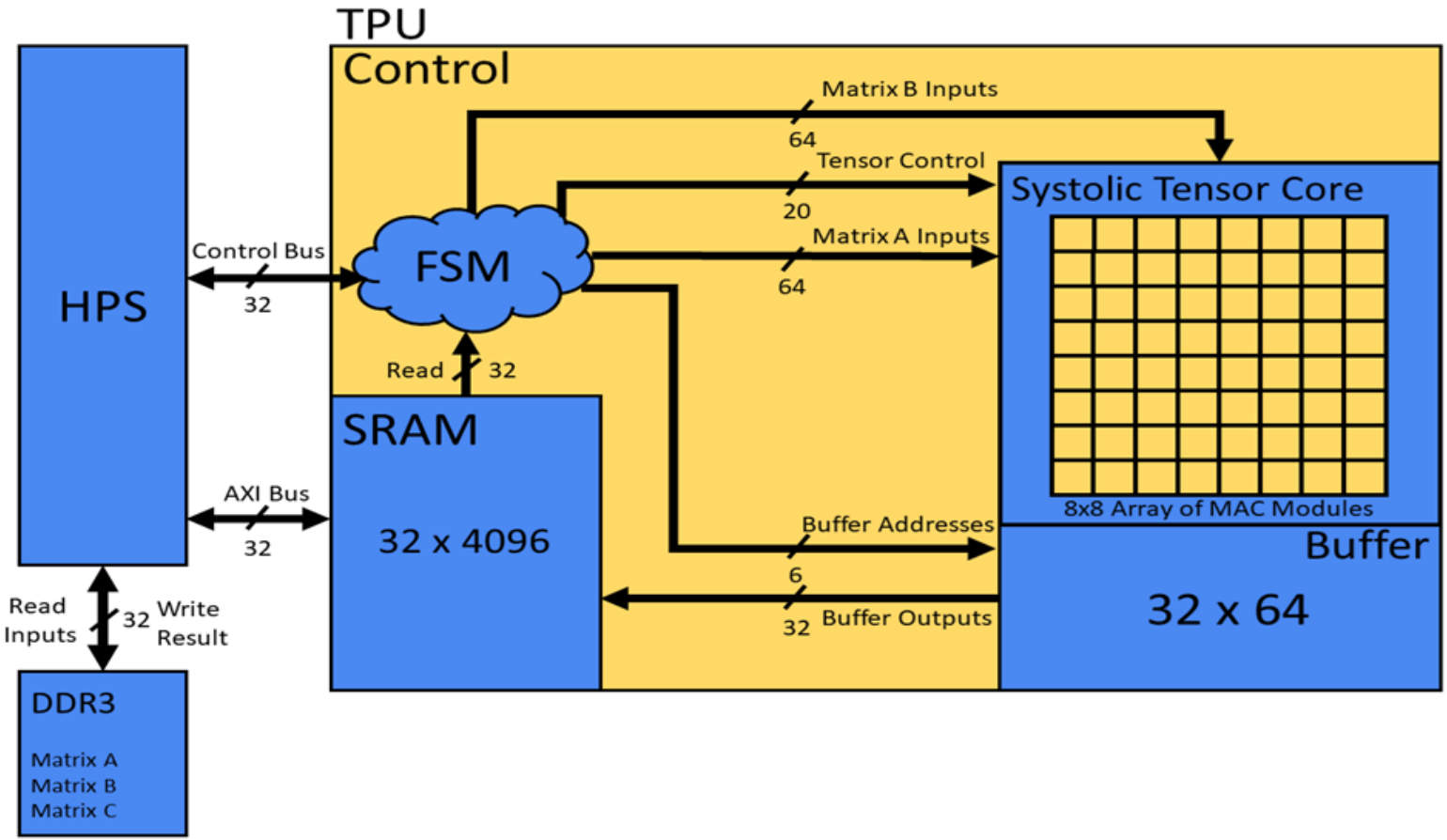
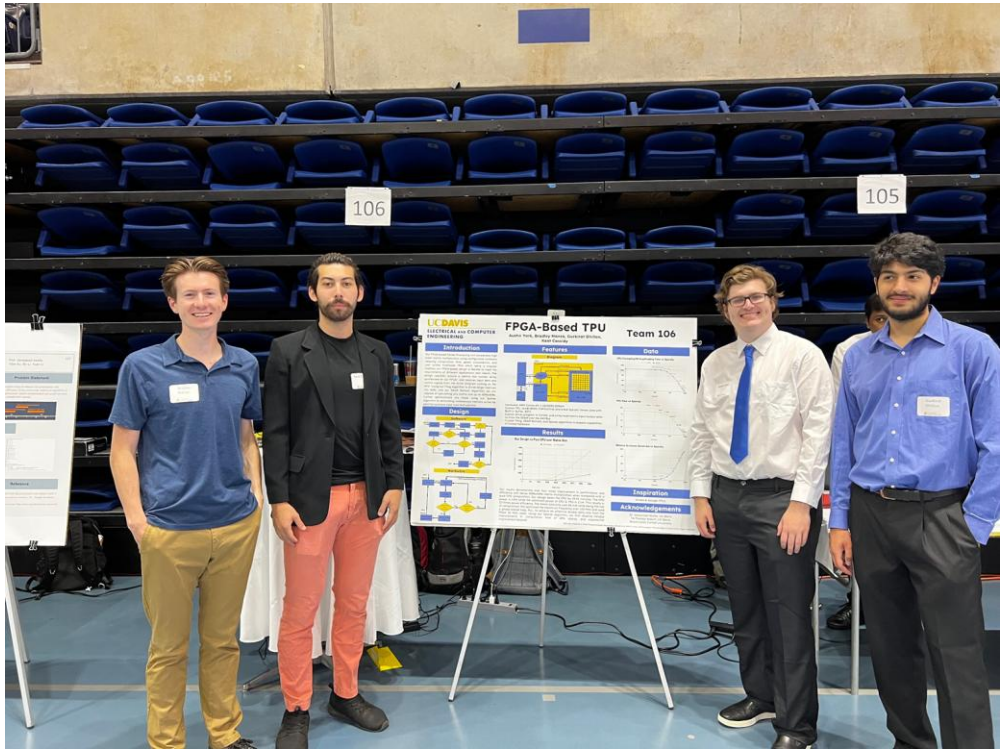
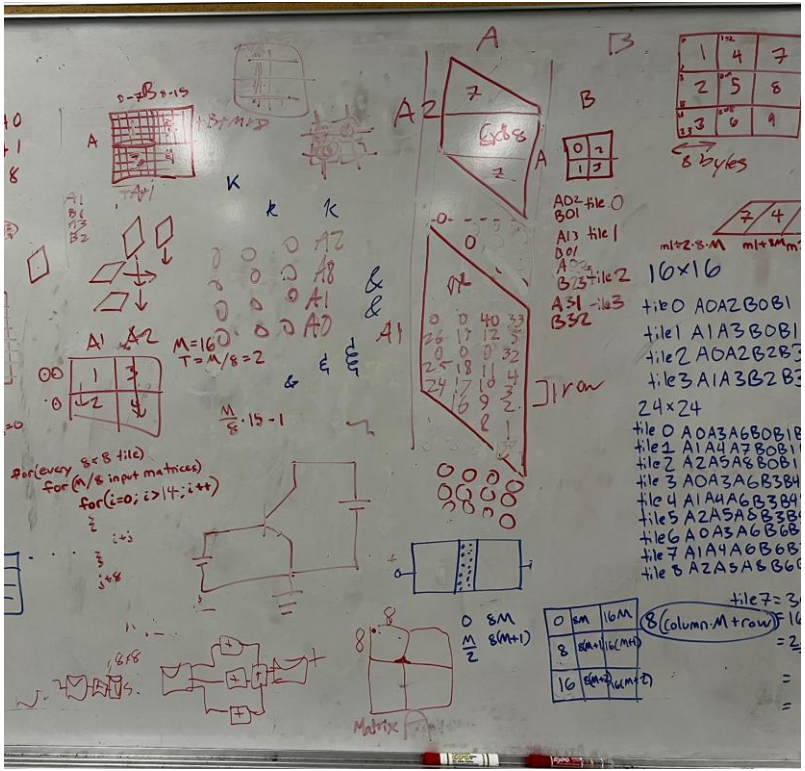

TPU
Control
HPS
Control Bus
32
FSM
Matrix B Inputs
64
Tensor Control
20
Matrix A Inputs
64
Systolic Tensor Core
8x8 Array of MAC Modules
Read
32
SRAM
32 x 4096
AXI Bus
32
Buffer Addresses
6
32
Buffer Outputs
Buffer
32 x 64
Read
Inputs
32
Write
Result
DDR3
Matrix A
Matrix B
Matrix C

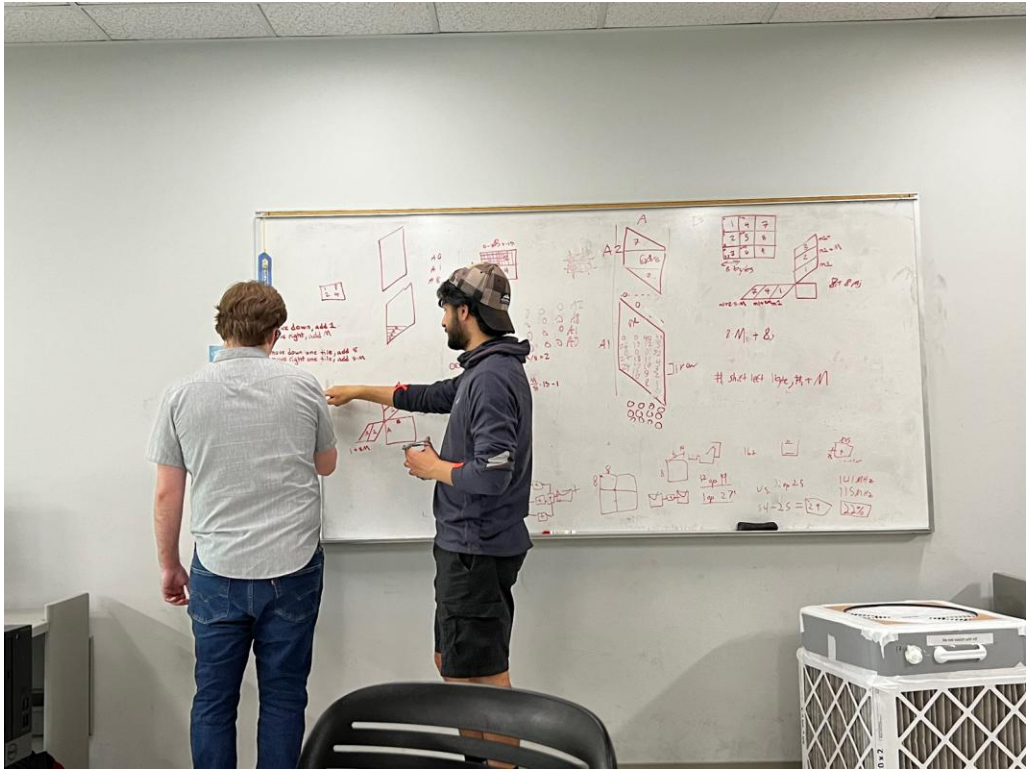


EEC 181 TPU Design Group

- 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

Teamwork (Software-Hardware Codesign)

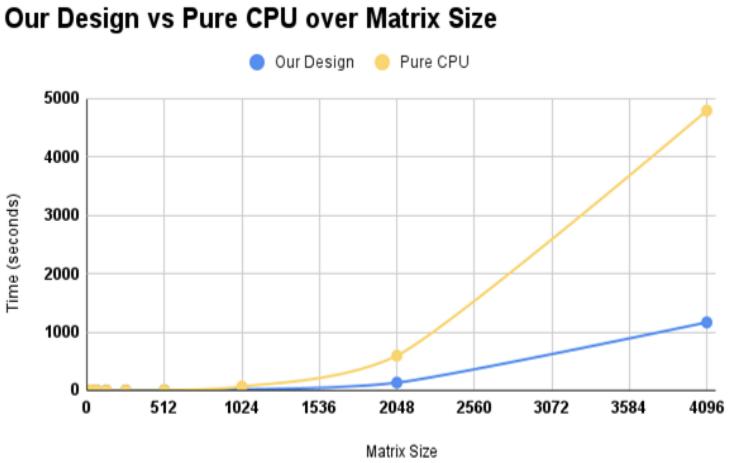


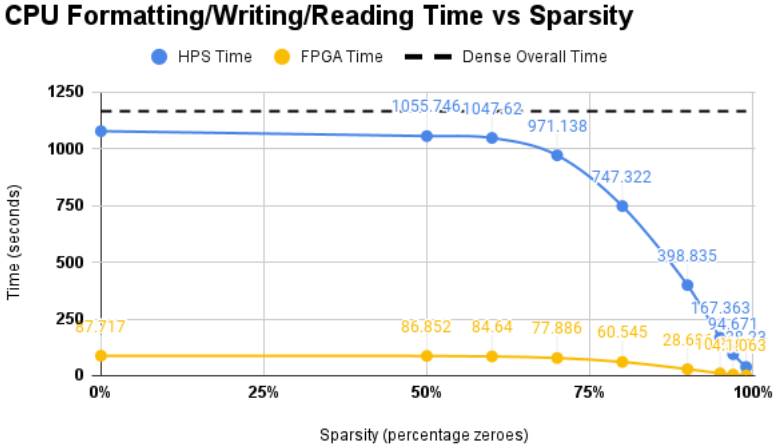


Results of Senior Design

Assigned by Professor Ventakesh Akella

- 4096x4096 Matrix Multiplication with tiling
- Sparse aware data orchestration via
- sparse row/column detection





Challenges and Limitations

- 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
  - Limited RAM, must write to file rather than to local array

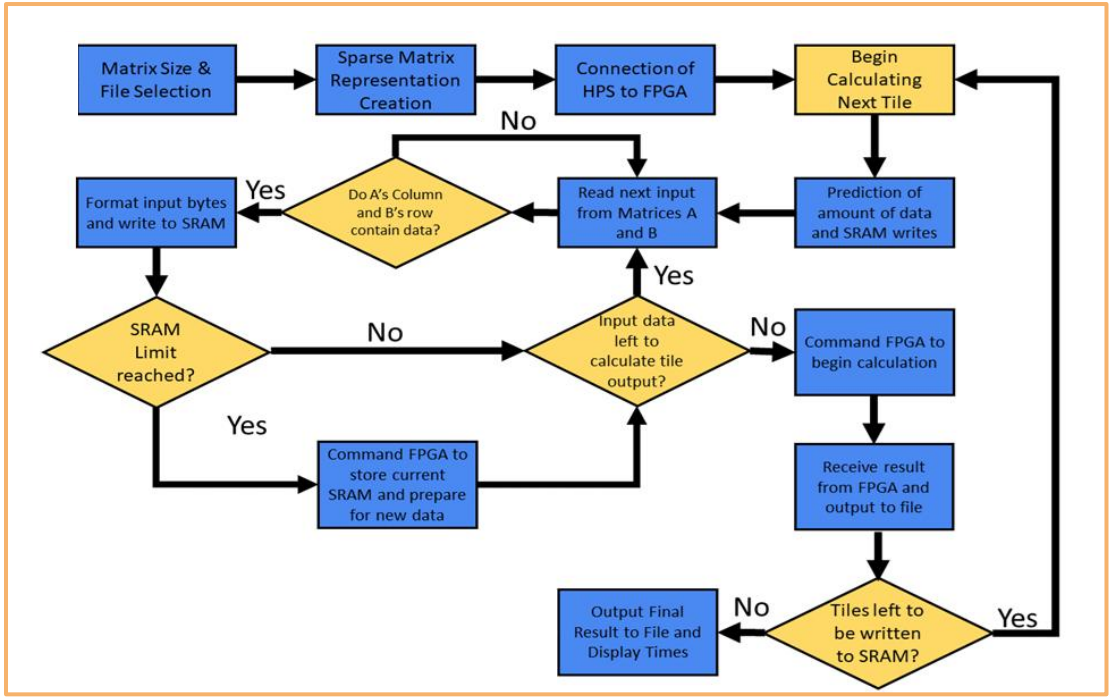
Fall Quarter EEC 289Q Final Project Accomplishments

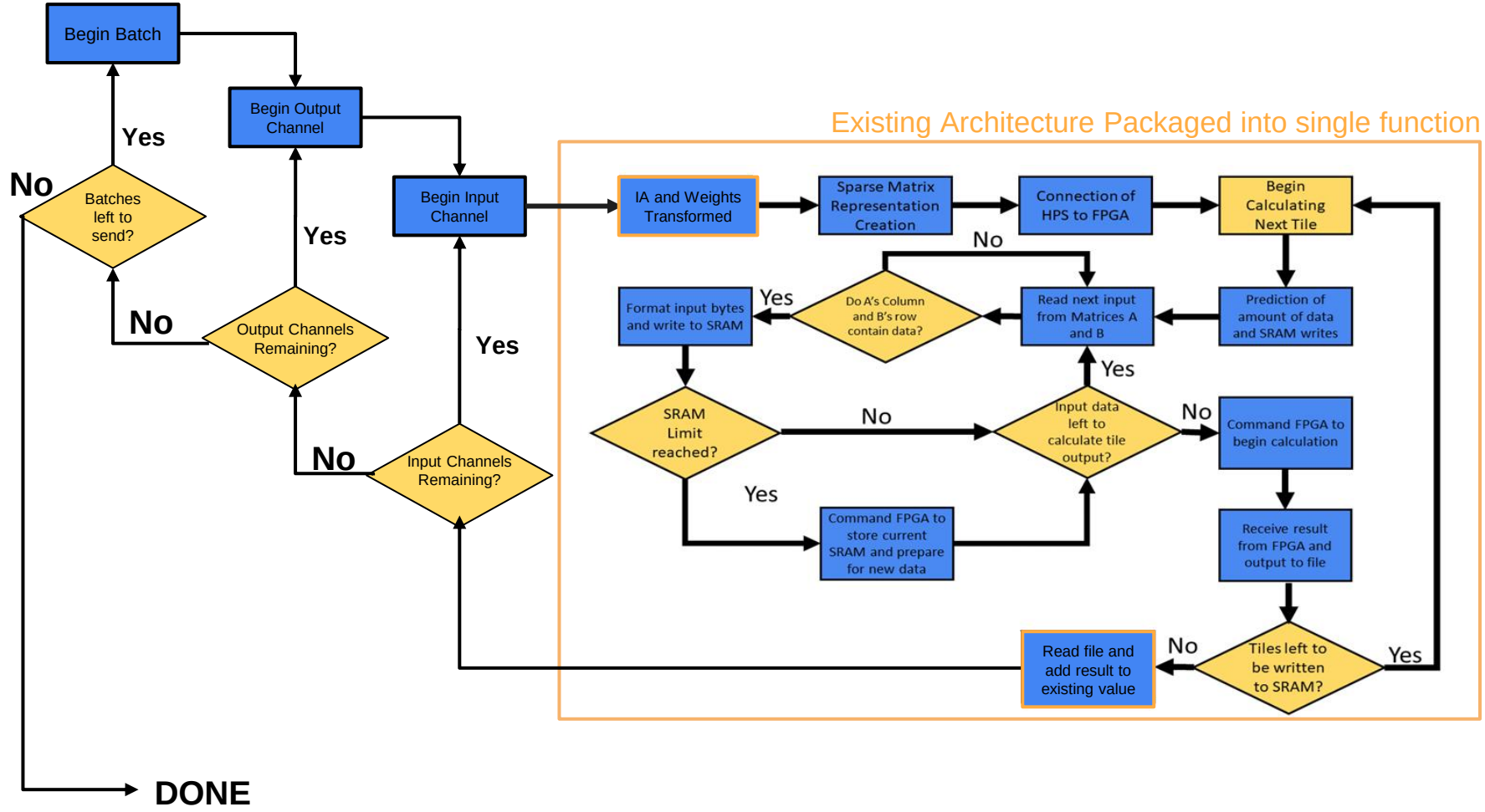
- **Enabled MxN Matrix Multiplication**
  - **Data sent is the minimum amount of tiles to store input, (padded to 8 with zeros).**
- **Logic to trim excess padded zeros from output**
  - **Considers actual tile utilization and ignores irrelevant zeros**
- **Logic to correctly interpret the the out_file without the context of zeros**

- **Convolution via GEMM with methods to transform kernel, and IA, convolution mode to print output correctly**
- **Convolution Benchmark in software and printing method to diff with hardware output**

- **Divided main into four libraries. Modularized methods, IO regions, and packaged TPU operation within one function.**
- **K-Channel Convolution supporting multiple Batches, Output Channels, and Input Channels**

Existing Architecture




Begin Batch
Begin Output Channel
Begin Input Channel
Batches left to send?
Yes
No
Output Channels Remaining?
Yes
No
Input Channels Remaining?
Yes
No
DONE
Existing Architecture Packaged into single function
IA and Weights Transformed
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?
No
Yes
Read file and add result to existing value

1. Accept Input weights and IA from User

- Process:
  - Input Matrices are:
    - File Stream (Volatile 1D array)
    - Column-Major Order
    - Binary Format (.bin)
  - Flatten weights into row, transform IA into columns of window positions
  - Call M_size() to determine padded dimensions of new matrices
  - 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	

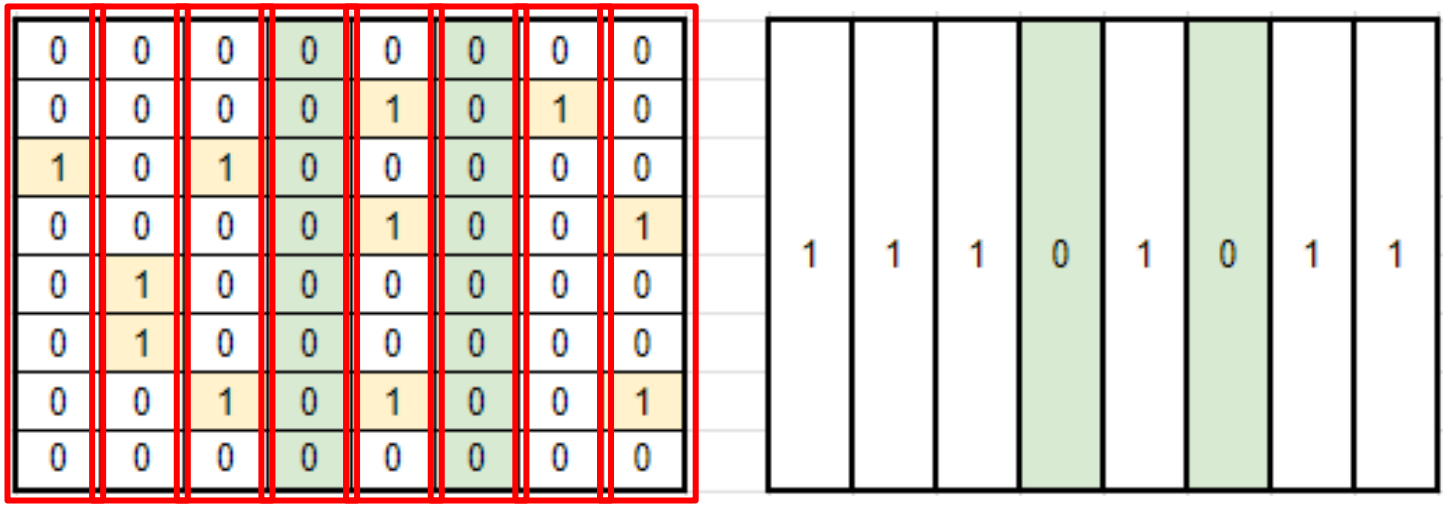
Local 1D uint_8 Array

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

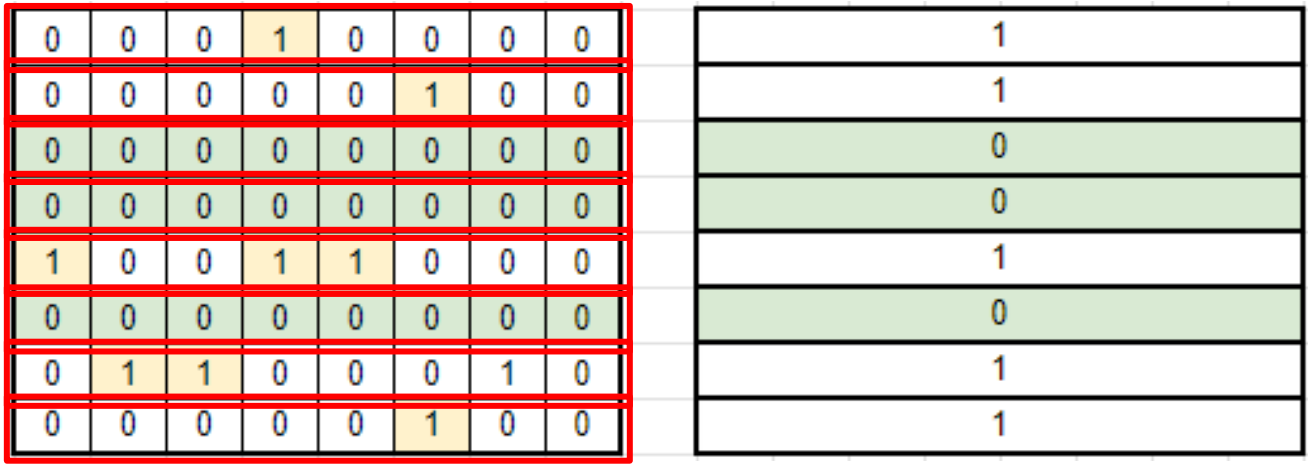
2. Flag Sparse rows within input matrices

- 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)



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)



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

- 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

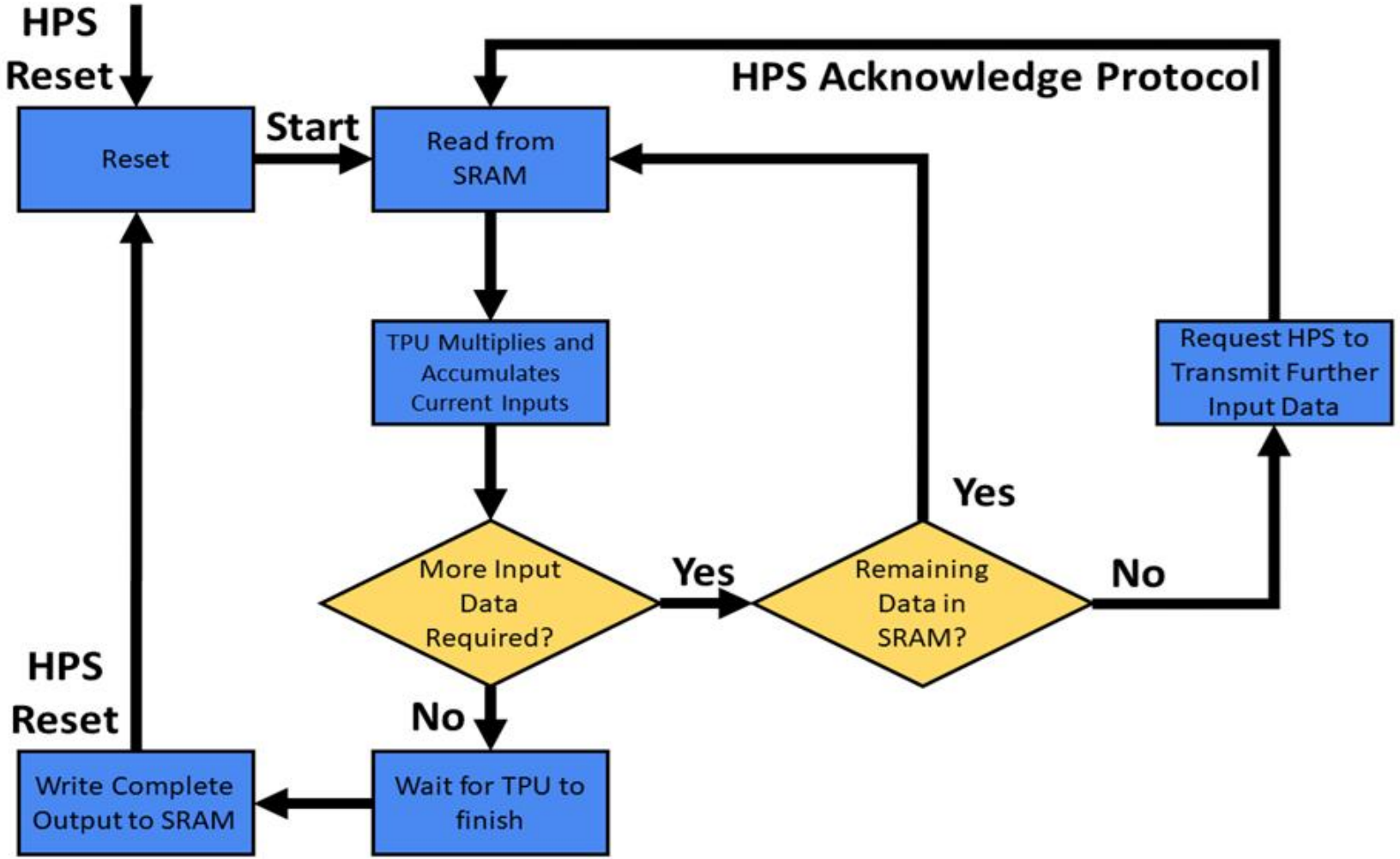
A03	A02	A01	A00	
A07	A06	A05	A04	A Col 0
B24	B16	B08	B00	
B56	B48	B40	B32	B Row 0
A11	A10	A09	A08	
A15	A14	A13	A12	A Col 1
B25	B17	B09	B01	
B57	B49	B41	B33	B Row 1
A19	A18	A17	A16	
A23	A22	A21	A20	A Col 2
B26	B18	B10	B02	
B58	B50	B42	B34	B Row 2
A27	A26	A25	A24	
A31	A30	A29	A28	A Col 3
B27	B19	B11	B03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

```
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++;
}
```

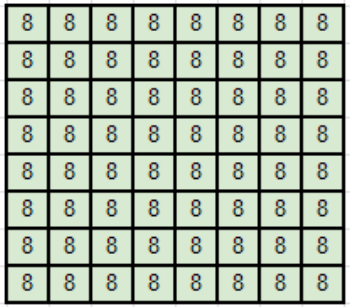
4. Write Input Data to SRAM

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

- 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



- 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

- Each output element is the sum of all previous channels
  - update existing element for each channel
- Generate new output file for each OA channel
- Generate new set of output files for each batch
- 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)

Results

- With no additional data reuse or modifications to protocol, there is no time saved compared to software :(
- Current sparse detection incompatible with flattened kernel matrix
  - zero sparse columns means 100% dense, despite seven sparse rows being 12.5%
  - Additional modifications to hardware protocol would allow for entire kernel to be cached in TPU SRAM without zeros
- All data resent each tile despite reusing kernel row each cycle
  - Additional detection at SRAM write stage to skip over existing kernel data could solve this
- Trouble storing outputs in local array despite being a single row
  - Attempted local array but returned to external file
  - Dual approach (File for dense, local array for convolution)
  - Additional Consideration to be paid towards memory utilization

Future Goals

- Software Goals
  - Convolution
    - Optimize with smarter memory utilization and SRAM writes
  - 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**
  - **New sparse protocol (First 3 bits indicate position to write to systolic array)**