

# Homework 5 – SystemC High-Level Modeling and Communication Modeling

Taghizadeh,  
810198373

## I. INTRODUCTION

In this homework, we are about to make a full digital system containing processor, data memory and some peripherals like an intermediate component and 16-bit multiplier which are connected via a FIFO queue channel.

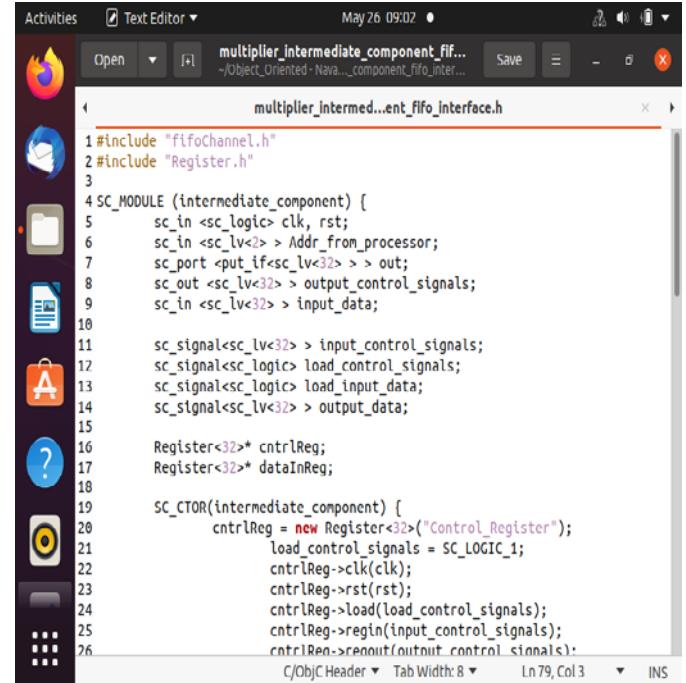
The processor is going to execute the following program:

```
Write 128 32-bit data to data memory  
Read 128 32-bit data from data memory  
Loop 4 times:  
    if (FIFO queue channel is empty) {  
        Write a 32 32-bit data into the  
            intermediate component  
        issue start of multiplier  
        do something else  
    Loop 32 times:  
        If ( mult result is ready)  
            Read mult result  
        Else  
            do something else  
    end  
}  
else  
    do something else  
end
```

## II. DIGITAL SYSTEM

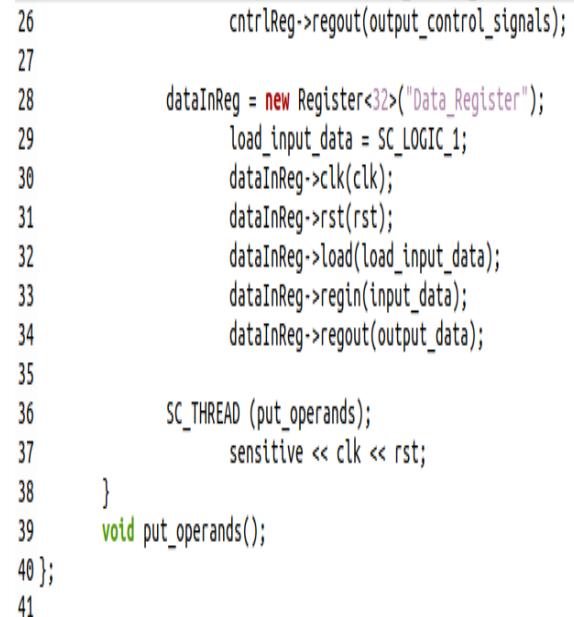
### A. Intermediate Component in SystemC

The intermediate component consists of two registers, one is control register and the other is data register. Input output data and control signals are declared as in the front image:



```
Activities Text Editor May 26 09:02 multiplier_intermediate_component_fifo... Save X  
multiplier_intermediate_component_fifo...ent_fifo_interface.h  
1 #include "fifoChannel.h"  
2 #include "Register.h"  
3  
4 SC_MODULE (intermediate_component) {  
5     sc_in <sc_lv<32> > Addr_from_processor;  
6     sc_port <put_if<sc_lv<32>> > out;  
7     sc_in <sc_lv<32> > output_control_signals;  
8     sc_in <sc_lv<32> > input_data;  
9  
10    sc_signal<sc_lv<32> > input_control_signals;  
11    sc_signal<sc_logic> load_control_signals;  
12    sc_signal<sc_logic> load_input_data;  
13    sc_signal<sc_lv<32> > output_data;  
14  
15    Register<32>* cntrlReg;  
16    Register<32>* dataInReg;  
17  
18    SC_CTOR(intermediate_component) {  
19        cntrlReg = new Register<32>("Control_Register");  
20        load_control_signals = SC_LOGIC_1;  
21        cntrlReg->clk(clk);  
22        cntrlReg->rst(rst);  
23        cntrlReg->load(load_control_signals);  
24        cntrlReg->regin(input_control_signals);  
25        cntrlReg->regout(output_control_signals);  
26    }  
27  
28    dataInReg = new Register<32>("Data_Register");  
29    load_input_data = SC_LOGIC_1;  
30    dataInReg->clk(clk);  
31    dataInReg->rst(rst);  
32    dataInReg->load(load_input_data);  
33    dataInReg->regin(input_data);  
34    dataInReg->regout(output_data);  
35  
36    SC_THREAD (put_operands);  
37        sensitive << clk << rst;  
38    }  
39    void put_operands();  
40};  
41
```

Fig. 1 Intermediate component module SystemC



```
26        cntrlReg->regout(output_control_signals);  
27  
28    dataInReg = new Register<32>("Data_Register");  
29    load_input_data = SC_LOGIC_1;  
30    dataInReg->clk(clk);  
31    dataInReg->rst(rst);  
32    dataInReg->load(load_input_data);  
33    dataInReg->regin(input_data);  
34    dataInReg->regout(output_data);  
35  
36    SC_THREAD (put_operands);  
37        sensitive << clk << rst;  
38    }  
39    void put_operands();  
40};  
41
```

Fig. 2 Intermediate component module SystemC

“*input\_data*” is defined as input signal to receive the incoming data from the processor.

“*output\_control\_signals*” is defined as output signal to inform the processor that FIFO is empty or not by the first bit of “*output\_control\_signals*”.

“*Addr\_from\_processor*” is for communication between this module and the processor. Whenever processor wants to communicate with this module, it initialises “*Addr\_from\_processor*” to “01” (which is intermediate component’s identifier).

“*out*” is put interface which is used to send incoming data to the queue channel.

“*put\_operands*” method is used to put the incoming data from the processor into the queue channel to store data in FIFO.

“*put\_operands*” method is sensitive to clock. So each posedge of clock this method is called. At first we wait until posedge of clock occurs by “*wait()*”. If “*Addr\_from\_processor*” is equal to “01” then we realise that processor wants to send data to *intermediate component*. First we check if FIFO is empty. Then we must inform the processor that FIFO is empty by initialising the first bit of “*input\_control\_signals*”(which is registered with “*output\_control\_signals*”). So processor will send its data. In the end we put the incoming data into FIFO about 32 times by a loop. Each time data is put into the FIFO, we must wait about a clock (which has 100 NS period) to put data clock by clock.

```

1 #include "multiplier_intermediate_component_fifo_interface.h"
2
3 void intermediate_component::put_operands() {
4     while(true){
5         wait();
6         if(Addr_from_processor.read() == "01"){
7             if(out->is_fifo_empty()){
8                 input_control_signals = "00000000000000000000000000000001"
9                 wait(300, SC_NS);
10                for (int i = 0; i < 32; i++){
11                    {
12                        wait(100, SC_NS);
13                        out->put(output_data);
14                        cout << "Data: " << output_data << " was transmitted
15                        << sc_time_stamp() << "\n";
16                    }
17                }
18            }
19            input_control_signals = "00000000000000000000000000000000"
20        }
21    }
22 }
23 }
```

Fig. 3 Intermediate component *put\_operands* method

## B. 16-bit Multiplier in SystemC

The 16-bit multiplier consists of two registers, one is control register and the other is data register. Input output data and control signals are declared as in the below image:

```

Activities Text Editor May 26 09:03
multiplier_intermediate_component_fifo...
~/Object_Oriented_Nav..._component_fifo_inter...
Save E X
multiplier_intermediate_fifo_interface.h
41
42 SC_MODULE (multiplier_16_bit) {
43     sc_in <sc_lv<2> Addr_from_processor;
44     sc_in <sc_lv<32> > in;
45     sc_out <sc_lv<32> > output_data;
46     sc_out <sc_lv<32> > output_ctrl;
47     sc_out <sc_lv<32> > output_ctrl;
48
49     sc_signal<sc_lv<32> > control_signals;
50     sc_signal<sc_logic> load_ctrl;
51     sc_signal<sc_logic> load_output_data;
52     sc_signal<sc_lv<32> > mult_result;
53
54     Register<32>* ctrlReg;
55     Register<32>* dataOutReg;
56
57     SC_CTOR(multiplier_16_bit) {
58         ctrlReg = new Register<32>("Control_Register");
59         load_ctrl = SC_LOGIC_1;
60         ctrlReg->clk(clk);
61         ctrlReg->rst(rst);
62         ctrlReg->load(load_ctrl);
63         ctrlReg->regin(control_signals);
64         ctrlReg->regout(output_ctrl);
65     }
66 }
```

Fig. 4 16-bit Multiplier module SystemC

```

Activities Text Editor May 26 09:03
multiplier_intermediate_component_fifo...
~/Object_Oriented_Nav..._component_fifo_inter...
Save E X
multiplier_intermediate_fifo_interface.h
54
55     Register<32>* ctrlReg;
56     Register<32>* dataOutReg;
57
58     SC_CTOR(multiplier_16_bit) {
59         ctrlReg = new Register<32>("Control_Register");
60         load_ctrl = SC_LOGIC_1;
61         ctrlReg->clk(clk);
62         ctrlReg->rst(rst);
63         ctrlReg->load(load_ctrl);
64         ctrlReg->regin(control_signals);
65         ctrlReg->regout(output_ctrl);
66
67         dataOutReg = new Register<32>("Data_Register");
68         load_output_data = SC_LOGIC_1;
69         dataOutReg->clk(clk);
70         dataOutReg->rst(rst);
71         dataOutReg->load(load_output_data);
72         dataOutReg->regin(mult_result);
73         dataOutReg->regout(output_data);
74
75         SC_THREAD (get_operands_and_evaluate);
76             sensitive << clk << rst;
77     }
78     void get_operands_and_evaluate();
79 }
```

Fig. 5 16-bit Multiplier module SystemC

“*output\_data*” is defined as output signal to send the multiplication result to the processor.

“*input\_control\_signals*” is defined as input signal to inform the multiplier to start the multiplication by the “*start*” bit (which is the first bit of “*input\_control\_signals*”) which is initialised by the processor.

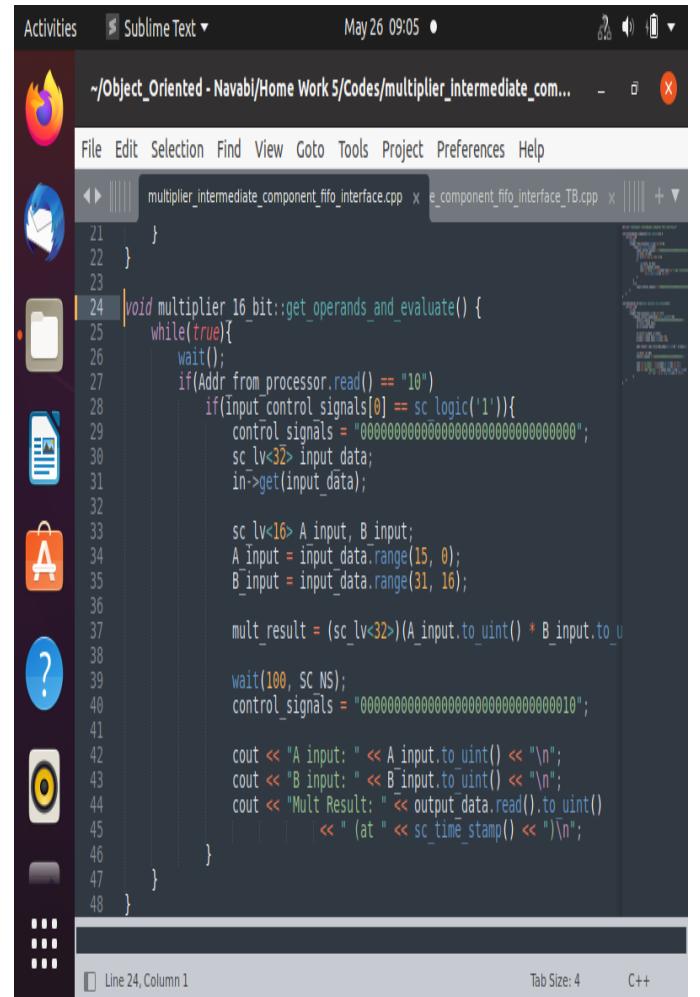
“*output\_control\_signals*” is defined as output signal to inform the processor that multiplication result is ready by the “*ready*” bit (which is the second bit of “*output\_control\_signals*”) which is initialised by the multiplier.

“*Addr\_from\_processor*” is for communication between this module and the processor. Whenever processor wants to communicate with this module, it initialises “*Addr\_from\_processor*” to “10” (which is 16-bit multiplier’s identifier).

“*in*” is get interface which is used to get the operands from FIFO queue channel to calculate the multiplication.

“*get\_operand\_and\_evaluate*” method is used to get operands from FIFO and evaluate the multiplication of the operands.

“*get\_operand\_and\_evaluate*” method is sensitive to clock. So each posedge of clock this method is called. At first we wait until posedge of clock occurs by “*wait()*”. If “*Addr\_from\_processor*” is equal to “10” then we realise that processor wants to communicate to *16-bit multiplier*. First we check if start is issued by the processor (which is the first bit of “*input\_control\_signals*”) after that we get the operands by “*get*” method which belongs to “*in*” port. By “*range*” method, data from FIFO is split into two 16 bits data, which are two operands of multiplication. In the end two operands are multiplied and stored in “*mult\_result*” (which is registered with “*output\_data*”) to send multiplication result to the processor. At least 0 SC\_NS must be passed to see updated value of “*mult\_result*”, so we wait (100 NS) to store the updated value of “*mult\_result*” into “*output\_data*”. To inform the processor that the multiplication result is ready, the “*ready*” bit (which is the second bit of “*control\_signals*”, which is registered with “*output\_control\_signals*”) is initialised.



```

Activities Sublime Text May 26 09:05
~/Object_Oriented - Navabi/Home Work 5/Codes/multiplier_intermediate_com...
File Edit Selection Find View Goto Tools Project Preferences Help
multiplier_intermediate_component_fifo_interface.cpp x multiplier_intermediate_TB.cpp x
21 }
22 }
23
24 void multiplier_16_bit::get_operands_and_evaluate() {
25     while(true){
26         wait();
27         if(Addr_from_processor.read() == "10")
28             if(Input_control_signals[0] == sc_logic('1')){
29                 control_signals = "00000000000000000000000000000000";
30                 sc_lv<32> input_data;
31                 in->get(input_data);
32
33                 sc_lv<16> A_input, B_input;
34                 A_input = input_data.range(15, 0);
35                 B_input = input_data.range(31, 16);
36
37                 mult_result = (sc_lv<32>)(A_input.to_uint() * B_input.to_u
38
39                 wait(100, SC_NS);
40                 control_signals = "00000000000000000000000000000010";
41
42                 cout << "A input: " << A_input.to_uint() << "\n";
43                 cout << "B input: " << B_input.to_uint() << "\n";
44                 cout << "Mult Result: " << output_data.read().to_uint();
45                 cout << " (at " << sc_time_stamp() << ")\n";
46             }
47     }
48 }

```

Line 24, Column 1 Tab Size: 4 C++

Fig. 6 16-bit Multiplier *get\_operands\_and\_evaluate* meyjod

### C. FIFO Queue channel in SystemC

We simply used "*FIFO put get interface queue channel*" provided in Channel\_codes.zip. The "*is\_fifo\_empty*" method is added, which simply counts number of elements stored in FIFO to check if FIFO is empty or not.

```
1 #include "interfaceClasses.h"
2
3 template <class T, int Max>
4 class fifo : public put_if<T>, public get_if<T>
5 {
6     int size;
7     int elems, head;
8     T queueContents[Max];
9     sc_event put_event, get_event;
10
11     public:
12         fifo() : size(Max), elems(0), head(0) {};
13         ~fifo() {};
14         void put(T data){
15             if (elems == size) wait(get_event);
16             queueContents[(head + elems) % size] = data;
17             elems = elems + 1;
18             put_event.notify();
19         }
20         void get(T &data){
21             if (elems == 0) wait(put_event);
22             data = queueContents[head];
23             elems = elems - 1;
24             head = (head + 1) % size;
25             get_event.notify();
26         }
27         bool is_fifo_empty(){
28             if (elems > 0)
29                 return false;
30             else
31                 return true;
32         }
33 }
```

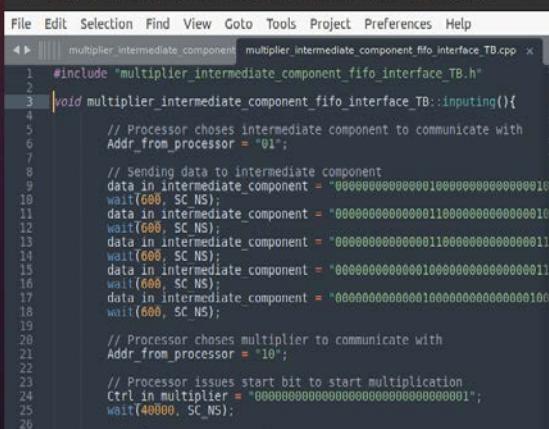
Fig. 7 FIFO Queue channel *module SystemC*

```
Activities Text Editor May 26 09:04
fifoChannel.h Save
-/Object_Oriented - Navabi/Home Work 5/Code...p...
1 #include <queue>
2
3 class fifoChannel {
4 public:
5     int size;
6     int elems, head;
7     T queueContents[Max];
8     sc_event put_event, get_event;
9
10    fifoChannel() : size(Max), elems(0), head(0) {};
11    ~fifoChannel() {};
12    void put(T data){
13        if (elems == size) wait(get_event);
14        queueContents[(head + elems) % size] = data;
15        elems = elems + 1;
16        put_event.notify();
17    }
18    void get(T &data){
19        if (elems == 0) wait(put_event);
20        data = queueContents[head];
21        elems = elems - 1;
22        head = (head + 1) % size;
23        get_event.notify();
24    }
25    bool is_fifo_empty(){
26        if(elems > 0)
27            return false;
28        return true;
29    }
30
31 }
32 }
```

Fig. 8 FIFO Queue channel *module SystemC*

#### D. Intermediate component, Multiplier and FIFO Test Bench

Simply instances of Intermediate component, 16-bit multiplier and FIFO queue channel are taken. *Intermediate component* must be attached to "out" (which is put interface of queue FIFO channel), to put processor's data into FIFO, whereas *16-bit Multiplier* must be attached to "in" (which is get interface of queue FIFO channel) to get data(operands) from FIFO. "inputting", "clocking" and "resetting" methods are used to generate test case, generate clock signal and reset respectfully.



The screenshot shows the Sublime Text editor interface with the following details:

- Title Bar:** Activities, Sublime Text, May 26 09:05
- File Path:** ~/Object\_Oriented - Navabi/Home Work 5/Codes/multiplier\_intermediate\_com...
- Toolbar:** File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, Help
- Code Content:** A C++ code snippet for a multiplier intermediate component. The code includes comments explaining the communication process between the processor and the multiplier component, involving FIFO interfaces and start bits.

```
#include "multiplier_intermediate_component_fifo_interface_TB.h"
void multiplier_intermediate_component_fifo_interface_TB::inputing(){
    // Processor chooses intermediate component to communicate with
    Addr_from_processor = "01";
    // Sending data to intermediate component
    data_in_intermediate_component = "0000000000000000000000000000000010";
    wait(600, SC_NS);
    data_in_intermediate_component = "0000000000000000000000000000000010";
    wait(600, SC_NS);
    data_in_intermediate_component = "0000000000000000000000000000000011";
    wait(600, SC_NS);
    data_in_intermediate_component = "0000000000000000000000000000000011";
    wait(600, SC_NS);
    data_in_intermediate_component = "0000000000000000000000000000000010";
    wait(600, SC_NS);
    data_in_intermediate_component = "0000000000000000000000000000000010";
    // Processor chooses multiplier to communicate with
    Addr_from_processor = "10";
    // Processor issues start_bit to start multiplication
    Ctrl_in_multiplier = "000000000000000000000000000000000000000000000001";
    wait(40000, SC_NS);
    Addr_from_processor = "00";
}
```

Fig. 9 Intermediate, Multiplier and FIFO *Test Bench inputting method*

```
File Edit Selection Find View Goto Tools Project Preferences Help
multiplier_intermediate_component_fifo_interface_TB.cpp x
20     // Processor chooses multiplier to communicate with
21     Addr_from_processor = "16";
22
23     // Processor issues start bit to start multiplication
24     Ctrl_in_multiplier = "00000000000000000000000000000001";
25     wait(40000, SC_NS);
26
27     Addr_from_processor = "00";
28 }
29
30 void multiplier_intermediate_component_fifo_interface_TB::clocking()
31 {
32     clk = sc_logic('1');
33     for (int i = 0; i <= 400; i++)
34     {
35         clk = sc_logic('0');
36         wait(50, SC_NS);
37         clk = sc_logic('1');
38         wait(50, SC_NS);
39     }
40
41 void multiplier_intermediate_component_fifo_interface_TB::reseting()
42 {
43     rst = (sc_logic)'0';
44     wait(5, SC_NS);
45     rst = (sc_logic)'1';
46     wait(5, SC_NS);
47     rst = (sc_logic)'0';
48 }
```

Fig. 10 Intermediate, Multiplier and FIFO Test Bench clocking resetting method

```

aryataghizadeh@ariyataghizadeh-VirtualBox:~/Object_Oriented - Navabi/Home Work S/Codes/multiplier_intermediate_component_fifo_interface$ ./a.out

SystemC 2.3.1-Accellera ... Apr 29 2023 04:07:43
Copyright (c) 1996-2014 by all Contributors,
ALL RIGHTS RESERVED

Data: (00000000000000000000000000000010) was transmitted at: 400 ns
Data: (00000000000000000000000000000010) was transmitted at: 500 ns
Data: (00000000000000000000000000000010) was transmitted at: 600 ns
Data: (00000000000000000000000000000010) was transmitted at: 700 ns
Data: (00000000000000000000000000000010) was transmitted at: 800 ns
Data: (00000000000000000000000000000010) was transmitted at: 900 ns
Data: (00000000000000000000000000000010) was transmitted at: 1 us
Data: (00000000000000000000000000000010) was transmitted at: 1000 ns
Data: (00000000000000000000000000000010) was transmitted at: 1200 ns
Data: (00000000000000000000000000000011) was transmitted at: 1300 ns
Data: (00000000000000000000000000000011) was transmitted at: 1400 ns
Data: (00000000000000000000000000000011) was transmitted at: 1500 ns
Data: (00000000000000000000000000000011) was transmitted at: 1600 ns
Data: (00000000000000000000000000000011) was transmitted at: 1700 ns
Data: (00000000000000000000000000000011) was transmitted at: 1800 ns
Data: (00000000000000000000000000000011) was transmitted at: 1900 ns
Data: (00000000000000000000000000000011) was transmitted at: 2 us
Data: (00000000000000000000000000000011) was transmitted at: 2100 ns
Data: (00000000000000000000000000000011) was transmitted at: 2200 ns
Data: (00000000000000000000000000000011) was transmitted at: 2300 ns
Data: (00000000000000000000000000000011) was transmitted at: 2400 ns
Data: (00000000000000000000000000000010) was transmitted at: 2500 ns
Data: (00000000000000000000000000000010) was transmitted at: 2600 ns

```

Fig. 11 Intermediate Component Multiplier FIFO interface Test Bench results

```

aryataghizadeh@ariyataghizadeh-VirtualBox:~/Object_Oriented - Navabi/Home Work S/Codes/multiplier_intermediate_component_fifo_interface$ ./a.out

A input: 4
B input: 4
Mult Result: 16 (at 5400 ns)
A input: 4
B input: 4
Mult Result: 16 (at 5500 ns)
?
```

Fig. 13 Intermediate component Multiplier FIFO interface Test Bench results

```

aryataghizadeh@ariyataghizadeh-VirtualBox:~/Object_Oriented - Navabi/Home Work S/Codes/multiplier_intermediate_component_fifo_interface$ ./a.out

A input: 2
B input: 3
Mult Result: 6 (at 3600 ns)
A input: 2
B input: 3
Mult Result: 6 (at 3700 ns)
A input: 2
B input: 3
Mult Result: 6 (at 3800 ns)
A input: 2
B input: 3
Mult Result: 6 (at 3900 ns)
A input: 3
B input: 3
Mult Result: 9 (at 4 us)
A input: 3
B input: 3
Mult Result: 9 (at 4100 ns)
A input: 3
B input: 3
Mult Result: 9 (at 4200 ns)
A input: 3
B input: 3
Mult Result: 9 (at 4300 ns)
A input: 3
B input: 3
Mult Result: 9 (at 4400 ns)
A input: 3
B input: 3

```

Fig. 12 Intermediate component Multiplier FIFO interface Test Bench results

```

1 #include "memoryAccessChannel.h"
2
3 SC_MODULE(memory) {
4
5     sc_in<sc_logic> clk, rst, Ctrl;
6     sc_in<sc_lv<2>> Addr;
7     sc_in<sc_lv<32>> input_address;
8     sc_in<sc_lv<32>> input_data;
9     sc_out<sc_lv<32>> output_data;
10
11    sc_lv<32>* mem;
12
13    sc_export<MemRespond_if<sc_lv<32>>> in;
14    memoryAccess<sc_lv<32>>* memBus;
15
16    SC_CTOR(memory) {
17        mem = new sc_lv<32>[256];
18        for (int i = 0; i < 256; i++) mem[i] = (sc_lv<32>) i;
19
20        memBus = new memoryAccess<sc_lv<32>>;
21        SC_IHREAD(requesting);
22            sensitive << clk << rst;
23        SC_THREAD(responding);
24            sensitive << clk << rst;
25    }
26
27    void requesting();
28    void responding();

```

Fig. 14 Memory module SystemC

“requesting” method is used to request to the memory to load/store data, which uses memBus interface’s “requestMem” method to request to memory. Whenever “Addr\_from\_processor” is equal to “00” (which is Memory’s identifier). Processor wants to send/receive data from memory so “requesting” method is called to inform the memory that processor has a request from memory.



The screenshot shows a software development environment with a toolbar on the left containing icons for file operations like Open, Save, and Close, as well as navigation and search functions. The main window displays a C++ header file named `memoryReadWrites.h`. The code defines a class `memory` with methods `responding`, `requesting`, and `responding`. The `responding` method handles memory reads and writes. The `requesting` method reads data from a controller and sends it to memory. The `responding` method sends data from memory to the controller. The code uses memory-mapped I/O with addresses and data sizes of 32 bits.

```
Activities Text Editor May 26 05:55
Open memoryReadWrites.h ->Object_Oriented - Navabi/Home Work 5/Codes/M...
Save
memoryReadWrites.h
void responding();
virtual ~memory(){
    delete[] mem;
}
};

void memory::requesting() {
    while(true){
        wait();
        if(Addr.read() == "00"){
            bool rwbar = (Ctrl.read() == '1') ? true : false;
            sc_lv<32> data = input_data;
            sc_lv<32> address = input_address;
            memBus->requestMem(data, address, rwbar);
        }
    }
}

void memory::responding() {
    while (true)
    {
        wait();
        if(Addr.read() == "00"){
            bool rwbar = (Ctrl.read() == '1') ? true : false;
            sc_lv<32> data = input_data;
            sc_lv<32> address = input_address;
            memBus->memForward(data, address, rwbar);
            if(rwbar)
        }
    }
}
```

Fig. 15 Memory requesting method

“*responding*” method is used to respond to processor. If “*rwbar*” (which is “*Ctrl*” input signal equivalent Boolean value that determines which of read or write is requested by processor) is “1” so processor wants to read from memory else processor wants to write some data in memory. The below images shows how read and write from memory is handled by “*responding*” method:

Activities Text Editor May 26 05:55

memoryReadWrites.h

```
41             memBus->requestMem(data, address, rwbar);  
42         }  
43     }  
44 }  
45  
46 void memory::responding() {  
47     while (true)  
48     {  
49         wait();  
50         if(Addr.read() == "00") {  
51             bool rwbar = (Ctrl.read() == '1') ? true : false;  
52             sc_lv<32> data = input_data;  
53             sc_lv<32> address = input_address;  
54             memBus->memForward(data, address, rwbar);  
55             if (rwbar)  
56                 output_data =  
mem[input_address.read().to_uint()];  
57             else mem[input_address.read().to_uint()] =  
input_data;  
58             memBus->memBackward(data);  
59             cout << "Memory " << (rwbar ? "READ" : "WROTE")  
60             << " Data:" << (rwbar ?  
output_data.read().to_uint() : input_data.read().to_uint()) << " Address:"  
<< input_address.read().to_uint()  
61             << " requested by Processor \n";  
62         }  
63     }  
64 }
```

Fig. 16 *Memory responding method*

Activities Text Editor May 26 05:55 •

memoryAccessChannel.h Save

```
Open - /Object_Oriented - Navalp/Home Work 5/Codey/M... X
```

```
1 #include "InterfaceClasses.h"
2
3 // data and address type, number of Initiators
4 template <class T>
5 class memoryAccess : public requestMem_if<T>, public memRespond_if<T>
6 {
7     T incomingData;
8     T incomingAddress;
9     T outgoingData;
10    bool read;
11    bool memoryRequested;
12    sc_event memoryCalledFor;
13    sc_event memoryCompleted;
14
15    sc_mutex busBusy;
16
17 public:
18     memoryAccess() : memoryRequested(false) {};
19     ~memoryAccess() {};
20
21     void requestMem(T &data, T &address, bool rwbar){
22         wait(0, SC_NS);
23         busBusy.lock();
24         if (!rwbar) incomingData = data;
25         incomingAddress = address;
26         read = rwbar;
27         memoryRequested = true;
28         memoryCalledFor.notify();
29     }
30 }
```

Fig. 17 Memory memroyAccess interface

```

18     memoryAccess() : memoryRequested(false) {};
19     ~memoryAccess() {};
20
21     void requestMem(T &data, T &address, bool rwbar){
22         wait(0, SC_NS);
23         busBusy.lock();
24         if (!rwbar) incomingData = data;
25         incomingAddress = address;
26         read = rwbar;
27         memoryRequested = true;
28         memoryCalledFor.notify();
29         wait(memoryCompleted);
30         busBusy.unlock();
31     }
32     void memForward(T &data, T &address, bool &rwbar){
33         if (!memoryRequested) wait(memoryCalledFor);
34         memoryRequested = false;
35         if (!read) data = incomingData;
36         address = incomingAddress;
37         rwbar = read;
38         // Ready for Backward operation
39     }
40     void memBackward(T &data){
41         outgoingData = data;
42         wait(0, SC_NS);
43         memoryCompleted.notify();
44     }
45 };

```

Fig. 18 Memory memoryAccess interface methods

#### F. Memory Test Bench

Simply instance of memory is taken. “*inputting*”, “*clocking*” and “*resetting*” methods are used to generate test case, generate clock signal and reset respectfully.

```

1 #include "memoryReadWrites.h"
2
3 SC_MODULE(memory_TB) {
4
5     sc_signal <sc_logic> clk, rst, rwbar;
6     sc_signal <sc_lv<2>> Addr_from_processor;
7     sc_signal <sc_lv<32>> input_data, output_data;
8     sc_signal <sc_lv<32>> input_address;
9     memory* MEM;
10
11     SC_CTOR(memory_TB) {
12
13         MEM = new memory("Memory");
14         MEM->in(*MEM->memBus);
15         MEM->clk(clk);
16         MEM->rst(rst);
17         MEM->ctrl(rwbar);
18         MEM->input_data(input_data);
19         MEM->output_data(output_data);
20         MEM->Addr(Addr_from_processor);
21         MEM->input_address(input_address);
22
23         SC_THREAD(inputting);
24         SC_THREAD(reseting);
25         SC_THREAD(clocking);
26     }
27     void inputting();
28     void resetting();
29
30 };

```

Fig. 20 Memory Test Bench

```

1 #include <systenc.h>
2
3 template <class T>
4 class requestMem_if : virtual public sc_interface
5 {
6 public:
7     virtual void requestMem(T &data, T &address, bool rwbar) = 0;
8
9 };
10
11 template <class T>
12 class memRespond_if : virtual public sc_interface
13 {
14 public:
15     virtual void memForward(T &data, T &address, bool &rwbar) = 0;
16     virtual void memBackward(T &data) = 0;
17 };

```

Fig. 19 Memory requestMem and memRespond interfaces

```

3 SC_MODULE(memory_TB) {
4
5     sc_signal <sc_logic> clk, rst, rwbar;
6     sc_signal <sc_lv<2>> Addr_from_processor;
7     sc_signal <sc_lv<32>> input_data, output_data;
8     sc_signal <sc_lv<32>> input_address;
9     memory* MEM;
10
11     SC_CTOR(memory_TB) {
12
13         MEM = new memory("Memory");
14         MEM->in(*MEM->memBus);
15         MEM->clk(clk);
16         MEM->rst(rst);
17         MEM->ctrl(rwbar);
18         MEM->input_data(input_data);
19         MEM->output_data(output_data);
20         MEM->Addr(Addr_from_processor);
21         MEM->input_address(input_address);
22
23         SC_THREAD(inputting);
24         SC_THREAD(reseting);
25         SC_THREAD(clocking);
26     }
27     void inputting();
28     void resetting();
29     void clocking();
30 };

```

Fig. 21 Memory Test Bench

In “*inputting*”, address (which we want to read from or write to) is increased in each iteration of a loop, so by the end of this loop all data are read/write from/to addresses 0000\_0008 till 0000\_00087. In this test bench memory input data starts from 0 and is increased in each iteration of a loop, so data will be from 0 to 127. we must wait about a clock (which has 100 NS period) to read/write data clock by clock and be synchronous.

Activities Sublime Text May 26 05:57

~/Object\_Oriented - Navabi/Home Work 5/Codes/Memory/Memory\_TB.cpp - Su...

File Edit Selection Find View Goto Tools Project Preferences Help

Memory\_TB.cpp x multiplier\_intermediate\_component\_mto\_interface.cpp x tem\_TB.cpp x + ▾

```
1 #include "Memory_TB.h"
2
3 void memory_TB::inputing(){
4
5     // Processor chooses memory to communicate with
6     Addr_from_processor = "00";
7
8     // Writing 128 data to memory start from address 0x00000000
9     rwbar = sc_logic('0');
10    input_address = "00000000000000000000000000000000";
11    input_data = "00000000000000000000000000000000";
12    for(int i = 0; i < 128; i++){
13        wait(100, SC_NS);
14        input_data = input_data.read().to_uint() + 1;
15        input_address = input_address.read().to_uint() + 1;
16    }
17
18    // Reading 128 data from memory start from address 0x00000000
19    rwbar = sc_logic('1');
20    input_address = "00000000000000000000000000000000";
21    input_data = "00000000000000000000000000000000";
22    for(int i = 0; i < 128; i++){
23        wait(100, SC_NS);
24        input_data = input_data.read().to_uint() + 1;
25        input_address = input_address.read().to_uint() + 1;
26    }
27
28    Addr_from_processor = "01";
}
```

Fig. 22 Memory Test Bench inputting method

Activities Sublime Text ▾ May 26 05:57

~/Object\_Oriented - Navabi/Home Work 5/Codes/Memory/Memory\_TB.cpp - Su...

File Edit Selection Find View Goto Tools Project Preferences Help

Memory\_TB.cpp x multiplier\_intermediate\_component\_fifo\_interface.cpp x item\_TB.cpp x +

```
22     for(int i = 0; i < 128; i++)
23         wait(100, SC_NS);
24         input_data = input_data.read().to_uint() + 1;
25         input_address = input_address.read().to_uint() + 1;
26     }
27
28     Addr from processor = "01";
29     wait(6000, SC_NS);
30 }
31
32 void memory_TB::clocking(){
33     clk = sc_logic('1');
34     for (int i = 0; i <= 600; i++)
35     {
36         clk = sc_logic('0');
37         wait(50, SC_NS);
38         clk = sc_logic('1');
39         wait(50, SC_NS);
40     }
41 }
42
43 void memory_TB::reseting(){
44     rst = (sc_logic)'0';
45     wait(5, SC_NS);
46     rst = (sc_logic)'1';
47     wait(5, SC_NS);
48     rst = (sc_logic)'0';
49 };
```

Fig. 23 *Memory Test Bench* clocking and resetting methods

Activities Terminal May 26 07:06

ariyataghizadeh@ariyataghizadeh-VirtualBox:~/Object\_Oriented - Navabi/Home Work 5/Codes/Memory\$ ./a.out

SystemC 2.3.1-Accellera ... Apr 29 2023 04:07:43  
Copyright (c) 1996-2014 by all Contributors,  
ALL RIGHTS RESERVED

Memory WROTE Data:0 Address:8 requested by Processor  
Memory WROTE Data:1 Address:9 requested by Processor  
Memory WROTE Data:1 Address:9 requested by Processor  
Memory WROTE Data:2 Address:10 requested by Processor  
Memory WROTE Data:2 Address:10 requested by Processor  
Memory WROTE Data:3 Address:11 requested by Processor  
Memory WROTE Data:3 Address:11 requested by Processor  
Memory WROTE Data:4 Address:12 requested by Processor  
Memory WROTE Data:4 Address:12 requested by Processor  
Memory WROTE Data:5 Address:13 requested by Processor  
Memory WROTE Data:5 Address:13 requested by Processor  
Memory WROTE Data:6 Address:14 requested by Processor  
Memory WROTE Data:6 Address:14 requested by Processor  
Memory WROTE Data:7 Address:15 requested by Processor  
Memory WROTE Data:7 Address:15 requested by Processor  
Memory WROTE Data:8 Address:16 requested by Processor  
Memory WROTE Data:8 Address:16 requested by Processor  
Memory WROTE Data:9 Address:17 requested by Processor  
Memory WROTE Data:9 Address:17 requested by Processor  
Memory WROTE Data:10 Address:18 requested by Processor

Fig. 24 *Memory Test Bench results*

```
Activities Terminal May 26 07:06
ariyataghizadeh@ariyataghizadeh-VirtualBox:~/Object_Ori...
Memory WROTE Data:113 Address:121 requested by Processor
Memory WROTE Data:113 Address:121 requested by Processor
Memory WROTE Data:114 Address:122 requested by Processor
Memory WROTE Data:114 Address:122 requested by Processor
Memory WROTE Data:115 Address:123 requested by Processor
Memory WROTE Data:115 Address:123 requested by Processor
Memory WROTE Data:116 Address:124 requested by Processor
Memory WROTE Data:116 Address:124 requested by Processor
Memory WROTE Data:117 Address:125 requested by Processor
Memory WROTE Data:117 Address:125 requested by Processor
Memory WROTE Data:118 Address:126 requested by Processor
Memory WROTE Data:118 Address:126 requested by Processor
Memory WROTE Data:119 Address:127 requested by Processor
Memory WROTE Data:119 Address:127 requested by Processor
Memory WROTE Data:120 Address:128 requested by Processor
Memory WROTE Data:120 Address:128 requested by Processor
Memory WROTE Data:121 Address:129 requested by Processor
Memory WROTE Data:121 Address:129 requested by Processor
Memory WROTE Data:122 Address:130 requested by Processor
Memory WROTE Data:122 Address:130 requested by Processor
Memory WROTE Data:123 Address:131 requested by Processor
Memory WROTE Data:123 Address:131 requested by Processor
Memory WROTE Data:124 Address:132 requested by Processor
Memory WROTE Data:124 Address:132 requested by Processor
Memory WROTE Data:125 Address:133 requested by Processor
Memory WROTE Data:125 Address:133 requested by Processor
Memory WROTE Data:126 Address:134 requested by Processor
Memory WROTE Data:126 Address:134 requested by Processor
Memory WROTE Data:127 Address:135 requested by Processor
```

Fig. 25 *Memory Test Bench results*

```

Activities Terminal May 26 07:06
ariyataghizadeh@ariyataghizadeh-VirtualBox: ~/Object_Ori...
Memory READ Data:0 Address:8 requested by Processor
Memory READ Data:0 Address:8 requested by Processor
Memory READ Data:1 Address:9 requested by Processor
Memory READ Data:1 Address:9 requested by Processor
Memory READ Data:2 Address:10 requested by Processor
Memory READ Data:2 Address:10 requested by Processor
Memory READ Data:3 Address:11 requested by Processor
Memory READ Data:3 Address:11 requested by Processor
Memory READ Data:4 Address:12 requested by Processor
Memory READ Data:4 Address:12 requested by Processor
Memory READ Data:5 Address:13 requested by Processor
Memory READ Data:5 Address:13 requested by Processor
Memory READ Data:6 Address:14 requested by Processor
Memory READ Data:6 Address:14 requested by Processor
Memory READ Data:7 Address:15 requested by Processor
Memory READ Data:7 Address:15 requested by Processor
Memory READ Data:8 Address:16 requested by Processor
Memory READ Data:8 Address:16 requested by Processor
Memory READ Data:9 Address:17 requested by Processor
Memory READ Data:9 Address:17 requested by Processor
Memory READ Data:10 Address:18 requested by Processor
Memory READ Data:10 Address:18 requested by Processor
Memory READ Data:11 Address:19 requested by Processor
Memory READ Data:11 Address:19 requested by Processor
Memory READ Data:12 Address:20 requested by Processor
Memory READ Data:12 Address:20 requested by Processor
Memory READ Data:13 Address:21 requested by Processor
Memory READ Data:13 Address:21 requested by Processor
Memory READ Data:14 Address:22 requested by Processor

```

Fig. 26 Memory Test Bench results

```

Activities Terminal May 26 07:06
ariyataghizadeh@ariyataghizadeh-VirtualBox: ~/Object_Ori...
Memory READ Data:114 Address:122 requested by Processor
Memory READ Data:115 Address:123 requested by Processor
Memory READ Data:115 Address:123 requested by Processor
Memory READ Data:116 Address:124 requested by Processor
Memory READ Data:116 Address:124 requested by Processor
Memory READ Data:117 Address:125 requested by Processor
Memory READ Data:117 Address:125 requested by Processor
Memory READ Data:118 Address:126 requested by Processor
Memory READ Data:118 Address:126 requested by Processor
Memory READ Data:119 Address:127 requested by Processor
Memory READ Data:119 Address:127 requested by Processor
Memory READ Data:120 Address:128 requested by Processor
Memory READ Data:120 Address:128 requested by Processor
Memory READ Data:121 Address:129 requested by Processor
Memory READ Data:121 Address:129 requested by Processor
Memory READ Data:122 Address:130 requested by Processor
Memory READ Data:122 Address:130 requested by Processor
Memory READ Data:123 Address:131 requested by Processor
Memory READ Data:123 Address:131 requested by Processor
Memory READ Data:124 Address:132 requested by Processor
Memory READ Data:124 Address:132 requested by Processor
Memory READ Data:125 Address:133 requested by Processor
Memory READ Data:125 Address:133 requested by Processor
Memory READ Data:126 Address:134 requested by Processor
Memory READ Data:126 Address:134 requested by Processor
Memory READ Data:127 Address:135 requested by Processor
Memory READ Data:127 Address:135 requested by Processor

```

Fig. 27 Memory Test Bench results

### G. Bus Interface

All components are in communicate with Bus interface, so all input/output of modules are declared in “Bus” as shown in the below images:

```

Activities Text Editor May 26 05:53
Bus.h
1 #include <systemc.h>
2
3 SC_MODULE(bus) {
4
5     sc_in <sc_lv<2> > Addr_from_processor;
6     sc_out <sc_lv<32> > input_data_processor;
7     sc_in <sc_lv<32> > output_data_processor;
8     sc_in <sc_lv<32> > output_address_processor;
9     sc_out <sc_lv<32> > input_control_signals_processor;
10    sc_in <sc_lv<32> > output_control_signals_processor;
11
12    sc_out <sc_lv<32> > input_address_mem;
13    sc_out <sc_lv<32> > input_data_mem;
14    sc_in <sc_lv<32> > output_data_mem;
15    sc_out <sc_logic> Ctrl_mem;
16
17    sc_in <sc_lv<32> > output_control_signals_intermediate_component;
18    sc_out <sc_lv<32> > input_data_intermediate_component;
19
20    sc_out <sc_lv<32> > input_control_signals_multiplier_16_bit;
21    sc_in <sc_lv<32> > output_control_signals_multiplier_16_bit;
22    sc_in <sc_lv<32> > output_data_multiplier_16_bit;
23
24 SC_CTOR(bus) {
25     SC_THREAD (address_logic);
26         sensitive << Addr_from_processor
27                         << output_data_processor
28                         << output_control_signals_processor;
29
30     }
31
32     void address_logic();
33
34 }
```

Fig. 28 Bus Interface module SystemC

```

Activities Text Editor May 26 05:53
Bus.h
28     << output_control_signals_processor
29     << output_address_processor
30     << output_data_mem
31     << output_control_signals_intermediate_component
32     << output_control_signals_multiplier_16_bit
33     << output_data_multiplier_16_bit;
34
35     void address_logic();
36 }
37
38 void bus::address_logic() {
39     while(true){
40         wait();
41         if(Addr_from_processor.read() == "00"){
42             Ctrl_mem = output_control_signals_processor.read();
43             [0];
44             input_address_mem = output_address_processor;
45             input_data_mem = output_data_processor;
46             input_data_processor = output_data_mem;
47         }
48         else if(Addr_from_processor.read() == "01"){
49             input_data_intermediate_component =
50             output_data_processor;
51             input_control_signals_processor =
52             output_control_signals_intermediate_component;
53         }
54         else if(Addr_from_processor.read() == "10"){
55             C/ObjC Header Tab Width: 8 Ln 5, Col 47 INS
56         }
57     }
58 }
```

Fig. 29 Bus Interface SystemC

“*address\_logic*” method is used to decode “*Addr\_from\_processor*” to identify which module does processor wants to communicate with (“00” means that processor wants to communicate with “*memory*”, “01” is for “*intermediate component*” and “10” is for “*16-bit Multiplier*”). After we realised that which module does processor wants to communicate, we simply connect data and control input/output signals of that module to processor data and control input/output signals. So first we identify which module does processor wants to communicate with, then we connect that module to the processor. So the processor and that module can send data to and handshaking with each other.

```

Activities Text Editor May 26 05:53 •
Bus.h -/Object_Oriented - Navabi/Home Work 5/Codes/Bus Save
35
36     void address_logic();
37 };
38
39 void bus::address_logic() {
40     while(true){
41         wait();
42         if(Addr_from_processor.read() == "00"){
43             Ctrl_mem = output_control_signals_processor.read()-[0];
44             input_address_mem = output_address_processor;
45             input_data_mem = output_data_processor;
46             input_data_processor = output_data_mem;
47         }
48         else if(Addr_from_processor.read() == "01"){
49             input_data_intermediate_component =
50                 output_data_processor;
51             output_control_signals_processor =
52                 output_control_signals_intermediate_component;
53         }
54         else if(Addr_from_processor.read() == "10"){
55             input_control_signals_multiplier_16_bit =
56                 output_control_signals_processor;
57             output_control_signals_processor =
58                 output_control_signals_multiplier_16_bit;
59         }
60     }
61 }

```

Fig. 30 Bus Interface *address\_logic* method

#### H. Processor

To store read data from memory an embedded Register File (depth = 128, word = 32 bit) is defined in processor. “*processor\_function*” method is the main program that processor is going to execute:

```

Activities Text Editor May 26 06:00 •
Processor.h -/Object_Oriented - Navabi/Home Work 5/Codes/P... Save
1 #include <systemc.h>
2
3 SC_MODULE(processor) {
4     sc_in <sc_logic> clk, rst;
5
6     sc_out <sc_lv<2>> Addr_from_processor;
7     sc_in <sc_lv<32>> input_data_processor;
8     sc_out <sc_lv<32>> output_data_processor;
9     sc_out <sc_lv<32>> output_address_processor;
10    sc_in <sc_lv<32>> input_control_signals_processor;
11    sc_out <sc_lv<32>> output_control_signals_processor;
12
13    sc_lv<32>* Register_File;
14
15    SC_CTOR(processor) {
16        Register_File = new sc_lv<32>[128];
17        SC_THREAD (processor_function);
18        sensitive << clk.pos();
19    }
20
21
22    void processor_function();
23}

```

Fig. 31 Processor module SystemC

“rwbar” (the first bit of “out\_put\_control\_signal\_processors”) must be ‘0’ for writing to memory and ‘1’ for reading from memory:

```

File Edit Selection Find View Goto Tools Project Preferences Help
Processor.cpp x Memory_TB.cpp x multiplier_intermediate_component_fifo_interface.cpp x cpp x + ▾
1 #include "Processor.h"
2
3 void processor::processor_function(){
4
5     wait();
6
7     // Processor chooses memory to communicate with
8     Addr_from_processor = "00";
9
10    // Writing 128 data to memory start from adress 0X00000008
11    output control_signals_processor = "00000000000000000000000000000000";
12    output address_processor = "000000000000000000000000000000001000";
13    output data_processor = "000000000000000000000000000000001100000000000000";
14
15    wait(0, SC_NS);
16
17    for(int i = 0; i < 128; i++){
18        wait(100, SC_NS);
19        output data_processor = output data_processor.read().to uint() + 1;
20        output address_processor = output address_processor.read().to uint() + 1;
21    }
22
23    wait(0, SC_NS);
24
25    // Reading 120 data from memory start from adress 0X00000000
26    output control_signals_processor = "00000000000000000000000000000001";
27    output address_processor = "000000000000000000000000000000001000";
28
29    wait(0, SC_NS);
30
31    for(int i = 0; i < 128; i++){
32        wait(100, SC_NS);
33        Register_file[output address_processor.read().to uint() - 0] = input data.proc

```

Line 71, Column 2      Spaces: 3      C++

Fig. 32 Processor processor\_function method

```

File Edit Selection Find View Goto Tools Project Preferences Help
Processor.cpp x Memory_TB.cpp x multiplier_intermediate_component_fifo_interface.cpp x cpp x + ▾
32    wait(100, SC_NS);
33    Register_file[output address_processor.read().to uint() - 0] = input data.proc
34    output address_processor = output address_processor.read().to uint() + 1;
35}
36
37 wait(0, SC_NS);
38
39 for(int i = 0; i < 4; i++){
40     // Processor chooses intermediate component to communicate with
41     Addr_from_processor = "01";
42
43     wait(200, SC_NS);
44
45     // Sending 32 32 bit data to intermediate component if fifo is empty
46     if(input control_signals_processor.read()[0] == '1'){
47         for(int j = 0; j < 33; j++){
48             wait(100, SC_NS);
49             output data_processor = Register_file[i*32 + j];
50         }
51
52     wait(0, SC_NS);
53
54     // Processor chooses multiplier to communicate with
55     Addr_from_processor = "10";
56
57     wait(200, SC_NS);
58
59     // Reading multiplication results from multiplier if result is ready
60     for(int k = 0; k < 33; k++){
61         wait(100, SC_NS);
62         if(input control_signals_processor.read()[1] == '1')
63             (wait(200, SC_NS));

```

Line 71, Column 2      Spaces: 3      C++

Fig. 33 Processor processor\_function method

```

File Edit Selection Find View Goto Tools Project Preferences Help
Processor.cpp x Memory_TB.cpp x multiplier_intermediate_component_fifo_interface.cpp x cpp x + ▾
41     Addr_from_processor = "01";
42
43     wait(200, SC_NS);
44
45     // Sending 32 32 bit data to intermediate component if fifo is empty
46     if(input control_signals_processor.read()[0] == '1'){
47         for(int j = 0; j < 33; j++){
48             wait(100, SC_NS);
49             output data_processor = Register_file[i*32 + j];
50         }
51
52     wait(0, SC_NS);
53
54     // Processor chooses multiplier to communicate with
55     Addr_from_processor = "10";
56
57     wait(200, SC_NS);
58
59     // Reading multiplication results from multiplier if result is ready
60     for(int k = 0; k < 33; k++){
61         wait(100, SC_NS);
62         if(input control_signals_processor.read()[1] == '1')
63             (wait(200, SC_NS));
64     }
65
66     wait(0, SC_NS);
67 }
68
69 Addr from processor = "11";
70 wait(6000, SC_NS);
71
72 }

```

Line 14, Column 1      Spaces: 3      C++

Fig. 34 Processor processor\_function method

### *I. Digital System*

Here the entire digital system is made by putting all above components together. Simply we just take some instances from all above modules and connect them to each other properly.

Activities Text Editor • May 26 06:18 •

Digital\_System.h  
Save

```
1 #include "../Bus/Bus.h"
2 #include "../Processor/Processor.cpp"
3 #include "../Memory/memoryReadWrites.h"
4 #include "../multiplier_intermediate_component_fifo_interface/
    multiplier_intermediate_component_fifo_interface.cpp"
5
6 SC_MODULE(digital_system){
7
8     sc_in <sc_logic> clk, rst;
9
10    sc_signal <sc_lv<2> > Addr_from_processor;
11    sc_signal <sc_lv<32> > input_data_processor;
12    sc_signal <sc_lv<32> > output_data_processor;
13    sc_signal <sc_lv<32> > input_control_signals_processor;
14    sc_signal <sc_lv<32> > output_control_signals_processor;
15
16    sc_signal <sc_logic> Ctrl_mem;
17    sc_signal <sc_lv<32> > input_data_mem, output_data_mem;
18    sc_signal <sc_lv<32> > input_address_mem;
19
20    sc_signal <sc_lv<32> >
        output_control_signals_intermediate_component;
21    sc_signal <sc_lv<32> > input_data_intermediate_component;
22
23    sc_signal <sc_lv<32> > input_control_signals_multiplier_16_bit;
24    sc_signal <sc_lv<32> > output_control_signals_multiplier_16_bit;
```

Fig. 35 *Digital System SystemC*

Activities Text Editor May 26 06:18

Digital\_System.h

~/Object\_Oriented\_Navabi/Home Work 5/Codes/Digital\_System.h

Save

Open

File

Object

Processor

FIFO

Intermediate Component

Multplier

SC\_CTOR(digital\_system){

```
25 sc_signal <sc_lv<32> > output_control_signals_multiplier_16_bit;
26 sc_signal <sc_lv<32> > output_data_multiplier_16_bit;
27
28 bus* Bus;
29 memory* MEM;
30 processor* Processor;
31 fifo<sc_lv<32>, 32> FIFO;
32 intermediate_component* Intermediate_Component;
33 multplier_16_bit* Multplier;
34
35 SC_CTOR(digital_system){
36
37     Processor = new processor("Processor");
38     Processor->clk(clk);
39     Processor->rst(rst);
40     Processor->addr_from_processor(Addr_from_processor);
41     Processor-
42         >input_data_processor(input_data_processor);
43         Processor-
44         >output_data_processor(output_data_processor);
45         Processor-
46         >output_address_processor(output_address_processor);
47             Processor-
48             >input_control_signals_processor(input_control_signals_processor);
49                 Processor-
50                 >output_control_signals_processor(output_control_signals_processor);
51
52 }
```

Fig. 36 *Digital System SystemC*

Activities Text Editor May 26 06:18

Open Save Digital\_System.h ~/Object\_Oriented - Navali/Home Work 5/Codes/D...

```
>output_control_signals_processor(output_control_signals_processor);
46
47     MEM = new memory("Memory");
48     MEM->in(*MEM->memBus);
49     MEM->clk(clk);
50     MEM->rst(rst);
51     MEM->Ctrl(Ctrl_mem);
52     MEM->Addr(Addr_from_processor);
53     MEM->input_data(input_data_mem);
54     MEM->output_data(output_data_mem);
55     MEM->input_address(input_address_mem);
56
57     Intermediate_Component = new
intermediate_component("Intermediate_Component");
58             Intermediate_Component->clk(clk);
59             Intermediate_Component->rst(rst);
60             Intermediate_Component-
>Addr_from_processor(Addr_from_processor);
61             Intermediate_Component-
>input_data(input_data_intermediate_component);
62             Intermediate_Component-
>output_control_signals(output_control_signals_intermediate_component);
63
64             Intermediate_Component->out(FIFO);
65
66     Multiplier = new multiplier_16_bit("Multiplier");
67     Multiplier->clk(clk);
```

Fig. 37 Digital System SystemC

Activities Text Editor May 26 06:18

Digital\_System.h  
~/Object\_Oriented-Navabi/Home Work 5/Codes/D...

Open Save ⌂ X

```
63
64         Intermediate_Component->out(FIFO);
65
66     Multiplier = new multiplier_16_bit("Multiplier");
67         Multiplier->clk(clk);
68         Multiplier->rst(rst);
69         Multiplier->Addr_from_processor(Addr_from_processor);
70         Multiplier-
71             >output_data(output_data_multiplier_16_bit);
72             Multiplier-
73                 >input_control_signals(input_control_signals_multiplier_16_bit);
74                 Multiplier-
75                     >output_control_signals(output_control_signals_multiplier_16_bit);
76
77         Multiplier->in(FIFO);
78
79     Bus = new bus("Bus");
80         Bus->Addr_from_processor(Addr_from_processor);
81         Bus->input_data_processor(input_data_processor);
82         Bus->output_data_processor(output_data_processor);
83         Bus-
84             >output_address_processor(output_address_processor);
85             Bus-
86                 >input_control_signals_processor(input_control_signals_processor);
87                 Bus-
88                     >output_control_signals_processor(output_control_signals_processor);
89
90
```

Fig. 38 *Digital System SystemC*

```

75     Bus = new bus("Bus");
76         Bus->Addr_from_processor(Addr_from_processor);
77         Bus->input_data_processor(input_data_processor);
78         Bus->output_data_processor(output_data_processor);
79         Bus-
80             >output_address_processor(output_address_processor);
81             Bus-
82                 >input_control_signals_processor(input_control_signals_processor);
83                     Bus-
84                         Bus->input_address_mem(input_address_mem);
85                         Bus->input_data_mem(input_data_mem);
86                         Bus->output_data_mem(output_data_mem);
87                         Bus->Ctrl_mem(Ctrl_mem);
88 
89             Bus-
90                 >output_control_signals_intermediate_component(output_control_signals_interme
91                     Bus-
92                         >input_data_intermediate_component(input_data_intermediate_component);
93                         Bus-
94                             >input_control_signals_multiplier_16_bit(input_control_signals_multiplier_1
95                                 Bus-
96                                     >output_control_signals_multiplier_16_bit(output_control_signals_multiplier_1
97     });

```

Fig. 39 Digital System SystemC

```

aryataghizadeh@aryataghizadeh-VirtualBox:~/Object_Oriented - Navabi/Home Work 5/Codes/D...
aryataghizadeh@aryataghizadeh-VirtualBox:~/Object_Oriented - Navabi/Home Work 5/Codes/Digital_System$ ./a.out
SystemC 2.3.1-Accellera --- Apr 29 2023 04:07:43
Copyright (c) 1996-2014 by all Contributors,
ALL RIGHTS RESERVED
Memory WROTE Data:196608 Address:8 requested by Processor
Memory WROTE Data:196609 Address:9 requested by Processor
Memory WROTE Data:196609 Address:9 requested by Processor
Memory WROTE Data:196610 Address:10 requested by Processor
Memory WROTE Data:196610 Address:10 requested by Processor
Memory WROTE Data:196611 Address:11 requested by Processor
Memory WROTE Data:196611 Address:11 requested by Processor
Memory WROTE Data:196612 Address:12 requested by Processor
Memory WROTE Data:196612 Address:12 requested by Processor
Memory WROTE Data:196613 Address:13 requested by Processor
Memory WROTE Data:196613 Address:13 requested by Processor
Memory WROTE Data:196614 Address:14 requested by Processor
Memory WROTE Data:196614 Address:14 requested by Processor
Memory WROTE Data:196615 Address:15 requested by Processor
Memory WROTE Data:196615 Address:15 requested by Processor
Memory WROTE Data:196616 Address:16 requested by Processor
Memory WROTE Data:196616 Address:16 requested by Processor
Memory WROTE Data:196617 Address:17 requested by Processor
Memory WROTE Data:196617 Address:17 requested by Processor
Memory WROTE Data:196618 Address:18 requested by Processor
Memory WROTE Data:196618 Address:18 requested by Processor
Memory WROTE Data:196619 Address:19 requested by Processor
Memory WROTE Data:196619 Address:19 requested by Processor

```

Fig. 41 Digital System Test Bench results

```

79     Bus->output_data_processor(output_data_processor);
80     Bus-
81         >output_address_processor(output_address_processor);
82             Bus-
83                 >input_control_signals_processor(input_control_signals_processor);
84                     Bus-
85                         Bus->input_address_mem(input_address_mem);
86                         Bus->input_data_mem(input_data_mem);
87                         Bus->output_data_mem(output_data_mem);
88                         Bus->Ctrl_mem(Ctrl_mem);
89 
90             Bus-
91                 >output_control_signals_intermediate_component(output_control_signals_interme
92                     Bus-
93                         >input_data_intermediate_component(input_data_intermediate_component);
94                         Bus-
95                             >input_control_signals_multiplier_16_bit(input_control_signals_multiplier_1
96                                 Bus-
97                                     >output_data_multiplier_16_bit(output_data_multiplier_16_bit);
98     };

```

Fig. 40 Digital System SystemC

```

aryataghizadeh@aryataghizadeh-VirtualBox:~/Object_Oriented - Navabi/Home Work 5/Codes/D...
aryataghizadeh@aryataghizadeh-VirtualBox:~/Object_Oriented - Navabi/Home Work 5/Codes/Digital_System$ ./a.out
Memory WROTE Data:196721 Address:121 requested by Processor
Memory WROTE Data:196722 Address:122 requested by Processor
Memory WROTE Data:196722 Address:122 requested by Processor
Memory WROTE Data:196723 Address:123 requested by Processor
Memory WROTE Data:196723 Address:123 requested by Processor
Memory WROTE Data:196724 Address:124 requested by Processor
Memory WROTE Data:196724 Address:124 requested by Processor
Memory WROTE Data:196725 Address:125 requested by Processor
Memory WROTE Data:196725 Address:125 requested by Processor
Memory WROTE Data:196726 Address:126 requested by Processor
Memory WROTE Data:196726 Address:126 requested by Processor
Memory WROTE Data:196727 Address:127 requested by Processor
Memory WROTE Data:196727 Address:127 requested by Processor
Memory WROTE Data:196728 Address:128 requested by Processor
Memory WROTE Data:196728 Address:128 requested by Processor
Memory WROTE Data:196729 Address:129 requested by Processor
Memory WROTE Data:196729 Address:129 requested by Processor
Memory WROTE Data:196730 Address:130 requested by Processor
Memory WROTE Data:196730 Address:130 requested by Processor
Memory WROTE Data:196731 Address:131 requested by Processor
Memory WROTE Data:196731 Address:131 requested by Processor
Memory WROTE Data:196732 Address:132 requested by Processor
Memory WROTE Data:196732 Address:132 requested by Processor
Memory WROTE Data:196733 Address:133 requested by Processor
Memory WROTE Data:196733 Address:133 requested by Processor
Memory WROTE Data:196734 Address:134 requested by Processor
Memory WROTE Data:196734 Address:134 requested by Processor
Memory WROTE Data:196735 Address:135 requested by Processor
Memory WROTE Data:196735 Address:135 requested by Processor

```

Fig. 42 Digital System Test Bench results



```

Activities Terminal ▾ May 26 07:05 •
ariyataghizadeh@ariyataghizadeh-VirtualBox: ~/Object_Ori...
A input: 0
B input: 3
Mult Result: 0 (at 29300 ns)
A input: 1
B input: 3
Mult Result: 3 (at 29400 ns)
A input: 2
B input: 3
Mult Result: 6 (at 29500 ns)
A input: 3
B input: 3
Mult Result: 9 (at 29600 ns)
A input: 4
B input: 3
Mult Result: 12 (at 29700 ns)
A input: 5
B input: 3
Mult Result: 15 (at 29800 ns)
A input: 6
B input: 3
Mult Result: 18 (at 29900 ns)
A input: 7
B input: 3
Mult Result: 21 (at 30 us)
A input: 8
B input: 3
Mult Result: 24 (at 30100 ns)
A input: 9
B input: 3

```

Fig. 47 Digital System Test Bench results

```

Activities Terminal ▾ May 26 07:05 •
ariyataghizadeh@ariyataghizadeh-VirtualBox: ~/Object_Ori...
A input: 119
B input: 3
Mult Result: 357 (at 52500 ns)
A input: 120
B input: 3
Mult Result: 360 (at 52600 ns)
A input: 121
B input: 3
Mult Result: 363 (at 52700 ns)
A input: 122
B input: 3
Mult Result: 366 (at 52800 ns)
A input: 123
B input: 3
Mult Result: 369 (at 52900 ns)
A input: 124
B input: 3
Mult Result: 372 (at 53 us)
A input: 125
B input: 3
Mult Result: 375 (at 53100 ns)
A input: 126
B input: 3
Mult Result: 378 (at 53200 ns)
A input: 127
B input: 3
Mult Result: 381 (at 53300 ns)
aryyataghizadeh@ariyataghizadeh-VirtualBox:~/Object_Oriented - Navabi/Home Work
5/Codes/Digital_System$
```

Fig. 49 Digital System Test Bench results

```

Activities Terminal ▾ May 26 07:05 •
ariyataghizadeh@ariyataghizadeh-VirtualBox: ~/Object_Ori...
A input: 97
B input: 3
Mult Result: 291 (at 50300 ns)
A input: 98
B input: 3
Mult Result: 294 (at 50400 ns)
A input: 99
B input: 3
Mult Result: 297 (at 50500 ns)
A input: 100
B input: 3
Mult Result: 300 (at 50600 ns)
A input: 101
B input: 3
Mult Result: 303 (at 50700 ns)
A input: 102
B input: 3
Mult Result: 306 (at 50800 ns)
A input: 103
B input: 3
Mult Result: 309 (at 50900 ns)
A input: 104
B input: 3
Mult Result: 312 (at 51 us)
A input: 105
B input: 3
Mult Result: 315 (at 51100 ns)
A input: 106
B input: 3

```

Fig. 48 Digital System Test Bench results