Bridge of Life Education

ASoC Final Project Presentation
Convolutional Neural Network Accelerator

NYCU team1

412510020 高振翔

311511022 邱政岡

Agenda

- SW: Convolutional Neural Network training
- HW: Matrix multiplication
- System Integration
- Synthesis
- Analysis - Insight & Finding
- Reference

SW: Convolutional Neural Network training

Section Overview @CNN training

- Background Introduction
- Model training
- Objective
- Convolution to linear matrix multiplication
- Overhead

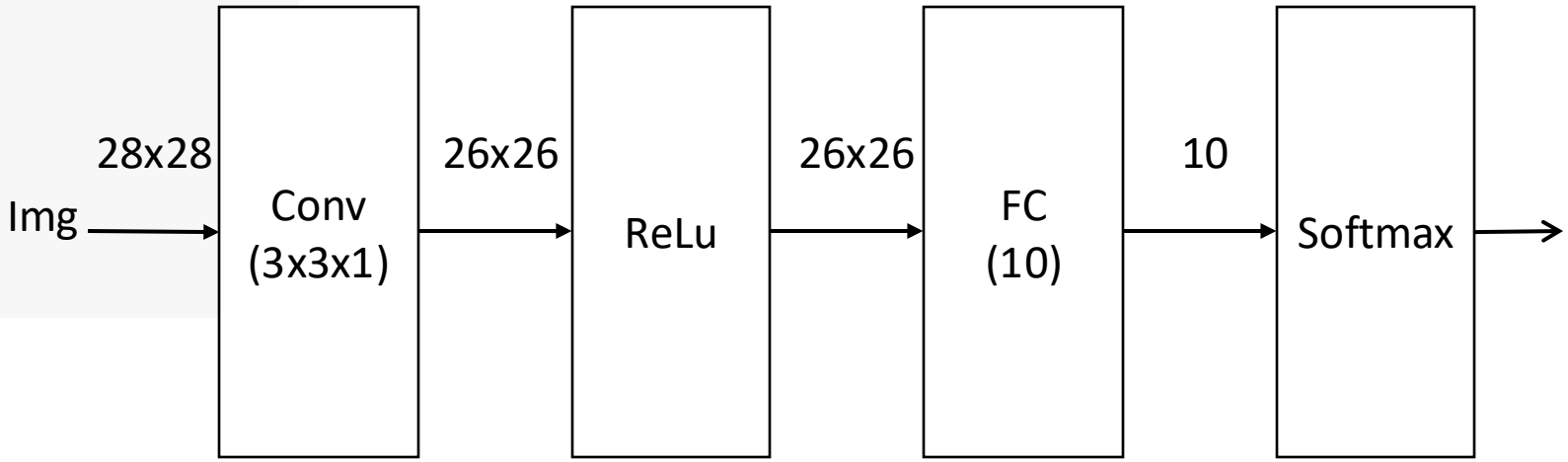
CNN Model training

- Scope at the application level
- Profiling to identify functions to accelerate
- SW application parallelization

Network

1. A simple CNN for classification
2. Includes 1 convolution layer, 1 fully-connected layer

```python
class Network(object):
    def __init__(self):
        self.cnn1 = Convolution(3, 1, 28)
        self.act1 = Relu()
        self.fc1 = FullyConnected(26*26, 10)
        self.loss = SoftmaxWithloss()

    def forward(self, input, target):
        h1 = self.cnn1.forward(input.reshape(input.shape[0], 28, 28))
        n1 = self.act1.forward(h1)
        h2 = self.fc1.forward(n1)
        pred, loss = self.loss.forward(h2, target)
        loss_total = loss.mean()
        return pred, loss_total

    def backward(self):
        loss_grad = self.loss.backward()
        h2_grad = self.fc1.backward(loss_grad)
        n1_grad = self.act1.backward(h2_grad)
        self.cnn1.backward(n1_grad)
```



Input data (10 class)

Running on pynq FPGA

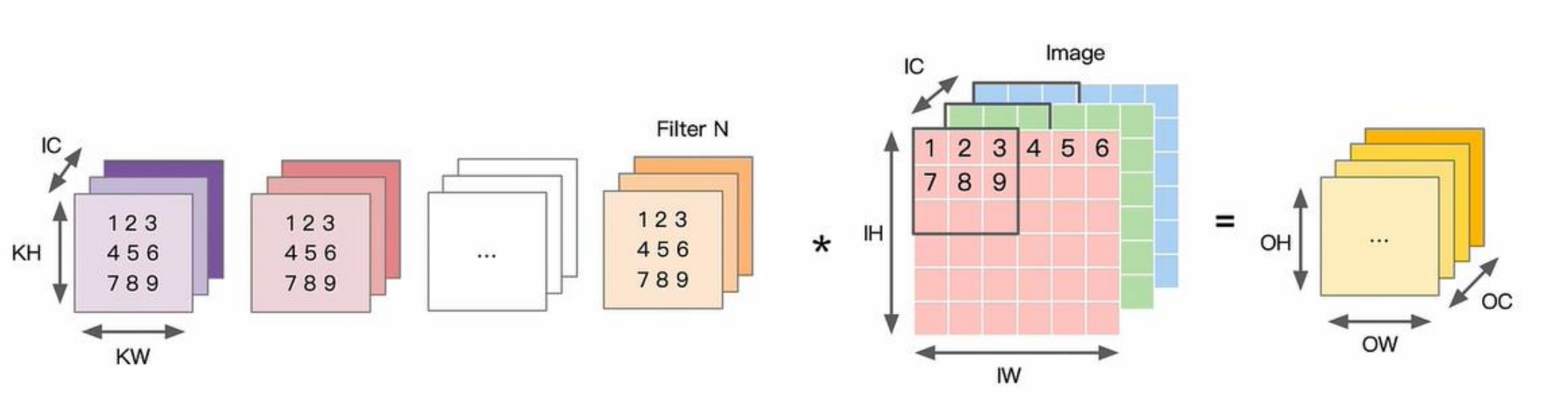
```python
# Training
net = Network()

train_batch_num = (train_image_num  -  val_image_num  )//Batch_size
val_batch_num = (val_image_num)//Batch_size
# test_batch_num = test_image_num//Batch_size

print("[   0.000] start training...")
start_time = time()
for epoch in range(1, EPOCH+1):
    train_hit = 0
    val_hit = 0
    total_train_loss = 0
    total_val_loss = 0
    for it in range(train_batch_num):
        pred, train_loss = net.forward(train_data[it*Batch_size:(it+1)*Batch_size], train_label_onehot[it*Batch_
        pred_index = np.argmax(pred, axis=1)
        train_hit += (pred_index==train_label[it*Batch_size:(it+1)*Batch_size]).sum()
        total_train_loss += train_loss

        step = epoch * train_batch_num + it
        Learning_rate = d_model**(-0.5) * min(step**(-0.5), step*(warmup_steps**(1.5)))
        net.backward()
        net.update(0.1*Learning_rate)
        print('[%8.3f]'%(time()-start_time), 'train batch:', '%5d'%it, '/', '%5d'%train_batch_num)

    for titt in range(val_batch_num):
        tit=train_batch_num+titt
        pred, val_loss = net.forward(train_data[tit*Batch_size:(tit+1)*Batch_size], train_label_onehot[tit*Batch
        pred_index = np.argmax(pred, axis=1)
        val_hit += (pred_index==train_label[tit*Batch_size:(tit+1)*Batch_size]).sum()
        total_val_loss += val_loss
        print('[%8.3f]'%(time()-start_time), 'val batch:', '%5d'%it, '/', '%5d'%train_batch_num)

    run_time = time() - start_time
    print('[%8.3f]'%run_time,'Epoch:%3d'%epoch, '|Train Loss:%8.4f'%(total_train_loss/train_batch_num), '|Train
          , '|Val Loss:%8.4f'%(total_val_loss/val_batch_num), '|Val Acc:%3.4f'%(val_hit/val_image_num*100.0))
```

```
[   0.000] start training...
[ 529.788] Epoch:  1 |Train Loss:  6.4527 |Train Acc:35.9094 |Val Loss:  4.8787 |Val Acc:46.7917
[1057.191] Epoch:  2 |Train Loss:  4.2771 |Train Acc:50.6159 |Val Loss:  4.0273 |Val Acc:53.2917
[1586.383] Epoch:  3 |Train Loss:  3.6445 |Train Acc:55.1540 |Val Loss:  3.5999 |Val Acc:55.8125
[2115.598] Epoch:  4 |Train Loss:  3.2717 |Train Acc:57.7627 |Val Loss:  3.3182 |Val Acc:57.6667
[2644.285] Epoch:  5 |Train Loss:  3.0092 |Train Acc:59.5507 |Val Loss:  3.1106 |Val Acc:58.8333
```

Objective

- Reduce calculation time of convolution by supplying more parallel-processing capabilities than CPU.
- Speed up overall CNN model training.

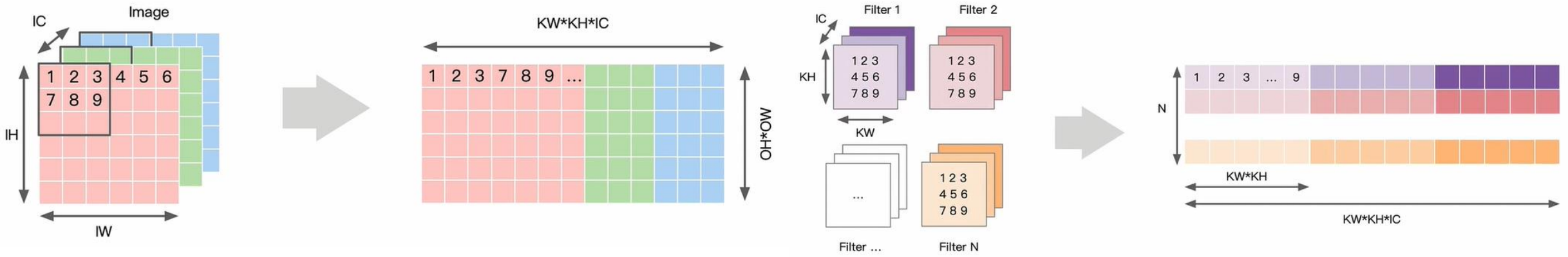
Normal Convolution



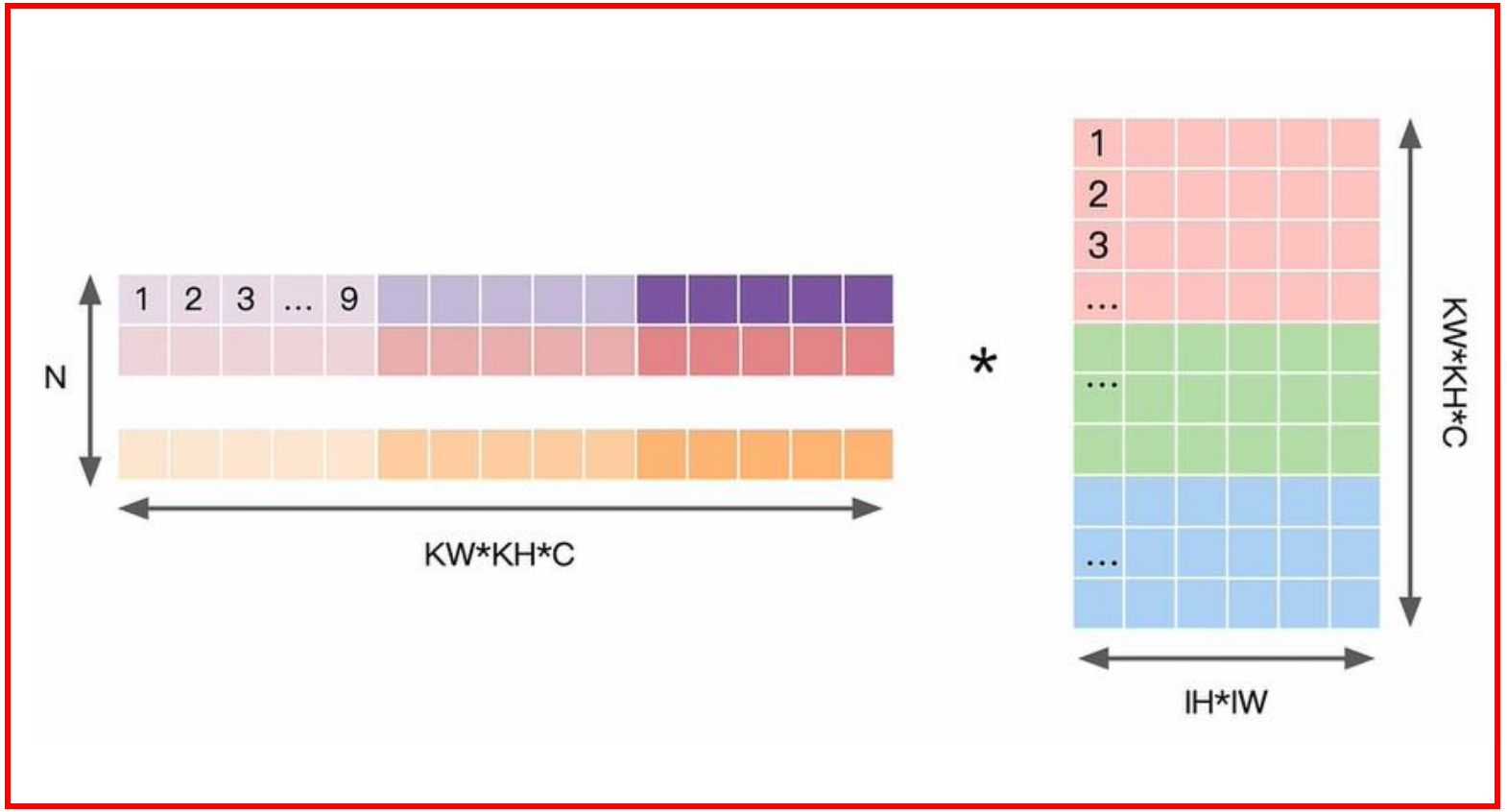
Convolution to linear matrix multiplication

- Software Im2col
- Use numpy.lib.stride_tricks.as_strided function

```
cmatrix = np.lib.stride_tricks.as_strided(
    input[index],
    shape=(self.out_size, self.out_size, self.kernal_size, self.kernal_size),
    strides=(data_stride*self.stride*self.in_size, data_stride*self.stride, data_stride*self.in_size
)
cmatrix = cmatrix.reshape(-1, self.kernal_size*self.kernal_size)
```



Convolution to linear matrix multiplication



Accelerated by Hardware

Convolution to linear matrix multiplication

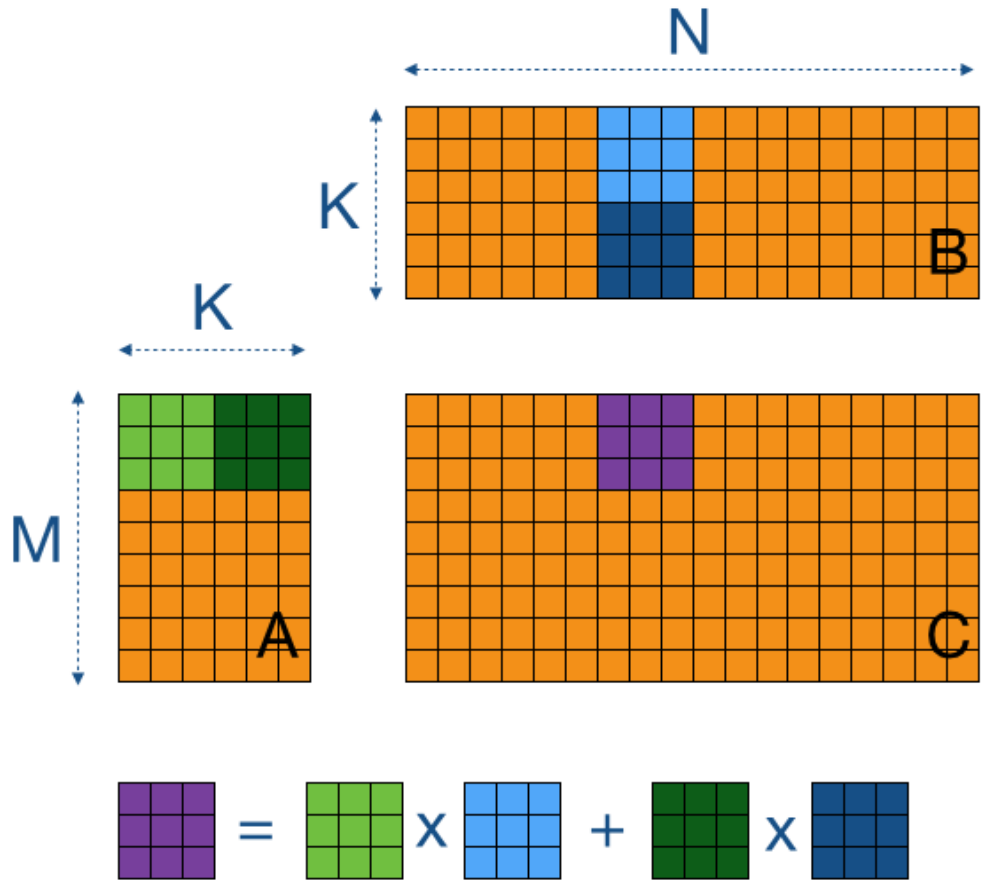
- Unit Test conv2D

```
Generated Pattern from Convolution:
[[  26  -56   98  -18 -120  -77  101  -81]
 [ -79  -87   64   39   10  -17   91   -3]
 [  91 -128  100   60 -108 -122  102  -60]
 [ -45  -17   21  -57    9   82 -128   88]]
Difference between generated pattern and golden pattern:
[[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]]
The generated pattern matches the golden pattern.
```

Overhead - Matrix Tiling

- (M, K) x (K, N) for example
  - Tile size = 3

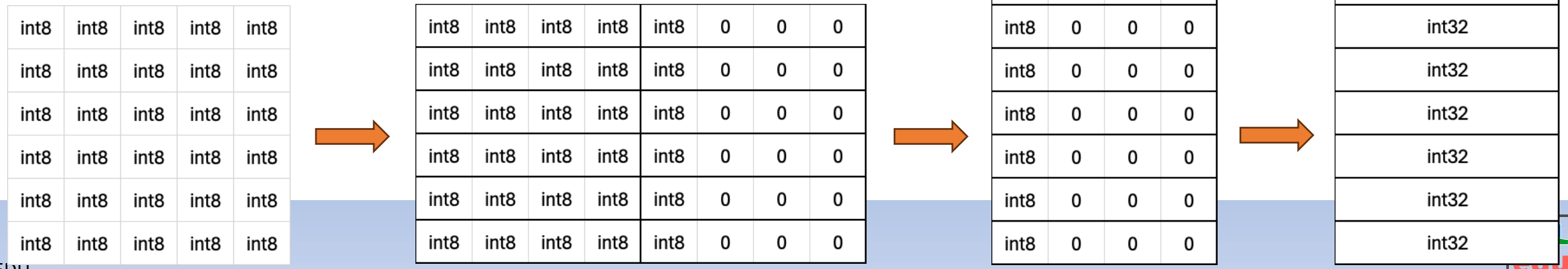
```python
def matrix_mul_with_tiling(A, B, t):
    """
    Multiplies two matrices A and B using tiling.

    Parameters:
    A (numpy.ndarray): First matrix of size (m, k).
    B (numpy.ndarray): Second matrix of size (k, n).
    t (int): Tile size.

    Returns:
    numpy.ndarray: Resultant matrix of size (m, n) after multiplication.
    """
    m, k1 = A.shape
    k2, n = B.shape

    # Ensure the matrices can be multiplied
    if k1 != k2:
        raise ValueError("The number of columns in A must be equal to the number of rows in B")

    # Initialize the result matrix with zeros
    C = np.zeros((m, n))

    # Perform tiled matrix multiplication
    for i in range(0, m, t):
        for j in range(0, n, t):
            for k in range(0, k1, t):
                # Define the end points for the current tile
                i_end = min(i + t, m)
                j_end = min(j + t, n)
                k_end = min(k + t, k1)

                # Perform the multiplication for the current tile
                # C[i:i_end, j:j_end] += np.dot(A[i:i_end, k:k_end], B[k:k_end, j:j_end])
                C[i:i_end, j:j_end] += matrix_mul_with_preproc(A[i:i_end, k:k_end], B[k:k_end, j:j_end])

    return C
```



Overhead – Data pre-processing

- Change matrix format to fit the hardware input

```python
def mat_preproc(matrix):
    matrix = matrix.astype(np.int8)
    m, n = matrix.shape
    if n % 4 != 0:
        matrix = np.concatenate((matrix, np.zeros((m, 4 - (n % 4)), dtype=np.int8)), axis=1)
        n = matrix.shape[1]

    matrix_proc = np.lib.stride_tricks.as_strided(
        matrix,
        shape=(n//4, m, 4),
        strides=(4, 8, 1)
    ).reshape(-1, 4)
    # print(matrix_proc)

    # Convert the int8 matrix to uint8 to handle bytes correctly
    uint8_matrix = np.flip(matrix_proc, axis=1).astype(np.uint8)

    # View the reshaped matrix as int32
    # And convert the int32 matrix to a native Python int array
    return uint8_matrix.view(np.int32).ravel().tolist()
```



HW: Matrix Multiplication

Section Overview

- Design Introduction
  - Background Introduction
  - Kernel IP Architecture
  - TPU Controller FSM
  - Block diagram

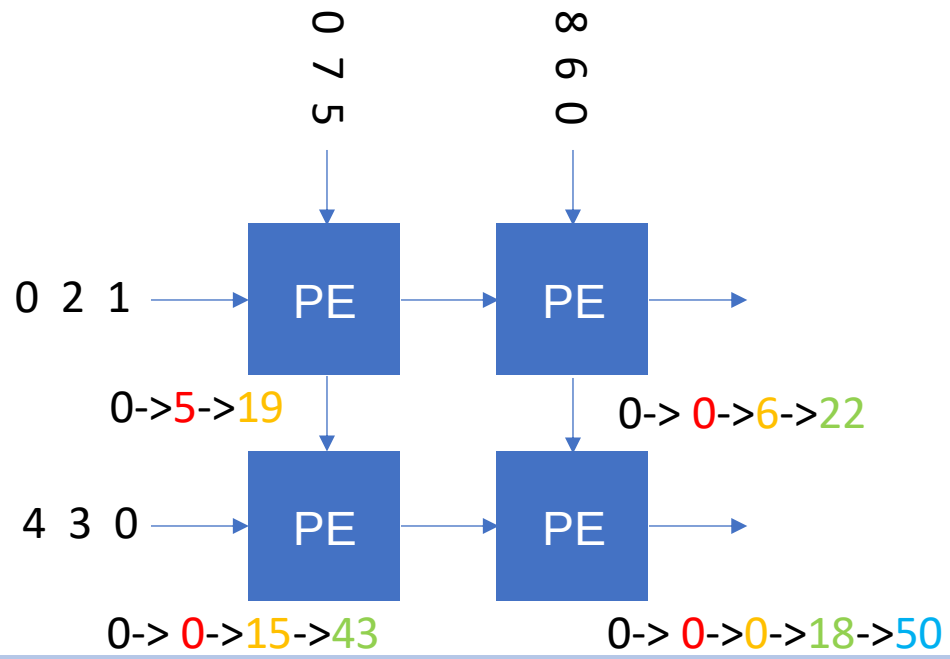
- Hardware Simulation
  - Kernel IP level: Matrix Multiply
  - System level: fsic integration

Background Introduction -- Time Optimization

- Before:
  - It consumes N*N*N cycles -> time complexity is O(N^3)
- After:
  - It consumes approximately 3N cycles -> time complexity is O(N)

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

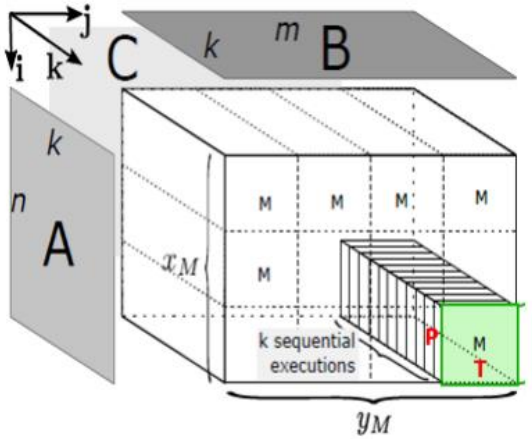
SA

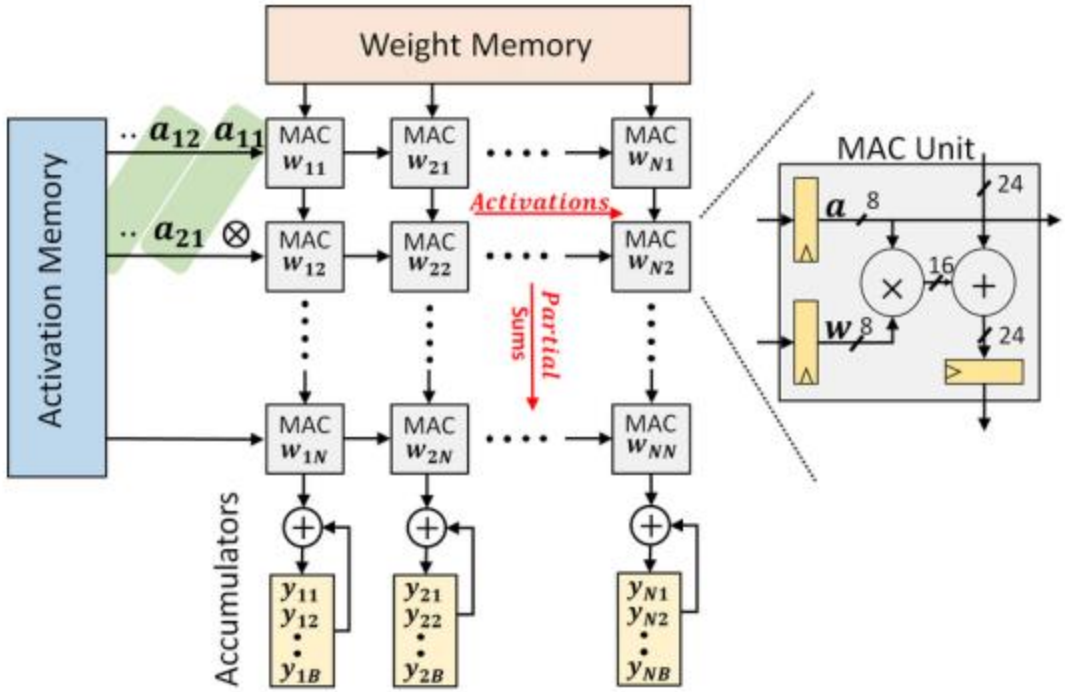


Background Introduction -- Systolic Array

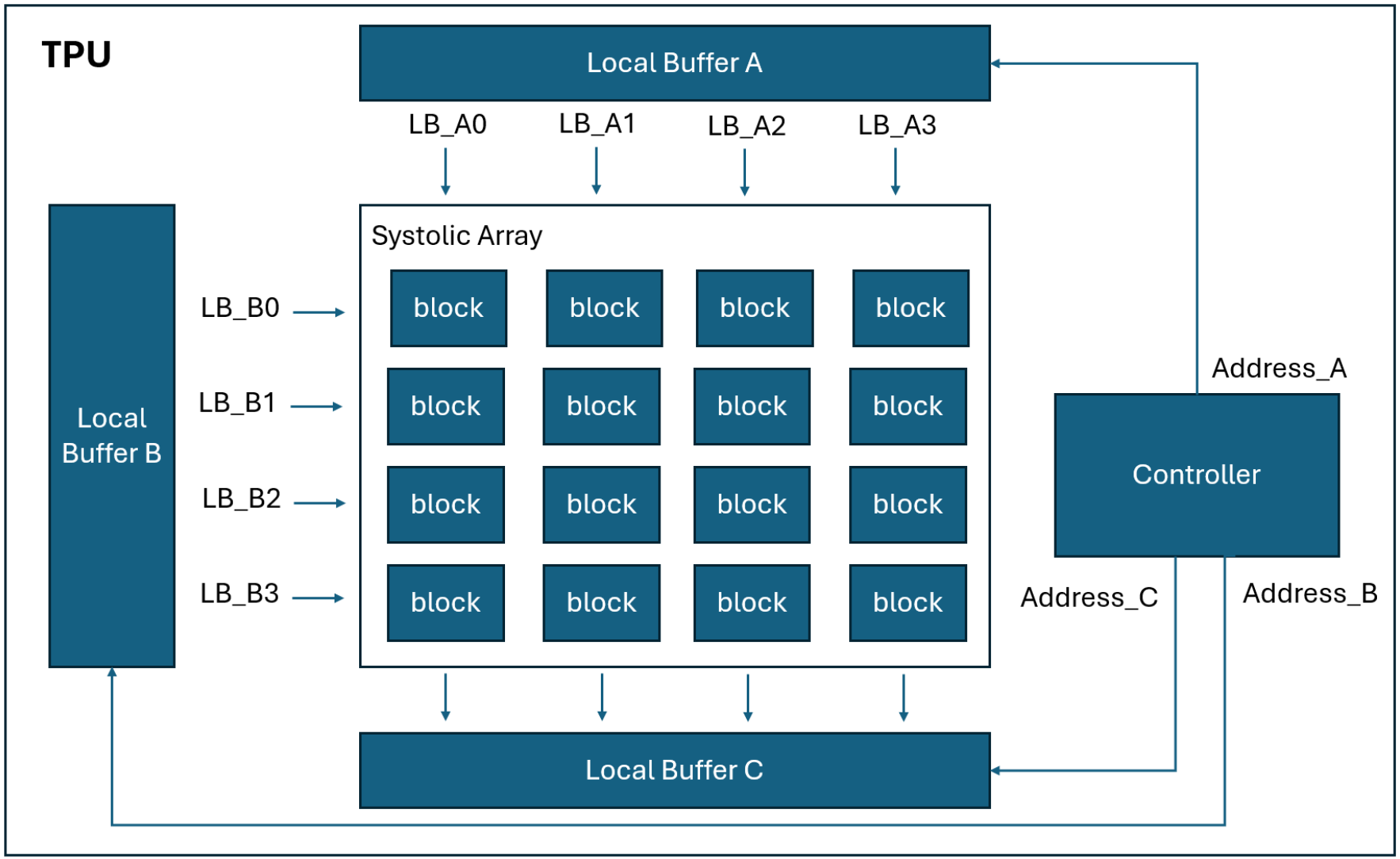
- Consists of a grid of PEs arranged in a regular pattern
  - Each PE has its own local memory.
  - PEs perform computations on the data they receive and passing it along to the next PE.

- Advantage
  - high throughput and efficiency
  - scalable solution for parallel processing tasks





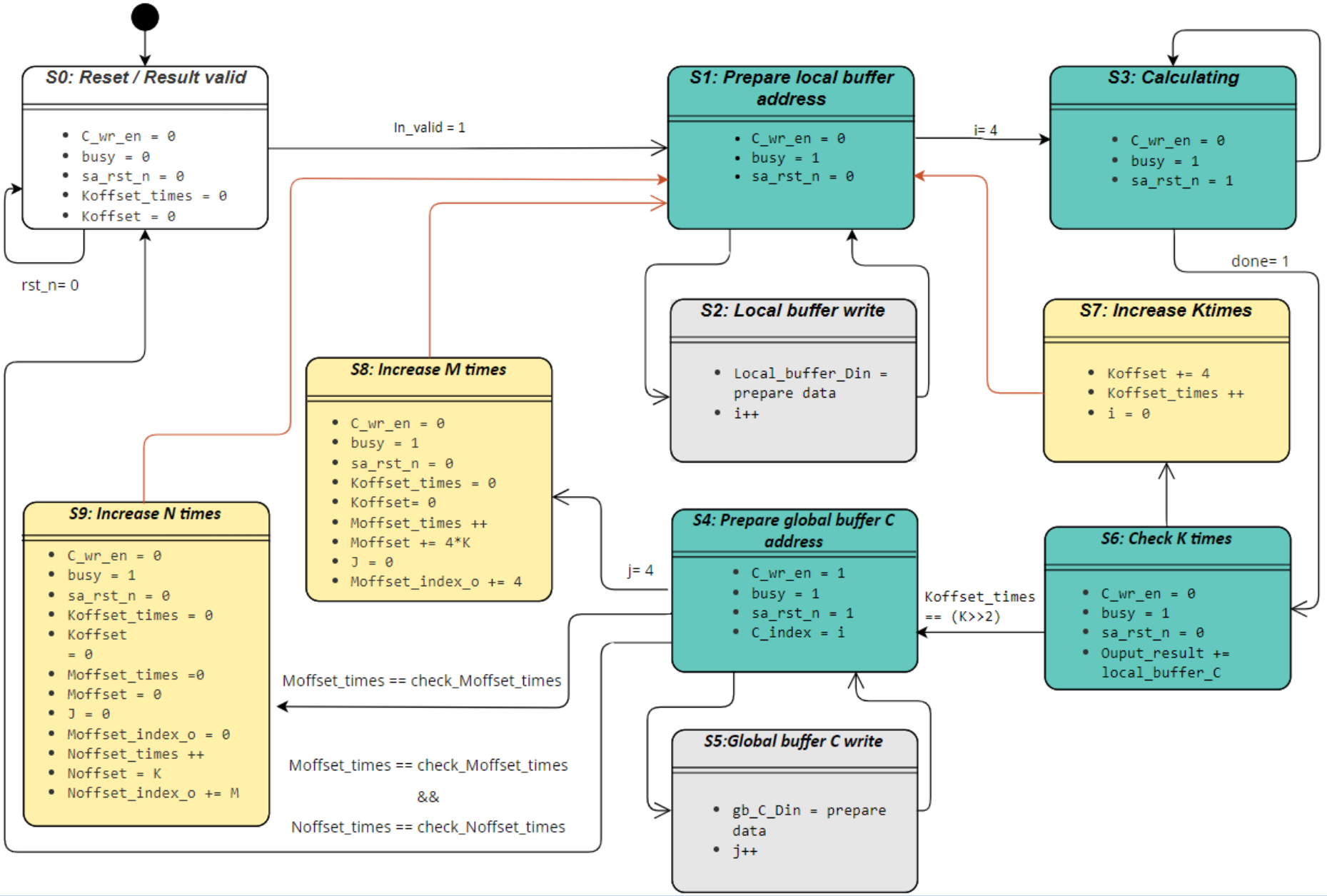
Kernel IP Architecture



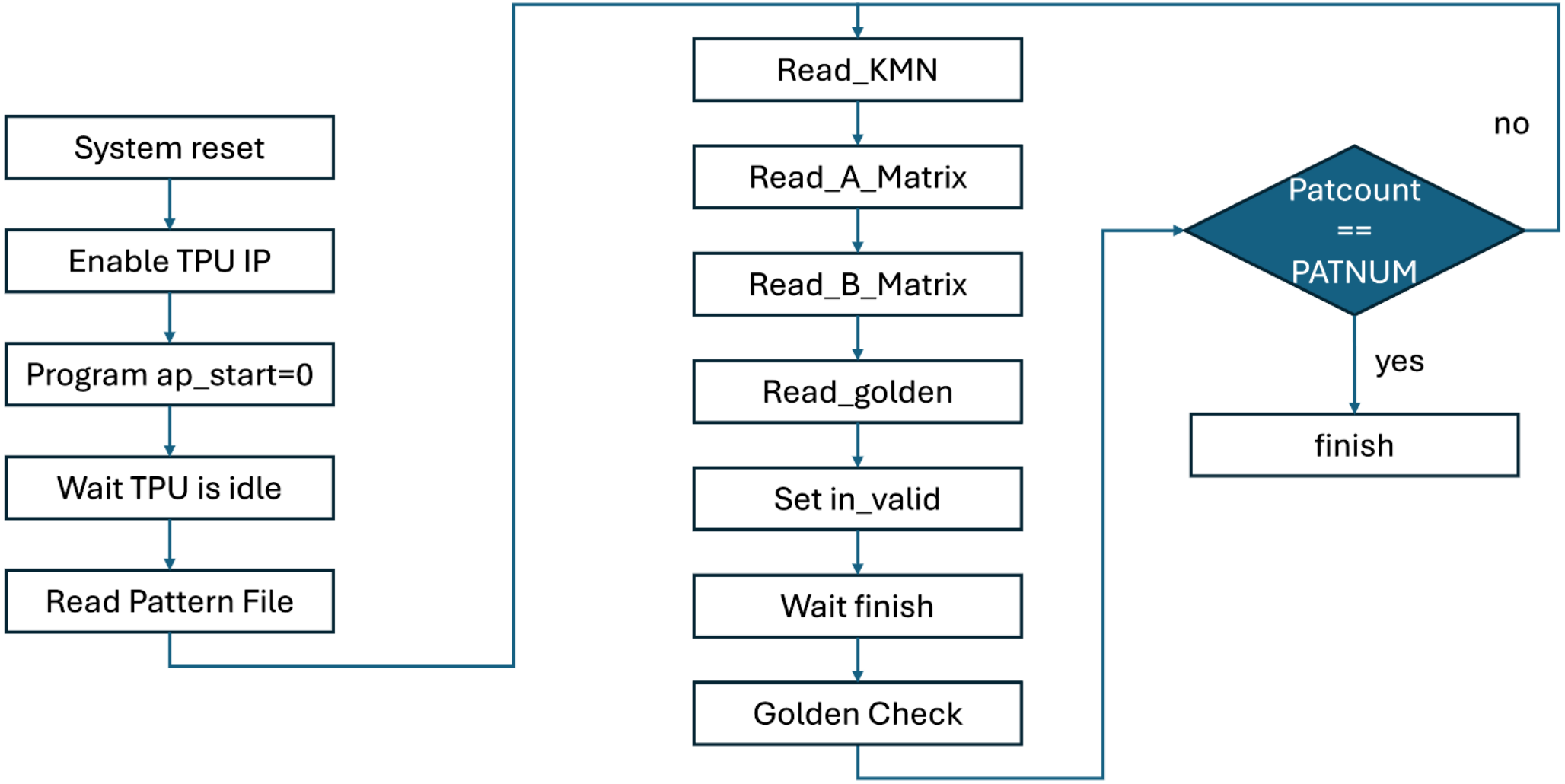
MMIO

```
//========================================================================
//    DATA PATH & CONTROL
//========================================================================
//---------- AXI-lite slave Interface ----------
// Address map
// 0x00: config ([2]ap_idle, [1]ap_done, [0]ap_start)
//
// 0x10: M
// 0x14: K
// 0x18: N
//
// 0x20: buf_A_address
// 0x24: buf_A_din
//
// 0x30: buf_B_address
// 0x34: buf_B_din
//
// 0x40: buf_C_address
// 0x44: buf_C_dout_0
// 0x48: buf_C_dout_1
// 0x4c: buf_C_dout_2
// 0x50: buf_C_dout_3
```

TPU Controller FSM



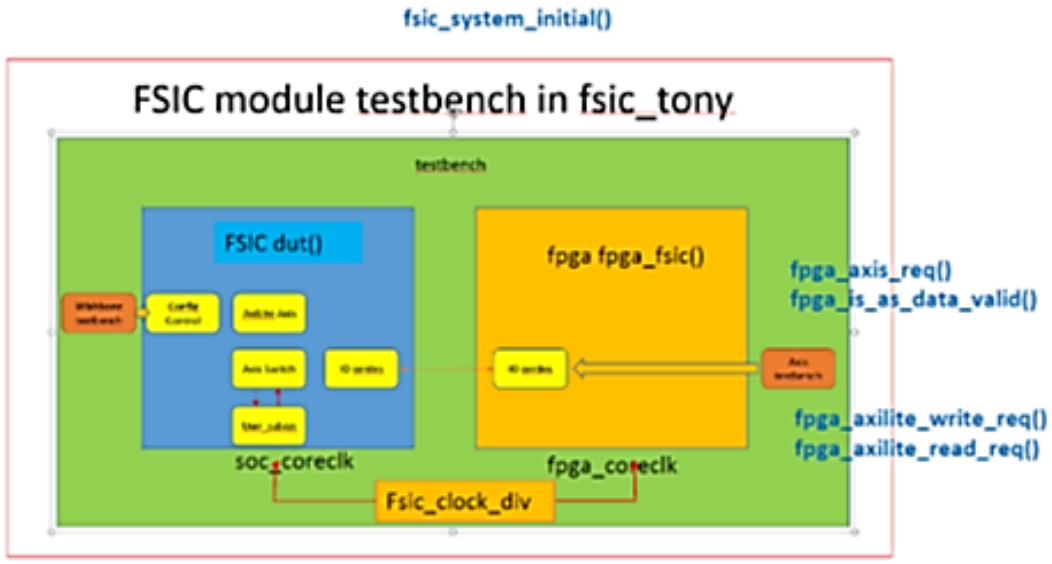
Block Diagram - Test Code Flow



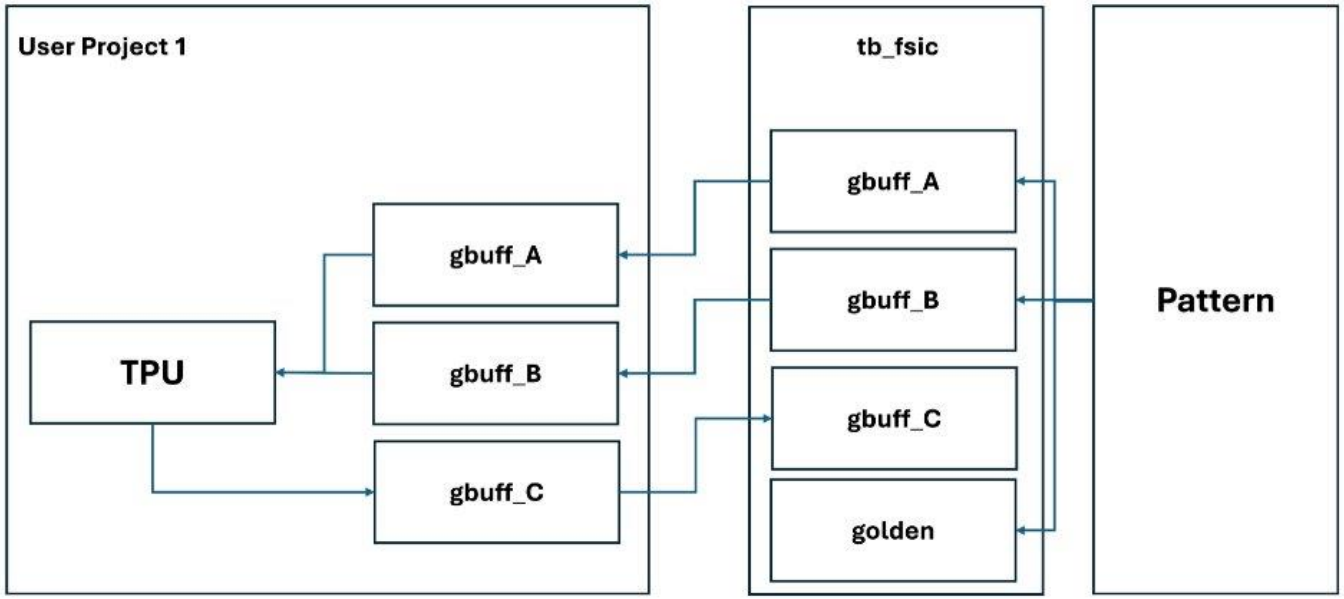
Simulation in IP Level

- Config from SoC
- Config from FPGA

Blcok Diagram



Blcok Diagram



Generate Test Pattern -- Pattern 0

- Matrix Size less than 4*4

```
-------------------------------------------
                Case 0
-------------------------------------------
K:   2 M:   2 N:  2
A:
   196    150
   177    208

B:
    23     77
    40     73
```

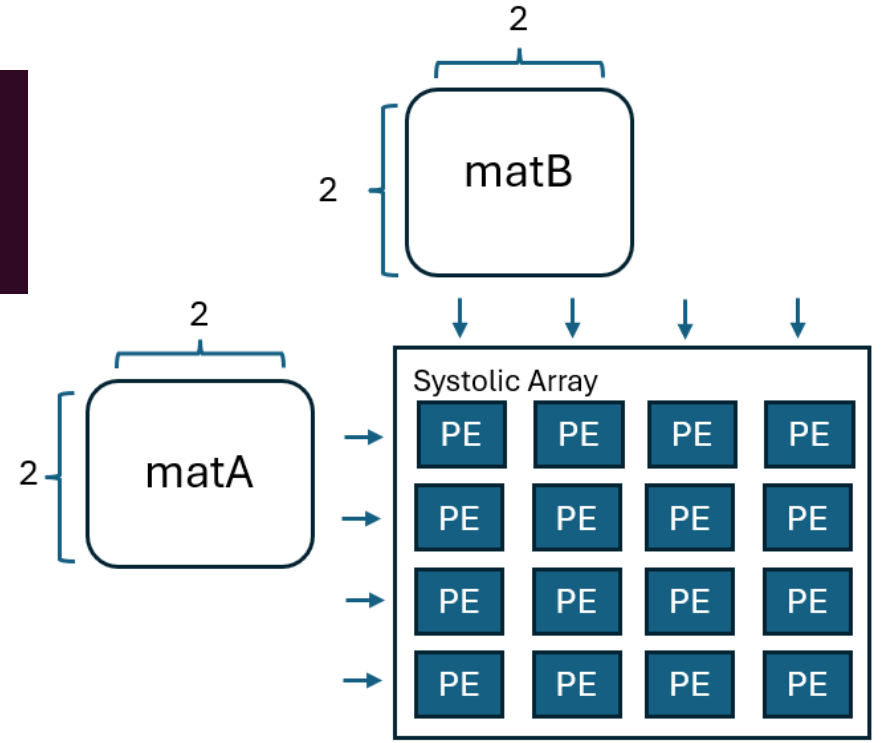
```
C:
 10508  26042
 12391  28813
```

Convert to hex

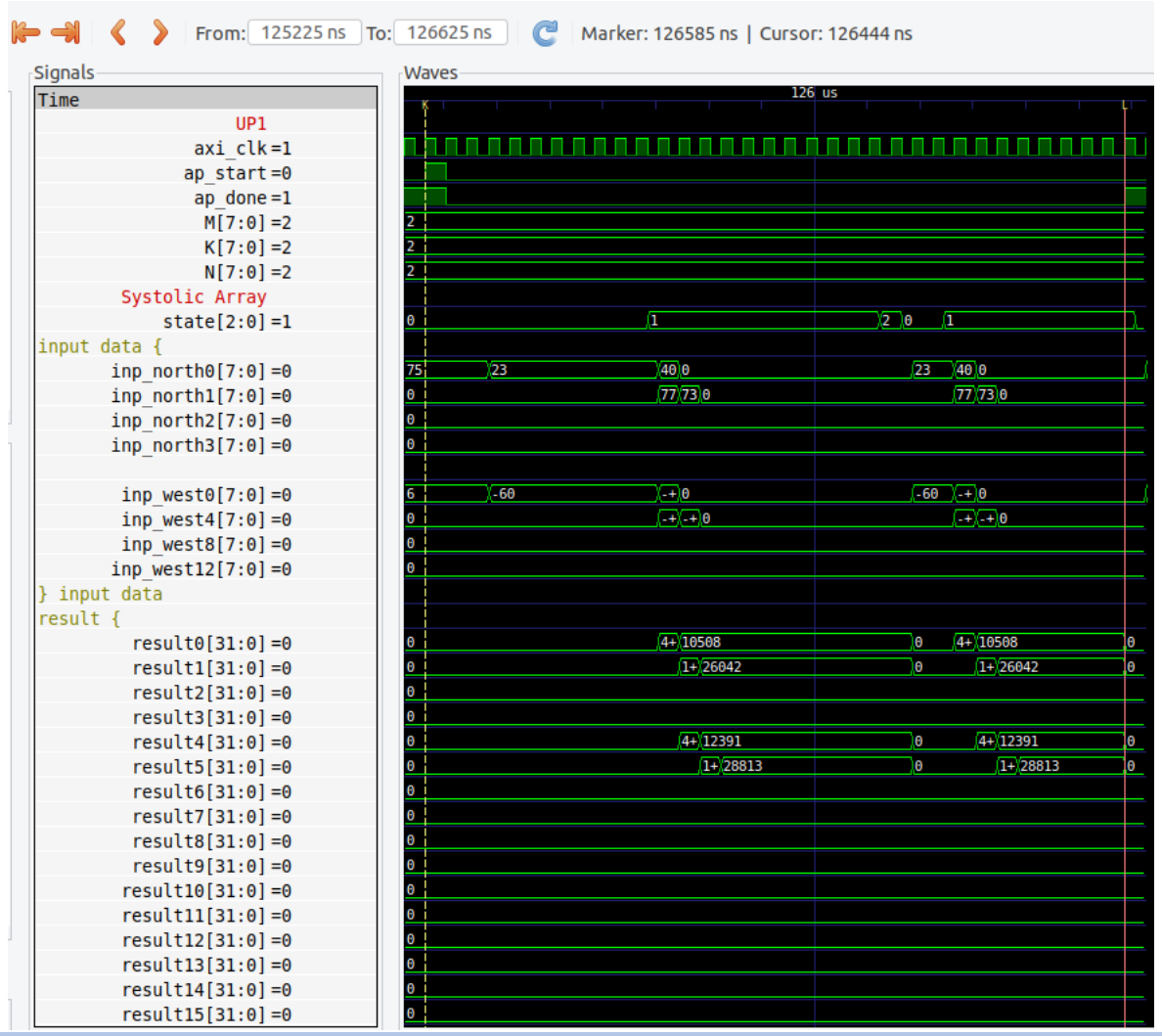
```
     290c      65ba         0         0
     3067      708d         0         0
```

Simulation Result

```
GOLDEN[          0] = 0000290c 000065ba 00000000 00000000
GOLDEN[          1] = 00003067 0000708d 00000000 00000000
patcount[          0] ap_start = 1
patcount[          0] ap_start = 0
C[          0] = 0000290c000065ba0000000000000000
C[          1] = 000030670000708d0000000000000000
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ Yes, you pass answoer
```



Wavform -- Pattern 0



Generate Test Pattern -- Pattern 1

- Matrix Size the same as 4*4

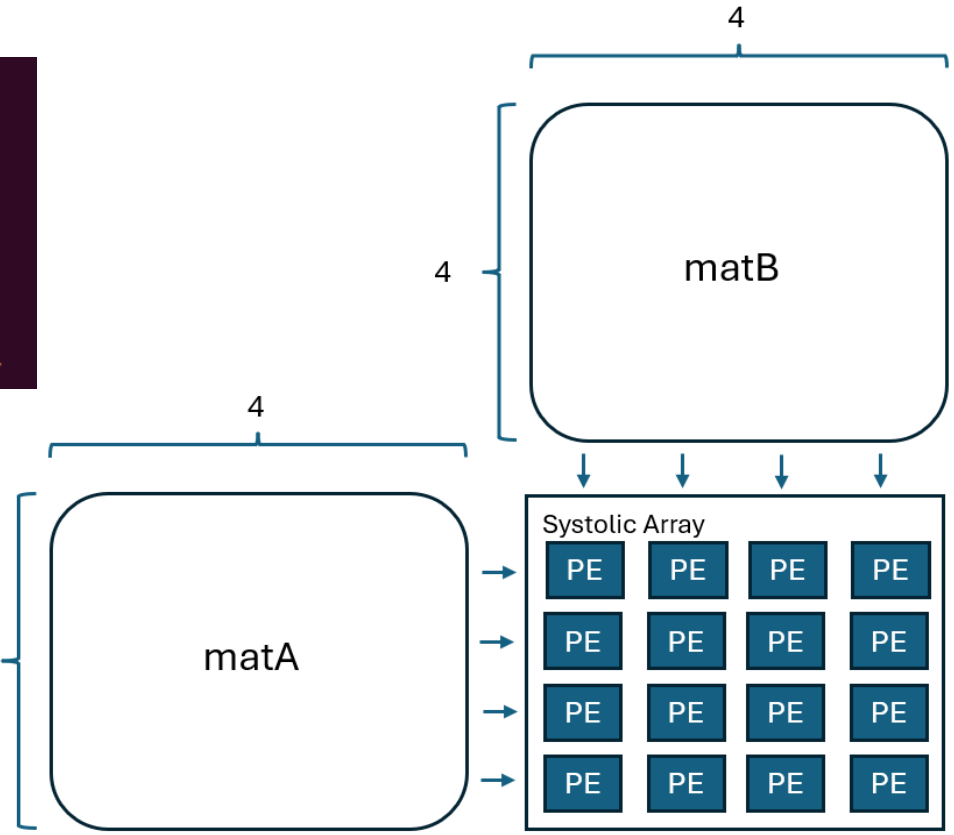
```
-------------------------------------------
                Case 0
-------------------------------------------
K:   4 M:   4 N:  4
A:
    34     60    172    105
   206     95    191    119
   219     74     38    157
   155    221    201      6

B:
   106    148     81      0
   245     19    118     36
   108    135    136    178
    55    162    188    147

C:
 42655  46402  52966  48211
 72284  77356  76244  54911
 54083  64382  61155  32507
 92613  55246  67097  44616
```

Convert to hex

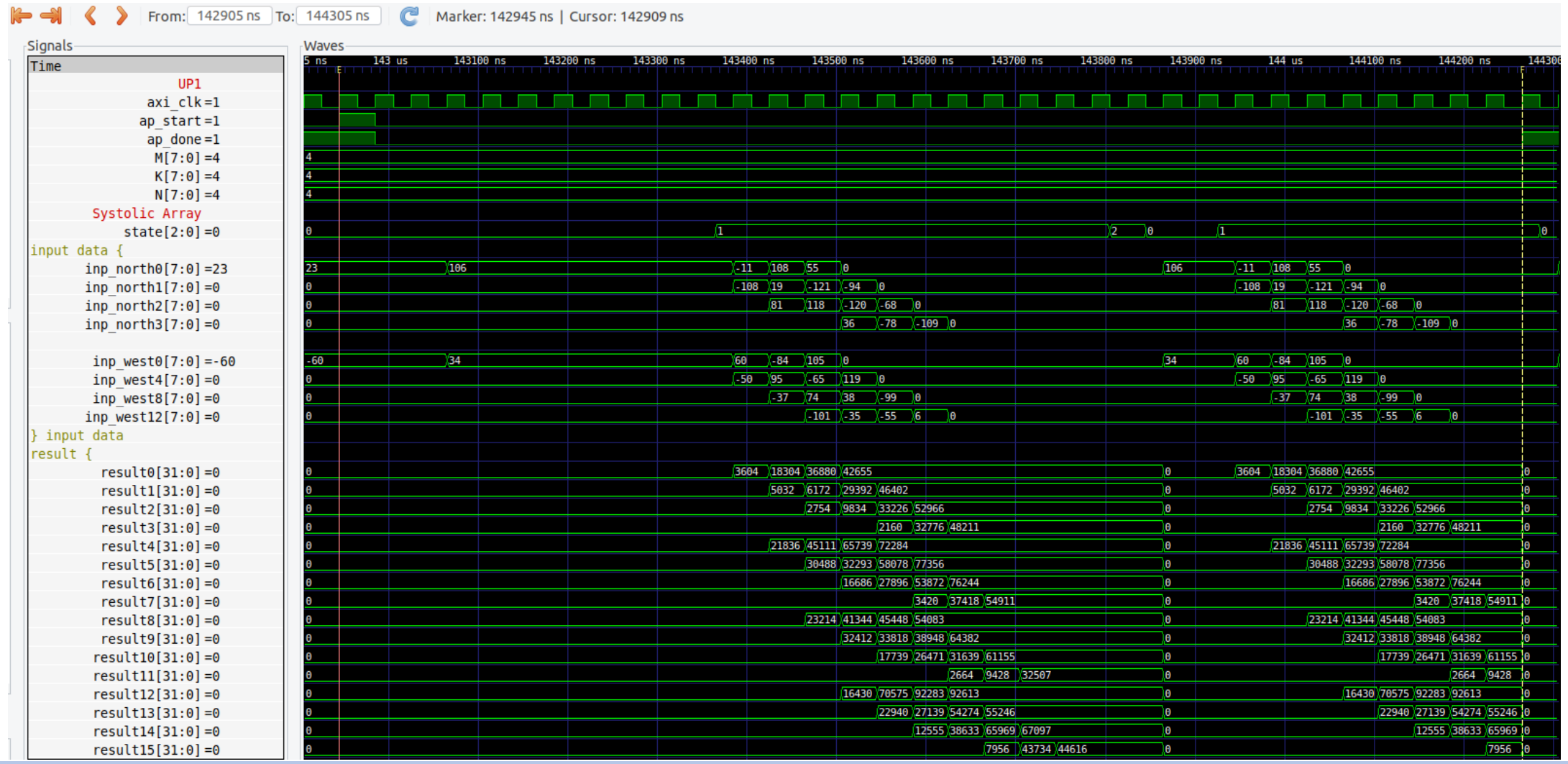
a69f	b542	cee6	bc53
11a5c	12e2c	129d4	d67f
d343	fb7e	eee3	7efb
169c5	d7ce	10619	ae48

Simulation Result

```
GOLDEN[          0] = 0000a69f 0000b542 0000cee6 0000bc53
GOLDEN[          1] = 00011a5c 00012e2c 000129d4 0000d67f
GOLDEN[          2] = 0000d343 0000fb7e 0000eee3 00007efb
GOLDEN[          3] = 000169c5 0000d7ce 00010619 0000ae48
patcount[          0] ap_start = 1
patcount[          0] ap_start = 0
C[          0] = 0000a69f0000b5420000cee60000bc53
C[          1] = 00011a5c00012e2c000129d40000d67f
C[          2] = 0000d3430000fb7e0000eee300007efb
C[          3] = 000169c50000d7ce000106190000ae48
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ Yes, you pass answoer
```



Wavform -- Pattern 1



Generate Test Pattern -- Pattern 2

- Matrix Size partial larger than 4*4

```
-----------------------------------------------
                Case 0
-----------------------------------------------
K:  15 M:   4 N:  4
A:
   182    141    240    169    148    126    130     67    155    124     41      5    249     32    164
   248    232    246    238      8    129    146     81    195    192    187     20     69    200     55
   244    100      9    158    149    174     16    245     86    228    107     87    178    239    123
   177     57     59     56    120      9    100     24    166    170    134    115     96    156      6

B:
   160     81     27    172
    48    151    254    244
   224     70    144    232
    43     13    230     23
   241     78    111      4
   188    235    142    178
   122    191     15    121
    49     99     30    214
    64    252    149    177
    11     41     33      3
    49    251    220     85
   110     80    247    159
   167    214     74    170
    35    237    209    152
   156    236    197     95
```

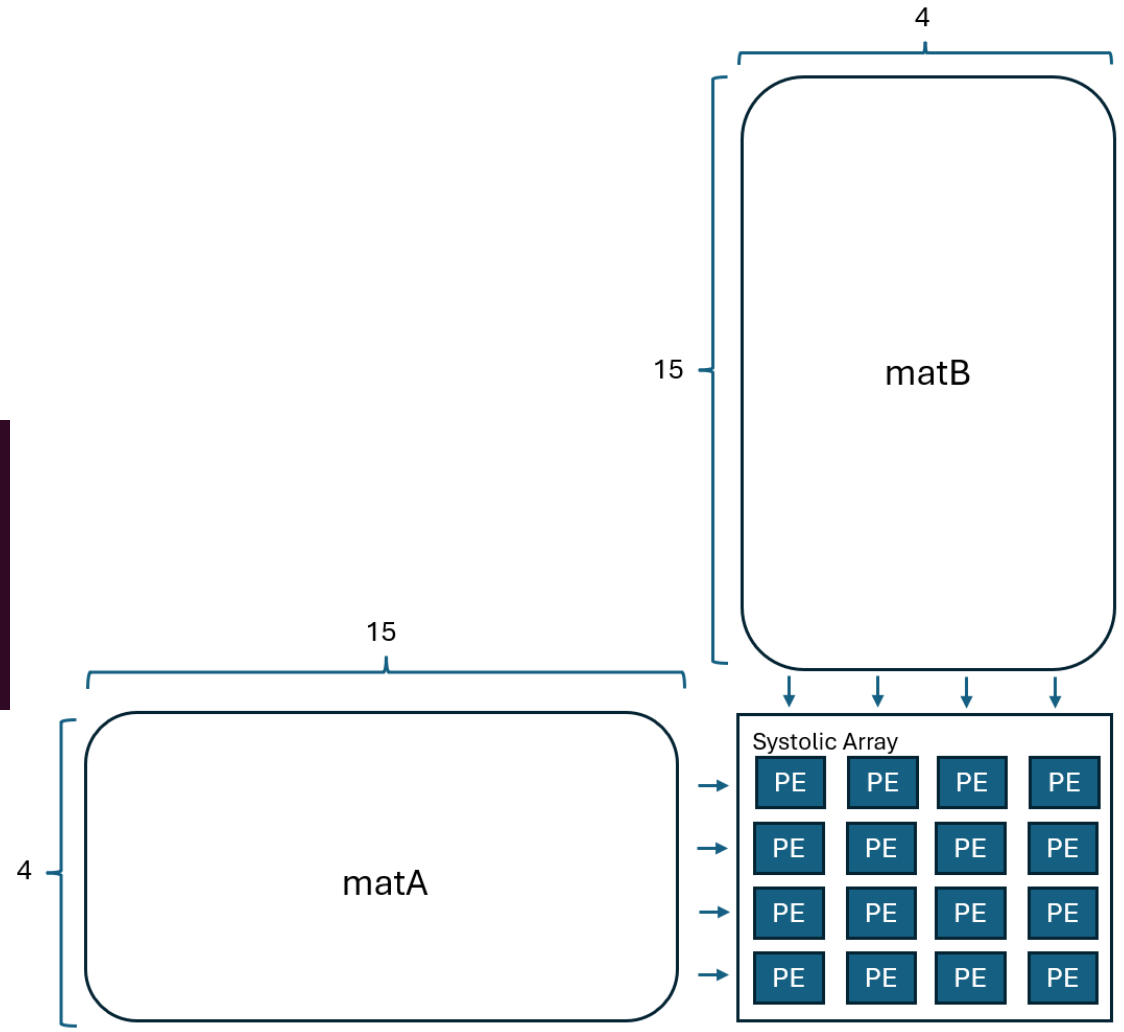
```
GOLDEN[           0] = 0003ee08 00044dc8 0003c606 00042b48
GOLDEN[           1] = 00035055 0004ed9d 0004dd6a 0004e72d
GOLDEN[           2] = 00034924 0004ab33 0004234e 000432bb
GOLDEN[           3] = 000235a6 00033979 0002d8ca 0002ba7f
patcount[           0] ap_start = 1
patcount[           0] ap_start = 0
C[           0] = 0003ee0800044dc80003c60600042b48
C[           1] = 000350550004ed9d0004dd6a0004e72d
C[           2] = 000349240004ab330004234e000432bb
C[           3] = 000235a6000339790002d8ca0002ba7f
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ Yes, you pass answoer
```

Simulation Result

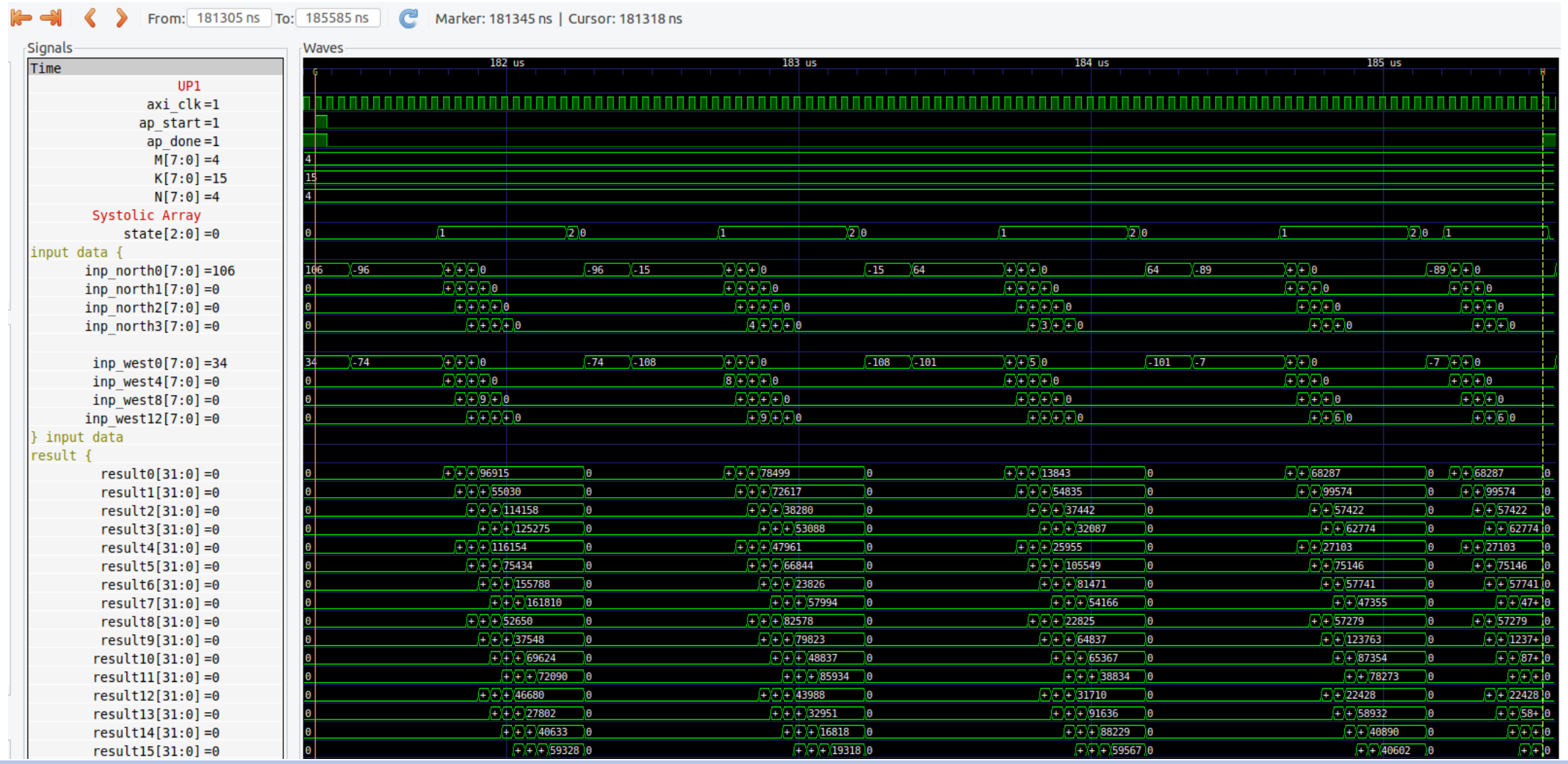
```
C:
257544 282056 247302 273224
217173 322973 318826 321325
215332 305971 271182 275131
144806 211321 186570 178815
```

Convert to hex

3ee08	44dc8	3c606	42b48
35055	4ed9d	4dd6a	4e72d
34924	4ab33	4234e	432bb
235a6	33979	2d8ca	2ba7f



Wavform -- Pattern 2



Generate Test Pattern -- Pattern 3

- Matrix Size greater than 4*4

```
-------------------------------------------
                Case 0
-------------------------------------------
K:  13 M:   4 N:  9
A:
    40    159      9    111    222    129     30    125    110     17     71    140      6
   150    163    246    201    127    117    221     60    112     89     45     68     88
   107    133    233    104    104    154    229     47    253     45    212    138    197
   239     34     62     33    193     49     74     77    188    201     74    148    208

B:
   183     92    131     79    123    172     78    222    210
    88    104    254     43      0    109     35     47    127
   241    170    131     26    103    184     87    143     41
   209     59    107      4    206    154    232    219    252
   238    104    224    234     52    240    253     69    163
   187    162    186     33     39    234     71     60    112
   219    100    212    166    226    113     27    100     50
   174    200     82    115    140    197     73     56    222
    77    254    133     38     22    104    124     39    101
   205     63     67     89    212    135    182    192    130
    23      3    205    246    197    206     81    168     38
   182    135      2    240    236    235    192     48    198
    96     31    161     14     79     32    102    198     75
```

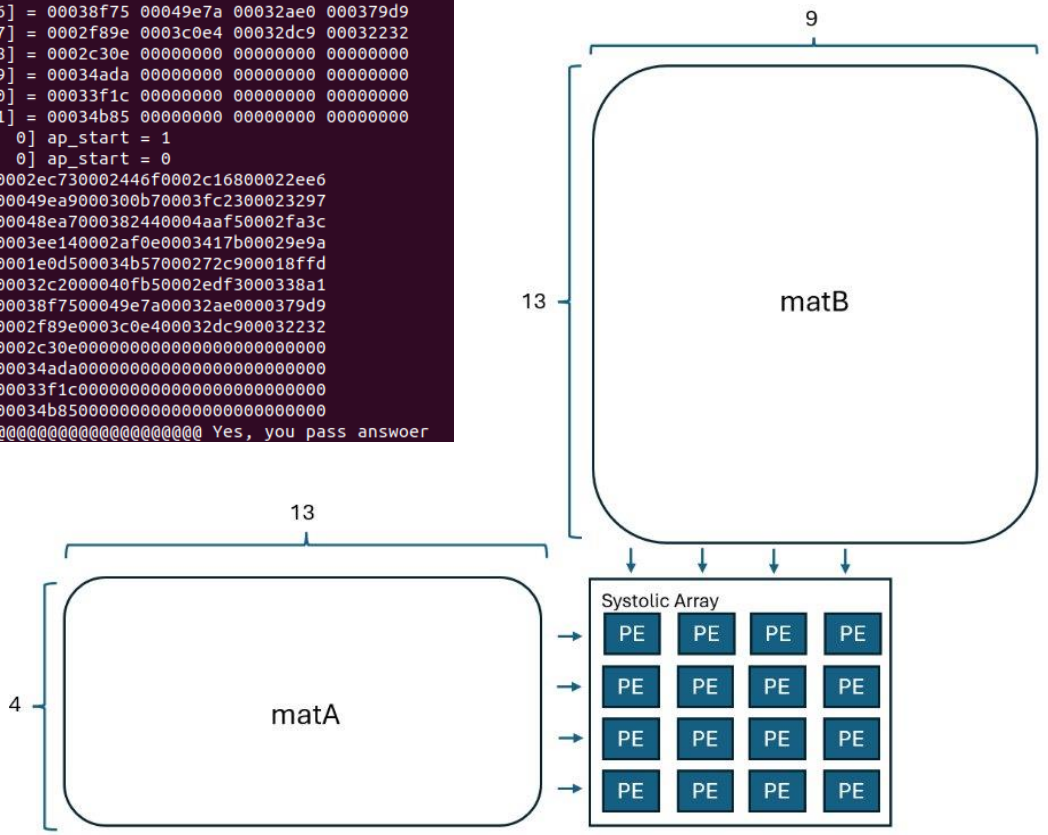
Simulation Result

```
GOLDEN[          0] = 0002ec73 0002446f 0002c168 00022ee6
GOLDEN[          1] = 00049ea9 000300b7 0003fc23 00023297
GOLDEN[          2] = 00048ea7 00038244 0004aaf5 0002fa3c
GOLDEN[          3] = 0003ee14 0002af0e 0003417b 00029e9a
GOLDEN[          4] = 0001e0d5 00034b57 000272c9 00018ffd
GOLDEN[          5] = 00032c20 00040fb5 0002edf3 000338a1
GOLDEN[          6] = 00038f75 00049e7a 00032ae0 000379d9
GOLDEN[          7] = 0002f89e 0003c0e4 00032dc9 00032232
GOLDEN[          8] = 0002c30e 00000000 00000000 00000000
GOLDEN[          9] = 00034ada 00000000 00000000 00000000
GOLDEN[         10] = 00033f1c 00000000 00000000 00000000
GOLDEN[         11] = 00034b85 00000000 00000000 00000000
patcount[          0] ap_start = 1
patcount[          0] ap_start = 0
C[          0] = 0002ec730002446f0002c16800022ee6
C[          1] = 00049ea9000300b70003fc2300023297
C[          2] = 00048ea70003824400004aaf50002fa3c
C[          3] = 0003ee140002af0e0003417b00029e9a
C[          4] = 0001e0d500034b57000272c900018ffd
C[          5] = 00032c2000040fb50002edf3000338a1
C[          6] = 00038f7500049e7a00032ae0000379d9
C[          7] = 0002f89e0003c0e400032dc900032232
C[          8] = 0002c30e00000000000000000000000
C[          9] = 00034ada00000000000000000000000
C[         10] = 00033f1c00000000000000000000000
C[         11] = 00034b8500000000000000000000000
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ Yes, you pass answoer
```

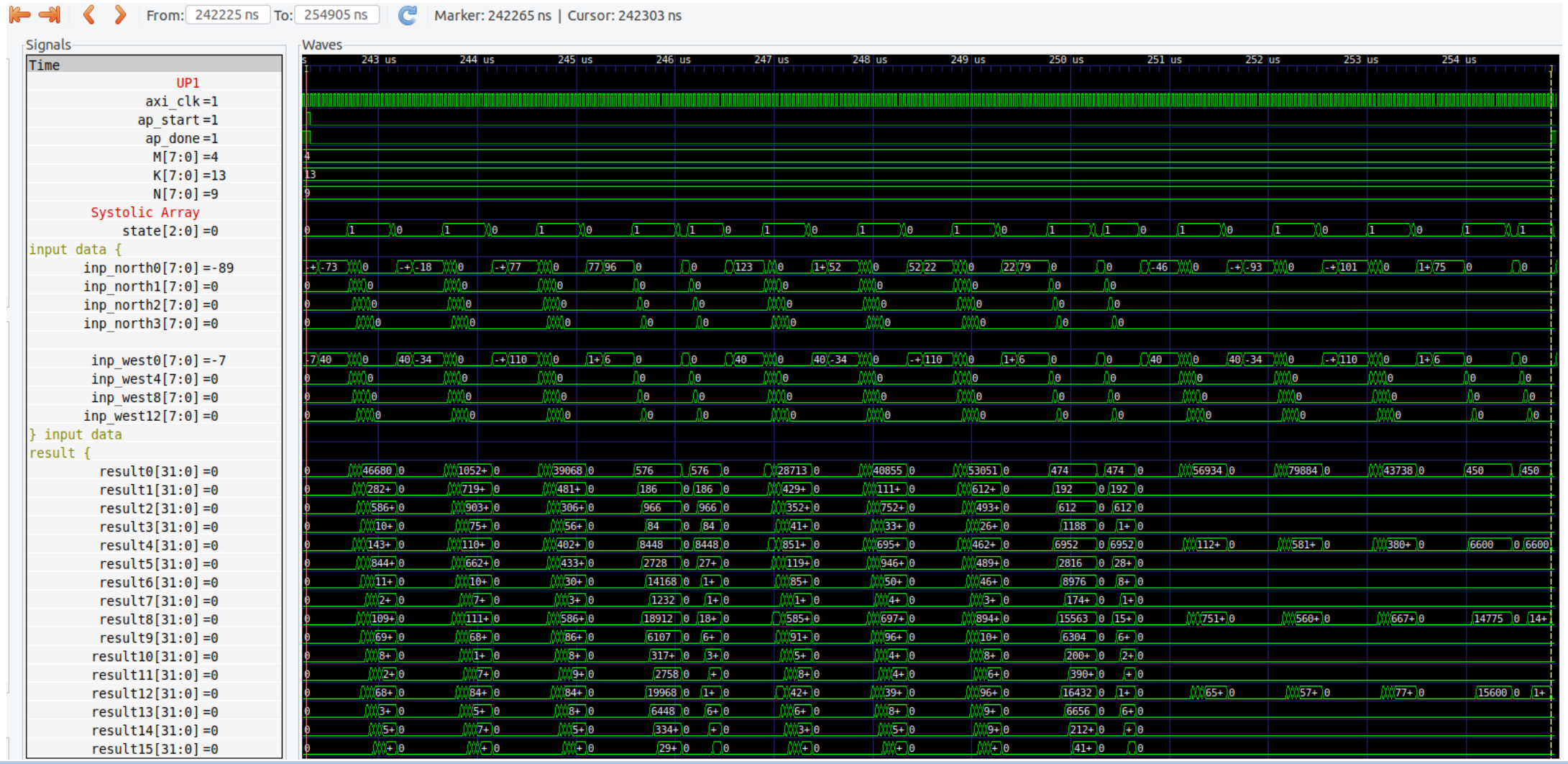
```
C:
191603 148591 180584 143078 123093 215895 160457 102397 181006
302761 196791 261155 144023 207904 266165 191987 211105 215770
298663 229956 305909 195132 233333 302714 207584 227801 212764
257556 175886 213371 171674 194718 245988 208329 205362 215941
```

Convert to hex

2ec73	2446f	2c168	22ee6
49ea9	300b7	3fc23	23297
48ea7	38244	4aaf5	2fa3c
3ee14	2af0e	3417b	29e9a
1e0d5	34b57	272c9	18ffd
32c20	40fb5	2edf3	338a1
38f75	49e7a	32ae0	379d9
2f89e	3c0e4	32dc9	32232
2c30e	0	0	0
34ada	0	0	0
33f1c	0	0	0
34b85	0	0	0

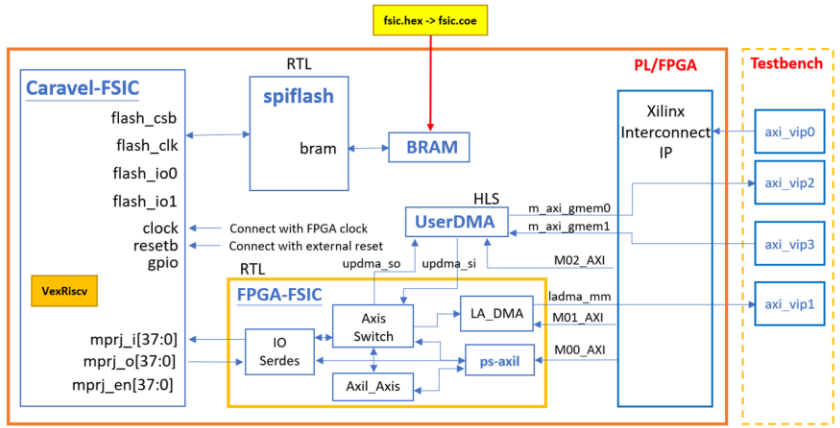


Wavform -- Pattern 3



Simulation in Vivado Flow

- Integrate with FSIC
  - Implement complete Caravel-FSIC and FPGA-FSIC



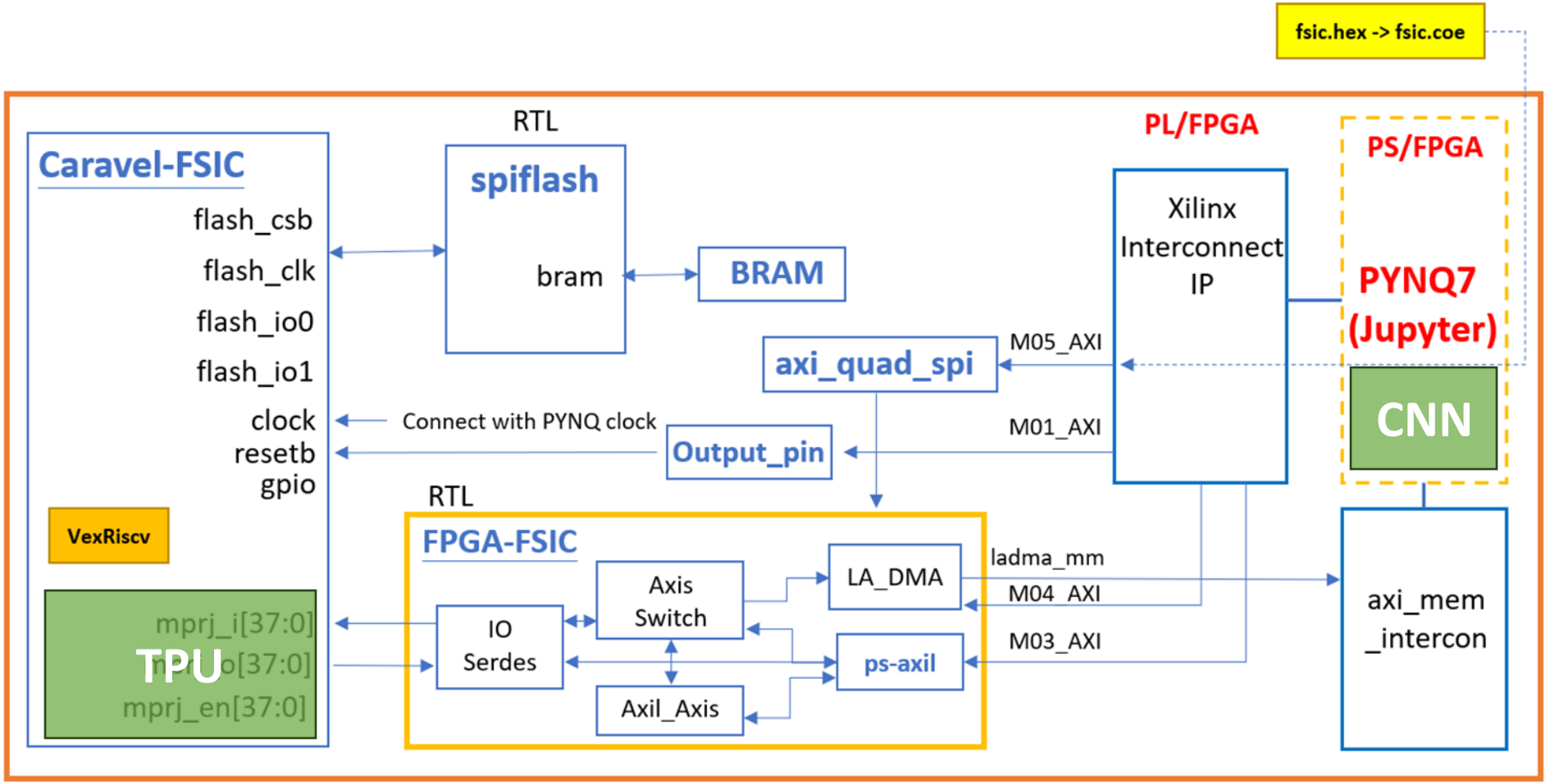

```
C[          8] = 0002c30e00000000000000000000000000
            10160858=> WriteCyc, offset 040 = 00000009, PASS
            10163458=> AXI4LITE_WRITE_BURST 60000040, value: 0009, resp: 00
            10175658=> AXI4LITE_READ_BURST 60000044, value: 0000, resp: 00
            10186258=> AXI4LITE_READ_BURST 60000048, value: 0000, resp: 00
            10196858=> AXI4LITE_READ_BURST 6000004c, value: 0000, resp: 00
            10207458=> AXI4LITE_READ_BURST 60000050, value: 34ada, resp: 00
C[          9] = 00034ada00000000000000000000000000
            10207458=> WriteCyc, offset 040 = 0000000a, PASS
            10210058=> AXI4LITE_WRITE_BURST 60000040, value: 000a, resp: 00
            10222258=> AXI4LITE_READ_BURST 60000044, value: 0000, resp: 00
            10232858=> AXI4LITE_READ_BURST 60000048, value: 0000, resp: 00
            10243458=> AXI4LITE_READ_BURST 6000004c, value: 0000, resp: 00
            10254058=> AXI4LITE_READ_BURST 60000050, value: 33f1c, resp: 00
C[         10] = 00033f1c00000000000000000000000000
            10254058=> WriteCyc, offset 040 = 0000000b, PASS
            10256658=> AXI4LITE_WRITE_BURST 60000040, value: 000b, resp: 00
            10268858=> AXI4LITE_READ_BURST 60000044, value: 0000, resp: 00
            10279458=> AXI4LITE_READ_BURST 60000048, value: 0000, resp: 00
            10290058=> AXI4LITE_READ_BURST 6000004c, value: 0000, resp: 00
            10300658=> AXI4LITE_READ_BURST 60000050, value: 34b85, resp: 00
C[         11] = 00034b8500000000000000000000000000
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ Yes, you pass answoer   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
================================================================================================
================================================================================================
================================================================================================
10399998.000 ns MSG fsic_tb, +100000 cycles, finish_flag=0,  repeat_cnt=0026
10799998.000 ns MSG fsic_tb, +100000 cycles, finish_flag=0,  repeat_cnt=0027
          10800676=> End of the test...
Executing Axi4 End Of Simulation checks
Executing Axi4 End Of Simulation checks
Executing Axi4 End Of Simulation checks
Executing Axi4 End Of Simulation checks
$finish called at time : 10800676 ns : File "/home/ubuntu/Desktop/ASoC_Final/vivado/fsic_tb.sv" Line 143
run: Time (s): cpu = 00:01:13 ; elapsed = 00:04:28 . Memory (MB): peak = 3034.938 ; gain = 0.000 ; free physical = 1786 ; free virtual = 6249
# exit
INFO: xsimkernel Simulation Memory Usage: 171564 KB (Peak: 199732 KB), Simulation CPU Usage: 252740 ms
INFO: [Common 17-206] Exiting Vivado at Sun Jun 16 12:13:51 2024...
======================================================================
                        vivado complete
======================================================================
ubuntu@ubuntu2004:~/Desktop/ASoC_Final/vivado$
```

System Integration

Section Overview

- System Integration
  - System overview
  - Unit Test
    - Matrix Multiply
  - Final System Validation Results

System Overview



Unit Test -- Matrix Multiply

In [148]:
```
# Pattern 0
filepath = "/home/xilinx/jupyter_notebooks/PS/input0.txt"
matA, matB, k, m, n = read_input_file(filepath)
matrix_mul(matA, matB, k, m, n)
```

Out[148]:
```
array([[10508, 26042],
       [12391, 28813]])
```

In [149]:
```
# Pattern 1
filepath = "/home/xilinx/jupyter_notebooks/PS/input1.txt"
matA, matB, k, m, n = read_input_file(filepath)
matrix_mul(matA, matB, k, m, n)
```

Out[149]:
```
array([[42655, 46402, 52966, 48211],
       [72284, 77356, 76244, 54911],
       [54083, 64382, 61155, 32507],
       [92613, 55246, 67097, 44616]])
```

In [150]:
```
# Pattern 2
filepath = "/home/xilinx/jupyter_notebooks/PS/input2.txt"
matA, matB, k, m, n = read_input_file(filepath)
matrix_mul(matA, matB, k, m, n)
```

Out[150]:
```
array([[257544, 282056, 247302, 273224],
       [217173, 322973, 318826, 321325],
       [215332, 305971, 271182, 275131],
       [144806, 211321, 186570, 178815]])
```

In [151]:
```
# Pattern 3
filepath = "/home/xilinx/jupyter_notebooks/PS/input3.txt"
matA, matB, k, m, n = read_input_file(filepath)
matrix_mul(matA, matB, k, m, n)
```

Out[151]:
```
array([[191603, 148591, 180584, 143078, 123093, 215895, 160457, 102397,
        181006],
       [302761, 196791, 261155, 144023, 207904, 266165, 191987, 211105,
        215770],
       [298663, 229956, 305909, 195132, 233333, 302714, 207584, 227801,
        212764],
       [257556, 175886, 213371, 171674, 194718, 245988, 208329, 205362,
        215941]])
```

Final System Validation Results

In [*]:

```
# Training
net = Network()

train_batch_num = (train_image_num  -  val_image_num  )//Batch_size
val_batch_num = (val_image_num)//Batch_size
# test_batch_num = test_image_num//Batch_size

print("[   0.000] start training...")
start_time = time()
for epoch in range(1, EPOCH+1):
    train_hit = 0
    val_hit = 0
    total_train_loss = 0
    total_val_loss = 0
    for it in range(train_batch_num):
        pred, train_loss = net.forward(train_data[it*Batch_size:(it+1)*Batch_size], trai
        pred_index = np.argmax(pred, axis=1)
        train_hit += (pred_index==train_label[it*Batch_size:(it+1)*Batch_size]).sum()
        total_train_loss += train_loss

        step = epoch * train_batch_num + it
        Learning_rate = d_model**(-0.5) * min(step**(-0.5), step*(warmup_steps**(1.5)))
        net.backward()
        net.update(0.1*Learning_rate)
        print('[%8.3f]'%(time()-start_time), 'train batch:', '%5d'%it, '/', '%5d'%train_

    for titt in range(val_batch_num):
        tit=train_batch_num+titt
        pred, val_loss = net.forward(train_data[tit*Batch_size:(tit+1)*Batch_size], trai
        pred_index = np.argmax(pred, axis=1)
        val_hit += (pred_index==train_label[tit*Batch_size:(tit+1)*Batch_size]).sum()
        total_val_loss += val_loss
        print('[%8.3f]'%(time()-start_time), 'val batch:', '%5d'%it, '/', '%5d'%train_ba

    run_time = time() - start_time
    print('[%8.3f]'%run_time,'Epoch:%3d'%epoch, '|Train Loss:%8.4f'%(total_train_loss/tra
          , '|Val Loss:%8.4f'%(total_val_loss/val_batch_num), '|Val Acc:%3.4f'%(val_hit/
```

```
[   0.000] start training...
[   4.163] train batch:     0 /  6900
[   8.308] train batch:     1 /  6900
[  12.439] train batch:     2 /  6900
[  16.586] train batch:     3 /  6900
[  20.722] train batch:     4 /  6900
[  24.880] train batch:     5 /  6900
[  29.019] train batch:     6 /  6900
[  33.058] train batch:     7 /  6900
[  37.112] train batch:     8 /  6900
[  41.240] train batch:     9 /  6900
[  45.364] train batch:    10 /  6900
[  49.481] train batch:    11 /  6900
[  53.624] train batch:    12 /  6900
[  57.755] train batch:    13 /  6900
[  61.911] train batch:    14 /  6900
[  65.963] train batch:    15 /  6900
[  70.028] train batch:    16 /  6900
[  74.125] train batch:    17 /  6900
```

Synthesis

QoR Report @500MHz

- Freqency: 500MHz
- Slack: 0
- Design Area: 112797

```
Timing Path Group 'axi_clk'
-----------------------------------
Levels of Logic:                12.00
Critical Path Length:            0.72
Critical Path Slack:             0.00
Critical Path Clk Period:        2.00
Total Negative Slack:            0.00
No. of Violating Paths:          0.00
Worst Hold Violation:            0.00
Total Hold Violation:            0.00
No. of Hold Violations:          0.00
-----------------------------------
```

```
Area
-----------------------------------
Combinational Area:    22803.840388
Noncombinational Area: 16279.304868
Buf/Inv Area:           8010.914588
Total Buffer Area:         6745.87
Total Inverter Area:       1265.04
Macro/Black Box Area:     0.000000
Net Area:              73714.018122
-----------------------------------
Cell Area:             39083.145256
Design Area:          112797.163378
```

QoR Report @200MHz

- Freqency: 200MHz
- Slack: 1.03
- Design Area: 109794

```
Timing Path Group 'axi_clk'
-----------------------------------
Levels of Logic:                17.00
Critical Path Length:            1.19
Critical Path Slack:             1.03
Critical Path Clk Period:        5.00
Total Negative Slack:            0.00
No. of Violating Paths:          0.00
Worst Hold Violation:            0.00
Total Hold Violation:            0.00
No. of Hold Violations:          0.00
-----------------------------------
```

```
Area
-----------------------------------
Combinational Area:    21845.199994
Noncombinational Area: 16279.082868
Buf/Inv Area:           7679.335399
Total Buffer Area:         6975.55
Total Inverter Area:        703.78
Macro/Black Box Area:     0.000000
Net Area:              71669.849116
-----------------------------------
Cell Area:             38124.282862
Design Area:          109794.131977
```

Analysis - Insight & Finding

Improvement and solution effectivenss

- Matrix Multiplication runtime
  - software: 1.624 ms
  - hardware: 7.229 ms

```
1 mmtestA = np.random.randn(8, 8)
2 mmtestB = np.random.randn(8, 8)
3 print(time_ns())
4 mmtestC = matrix_mul_sw(mmtestA.astype(np.int8), mmtestB.astype(np.int8))
5 print(time_ns())
6 mmtestC = matrix_mul_with_preproc(mmtestA.astype(np.int8), mmtestB.astype(np.int8))
7 print(time_ns())
```

```
1718710448772761067
1718710448774385189
1718710448781614901
```

- Actual compute time on Hardware
  - 1.276 ms

```
1 print(time_ns())
2 mmtestC = matrix_mul_with_preproc(mmtestA.astype(np.int8), mmtestB.astype(np.int8))
3 print(time_ns())
```

```
1718710859986394486
1718710859994650386
1718710859995927243
1718710859999895485
```

```
print(time_ns())
# ap_start
mmio.write(SOC_UP + TPU_CTRL_OFFSET, 0x01)
# wait ap_done

while (mmio.read(SOC_UP + TPU_CTRL_OFFSET) & 0x02) == 0:
    continue
print(time_ns())
```

- Some extra overhead on Hardware
  - Data pre-processing
  - Data transmission

Improvement and solution effectivenss

- CNN model training
  - 67.77x slower

```
[   0.000] start training...
[   0.057] train batch:     0 /  6900
[   0.116] train batch:     1 /  6900
[   0.175] train batch:     2 /  6900
[   0.240] train batch:     3 /  6900
[   0.299] train batch:     4 /  6900
[   0.358] train batch:     5 /  6900
[   0.417] train batch:     6 /  6900
[   0.481] train batch:     7 /  6900
[   0.540] train batch:     8 /  6900
[   0.599] train batch:     9 /  6900
[   0.659] train batch:    10 /  6900
[   0.722] train batch:    11 /  6900
[   0.782] train batch:    12 /  6900
[   0.841] train batch:    13 /  6900
[   0.899] train batch:    14 /  6900
[   0.963] train batch:    15 /  6900
[   1.022] train batch:    16 /  6900
[   1.080] train batch:    17 /  6900
[   1.139] train batch:    18 /  6900
[   1.203] train batch:    19 /  6900
[   1.265] train batch:    20 /  6900
```

PYNQ CPU: 1.265s for 20 batch

```
[   0.000] start training...
[   4.103] train batch:     0 /  6900
[   8.143] train batch:     1 /  6900
[  12.182] train batch:     2 /  6900
[  16.227] train batch:     3 /  6900
[  20.291] train batch:     4 /  6900
[  24.329] train batch:     5 /  6900
[  28.455] train batch:     6 /  6900
[  32.470] train batch:     7 /  6900
[  36.520] train batch:     8 /  6900
[  40.588] train batch:     9 /  6900
[  44.711] train batch:    10 /  6900
[  48.842] train batch:    11 /  6900
[  52.968] train batch:    12 /  6900
[  57.047] train batch:    13 /  6900
[  61.096] train batch:    14 /  6900
[  65.154] train batch:    15 /  6900
[  69.202] train batch:    16 /  6900
[  73.320] train batch:    17 /  6900
[  77.457] train batch:    18 /  6900
[  81.594] train batch:    19 /  6900
[  85.734] train batch:    20 /  6900
```

Accelerator: 85.734s for 20 batch

Overhead of software tiling

- Tile size = 8 v.s. Tile size = 4
  - 1.97x slower

```
[   0.000] start training...
[   4.103] train batch:     0 /  6900
[   8.143] train batch:     1 /  6900
[  12.182] train batch:     2 /  6900
[  16.227] train batch:     3 /  6900
[  20.291] train batch:     4 /  6900
[  24.329] train batch:     5 /  6900
[  28.455] train batch:     6 /  6900
[  32.470] train batch:     7 /  6900
[  36.520] train batch:     8 /  6900
[  40.588] train batch:     9 /  6900
[  44.711] train batch:    10 /  6900
[  48.842] train batch:    11 /  6900
[  52.968] train batch:    12 /  6900
[  57.047] train batch:    13 /  6900
[  61.096] train batch:    14 /  6900
[  65.154] train batch:    15 /  6900
[  69.202] train batch:    16 /  6900
[  73.320] train batch:    17 /  6900
[  77.457] train batch:    18 /  6900
[  81.594] train batch:    19 /  6900
[  85.734] train batch:    20 /  6900
```

Tile size = 8: 85.734s for 20 batch

```
[   0.000] start training...
[   7.962] train batch:     0 /  6900
[  16.015] train batch:     1 /  6900
[  24.122] train batch:     2 /  6900
[  32.179] train batch:     3 /  6900
[  40.311] train batch:     4 /  6900
[  48.373] train batch:     5 /  6900
[  56.488] train batch:     6 /  6900
[  64.448] train batch:     7 /  6900
[  72.536] train batch:     8 /  6900
[  80.711] train batch:     9 /  6900
[  88.776] train batch:    10 /  6900
[  96.869] train batch:    11 /  6900
[ 104.887] train batch:    12 /  6900
[ 113.016] train batch:    13 /  6900
[ 121.114] train batch:    14 /  6900
[ 129.024] train batch:    15 /  6900
[ 137.072] train batch:    16 /  6900
[ 145.071] train batch:    17 /  6900
[ 153.121] train batch:    18 /  6900
[ 161.156] train batch:    19 /  6900
[ 169.224] train batch:    20 /  6900
```

Tile size = 4: 169.224s for 20 batch

Conclusion

- The huge overhead of data transmission
  - DMA is needed
- The huge overhead of software tiling
  - Use SRAM instead of registers as the data buffer
  - Tiling and MAC in hardware
- Inefficient hardware and software cooperation
  - Optimize the interface between hardware and software
  - Optimize the hardware for the application(ex: buffer size, PE array size)

What we learn from the final project

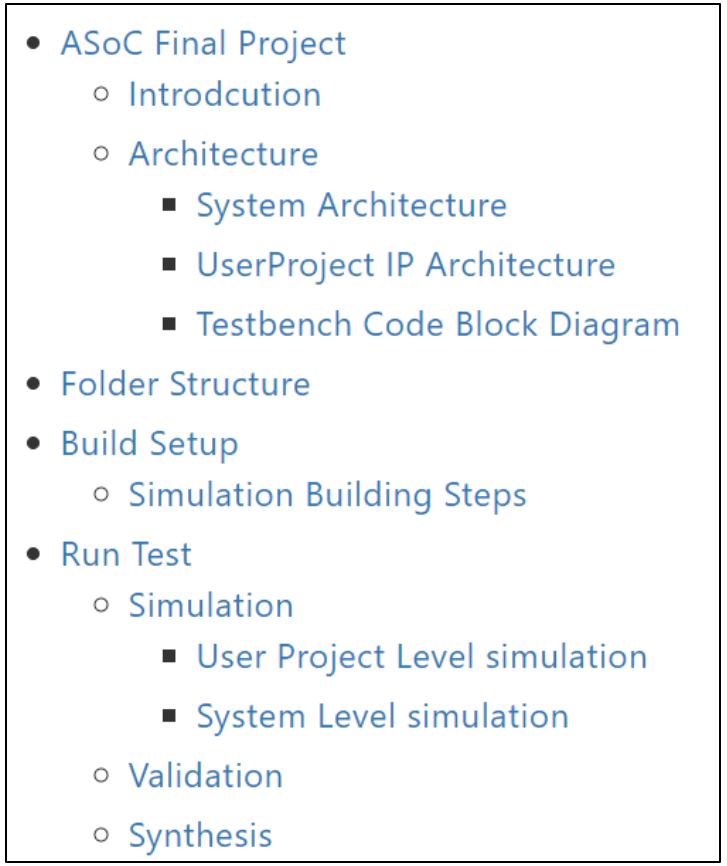
- Pattern generation is quite important
  - Ensures Consistency:
    - Validates each stage of development against a common set of patterns.
  - Enhances Communication:
    - Promotes effective communication and collaboration among team members.

- Practice running a software application through hardware acceleration

Reference

Github

- https://github.com/michael61112/ASoC_Final



Reference

- Machine Heart's Deep Dive into Google TPU Architecture
- Understanding Matrix Multiplication on a Weight-Stationary Systolic Architecture
- Exploration of TPUs for AI Applications
- Scale-out Systolic Arrays
- Matrix multiplication tiling