

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

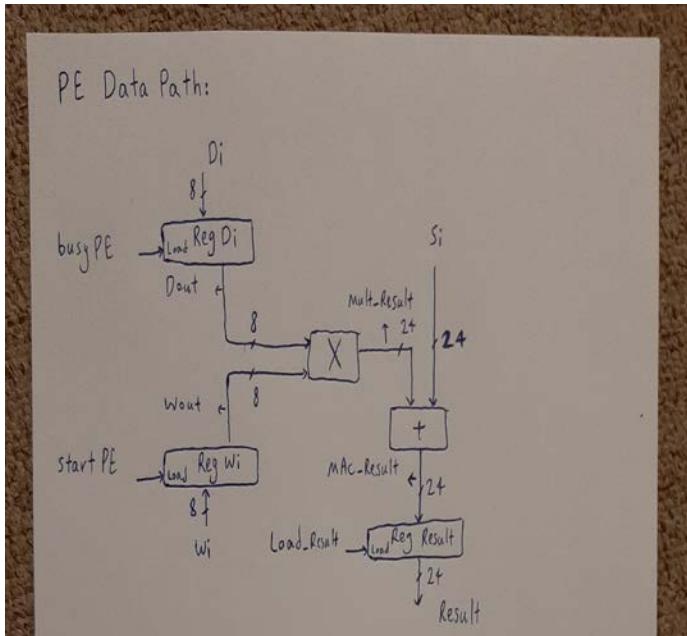


Fig. 1 PE's Data Path Schematic

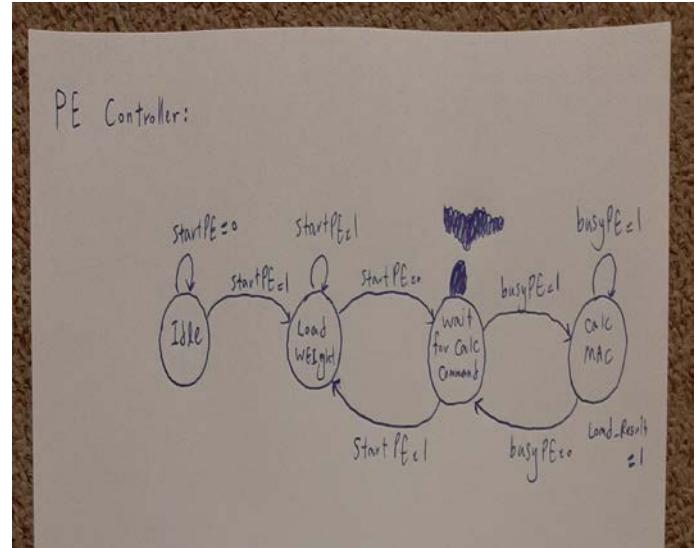


Fig. 2 PE's Controller Schematic

PE's Data Path and Controller in SystemC

Data Path:

```
PE_datapath.h
~Object_Oriented - Navalp/Home Work 4/Codes/S...
Save
PE_datapath.h
1 #include <systemc.h>
2 #include "Register.h"
3 #include "Adder.h"
4 #include "Mult.h"
5
6 SC_MODULE(PE_datapath){
7
8     // Port Declaration
9     sc_in<sc_logic> clk, rst;
10    sc_in<sc_logic> startPE, busyPE;
11    sc_in<sc_lv<8>> D1, W1;
12    sc_in<sc_lv<24>> S1;
13    sc_in<sc_logic> Load_Result;
14    sc_out<sc_lv<24>> Result;
15    sc_out<sc_lv<8>> Dout, Wout;
16
17    // Signal Declaration
18    sc_signal <sc_lv<24>> Mult_Result;
19    sc_signal <sc_lv<24>> MAC_Result;
20
21    // Instantiation
22    Register<8*> input_data;
23    Register<8*> input_weight;
24    Register<24*> result;
25    Adder<24*> adder;
26    Mult<8, 24*> multiplier;
27
28    hc_EOR(PE_datapath);
}
C:/b2C Header Tab Width: 8 Ln 28, Col 9 INS
```

Fig. 3 PE's Data Path *SystemC*

```

28     SC_CTOR(PE_datapath)[
29
30         input_data = new Register<8>("Data_Register");
31         input_data->clk(clk);
32         input_data->rst(rst);
33         input_data->load(busyPE);
34         input_data->regin(Di);
35         input_data->regout(Dout);
36
37         input_weight = new Register<8>("Weight_Register");
38         input_weight->clk(clk);
39         input_weight->rst(rst);
40         input_weight->load(startPE);
41         input_weight->regin(Wi);
42         input_weight->regout(Wout);
43
44         multiplier = new Mult<8, 24>("Multiplier");
45         multiplier->in1(Dout);
46         multiplier->in2(Wout);
47         multiplier->out(Mult_Result);
48
49         adder = new Adder<24>("adder");
50         adder->in1(Si);
51         adder->in2(Mult_Result);
52         adder->out(MAC_Result);
53
54         result = new Register<24>("Result_Register");
55
56     }

```

Fig. 4 PE's Data Path SystemC

```

1 #include <sysc.h>
2
3 SC_MODULE (PE_controller){
4
5     SC_in <sc_logic> rst, clk, startPE, busyPE;
6     SC_out <sc_logic> Load_Result;
7
8     enum states {IDLE, LOAD_WEIGHT, WAIT_FOR_START_CALC_COMMAND,
9                  CALC_MAC};
10    SC_signal <states> Pstate, Nstate;
11
12    SC_CTOR(PE_controller)
13    {
14        SC_METHOD (comb_function);
15        sensitive << Pstate << startPE << busyPE;
16        SC_THREAD (seq_function);
17        sensitive << clk << rst;
18    };
19
20    void comb_function();
21    void seq_function();
22 };

```

Fig. 6 PE's Controller SystemC

```

34     input_data->regin(Di);
35     input_data->regout(Dout);
36
37     input_weight = new Register<8>("Weight_Register");
38     input_weight->clk(clk);
39     input_weight->rst(rst);
40     input_weight->load(startPE);
41     input_weight->regin(Wi);
42     input_weight->regout(Wout);
43
44     multiplier = new Mult<8, 24>("Multiplier");
45     multiplier->in1(Dout);
46     multiplier->in2(Wout);
47     multiplier->out(Mult_Result);
48
49     adder = new Adder<24>("adder");
50     adder->in1(Si);
51     adder->in2(Mult_Result);
52     adder->out(MAC_Result);
53
54     result = new Register<24>("Result_Register");
55     result->clk(clk);
56     result->rst(rst);
57     result->load(Load_Result);
58     result->regin(MAC_Result);
59     result->regout(Result);
60
61 };

```

Fig. 5 PE's Data Path SystemC

```

1 #include "PE_controller.h"
2
3 void PE_controller::comb_function()
4 {
5     // Inactive output values
6     Load_Result = SC_LOGIC_0;
7
8     switch( Pstate ){
9         case IDLE:
10            if (startPE == SC_LOGIC_1) Nstate = LOAD_WEIGHT;
11            else Nstate = IDLE;
12            break;
13        case LOAD_WEIGHT:
14            if (startPE == SC_LOGIC_0) Nstate = WAIT_FOR_START_CALC_COMMAND;
15            else Nstate = LOAD_WEIGHT;
16            break;
17        case WAIT_FOR_START_CALC_COMMAND:
18            if (busyPE == SC_LOGIC_1) Nstate = CALC_MAC;
19            else{
20                if(startPE == SC_LOGIC_1) Nstate = LOAD_WEIGHT;
21                else Nstate = WAIT_FOR_START_CALC_COMMAND;
22            }
23            break;
24        case CALC_MAC:
25            Load_Result = SC_LOGIC_1;
26            if (busyPE == SC_LOGIC_0) Nstate = WAIT_FOR_START_CALC_COMMAND;
27            break;
28        default:
29            Nstate = IDLE;
30            break;
31    }
32 }

```

Fig. 7 PE's Controller Combinational SystemC

```

13     case LOAD_WEIGHT:
14         if (startPE == SC_LOGIC_0) Nstate = WAIT_FOR_START_CALC_COMMAND;
15         else Nstate = LOAD_WEIGHT;
16         break;
17     case WAIT_FOR_START_CALC_COMMAND:
18         if (busyPE == SC_LOGIC_1) Nstate = CALC_MAC;
19         else{
20             if(startPE == SC_LOGIC_1) Nstate = LOAD_WEIGHT;
21             else Nstate = WAIT_FOR_START_CALC_COMMAND;
22         }
23         break;
24     case CALC_MAC:
25         Load_Result = SC_LOGIC_1;
26         if (busyPE == SC_LOGIC_0) Nstate = WAIT_FOR_START_CALC_COMMAND;
27         break;
28     default:
29         Nstate = IDLE;
30         break;
31     }
32 }
33
34 void PE_controller::seq_function(){
35     while (1){
36         if (rst == '1')
37             Pstate = IDLE;
38         else if ((clk->event()) &&(clk == '1'))
39             Pstate = Nstate;
40         wait();
41     }
42 }

```

Line 34, Column 1 Tab Size: 4 C++

Fig. 8 PE's Controller Sequential SystemC

Top Module:

```

1 #include "PE_datapath.h"
2 #include "PE_controller.h"
3
4 SC_MODULE(PE){
5
6     // Port Declaration
7     sc_in <sc_logic> clk, rst;
8     sc_in <sc_logic> startPE, busyPE;
9     sc_in <sc_lv<8>> Di, Wi;
10    sc_in <sc_lv<24>> Si;
11    sc_out <sc_lv<24>> Result;
12    sc_out <sc_lv<8>> Dout, Wout;
13
14     // Signal Declaration
15     sc_signal <sc_logic> Load_Result;
16
17     PE_datapath* DataPath;
18     PE_controller* Controller;
19
20     SC_CTOR(PE){
21         DataPath = new PE_datapath ("Datapath");
22         (*DataPath) (clk, rst, startPE, busyPE, Di, Wi, Si,
23 Load_Result, Result, Dout, Wout);
24
25         Controller = new PE_controller ("Controller");
26         (*Controller) (rst, clk, startPE, busyPE,
27             C/ObjC Header ▾ Tab Width: 8 ▾ Ln 4, Col 1 ▾ INS

```

Fig. 9 PE's Top Module SystemC

```

1 PE_topModule.h
2
3 PE_datapath.h x PE_controller.h x PE_topModule.h
4
5 SC_MODULE(PE){
6
7     // Port Declaration
8     sc_in <sc_logic> clk, rst;
9     sc_in <sc_logic> startPE, busyPE;
10    sc_in <sc_lv<8>> Di, Wi;
11    sc_in <sc_lv<24>> Si;
12    sc_out <sc_lv<24>> Result;
13    sc_out <sc_lv<8>> Dout, Wout;
14
15     // Signal Declaration
16     sc_signal <sc_logic> Load_Result;
17
18     PE_datapath* DataPath;
19     PE_controller* Controller;
20
21     SC_CTOR(PE){
22         DataPath = new PE_datapath ("Datapath");
23         (*DataPath) (clk, rst, startPE, busyPE, Di, Wi, Si,
24 Load_Result, Result, Dout, Wout);
25
26         Controller = new PE_controller ("Controller");
27         (*Controller) (rst, clk, startPE, busyPE,
28             C/ObjC Header ▾ Tab Width: 8 ▾ Ln 4, Col 1 ▾ INS

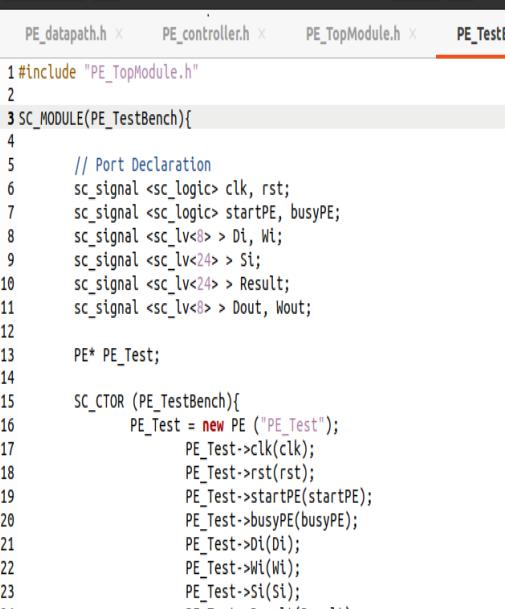
```

Fig. 10 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.

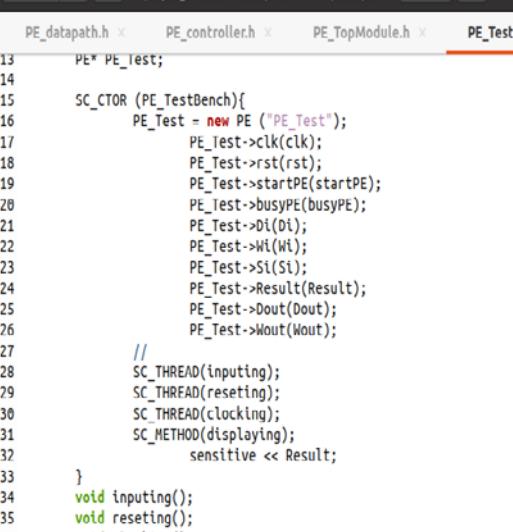
Test Bench:



```
PE_TestBench.h
~Object_Oriented - Navab/Home Work 4/Codes/S...
Save
PE_datapath.h x PE_controller.h x PE_TopModule.h x PE_TestBench.h x

1 #include "PE_TopModule.h"
2
3 SC_MODULE(PE_TestBench){
4
5     // Port Declaration
6     sc_signal <sc_logic> clk, rst;
7     sc_signal <sc_logic> startPE, busyPE;
8     sc_signal <sc_lv<8>> Di, Wi;
9     sc_signal <sc_lv<24>> Si;
10    sc_signal <sc_lv<24>> Result;
11    sc_signal <sc_lv<8>> Dout, Wout;
12
13    PE* PE_Test;
14
15    SC_CTOR (PE_TestBench){
16        PE_Test = new PE ("PE_Test");
17        PE_Test->clk(clk);
18        PE_Test->rst(rst);
19        PE_Test->startPE(startPE);
20        PE_Test->busyPE(busyPE);
21        PE_Test->i(Di);
22        PE_Test->Wi(Wi);
23        PE_Test->s(Si);
24        PE_Test->Result(Result);
25        PE_Test->Dout(Dout);
26        PE_Test->Wout(Wout);
27    }
28}
```

Fig. 11 PE's Test Bench SystemC



Activities Text Editor May 7 11:24

PE_TestBench.h Save

Object_Oriented_Navabi/Home Work 4/Codes/S...

PE_datopath.h PE_controller.h PE_TopModule.h PE_TestBench.h

```
13 PE* PE_Test;
14
15 SC_CTOR(PE_TestBench){
16     PE_Test = new PE ("PE_Test");
17     PE_Test->clk(clk);
18     PE_Test->rst(rst);
19     PE_Test->startPE(startPE);
20     PE_Test->busyPE(busyPE);
21     PE_Test->Di(Di);
22     PE_Test->Wi(Wi);
23     PE_Test->Si(Si);
24     PE_Test->Result(Result);
25     PE_Test->Dout(Dout);
26     PE_Test->Wout(Wout);
27
28     //SC_THREAD(inputting);
29     //SC_THREAD(reseting);
30     //SC_THREAD(clocking);
31     //SC_METHOD(displaying);
32     //sensitive << Result;
33 }
34 void inputting();
35 void resetting();
36 void clocking();
37 void displaying();
38 };
```

Fig. 12 PE's Test Bench SystemC

The screenshot shows a Sublime Text window with the following details:

- Title Bar:** Activities, Sublime Text ▾, May 7 11:24 •
- File Path:** ~Object_Oriented - Navabi/Home Work 4/Codes/Systolic_Array/PE/PE_TestBe...
- Menu Bar:** File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, Help
- Toolbar:** Includes icons for back, forward, search, and file operations.
- Code Editor:** The main area contains C++ code for a test bench. The code initializes a PE object, sets input values (Wi, Di), and performs multiple cycles of starting the PE, waiting for it to finish, and checking its busy status. It also sets output values (Si) and waits for the process to complete.
- Right Panel:** Shows a sidebar with project files and a vertical panel with navigation and search tools.

```
1 #include "PE_TestBench.h"
2
3 void PE TestBench::inputing(){
4     Wi = "00000011";
5     startPE = SC_LOGIC_0;
6     wait(100, SC_NS);
7     startPE = SC_LOGIC_1;
8     wait(100, SC_NS);
9     startPE = SC_LOGIC_0;
10    wait(100, SC_NS);
11    busyPE = SC_LOGIC_1;
12    wait(500, SC_NS);
13    Si = "11111000000000000000000000000000";
14    Di = "10000000";
15    busyPE = SC_LOGIC_0;
16    wait(100, SC_NS);
17    busyPE = SC_LOGIC_1;
18    wait(500, SC_NS);
19    Si = "11111111111111111111111111111111";
20    Di = "00000001";
21    busyPE = SC_LOGIC_0;
22    wait(100, SC_NS);
23    busyPE = SC_LOGIC_1;
24    wait(500, SC_NS);
25    Si = "0000000000000000000000000000000011111";
26    Di = "10000000";
27    busyPE = SC_LOGIC_0;
28    wait(100, SC_NS);
29    busyPE = SC_LOGIC_1;
30    wait(500, SC_NS);
```

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

The screenshot shows a Sublime Text window with the following details:

- Activity Bar:** Activities, Sublime Text, May 7 11:24.
- File Explorer:** Shows icons for Home Work 4, Codes, Systolic_Array, PE, PE_TestBench, and a large file named Oriented - Navabi... (partially visible).
- Text Editor:** The main pane displays C++ code for a test bench. The code includes methods for clocking, resetting, and displaying results. It uses SC logic and SC_NS for timing.
- Status Bar:** Line 33, Column 1, Tab Size: 4, C++.

```
File Edit Selection Find View Goto Tools Project Preferences Help
PE_controller.cpp x PE_TestBench.cpp x Oriented - Navabi.../Systolic_Array_TestBench.x... x + ▾
29     busType = DL_LOGIC_1;
30     wait(500, SC_NS);
31 }
32
33 void PE_TestBench::clocking(){
34     int i;
35     clk = sc_logic('1');
36     for (i = 0; i <= 50; i++)
37     {
38         clk = sc_logic('0');
39         wait(50, SC_NS);
40         clk = sc_logic('1');
41         wait(50, SC_NS);
42     }
43 }
44
45 void PE_TestBench::resetting(){
46     rst = (sc_logic)'0';
47     wait(5, SC_NS);
48     rst = (sc_logic)'1';
49     wait(5, SC_NS);
50     rst = (sc_logic)'0';
51 };
52
53 void PE_TestBench::displaying(){
54     cout << " Si = " << Si.read() << endl;
55     cout << " Di = " << Di.read() << endl;
56     cout << " Wi = " << Wi.read() << endl;
57     cout << " Result = " << Result.read() << endl;
58 }
```

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

Main:

A vertical dock on the left side of the screen containing nine icons: a flame, a person, a folder, a document, a letter, a question mark, a target, a grid, and a terminal.

Activities Sublime Text May 7 11:24

~Object_Oriented - Navabi/Home Work 4/Codes/Systolic_Array/PE/main.cpp - ... - ⌂ X

File Edit Selection Find View Goto Tools Project Preferences Help

◀ | PE_TestBench.cpp x main.cpp - Object_Oriented - Navabi.../PE x Systolic_Array_Test_Larger x + ▼

```
1 #include "PE_TestBench.cpp"
2
3 int sc_main (int argc, char **argv){
4
5     PE_TestBench PE ("testbench");
6
7     sc_start (12000, SC_NS);
8
9 }
```

Line 3, Column 1: Detect Indentation: Setting indentation to tabs with width 4 Tab Size: 4

Fig. 15 PE's Main SystemC

PE's Test Bench Results:

```
Si = 1111100000000000000000000000  
Di = 10000000  
Wi = 00000011  
Result = 111110000000000011000
```

```
Si = 1111111111111111111111  
Di = 0000001  
Wi = 0000011  
Result = 00000000000000000000
```

```
Si = 0000000000000000000000111111  
Di = 1000000  
Wi = 00000011  
Result = 00000000000000000000001100111111
```

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

B. RTL Model of Systolic Array in SystemC

Systolic Array's Data Path and Controller Schematic

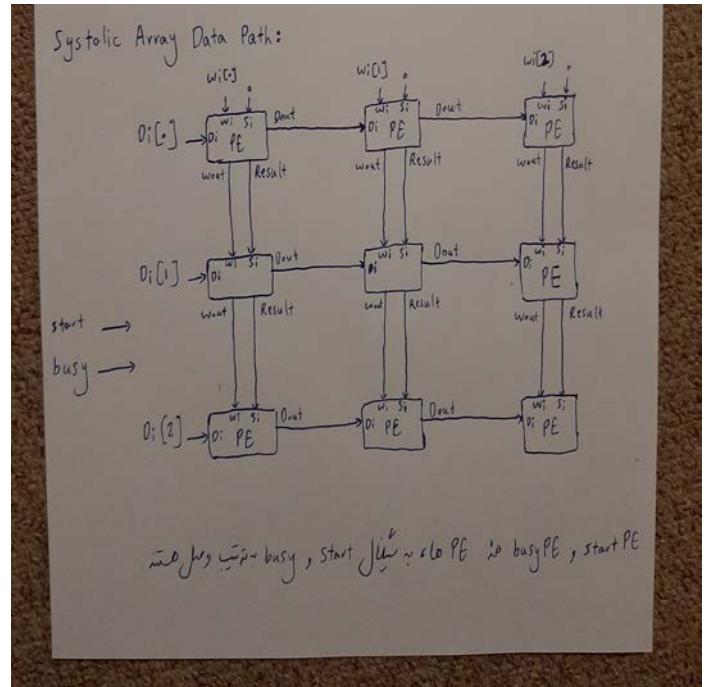


Fig. 17 Systolic Array's Data Path Schematic

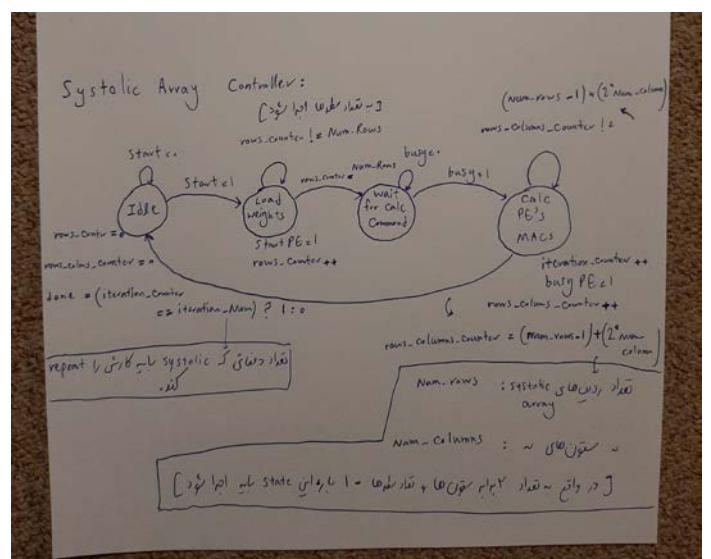
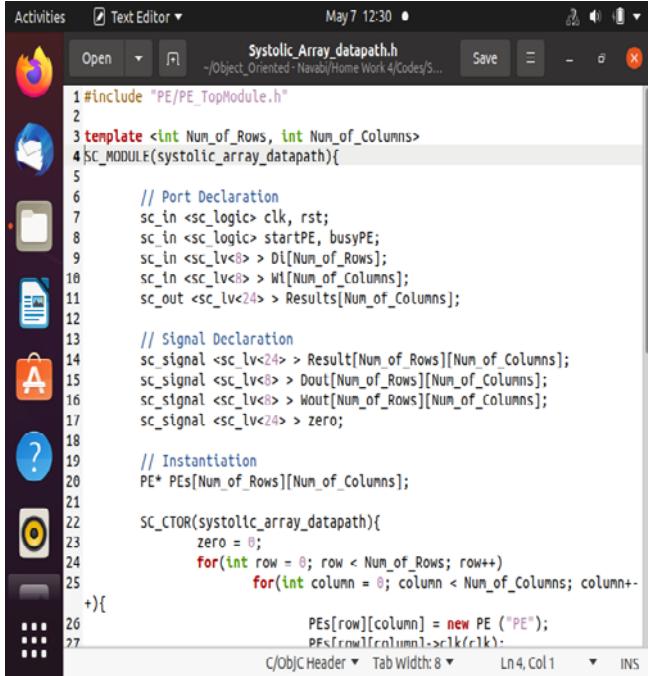


Fig. 18 Systolic Array's Controller Schematic

Systolic Array's Data Path and Controller in SystemC

To design the Data Path of *Systolic Array*, simply an array of *PEs* 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:

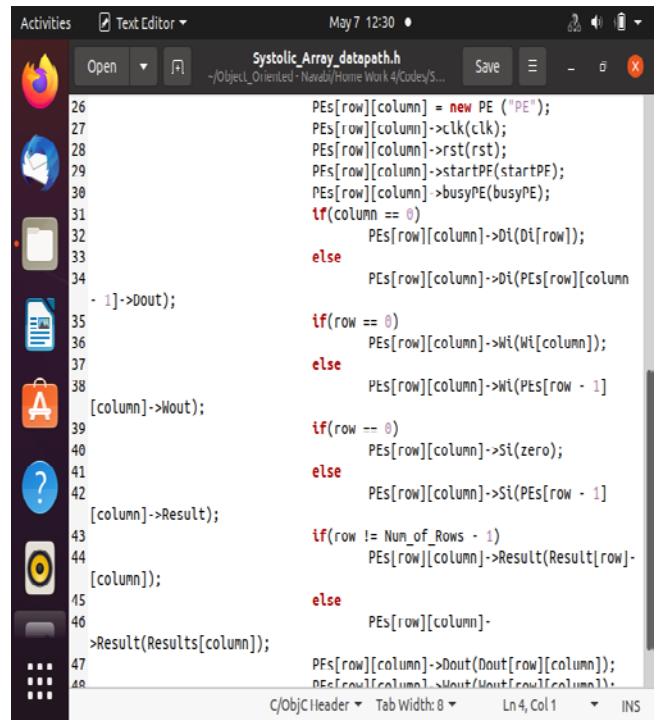


```

1 #include "PE/PE_TopoModule.h"
2
3 template <int Num_of_Rows, int Num_of_Columns>
4 SC_MODULE(systolic_array_datapath){
5
6     // Port Declaration
7     sc_in <sc_logic> clk, rst;
8     sc_in <sc_logic> startPE, busyPE;
9     sc_in <sc_lv<8>> DIn[Num_of_Rows];
10    sc_in <sc_lv<8>> WIn[Num_of_Columns];
11    sc_out <sc_lv<24>> Results[Num_of_Columns];
12
13    // Signal Declaration
14    sc_signal <sc_lv<24>> Result[Num_of_Rows][Num_of_Columns];
15    sc_signal <sc_lv<8>> Dout[Num_of_Rows][Num_of_Columns];
16    sc_signal <sc_lv<8>> Wout[Num_of_Rows][Num_of_Columns];
17    sc_signal <sc_lv<24>> zero;
18
19    // Instantiation
20    PE* PEs[Num_of_Rows][Num_of_Columns];
21
22    SC_CTOR(systolic_array_datapath){
23        zero = 0;
24        for(int row = 0; row < Num_of_Rows; row++)
25            for(int column = 0; column < Num_of_Columns; column++)
26            {
27                PEs[row][column] = new PE ("PE");
28                PEs[row][column]->clk(c1k);
29                PEs[row][column]->rst(rst);
30                PEs[row][column]->startPE(startPE);
31                PEs[row][column]->busyPE(busyPE);
32                if(column == 0)
33                    PEs[row][column]->Di(Di[row]);
34                else
35                    PEs[row][column]->Di(PEs[row][column - 1]->Dout);
36
37                [column]->Wout;
38
39                [column]->Result;
40
41                [column]->Result;
42
43                >Result(Results[column]);
44
45                [column]->Result(Results[column]);
46
47                PEs[row][column]->Dout(Dout[row][column]);
48                PEs[row][column]->Wout(Wout[row][column]);
49            }
50        }
51    };

```

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

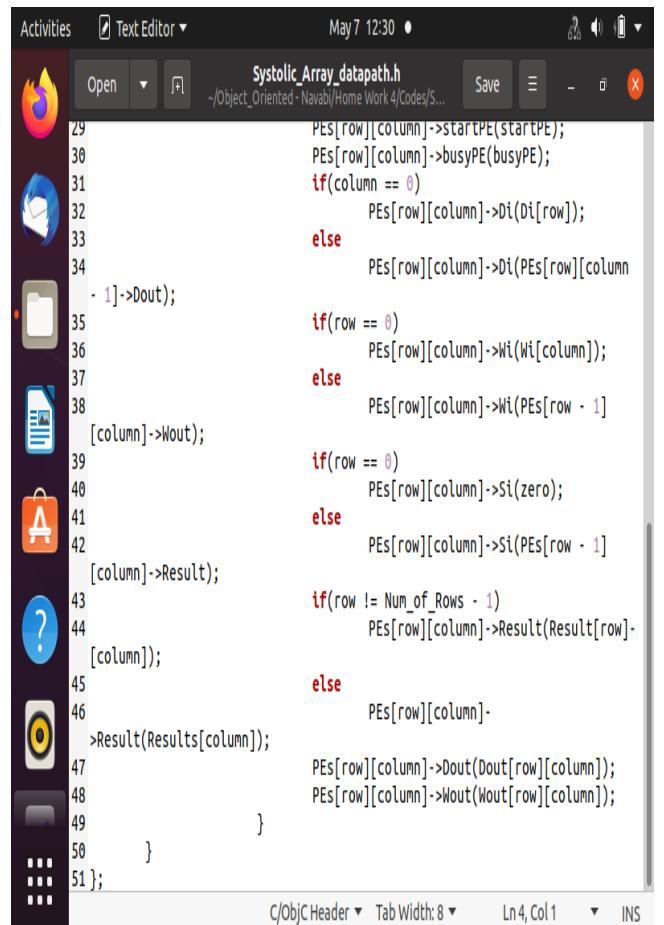


```

26 PEs[row][column] = new PE ("PE");
27 PEs[row][column]->clk(clk);
28 PEs[row][column]->rst(rst);
29 PEs[row][column]->startPE(startPE);
30 PEs[row][column]->busyPE(busyPE);
31 if(column == 0)
32     PEs[row][column]->Di(Di[row]);
33 else
34     PEs[row][column]->Di(PEs[row][column - 1]->Dout);
35
36 if(row == 0)
37     PEs[row][column]->Wt(Wt[0]);
38 else
39     PEs[row][column]->Wt(PEs[row - 1]);
40
41 if(row == 0)
42     PEs[row][column]->Si(zero);
43 else
44     PEs[row][column]->Si(PEs[row - 1]);
45
46 if(row != Num_of_Rows - 1)
47     PEs[row][column]->Result(Result[row]);
48 else
49     PEs[row][column]->Result(Results[column]);
50
51 PEs[row][column]->Dout(Dout[row][column]);
52 PEs[row][column]->Wout(Wout[row][column]);

```

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



```

29 PEs[row][column]->startPE(startPE);
30 PEs[row][column]->busyPE(busyPE);
31 if(column == 0)
32     PEs[row][column]->Di(Di[row]);
33 else
34     PEs[row][column]->Di(PEs[row][column - 1]->Dout);
35
36 if(row == 0)
37     PEs[row][column]->Wt(Wt[0]);
38 else
39     PEs[row][column]->Wt(PEs[row - 1]);
40
41 if(row == 0)
42     PEs[row][column]->Si(zero);
43 else
44     PEs[row][column]->Si(PEs[row - 1]);
45
46 if(row != Num_of_Rows - 1)
47     PEs[row][column]->Result(Result[row]);
48 else
49     PEs[row][column]->Result(Results[column]);
50
51 PEs[row][column]->Dout(Dout[row][column]);
52 PEs[row][column]->Wout(Wout[row][column]);

```

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

Controller:

```

Activities Text Editor May 7 12:31
Systolic_Array_controller.h
~/Object_Oriented - Navabi/Home Work 4/Codes/Systolic_Array/S...
Open Save
Systolic_Array_datapath.h Systolic_Array_controller.h
1 #include <systemc.h>
2
3 template <int Num_of_Rows, int Num_of_Columns, int Num_of_Iterations>
4 SC_MODULE (systolic_array_controller){
5
6     sc_in <sc_logic> rst, clk, start, busy;
7     sc_out <sc_logic> startPE, busyPE, done;
8
9     enum states {IDLE, LOAD_WEIGHTS, WAIT_FOR_START_CALC_COMMAND,
10
11     CALC_PEs_MACS};
12
13     sc_signal <states> Pstate, Nstate;
14     sc_signal <sc_logic> co_rows_counter, co_rows_and_columns_counter;
15
16     int iteration_counter;
17     int rows_and_columns_counter;
18     int rows_counter;
19
20     SC_CTOR(systolic_array_controller)
21     {
22         SC_METHOD (comb_function);
23         sensitive << co_rows_counter << co_rows_and_columns_counter
24 << start << busy;
25         SC_THREAD (seq_function);
26         sensitive << clk << rst;
27     };
28
29     void comb_function();
30     void seq_function();
31 };

```

Fig. 22 Systolic Array's Controller SystemC

```

Activities Text Editor May 7 12:31
Systolic_Array_controller.h
~/Object_Oriented - Navabi/Home Work 4/Codes/Systolic_Array/S...
Open Save
Systolic_Array_datapath.h Systolic_Array_controller.h
7     sc_out <sc_logic> startPE, busyPE, done;
8
9     enum states {IDLE, LOAD_WEIGHTS, WAIT_FOR_START_CALC_COMMAND,
10
11     CALC_PEs_MACS};
12
13     sc_signal <states> Pstate, Nstate;
14     sc_signal <sc_logic> co_rows_counter, co_rows_and_columns_counter;
15
16     int iteration_counter;
17     int rows_and_columns_counter;
18     int rows_counter;
19
20     SC_CTOR(systolic_array_controller)
21     {
22         SC_METHOD (comb_function);
23         sensitive << co_rows_counter << co_rows_and_columns_counter
24 << start << busy;
25         SC_THREAD (seq_function);
26         sensitive << clk << rst;
27     };
28
29     void comb_function();
30     void seq_function();
31 };

```

Fig. 23 Systolic Array's Controller SystemC

```

Activities Sublime Text May 7 12:31
~/Object_Oriented - Navabi/Home Work 4/Codes/Systolic_Array/Systolic_Array...
File Edit Selection Find View Goto Tools Project Preferences Help
main.cpp - Object_Oriented Systolic_Array_controller.cpp x Navabi.../Systolic_Array_Test_Larger x + ▾
1 #include "Systolic_Array_controller.h"
2
3 template <int Num_of_Rows, int Num_of_Columns, int Num_of_Iterations>
4 void systolic_array_controller<Num_of_Rows, Num_of_Columns, Num_of_Iterations>(SC_THREAD_ARGS)
5 {
6     // Inactive output values
7     done = SC_LOGIC_0;
8     busyPE = SC_LOGIC_0;
9     startPE = SC_LOGIC_0;
10
11     switch( Pstate ){
12         case IDLE:
13             done = (iteration_counter == Num_of_Iterations) ? SC_LOGIC_1 : SC_LOGIC_0;
14             rows_counter = 0;
15             rows_and_columns_counter = 0;
16             if (start == SC_LOGIC_1) Nstate = LOAD_WEIGHTS;
17             else Nstate = IDLE;
18             break;
19         case LOAD_WEIGHTS:
20             if (co_rows_counter == SC_LOGIC_1) Nstate = WAIT_FOR_START_CALC_COMMAND;
21             else startPE = SC_LOGIC_1;
22             break;
23         case WAIT_FOR_START_CALC_COMMAND:
24             if (busy == SC_LOGIC_1) Nstate = CALC_PEs_MACS;
25             else Nstate = WAIT_FOR_START_CALC_COMMAND;
26             break;
27         case CALC_PEs_MACS:
28             if (co_rows_and_columns_counter == SC_LOGIC_1){
29                 Nstate = IDLE;
30                 iteration_counter++;
31             }
32         else busyPE = SC_LOGIC_1;
33         break;
34     default:
35         Nstate = IDLE;
36         break;
37     }
38 }
39
40 template <int Num_of_Rows, int Num_of_Columns, int Num_of_Iterations>
41 void systolic_array_controller<Num_of_Rows, Num_of_Columns, Num_of_Iterations>(SC_THREAD_ARGS)
42 while(1){
43     if (rst == '1'){
44         Pstate = IDLE;
45         iteration_counter = 0;
46     }
47     else if (clk->event() && (clk == '1')){
48         Pstate = Nstate;
49         if (Pstate == LOAD_WEIGHTS) {
50             rows_counter++;
51             co_rows_counter = (rows_counter > Num_of_Rows) ? SC_LOGIC_1 : SC_LOGIC_0;
52         }
53         if (Pstate == CALC_PEs_MACS) {
54             rows_and_columns_counter++;
55             co_rows_and_columns_counter = (rows_and_columns_counter > Num_of_Columns) ? SC_LOGIC_1 : SC_LOGIC_0;
56         }
57         wait();
58     }
59 }

```

Fig. 24 Systolic Array's Controller Combinational SystemC

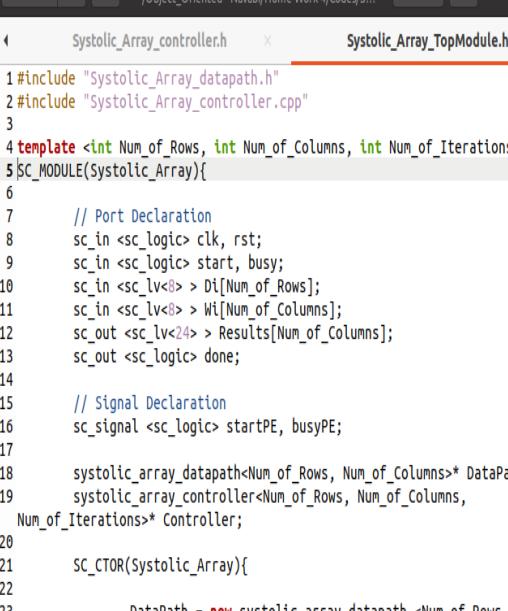
```

Activities Sublime Text May 7 12:31
~/Object_Oriented - Navabi/Home Work 4/Codes/Systolic_Array/Systolic_Array...
File Edit Selection Find View Goto Tools Project Preferences Help
main.cpp - Object_Oriented Systolic_Array_controller.cpp x Navabi.../Systolic_Array_Test_Larger x + ▾
31     }
32     else busyPE = SC_LOGIC_1;
33     break;
34 default:
35     Nstate = IDLE;
36     break;
37 }
38 }
39
40 template <int Num_of_Rows, int Num_of_Columns, int Num_of_Iterations>
41 void systolic_array_controller<Num_of_Rows, Num_of_Columns, Num_of_Iterations>(SC_THREAD_ARGS)
42 while(1){
43     if (rst == '1'){
44         Pstate = IDLE;
45         iteration_counter = 0;
46     }
47     else if (clk->event() && (clk == '1')){
48         Pstate = Nstate;
49         if (Pstate == LOAD_WEIGHTS) {
50             rows_counter++;
51             co_rows_counter = (rows_counter > Num_of_Rows) ? SC_LOGIC_1 : SC_LOGIC_0;
52         }
53         if (Pstate == CALC_PEs_MACS) {
54             rows_and_columns_counter++;
55             co_rows_and_columns_counter = (rows_and_columns_counter > Num_of_Columns) ? SC_LOGIC_1 : SC_LOGIC_0;
56         }
57         wait();
58     }
59 }

```

Fig. 25 Systolic Array's Controller Sequential SystemC

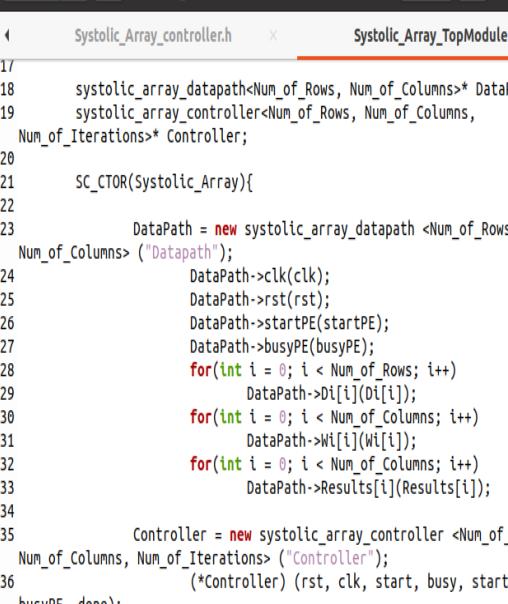
Top Module:



The screenshot shows a software development environment with a toolbar on the left containing icons for file operations, a search bar, and tabs. The main area displays the code for `Systolic_Array_TopModule.h`. The code includes header imports, a template class definition for `Systolic_Array`, and declarations for ports and signals. It also defines two objects: `DataPath` and `Controller`, and initializes `DataPath` with a constructor call.

```
1 #include "Systolic_Array_datapath.h"
2 #include "Systolic_Array_controller.cpp"
3
4 template <int Num_of_Rows, int Num_of_Columns, int Num_of_Iterations>
5 SC_MODULE(Systolic_Array){
6
7     // Port Declaration
8     sc_in <sc_logic> clk, rst;
9     sc_in <sc_logic> start, busy;
10    sc_in <sc_lv<16>> D1[Num_of_Rows];
11    sc_in <sc_lv<8>> W1[Num_of_Columns];
12    sc_out <sc_lv<24>> Results[Num_of_Columns];
13    sc_out <sc_logic> done;
14
15    // Signal Declaration
16    sc_signal <sc_logic> startPE, busyPE;
17
18    systolic_array_datapath<Num_of_Rows, Num_of_Columns>* DataPath;
19    systolic_array_controller<Num_of_Rows, Num_of_Columns,
20        Num_of_Iterations>* Controller;
21
22    SC_CTOR(Systolic_Array){
23
24        DataPath = new systolic_array_datapath <Num_of_Rows,
25            Num_of_Columns> ("Datapath");
26        DataPath->clk(clk);
27    }
28}
```

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



The screenshot shows a software development environment with a sidebar containing icons for file operations (Open, Save, Undo, Redo) and tabs for different files. The main area displays the code for `Systolic_Array_TopModule.h`. The code defines a class `Systolic_Array_controller.h` with a constructor `SC_CTOR(Systolic_Array)`. Inside the constructor, it initializes a `systolic_array_datapath` object named `DataPath` with parameters `Num_of_Rows`, `Num_of_Columns`, and `Num_of_Iterations`. The `DataPath` object is then configured with clock (`clk`), reset (`rst`), start PE (`startPE`), busy PE (`busyPE`), and three nested loops for data exchange between `Di`, `Wi`, and `Results` arrays. Finally, a `systolic_array_controller` object named `Controller` is created with parameters `Num_of_Rows`, `Num_of_Columns`, and `Num_of_Iterations`, and assigned to the `*Controller` pointer.

```
17
18     systolic_array_datapath<Num_of_Rows, Num_of_Columns>* DataPath;
19     systolic_array_controller<Num_of_Rows, Num_of_Columns,
20     Num_of_Iterations>* Controller;
21
22     SC_CTOR(Systolic_Array){
23
24         DataPath = new systolic_array_datapath <Num_of_Rows,
25             Num_of_Columns> ("Datapath");
26
27         DataPath->clk(clk);
28         DataPath->rst(rst);
29         DataPath->startPE(startPE);
30         DataPath->busyPE(busyPE);
31
32         for(int i = 0; i < Num_of_Rows; i++)
33             DataPath->Di[i](D[i]);
34
35         for(int i = 0; i < Num_of_Columns; i++)
36             DataPath->Wi[i](W[i]);
37
38         for(int i = 0; i < Num_of_Columns; i++)
39             DataPath->Results[i](Results[i]);
40
41         Controller = new systolic_array_controller <Num_of_Rows,
42             Num_of_Columns, Num_of_Iterations> ("Controller");
43
44         (*Controller) (rst, clk, start, busy, startPE,
45             busyPE, done);
46     }
47 }
```

Fig. 27 Systolic Array's Top Module SystemC

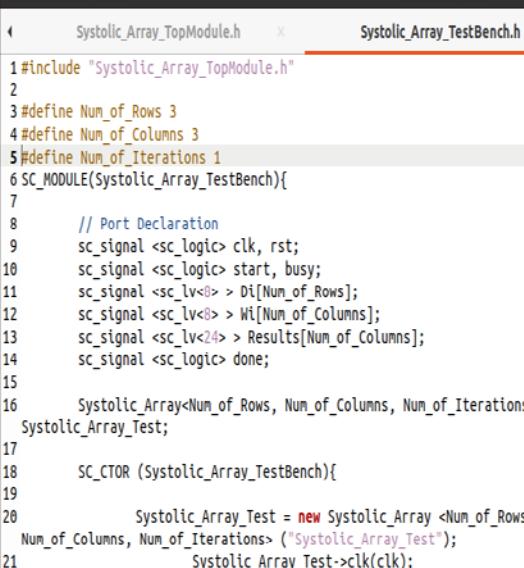
Systolic Array's Test Bench

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:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{matrix}$$

Test Bench:



The screenshot shows a software development environment with multiple tabs open. The current tab is `Systolic_Array_TestBench.h`, which contains the following C++ code:

```
#include "Systolic_Array_TopModule.h"
#define Num_of_Rows 3
#define Num_of_Columns 3
#define Num_of_Iterations 1
SC_MODULE(Systolic_Array_TestBench){
    // Port Declaration
    sc_signal <sc_logic> clk, rst;
    sc_signal <sc_logic> start, busy;
    sc_signal <sc_lv<8>> Dl[Num_of_Rows];
    sc_signal <sc_lv<8>> Wi[Num_of_Columns];
    sc_signal <sc_lv<24>> Results[Num_of_Columns];
    sc_signal <sc_logic> done;
    Systolic_Array<Num_of_Rows, Num_of_Columns, Num_of_Iterations>*
    Systolic_Array_Test;
    SC_CTOR (Systolic_Array_TestBench)
        Systolic_Array_Test = new Systolic_Array <Num_of_Rows,
        Num_of_Columns, Num_of_Iterations> ("Systolic_Array_Test");
        Systolic_Array_Test->clk(clk);
        Systolic_Array_Test->rst(rst);
        Systolic_Array_Test->start(start);
        Systolic_Array_Test->busw(busw);
```

Fig. 28 Systolic Array's Test Bench SystemC

```

19     Systolic_Array_Test = new Systolic_Array <Num_of_Rows,
20     Num_of_Columns, Num_of_Iterations> ("Systolic_Array_Test");
21     Systolic_Array_Test->clk(clk);
22     Systolic_Array_Test->rst(rst);
23     Systolic_Array_Test->start(start);
24     Systolic_Array_Test->busy(busy);
25     for(int i = 0; i < Num_of_Rows; i++)
26         Systolic_Array_Test->Di[i](Di[i]);
27     for(int i = 0; i < Num_of_Columns; i++)
28         Systolic_Array_Test->Wi[i](Wi[i]);
29     for(int i = 0; i < Num_of_Columns; i++)
30         Systolic_Array_Test->Results[i](Results[i]);
31     Systolic_Array_Test->done(done);
32 
33     SC_THREAD(inputting);
34     SC_THREAD(reseting);
35     SC_THREAD(clocking);
36     SC_METHOD(displaying);
37     sensitive << Results[0] << Results[1] << Results[2];
38 }
39 void inputting();
40 void resetting();
41 void clocking();
42 void displaying();
43 };

```

Fig. 29 Systolic Array's Test Bench SystemC

```

1 #include "Systolic_Array_TestBench.h"
2 
3 void Systolic_Array_TestBench::inputting(){
4     // Preloading Weights
5     start = SC_LOGIC_1;
6     wait(100, SC_NS);
7     Wi[0] = "00000001"; Wi[1] = "00000001"; Wi[2] = "00000001";
8     wait(100, SC_NS);
9     Wi[0] = "00000001"; Wi[1] = "00000001"; Wi[2] = "00000001";
10    wait(100, SC_NS);
11    Wi[0] = "00000001"; Wi[1] = "00000001"; Wi[2] = "00000001";
12    wait(100, SC_NS);
13    start = SC_LOGIC_0;
14    wait(500, SC_NS);
15 
16    // loading Input Matrix
17    Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000";
18    busy = SC_LOGIC_1;
19    wait(250, SC_NS);
20    Di[0] = "00000001"; Di[1] = "00000000"; Di[2] = "00000000";
21    wait(100, SC_NS);
22    Di[0] = "00000001"; Di[1] = "00000001"; Di[2] = "00000000";
23    wait(100, SC_NS);
24    Di[0] = "00000001"; Di[1] = "00000001"; Di[2] = "00000001";
25    wait(100, SC_NS);
26    Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000";
27    wait(100, SC_NS);
28    Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000";
29    wait(1000, SC_NS);
30    busy = SC_LOGIC_0;

```

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

```

19     wait(1000, SC_NS);
20     busy = SC_LOGIC_0;
21 }
22 
23 void Systolic_Array_TestBench::clocking(){
24     int i;
25     clk = sc_logic('1');
26     for (i = 0; i <= 50; i++)
27     {
28         clk = sc_logic('0');
29         wait (50, SC_NS);
30         clk = sc_logic('1');
31         wait (50, SC_NS);
32     }
33 }
34 
35 void Systolic_Array_TestBench::reseting(){
36     rst = (sc_logic)'0';
37     wait (5, SC_NS);
38     rst = (sc_logic)'1';
39     wait (5, SC_NS);
40     rst = (sc_logic)'0';
41 }
42 
43 void Systolic_Array_TestBench::displaying(){
44     cout << " Done = " << done.read() << endl;
45     cout << " Result 1 = " << Results[0].read() << endl;
46     cout << " Result 2 = " << Results[1].read() << endl;
47     cout << " Result 3 = " << Results[2].read() << endl;
48 }
49 
50 
51 
52 
53 
54 
55 
56 
57 
58 

```

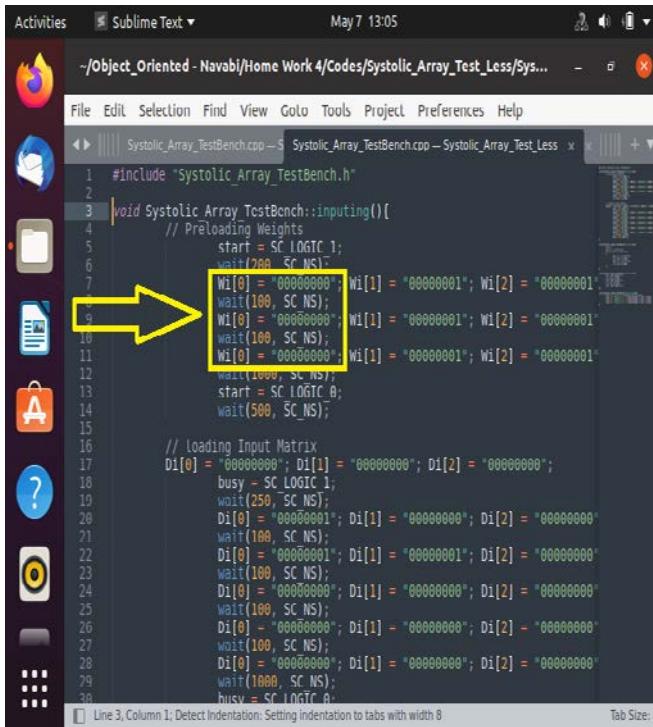
Fig. 31 Systolic Array's Test Bench *clocking*, *resetting*, *displaying* SystemC

```

1 #include "Systolic_Array_TestBench.cpp"
2 
3 int sc_main (int argc, char **argv){
4     Systolic_Array_TestBench Systolic_Array ("testbench");
5 
6     sc_trace_file* vcdfile;
7     vcdfile = sc_create_vcd_trace_file("Systolic_Array_test");
8     sc_trace(vcdfile, Systolic_Array.clk, "clk");
9     sc_trace(vcdfile, Systolic_Array.rst, "rst");
10    sc_trace(vcdfile, Systolic_Array.start, "start");
11    sc_trace(vcdfile, Systolic_Array.busy, "busy");
12    sc_trace(vcdfile, Systolic_Array.done, "done");
13    sc_trace(vcdfile, Systolic_Array.Results[0], "Result_1");
14    sc_trace(vcdfile, Systolic_Array.Results[1], "Result_2");
15    sc_trace(vcdfile, Systolic_Array.Results[2], "Result_3");
16 
17    sc_start (12000, SC_NS);
18    return 0;
19 }
20 
```

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*.



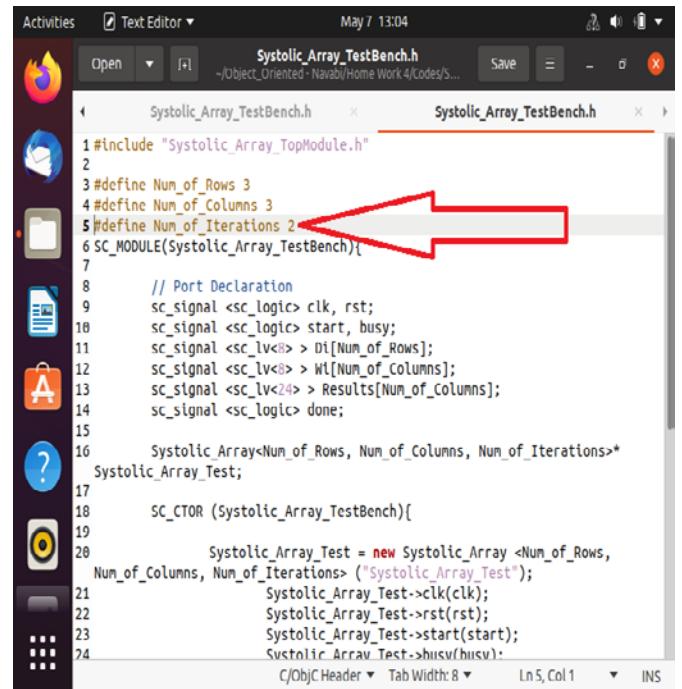
```

Activities Sublime Text May 7 13:05
-/Object_Oriented - Navabi/Home Work 4/Codes/Systolic_Array_Test_Less/Syst...
File Edit Selection Find View Goto Tools Project Preferences Help
Systolic_Array_TestBench.cpp - Systolic_Array_TestBench.cpp - Systolic_Array_Test_Less.x...
1 #include "Systolic_Array_TestBench.h"
2
3 void Systolic_Array_TestBench::inputting(){
4     // Preloading Weights
5     start = SC_LOGIC_1;
6     wait(100, SC_NS);
7     Wi[0] = "00000000"; Wi[1] = "00000001"; Wi[2] = "00000001";
8     wait(100, SC_NS);
9     Wi[0] = "00000000"; Wi[1] = "00000001"; Wi[2] = "00000001";
10    wait(100, SC_NS);
11    Wi[0] = "00000000"; Wi[1] = "00000001"; Wi[2] = "00000001";
12    wait(100, SC_NS);
13    start = SC_LOGIC_0;
14    wait(500, SC_NS);
15
16    // Loading Input Matrix
17    Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000";
18    busy = SC_LOGIC_1;
19    wait(250, SC_NS);
20    Di[0] = "00000001"; Di[1] = "00000000"; Di[2] = "00000000";
21    wait(100, SC_NS);
22    Di[0] = "00000001"; Di[1] = "00000001"; Di[2] = "00000000";
23    wait(100, SC_NS);
24    Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000";
25    wait(100, SC_NS);
26    Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000";
27    wait(100, SC_NS);
28    Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000";
29    busy = SC_LOGIC_0;
30
Line 3, Column 1: Detect Indentation: Setting Indentation to tabs with width 8 Tab Size: 8

```

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 *Iteration_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.

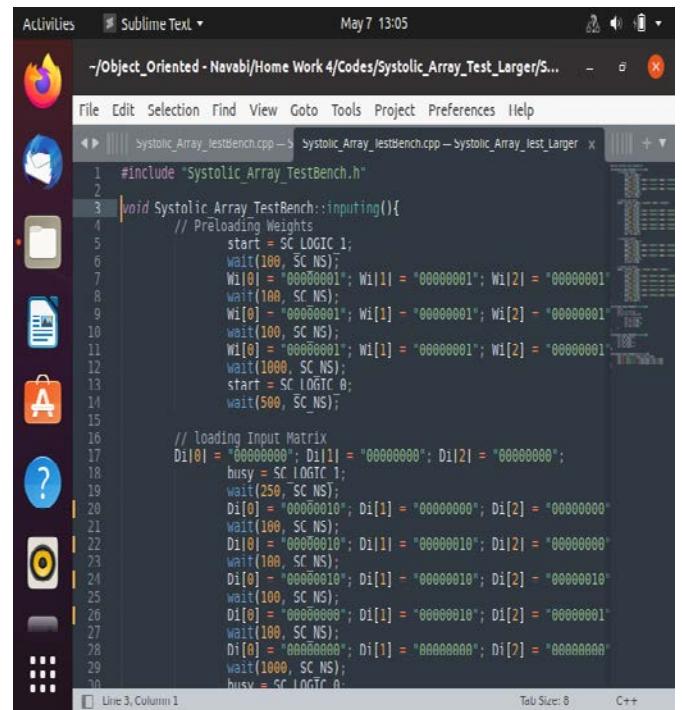


```

Activities Text Editor May 7 13:04
Systolic_Array_TestBench.h
1 #include "Systolic_Array_TopModule.h"
2
3 #define Num_of_Rows 3
4 #define Num_of_Columns 3
5 #define Num_of_Iterations 2
6 SC_MODULE(Systolic_Array_TestBench){
7
8     // Port Declaration
9     sc_signal <sc_logic> clk, rst;
10    sc_signal <sc_lv<8>> Di[Num_of_Rows];
11    sc_signal <sc_lv<8>> Wi[Num_of_Columns];
12    sc_signal <sc_lv<24>> Results[Num_of_Columns];
13    sc_signal <sc_logic> done;
14
15
16    Systolic_Array<Num_of_Rows, Num_of_Columns, Num_of_Iterations>*
17    Systolic_Array_Test;
18
19    SC_CTOR(Systolic_Array_TestBench){
20
21        Systolic_Array_Test = new Systolic_Array<Num_of_Rows,
22            Num_of_Columns, Num_of_Iterations> ("Systolic_Array_Test");
23        Systolic_Array_Test->clk(clk);
24        Systolic_Array_Test->rst(rst);
25        Systolic_Array_Test->start(start);
26        Systolic_Array_Test->busv(busv);
27
28
29
30
C/Object Header Tab Width: 8 Ln 5, Col 1 INS

```

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



```

Activities Sublime Text May 7 13:05
-/Object_Oriented - Navabi/Home Work 4/Codes/Systolic_Array_Test_Larger/S...
File Edit Selection Find View Goto Tools Project Preferences Help
Systolic_Array_TestBench.cpp - Systolic_Array_TestBench.cpp - Systolic_Array_Test_Larger.x...
1 #include "Systolic_Array_TestBench.h"
2
3 void Systolic_Array_TestBench::inputting(){
4     // Preloading Weights
5     start = SC_LOGIC_1;
6     wait(100, SC_NS);
7     Wi[0] = "00000001"; Wi[1] = "00000001"; Wi[2] = "00000001";
8     wait(100, SC_NS);
9     Wi[0] = "00000001"; Wi[1] = "00000001"; Wi[2] = "00000001";
10    wait(100, SC_NS);
11    Wi[0] = "00000001"; Wi[1] = "00000001"; Wi[2] = "00000001";
12    wait(100, SC_NS);
13    start = SC_LOGIC_0;
14    wait(500, SC_NS);
15
16    // loading Input Matrix
17    Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000";
18    busy = SC_LOGIC_1;
19    wait(250, SC_NS);
20    Di[0] = "00000010"; Di[1] = "00000000"; Di[2] = "00000000";
21    wait(100, SC_NS);
22    Di[0] = "00000010"; Di[1] = "00000010"; Di[2] = "00000000";
23    wait(100, SC_NS);
24    Di[0] = "00000010"; Di[1] = "00000010"; Di[2] = "00000010";
25    wait(100, SC_NS);
26    Di[0] = "00000000"; Di[1] = "00000010"; Di[2] = "00000000";
27    wait(100, SC_NS);
28    Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000";
29    busy = SC_LOGIC_0;
30
Line 3, Column 1: Detect Indentation: Setting Indentation to tabs with width 8 Tab Size: 8 C+++

```

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

```

29         busy = SC_LOGIC_0;
30
31     // Preloading Weights
32     start = SC_LOGIC_1;
33     wait(100, SC_NS);
34     Wi[0] = "00000001"; Wi[1] = "00000001"; Wi[2] = "00000001"
35     wait(100, SC_NS);
36     Wi[0] = "00000001"; Wi[1] = "00000001"; Wi[2] = "00000001"
37     wait(100, SC_NS);
38     Wi[0] = "00000001"; Wi[1] = "00000001"; Wi[2] = "00000001"
39     wait(100, SC_NS);
40     Wi[0] = "00000001"; Wi[1] = "00000001"; Wi[2] = "00000001"
41     start = SC_LOGIC_0;
42     wait(500, SC_NS);
43
44     // loading Input Matrix
45     Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000";
46     busy = SC_LOGIC_1;
47     wait(250, SC_NS);
48     Di[0] = "00000010"; Di[1] = "00000000"; Di[2] = "00000000"
49     wait(100, SC_NS);
50     Di[0] = "00000010"; Di[1] = "00000001"; Di[2] = "00000000"
51     wait(100, SC_NS);
52     Di[0] = "00000001"; Di[1] = "00000000"; Di[2] = "00000000"
53     wait(100, SC_NS);
54     Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000"
55     wait(100, SC_NS);
56     Di[0] = "00000000"; Di[1] = "00000000"; Di[2] = "00000000"
57     wait(100, SC_NS);
58     busy = SC_LOGIC_0;
59 }

```

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

Input Matrix is equal to *Systolic Array*:

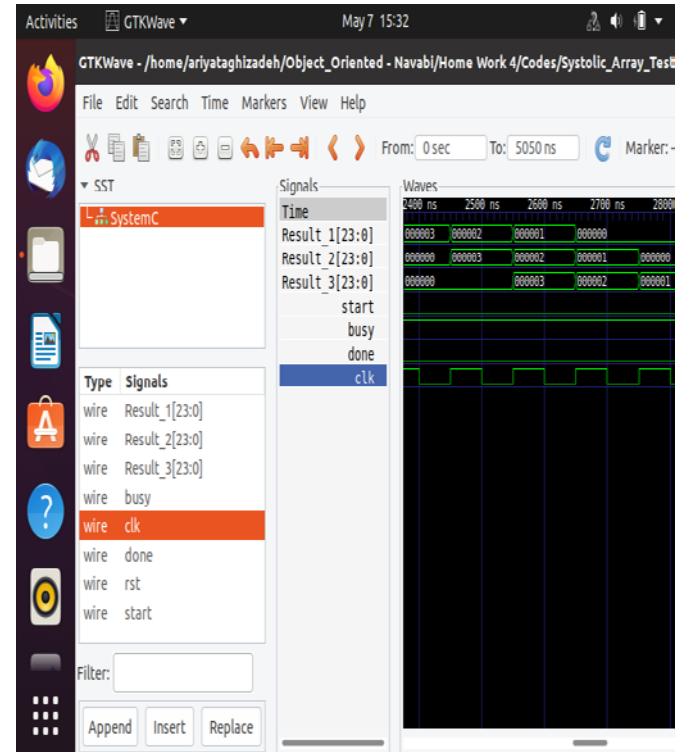


Fig. 37 Systolic Array's Test Bench Result SystemC (Equal)

Systolic Array's Test Bench Results:

The input weigh matrix is:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

The input data matrix is:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{matrix}$$

So the output matrix must be:

$$\begin{matrix} 3 & 2 & 1 \\ 3 & 2 & 1 \\ 3 & 2 & 1 \end{matrix}$$

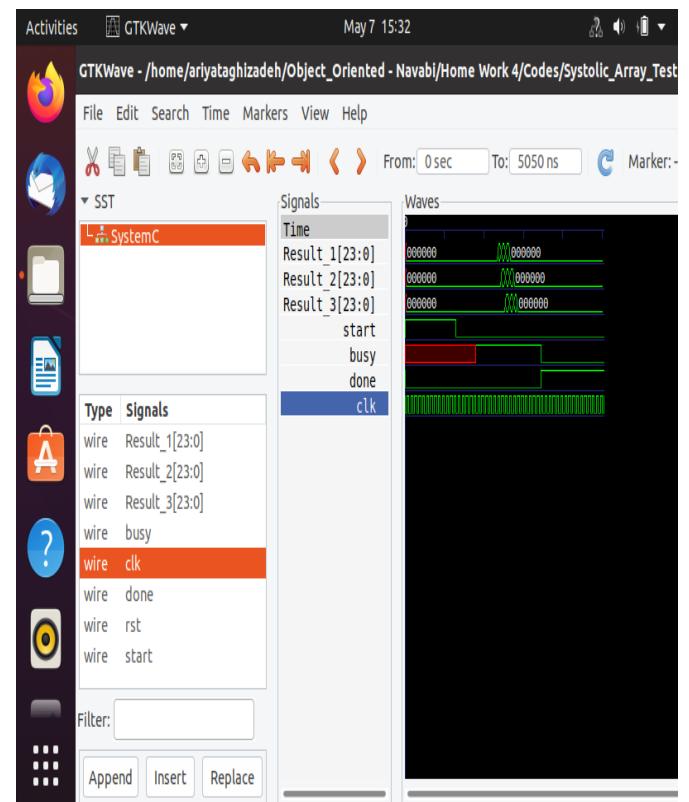


Fig. 38 Systolic Array's Test Bench Result SystemC (Equal)

Input Matrix is smaller than *Systolic Array*:

The input weigh matrix is:

$$\begin{matrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{matrix}$$

The input data matrix is:

$$\begin{matrix} 1 & 1 \\ 1 & 0 \end{matrix}$$

So the output matrix must be:

$$\begin{matrix} 2 & 1 \\ 2 & 1 \end{matrix}$$

Input Matrix is larger than *Systolic Array*:

The input weigh matrix is:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

The input data matrix is:

$$\begin{matrix} 2 & 2 & 2 & 2 & 2 & 1 \\ 2 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 \end{matrix}$$

So the output matrix must be:

$$\begin{matrix} 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 5 & 4 & 3 & 2 & 1 \end{matrix}$$

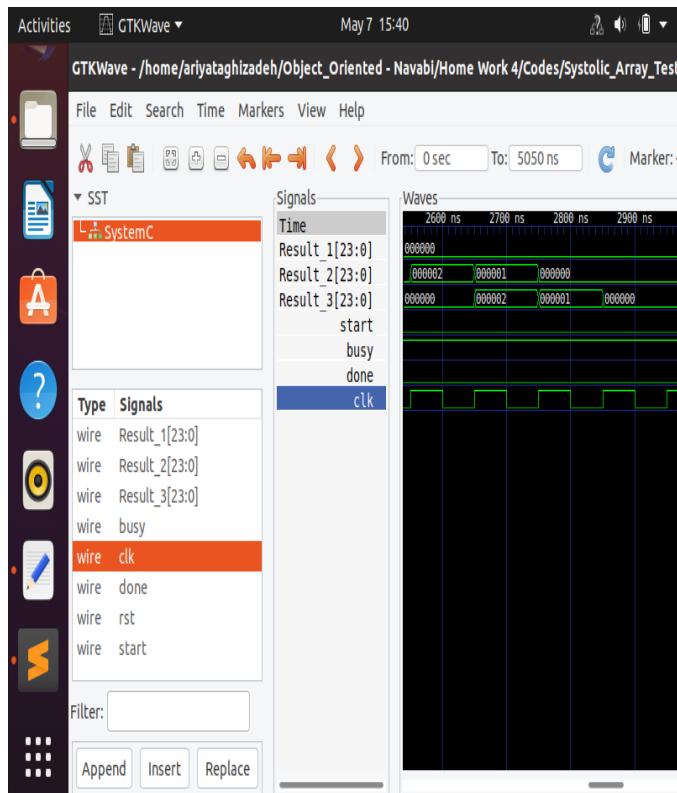


Fig. 39 Systolic Array's Test Bench *Result SystemC* (Smaller)

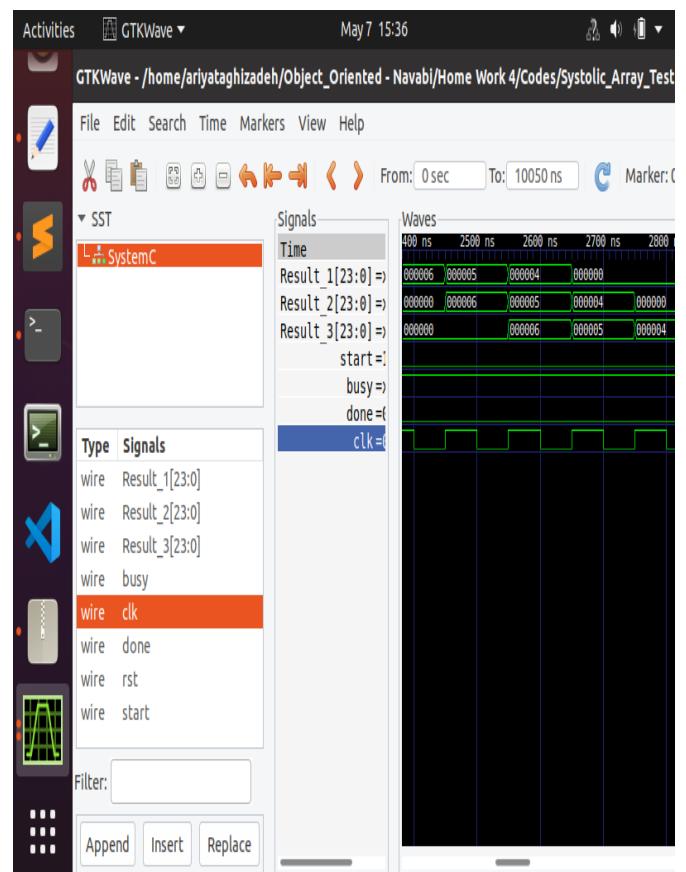


Fig. 40 Systolic Array's Test Bench *Result SystemC* (Larger)

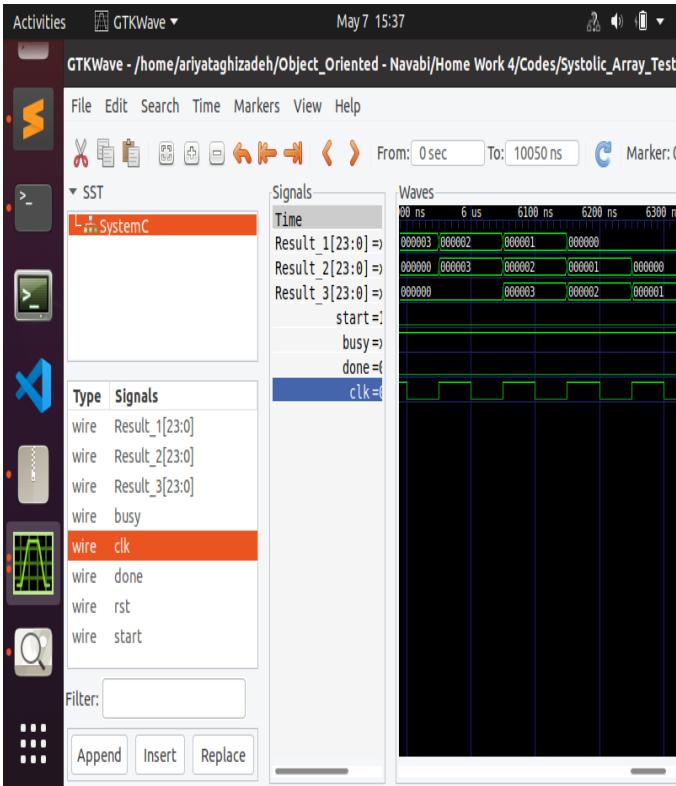


Fig. 41 Systolic Array's Test Bench Result SystemC (Larger)

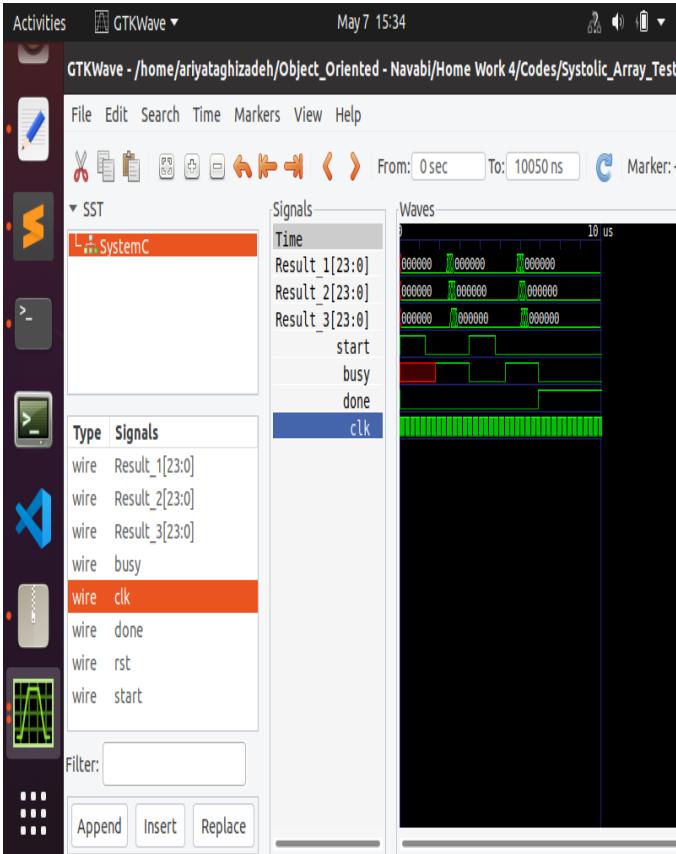


Fig. 42 Systolic Array's Test Bench Result SystemC (Larger)

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:

```

1 #include <systemc.h>
2
3 SC_MODULE(PE){
4
5     // Port Declaration
6     sc_in<sc_logic> clk, rst;
7     sc_in<sc_logic> startPE, busyPE;
8     sc_in<sc_lv<8>> Di, Wi;
9     sc_in<sc_lv<24>> Si;
10    sc_out<sc_lv<24>> Result;
11    sc_out<sc_lv<8>> Dout, Wout;
12
13    SC_CTOR(PE){
14        SC_THREAD(operation)
15            sensitive << clk << rst;
16    }
17
18    void operation();
19 };
20
21 void PE::operation()
22 {
23     while (true)
24     {
25         if (rst == '1')
26     }
}

```

Fig. 43 PE's Bus Functional Model SystemC

```

20
21 void PE::operation()
22 {
23     while (true)
24     {
25         if (rst == '1')
26         {
27             Result = 0;
28             Dout = 0;
29             Wout = 0;
30         }
31         else if ((clk == '1') && (clk->event()))
32         {
33             if (startPE == '1')
34                 Wout = Wi;
35
36             if(busyPE == '1')
37             {
38                 Dout = Di;
39                 Result = (Di->read().to_uint() * Wi-
>read().to_uint()) + Si->read().to_uint();
39
40             }
41         }
42         wait();
43     }
44 }

```

C/Object Header ▾ Tab Width: 8 ▾ Ln3, Col3 ▾ INS

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:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

The input data matrix is:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{matrix}$$

So the output matrix must be:

$$\begin{matrix} 3 & 2 & 1 \\ 3 & 2 & 1 \\ 3 & 2 & 1 \end{matrix}$$

Input Matrix is equal to Systolic Array:

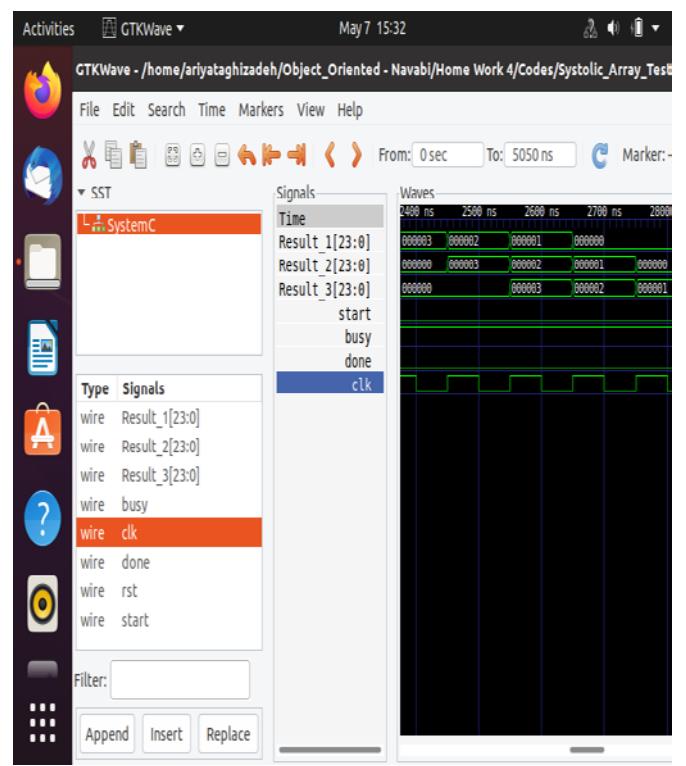


Fig. 45 Systolic Array's BFM Test Bench Result SystemC (Equal)

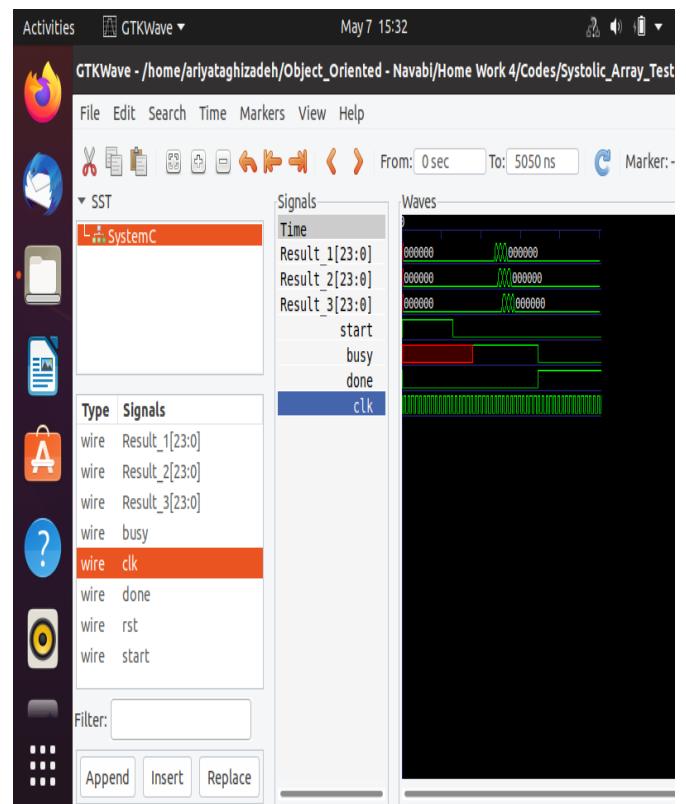


Fig. 46 Systolic Array's BFM Test Bench Result SystemC (Equal)

`Input Matrix is smaller than *Systolic Array*:

The input weigh matrix is:

$$\begin{matrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{matrix}$$

The input data matrix is:

$$\begin{matrix} 1 & 1 \\ 1 & 0 \end{matrix}$$

So the output matrix must be:

$$\begin{matrix} 2 & 1 \\ 2 & 1 \end{matrix}$$

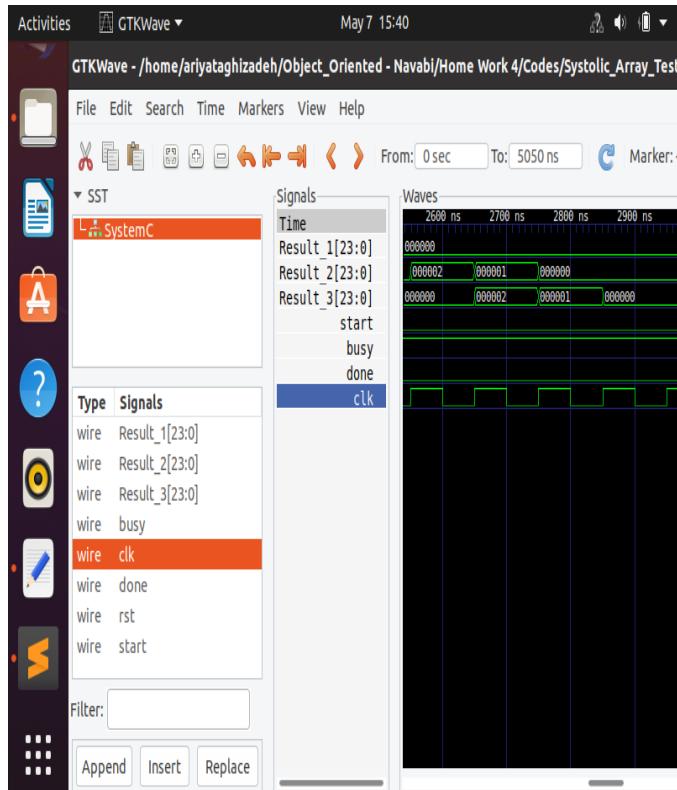


Fig. 47 Systolic Array's BFM Test Bench *Result SystemC* (Smaller)

Input Matrix is larger than *Systolic Array*:

The input weigh matrix is:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

The input data matrix is:

$$\begin{matrix} 2 & 2 & 2 & 2 & 2 & 1 \\ 2 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 \end{matrix}$$

So the output matrix must be:

$$\begin{matrix} 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 5 & 4 & 3 & 2 & 1 \end{matrix}$$

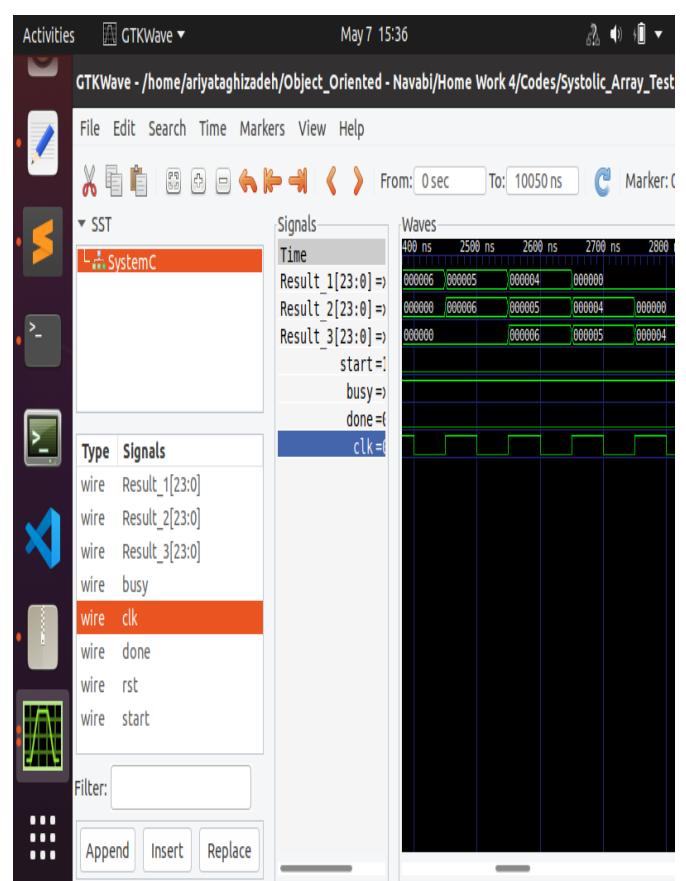


Fig. 48 Systolic Array's Test Bench *Result SystemC* (Larger)

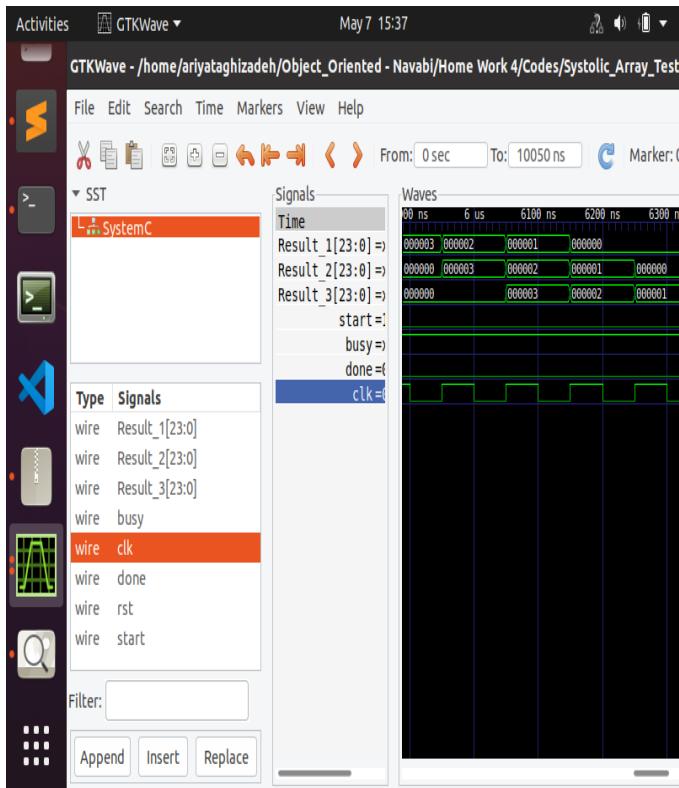


Fig. 49 Systolic Array's BFM Test Bench Result SystemC (Larger)

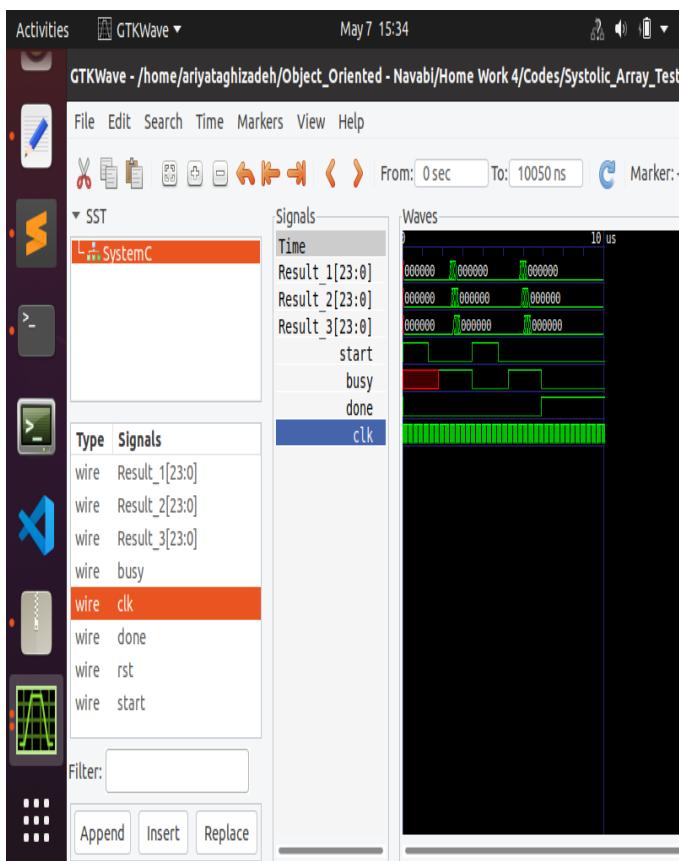


Fig. 50 Systolic Array's BFM Test Bench Result SystemC (Larger)