Final Project Report – ECE 564

Name: Rushiraj Chaitanyakumar Sheth
Unityid: rsheth@ncsu.edu
StudentID: 200596988

Delay (ns to run provided provided example).
Clock period: **5.80 ns**
cycles": **3354**
Delay (clock period * #cycles): **19453.2 ns**

Logic Area: (um^2)
12866.6859

Memory: N/A

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

4.00×10^{-9} ($ns^{-1}. um^{-1}$)

Delay (TA provided example. TA to complete)

1/(delay.area) (TA)

RTL Design: Self-attention matrix – LLM Transformers

Rushiraj Chaitanyakumar Sheth

Abstract

This project focuses on the hardware implementation of the scaled dot-product attention mechanism $(\frac{QK^T}{\sqrt{d_k}})V$.

The objective is to design and verify Register Transfer Level (RTL) architecture that efficiently computes the attention scores, leveraging Verilog for its implementation. The project involves three main computational stages:

- forming the query, key, and value matrices by multiplying input embeddings with weight matrices,
- computing the score matrix through the multiplication of the query and the transposed key matrices,
- obtaining the scaled dot-product attention output by multiplying the score matrix with the value matrix.

The design is verified using a testbench that handles the communication and synchronization of data via SRAM interfaces, considering timing constraints and ensuring accuracy. The synthesized design is optimized for performance and verified for correctness through Synopsys and ModelSim tools.

Report Structure: The report proceeds as under:

1. Introduction

The module performs three main operations:

- **Matrix Multiplications:** Compute query *(Q),* key *(K),* and value *(V)* matrices by multiplying input embeddings with corresponding weight matrices.
- **Score Calculation:** Multiply the query matrix (Q) with the transposed key matrix (K^T) to produce the attention scores.
- **Scaled Dot-Product Attention:** Compute the final output by multiplying the score matrix with the value matrix (V).

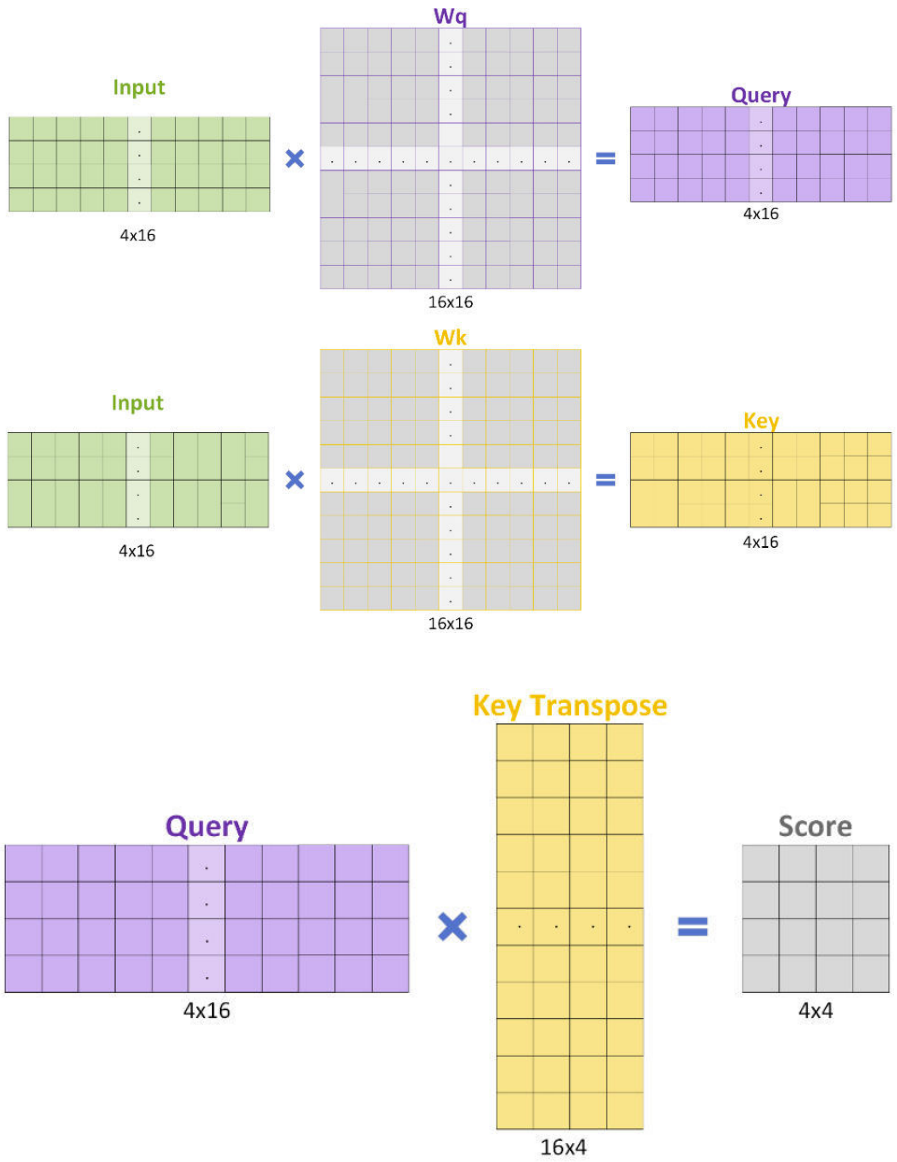


Figure 1: Calculation of Query, Key and Score matrices

To optimize performance, the design uses pipelining, allowing the overlapping of multiple computation stages to minimize latency. Additionally, parallel processing techniques accelerate matrix operations. Data is managed through an SRAM interface, ensuring efficient memory access and synchronization using control signals.

Key Features:

- **Pipelining:** Implements a multi-stage pipeline to enhance throughput and reduce overall computation time.
- **Parallelism:** Leverages parallel data paths to perform matrix operations efficiently.

2. Micro-Architecture:

2.1 Hardware "algorithmic" approach implemented:

2.1.1 Matrix Multiply:

- In order to perform matrix multiplication of two matrices, three counters are used, namely counter "i", "j" and "k".
- "Counter i" keeps track of no. of rows of input matrix. "Counter j" keeps track of no. of columns of weight matrix and "counter k" keeps track of no. of rows of weight matrix.
- All the three counter are down-counters. And they are initialized as: counter i = rows_of_input_matrix; counter j = col_of_weight_matrix; counter k = rows_of_weight_matrix.
- When the SRAM produces required matrix elements, the counters start decrementing.
- This approach is similar to matrix multiplication algorithm in high level languages like C++ i.e.

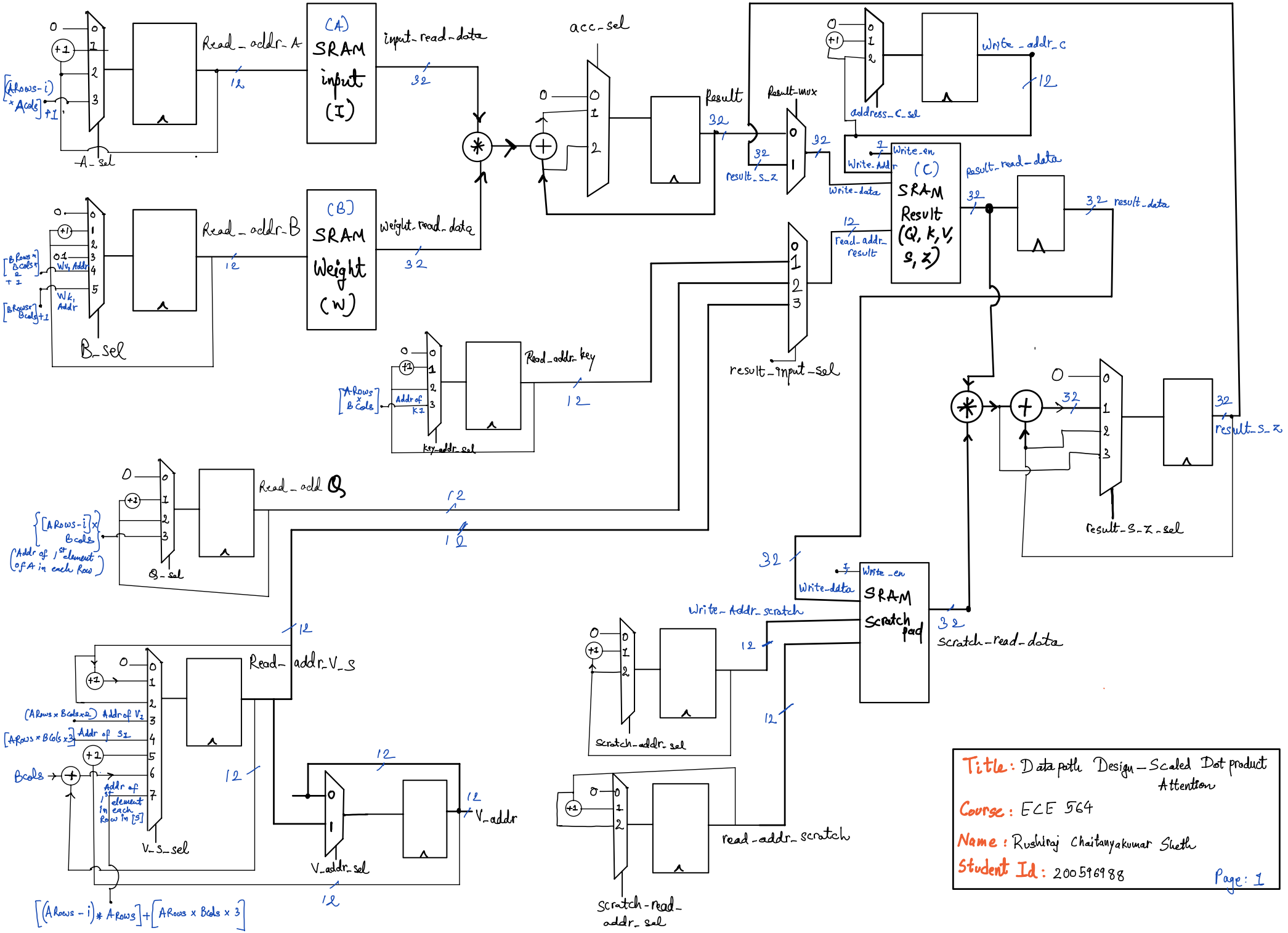
```
for(count i :1 to A_rows ){
        for(count j:1 to B_cols ){
                for(count k:1 to B_rows)
                        {       result[i][j] += (input[i][k]*weight[k][j]);
}}}
```

- Finally, since there are three weight matrices, the above multiplication is repeated three times i.e. computing Query(Q), Key(K) and value(V) matrix.
- "Counter c" is used to keep track of this.
- The computed elements of matrix Query, Key and Value are stored in result SRAM respectively.
- Further, to compute the score matrix, the elements of Key matrix are read from Result SRAM (in transpose order) and stored in scratchpad SRAM.
- Then, further the matrix multiplication is performed between elements of Query matrix (from result SRAM) and that of key matrix (read from scratchpad SRAM) and the convolution result as **Score matrix,** is stored in Result SRAM at address next to last stored value in it i.e. after elements of value matrix.
- Finally, the matrix V is stored in scratchpad SRAM in column major order and then convolution is performed between matrix S (read from result SRAM) and matrix V (read from Scratchpad SRAM in col. major order) and the result as **Attention matrix** is stored in result SRAM.

2.1.2 Transpose of matrix:

- In order to compute the transpose of matrix, the elements are stored in scratchpad SRAM in column major order and then referenced further.

2.1.3 Datapath design:

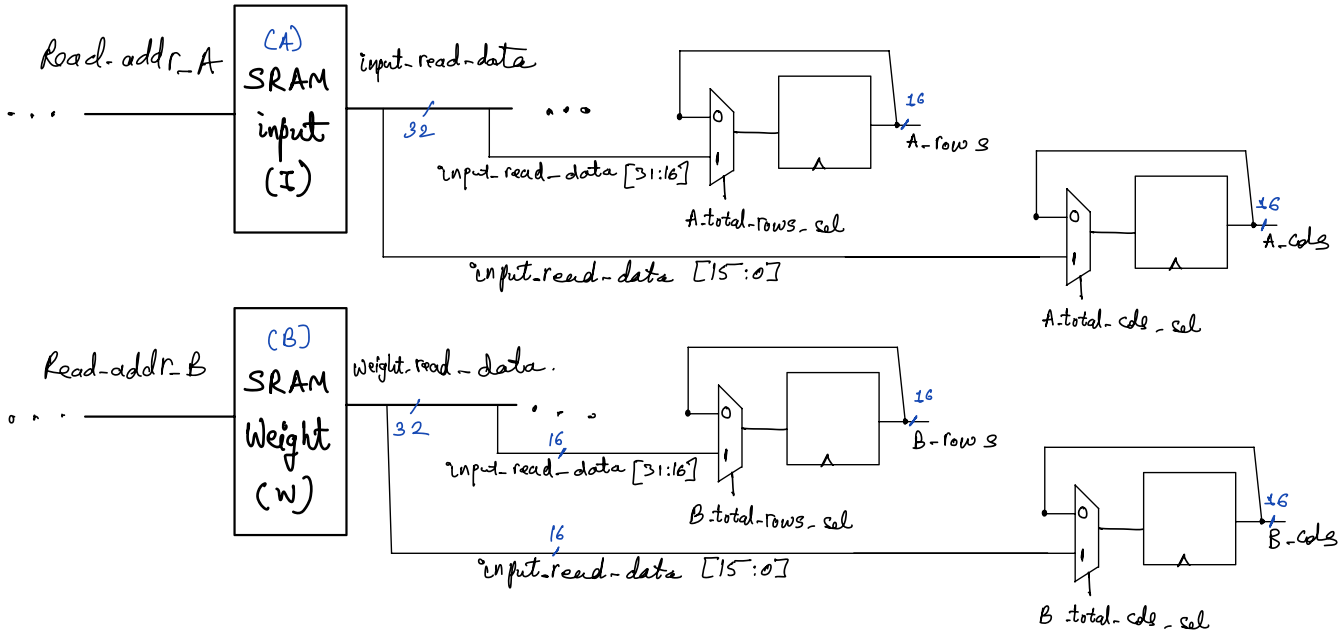


Title: Data path Design – Scaled Dot product Attention

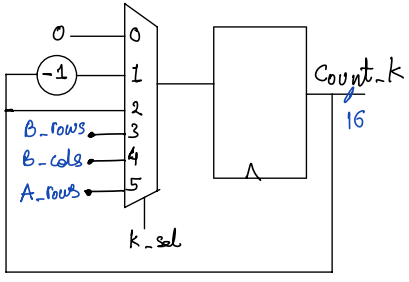
Course: ECE 564

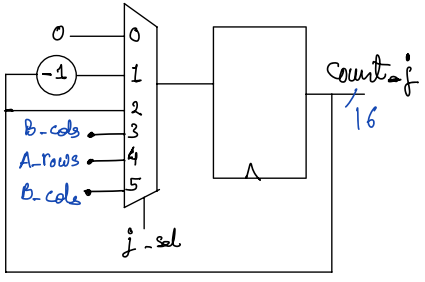
Name: Rushiraj Chaitanyakumar Sheth

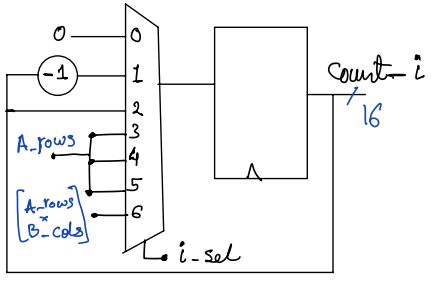
Student Id: 20059688

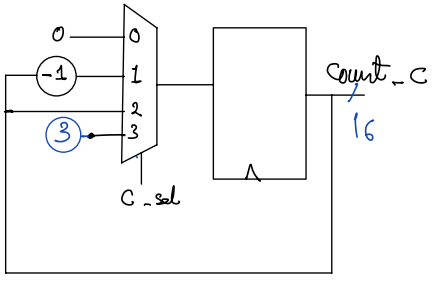


✱ Counters:









Title: Data path Design – Scaled Dot product Attention

Course: ECE 564

Name: Rushiraj Chaitanyakumar Sheth

Student Id: 20059698

3. Interface Specification

3.1 List of signals, their widths and functions:

Sr. no.	Signal Name	Width(bits)	Function
1	A_sel	2	generates read address for input SRAM
2	B_sel	3	generates read address for weight SRAM
3	acc_sel	2	used for accumulating and storing the convolution result of input and weight matrices.
4	address_c_sel	2	generates write address for result SRAM
5	result_mux	1	0: store convolution data of input and weight matrices i.e. Q, K, V to result SRAM. 1: store convolution data of ($Q*K^T = S$) score and ($S*V = Z$) attention matrices to result SRAM.
6	key_addr_sel	2	generates read address for key matrix stored in Result SRAM.
7	Q_sel	2	Generates read address of Q matrix stored in result SRAM.
8	v_s_sel	3	Generates read address of matrix S and matrix V, both of which are stored in Result SRAM.
9	result_input_sel	2	Value 1: connects read address of key matrix to read_addr of Result SRAM. Value 2: connects read address of Query/Q matrix to read_addr of Result SRAM. Value 3: connects read address of value(V) and score(S) matrix to read_addr of Result SRAM.
10	scratch_addr_sel	2	Generates write address for scratchpad SRAM.
11	scratch_read_addr_sel	2	Generates read address for scratchpad SRAM.
12	result_s_z_sel	2	used for accumulating and storing the convolution result of score (s) and attention (z) matrices.

13	k_sel	3	Counter k as count down counter (usage described in 2.3)
14	j_sel	3	Counter j as count down counter (usage described in 2.3)
15	i_sel	3	Counter i as count down counter (usage described in 2.3)
16	c_sel	3	Counter c as count down counter (usage described in 2.3)
17	A_total_rows_sel	1	Used to store total rows of input matrix.
18	A_total_cols_sel	1	Used to store total columns of input matrix.
19	B_total_rows_sel	1	Used to store total rows of weight matrix.
20	B_total_cols_sel	1	Used to store total columns of weight matrix.

3.2 Timing Diagram:

- The timing diagram obtained after successful completion of all test cases is as below.
- Below diagrams are related to testcase no. 1 only.

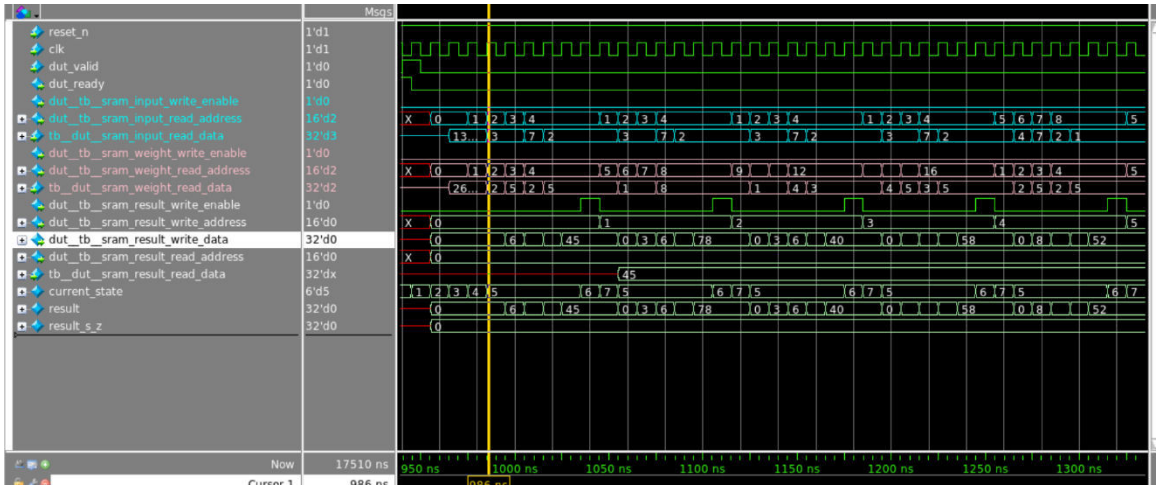


Figure 2: Test case 1 timing diagram

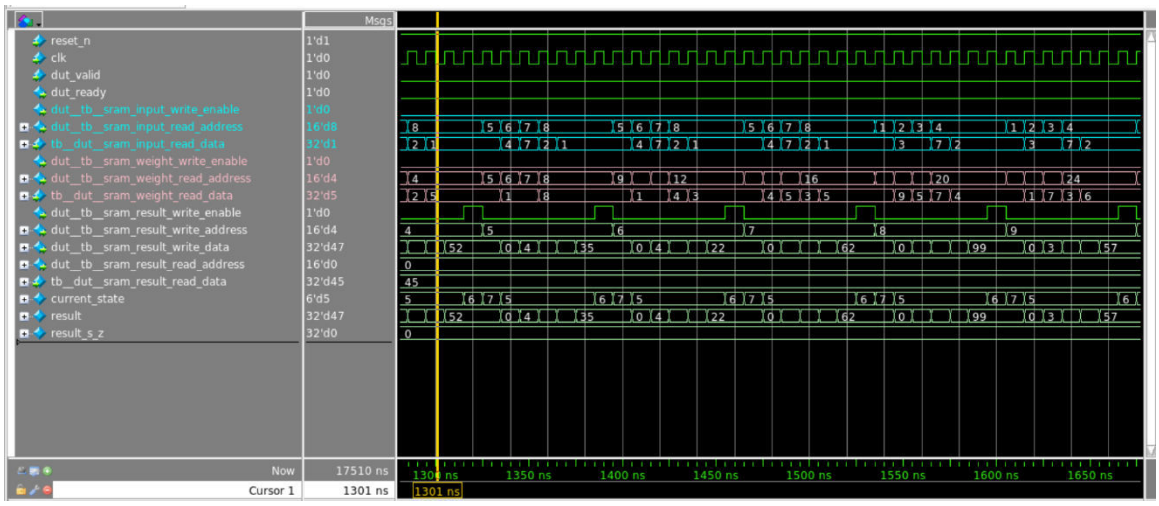


Figure 2: Test case 1 timing diagram

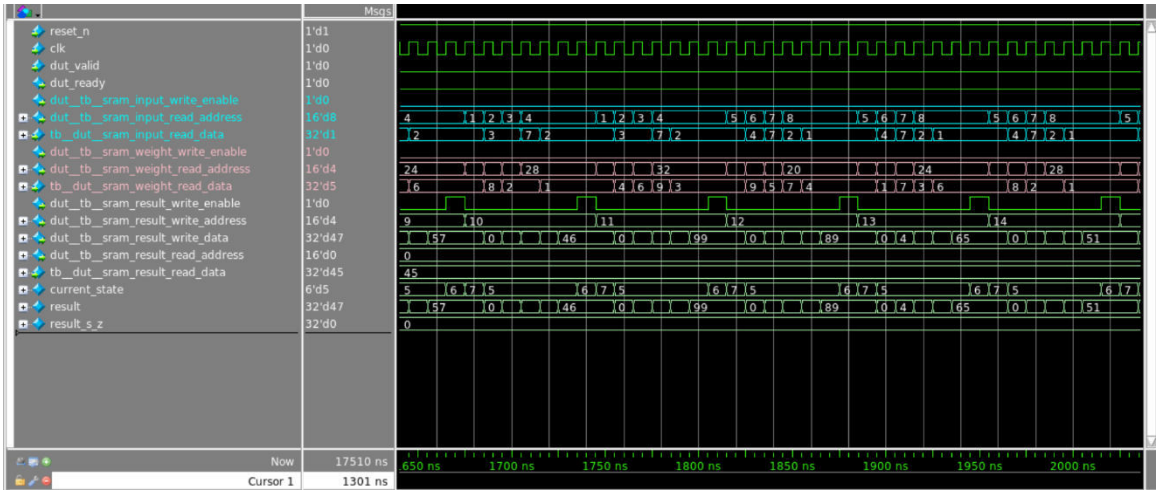


Figure 3: Test case 1 timing diagram

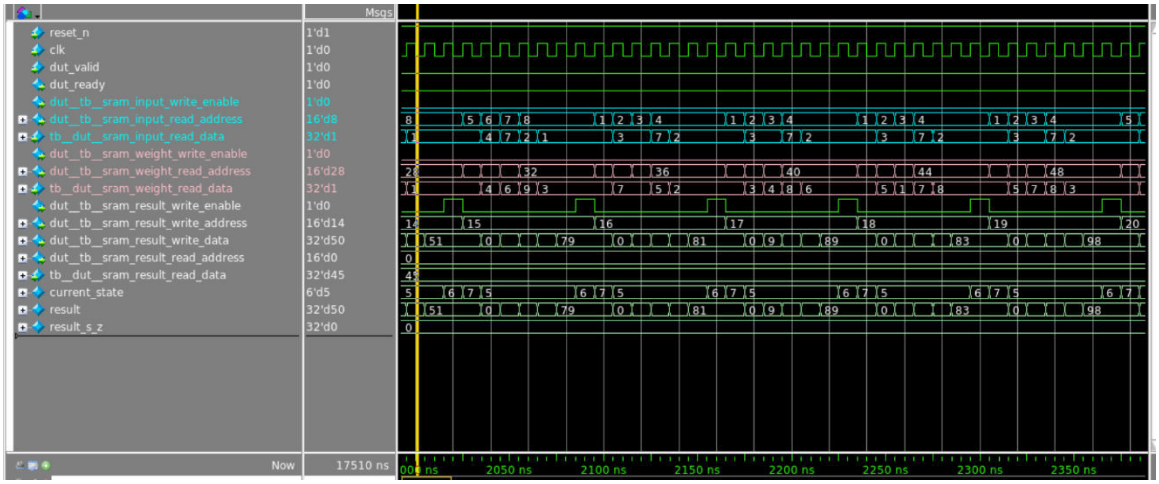


Figure 4: Test case 1 timing diagram

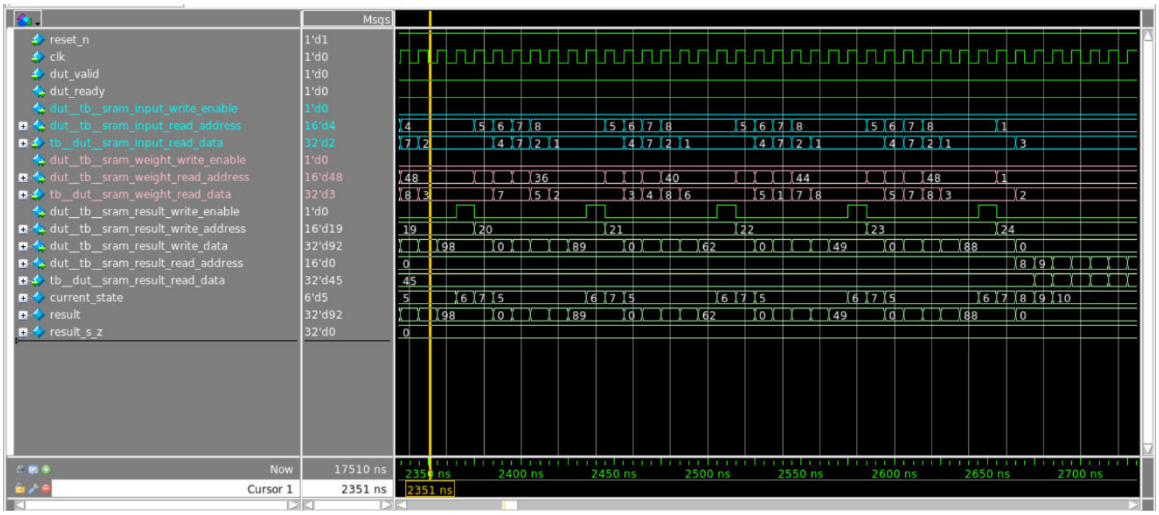


Figure 5: Test case 1 timing diagram

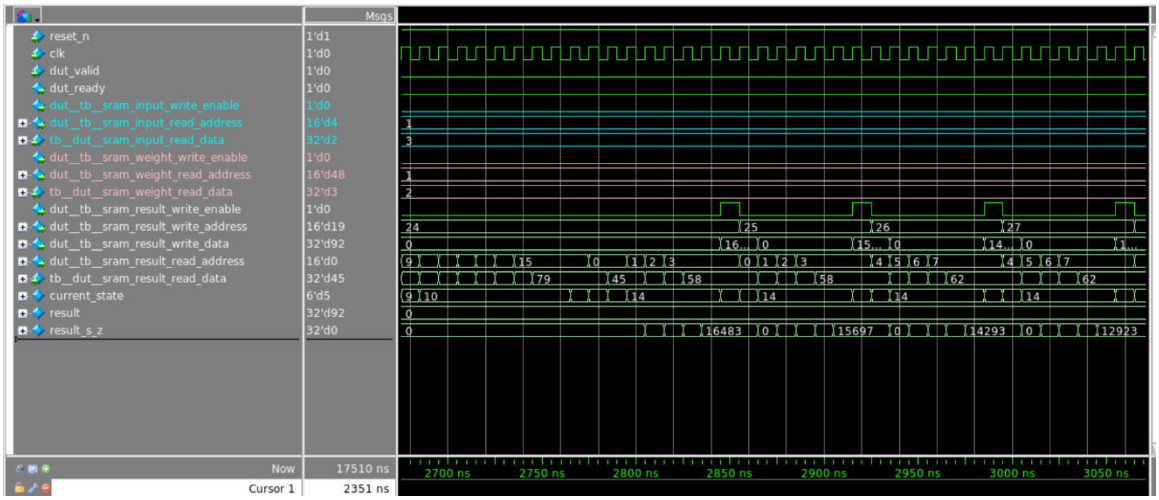


Figure 6: Test case 1 timing diagram

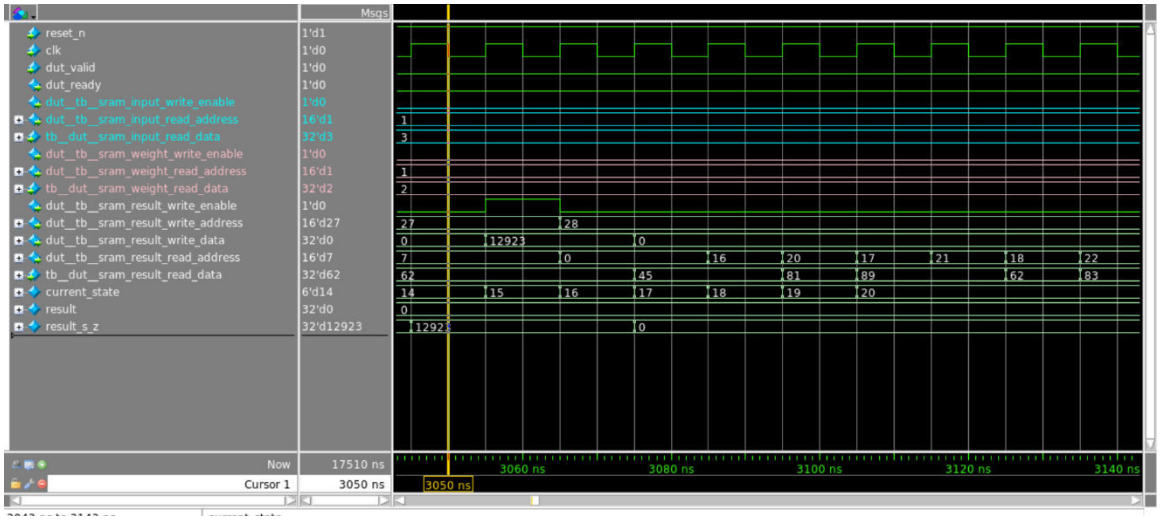


Figure 7: Test case 1 timing diagram

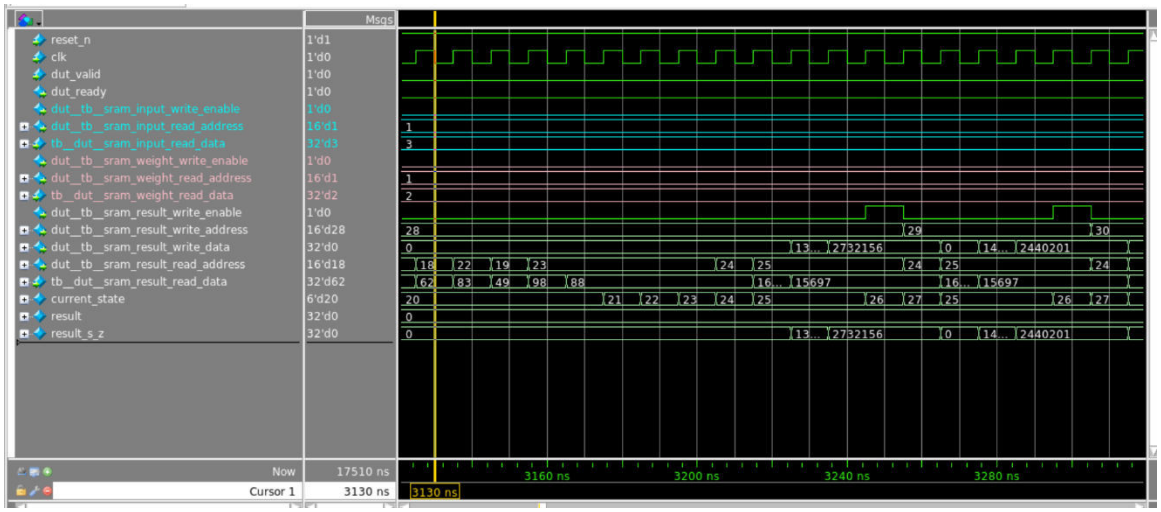


Figure 8: Test case 1 timing diagram

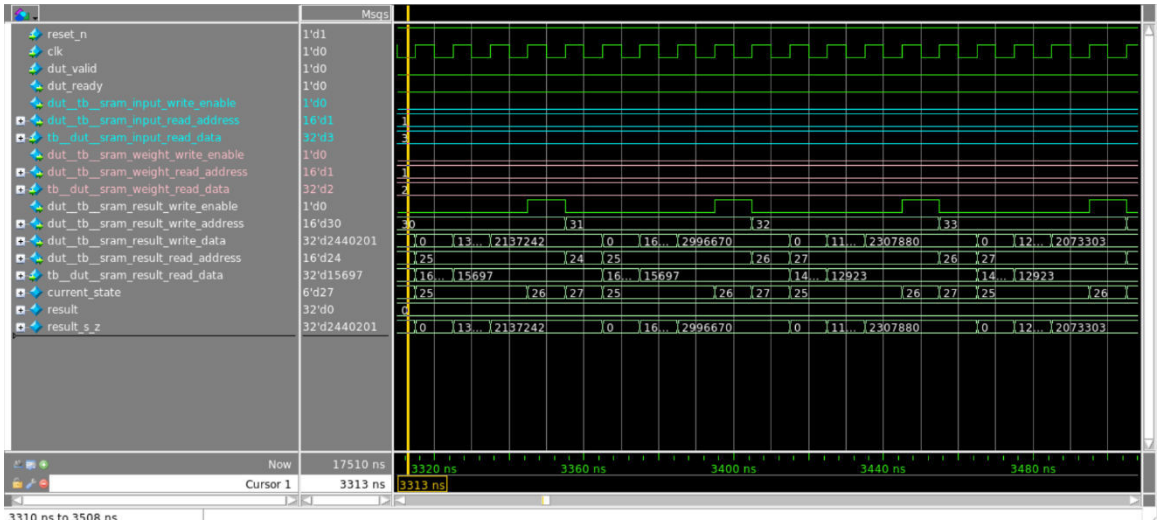


Figure 9: Test case 1 timing diagram

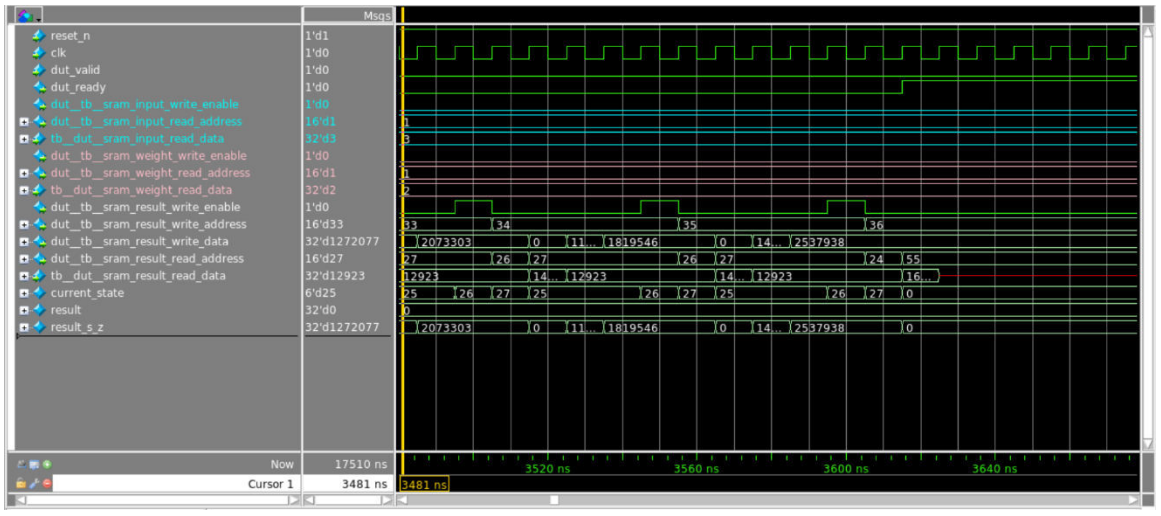


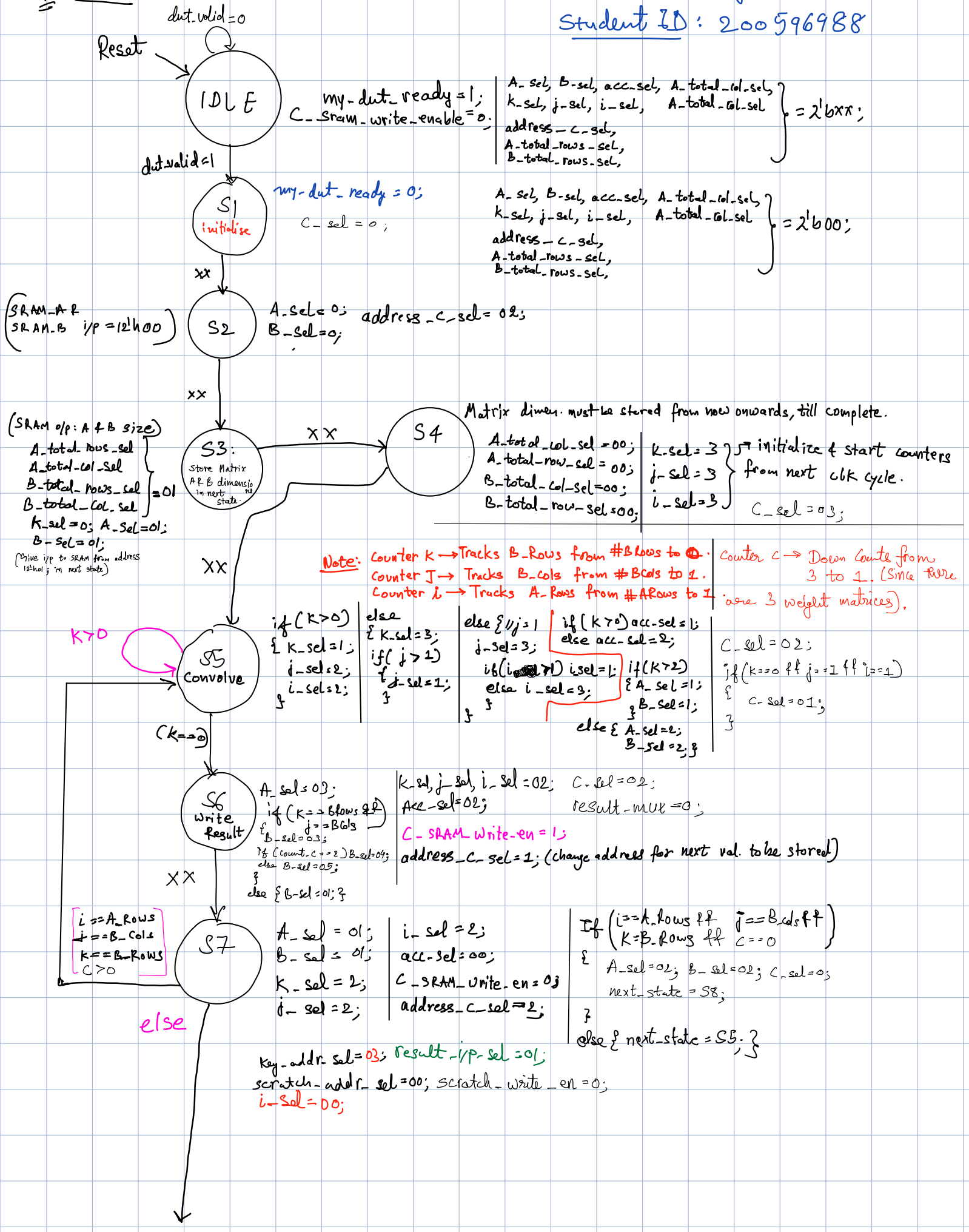
Figure 10: Test case 1 timing diagram

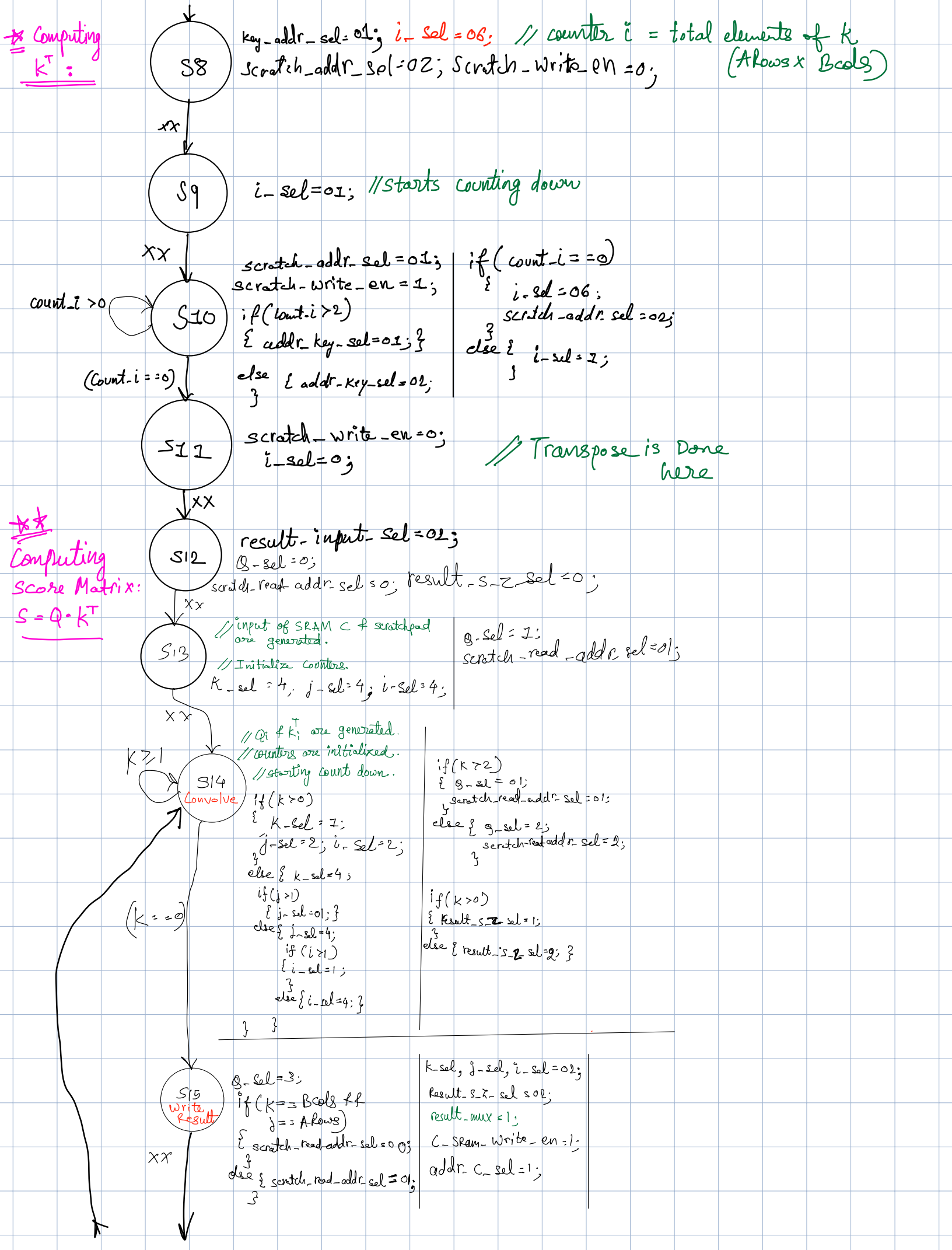
4. **Finite state machine design:**
    - The designed FSM based on Moore machine is as mentioned below.
    - It consists of 32 states.
    - For optimizing area consumption, binary encoding is used to describe each state in Verilog.

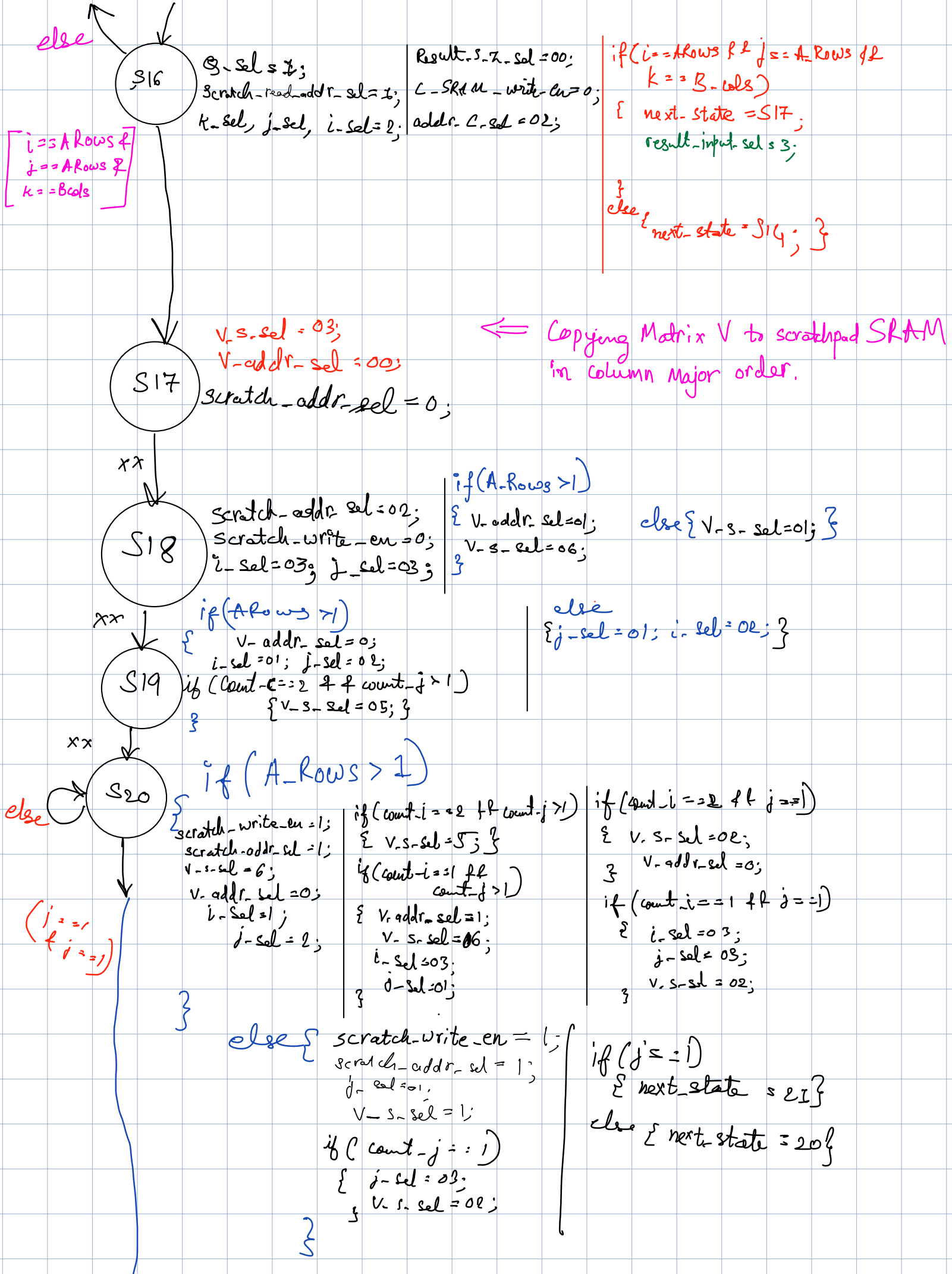
FSM:

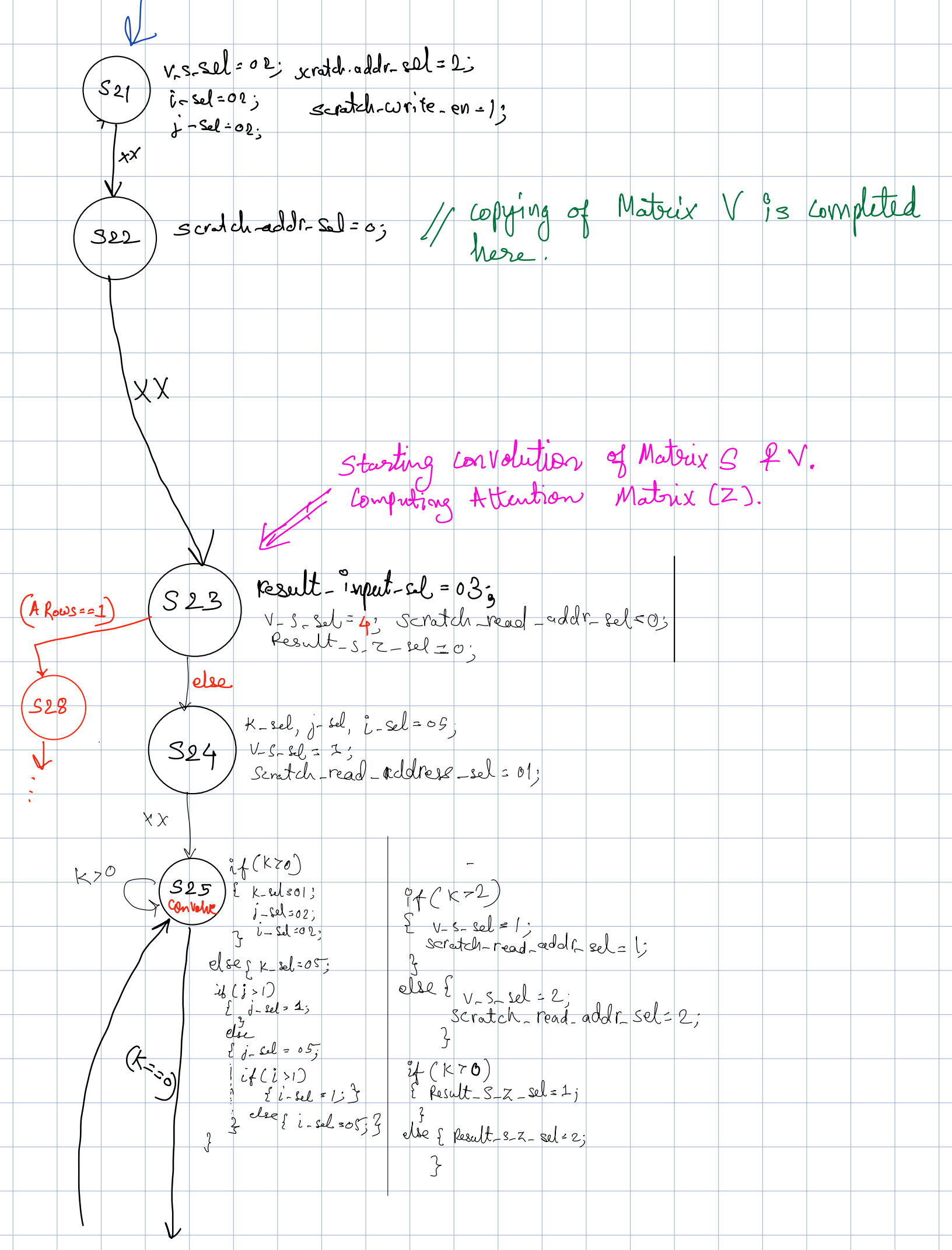
Name: Rushiraj Sheth

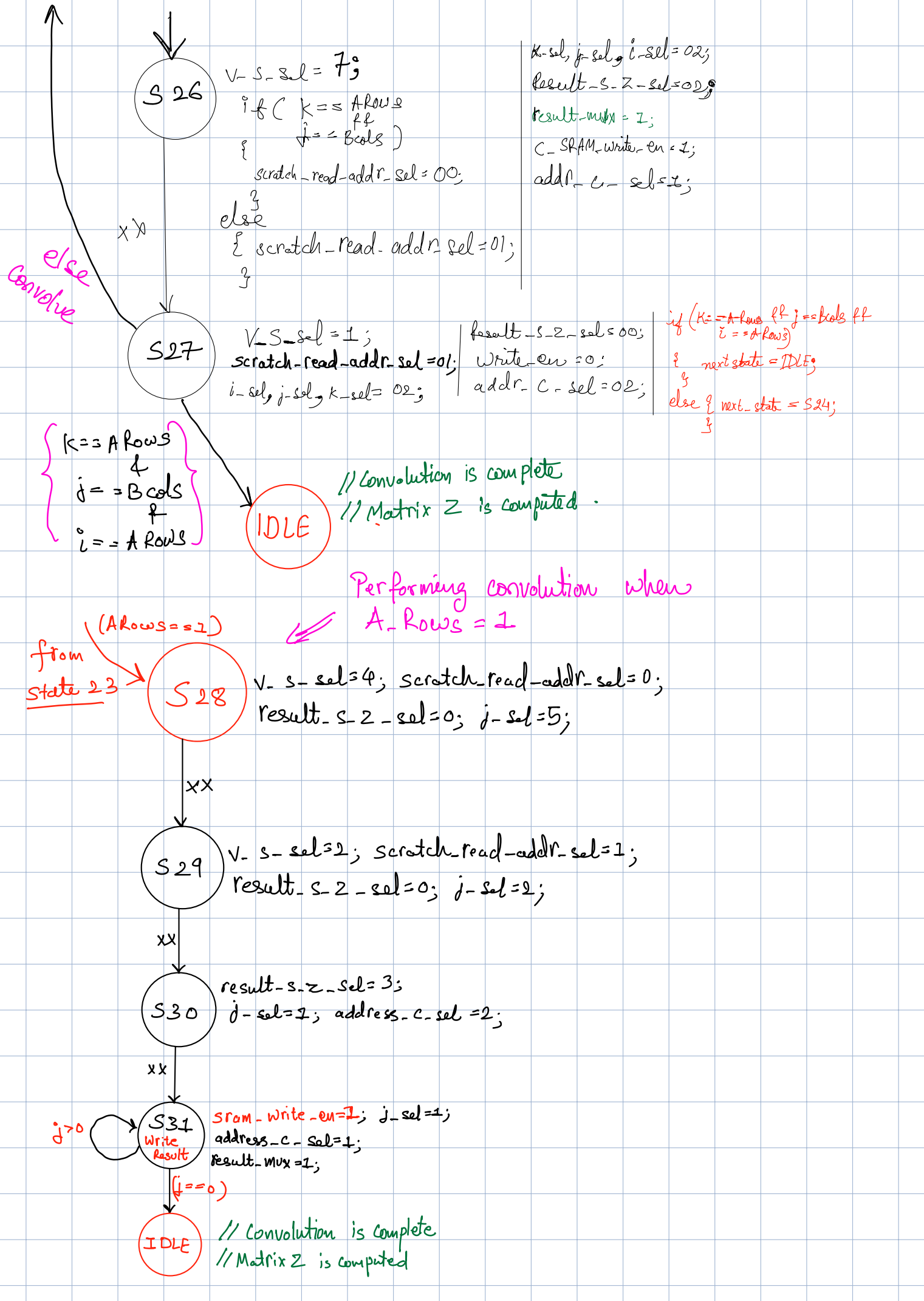
Student ID: 200596988




** Computing K^T :
S8
key_addr_sel=01; i_sel=06; // counter i = total elements of K (ARows x BCols)
scratch_addr_sel=02; scratch_write_en=0;
xx
S9
i_sel=01; //starts counting down
xx
S10
count_i >0
scratch_addr_sel=01;
scratch_write_en=1;
if(count_i>2)
{ addr_key_sel=01; }
else { addr_key_sel=02; }
if(count_i==0)
{ i_sel=06;
scratch_addr_sel=02; }
else { i_sel=1; }
(Count_i==0)
S11
scratch_write_en=0;
i_sel=0;
// Transpose is Done here
xx
** Computing Score Matrix: $S = Q \cdot K^T$
S12
result_input_sel=01;
Q_sel=0;
scratch_read_addr_sel=0; result_s_z_sel=0;
xx
S13
// input of SRAM C & scratchpad are generated.
// Initialize counters.
K_sel=4; j_sel=4; i_sel=4;
Q_sel=1;
scratch_read_addr_sel=01;
xx
S14
Convolve
K>=1
// Qi & K_i^T are generated.
// counters are initialized.
// starting count down.
if(K>0)
{ K_sel=1;
j_sel=2; i_sel=2; }
else { k_sel=4;
if(j>1)
{ j_sel=01; }
else { j_sel=4;
if(i>1)
{ i_sel=1; }
else { i_sel=4; } } }
if(K>2)
{ Q_sel=01;
scratch_read_addr_sel=01; }
else { Q_sel=2;
scratch_read_addr_sel=2; }
if(K>0)
{ Result_s_x_sel=1; }
else { result_s_z_sel=2; }
(K==0)
S15
Write Result
Q_sel=3;
if(K==BCols && j==ARows)
{ scratch_read_addr_sel=00; }
else { scratch_read_addr_sel=01; }
k_sel, j_sel, i_sel=02;
Result_s_x_sel=02;
result_mux=1;
C_sram_write_en=1;
addr_C_sel=1;
xx


else
S16
B_sel = 1;
scratch_read_addr_sel = 1;
k_sel, j_sel, i_sel = 2;
Result_s_2_sel = 00;
C_SRAM_write_en = 0;
addr_C_sel = 02;
if(i == A_Rows && j == A_Rows && k == B_cols)
{ next_state = S17;
result_input_sel = 3;
}
else { next_state = S16; }
i == A Rows && j == A Rows && k == B cols
S17
V_s_sel = 03;
V_addr_sel = 00;
scratch_addr_sel = 0;
Copying Matrix V to scratchpad SRAM in column Major order.
xx
S18
scratch_addr_sel = 02;
scratch_write_en = 0;
i_sel = 03; j_sel = 03;
if(A_Rows > 1)
{ V_addr_sel = 01;
V_s_sel = 06;
}
else { V_s_sel = 01; }
xx
S19
if(A_Rows > 1)
{ V_addr_sel = 0;
i_sel = 01; j_sel = 02;
if (Count_C == 2 && count_j > 1)
{ V_s_sel = 05; }
}
else
{ j_sel = 01; i_sel = 02; }
xx
S20
else
if (A_Rows > 1)
{ scratch_write_en = 1;
scratch_addr_sel = 1;
V_s_sel = 6;
V_addr_sel = 0;
i_sel = 1;
j_sel = 2;
if(count_i == 2 && count_j > 1)
{ V_s_sel = 5; }
if(count_i == 1 && count_j > 1)
{ V_addr_sel = 1;
V_s_sel = 06;
i_sel = 03;
j_sel = 01;
}
if(count_i == 2 && j == 1)
{ V_s_sel = 02;
V_addr_sel = 0;
}
if(count_i == 1 && j == 1)
{ i_sel = 03;
j_sel = 03;
V_s_sel = 02;
}
}
else { scratch_write_en = 1;
scratch_addr_sel = 1;
j_sel = 01;
V_s_sel = 1;
if (count_j == 1)
{ j_sel = 03;
V_s_sel = 02;
}
}
if (j == 1)
{ next_state = 21 }
else { next_state = 20 }
(j == 1 && i == 1)


S21
V_s_sel = 02; scratch_addr_sel = 2;
i_sel = 02; scratch_write_en = 1;
j_sel = 02;
xx
S22
scratch_addr_sel = 0;
// copying of Matrix V is completed here.
XX
Starting convolution of Matrix S & V.
Computing Attention Matrix (Z).
S23
Result_input_sel = 03;
V_s_sel = 4; scratch_read_addr_sel = 0;
Result_s_z_sel = 0;
(A Rows == 1)
else
S28
S24
K_sel, j_sel, i_sel = 05;
V_s_sel = 1;
Scratch_read_address_sel = 01;
XX
K>0
S25
Convolve
if(K>0)
{ K_sel=01;
j_sel=02;
i_sel=02;
}
else { K_sel=05;
if(j>1)
{ j_sel = 1;
}
else
{ j_sel = 05;
if(i>1)
{ i_sel = 1; }
else { i_sel=05; }
}
}
if(K==0)
if(K>2)
{ V_s_sel = 1;
scratch_read_addr_sel = 1;
}
else { V_s_sel = 2;
scratch_read_addr_sel = 2;
}
if(K>0)
{ Result_s_z_sel = 1;
}
else { Result_s_z_sel = 2;
}


S26
V_S_sel = 7;
if(k==A Rows && j==Bcols)
{ scratch_read_addr_sel = 00;
}
else
{ scratch_read_addr_sel = 01;
}
k_sel, j_sel, i_sel = 02;
result_s_2_sel = 00;
result_mux = 1;
C_SRAM_write_en = 1;
addr_c_sel = 1;
xx
else Convolve
S27
V_S_sel = 1;
scratch_read_addr_sel = 01;
i_sel, j_sel, k_sel = 02;
result_s_2_sel = 00;
write_en = 0;
addr_c_sel = 02;
if (k==A Rows && j==Bcols && i==A Rows)
{ next state = IDLE;
}
else { next_state = S24;
}
k==A Rows & j==Bcols & i==A Rows
IDLE
// Convolution is complete
// Matrix Z is computed.
Performing convolution when A_Rows = 1
(ARows==1)
from state 23
S28
V_s_sel=4; scratch_read_addr_sel=0;
result_s_2_sel=0; j_sel=5;
xx
S29
V_s_sel=2; scratch_read_addr_sel=1;
result_s_2_sel=0; j_sel=2;
xx
S30
result_s_2_sel=3;
j_sel=1; address_c_sel=2;
xx
S31
Write Result
j>0
sram_write_en=1; j_sel=1;
address_c_sel=1;
result_mux=1;
(j==0)
IDLE
// Convolution is complete
// Matrix Z is computed

5. Results Achieved

- **Clock period:** 5.80 ns
- **# cycles":** 3354
- **Delay (clock period * #cycles):** 19453.2 ns
- **Area (um^2):** 12866.6859 um^2
- **1/(delay*area) $ns^{-1}.um^{-2}$:** 4 x 10^{-9} $ns^{-1}.um^{-2}$
- **Timing constraints:** All constraints MET.

6. Conclusions

- In this project, I understood the concept of self attention in LLM transformers and also implemented the RTL design using Verilog (2001) to realize the scaled dot product attention.
- The Finite state machine working as per Moore machine is implemented using binary encoding and consists of 32 states.
- Concept of pipelining is implemented for minimizing delay and efficient computation.
- Successfully synthesized the design using Synopsys with no unintentional latches, meeting all the timing constraints, optimal clock period and optimized area.