Homework 4 – 2D Weight-Stationary Systolic Architecture for Matrix Multiplication

Taghizadeh,
810198373

I. Introduction

In this homework, we are about to make a *Systolic Array* to calculate matrix multiplication. To make a *Systolic Array* we use an array of *PE* (Processing Element) which are connected to each other in rows and columns. Each *PE* can do a MAC measurement which is needed to calculate matrix multiplication.

II. Structural RTL Modelling Phase

A. *RTL Model of PE in SystemC*

`

PE's Data Path and Controller Schematic

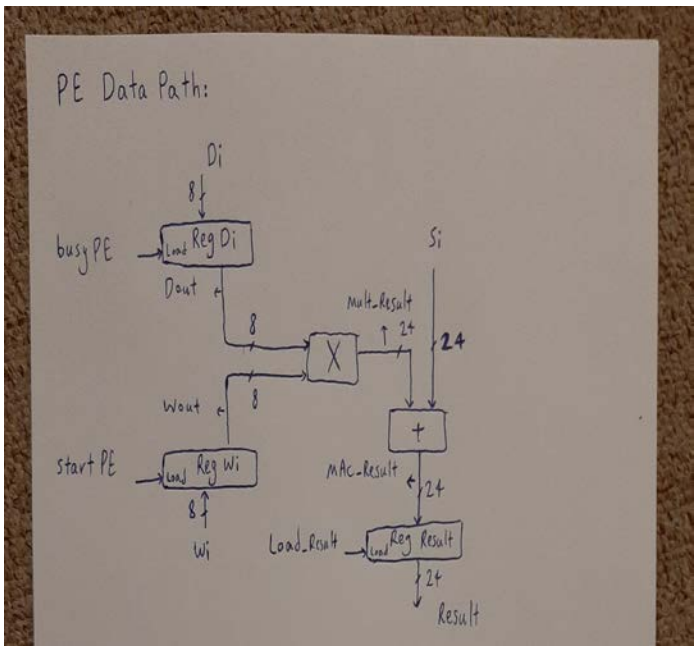

Fig. 1 *PE*'s Data Path Schematic

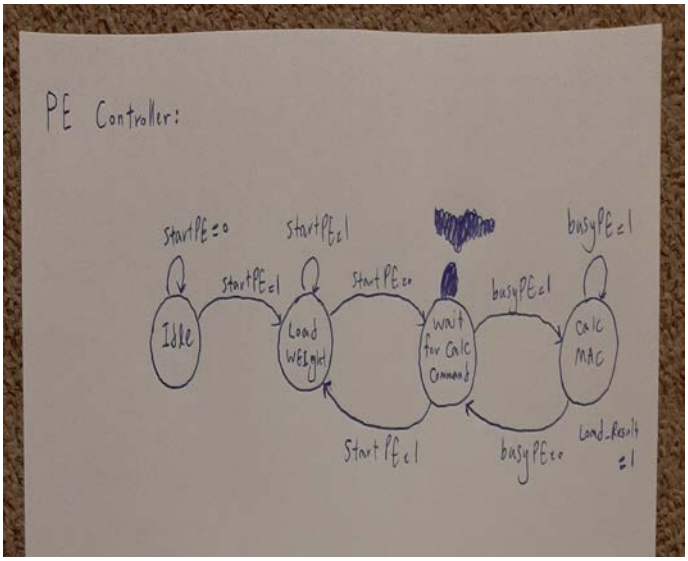

Fig. 2 *PE*'s Controller Schematic

PE's Data Path and Controller in SystemC

Data Path:

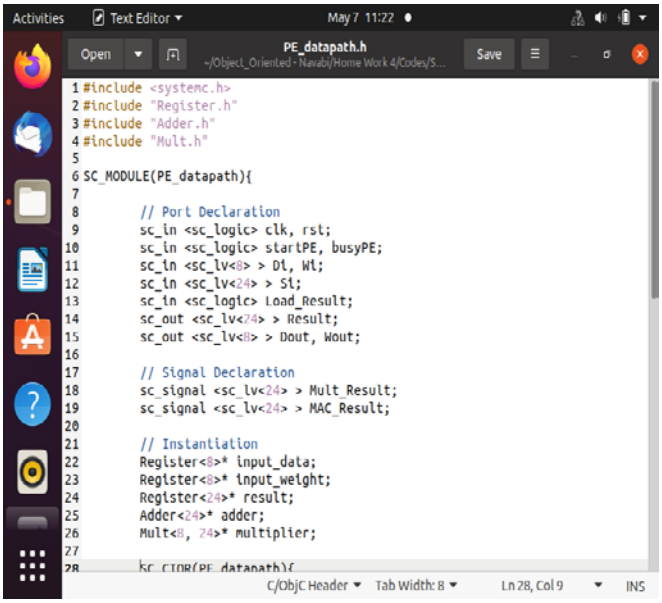

Fig. 3 *PE*'s Data Path *SystemC*

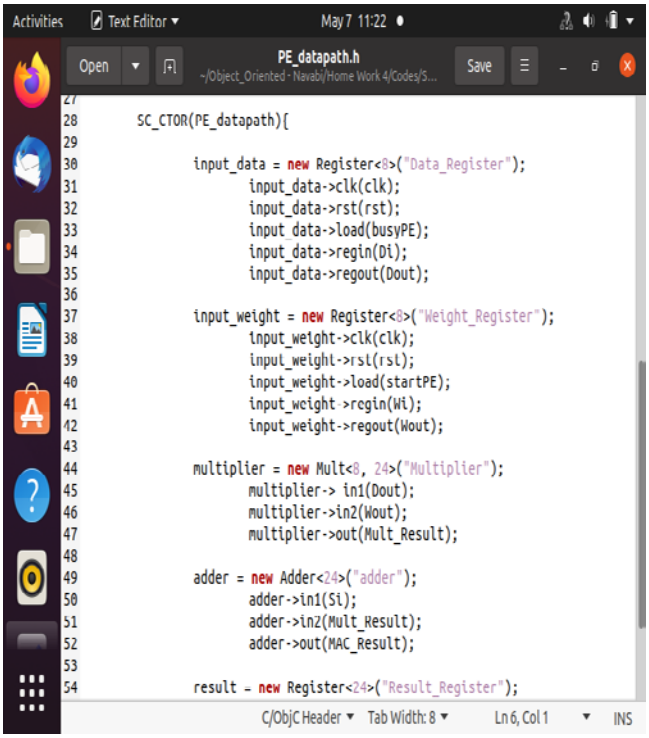

Fig. 4 *PE*'s Data Path *SystemC*

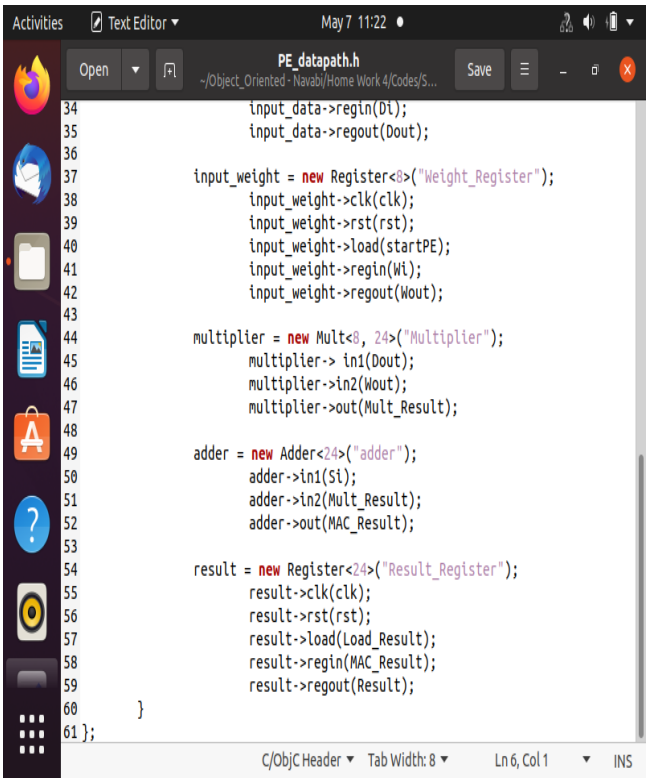

Fig. 5 *PE*'s Data Path *SystemC*

Controller:

Fig. 6 *PE*'s Controller *SystemC*

Fig. 7 *PE*'s Controller Combinational *SystemC*

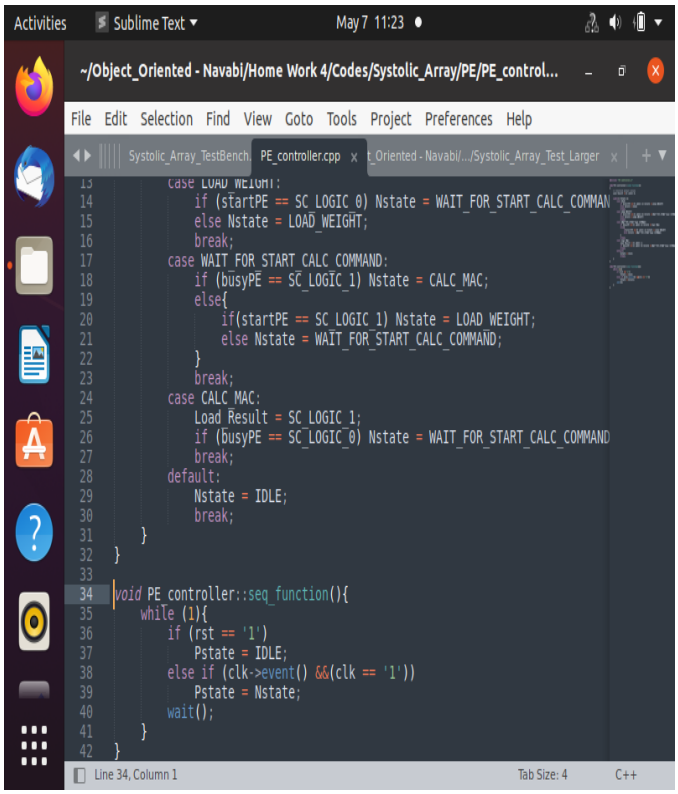

Fig. 8 *PE*'s Controller Sequential *SystemC*

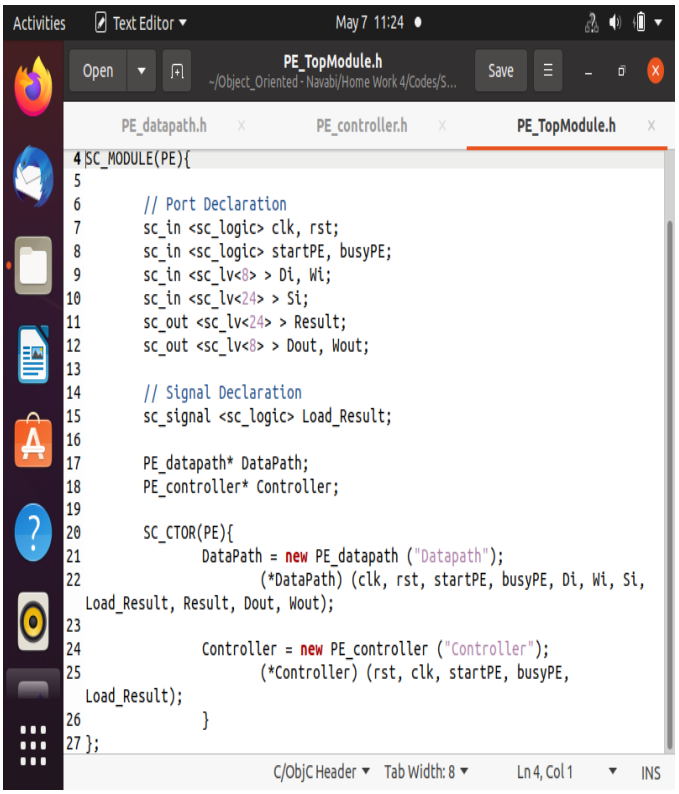

Fig. 10 *PE*'s Top Module *SystemC*

Top Module:

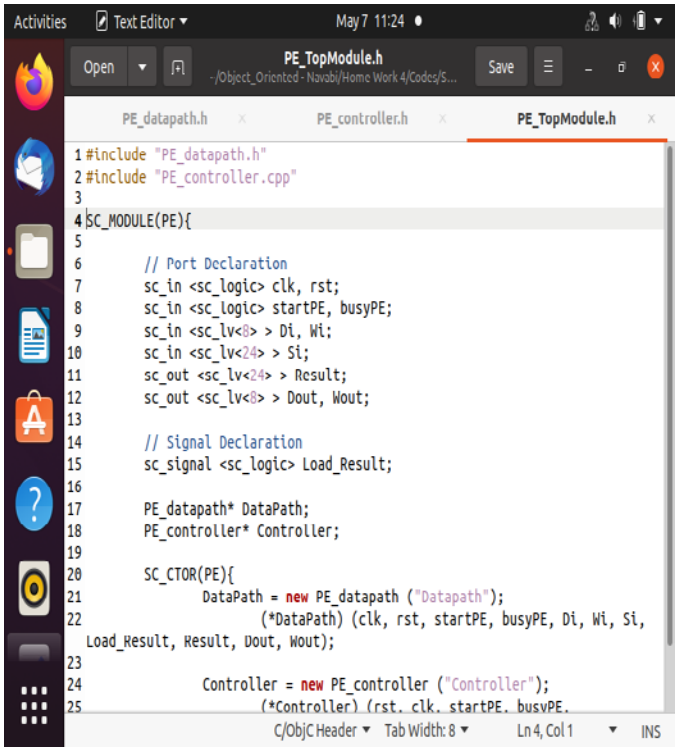

Fig. 9 *PE*'s Top Module *SystemC*

PE's Test Bench

To test the designed *PE*, first the input and output signals are declared in *PE_TestBench.h*, then an instance of *PE* is made. To reset, clock generation and giving inputs, *resetting*, *clocking*, *inputting* and *displaying* functions are declared as a *SC_THREAD* and *SC_METHOD*. To giving the inputs, first input weight is initialized and *startPE* signal is issued. Then we wait about a few number of clocks (each clock period is 100 ns) to load the weight of *PE*. Now weight is loaded, so *Di* and *Si* input signals are initialized and *busyPE* signal is issued to have (*Di***Wi* + *Si*) calculated in the next clock. This sequence is repeated several times to test different scenarios.

Test Bench:

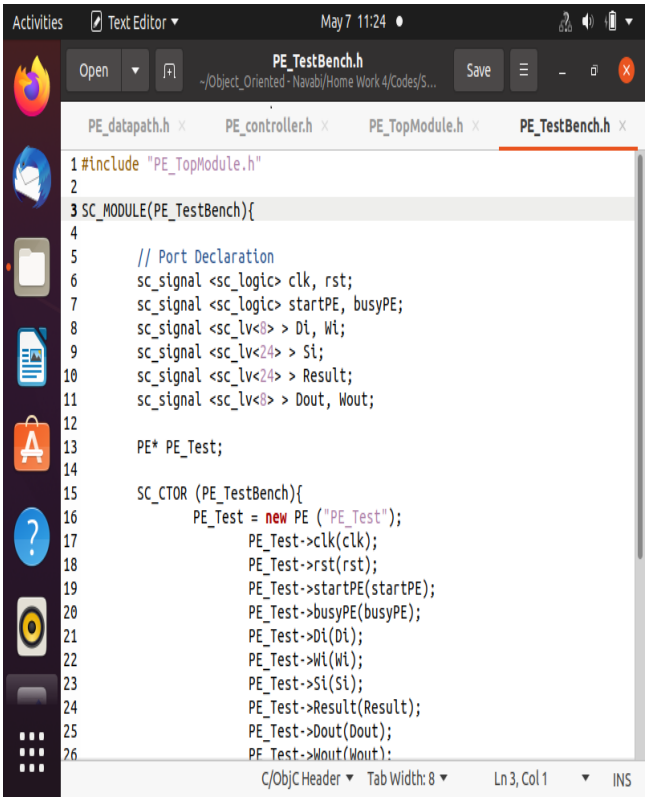

Fig. 11 *PE*'s Test Bench *SystemC*

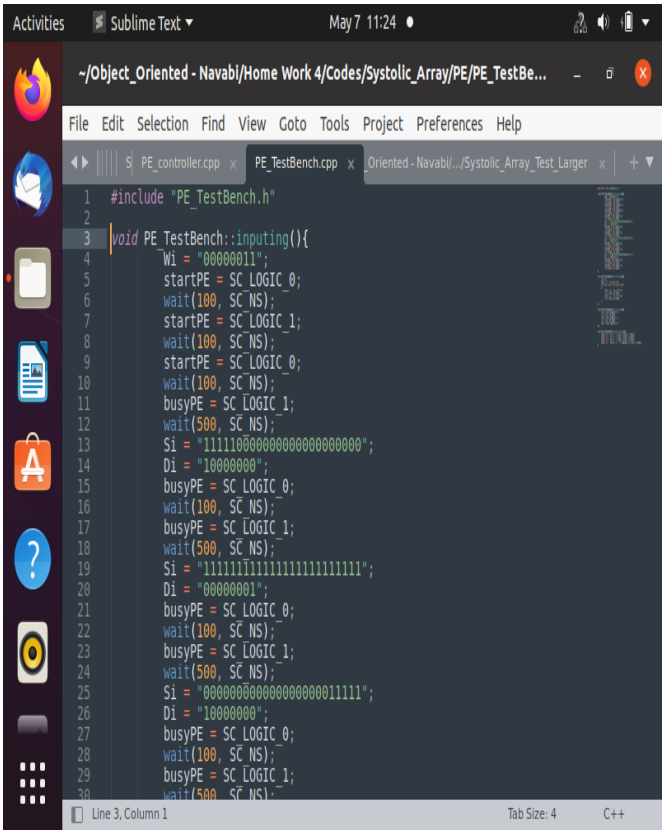

Fig. 13 *PE*'s Test Bench *inputting SystemC*

Fig. 12 *PE*'s Test Bench *SystemC*

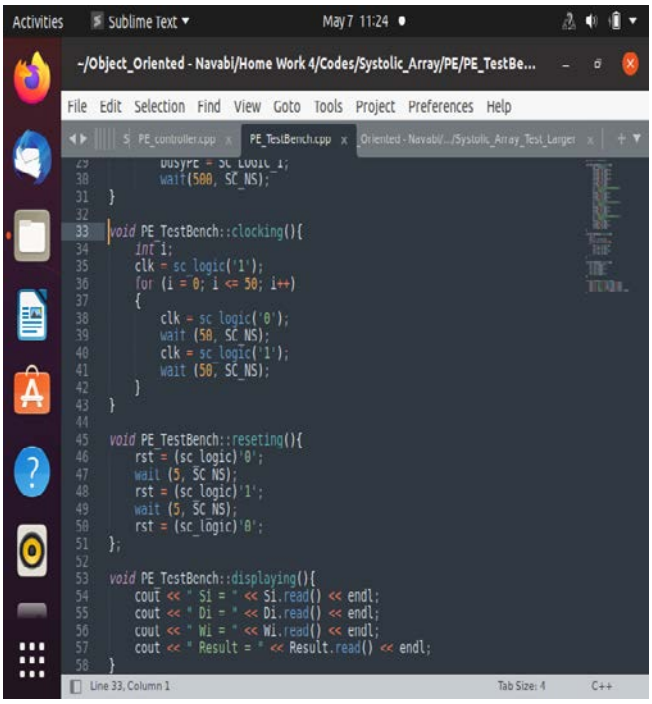

Fig. 14 *PE*'s Test Bench clocking, resetting, *displaying SystemC*

Main:

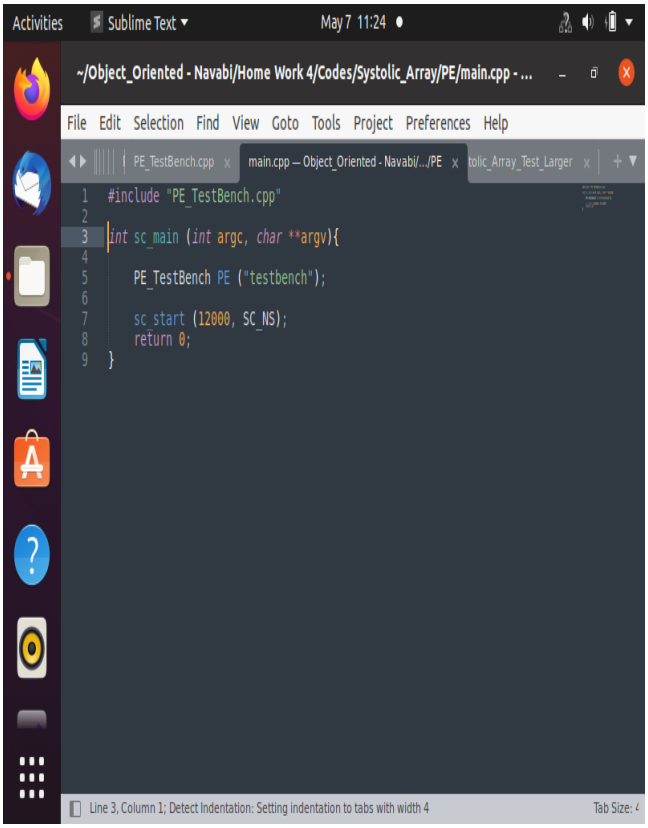

Fig. 15 *PE*'s Main *SystemC*

PE's Test Bench Results:

```
Si = 11111000000000000000000
Di = 10000000
Wi = 00000011
Result = 11111000000000110000000
```

```
Si = 11111111111111111111111
Di = 00000001
Wi = 00000011
Result = 00000000000000000000010
```

```
Si = 00000000000000000011111
Di = 10000000
Wi = 00000011
Result = 00000000000000110011111
```

Fig. 16 *PE*'s Test Bench Results *SystemC*

B. *RTL Model of Systolic Array in SystemC*

Systolic Array's Data Path and Controller Schematic

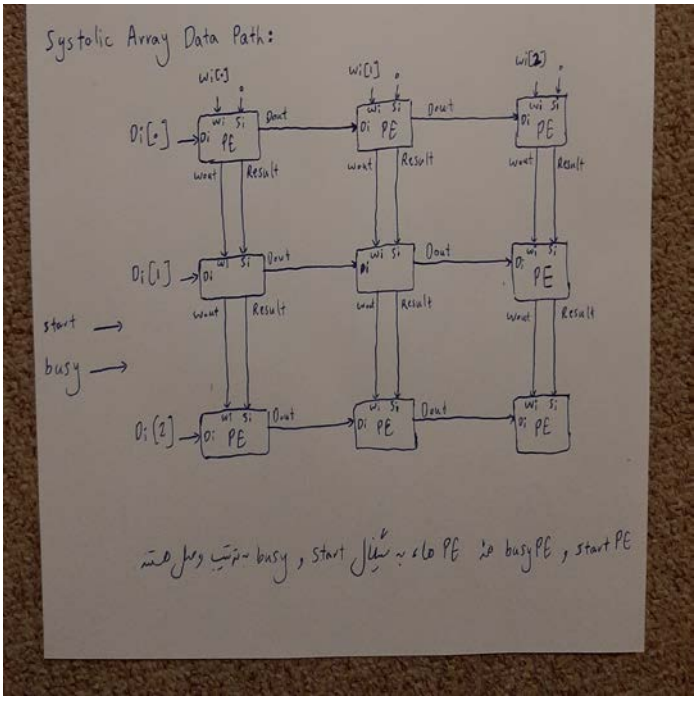

Fig. 17 *Systolic Array*'s Data Path Schematic

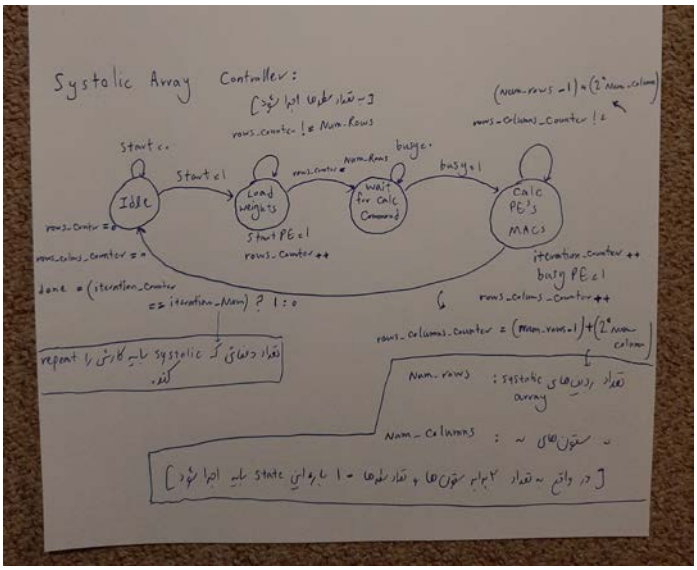

Fig. 18 *Systolic Array*'s Controller Schematic

Systolic Array's Data Path and Controller in SystemC

To design the Data Path of *Systolic Array*, simply an array of *PE*s are made by a nested for loop and then connected to each other as shown in the image of *Systolic Array's Data Path*. To make the *Systolic Array* configurable we use template.

Data Path:

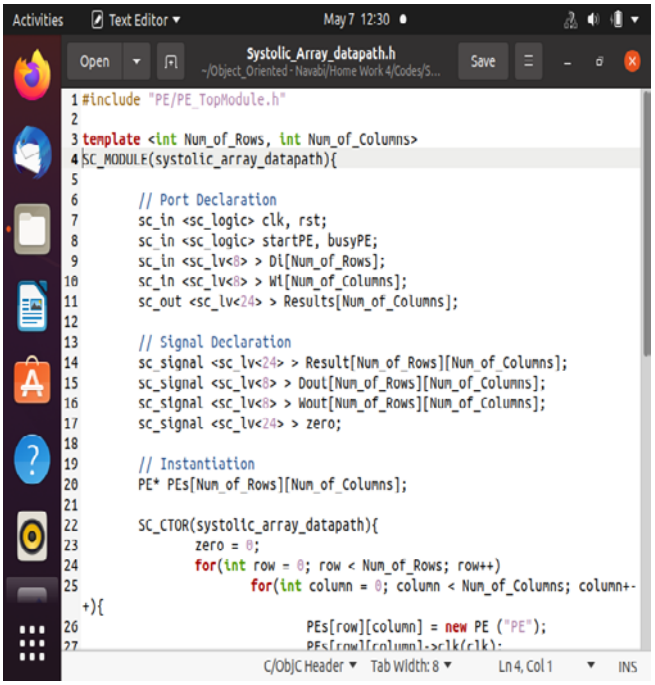

Fig. 19 *Systolic Array*'s Data Path *SystemC*

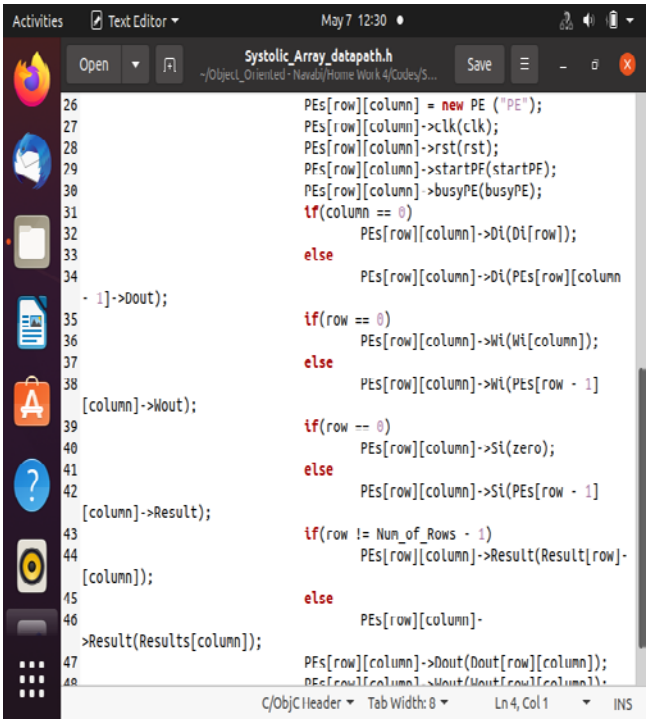

Fig. 20 *Systolic Array*'s Data Path *SystemC*

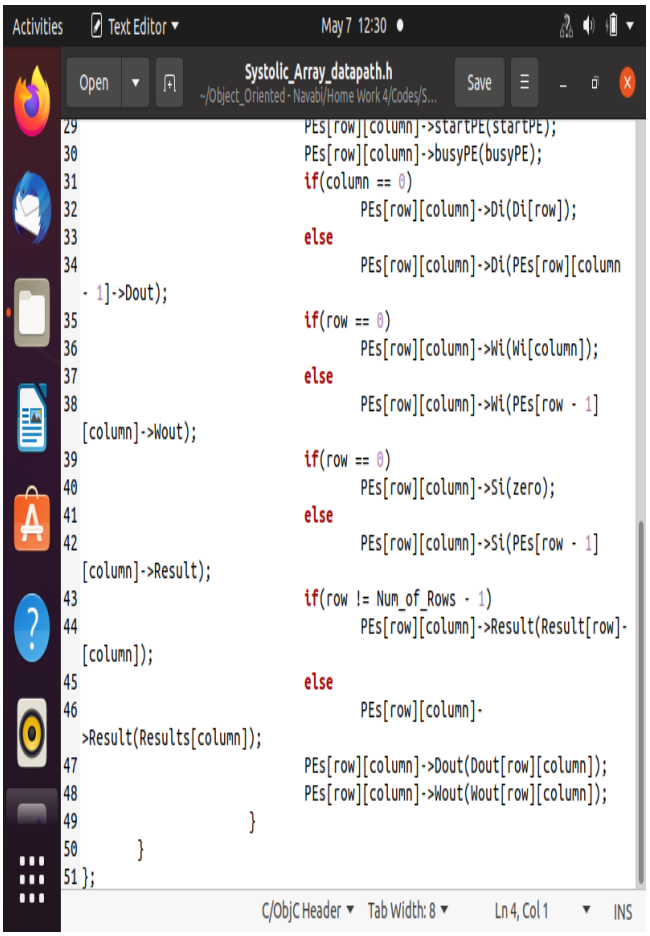

Fig. 21 *Systolic Array*'s Data Path *SystemC*

Controller:

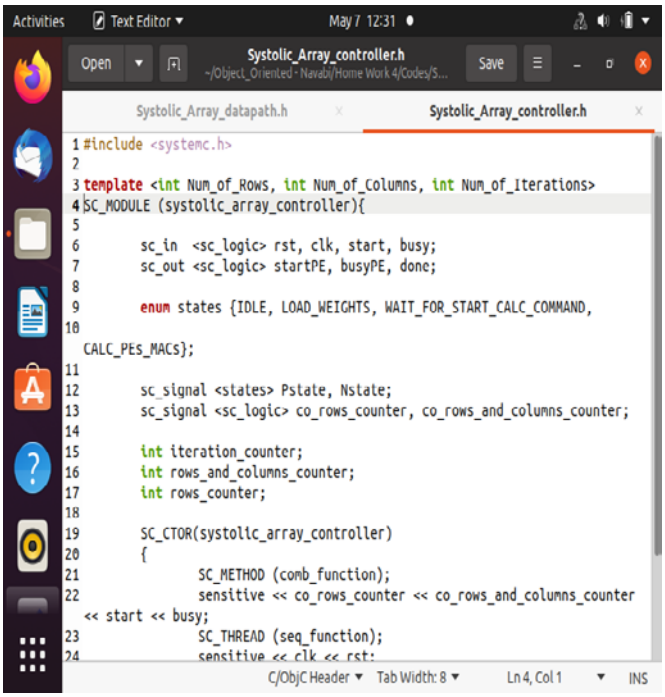

Fig. 22 *Systolic Array*'s Controller *SystemC*

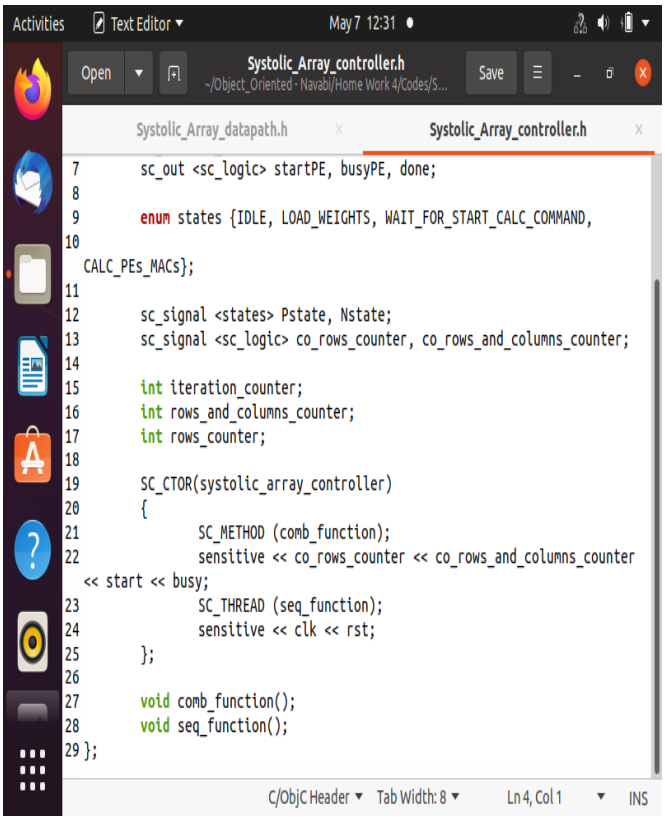

Fig. 23 *Systolic Array*'s Controller *SystemC*

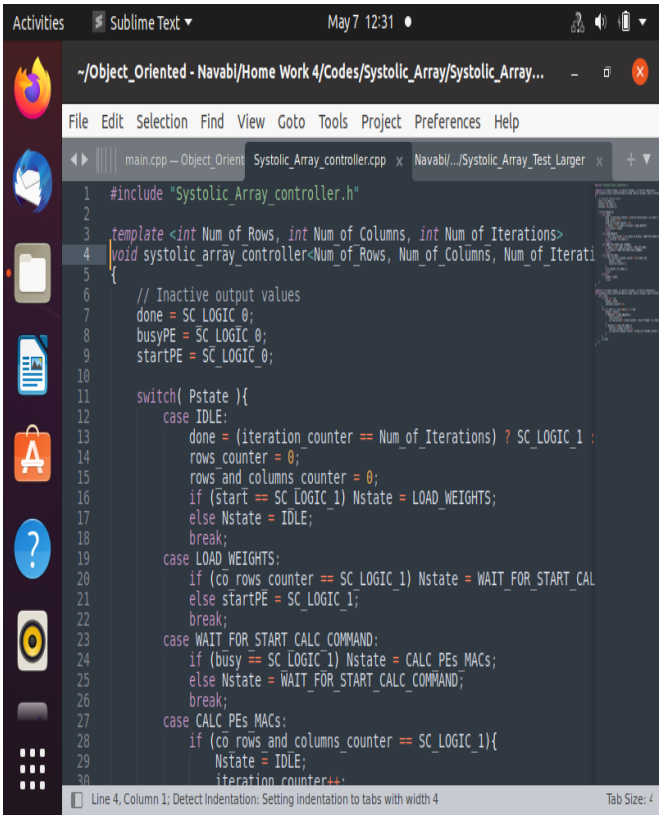

Fig. 24 *Systolic Array*'s Controller Combinational *SystemC*

Fig. 25 *Systolic Array*'s Controller Sequential *SystemC*

Top Module:

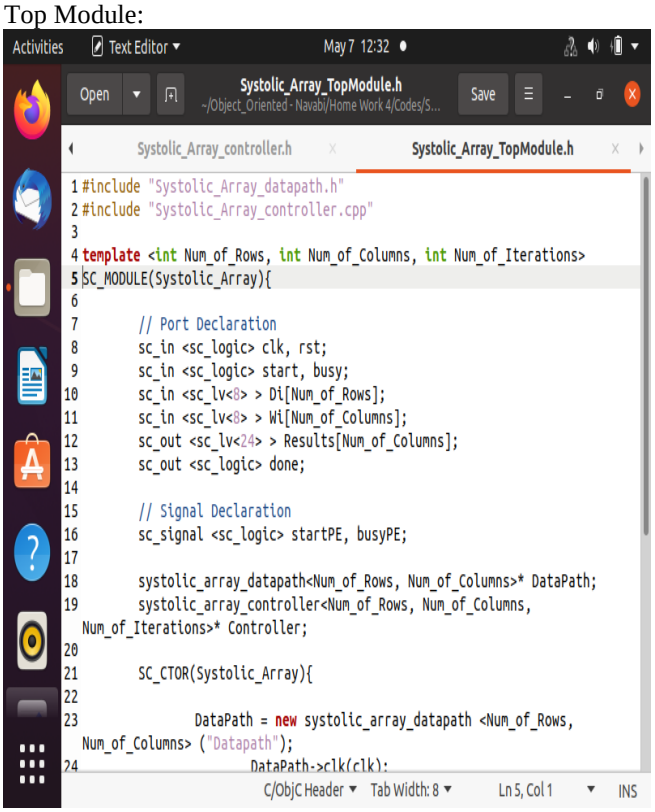

Fig. 26 *Systolic Array*'s Top Module *SystemC*

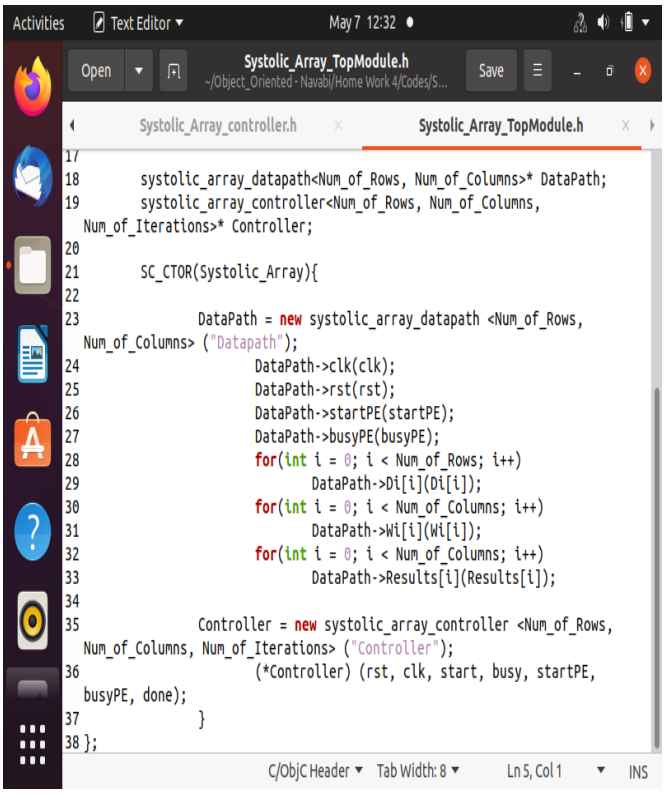

Fig. 27 *Systolic Array*'s Top Module *SystemC*

Systolic Array's Test Bench

To test designed Systolic Array, first input and output signals are declared in *Systolic_Array_TestBench.h*, then an instance of *Systolic Array* is made. To reset, clock generation and giving inputs, *resetting*, *clocking*, *inputting* and *displaying* functions are declared as a *SC_THREAD* and *SC_METHOD*. To giving the inputs, first *start* is issued and input weights are initialized clock by clock. Then we wait about a few number of clocks (each clock period is 100 ns) to load the weights of *Systolic Array*. Now weights are loaded, so *busy* signal is issued and input matrix elements(*Di*) signals are initialized clock by clock to put the input matrix into the *Systolic Array* to have multiplication calculated in the next clocks.

The input test matrix is:

1	1	1
1	1	0
1	0	0

Test Bench:

Fig. 28 *Systolic Array*'s Test Bench *SystemC*

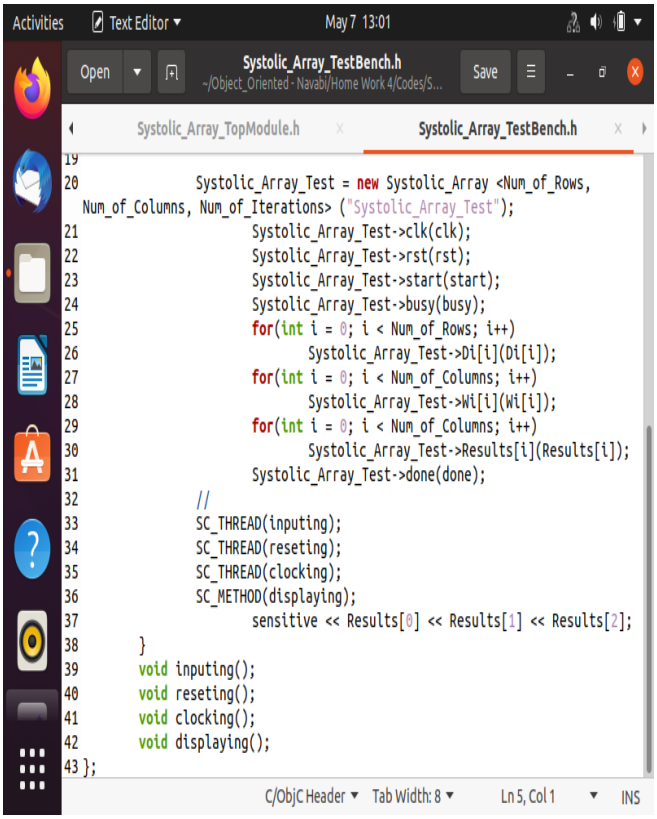

Fig. 29 *Systolic Array*'s Test Bench *SystemC*

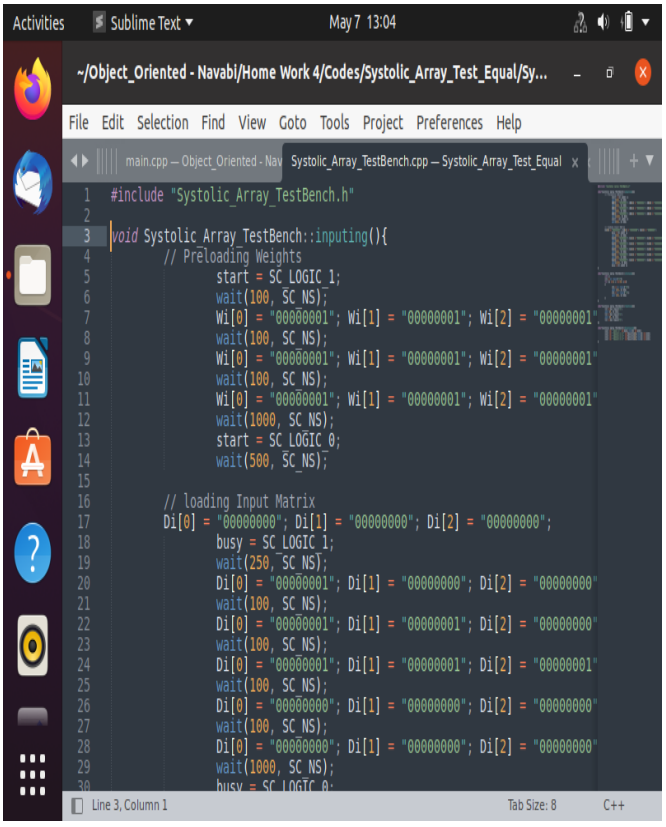

Fig. 30 *Systolic Array*'s Test Bench *inputting SystemC*

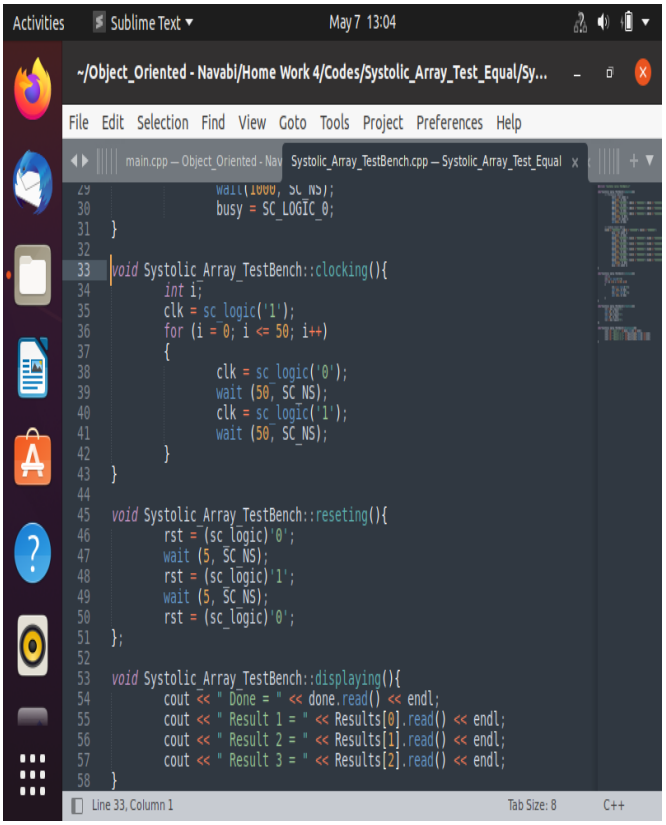

Fig. 31 *Systolic Array*'s Test Bench clocking, resetting, *displaying SystemC*

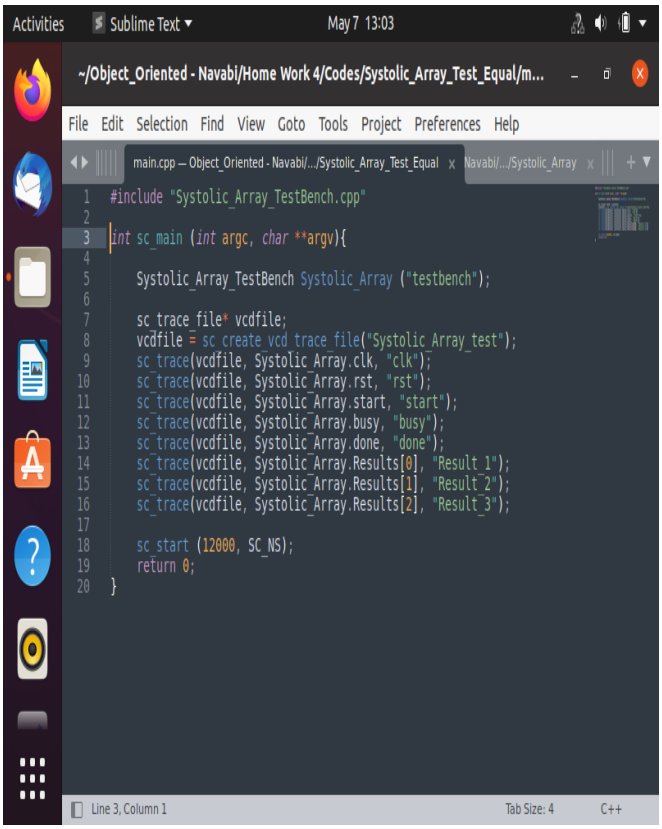

Fig. 32 *Systolic Array*'s Main *SystemC*

If input matrix is smaller than *Systolic Array*, then we must weight first column “0” to not using some *PE’s*.

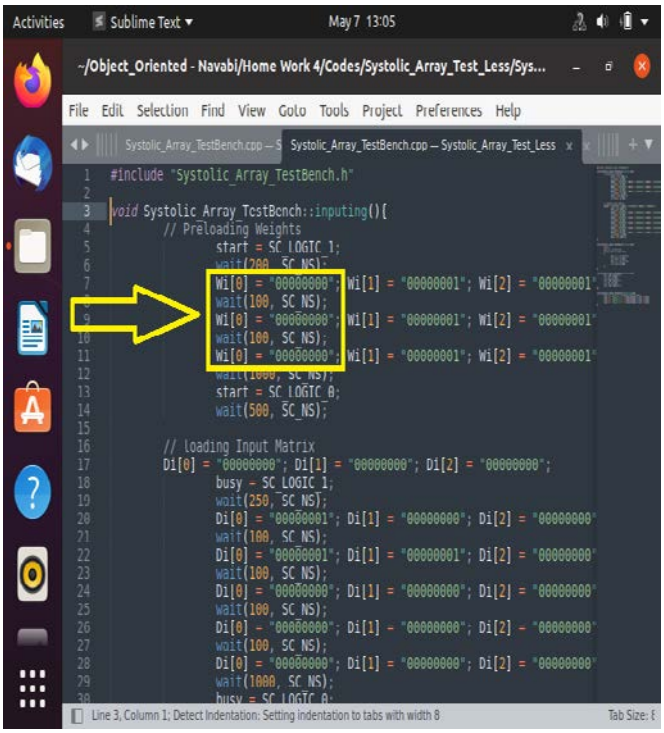

Fig. 33 *Systolic Array*’s Test Bench *inputting SystemC* (Less)

If input matrix is larger than *Systolic Array*, then we must repeat the scenario several times to complete the multiplication. this is done by giving *Iternation_Num* more than “1” to repeat the whole scenario more than once. In this case we must give inputs more than once, so preloading and loading input matrix is done more than once.

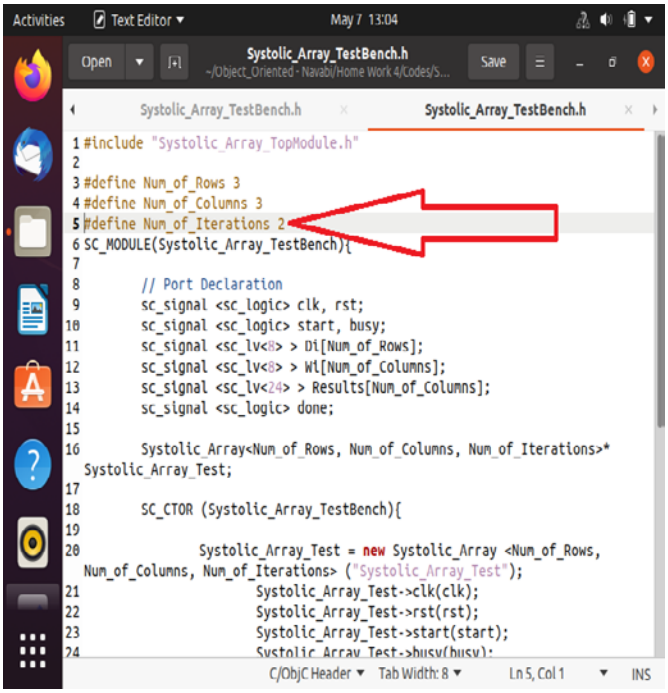

Fig. 34 *Systolic Array*’s Test Bench *SystemC* (Larger)

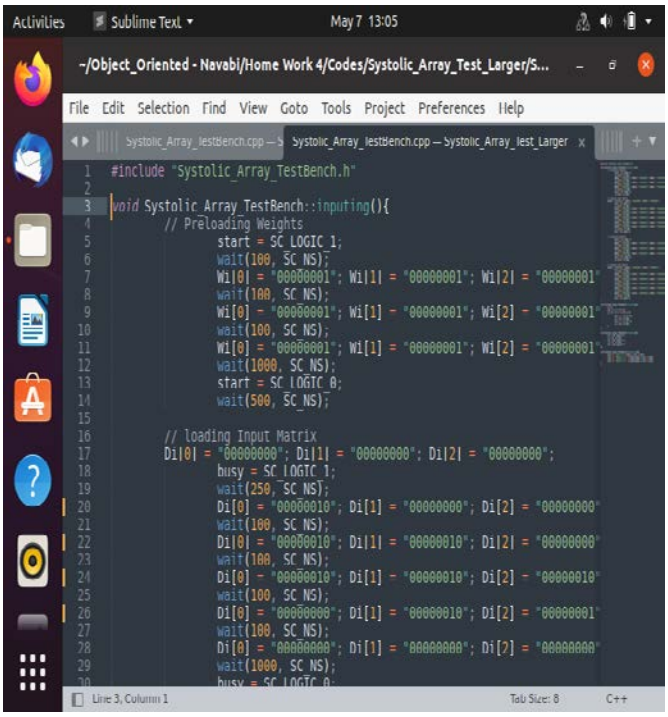

Fig. 35 *Systolic Array*’s Test Bench *inputting SystemC* (Larger)

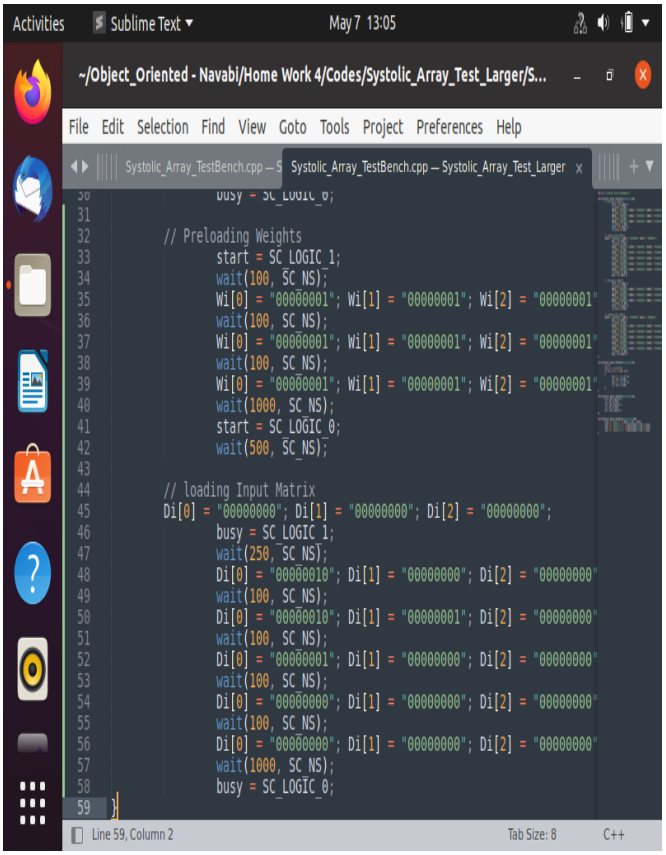

Fig. 36 *Systolic Array*'s Test Bench *inputting SystemC* (Larger)

Input Matrix is equal to *Systolic Array*:

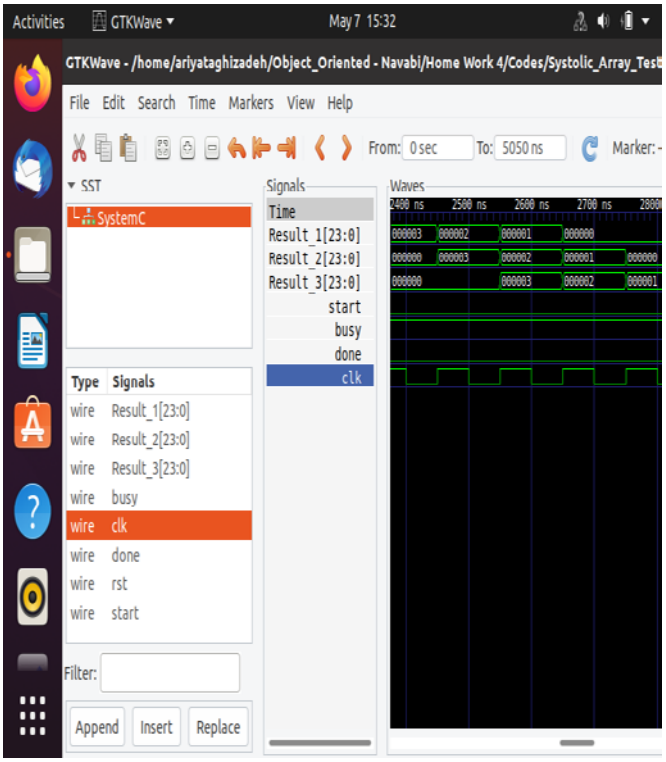

Fig. 37 *Systolic Array*'s Test Bench *Result SystemC* (Equal)

Systolic Array's Test Bench Results:

The input weigh matrix is:

1	1	1
1	1	1
1	1	1

The input data matrix is:

1	1	1
1	1	0
1	0	0

So the output matrix must be:

3	2	1
3	2	1
3	2	1

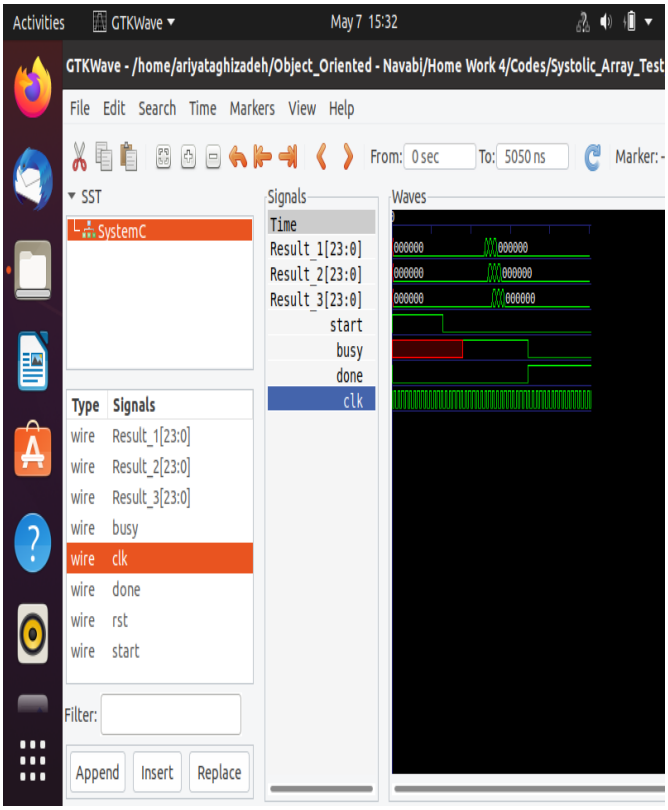

Fig. 38 *Systolic Array*'s Test Bench *Result SystemC* (Equal)

Input Matrix is smaller than *Systolic Array*:

The input weigh matrix is:

0	1	1
0	1	1
0	1	1

The input data matrix is:

1	1
1	0

So the output matrix must be:

2	1
2	1

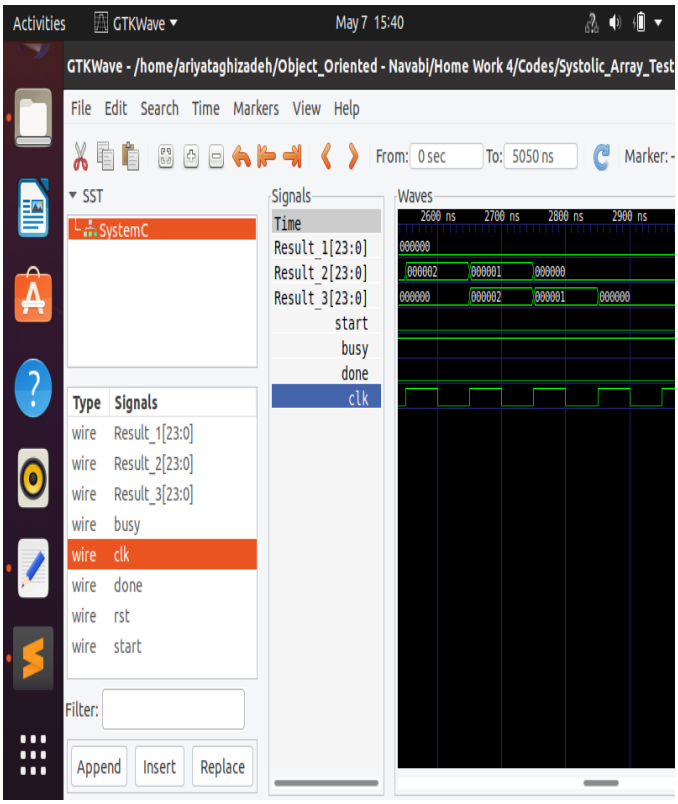

Fig. 39 *Systolic Array*'s Test Bench *Result SystemC* (Smaller)

Input Matrix is larger than *Systolic Array*:

The input weigh matrix is:

1	1	1
1	1	1
1	1	1

The input data matrix is:

2	2	2	2	2	1
2	2	2	1	0	0
2	1	0	0	0	0

So the output matrix must be:

6	5	4	3	2	1
6	5	4	3	2	1
6	5	4	3	2	1

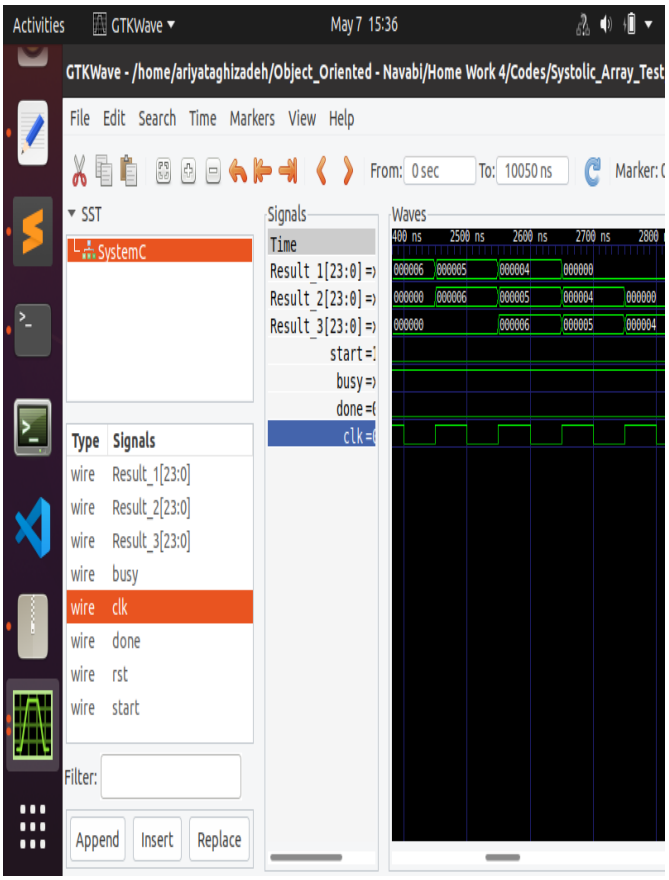

Fig. 40 *Systolic Array*'s Test Bench *Result SystemC* (Larger)

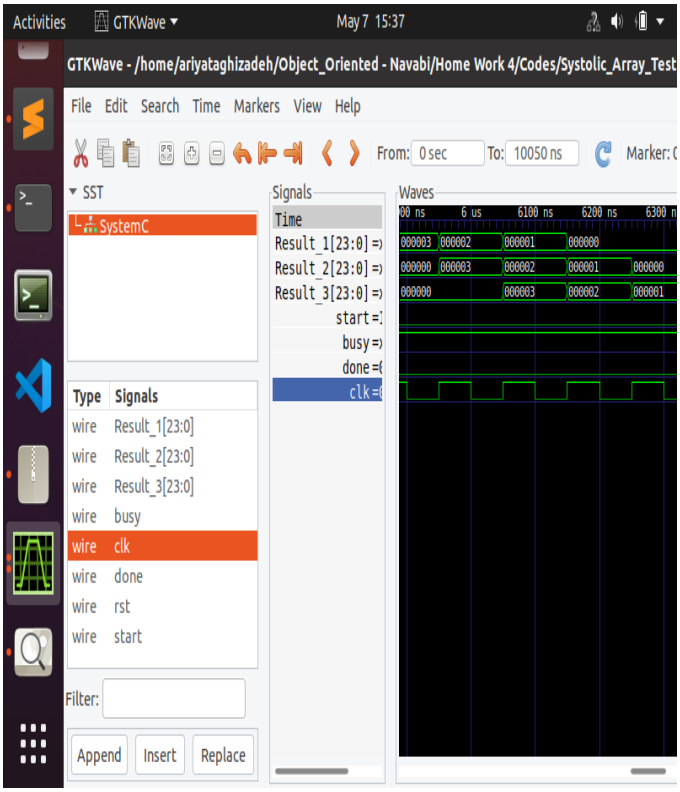

Fig. 41 *Systolic Array*'s Test Bench *Result SystemC* (Larger)

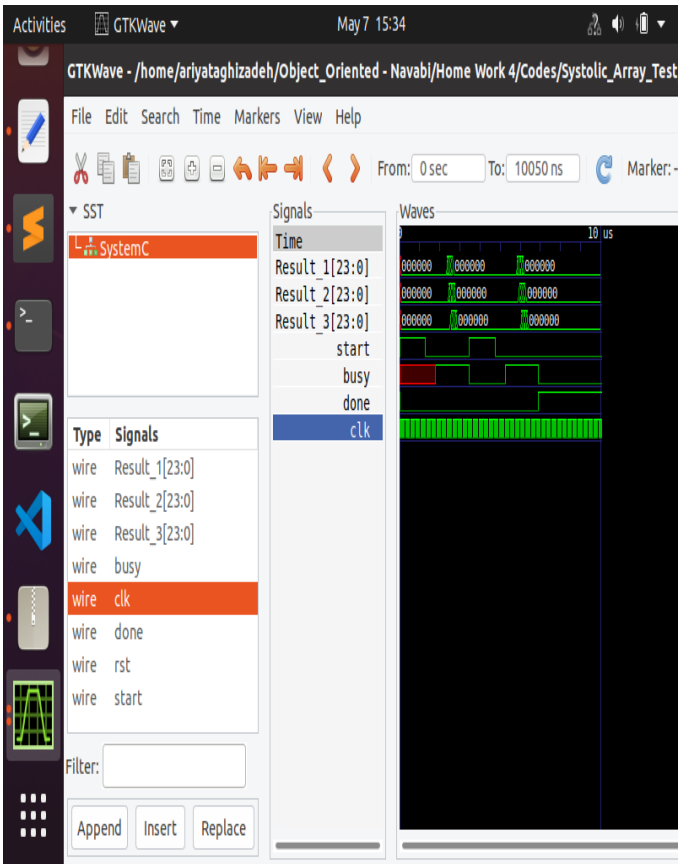

Fig. 42 *Systolic Array*'s Test Bench *Result SystemC* (Larger)

III. RTL Bus Functional Modelling Phase

To make the designed *Systolic Array* Bus functional Model, instead of using Data Path and Controller for *PE*, we can simply write its behavioural algorithm like the below image:

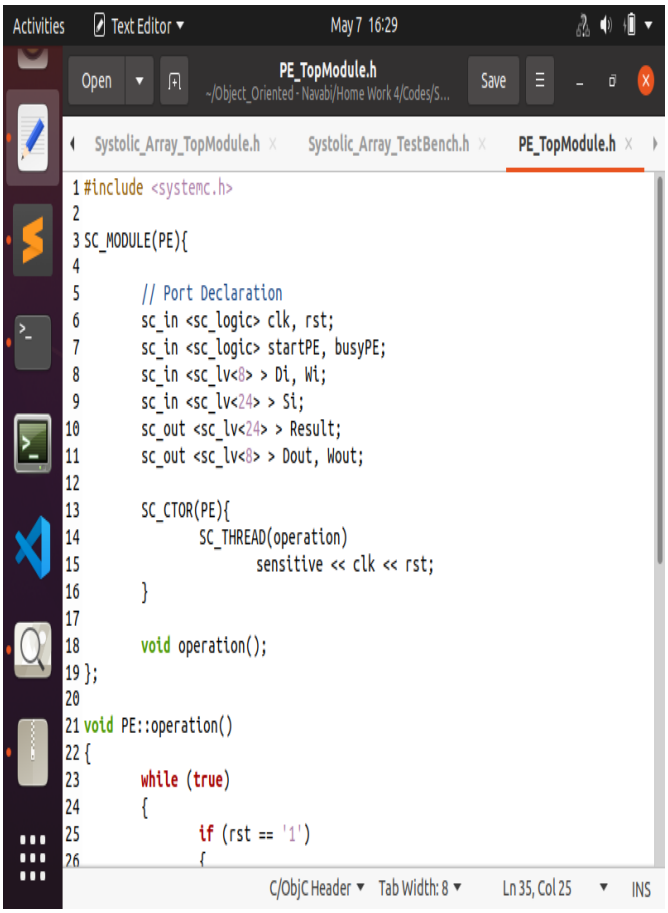

Fig. 43 *PE*'s Bus Functional Model *SystemC*

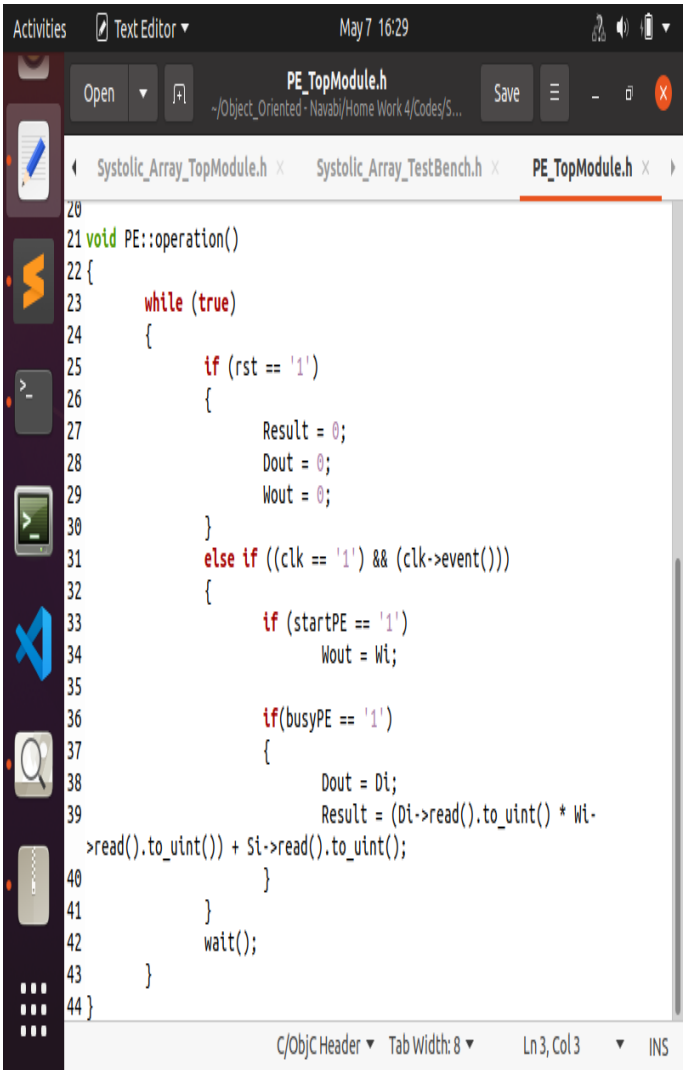

Fig. 44 *PE*'s Bus Functional Model *SystemC*

Systolic Array's BFM Test Bench Results:

Tests are the same as the previous part.

The input weigh matrix is:

1	1	1
1	1	1
1	1	1

The input data matrix is:

1	1	1
1	1	0
1	0	0

So the output matrix must be:

3	2	1
3	2	1
3	2	1

Input Matrix is equal to *Systolic Array*:

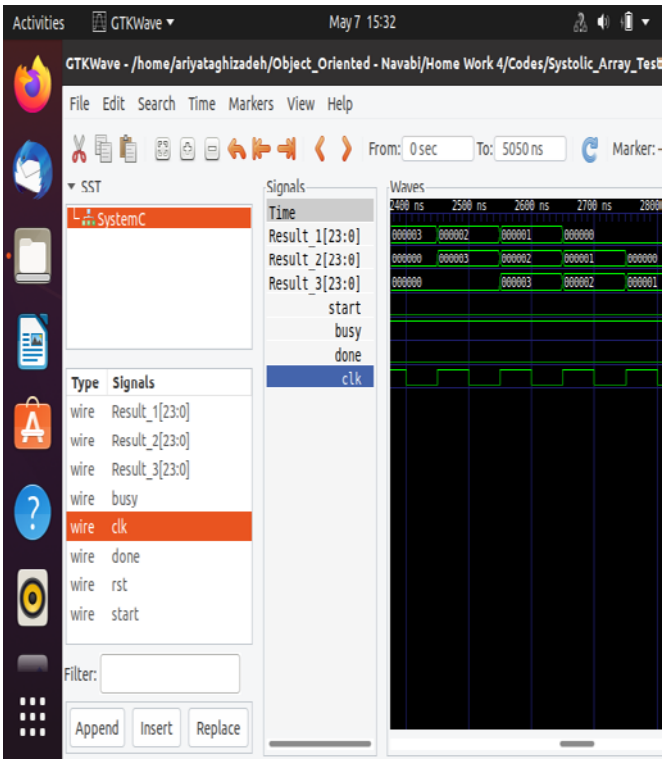

Fig. 45 *Systolic Array*'s BFM Test Bench *Result SystemC* (Equal)

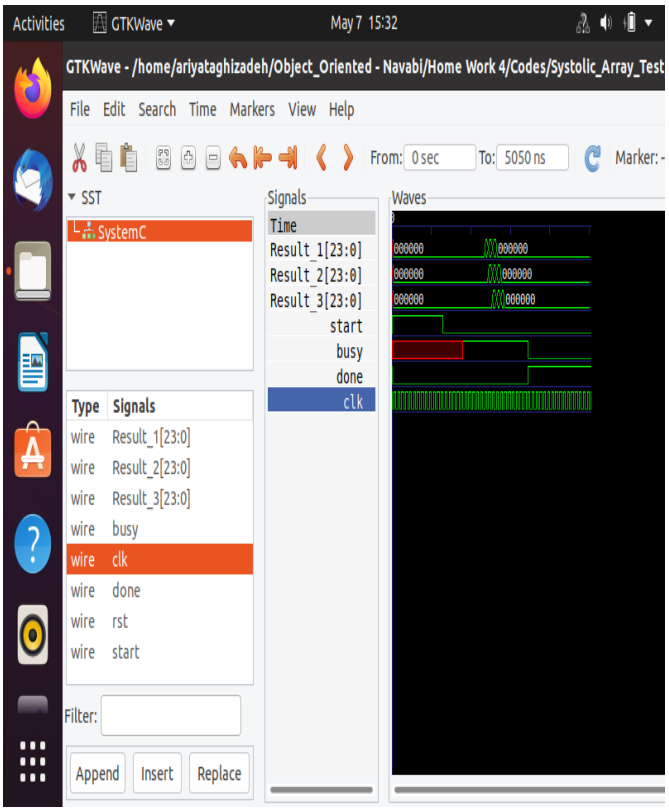

Fig. 46 *Systolic Array*'s BFM Test Bench *Result SystemC* (Equal)

`Input Matrix is smaller than *Systolic Array*:

The input weigh matrix is:

0	1	1
0	1	1
0	1	1

The input data matrix is:

1	1
1	0

So the output matrix must be:

2	1
2	1

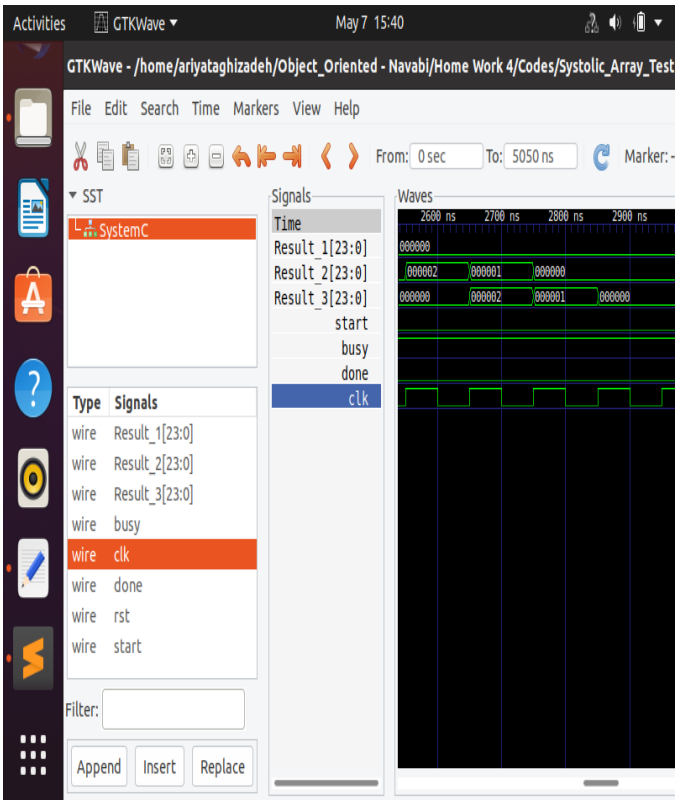

Fig. 47 *Systolic Array*'s BFM Test Bench *Result SystemC* (Smaller)

Input Matrix is larger than *Systolic Array*:

The input weigh matrix is:

1	1	1
1	1	1
1	1	1

The input data matrix is:

2	2	2	2	2	1
2	2	2	1	0	0
2	1	0	0	0	0

So the output matrix must be:

6	5	4	3	2	1
6	5	4	3	2	1
6	5	4	3	2	1

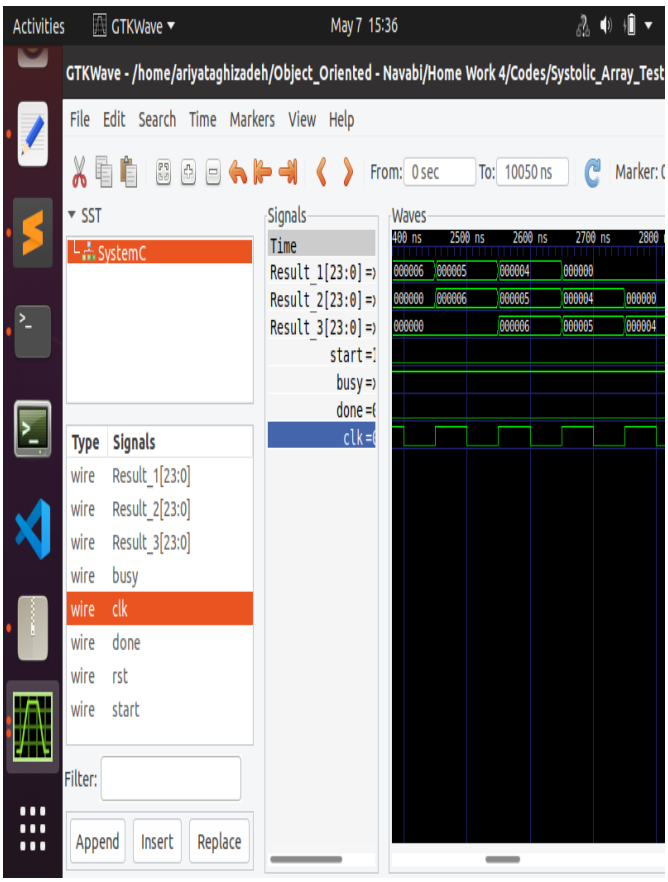

Fig. 48 *Systolic Array*'s Test Bench *Result SystemC* (Larger)

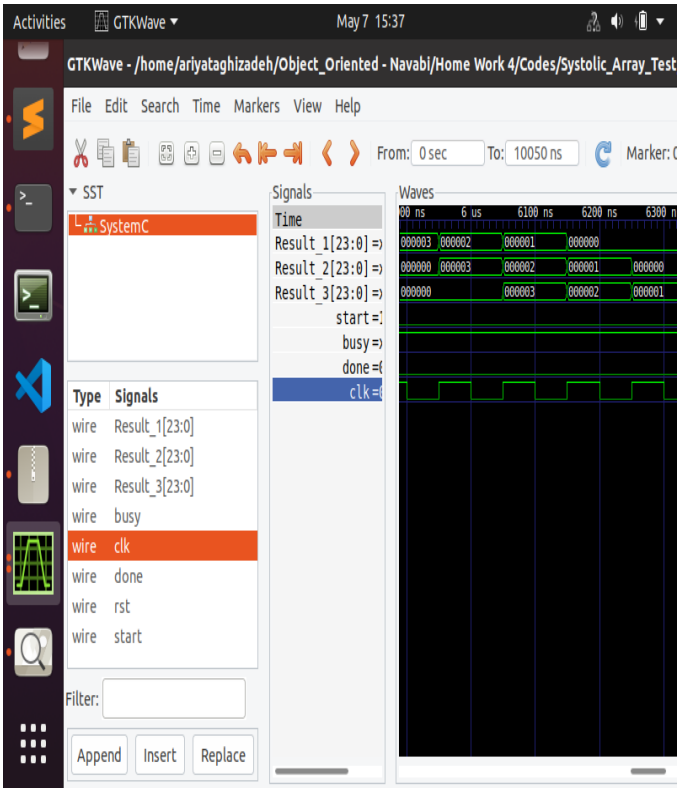

Fig. 49 *Systolic Array*'s BFM Test Bench *Result SystemC* (Larger)

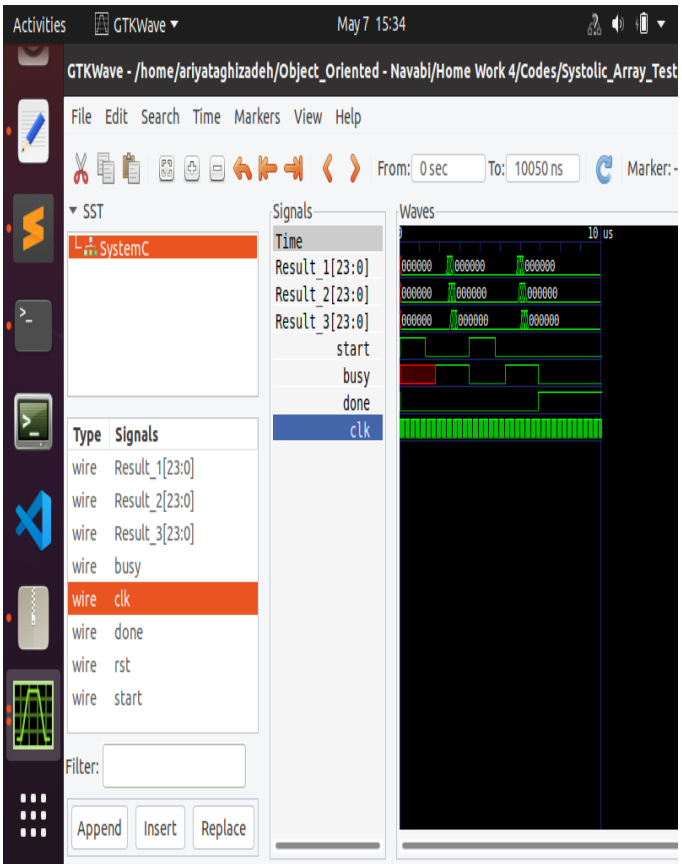

Fig. 50 *Systolic Array*'s BFM Test Bench *Result SystemC* (Larger)