: Timing Diagram