Final Project Report

Title: Scaled Dot-Product Attention for LLM Transformer Models using SystemVerilog

Delay: **13447.4 ns** (to run provided example). Clock period: **7.1ns** # cycles": **1894 cycles**	Logic Area: **8216.4740 (um^2)**	1/(delay.area): **$9.05 * 10^{-9}$** ($ns^{-1}.um^{-2}$)

Abstract:
This work presents the design and synthesis of a Scaled Dot-Product Attention module, optimized for Transformer-based large language models (LLMs). The module can support the Synopsys DesignWare library for double-precision floating-point matrix multiplication (if the values are floating point) in the multiply-accumulate operations essential to the attention mechanism. RTL logic is developed to pipeline input matrix reads from SRAM, manage matrix transposition and multiplication, and write the computed outputs back to SRAM. This pipelined architecture is synthesized under strict area and timing constraints to achieve minimal delay and maximum performance per unit area. With a delay of **13447.4 ns** for the provided example and a clock period of **7.1 ns** across **1894 cycles**, the design achieves an area of **8216.4740 μm^2**. The **performance per unit area is 9.05×10^{-9} $ns^{-1}\mu m^{-2}$**, highlighting the synthesis optimization achieved in terms of both speed and area for high-performance attention operations in LLM architectures.

1. Introduction:

The project designed a SystemVerilog module for calculating the Scaled Dot-Product Attention for LLM Transformer models. The operations needed to achieve this is as below:

1. Calculating the query (Q), key (K), and value (V) matrices.
    - Query (Q) is obtained by multiplying input embedding (I) with weight matrices (W^Q).
        i. $Q = I * W^Q$
    - Key (K) is obtained by multiplying input embedding (I) with weight matrices (W^K).
        i. $K = I * W^K$
    - Value (V) is obtained by multiplying input embedding (I) with weight matrices (W^V).
        i. $V = I * W^V$
2. Compute the score matrix (S).
    - Transpose the Key matrix from the previous step to obtain K^T.
    - Multiply Query (Q) with Key transpose (K^T).
        i. $S = QK^T$
3. Compute the scaled dot-product attention (Z).
    - Multiply the score (S) with the value (V).
        i. $Z = S * V$

Input and Weight matrices are stored in SRAM. The values from these matrices are read from the SRAM at each clock cycle in a pipelined fashion and sent to the MAC (Multiply-Accumulator) unit and written back to the SRAM. In this module these two matrices are stored in 2 different SRAMS (with one read and one write port each) and the result is written in a third 'result' SRAM. We need a scratchpad for placing the transposed values of the 'K' matrix.

The first four operations (calculating Q,K,V and S) are pipelined, but the fourth operation (Z calculation) has one stalled cycle after the S-calculation. This was needed to be done as the area would have increased. This a tradeoff between area and #cycles, but when tried with both the approaches, stalling seemed to give a better performance/area.

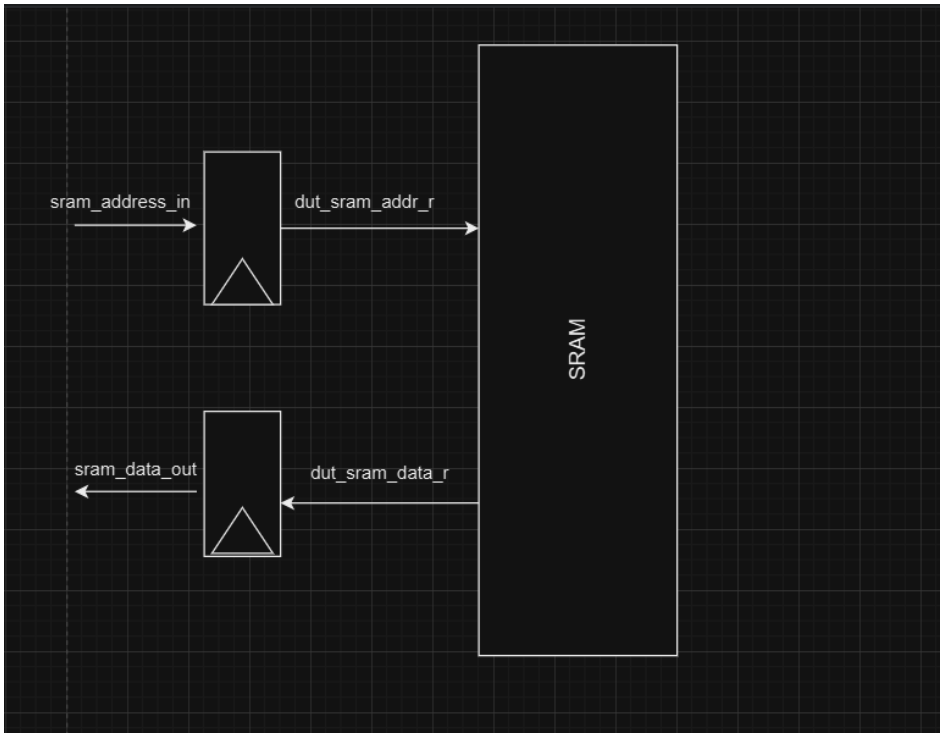
Three multipliers are used -> first one for calculating the dimensions of the weight matrix, second for calculating dimensions of Q/K/V matrix and the third one for the matrix multiplication operation (MAC Unit). Used 4 counters/iterators -> input matrix row and column iterator, weight matrix dimension iterator, weight matrix offset iterator(Q/K/V).

Control signals:

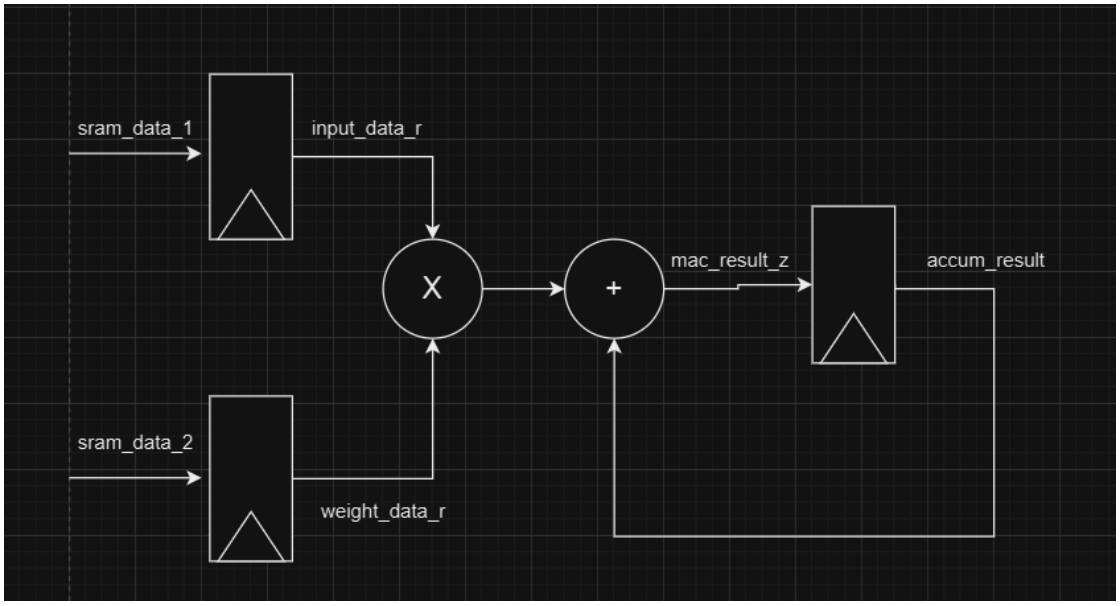
- *dut_valid:* used as part of a hand shack between the test fixture and the dut. Valid is used to signal that a valid input can be computed from the SRAM.
- *dut_ready:* used to signal that the dut is ready to receive new input from the SRAM.
- Together these two signals tell the test fixture the state of the dut. So, the dut should assert *dut_ready* on reset and wait for the *dut_valid* to be asserted.
- Once *dut_valid* is asserted by the test fixture, the dut should set *dut_ready* to low and can start reading from the SRAM.
- Dut should hold the *dut_ready* low until it has populated the result values in the SRAM. Once the results are stored in the SRAM the *dut_ready* will be asserted high, signaling that the result is valid, and is ready to be read from SRAM. Fig

2. Micro-Architecture:

SRAM Interface:

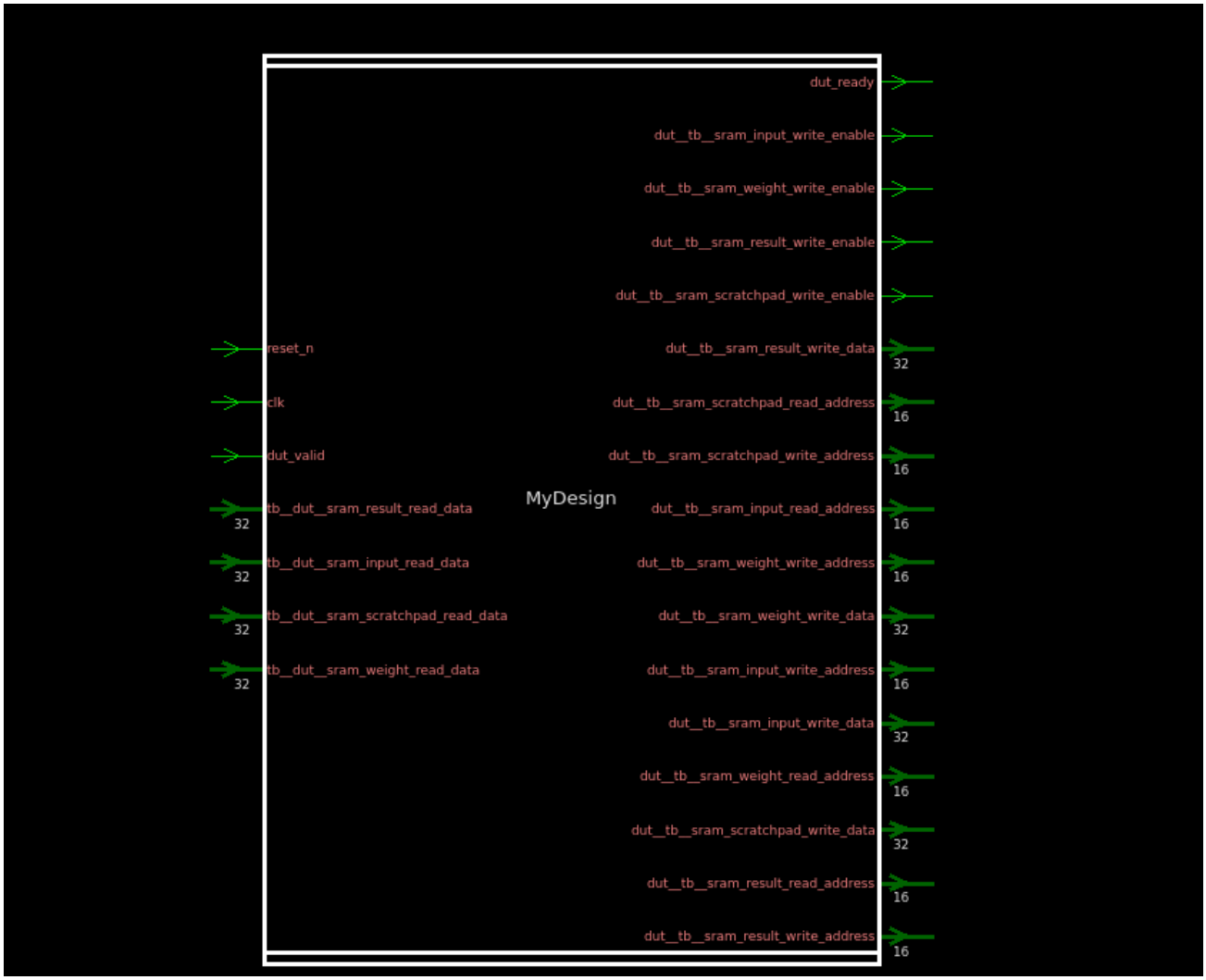


MAC Unit:



3. Interface Specification

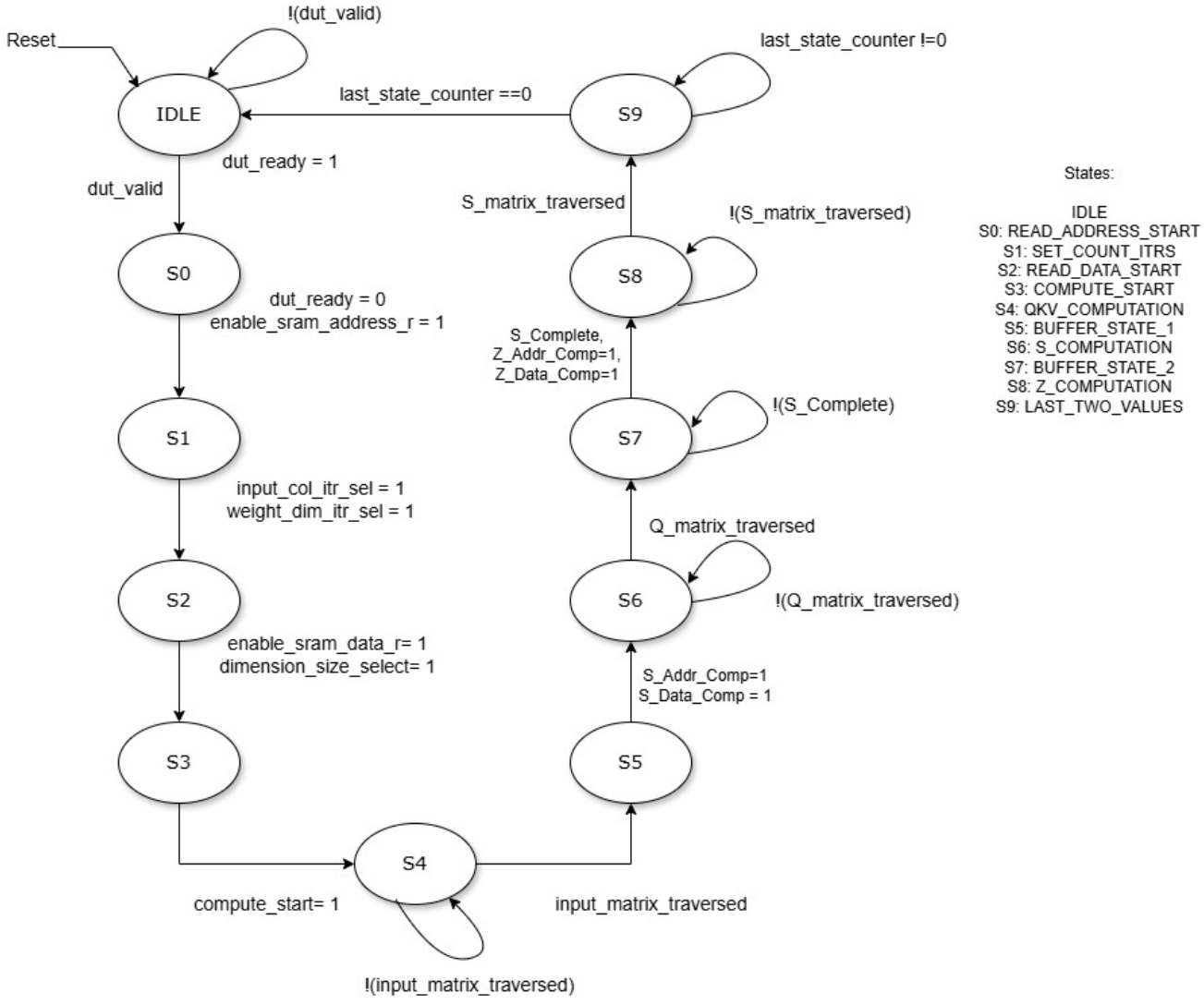
Top Level View:



Control Signal	Width	Function
compute_complete	1	Flag to signal compute done
dut_ready	1	Signal DUT Ready
dut_valid	1	DUT Valid from test fixture
enable_sram_address_r	1	Start SRAM Address Read
enable_sram_data_r	1	Start Reading SRAM Data
dimension_size_select	2	Measure all dimensions at that instance
input_col_itr_sel	1	Input matrix column iterator select
weight_col_itr_sel	1	Weight matrix dimension itr select
input_row_itr_sel	1	Input matrix row itr select
compute_start	1	Compute Start

result_write_en	1	Signal to set write enable to high
last_state_counter_sel	1	For computing last two values
which_weight_count_sel	1	Select which weight matrix (Q/K/V)
input_matrix_traversed	1	Has input matrix fully traversed?
s_addr_comp	1	Score matrix address read
z_addr_comp	1	Z matrix address read
sz_data_comp	1	Score/Z data read

4. Technical Implementation



5. Verification

- Robust Verification has been carried out to make sure that all the corner cases work and give perfect results. Corner cases include computing large matrix calculations, keeping the score matrix as only a single element and keeping only row/column matrices for input and weight.

- Handshaking between the DUT and test fixture was verified by running multiple test cases in a single simulation run to make sure that the DUT after completion of one test case goes back to IDLE state and waits for the next Valid signal from the test fixture.

6. Results Achieved:

Test Case #	Final Simulation Time (ns)	Final Simulation Cycle Count
Test Case 1	1770	354
Test Case 2	1130	226
Test Case 3	4970	994
Test Case 4	1600	320
Combined 4 Cases	9470	1894

Synthesis Clock Period: **7.1ns**
Logic Area: **8216.4740 μm^2**
Performance per unit area (for combined 4 cases): **$9.05 \times 10^{-9}\ ns^{-1}\mu m^{-2}$**

7. Conclusions:

The clock period was decreased from 10ns to 7ns and Area and Performance/Area was observed. Clock period of 7ns was achieved with minimum area without timing violations. But the slack was 0ns in this case. So, to be on the safer side the clock period was slightly increased to **7.1ns** and was used as the optimal clock period for the maximum Performance/Area, which gave a slack of **0.004ns**

Clock Period (ns)	Area (μm^2)	Performance/Area ($ns^{-1}\mu m^{-2}$)
7	8320.7460	9.065×10^{-9}
7.1	8216.4740	9.05×10^{-9}
8	8048.6281	8.2×10^{-9}
9	7874.9301	7.45×10^{-9}
10	7778.9041	6.78×10^{-9}