

**EE 789**  
**Algorithmic Design of**  
**Digital Systems**  
**Assignment 2**  
**Matrix Multiplication**



Name:	<b>Satvikkumar Patel</b>
Roll number:	23M1117
Program:	M.tech (EE5 - Electronic Systems)
Department:	Electrical Engineering
Date of Sub:	November 17, 2023

# EE 789

## Assignment 2: Matrix Multiplication

October 9, 2023

You are given two 16x16 matrices  $A$  and  $B$  and are expected to design a circuit which multiplies these matrices.

If  $A = [a_{ij}]$  and  $B = [b_{ij}]$  ( $0 \leq i, j \leq 15$ ) then the product matrix  $C = A \times B$  has entries  $c_{ij}$  with

$$c_{ij} = \sum_{k=0}^{15} a_{ik} b_{kj}$$

A 16x16 matrix can be viewed in row-form or column form. In row-form, we can write

$$A = \begin{pmatrix} rA_0^T \\ rA_1^T \\ \vdots \\ rA_{15}^T \end{pmatrix}$$

where  $rA_0^T$  is the first row and so on. In column form, we can write  $A$  as

$$A = (cA_0 \ cA_1 \ \dots \ cA_{15})$$

where  $cA_0^T$  is the first column and so on. Similarly for  $B$ .

The product matrix  $C$  can be built in different ways.

$$c_{ij} = rA_i^T \cdot cB_j^T$$

or (as a sum of rank-1 matrices):

$$C = \sum_{i=0}^{15} cA_i \cdot rB_i^T$$

## 1 Sample code

I will share sample code for the multiplication of two 16x16 matrices. This includes a dot-product based implementation and a test-bench.

## 2 Assignment

You will attempt to implement the mmul routine in various ways. The test-bench is unchanged.

- Implement the matrix multiplication by speeding up the dot product implementation used in the sample code (10).
- Parallelize the matrix multiplication by dividing  $A$  and  $B$  into four 8x8 blocks, doing work on the 8x8 blocks and then combining this work to produce the final result (10).
- Implement the matrix product using the sum of rank-1 matrices approach (10).

In each case, you will have to measure the time required by the mmul routine in your implementation.

## Sample code:

```

1 $parameter ORDER 16
2
3 //
4 // observation signals to keep track of progress.
5 //
6 $pipe mmul_I : $uint<8> $signal
7 $pipe mmul_J : $uint<8> $signal
8 $pipe mmul_START: $uint<8> $signal
9 $pipe mmul_END: $uint<8> $signal
10
11 $storage A B C: $array[ORDER][ORDER] $of $uint<32>
12
13 $module [storeA] $in (I J: $uint<8> wval: $uint<32>) $out () $is
14 {
15     A[I][J] := wval
16 }
17 $module [storeB] $in (I J: $uint<8> wval: $uint<32>) $out () $is
18 {
19     B[I][J] := wval
20 }
21
22 $module [loadC] $in (I J : $uint<8>) $out (Y : $uint<32>) $is
23 {
24     Y := C[I][J]
25 }
26
27
28 $module [dot_product] $in (I J: $uint<8>) $out (result: $uint<32>) $is
29 {
30     $branchblock[loop] {
31
32         $dopipeline $depth 31 $fullrate
33         $merge $entry $loopback
34         $phi K := $zero<8> $on $entry nK $on $loopback
35         $phi SUM0 := ($bitcast ($uint<32>) 0) $on $entry nSUM0 $on $loopback
36         $phi SUM1 := ($bitcast ($uint<32>) 0) $on $entry nSUM1 $on $loopback
37         $endmerge
38         $volatile nK := (K + 2)
39         $volatile continue_flag := (K < (ORDER-2))
40         $volatile K1 := (K+1)
41
42         nSUM0 := (SUM0 + (A[I][K] * B[K][J]))
43         nSUM1 := (SUM0 + (A[I][K1] * B[K1][J]))
44         $while continue_flag
45     } (nSUM0 => R0 nSUM1 => R1)
46     $volatile result := (R0 + R1)
47 }
48
49 $module [mmul] $in () $out () $is
50 {
51     mmul_START := 1
52     $branchblock[loop] {
53         u := mmul_START v := mmul_END // for logging
54         $merge $entry I_loopback
55         $phi I := $zero<8> $on $entry nI $on I_loopback
56         $endmerge
57         $volatile nI := (I + 1)
58
59         $dopipeline $depth 3

```

```

60     $merge $entry $loopback
61     $phi J := $zero<8> $on $entry nJ $on $loopback
62     $endmerge
63     $volatile nJ := (J + 1)
64     $volatile continue_flag := (J < (ORDER - 1))
65
66     // dot-product!
67     C[I][J] := ($call dot_product (I J))
68
69     mmul_I := I
70     mmul_J := J
71
72     $while continue_flag
73
74     $if (I < (ORDER-1)) $then $place [I_loopback] $endif
75 }
76 mmul_END := 1
77 }

```

## TestBench:

```

1  #include <signal.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <stdint.h>
5  #include <pthread.h>
6  #include <pthreadUtils.h>
7  #include <Pipes.h>
8  #include <pipeHandler.h>
9  #ifndef SW
10 #include "vhdlCStubs.h"
11 #endif
12
13 #define ORDER 16
14
15 // Give inputs as
16 // Matrix A:
17 // 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
18 // 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
19 // 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
20 // 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
21 // ...
22
23 // Matrix B:
24 // 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
25 // 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
26 // 3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3
27 // 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
28 // ...
29
30
31 int main(int argc, char* argv[])
32 {
33     int I, J;
34     for(I = 0; I < ORDER; I++)
35     {
36         for(J = 0; J < ORDER; J++)
37         {
38             storeA(I, J, (uint32_t) I+1);
39             storeB(I, J, (uint32_t) J+1);
40         }
41     }

```

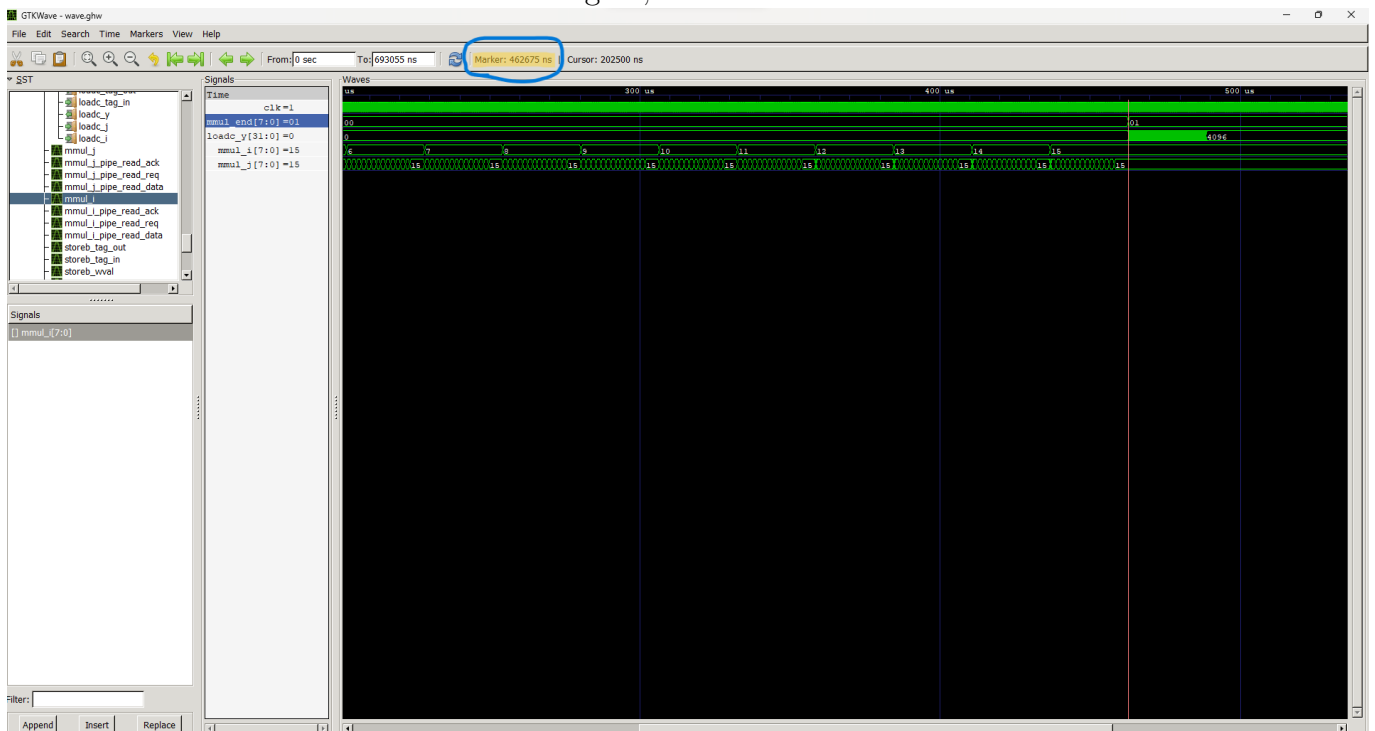
```

42  fprintf(stderr,"Stored A, B\n");
43
44  mmul();
45
46  fprintf(stderr,"finished dot_product, results:\n");
47  for(I = 0; I < ORDER; I++)
48  {
49      for(J = 0; J < ORDER; J++)
50      {
51          uint32_t result = loadC (I,J);
52          fprintf(stderr,"C[%d][%d] = %d.\n",I, J, result);
53      }
54  }
55
56
57  return(0);
58 }

```

## Results:

The time taken for the mmul\_end variable to go 1, is 462675ns



## Faster Dot product:

The method used here is unrolling the pipe by 2. The matrix multiplication part is left as it is.

```

1 $parameter ORDER 16
2
3 //
4 // observation signals to keep track of progress.
5 //
6 $pipe mmul_I : $uint<8> $signal
7 $pipe mmul_J : $uint<8> $signal
8 $pipe mmul_START: $uint<8> $signal
9 $pipe mmul_END: $uint<8> $signal
10
11 $storage A B C: $array[ORDER][ORDER] $of $uint<32>
12
13 $module [storeA] $in (I J: $uint<8> wval: $uint<32>) $out () $is
14 {
15     A[I][J] := wval
16 }
17 $module [storeB] $in (I J: $uint<8> wval: $uint<32>) $out () $is
18 {
19     B[I][J] := wval
20 }
21
22 $module [loadC] $in (I J : $uint<8>) $out (Y : $uint<32>) $is
23 {
24     Y := C[I][J]
25 }
26
27
28 $module [dot_product] $in (I J: $uint<8>) $out (result: $uint<32>) $is
29 {
30     $branchblock[loop] {
31
32         $dopipeline $depth 31 $fullrate
33         $merge $entry $loopback
34         $phi K := $zero<8> $on $entry nK $on $loopback
35         $phi SUM0 := ($bitcast ($uint<32>) 0) $on $entry nSUM0 $on $loopback
36         $phi SUM1 := ($bitcast ($uint<32>) 0) $on $entry nSUM1 $on $loopback
37         $endmerge
38         $volatile nK := (K + 2)
39         $volatile continue_flag := (K < (ORDER-2))
40         $volatile K1 := (K+1)
41
42         nSUM0 := (SUM0 + (A[I][K] * B[K][J]))
43         nSUM1 := (SUM0 + (A[I][K1] * B[K1][J]))
44         $while continue_flag
45     } (nSUM0 => R0 nSUM1 => R1)
46     $volatile result := (R0 + R1)
47 }
48
49 $module [mmul] $in () $out () $is
50 {
51     mmul_START := 1
52     $branchblock[loop] {
53         u := mmul_START v := mmul_END // for logging
54         $merge $entry I_loopback
55         $phi I := $zero<8> $on $entry nI $on I_loopback
56         $endmerge
57         $volatile nI := (I + 1)
58     }

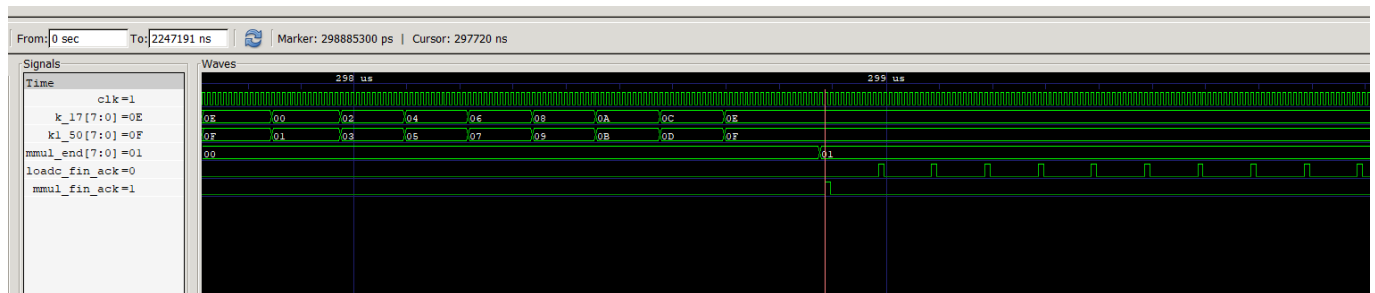
```

```

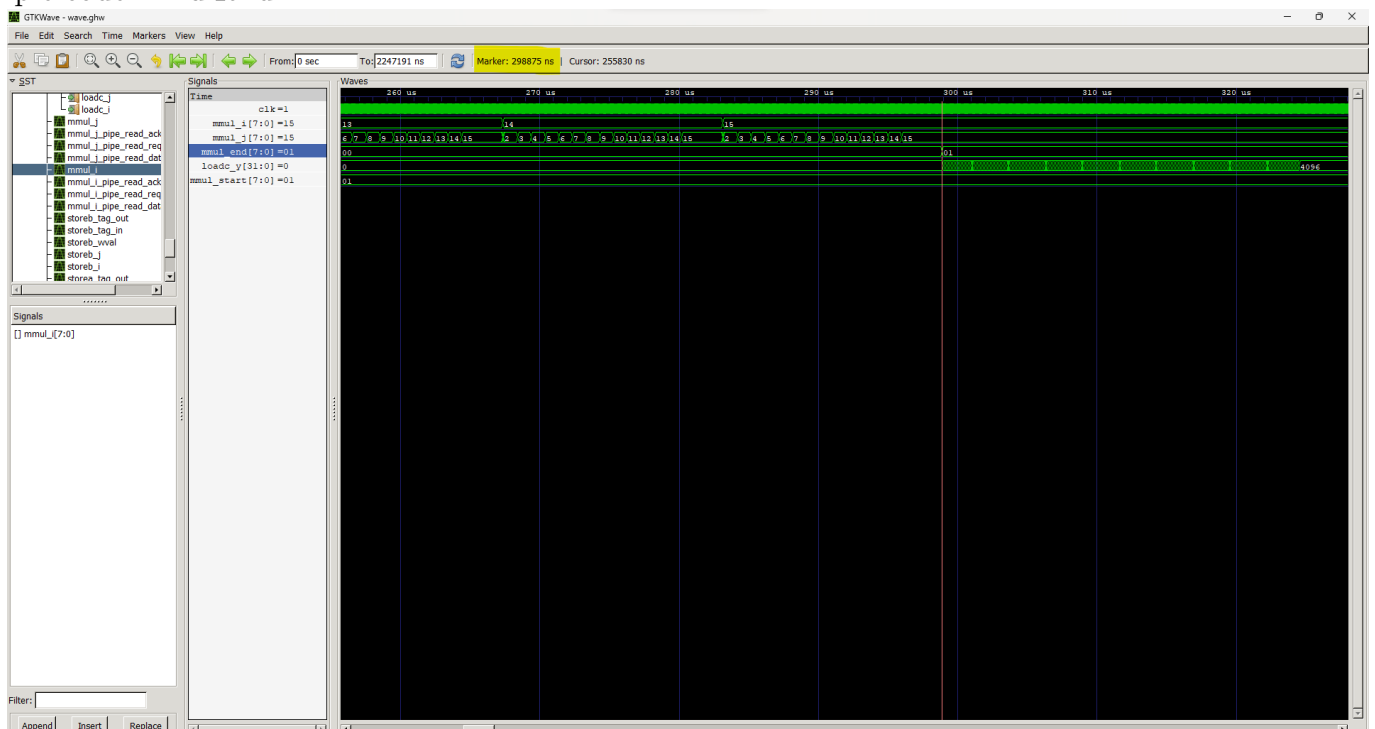
59  $dopipeline $depth 3
60  $merge $entry $loopback
61  $phi J := $zero<8> $on $entry nJ $on $loopback
62  $endmerge
63  $volatile nJ := (J + 1)
64  $volatile continue_flag := (J < (ORDER - 1))
65
66  // dot-product!
67  C[I][J] := ($call dot_product (I J))
68
69  mmul_I := I
70  mmul_J := J
71
72  $while continue_flag
73
74  $if (I < (ORDER-1)) $then $place [I_loopback] $endif
75 }
76 mmul_END := 1
77 }

```

## Results:

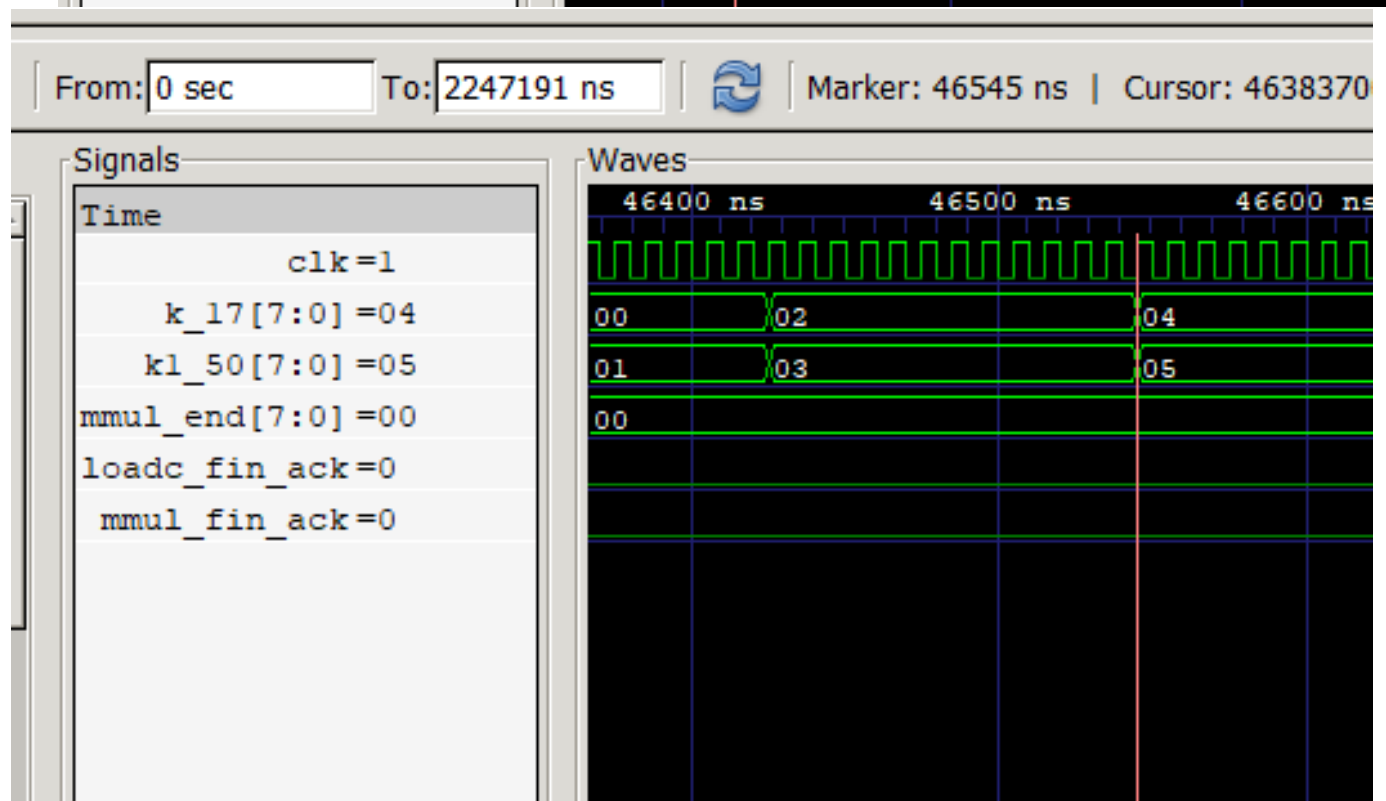
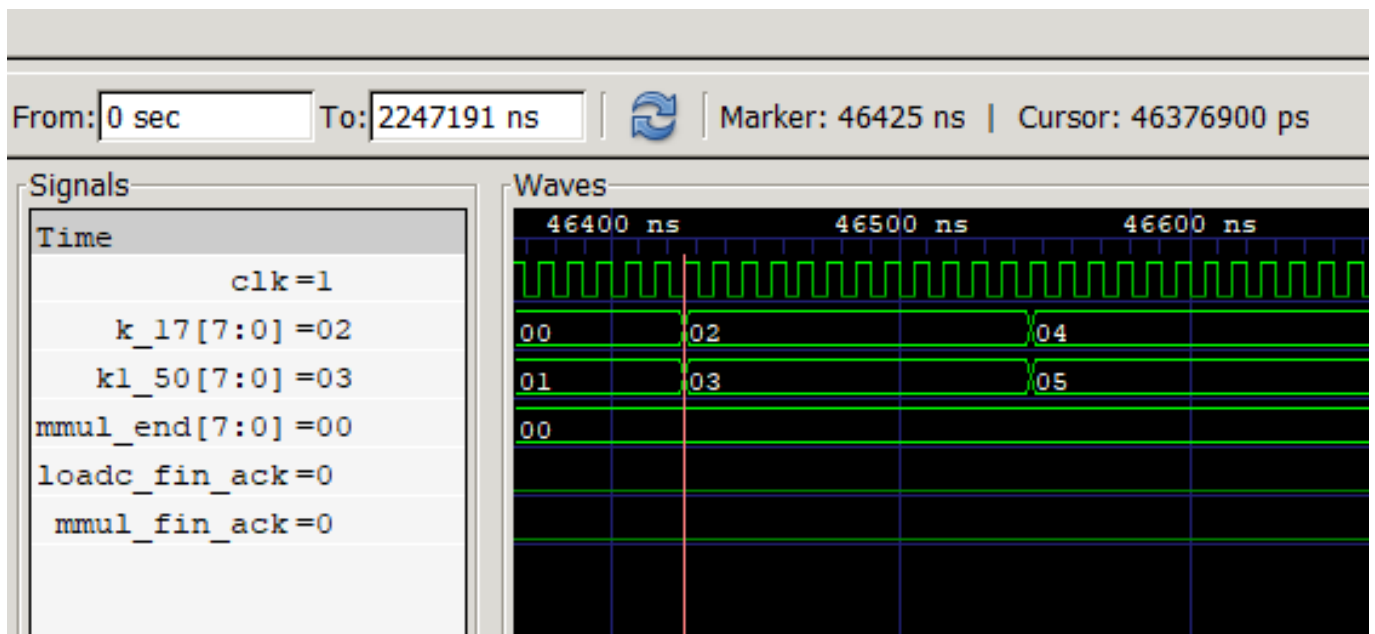


The time taken for the mmul\_end variable to go 1, is 298875ns, which is lesser compared to sample code mmul\_end.





The time taken for the loop variable to count up by 1 is,  $46545 - 46425 = 120\text{ns}$ . The clock time period being  $10\text{ns}$ , it takes 12 cycles for the loop interval to count up by 1.



# Matrix Multiplication by dividing the matrix

```

1 $parameter ORDER 16
2 $parameter ORDER_by_2 8
3
4 //
5 // observation signals to keep track of progress.
6 //
7 $pipe mmul_I : $uint<8> $signal
8 $pipe mmul_J : $uint<8> $signal
9 $pipe mmul_START: $uint<8> $signal
10 $pipe mmul_END: $uint<8> $signal
11
12 $storage A B C: $array[ORDER][ORDER] $of $uint<32>
13
14 $module [storeA] $in (I J: $uint<8> wval: $uint<32>) $out () $is
15 {
16     A[I][J] := wval
17 }
18 $module [storeB] $in (I J: $uint<8> wval: $uint<32>) $out () $is
19 {
20     B[I][J] := wval
21 }
22
23 $module [loadC] $in (I J : $uint<8>) $out (Y : $uint<32>) $is
24 {
25     Y := C[I][J]
26 }
27
28
29 $module [dot_product] $in (I J: $uint<8>) $out (result: $uint<32>) $is
30 {
31     $branchblock[loop] {
32
33         $dopipeline $depth 31 $fullrate
34         $merge $entry $loopback
35         $phi K := $zero<8> $on $entry nK $on $loopback
36         $phi SUM0 := ($bitcast ($uint<32>) 0) $on $entry nSUM0 $on $loopback
37         $phi SUM1 := ($bitcast ($uint<32>) 0) $on $entry nSUM1 $on $loopback
38         $endmerge
39         $volatile nK := (K + 2)
40         $volatile continue_flag := (K < (ORDER-2))
41         $volatile K1 := (K+1)
42
43         nSUM0 := (SUM0 + (A[I][K] * B[K][J]))
44         nSUM1 := (SUM0 + (A[I][K1] * B[K1][J]))
45         $while continue_flag
46     } (nSUM0 => R0 nSUM1 => R1)
47     $volatile result := (R0 + R1)
48 }
49
50 $module [mmul] $in () $out () $is
51 {
52     mmul_START := 1
53     $branchblock[loop1] {
54         u := mmul_START v := mmul_END // for logging
55         $merge $entry I_loopback
56         $phi I := $zero<8> $on $entry nI $on I_loopback
57         $endmerge
58         $volatile nI := (I + 1)
59

```

```

60     $dopipeline $depth 3
61         $merge $entry $loopback
62         $phi J := $zero<8> $on $entry nJ $on $loopback
63     $endmerge
64     $volatile nJ := (J + 1)
65     $volatile continue_flag := (J < (ORDER_by_2 - 1))
66
67     // dot-product!
68     C[I][J] := ($call dot_product (I J))
69
70     mmul_I := I
71     mmul_J := J
72
73     $while continue_flag
74
75         $if (I < (ORDER_by_2-1)) $then $place [I_loopback] $endif
76     }
77
78
79     $branchblock[loop2] {
80         u := mmul_START v := mmul_END // for logging
81         $merge $entry I_loopback
82         $phi I := $zero<8> $on $entry nI $on I_loopback
83     $endmerge
84     $volatile nI := (I + 1)
85
86     $dopipeline $depth 3
87         $merge $entry $loopback
88         $phi J := $zero<8> $on $entry nJ $on $loopback
89     $endmerge
90     $volatile nJ := (J + 1)
91     $volatile continue_flag := (J < (ORDER_by_2 - 1))
92     $volatile I_1 := (I + 8)
93     $volatile J_1 := (J + 8)
94
95     // dot-product!
96     C[I_1][J_1] := ($call dot_product (I_1 J_1))
97
98     mmul_I := I
99     mmul_J := J
100
101     $while continue_flag
102
103         $if (I < (ORDER_by_2-1)) $then $place [I_loopback] $endif
104     }
105
106     $branchblock[loop3] {
107         u := mmul_START v := mmul_END // for logging
108         $merge $entry I_loopback
109         $phi I := $zero<8> $on $entry nI $on I_loopback
110     $endmerge
111     $volatile nI := (I + 1)
112
113     $dopipeline $depth 3
114         $merge $entry $loopback
115         $phi J := $zero<8> $on $entry nJ $on $loopback
116     $endmerge
117     $volatile nJ := (J + 1)
118     $volatile continue_flag := (J < (ORDER_by_2 - 1))
119     // $volatile I_1 := (I + 8)
120     $volatile J_1 := (J + 8)
121

```

```

122     // dot-product!
123     C[I][J_1] := ($call dot_product (I J_1))
124
125     mmul_I := I
126     mmul_J := J
127
128     $while continue_flag
129
130     $if (I < (ORDER_by_2-1)) $then $place [I_loopback] $endif
131 }
132
133 $branchblock[loop4] {
134     u := mmul_START v := mmul_END // for logging
135     $merge $entry I_loopback
136     $phi I := $zero<8> $on $entry nI $on I_loopback
137     $endmerge
138     $volatile nI := (I + 1)
139
140     $dopipeline $depth 3
141     $merge $entry $loopback
142     $phi J := $zero<8> $on $entry nJ $on $loopback
143     $endmerge
144     $volatile nJ := (J + 1)
145     $volatile continue_flag := (J < (ORDER_by_2 - 1))
146     $volatile I_1 := (I + 8)
147     //$volatile J_1 := (J + 8)
148
149     // dot-product!
150     C[I_1][J] := ($call dot_product (I_1 J))
151
152     mmul_I := I
153     mmul_J := J
154
155     $while continue_flag
156
157     $if (I < (ORDER_by_2-1)) $then $place [I_loopback] $endif
158 }
159
160 mmul_END := 1
161 }

```

Four branch blocks are used, in order to find the resultant multiplication matrix. - The loop 1 is finding the top left matrix.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{18} \\ a_{21} & a_{22} & \cdots & a_{28} \\ \vdots & \vdots & \ddots & \vdots \\ a_{81} & a_{82} & \cdots & a_{88} \end{bmatrix}$$

```

root@f9740f6f81ab: /example x + v
C[5][1] = 192.
C[5][2] = 288.
C[5][3] = 384.
C[5][4] = 480.
C[5][5] = 576.
C[5][6] = 672.
C[5][7] = 768.
C[5][8] = 0.
C[5][9] = 0.
C[5][10] = 0.
C[5][11] = 0.
C[5][12] = 0.
C[5][13] = 0.
C[5][14] = 0.
C[5][15] = 0.
C[6][0] = 112.
C[6][1] = 224.
C[6][2] = 336.
C[6][3] = 448.
C[6][4] = 560.
C[6][5] = 672.
C[6][6] = 784.
C[6][7] = 896.
C[6][8] = 0.
C[6][9] = 0.
C[6][10] = 0.
C[6][11] = 0.
C[6][12] = 0.
C[6][13] = 0.
C[6][14] = 0.
C[6][15] = 0.
C[7][0] = 128.
C[7][1] = 256.
C[7][2] = 384.
C[7][3] = 512.
C[7][4] = 640.
C[7][5] = 768.
C[7][6] = 896.
C[7][7] = 1024.
C[7][8] = 0.
C[7][9] = 0.
C[7][10] = 0.
C[7][11] = 0.
C[7][12] = 0.
C[7][13] = 0.
C[7][14] = 0.
C[7][15] = 0.
C[8][0] = 0.
C[8][1] = 0.
C[8][2] = 0.

(126/210) ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17812:41:@134865ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 3e834aa
/ahir/release//vhdl/ahir.vhdl:17819:41:@134875ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 3e834aa
/ahir/release//vhdl/ahir.vhdl:17812:41:@134905ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17819:41:@134915ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17812:41:@134965ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 3ec34bu
/ahir/release//vhdl/ahir.vhdl:17819:41:@134975ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 3ec34bu
/ahir/release//vhdl/ahir.vhdl:17812:41:@135005ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17819:41:@135015ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17812:41:@135065ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 3f034be
/ahir/release//vhdl/ahir.vhdl:17819:41:@135075ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 3f034be
/ahir/release//vhdl/ahir.vhdl:17812:41:@135105ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17819:41:@135115ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17812:41:@135165ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 3f434c8
/ahir/release//vhdl/ahir.vhdl:17819:41:@135175ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 3f434c8
/ahir/release//vhdl/ahir.vhdl:17812:41:@135205ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17819:41:@135215ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17812:41:@135265ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 3f834d2
/ahir/release//vhdl/ahir.vhdl:17819:41:@135275ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 3f834d2
/ahir/release//vhdl/ahir.vhdl:17812:41:@135305ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17819:41:@135315ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17812:41:@135365ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 3fc34dc
/ahir/release//vhdl/ahir.vhdl:17819:41:@135375ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17812:41:@135405ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000
/ahir/release//vhdl/ahir.vhdl:17819:41:@135415ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000000

*9740f6f81ab* 18:33 17-Nov-23

```

- The loop 2 is finding the bottom right matrix.

$$\begin{bmatrix} a_{919} & a_{920} & \cdots & a_{916} \\ a_{1019} & a_{1020} & \cdots & a_{1016} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1619} & a_{1620} & \cdots & a_{1616} \end{bmatrix}$$

```

root@f9740f6f81ab: /example
C[12][15] = 3328.
C[13][0] = 0.
C[13][1] = 0.
C[13][2] = 0.
C[13][3] = 0.
C[13][4] = 0.
C[13][5] = 0.
C[13][6] = 0.
C[13][7] = 0.
C[13][8] = 2016.
C[13][9] = 2240.
C[13][10] = 2464.
C[13][11] = 2688.
C[13][12] = 2912.
C[13][13] = 3136.
C[13][14] = 3360.
C[13][15] = 3584.
C[14][0] = 0.
C[14][1] = 0.
C[14][2] = 0.
C[14][3] = 0.
C[14][4] = 0.
C[14][5] = 0.
C[14][6] = 0.
C[14][7] = 0.
C[14][8] = 2160.
C[14][9] = 2400.
C[14][10] = 2640.
C[14][11] = 2880.
C[14][12] = 3120.
C[14][13] = 3360.
C[14][14] = 3600.
C[14][15] = 3840.
C[15][0] = 0.
C[15][1] = 0.
C[15][2] = 0.
C[15][3] = 0.
C[15][4] = 0.
C[15][5] = 0.
C[15][6] = 0.
C[15][7] = 0.
C[15][8] = 2384.
C[15][9] = 2560.
C[15][10] = 2816.
C[15][11] = 3072.
C[15][12] = 3328.
C[15][13] = 3584.
C[15][14] = 3840.
C[15][15] = 4096.
root@f9740f6f81ab: /example/Question_2#
[0] 0:[tmux]

[0/210] ter=0 data= 700000a00
/ahir/release//vhdL/ahir.vhdl:17812:41:@198485ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7d00d804
/ahir/release//vhdL/ahir.vhdl:17819:41:@198495ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7d00d804
/ahir/release//vhdL/ahir.vhdl:17812:41:@198525ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000b00
/ahir/release//vhdL/ahir.vhdl:17819:41:@198535ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000c00
/ahir/release//vhdL/ahir.vhdl:17812:41:@198585ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7d80dd8e
/ahir/release//vhdL/ahir.vhdl:17819:41:@198595ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7d80dd8e
/ahir/release//vhdL/ahir.vhdl:17812:41:@198625ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000c00
/ahir/release//vhdL/ahir.vhdl:17819:41:@198635ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000c00
/ahir/release//vhdL/ahir.vhdl:17812:41:@198685ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7e80dd98
/ahir/release//vhdL/ahir.vhdl:17819:41:@198695ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7e80dd98
/ahir/release//vhdL/ahir.vhdl:17812:41:@198725ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000d00
/ahir/release//vhdL/ahir.vhdl:17819:41:@198735ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000d00
/ahir/release//vhdL/ahir.vhdl:17812:41:@198785ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7e80dda2
/ahir/release//vhdL/ahir.vhdl:17819:41:@198795ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7e80dda2
/ahir/release//vhdL/ahir.vhdl:17812:41:@198825ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000e00
/ahir/release//vhdL/ahir.vhdl:17819:41:@198835ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000e00
/ahir/release//vhdL/ahir.vhdl:17812:41:@198885ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7f00ddac
/ahir/release//vhdL/ahir.vhdl:17819:41:@198895ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7f00ddac
/ahir/release//vhdL/ahir.vhdl:17812:41:@198925ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000f00
/ahir/release//vhdL/ahir.vhdl:17819:41:@198935ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700000f00
/ahir/release//vhdL/ahir.vhdl:17812:41:@198985ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7f80ddb6
/ahir/release//vhdL/ahir.vhdl:17819:41:@198995ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7f80ddb6
/ahir/release//vhdL/ahir.vhdl:17812:41:@199025ns:(assertion note): WPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700001000
/ahir/release//vhdL/ahir.vhdl:17819:41:@199035ns:(assertion note): RPIPE loadC_out_buffer-bufPipe reques
ter=0 data= 700001000

```

- The loop 3 is finding bottom left matrix.

$$\begin{bmatrix} a_{911} & a_{912} & \cdots & a_{98} \\ a_{1011} & a_{1012} & \cdots & a_{108} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1611} & a_{1612} & \cdots & a_{168} \end{bmatrix}$$

```

root@f9740f6f81ab: /example
C[6][4] = 560.
C[6][5] = 672.
C[6][6] = 784.
C[6][7] = 896.
C[6][8] = 1008.
C[6][9] = 1120.
C[6][10] = 1232.
C[6][11] = 1344.
C[6][12] = 1456.
C[6][13] = 1568.
C[6][14] = 1680.
C[6][15] = 1792.
C[7][0] = 128.
C[7][1] = 256.
C[7][2] = 384.
C[7][3] = 512.
C[7][4] = 640.
C[7][5] = 768.
C[7][6] = 896.
C[7][7] = 1024.
C[7][8] = 1152.
C[7][9] = 1280.
C[7][10] = 1408.
C[7][11] = 1536.
C[7][12] = 1664.
C[7][13] = 1792.
C[7][14] = 1920.
C[7][15] = 2048.
C[8][0] = 0.
C[8][1] = 0.
C[8][2] = 0.
C[8][3] = 0.
C[8][4] = 0.
C[8][5] = 0.
C[8][6] = 0.
C[8][7] = 0.
C[8][8] = 1296.
C[8][9] = 1440.
C[8][10] = 1584.
C[8][11] = 1728.
C[8][12] = 1872.
C[8][13] = 2016.
C[8][14] = 2160.
C[8][15] = 2304.
C[9][0] = 0.
C[9][1] = 0.
C[9][2] = 0.
C[9][3] = 0.
C[9][4] = 0.
C[9][5] = 0.

[1] 3 (tmux)

107/210 ter=0 data= 700000a00
/ahir/release//vhd/ahir.vhdl:17812:41:@262085ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7d0665c
/ahir/release//vhd/ahir.vhdl:17819:41:@262095ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7d0665c
/ahir/release//vhd/ahir.vhdl:17812:41:@262125ns:(assertion note): WPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700000b00
/ahir/release//vhd/ahir.vhdl:17819:41:@262135ns:(assertion note): RPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700000b00
/ahir/release//vhd/ahir.vhdl:17812:41:@262185ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7d06666
/ahir/release//vhd/ahir.vhdl:17819:41:@262195ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7d06666
/ahir/release//vhd/ahir.vhdl:17812:41:@262225ns:(assertion note): WPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700000c00
/ahir/release//vhd/ahir.vhdl:17819:41:@262235ns:(assertion note): RPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700000c00
/ahir/release//vhd/ahir.vhdl:17812:41:@262285ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7e06670
/ahir/release//vhd/ahir.vhdl:17819:41:@262295ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7e06670
/ahir/release//vhd/ahir.vhdl:17812:41:@262325ns:(assertion note): WPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700000d00
/ahir/release//vhd/ahir.vhdl:17819:41:@262335ns:(assertion note): RPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700000d00
/ahir/release//vhd/ahir.vhdl:17812:41:@262385ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7e0667a
/ahir/release//vhd/ahir.vhdl:17819:41:@262395ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7e0667a
/ahir/release//vhd/ahir.vhdl:17812:41:@262425ns:(assertion note): WPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700000e00
/ahir/release//vhd/ahir.vhdl:17819:41:@262435ns:(assertion note): RPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700000e00
/ahir/release//vhd/ahir.vhdl:17812:41:@262485ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7f06684
/ahir/release//vhd/ahir.vhdl:17819:41:@262495ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7f06684
/ahir/release//vhd/ahir.vhdl:17812:41:@262525ns:(assertion note): WPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700000f00
/ahir/release//vhd/ahir.vhdl:17819:41:@262535ns:(assertion note): RPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700000f00
/ahir/release//vhd/ahir.vhdl:17812:41:@262585ns:(assertion note): WPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7f0668e
/ahir/release//vhd/ahir.vhdl:17819:41:@262595ns:(assertion note): RPIPE LoadGroup0-RxGen-rb-0-bufPipe r
equester=0 data= 7f0668e
/ahir/release//vhd/ahir.vhdl:17812:41:@262625ns:(assertion note): WPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700001000
/ahir/release//vhd/ahir.vhdl:17819:41:@262635ns:(assertion note): RPIPE LoadC_out_buffer-bufPipe reques
ter=0 data= 700001000

*f9700f6f81ab 18:48 17-Nov-23

```

- The loop 4 is finding top right matrix.

$$\begin{bmatrix} a_{19} & a_{110} & \cdots & a_{116} \\ a_{29} & a_{210} & \cdots & a_{216} \\ \vdots & \vdots & \ddots & \vdots \\ a_{89} & a_{810} & \cdots & a_{816} \end{bmatrix}$$

```

root@f9740f6f81ab: /example  X + v
C[12][15] = 3328.
C[13][0] = 224.
C[13][1] = 448.
C[13][2] = 672.
C[13][3] = 896.
C[13][4] = 1120.
C[13][5] = 1344.
C[13][6] = 1568.
C[13][7] = 1792.
C[13][8] = 2016.
C[13][9] = 2240.
C[13][10] = 2464.
C[13][11] = 2688.
C[13][12] = 2912.
C[13][13] = 3136.
C[13][14] = 3360.
C[13][15] = 3584.
C[14][0] = 240.
C[14][1] = 480.
C[14][2] = 720.
C[14][3] = 960.
C[14][4] = 1200.
C[14][5] = 1440.
C[14][6] = 1680.
C[14][7] = 1920.
C[14][8] = 2160.
C[14][9] = 2400.
C[14][10] = 2640.
C[14][11] = 2880.
C[14][12] = 3120.
C[14][13] = 3360.
C[14][14] = 3600.
C[14][15] = 3840.
C[15][0] = 256.
C[15][1] = 512.
C[15][2] = 768.
C[15][3] = 1024.
C[15][4] = 1280.
C[15][5] = 1536.
C[15][6] = 1792.
C[15][7] = 2048.
C[15][8] = 2304.
C[15][9] = 2560.
C[15][10] = 2816.
C[15][11] = 3072.
C[15][12] = 3328.
C[15][13] = 3584.
C[15][14] = 3840.
C[15][15] = 4096.
root@f9740f6f81ab: /example/Question_2#

```



## Results:

The time difference in finishing the matrix multiplication is can be seen. The right side waveforms are of the sample code while the left side waveforms is the modified matrix multiplication algorithm. The sample code mmul\_end is being reported at 462675ns and the modified code mmul\_end is reported at 300625ns.

