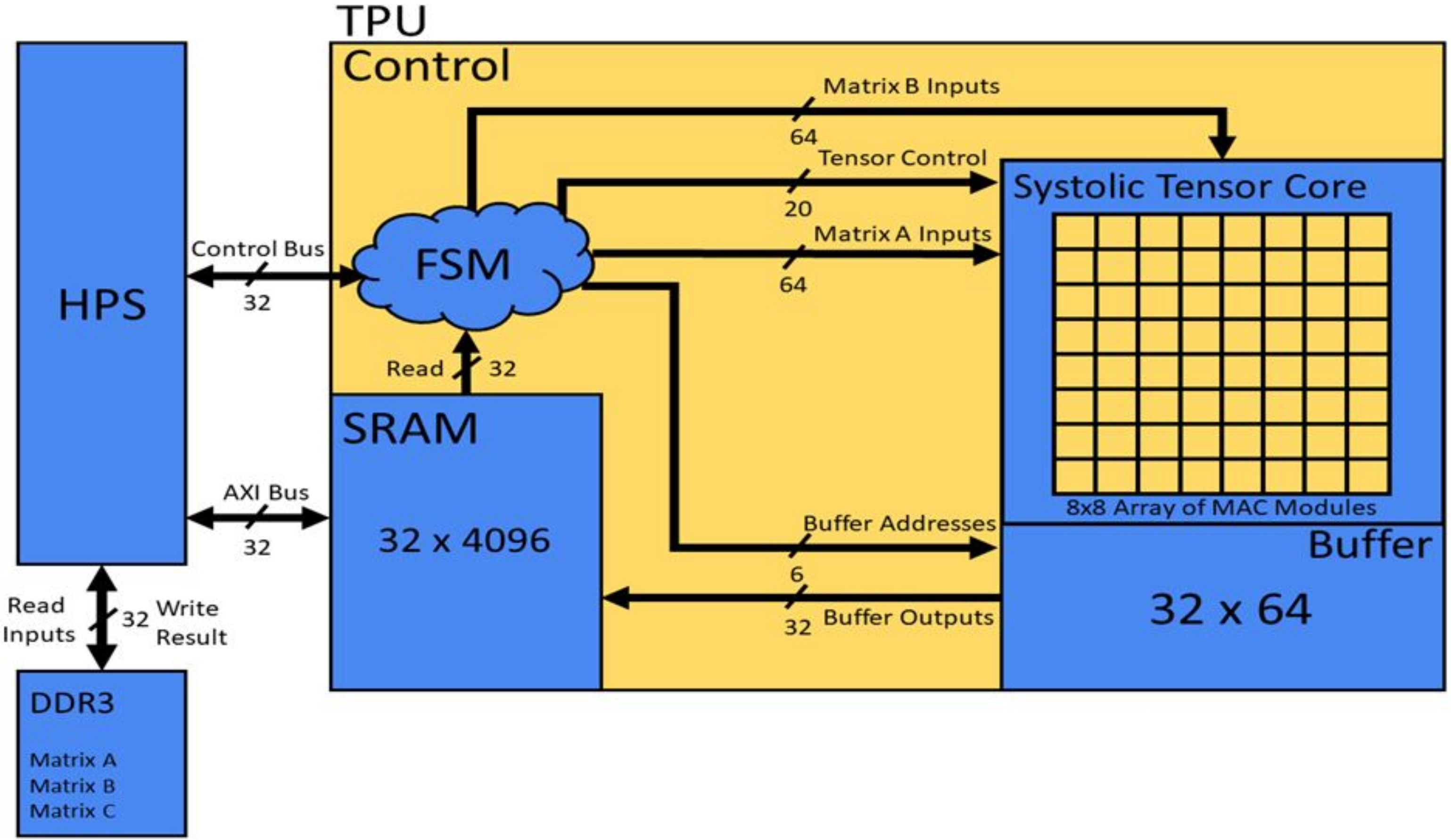
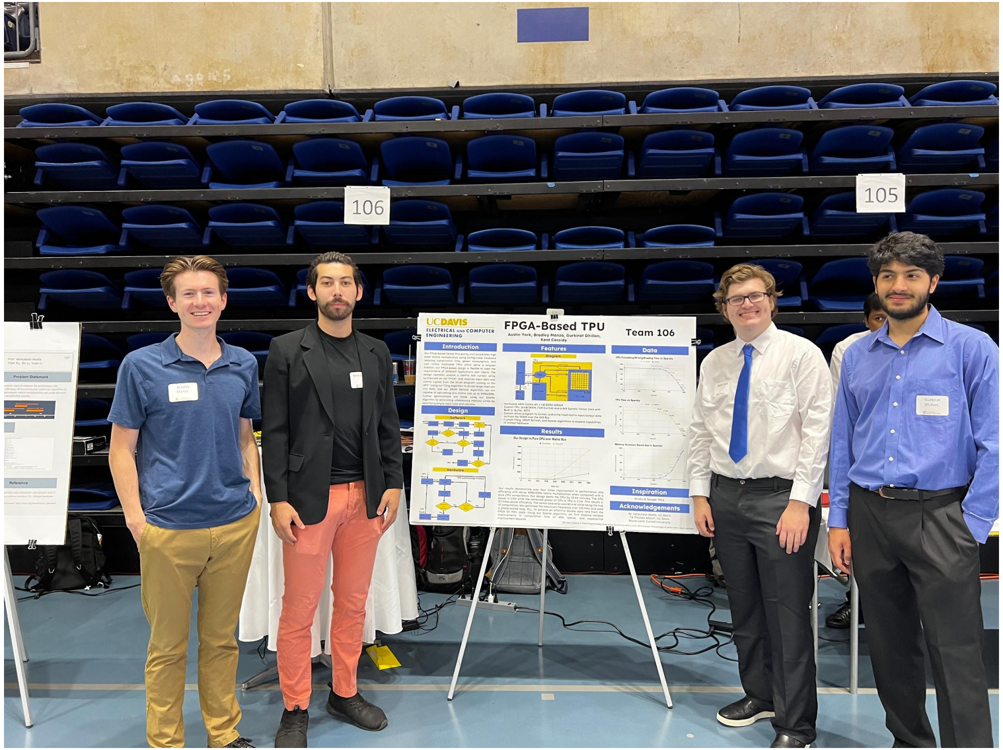

TPU
Control
Matrix B Inputs
64
Tensor Control
20
Systolic Tensor Core
Matrix A Inputs
64
FSM
Control Bus
32
HPS
Read
32
SRAM
AXI Bus
32
32 x 4096
8x8 Array of MAC Modules
Buffer Addresses
6
Buffer
32
Buffer Outputs
32 x 64
Read
Inputs
32
Write
Result
DDR3
Matrix A
Matrix B
Matrix C



EEC 181 TPU Design Group

- Bradley Manzo
  - Computer Engineer
  - Data Orchestration, Benchmarking, Software Design, Presentation Design
- Austin York
  - Electrical Engineer
  - Hardware and Protocol Design
- Max Dhillon
  - Electrical Engineer
  - FIFO Design
- Kent Cassidy
  - Computer Engineer
  - Demo and Validation design

Goals of project

Assigned by Professor Ventakesh Akella

- 4096x4096 Matrix Multiplication
- Sparse MM Functionality
- Benchmark against software on comparable hardware
- Awarded Outstanding Senior Design for our class

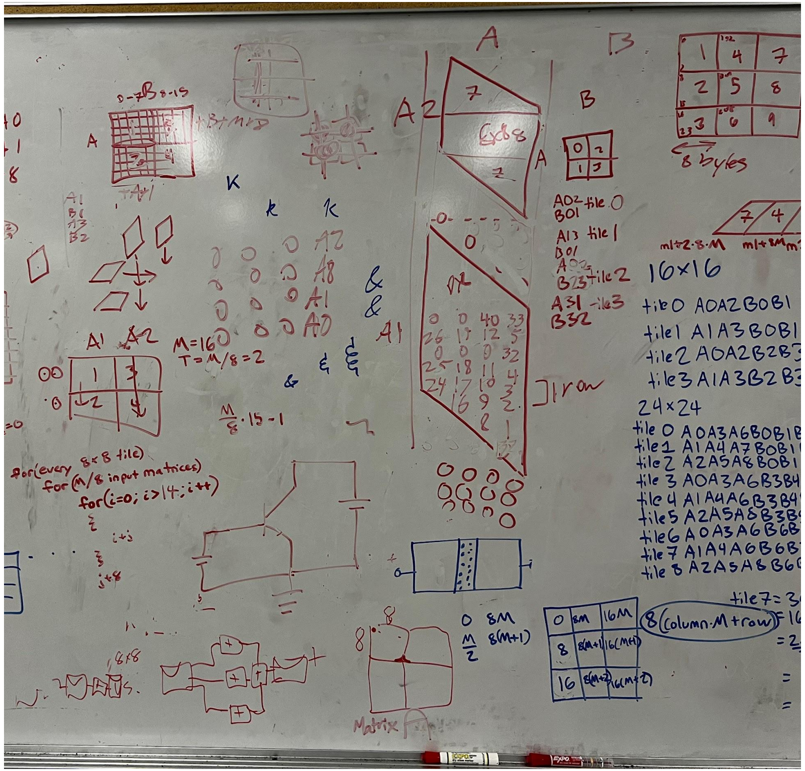
Resources

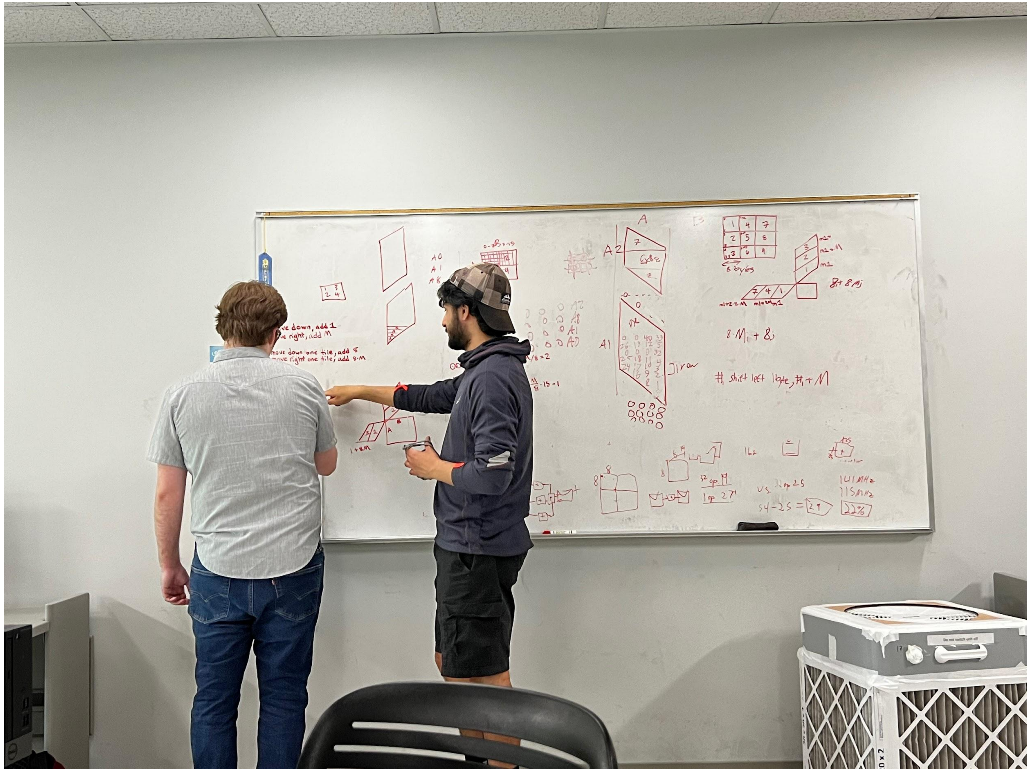
- Google TPU (2015)
- Cornell documentation on DMA for DE1-SoC

Challenges and Limitations

- Lack of experience/Knowledge in ML or DL Architecture/Design
- Limited to DE1-SoC Cyclone V FPGA
    - 32b bridge to communicate between HPS and FPGA
    - 128kB shared SRAM
    - Limited hardware, maximum Systolic array size: 8x8
    - Adder Carry Ripple Length

Teamwork (Software-Hardware Codesign)





Timeline

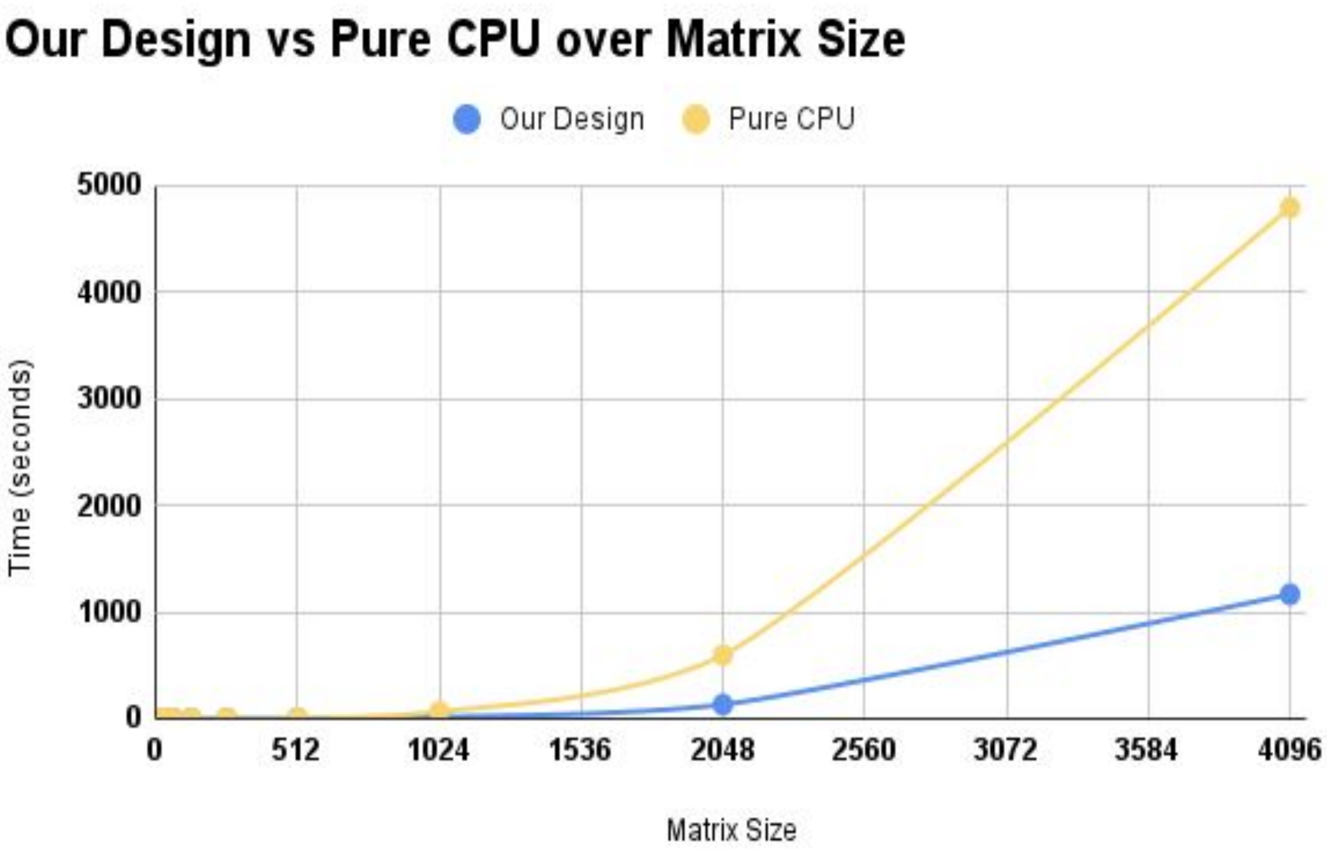
Winter Quarter

- Week 4 - Successfully wrote from program to shared SRAM
- Week 5 - FPGA operated on data sent to Shared SRAM
- Week 7 - Systolic Array working in Test bench
- Week 8 - Data Orchestration printing in correct order
- Week 9 - Single 8x8 functioning on TPU

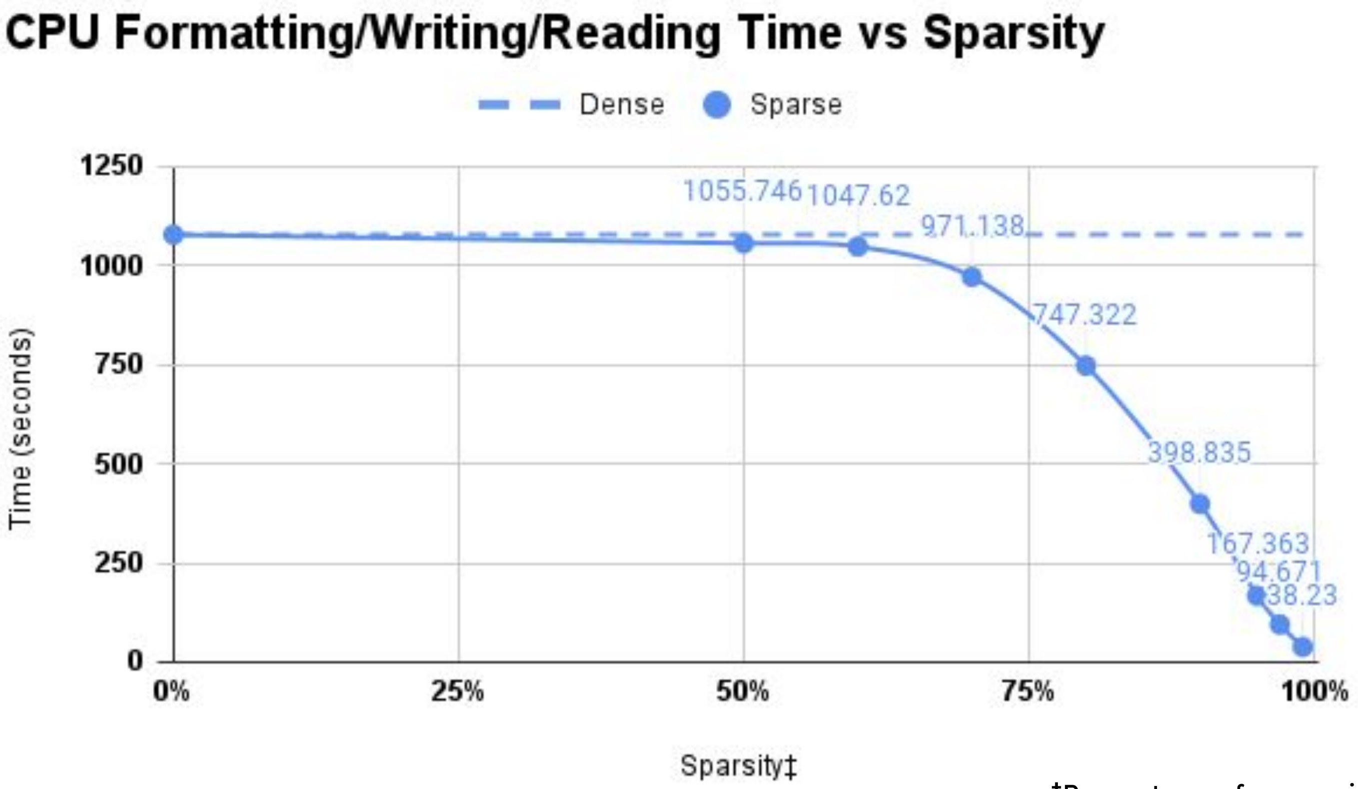
Spring Quarter

- Week 6 - Completed Tiling Algorithm
- Week 7 - Implemented Tiling and added supporting commands for TPU
- Week 8 - Completed Sparse Algorithm
- Week 10 - Completed SRAM algorithm for <4096x4096 MM

Results

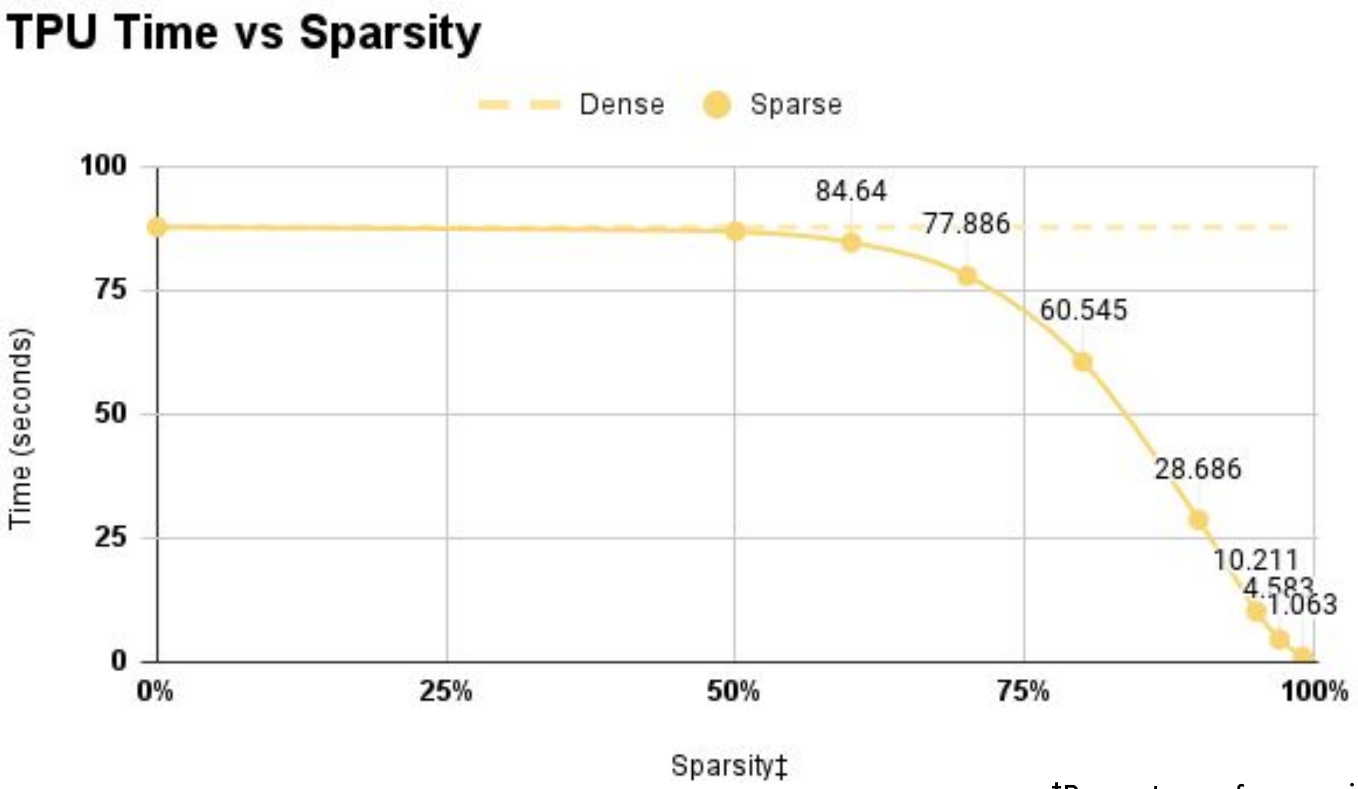

Our Design vs Pure CPU over Matrix Size
● Our Design
● Pure CPU
5000
4000
3000
2000
1000
0
Time (seconds)
0
512
1024
1536
2048
2560
3072
3584
4096
Matrix Size

Results



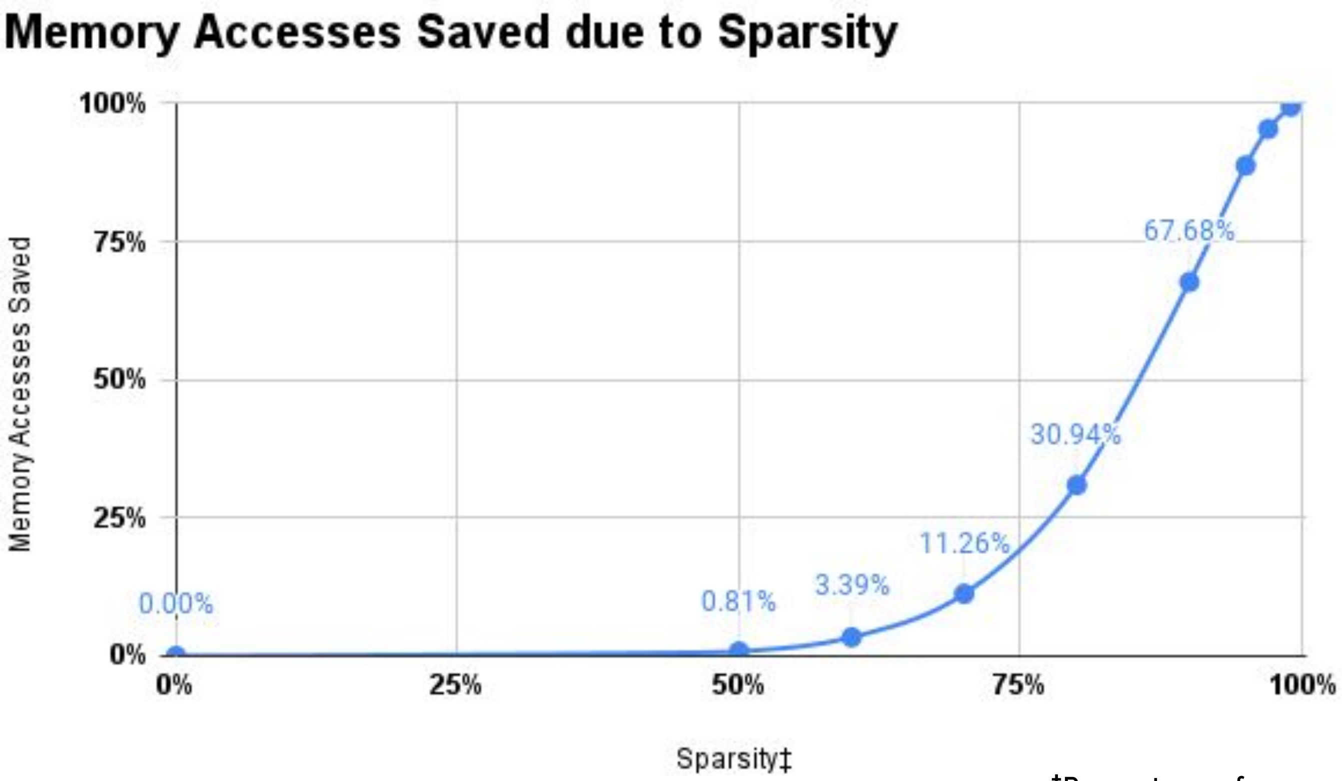
‡Percentage of zeros within input matrices

Results



‡Percentage of zeros within input matrices

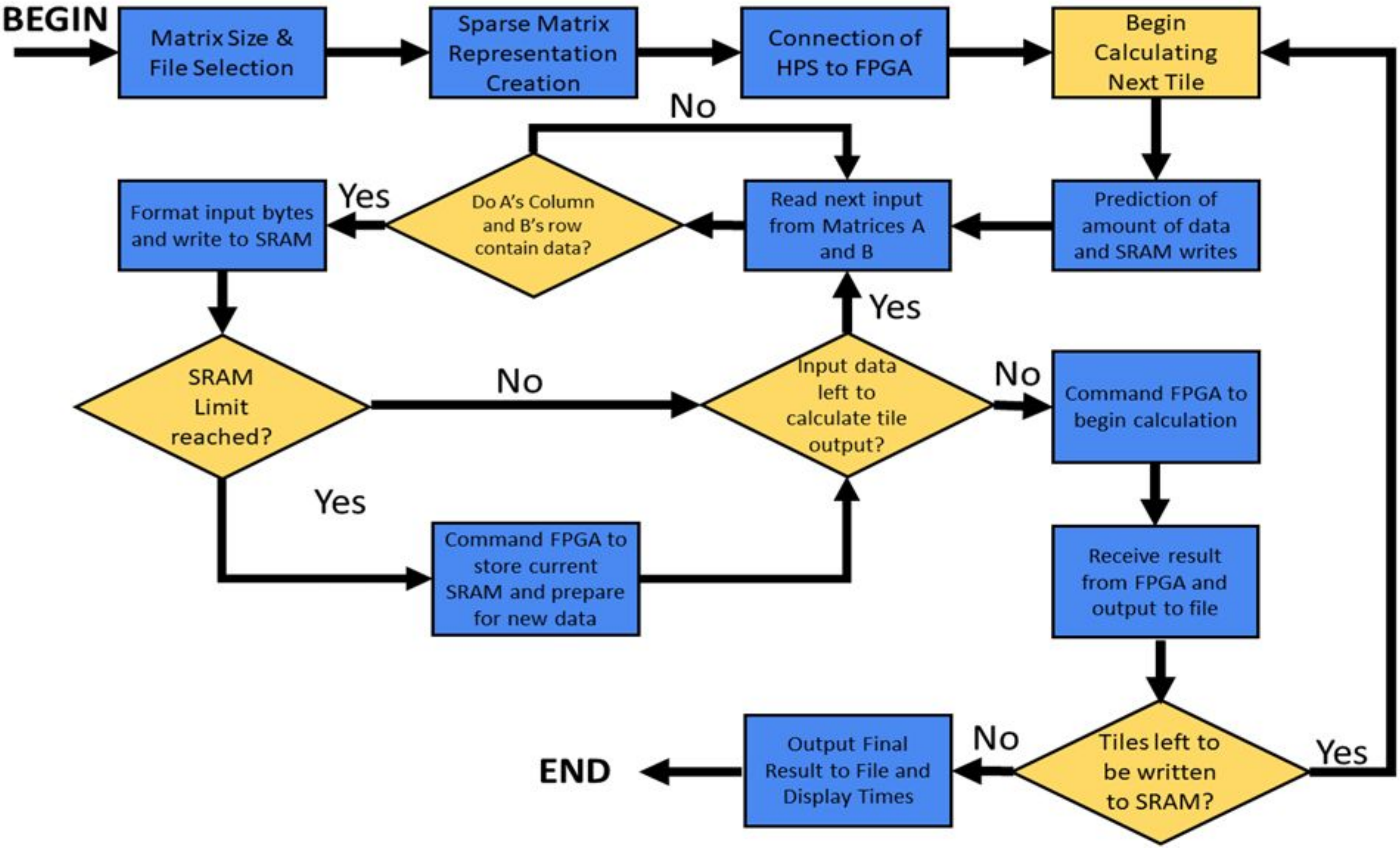
Results



‡Percentage of zeros within input matrices

Future Goals

- Software Goals
  - Convolution
  - Parallel Threading
  - Parameterize and Package Software as part of library function
- Hardware Goals
  - FIFOs to pipeline HPS and TPU, instead of SRAM
  - No data reuse -> Cache columns of Matrix B to reuse and reduce bandwidth
  - **Store column of Matrix B in SRAM, FIFO Row of Matrix A**
  - **Table Tile Partial Products and reuse instead of raw data**
  - Accumulator Trees
  - FPGA SDRAM Controller
  - Decrease overall memory transitions using MEM blocks
  - Reduce FPGA Logic and Interconnects



BEGIN

Matrix Size & File Selection

Sparse Matrix Representation Creation

Connection of HPS to FPGA

Begin Calculating Next Tile

Prediction of amount of data and SRAM writes

Read next input from Matrices A and B

Do A's Column and B's row contain data?

No

Yes

Format input bytes and write to SRAM

SRAM Limit reached?

No

Yes

Command FPGA to store current SRAM and prepare for new data

Input data left to calculate tile output?

Yes

No

Command FPGA to begin calculation

Receive result from FPGA and output to file

Tiles left to be written to SRAM?

Yes

No

Output Final Result to File and Display Times

END

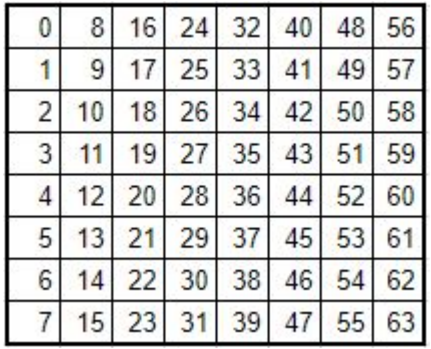
1. Accept Input Matrices from User

- User specifies dimensions (AH, AW, BH, BW)
    - If AW ≠ BH, invalid inputs
    - AH x BW is the size of the output matrix
- Input width and height must be multiples of 8
    - 8x8 systolic array requires 64 elements per tile
    - Solution: pad partial tiles with zeros (maintains original structure)
        - `#define M_size(M)       ((M % 8) ? (((M/8) + 1)*8) : (M))`
        - If a multiple of 8, no padding required
        - Otherwise, treat as next greatest multiple

1. Accept Input Matrices from User

AW = 8

AH = 8

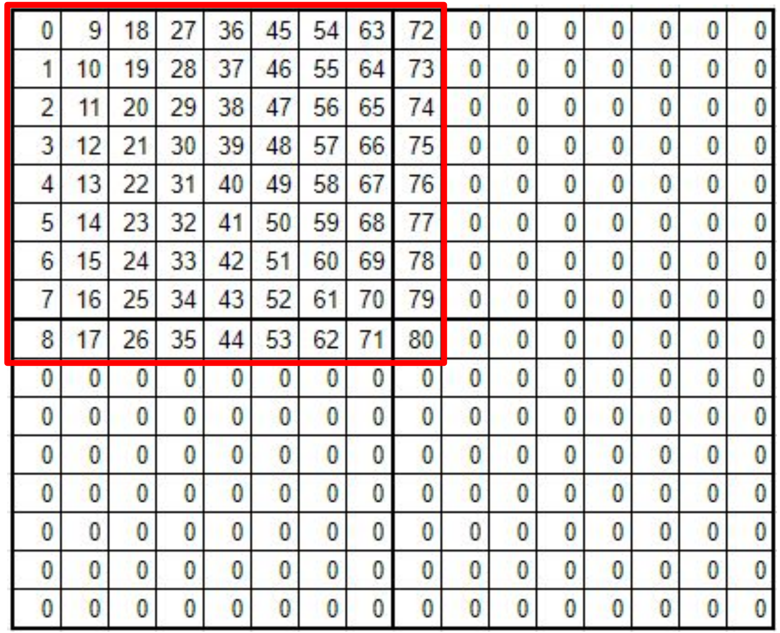


0	8	16	24	32	40	48	56
1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63

M_size(AH) = 8
M_size(AW) = 8
1x1 Tile

AW = 9

AH = 9



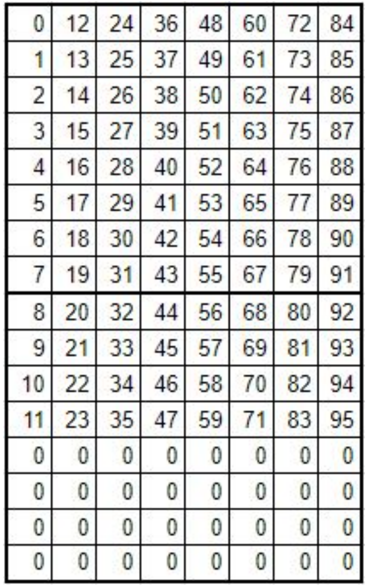
0	9	18	27	36	45	54	63	72	0	0	0	0	0	0	0
1	10	19	28	37	46	55	64	73	0	0	0	0	0	0	0
2	11	20	29	38	47	56	65	74	0	0	0	0	0	0	0
3	12	21	30	39	48	57	66	75	0	0	0	0	0	0	0
4	13	22	31	40	49	58	67	76	0	0	0	0	0	0	0
5	14	23	32	41	50	59	68	77	0	0	0	0	0	0	0
6	15	24	33	42	51	60	69	78	0	0	0	0	0	0	0
7	16	25	34	43	52	61	70	79	0	0	0	0	0	0	0
8	17	26	35	44	53	62	71	80	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

M_size(AH) = 16
M_size(AW) = 16
2x2 Tiles

1. Accept Input Matrices from User

AW = 8

AH = 12



0	12	24	36	48	60	72	84
1	13	25	37	49	61	73	85
2	14	26	38	50	62	74	86
3	15	27	39	51	63	75	87
4	16	28	40	52	64	76	88
5	17	29	41	53	65	77	89
6	18	30	42	54	66	78	90
7	19	31	43	55	67	79	91
8	20	32	44	56	68	80	92
9	21	33	45	57	69	81	93
10	22	34	46	58	70	82	94
11	23	35	47	59	71	83	95
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

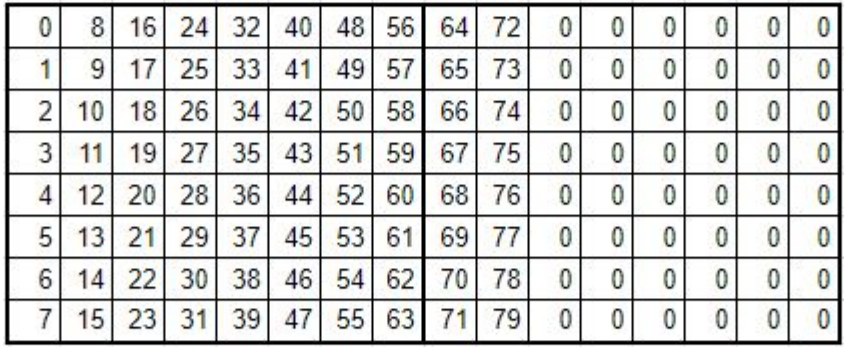
M_size(AH) = 16

M_size(AW) = 8

Tile Count = (16/8) * (8/8) = 2

AW = 10

AH = 8



0	8	16	24	32	40	48	56	64	72	0	0	0	0	0	0
1	9	17	25	33	41	49	57	65	73	0	0	0	0	0	0
2	10	18	26	34	42	50	58	66	74	0	0	0	0	0	0
3	11	19	27	35	43	51	59	67	75	0	0	0	0	0	0
4	12	20	28	36	44	52	60	68	76	0	0	0	0	0	0
5	13	21	29	37	45	53	61	69	77	0	0	0	0	0	0
6	14	22	30	38	46	54	62	70	78	0	0	0	0	0	0
7	15	23	31	39	47	55	63	71	79	0	0	0	0	0	0

M_size(AH) = 8

M_size(AW) = 16

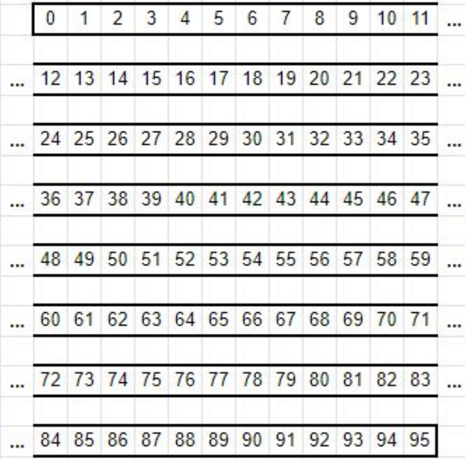
1x2 Tiles

Tile Count = (8/8) * (16/8)= 2

1. Accept Input Matrices from User

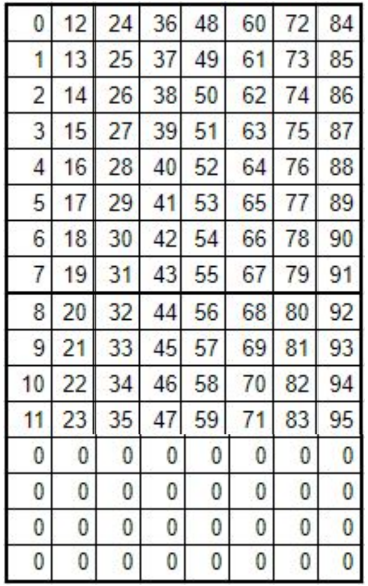
- Process:
  - Input Matrices are:
    - File Stream (Volatile 1D array)
    - Column-Major Order
    - Binary Format (.bin)
  - Call M_size() to determine padded dimensions
  - Initialize a 1D array of type uint_8 of these dimensions with zeros
    - (M_size(W)*M_size(H)) long
  - Read each column of input matrix from file, to the base of corresponding column in Array

1. Accept Input Matrices from User



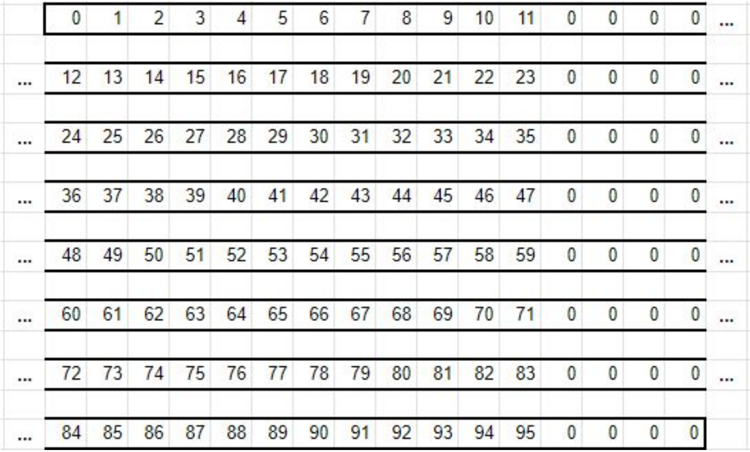
	0	1	2	3	4	5	6	7	8	9	10	11	...
...	12	13	14	15	16	17	18	19	20	21	22	23	...
...	24	25	26	27	28	29	30	31	32	33	34	35	...
...	36	37	38	39	40	41	42	43	44	45	46	47	...
...	48	49	50	51	52	53	54	55	56	57	58	59	...
...	60	61	62	63	64	65	66	67	68	69	70	71	...
...	72	73	74	75	76	77	78	79	80	81	82	83	...
...	84	85	86	87	88	89	90	91	92	93	94	95	

Binary filestream



0	12	24	36	48	60	72	84
1	13	25	37	49	61	73	85
2	14	26	38	50	62	74	86
3	15	27	39	51	63	75	87
4	16	28	40	52	64	76	88
5	17	29	41	53	65	77	89
6	18	30	42	54	66	78	90
7	19	31	43	55	67	79	91
8	20	32	44	56	68	80	92
9	21	33	45	57	69	81	93
10	22	34	46	58	70	82	94
11	23	35	47	59	71	83	95
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Padded Tile
Visualization



	0	1	2	3	4	5	6	7	8	9	10	11	0	0	0	0	...
...	12	13	14	15	16	17	18	19	20	21	22	23	0	0	0	0	...
...	24	25	26	27	28	29	30	31	32	33	34	35	0	0	0	0	...
...	36	37	38	39	40	41	42	43	44	45	46	47	0	0	0	0	...
...	48	49	50	51	52	53	54	55	56	57	58	59	0	0	0	0	...
...	60	61	62	63	64	65	66	67	68	69	70	71	0	0	0	0	...
...	72	73	74	75	76	77	78	79	80	81	82	83	0	0	0	0	...
...	84	85	86	87	88	89	90	91	92	93	94	95	0	0	0	0	

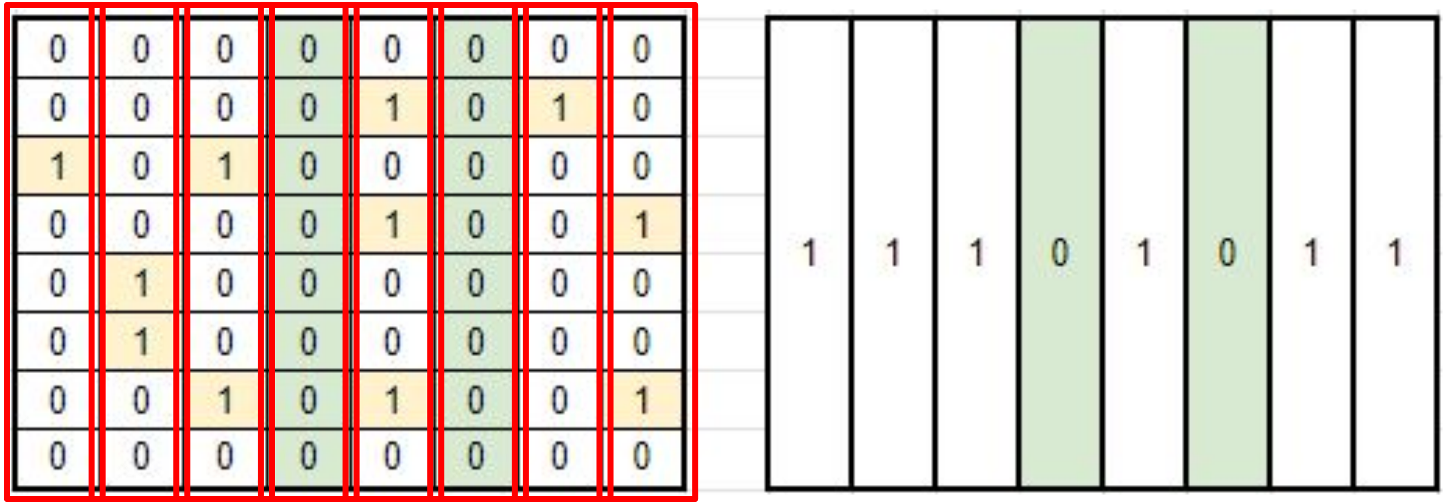
Local 1D uint_8 Array

2. Flag Sparse rows within input matrices

- If a row contains all zeros, result of that Systolic Array cycle will be zero
    - Can be ignored to reduce writes from SRAM
- Process:
    - Initialize boolean 1D arrays for Matrix A and B
    - 8 columns/rows per tile
    - Arrays of length 8 * Tile Count
    - Arrays referenced before writing to SRAM

2. Flag Sparse rows within input matrices

- Matrix A scanned by column, since matrix A is written column by column to the Systolic Array
- A zero means the column isn't written to SRAM
- TPU behavior unaffected (Output tile is a result of exclusively non-zero inputs)

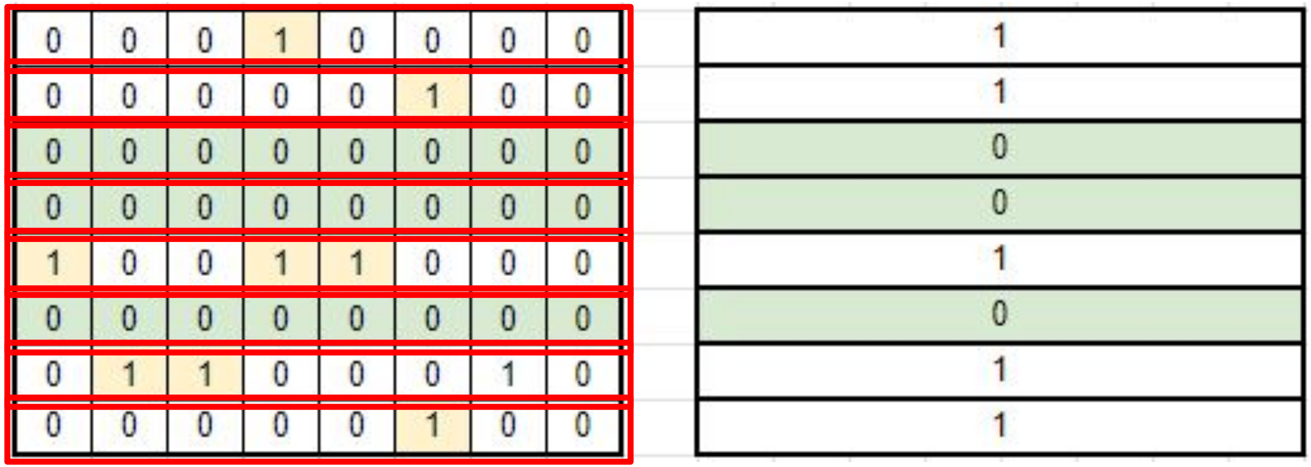


0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
1	0	1	0	0	0	0	0
0	0	0	0	1	0	0	1
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	0	0	0	0	0	0	0

1	1	1	0	1	0	1	1

2. Flag Sparse rows within input matrices

- Matrix B scanned by row, since matrix B is written row by row to the Systolic Array
- A zero means the row isn't written to SRAM
- TPU behavior unaffected (Output tile is a result of exclusively non-zero inputs)



0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0
0	1	1	0	0	0	1	0
0	0	0	0	0	1	0	0

1
1
0
0
1
0
1
1

3. Calculate Size of Input Data to write for each Output Tile

- Amount of input tiles required to calculate output
  - Determined by the common dimensions AW and BH

4x2 Tiles

A0	A4
A1	A5
A2	A6
A3	A7

2x3 Tiles

B0	B2	B4
B1	B3	B5

Output = 4x3 Tiles
Input Tiles per
Output Tile = 2

		B1		
		B1		
A4	A4	B1	C4	C8
A5	A5	C1	C5	C9
		C2	C6	C10
		C3	C7	C11

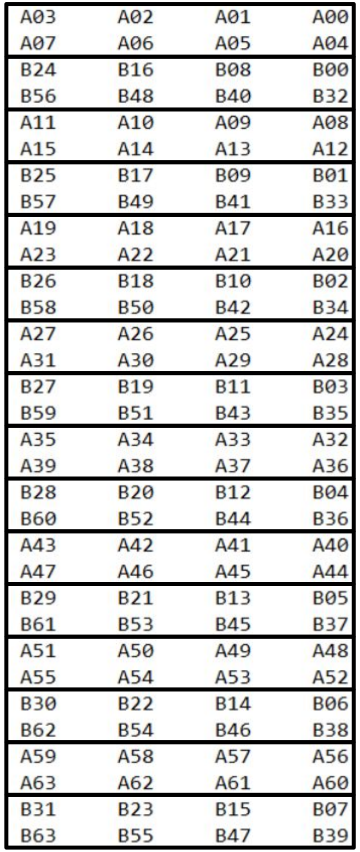
3. Calculate Size of Input Data to write for each Output Tile

- Ensure that Input Data to be written to SRAM wont overflow
  - 1024 Bytes (1KB) for each pair (A and B) of dense input tiles
  - SRAM region of the FPGA is of size 128 KB (⅛ MB!)
  - Limited to at most 128 pairs of Dense input tiles per Systolic Array Operation
    - Dense tile = 8 rows/columns => 1024 rows/columns

3. Calculate Size of Input Data to write for each Output Tile

- Sparsity introduces variable input sizes
  - Solution:
  - Simulate SRAM write, but with the Sparse Flag Matrices
  - If < 1024 rows/columns to be written, write all to SRAM, proceed normally
  - If > 1024 rows/columns
    - Write first 1024 rows/columns to TPU
    - Command TPU to calculate partial outputs
    - When TPU done with inputs, it will signal HPS to send next batch
    - When last batch written, TPU is commanded to write the output tile to SRAM

4. Write Input Data to SRAM

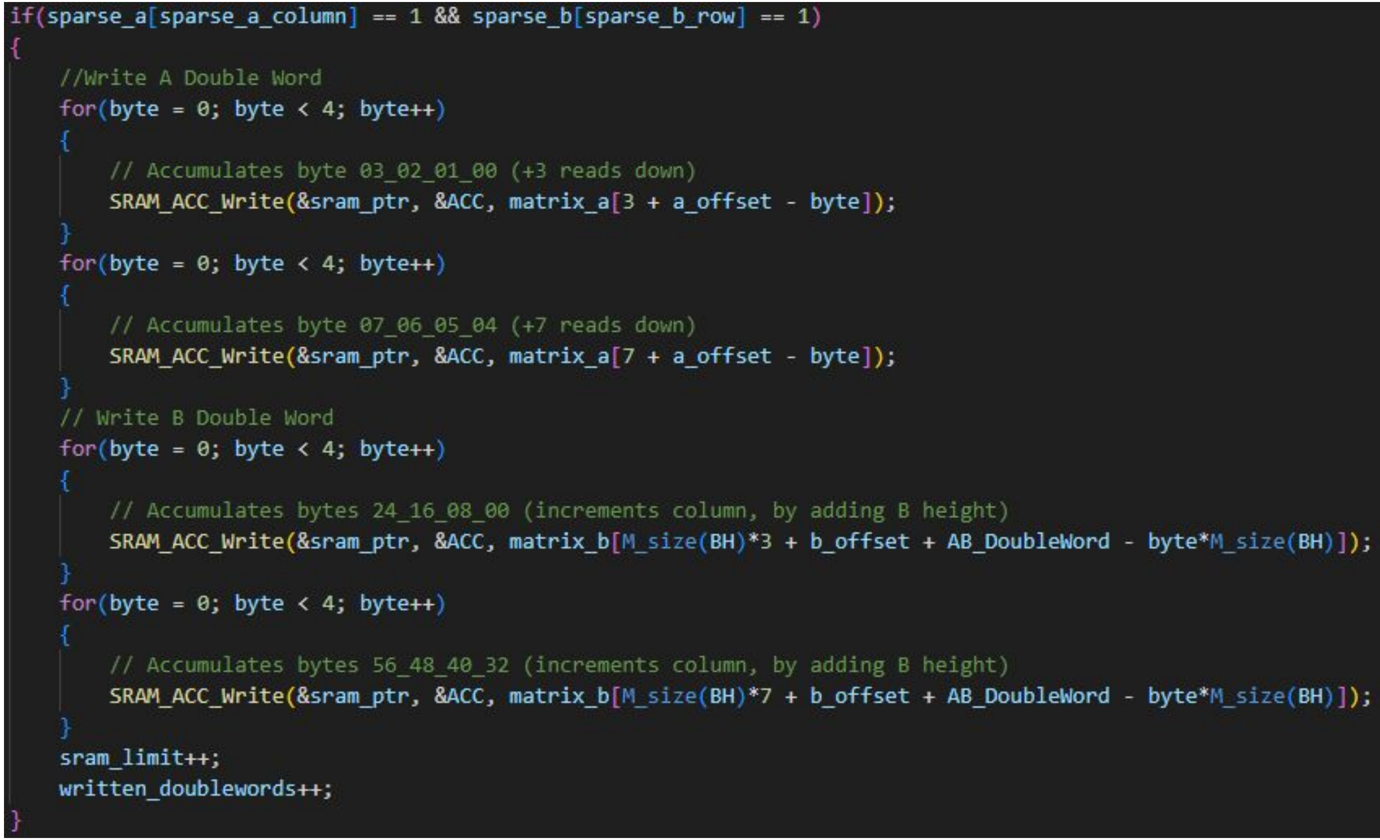


A03	A02	A01	A00	A Col 0
A07	A06	A05	A04	
B24	B16	B08	B00	B Row 0
B56	B48	B40	B32	
A11	A10	A09	A08	A Col 1
A15	A14	A13	A12	
B25	B17	B09	B01	B Row 1
B57	B49	B41	B33	
A19	A18	A17	A16	A Col 2
A23	A22	A21	A20	
B26	B18	B10	B02	B Row 2
B58	B50	B42	B34	
A27	A26	A25	A24	A Col 3
A31	A30	A29	A28	
B27	B19	B11	B03	B Row 3
B59	B51	B43	B35	
A35	A34	A33	A32	A Col 4
A39	A38	A37	A36	
B28	B20	B12	B04	B Row 4
B60	B52	B44	B36	
A43	A42	A41	A40	A Col 5
A47	A46	A45	A44	
B29	B21	B13	B05	B Row 5
B61	B53	B45	B37	
A51	A50	A49	A48	A Col 6
A55	A54	A53	A52	
B30	B22	B14	B06	B Row 6
B62	B54	B46	B38	
A59	A58	A57	A56	A Col 7
A63	A62	A61	A60	
B31	B23	B15	B07	B Row 7
B63	B55	B47	B39	

- Write to SRAM over 32b bridge
  - Limited to 4B or 4 inputs per write
  - Alternate complete col of A and row of B
- Process:
  - Program selects desired inputs from 1D arrays storing input matrices A and B
  - To read a column, +1 each read
  - To read a row, + Matrix Height each read

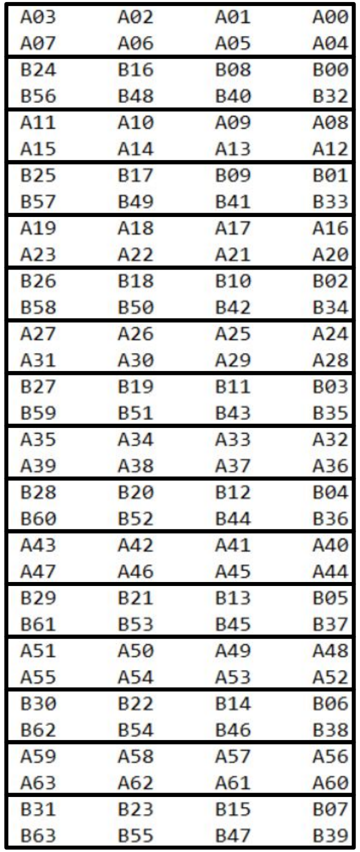
4. Write Input Data to SRAM

A03	A02	A01	A00	A Col 0
A07	A06	A05	A04	
B24	B16	B08	B00	B Row 0
B56	B48	B40	B32	
A11	A10	A09	A08	A Col 1
A15	A14	A13	A12	
B25	B17	B09	B01	B Row 1
B57	B49	B41	B33	
A19	A18	A17	A16	A Col 2
A23	A22	A21	A20	
B26	B18	B10	B02	B Row 2
B58	B50	B42	B34	
A27	A26	A25	A24	A Col 3
A31	A30	A29	A28	
B27	B19	B11	B03	B Row 3
B59	B51	B43	B35	
A35	A34	A33	A32	A Col 4
A39	A38	A37	A36	
B28	B20	B12	B04	B Row 4
B60	B52	B44	B36	
A43	A42	A41	A40	A Col 5
A47	A46	A45	A44	
B29	B21	B13	B05	B Row 5
B61	B53	B45	B37	
A51	A50	A49	A48	A Col 6
A55	A54	A53	A52	
B30	B22	B14	B06	B Row 6
B62	B54	B46	B38	
A59	A58	A57	A56	A Col 7
A63	A62	A61	A60	
B31	B23	B15	B07	B Row 7
B63	B55	B47	B39	

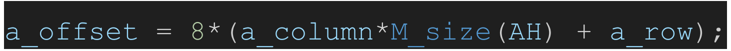


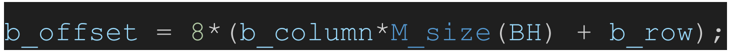
```
if(sparse_a[sparse_a_column] == 1 && sparse_b[sparse_b_row] == 1)
{
    //Write A Double Word
    for(byte = 0; byte < 4; byte++)
    {
        // Accumulates byte 03_02_01_00 (+3 reads down)
        SRAM_ACC_Write(&sram_ptr, &ACC, matrix_a[3 + a_offset - byte]);
    }
    for(byte = 0; byte < 4; byte++)
    {
        // Accumulates byte 07_06_05_04 (+7 reads down)
        SRAM_ACC_Write(&sram_ptr, &ACC, matrix_a[7 + a_offset - byte]);
    }
    // Write B Double Word
    for(byte = 0; byte < 4; byte++)
    {
        // Accumulates bytes 24_16_08_00 (increments column, by adding B height)
        SRAM_ACC_Write(&sram_ptr, &ACC, matrix_b[M_size(BH)*3 + b_offset + AB_DoubleWord - byte*M_size(BH)]);
    }
    for(byte = 0; byte < 4; byte++)
    {
        // Accumulates bytes 56_48_40_32 (increments column, by adding B height)
        SRAM_ACC_Write(&sram_ptr, &ACC, matrix_b[M_size(BH)*7 + b_offset + AB_DoubleWord - byte*M_size(BH)]);
    }
    sram_limit++;
    written_doublewords++;
}
```

4. Write Input Data to SRAM

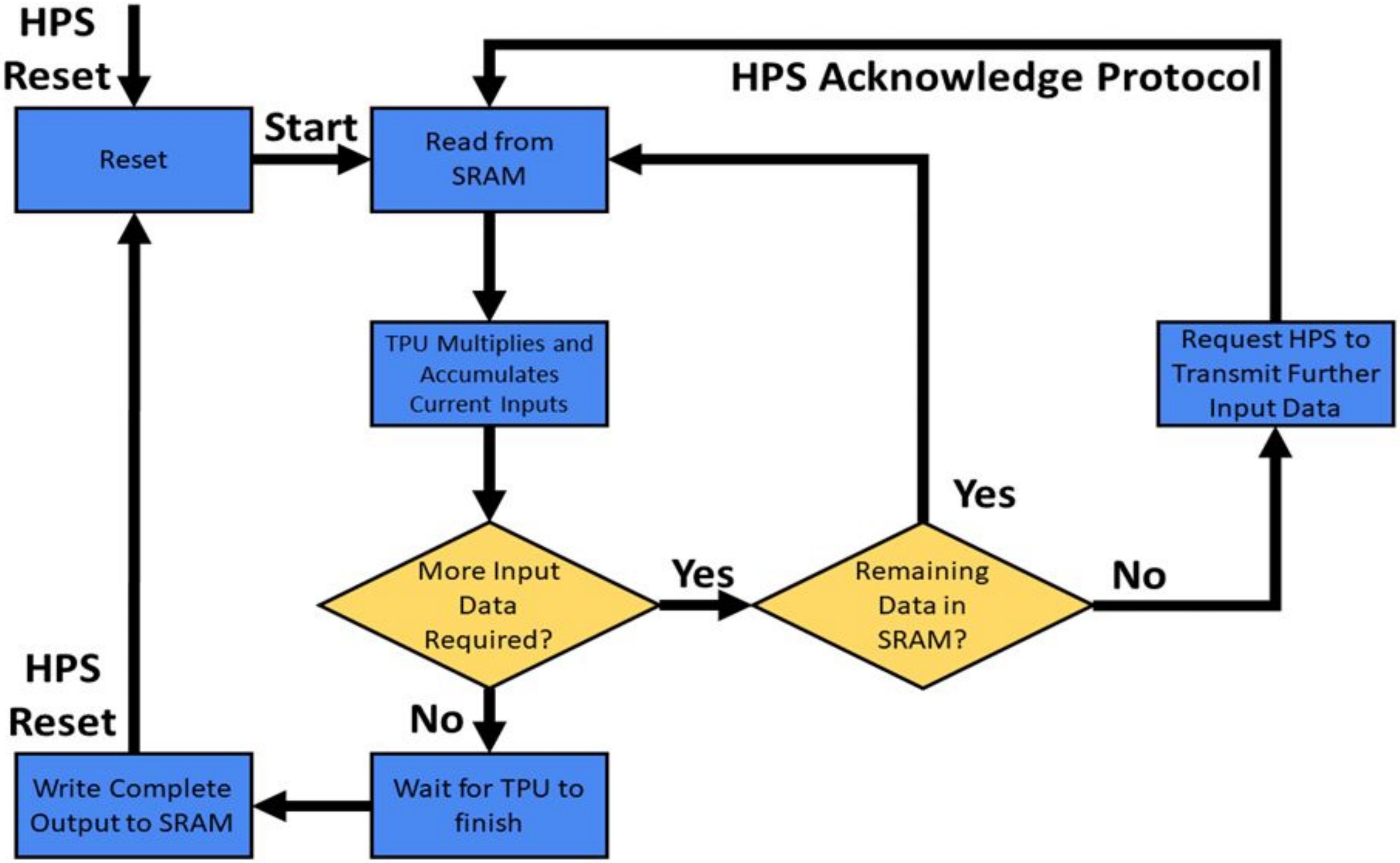


A03	A02	A01	A00	A Col 0
A07	A06	A05	A04	
B24	B16	B08	B00	B Row 0
B56	B48	B40	B32	
A11	A10	A09	A08	A Col 1
A15	A14	A13	A12	
B25	B17	B09	B01	B Row 1
B57	B49	B41	B33	
A19	A18	A17	A16	A Col 2
A23	A22	A21	A20	
B26	B18	B10	B02	B Row 2
B58	B50	B42	B34	
A27	A26	A25	A24	A Col 3
A31	A30	A29	A28	
B27	B19	B11	B03	B Row 3
B59	B51	B43	B35	
A35	A34	A33	A32	A Col 4
A39	A38	A37	A36	
B28	B20	B12	B04	B Row 4
B60	B52	B44	B36	
A43	A42	A41	A40	A Col 5
A47	A46	A45	A44	
B29	B21	B13	B05	B Row 5
B61	B53	B45	B37	
A51	A50	A49	A48	A Col 6
A55	A54	A53	A52	
B30	B22	B14	B06	B Row 6
B62	B54	B46	B38	
A59	A58	A57	A56	A Col 7
A63	A62	A61	A60	
B31	B23	B15	B07	B Row 7
B63	B55	B47	B39	

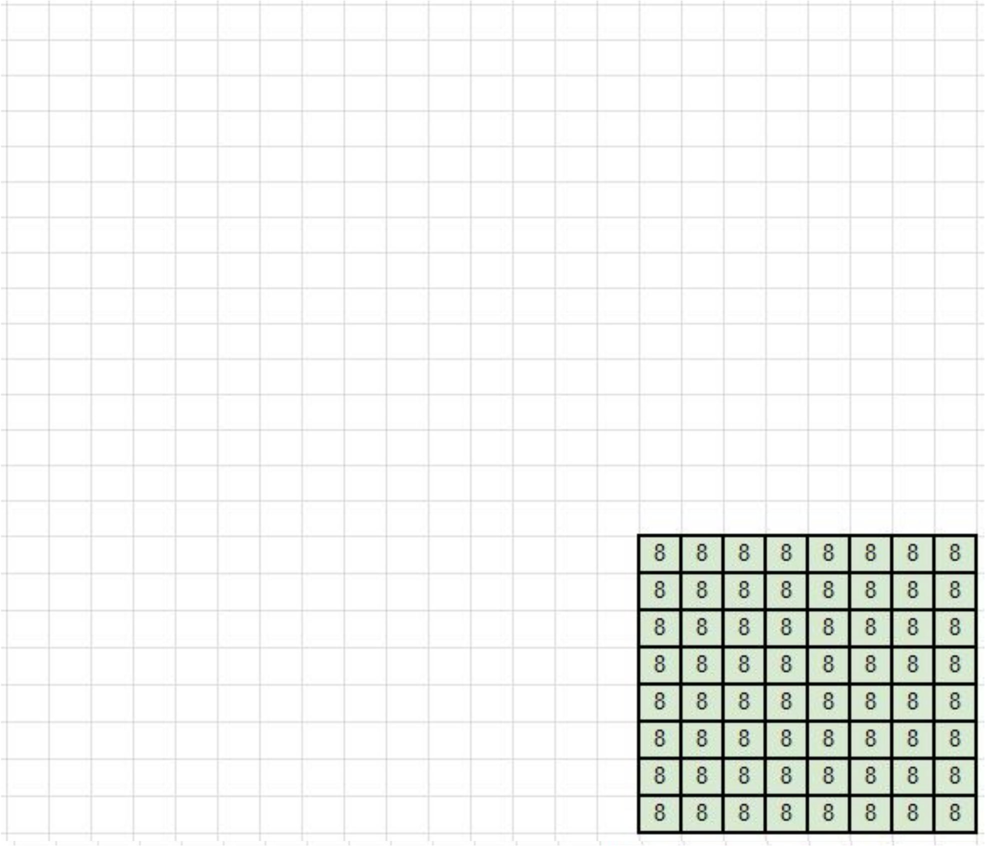
- Write to SRAM over 32b bridge
  - Limited to 4B or 4 inputs per write
  - Alternate complete col of A and row of B
- Process:
  - Row and Column of input tile internally maintained
  - Beginning of tile from 0 = 8*($column \cdot$M_Height + row)
  - 

    ```
    a_offset = 8*(a_column*M_size(AH) + a_row);
    ```
  - 

    ```
    b_offset = 8*(b_column*M_size(BH) + b_row);
    ```
  - Address of A column element = Array[tile offset + i]
  - Address of B row element = Array[tile offset + i*M_Height]
- Four 8b elements accumulated onto a 32b int, written to the next available address of SRAM


HPS
Reset
Reset
Start
Read from SRAM
HPS Acknowledge Protocol
TPU Multiplies and Accumulates Current Inputs
Request HPS to Transmit Further Input Data
Yes
More Input Data Required?
Yes
Remaining Data in SRAM?
No
HPS
Reset
No
Write Complete Output to SRAM
Wait for TPU to finish

5. TPU Calculates Systolic Array Result



8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8

- Inputs written to SRAM in order
- TPU reads from SRAM
  - feeds each input into the top and left rows of MACS

- Outputs buffered when MAC completes, and written to SRAM when MAC 7,7 concludes
- **Output Stationary**

5. TPU Calculates Systolic Array Result

- MAC Specs
    - 1 8b Multiplier
    - 1 32b Adder
    - 64 23b registers for output stationary

- Performance
    - Amount of cycles to compute one tile
    - (65+8+(TileWidth*8))*(TotalTiles)
        - 65 cycles to write outputs
        - 8 cycles of overhead
    - 137 cycles per tile

6. Program receives Tile Output from TPU

- Outputs stored in column major order
- Written from SRAM to an output file
  - Local Array became too large for the ARM core
  - Zeros within partial tiles are neglected
- Tiles stored in column-major order
  - The 64 elements within each tile are also in column-major order
  - One tile to the tile below (+64)
  - One tile to the right (+64*Output_Matrix_Tile_Height)