VLSI DESIGN LAB. ASSIGNMENT -9

SAURAV GUPTA

ROLL No. – 153070046

Dept. - Electrical Engineering

Specialisation – Microelectronics (TA)

Batch - 2015-17

Teacher concerned – VIRENDRA SINGH

Problem Statement:

Design a router using *systemc* to be used with network-on-chip (NoC) that is organized in an array of 4x4. A router consists of 4 routing ports (*N, S, E, W*) and a local port for the connected core. Keeping it a low cost router, it should use dimension order routing (DOR) so that it does not requires to store routing table. Data is transferred as flit. It uses virtual channels (2 virtual channel per input) to prevent deadlock condition.

Flit Format: The flit format is presented in Table 1 and 2. The 32-bit packet consists of a header flit followed by payload flits. Two additional 2-bit heads are type and (Identity ID) bits. The flit type can be as header (Header flit), message body, and the end of message body or the last flit of the message (Tail flit).

Table 1: Flit format

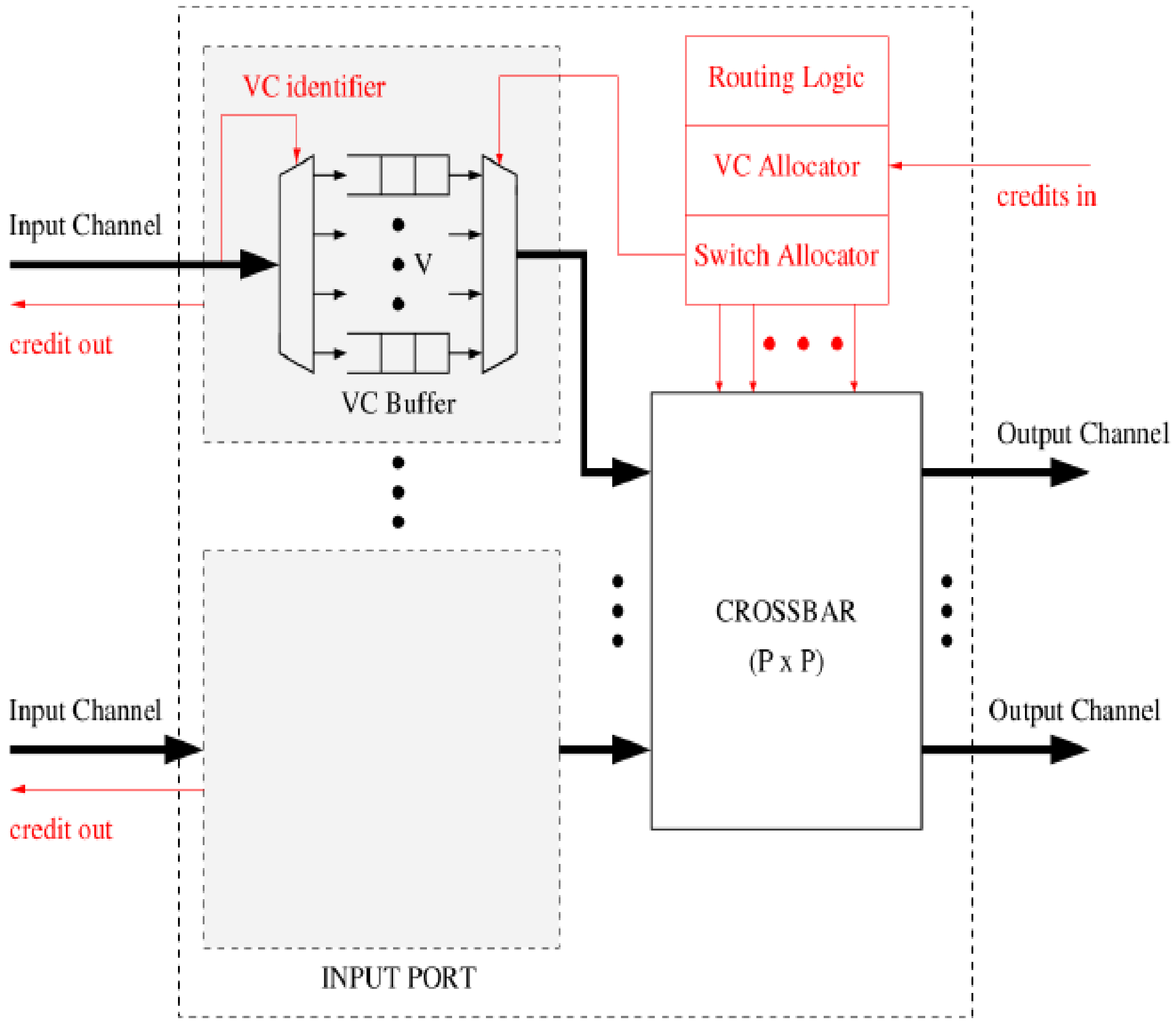
Type header (2 bits)	ID (2 bits)	Source X (2 bits)	Source Y (2 bits)	Dest. X (2 bits)	Dest. Y (2 bits)	Payload (20 bits)
Type header	ID	Payload (28 bits)				
		. . . .				
Type header	ID	Payload (28 bits)				

Table 2: Flit types

Type of flits	Binary (Decode) code
No flit	00 (0)
Header Flit	01 (1)
Message/Data body	10 (2)
Tail flit	11 (3)

In order to control the flow every router sends a credit (availability of one flit space or not). Based on the credit it transfers the flit to the neighbor.

ROUTER ARCHITECTURE



Sol:

Components:

1. **Router –** sixteen – 4X4 grid structure
   - **1.1 Input port –** five – Local, North, South, East, West
   - **1.2 Delay Element – five** – Local, North,South,East,West
   - **1.3 Routing Logic –** DOR-XY

1.4 **Switch Allocator –** Credit Based flow control

1.5 **Crossbar Switch –** 5X5 crossbar switch

2. **Router_16**
3. **Router_16_tb**

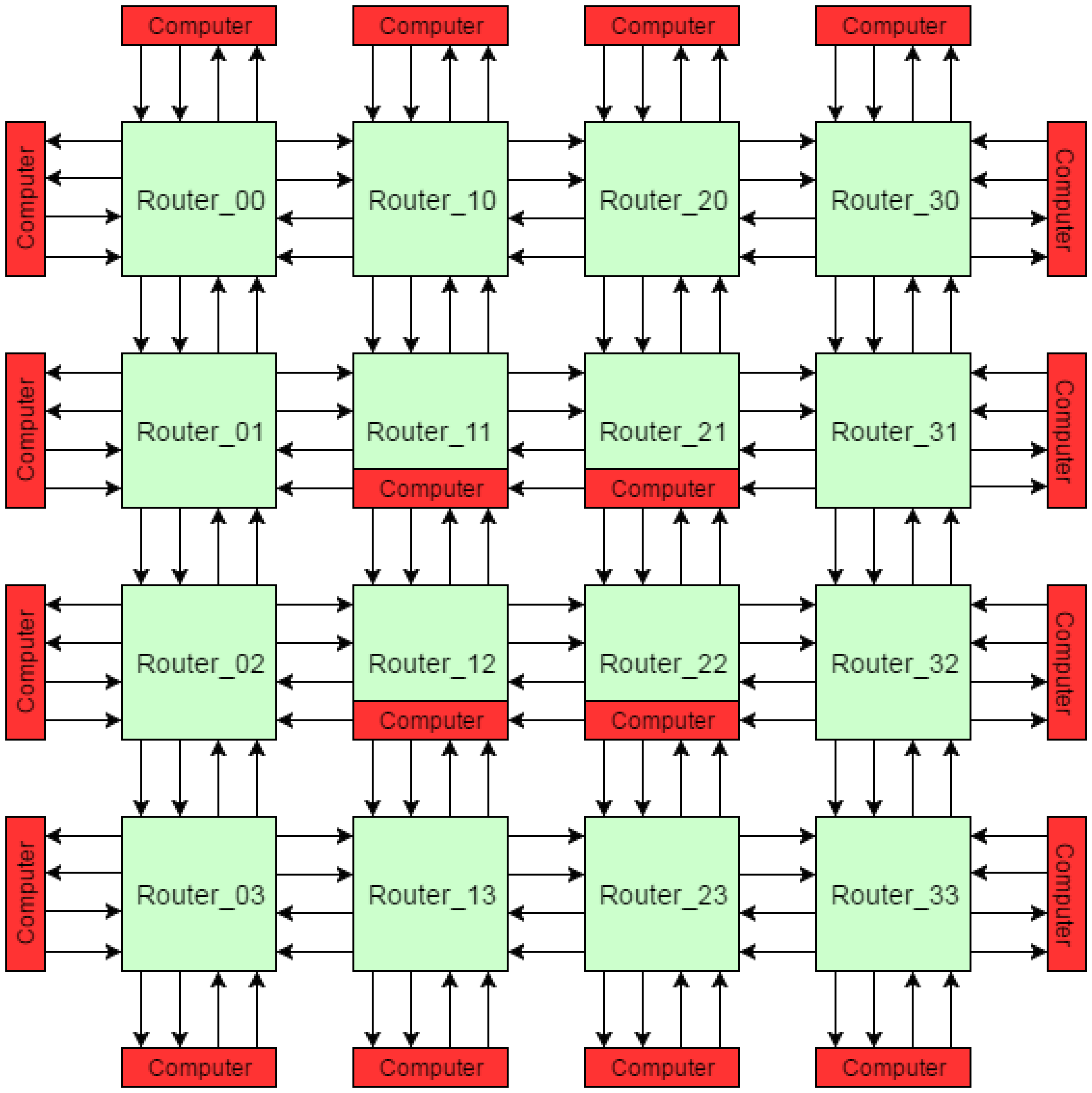


Fig.1 Overall Network of 4x4 NOC Routers

1. Router

Each Router has 5 data input ports, 5 data output ports, 5 credit in ports and 5 credit out ports in general except the edge ones.

For the edge ones few dummy inputs are connected since there are no other at one or the other edge.

//router.h

```
#include "Input_port.h"
#include "crossbar_switch.h"
#include "routing_logic.h"
#include "Switching_Allocator.h"
#include "delay_element.h"


SC_MODULE(router){

	sc_in_clk clock;
	sc_in<sc_bv<2> > XPresent, YPresent;
	sc_in<sc_bv<32> > input_flit_Local, input_flit_North, input_flit_South, input_flit_East,
input_flit_West;
	sc_in<bool> credit_inL, credit_inN, credit_inS, credit_inE, credit_inW;
	sc_out<bool> Credit_outL, Credit_outN, Credit_outS, Credit_outE, Credit_outW;
	sc_out<sc_bv<32> > Local_output, North_output, South_output, East_output, West_output;

	sc_signal<sc_bv<32> > output_flitL, output_flitN, output_flitS, output_flitE,
output_flitW;
	sc_signal<sc_bv<32> > Output_flitL_delayed, Output_flitN_delayed, Output_flitS_delayed,
Output_flitE_delayed, Output_flitW_delayed;
	sc_signal<bool> Data_RequestL, Data_RequestN, Data_RequestS, Data_RequestE,
Data_RequestW;
	sc_signal<bool> Data_RequestLR, Data_RequestNR, Data_RequestSR, Data_RequestER,
Data_RequestWR;
	sc_signal<bool> MoveUpL, MoveDownL, MoveRightL, MoveLeftL, MoveDownN, MoveRightN,
MoveLeftN, MoveCoreN, MoveUpS, MoveRightS, MoveLeftS, MoveCoreS, MoveUpE, MoveDownE, MoveLeftE,
MoveCoreE, MoveUpW, MoveDownW, MoveRightW, MoveCoreW;
	sc_signal<sc_bv<4> > NSEWtoL, LSEWtoN, LNEWtoS, LNSWtoE, LNSEtoW;

	sc_signal<sc_bv<2> > Type_headerL, Type_headerN, Type_headerS, Type_headerE,
Type_headerW;
	sc_signal<sc_bv<2> > XDestinationL, YDestinationL, XDestinationN, YDestinationN,
XDestinationS, YDestinationS, XDestinationE, YDestinationE, XDestinationW, YDestinationW;
	sc_bv<32> outputL, outputN, outputS, outputE, outputW;

	Input_port *input_portL, *input_portN, *input_portS, *input_portE, *input_portW;
	crossbar_switch *crossbar;
	routing_logic *routing;
	Switching_Allocator *Switching;
	delay_element *delayL, *delayN, *delayS, *delayE, *delayW;


	void output();

	SC_CTOR(router){
		SC_METHOD(output);
		sensitive << output_flitL << output_flitN << output_flitS << output_flitE <<
output_flitW;

		input_portL = new Input_port("inputL");
		(*input_portL)(clock, Data_RequestL, Data_RequestLR, input_flit_Local, 
Credit_outL, output_flitL);
		input_portN = new Input_port("inputN");
		(*input_portN)(clock, Data_RequestN, Data_RequestNR, input_flit_North, 
Credit_outN, output_flitN);
		input_portS = new Input_port("inputS");
		(*input_portS)(clock, Data_RequestS, Data_RequestSR, input_flit_South, 
Credit_outS, output_flitS);
		input_portE = new Input_port("inputE");
		(*input_portE)(clock, Data_RequestE, Data_RequestER, input_flit_East, Credit_outE, 
output_flitE);
		input_portW = new Input_port("inputW");
		(*input_portW)(clock, Data_RequestW, Data_RequestWR, input_flit_West, Credit_outW, 
output_flitW);
		delayL = new delay_element("delayL");
		(*delayL)(clock, output_flitL, Output_flitL_delayed);
		delayN = new delay_element("delayN");
		(*delayN)(clock, output_flitN, Output_flitN_delayed);
		delayS = new delay_element("delayS");
		(*delayS)(clock, output_flitS, Output_flitS_delayed);
		delayE = new delay_element("delayE");
		(*delayE)(clock, output_flitE, Output_flitE_delayed);
		delayW = new delay_element("delayW");
		(*delayW)(clock, output_flitW, Output_flitW_delayed);
		crossbar = new crossbar_switch("crossbar1");
		(*crossbar)(clock, Output_flitL_delayed, Output_flitN_delayed, 
Output_flitS_delayed, Output_flitE_delayed, Output_flitW_delayed, NSEWtoL, LSEWtoN, LNEWtoS, 
LNSWtoE, LNSEtoW, Local_output, North_output, South_output, East_output, West_output);
		routing = new routing_logic("routing1");
		(*routing)(clock, Type_headerL, Type_headerN, Type_headerS, Type_headerE, 
Type_headerW, XDestinationL, YDestinationL, XDestinationN, YDestinationN, XDestinationS, 
YDestinationS, XDestinationE, YDestinationE, XDestinationW, YDestinationW, XPresent, YPresent, 
MoveUpL, MoveDownL, MoveRightL, MoveLeftL, MoveDownN, MoveRightN, MoveLeftN, MoveCoreN, MoveUpS, 
MoveRightS, MoveLeftS, MoveCoreS, MoveUpE, MoveDownE, MoveLeftE, MoveCoreE, MoveUpW, MoveDownW, 
MoveRightW, MoveCoreW, Data_RequestLR, Data_RequestNR, Data_RequestSR, Data_RequestER, 
Data_RequestWR);
		Switching = new Switching_Allocator("Switching1");
		(*Switching)(clock, credit_inL, credit_inN, credit_inS, credit_inE, credit_inW, 
MoveUpL, MoveDownL, MoveRightL, MoveLeftL, MoveCoreN, MoveDownN, MoveRightN, MoveLeftN, 
MoveCoreS, MoveUpS, MoveRightS, MoveLeftS, MoveCoreE, MoveUpE, MoveDownE, MoveLeftE, MoveCoreW, 
MoveUpW, MoveDownW, MoveRightW, NSEWtoL, LSEWtoN, LNEWtoS, LNSWtoE, LNSEtoW, Data_RequestL, 
Data_RequestN, Data_RequestS, Data_RequestE, Data_RequestW);
	}
	~router(){
		delete input_portL;
		delete input_portN;
		delete input_portS;
		delete input_portE;
		delete input_portW;
		delete crossbar;
		delete routing;
		delete Switching;
	}
	};
```

//router.cpp

```
#include "router.h"

void router:: output(){
	outputL = output_flitL;
	outputN = output_flitN;
	outputS = output_flitS;
	outputE = output_flitE;
	outputW = output_flitW;

	Type_headerL = outputL.range(31,30);
	Type_headerN = outputN.range(31,30);
	Type_headerS = outputS.range(31,30);
```

```
        Type_headerE = outputE.range(31,30);
        Type_headerW = outputW.range(31,30);

        XDestinationL = outputL.range(23, 22);
        YDestinationL = outputL.range(21, 20);
        XDestinationN = outputN.range(23, 22);
        YDestinationN = outputN.range(21, 20);
        XDestinationS = outputS.range(23, 22);
        YDestinationS = outputS.range(21, 20);
        XDestinationE = outputE.range(23, 22);
        YDestinationE = outputE.range(21, 20);
        XDestinationW = outputW.range(23, 22);
        YDestinationW = outputW.range(21, 20);
}
```

1.1 Input Port

Each input port contains FIFO Virtual Channel Buffer of 6-word length, where each word is of 32-bits.
Input flit comes at every positive edge of the clock and output flit leaves at positive edge of the clock. However, the decisions are made at negative edge of the clock to avoid any ambiguity in data.

Note that there is a data_request signal input coming from switch allocator which indicates that output has been taken. Whenever this signal comes VC Buffer is right shifted by 32 bits/1 word and Credit_out signal is made high indicating that one word of the buffer has been cleared and there is more room for new data.

The Block diagram of Input Port is shown in fig. 1.1

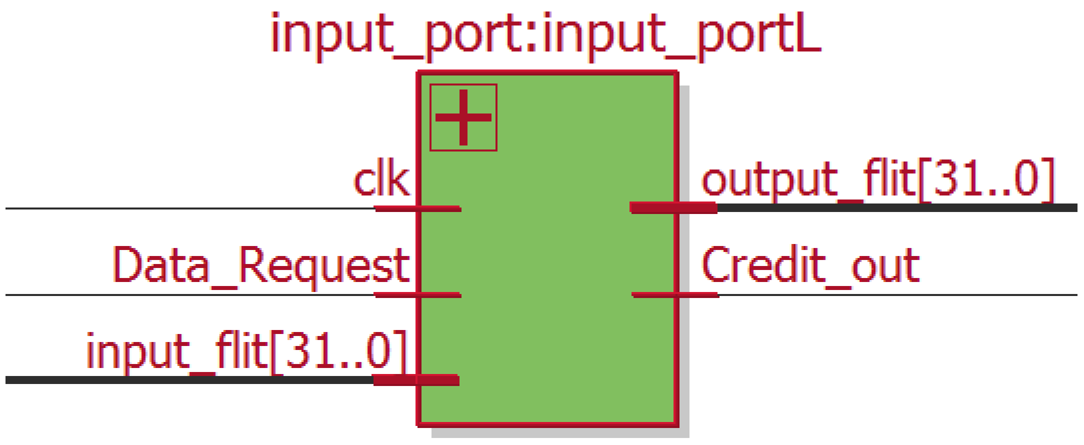


Fig. 1.1 Block Diagram of Input Port

Code:

//Input_Port.h

```
#include "systemc.h"

SC_MODULE(Input_port){
        sc_in_clk clock;
        sc_in<bool> Data_Request_switching;
        sc_in<bool> Data_Request_routing;
        sc_in<sc_bv<32> >input_flit;
        sc_out<bool> credit_out;
        sc_out<sc_bv<32> > output_flit;

        sc_bv<192> Virtual_buffer = 0x000000000000000000000000000000000000000000000000;
        bool Data_Request;
        sc_trace_file *wf;

        void input();
        void Request();

        SC_CTOR(Input_port){
                wf = sc_create_vcd_trace_file("input_port");
                sc_trace(wf, clock, "clock");
                sc_trace(wf, Data_Request, "Data_Request");
                sc_trace(wf, input_flit, "input_flit");
                sc_trace(wf, credit_out, "credit_out");
                sc_trace(wf, output_flit, "output_flit");
                sc_trace(wf, Virtual_buffer, "Virtual_buffer");
                SC_METHOD(input);
                sensitive << clock.neg();
                SC_METHOD(Request);
                sensitive << Data_Request_switching << Data_Request_routing;
        }

        ~Input_port(){
                sc_close_vcd_trace_file(wf);
        }
        };
```

//Input_port.cpp

```
#include "Input_port.h"

void Input_port::input() {
        if (Data_Request == 1){
                Virtual_buffer = Virtual_buffer >> 32;
                credit_out = 1;
        }

        if (Virtual_buffer.range(31, 30) == "00" || Virtual_buffer.range(31, 30) == "XX")
                Virtual_buffer.range(31, 0) = input_flit.read();
        else if (Virtual_buffer.range(63, 62) == "00" || Virtual_buffer.range(63, 62) == "XX")
                Virtual_buffer.range(63, 32) = input_flit.read();
        else if (Virtual_buffer.range(95, 94) == "00" || Virtual_buffer.range(95, 94) == "XX")
                Virtual_buffer.range(95, 64) = input_flit.read();
        else if (Virtual_buffer.range(127, 126) == "00" || Virtual_buffer.range(127, 126) ==
"XX")
                Virtual_buffer.range(127, 96) = input_flit.read();
        else if (Virtual_buffer.range(159, 158) == "00" || Virtual_buffer.range(159, 158) ==
"XX")
                Virtual_buffer.range(159, 128) = input_flit.read();
        else if (Virtual_buffer.range(191, 190) == "00" || Virtual_buffer.range(191, 190) ==
"XX")
                Virtual_buffer.range(191, 160) = input_flit.read();

        output_flit = Virtual_buffer.range(31, 0);
}

void Input_port::Request(){
        Data_Request = Data_Request_routing & Data_Request_switching;
        }
```

1.2 Delay Element

Delay element is used to delay the data input for the crossbar switch since it takes one clock cycle to make decision by the routing logic and the switch allocator.

CODE:

//delay_element.h

```
#include "systemc.h"

SC_MODULE(delay_element){
        sc_in_clk clock;
        sc_in<sc_bv<32> > Input_flit;
        sc_out<sc_bv<32> > Output_flit;

        sc_bv<32> temp;

        void delay_prc();
        void delay_prc1();

        SC_CTOR(delay_element){
                SC_METHOD(delay_prc1);
                sensitive << clock.pos();
                SC_METHOD(delay_prc);
                sensitive << clock.neg();
        }
};
```

//delay_element.cpp

```
#include "delay_element.h"

void delay_element::delay_prc1(){
        temp = Input_flit;
}

void delay_element::delay_prc(){
        Output_flit = temp;
}
```

1.3 Routing Logic

Routing Logic has 5 FSM for each input port.
FSM of Local input port is shown in fig.2.1

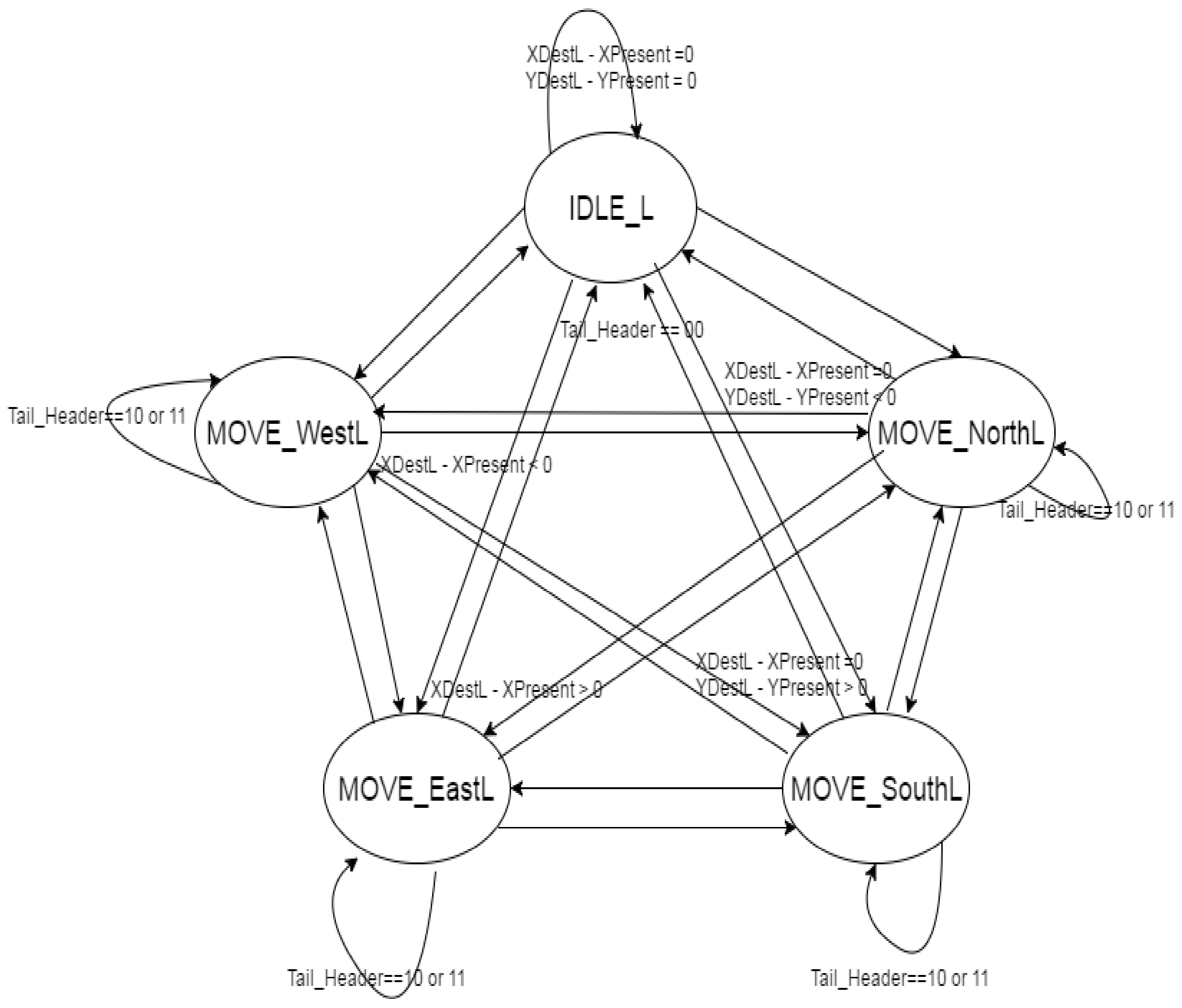


Fig. 2.1 FSM of Routing Logic for Local input port

FSM for North, South, East and West input port is similar to the above FSM. Here we have used Dimension order routing (DOR) , XY Routing to be precise.

Based on the XY coordinates of the Destination and the present address we determine where the data should be forwarded.

The Block diagram of Routing logic is given in fig. 2.2

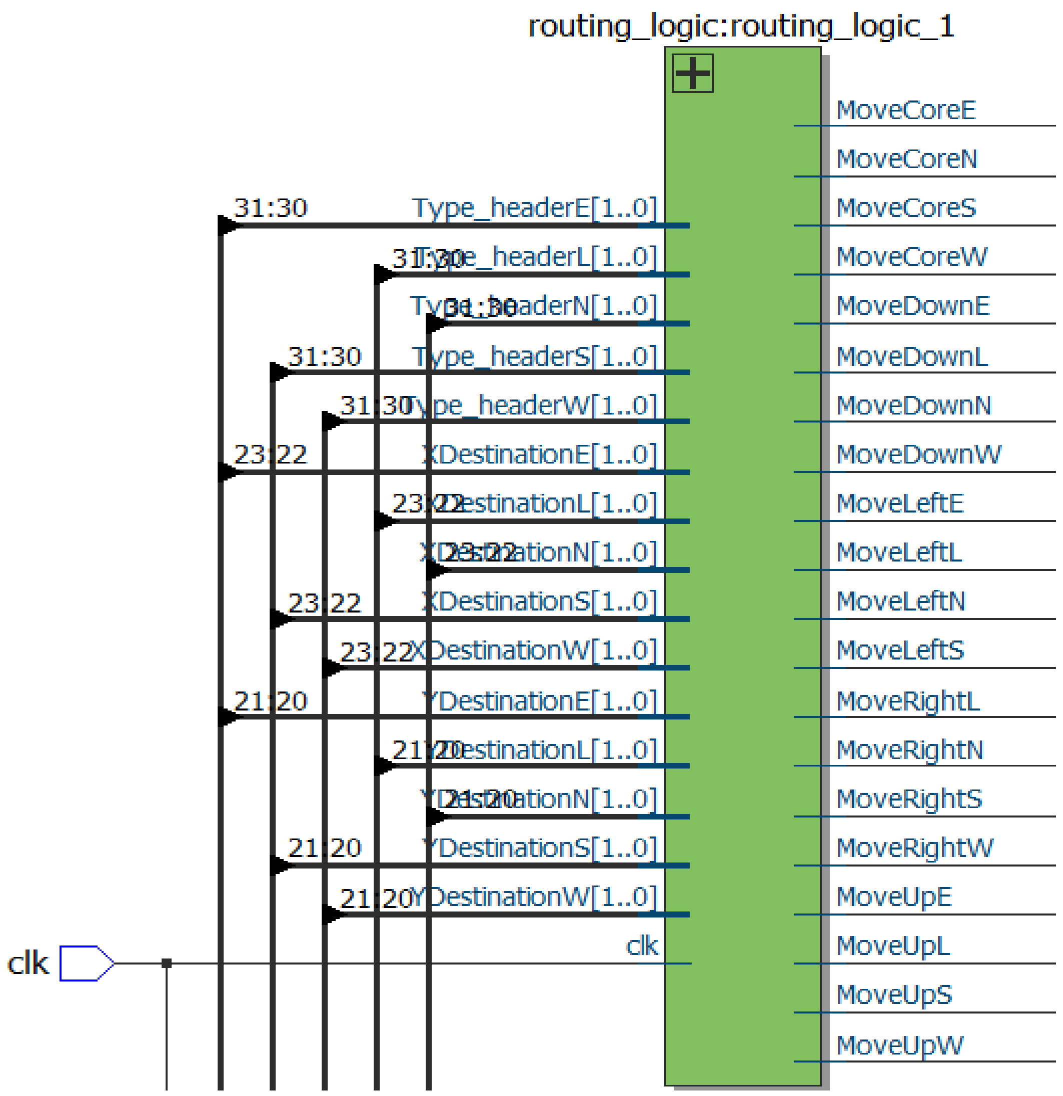


Fig. 2.2 Block Diagram of Routing logic

CODE:

// routing_logic.h

```
#include "systemc.h"

enum vm_state {
        IDLEL = 25, MoveNorthL, MoveSouthL, MoveEastL, MoveWestL
};
enum vm1_state {
        IDLEN = 30, MoveSouthN, MoveEastN, MoveWestN, MoveLocalN
};
enum vm2_state {
        IDLES = 35, MoveNorthS, MoveEastS, MoveWestS, MoveLocalS
};
enum vm3_state {
        IDLEE = 40, MoveNorthE, MoveSouthE, MoveWestE, MoveLocalE
};
enum vm4_state {
        IDLEW = 45, MoveNorthW, MoveSouthW, MoveEastW, MoveLocalW
```

```
};

SC_MODULE(routing_logic){
        sc_in_clk clock;
        sc_in<sc_bv<2> > Type_headerL, Type_headerN, Type_headerS, Type_headerE, Type_headerW;
        sc_in<sc_bv<2> > XDestinationL, YDestinationL, XDestinationN, YDestinationN,
XDestinationS, YDestinationS, XDestinationE, YDestinationE, XDestinationW, YDestinationW;
        sc_in<sc_bv<2> > XPresent, YPresent;
        sc_out<bool> MoveUpL, MoveDownL, MoveRightL, MoveLeftL;
        sc_out<bool> MoveDownN, MoveRightN, MoveLeftN, MoveCoreN;
        sc_out<bool> MoveUpS, MoveRightS, MoveLeftS, MoveCoreS;
        sc_out<bool> MoveUpE, MoveDownE, MoveLeftE, MoveCoreE;
        sc_out<bool> MoveUpW, MoveDownW, MoveRightW, MoveCoreW;
        sc_out<bool> Data_RequestL, Data_RequestN, Data_RequestS, Data_RequestE, Data_RequestW;
        sc_int<3> XdiffL, YdiffL, XdiffN, YdiffN, XdiffS, YdiffS, XdiffE, YdiffE, XdiffW, YdiffW;
        sc_uint<2> XPres, YPres;
        sc_uint<2> XDesL, YDesL, XDesN, YDesN, XDesS, YDesS, XDesE, YDesE, XDesW, YDesW;

        sc_signal<vm_state>state_signalL;
        sc_signal<vm_state>next_stateL; //Local input states
        sc_signal<vm1_state>state_signalN;
        sc_signal<vm1_state>next_stateN; //North input states
        sc_signal<vm2_state>state_signalS;
        sc_signal<vm2_state>next_stateS; //South input states
        sc_signal<vm3_state>state_signalE;
        sc_signal<vm3_state>next_stateE; //East input states
        sc_signal<vm4_state>state_signalW;
        sc_signal<vm4_state>next_stateW; //West input states

        sc_trace_file *fp;

        void getnextstateL();
        void getnextstateN();
        void getnextstateS();
        void getnextstateE();
        void getnextstateW();
        void setnextstate();
        void outputL();
        void outputN();
        void outputS();
        void outputE();
        void outputW();
        void address();
        void Data_Request();

        SC_CTOR(routing_logic){
                fp = sc_create_vcd_trace_file("routing_logic");
                sc_trace(fp, clock, "clock");
                sc_trace(fp, Type_headerL, "Type_headerL");
                sc_trace(fp, Type_headerN, "Type_headerN");
                sc_trace(fp, Type_headerS, "Type_headerS");
                sc_trace(fp, Type_headerE, "Type_headerE");
                sc_trace(fp, Type_headerW, "Type_headerW");
                sc_trace(fp, XDestinationL, "XDestinationL");
                sc_trace(fp, YDestinationL, "YDestinationL");
                sc_trace(fp, XDestinationN, "XDestinationN");
                sc_trace(fp, YDestinationN, "YDestinationN");
                sc_trace(fp, XDestinationS, "XDestinationS");
                sc_trace(fp, YDestinationS, "YDestinationS");
                sc_trace(fp, XDestinationE, "XDestinationE");
                sc_trace(fp, YDestinationE, "YDestinationE");
                sc_trace(fp, XDestinationW, "XDestinationW");
                sc_trace(fp, YDestinationW, " YDestinationW");

                sc_trace(fp, MoveUpL, "MoveUpL");
                sc_trace(fp, MoveDownL, "MoveDownL");
                sc_trace(fp, MoveRightL, "MoveRightL");
                sc_trace(fp, MoveLeftL, " MoveLeftL");

                sc_trace(fp, MoveDownN, "MoveDownN");
                sc_trace(fp, MoveRightN, "MoveRightN");
```

```
            sc_trace(fp, MoveLeftN, "MoveLeftN");
            sc_trace(fp, MoveCoreN, " MoveCoreN");

            sc_trace(fp, MoveUpS, "MoveUpS");
            sc_trace(fp, MoveRightS, "MoveRightS");
            sc_trace(fp, MoveLeftS, "MoveLeftS");
            sc_trace(fp, MoveCoreS, "MoveCoreS");

            sc_trace(fp, MoveUpE, "MoveUpE");
            sc_trace(fp, MoveDownE, "MoveDownE");
            sc_trace(fp, MoveLeftE, "MoveLeftE");
            sc_trace(fp, MoveCoreE, " MoveCoreE");

            sc_trace(fp, MoveUpW, "MoveUpW");
            sc_trace(fp, MoveDownW, "MoveDownW");
            sc_trace(fp, MoveRightW, "MoveRightW");
            sc_trace(fp, MoveCoreW, "MoveCoreW");

            sc_trace(fp, XDesE, "XDesE");
            sc_trace(fp, YDesE, "YDesE");
            sc_trace(fp, XdiffE, "XdiffE");
            sc_trace(fp, YdiffE, "YdiffE");
            sc_trace(fp, XDesL, "XDesL");
            sc_trace(fp, YDesL, "YDesL");
            sc_trace(fp, XdiffL, "XdiffL");
            sc_trace(fp, YdiffL, "YdiffL");
            sc_trace(fp, XdiffN, "XdiffN");
            sc_trace(fp, YdiffN, "YdiffN");


            SC_METHOD(getnextstateL);
            sensitive << state_signalL << Type_headerL << XDestinationL << YDestinationL;
            SC_METHOD(getnextstateN);
            sensitive << state_signalN << Type_headerN << XDestinationN << YDestinationN;
            SC_METHOD(getnextstateS);
            sensitive << state_signalS << Type_headerS << XDestinationS << YDestinationS;
            SC_METHOD(getnextstateE);
            sensitive << state_signalE << Type_headerE << XDestinationE << YDestinationE;
            SC_METHOD(getnextstateW);
            sensitive << state_signalW << Type_headerW << XDestinationE << YDestinationE;
            SC_METHOD(setnextstate);
            sensitive << clock.pos();
            SC_METHOD(outputL);
            sensitive << state_signalL;
            SC_METHOD(outputN);
            sensitive << state_signalN;
            SC_METHOD(outputS);
            sensitive << state_signalS;
            SC_METHOD(outputE);
            sensitive << state_signalE;
            SC_METHOD(outputW);
            sensitive << state_signalW;
            SC_METHOD(Data_Request);
            sensitive << clock.neg();
        }
        ~routing_logic(){
            sc_close_vcd_trace_file(fp);
        }
        };
```

//routing_logic.cpp

```
#include "routing_logic.h"

void routing_logic::getnextstateL() {
        XDesL = XDestinationL;
        YDesL = YDestinationL;
        XPres = XPresent;
        YPres = YPresent;
```

```
XdiffL = XDesL - XPres;
YdiffL = YDesL - YPres;

switch (state_signalL){
case IDLEL:
	if (Type_headerL.read() == "01") {
		if (XdiffL < 0)
			next_stateL = MoveWestL;
		else if (XdiffL > 0)
			next_stateL = MoveEastL;
		else {
			if (YdiffL < 0)
				next_stateL = MoveNorthL;
			else if (YdiffL > 0)
				next_stateL = MoveSouthL;
			else
				next_stateL = IDLEL;
		}
	}
	else
		next_stateL = IDLEL;
	break;


case MoveNorthL:
	if (Type_headerL.read() == "01") {
		if (XdiffL < 0)
			next_stateL = MoveWestL;
		else if (XdiffL > 0)
			next_stateL = MoveEastL;
		else {
			if (YdiffL < 0)
				next_stateL = MoveNorthL;
			else if (YdiffL > 0)
				next_stateL = MoveSouthL;
			else
				next_stateL = IDLEL;
		}
	}
	else if (Type_headerL.read() == "10" || Type_headerL.read() == "11")
		next_stateL = MoveNorthL;
	else
		next_stateL = IDLEL;
	break;

case MoveSouthL:
	if (Type_headerL.read() == "01") {
		if (XdiffL < 0)
			next_stateL = MoveWestL;
		else if (XdiffL > 0)
			next_stateL = MoveEastL;
		else {
			if (YdiffL < 0)
				next_stateL = MoveNorthL;
			else if (YdiffL > 0)
				next_stateL = MoveSouthL;
			else
				next_stateL = IDLEL;
		}
	}
	else if (Type_headerL.read() == "10" || Type_headerL.read() == "11")
		next_stateL = MoveSouthL;
	else
		next_stateL = IDLEL;
	break;

case MoveEastL:
	if (Type_headerL.read() == "01") {
		if (XdiffL < 0)
			next_stateL = MoveWestL;
		else if (XdiffL > 0)
```

```
                                next_stateL = MoveEastL;
                        else {
                                if (YdiffL < 0)
                                        next_stateL = MoveNorthL;
                                else if (YdiffL > 0)
                                        next_stateL = MoveSouthL;
                                else
                                        next_stateL = IDLEL;
                        }
                }
                else if (Type_headerL.read() == "10" || Type_headerL.read() == "11")
                        next_stateL = MoveEastL;
                else
                        next_stateL = IDLEL;
                break;

        case MoveWestL:
                if (Type_headerL.read() == "01") {
                        if (XdiffL < 0)
                                next_stateL = MoveWestL;
                        else if (XdiffL > 0)
                                next_stateL = MoveEastL;
                        else {
                                if (YdiffL < 0)
                                        next_stateL = MoveNorthL;
                                else if (YdiffL > 0)
                                        next_stateL = MoveSouthL;
                                else
                                        next_stateL = IDLEL;
                        }
                }
                else if (Type_headerL.read() == "10" || Type_headerL.read() == "11")
                        next_stateL = MoveWestL;
                else
                        next_stateL = IDLEL;
                break;

        default:
                next_stateL = IDLEL;
                break;
        }

}

void routing_logic::getnextstateN(){
        XDesN = XDestinationN;
        YDesN = YDestinationN;
        XPres = XPresent;
        YPres = YPresent;
        XdiffN = XDesN - XPres;
        YdiffN = YDesN - YPres;

        switch (state_signalN){
        case IDLEN:
                if (Type_headerN.read() == "01") {
                        if (XdiffN < 0)
                                next_stateN = MoveWestN;
                        else if (XdiffN > 0)
                                next_stateN = MoveEastN;
                        else {
                                if (YdiffN < 0)
                                        next_stateN = IDLEN; //won't happen;
                                else if (YdiffN > 0)
                                        next_stateN = MoveSouthN;
                                else
                                        next_stateN = MoveLocalN;
                        }
                }
                else
                        next_stateN = IDLEN;
                break;
```

```
		case MoveSouthN:
			if (Type_headerN.read() == "01") {
				if (XdiffN < 0)
					next_stateN = MoveWestN;
				else if (XdiffN > 0)
					next_stateN = MoveEastN;
				else {
					if (YdiffN < 0)
						next_stateN = IDLEN; //won't happen;
					else if (YdiffN > 0)
						next_stateN = MoveSouthN;
					else
						next_stateN = MoveLocalN;
				}
			}
			else if (Type_headerN.read() == "10" || Type_headerN.read() == "11")
				next_stateN = MoveSouthN;
			else
				next_stateN = IDLEN;
			break;

		case MoveEastN:
			if (Type_headerN.read() == "01") {
				if (XdiffN < 0)
					next_stateN = MoveWestN;
				else if (XdiffN > 0)
					next_stateN = MoveEastN;
				else {
					if (YdiffN < 0)
						next_stateN = IDLEN; //won't happen;
					else if (YdiffN > 0)
						next_stateN = MoveSouthN;
					else
						next_stateN = MoveLocalN;
				}
			}
			else if (Type_headerN.read() == "10" || Type_headerN.read() == "11")
				next_stateN = MoveEastN;
			else
				next_stateN = IDLEN;
			break;

		case MoveWestN:
			if (Type_headerN.read() == "01") {
				if (XdiffN < 0)
					next_stateN = MoveWestN;
				else if (XdiffN > 0)
					next_stateN = MoveEastN;
				else {
					if (YdiffN < 0)
						next_stateN = IDLEN; //won't happen;
					else if (YdiffN > 0)
						next_stateN = MoveSouthN;
					else
						next_stateN = MoveLocalN;
				}
			}
			else if (Type_headerN.read() == "10" || Type_headerN.read() == "11")
				next_stateN = MoveWestN;
			else
				next_stateN = IDLEN;
			break;

		case MoveLocalN:
			if (Type_headerN.read() == "01") {
				if (XdiffN < 0)
					next_stateN = MoveWestN;
				else if (XdiffN > 0)
					next_stateN = MoveEastN;
				else {
```

```
                                        if (YdiffN < 0)
                                                next_stateN = IDLEN; //won't happen;
                                        else if (YdiffN > 0)
                                                next_stateN = MoveSouthN;
                                        else
                                                next_stateN = MoveLocalN;
                                }
                        }
                        else if (Type_headerN.read() == "10" || Type_headerN.read() == "11")
                                next_stateN = MoveLocalN;
                        else
                                next_stateN = IDLEN;
                        break;

                default:
                        next_stateN = IDLEN;
                        break;
                }
        }

        void routing_logic::getnextstateS(){
                XDesS = XDestinationS;
                YDesS = YDestinationS;
                XPres = XPresent;
                YPres = YPresent;
                XdiffS = XDesS - XPres;
                YdiffS = YDesS - YPres;

                switch (state_signalS){
                case IDLES:
                        if (Type_headerS.read() == "01") {
                                if (XdiffS < 0)
                                        next_stateS = MoveWestS;
                                else if (XdiffS > 0)
                                        next_stateS = MoveEastS;
                                else {
                                        if (YdiffS < 0)
                                                next_stateS = MoveNorthS;
                                        else if (YdiffS > 0)
                                                next_stateS = IDLES; //won't happen
                                        else
                                                next_stateS = MoveLocalS;
                                }
                        }
                        else
                                next_stateS = IDLES;
                        break;

                case MoveNorthS:
                        if (Type_headerS.read() == "01") {
                                if (XdiffS < 0)
                                        next_stateS = MoveWestS;
                                else if (XdiffS > 0)
                                        next_stateS = MoveEastS;
                                else {
                                        if (YdiffS < 0)
                                                next_stateS = MoveNorthS;
                                        else if (YdiffS > 0)
                                                next_stateS = IDLES; //won't happen
                                        else
                                                next_stateS = MoveLocalS;
                                }
                        }
                        else if (Type_headerS.read() == "10" || Type_headerS.read() == "11")
                                next_stateS = MoveNorthS;
                        else
                                next_stateS = IDLES;
                        break;

                case MoveEastS:
                        if (Type_headerS.read() == "01") {
```

```
                        if (XdiffS < 0)
                                next_stateS = MoveWestS;
                        else if (XdiffS > 0)
                                next_stateS = MoveEastS;
                        else {
                                if (YdiffS < 0)
                                        next_stateS = MoveNorthS;
                                else if (YdiffS > 0)
                                        next_stateS = IDLES; //won't happen
                                else
                                        next_stateS = MoveLocalS;
                        }
                }
                else if (Type_headerS.read() == "10" || Type_headerS.read() == "11")
                        next_stateS = MoveEastS;
                else
                        next_stateS = IDLES;
                break;

        case MoveWestS:
                if (Type_headerS.read() == "01") {
                        if (XdiffS < 0)
                                next_stateS = MoveWestS;
                        else if (XdiffS > 0)
                                next_stateS = MoveEastS;
                        else {
                                if (YdiffS < 0)
                                        next_stateS = MoveNorthS;
                                else if (YdiffS > 0)
                                        next_stateS = IDLES; //won't happen
                                else
                                        next_stateS = MoveLocalS;
                        }
                }
                else if (Type_headerS.read() == "10" || Type_headerS.read() == "11")
                        next_stateS = MoveWestS;
                else
                        next_stateS = IDLES;
                break;

        case MoveLocalS:
                if (Type_headerS.read() == "01") {
                        if (XdiffS < 0)
                                next_stateS = MoveWestS;
                        else if (XdiffS > 0)
                                next_stateS = MoveEastS;
                        else {
                                if (YdiffS < 0)
                                        next_stateS = MoveNorthS;
                                else if (YdiffS > 0)
                                        next_stateS = IDLES; //won't happen
                                else
                                        next_stateS = MoveLocalS;
                        }
                }
                else if (Type_headerS.read() == "10" || Type_headerS.read() == "11")
                        next_stateS = MoveLocalS;
                else
                        next_stateS = IDLES;
                break;

        default:
                next_stateS = IDLES;
                break;
        }
}

void routing_logic::getnextstateE(){
        XDesE = XDestinationE;
        YDesE = YDestinationE;
        XPres = XPresent;
```

```
YPres = YPresent;
XdiffE = XDesE - XPres;
YdiffE = YDesE - YPres;
switch (state_signalE){
case IDLEE:
        if (Type_headerE.read() == "01") {
                if (XdiffE < 0)
                        next_stateE = MoveWestE;
                else if (XdiffE > 0)
                        next_stateE = IDLEE; //won't happen
                else {
                        if (YdiffE < 0)
                                next_stateE = MoveNorthE;
                        else if (YdiffE > 0)
                                next_stateE = MoveSouthE;
                        else
                                next_stateE = MoveLocalE;
                }
        }
        else
                next_stateE = IDLEE;
        break;

case MoveNorthE:
        if (Type_headerE.read() == "01") {
                if (XdiffE < 0)
                        next_stateE = MoveWestE;
                else if (XdiffE > 0)
                        next_stateE = IDLEE; //won't happen
                else {
                        if (YdiffE < 0)
                                next_stateE = MoveNorthE;
                        else if (YdiffE > 0)
                                next_stateE = MoveSouthE;
                        else
                                next_stateE = MoveLocalE;
                }
        }
        else if (Type_headerE.read() == "10" || Type_headerE.read() == "11")
                next_stateE = MoveNorthE;
        else
                next_stateE = IDLEE;
        break;

case MoveSouthE:
        if (Type_headerE.read() == "01") {
                if (XdiffE < 0)
                        next_stateE = MoveWestE;
                else if (XdiffE > 0)
                        next_stateE = IDLEE; //won't happen
                else {
                        if (YdiffE < 0)
                                next_stateE = MoveNorthE;
                        else if (YdiffE > 0)
                                next_stateE = MoveSouthE;
                        else
                                next_stateE = MoveLocalE;
                }
        }
        else if (Type_headerE.read() == "10" || Type_headerE.read() == "11")
                next_stateE = MoveSouthE;
        else
                next_stateE = IDLEE;
        break;

case MoveWestE:
        if (Type_headerE.read() == "01") {
                if (XdiffE < 0)
                        next_stateE = MoveWestE;
                else if (XdiffE > 0)
                        next_stateE = IDLEE; //won't happen
                        else {
                                if (YdiffE < 0)
                                        next_stateE = MoveNorthE;
                                else if (YdiffE > 0)
                                        next_stateE = MoveSouthE;
                                else
                                        next_stateE = MoveLocalE;
                        }
                }
                else if (Type_headerE.read() == "10" || Type_headerE.read() == "11")
                        next_stateE = MoveWestE;
                else
                        next_stateE = IDLEE;
                break;

        case MoveLocalE:
                if (Type_headerE.read() == "01") {
                        if (XdiffE < 0)
                                next_stateE = MoveWestE;
                        else if (XdiffE > 0)
                                next_stateE = IDLEE; //won't happen
                        else {
                                if (YdiffE < 0)
                                        next_stateE = MoveNorthE;
                                else if (YdiffE > 0)
                                        next_stateE = MoveSouthE;
                                else
                                        next_stateE = MoveLocalE;
                        }
                }
                else if (Type_headerE.read() == "10" || Type_headerE.read() == "11")
                        next_stateE = MoveLocalE;
                else
                        next_stateE = IDLEE;
                break;

        default:
                next_stateE = IDLEE;
                break;
        }
}

void routing_logic::getnextstateW(){
        XDesW = XDestinationW;
        YDesW = YDestinationW;
        XPres = XPresent;
        YPres = YPresent;
        XdiffW = XDesW - XPres;
        YdiffW = YDesW - YPres;
        switch (state_signalW)
        {
        case IDLEW:
                if (Type_headerW.read() == "01"){
                        if (XdiffW < 0)
                                next_stateW = IDLEW; //won't happen
                        else if (XdiffW > 0)
                                next_stateW = MoveEastW;
                        else {
                                if (YdiffW < 0)
                                        next_stateW = MoveNorthW;
                                else if (YdiffW > 0)
                                        next_stateW = MoveSouthW;
                                else
                                        next_stateW = MoveLocalW;
                        }
                }
                else
                        next_stateW = IDLEW;
                break;

        case MoveNorthW:
                if (Type_headerW.read() == "01"){
                        if (XdiffW < 0)
                                next_stateW = IDLEW; //won't happen
                        else if (XdiffW > 0)
                                next_stateW = MoveEastW;
                        else {
                                if (YdiffW < 0)
                                        next_stateW = MoveNorthW;
                                else if (YdiffW > 0)
                                        next_stateW = MoveSouthW;
                                else
                                        next_stateW = MoveLocalW;
                        }
                }
                else if (Type_headerW.read() == "10" || Type_headerW.read() == "11")
                        next_stateW = MoveNorthW;
                else
                        next_stateW = IDLEW;
                break;

        case MoveSouthW:
                if (Type_headerW.read() == "01"){
                        if (XdiffW < 0)
                                next_stateW = IDLEW; //won't happen
                        else if (XdiffW > 0)
                                next_stateW = MoveEastW;
                        else {
                                if (YdiffW < 0)
                                        next_stateW = MoveNorthW;
                                else if (YdiffW > 0)
                                        next_stateW = MoveSouthW;
                                else
                                        next_stateW = MoveLocalW;
                        }
                }
                else if (Type_headerW.read() == "10" || Type_headerW.read() == "11")
                        next_stateW = MoveSouthW;
                else
                        next_stateW = IDLEW;
                break;

        case MoveEastW:
                if (Type_headerW.read() == "01"){
                        if (XdiffW < 0)
                                next_stateW = IDLEW; //won't happen
                        else if (XdiffW > 0)
                                next_stateW = MoveEastW;
                        else {
                                if (YdiffW < 0)
                                        next_stateW = MoveNorthW;
                                else if (YdiffW > 0)
                                        next_stateW = MoveSouthW;
                                else
                                        next_stateW = MoveLocalW;
                        }
                }
                else if (Type_headerW.read() == "10" || Type_headerW.read() == "11")
                        next_stateW = MoveEastW;
                else
                        next_stateW = IDLEW;
                break;

        case MoveLocalW:
                if (Type_headerW.read() == "01"){
                        if (XdiffW < 0)
                                next_stateW = IDLEW; //won't happen
                        else if (XdiffW > 0)
                                next_stateW = MoveEastW;
                        else {
                                if (YdiffW < 0)
                                        next_stateW = MoveNorthW;
                        else if (YdiffW > 0)
                                next_stateW = MoveSouthW;
                        else
                                next_stateW = MoveLocalW;
                    }
                }
                else if (Type_headerW.read() == "10" || Type_headerW.read() == "11")
                    next_stateW = MoveLocalW;
                else
                    next_stateW = IDLEW;
                break;

        default:
                next_stateW = IDLEW;
                break;
        }
}

void routing_logic::setnextstate(){
        state_signalL = next_stateL;
        state_signalN = next_stateN;
        state_signalS = next_stateS;
        state_signalE = next_stateE;
        state_signalW = next_stateW;
}

void routing_logic::outputL(){
        switch (state_signalL){
        case IDLEL:
                MoveDownL = 0;
                MoveUpL = 0;
                MoveRightL = 0;
                MoveLeftL = 0;
                break;

        case MoveNorthL:
                MoveDownL = 0;
                MoveUpL = 1;
                MoveRightL = 0;
                MoveLeftL = 0;
                break;

        case MoveSouthL:
                MoveDownL = 1;
                MoveUpL = 0;
                MoveRightL = 0;
                MoveLeftL = 0;
                break;

        case MoveEastL:
                MoveDownL = 0;
                MoveUpL = 0;
                MoveRightL = 1;
                MoveLeftL = 0;
                break;

        case MoveWestL:
                MoveDownL = 0;
                MoveUpL = 0;
                MoveRightL = 0;
                MoveLeftL = 1;
                break;

        default:
                MoveDownL = 0;
                MoveUpL = 0;
                MoveRightL = 0;
                MoveLeftL = 0;
                break;

        }
```

```
    }

    void routing_logic::outputN(){
            switch (state_signalN){
            case IDLEN:
                    MoveCoreN = 0;
                    MoveDownN = 0;
                    MoveRightN = 0;
                    MoveLeftN = 0;
                    break;

            case MoveSouthN:
                    MoveCoreN = 0;
                    MoveDownN = 1;
                    MoveRightN = 0;
                    MoveLeftN = 0;
                    break;

            case MoveEastN:
                    MoveCoreN = 0;
                    MoveDownN = 0;
                    MoveRightN = 1;
                    MoveLeftN = 0;
                    break;
            case MoveWestN:
                    MoveCoreN = 0;
                    MoveDownN = 0;
                    MoveRightN = 0;
                    MoveLeftN = 1;
                    break;
            case MoveLocalN:
                    MoveCoreN = 1;
                    MoveDownN = 0;
                    MoveRightN = 0;
                    MoveLeftN = 0;
                    break;
            default:
                    MoveCoreN = 0;
                    MoveDownN = 0;
                    MoveRightN = 0;
                    MoveLeftN = 0;
            }
    }

    void routing_logic::outputS(){
            switch (state_signalS){
            case IDLES:
                    MoveCoreS = 0;
                    MoveUpS = 0;
                    MoveRightS = 0;
                    MoveLeftS = 0;
                    break;

            case MoveNorthS:
                    MoveCoreS = 0;
                    MoveUpS = 1;
                    MoveRightS = 0;
                    MoveLeftS = 0;
                    break;

            case MoveEastS:
                    MoveCoreS = 0;
                    MoveUpS = 0;
                    MoveRightS = 1;
                    MoveLeftS = 0;
                    break;

            case MoveWestS:
                    MoveCoreS = 0;
                    MoveUpS = 0;
                    MoveRightS = 0;
```

```
                        MoveLeftS = 1;
                        break;

                case MoveLocalS:
                        MoveCoreS = 1;
                        MoveUpS = 0;
                        MoveRightS = 0;
                        MoveLeftS = 0;
                        break;
                default:
                        MoveCoreS = 0;
                        MoveUpS = 0;
                        MoveRightS = 0;
                        MoveLeftS = 0;

                }
        }

        void routing_logic::outputE(){
                switch (state_signalE){
                case IDLEE:
                        MoveCoreE = 0;
                        MoveUpE = 0;
                        MoveDownE = 0;
                        MoveLeftE = 0;
                        break;

                case MoveNorthE:
                        MoveCoreE = 0;
                        MoveUpE = 1;
                        MoveDownE = 0;
                        MoveLeftE = 0;
                        break;

                case MoveSouthE:
                        MoveCoreE = 0;
                        MoveUpE = 0;
                        MoveDownE = 1;
                        MoveLeftE = 0;
                        break;

                case MoveWestE:
                        MoveCoreE = 0;
                        MoveUpE = 0;
                        MoveDownE = 0;
                        MoveLeftE = 1;
                        break;

                case MoveLocalE:
                        MoveCoreE = 1;
                        MoveUpE = 0;
                        MoveDownE = 0;
                        MoveLeftE = 0;
                        break;

                }
        }

        void routing_logic::outputW(){
                switch (state_signalW){
                case IDLEW:
                        MoveCoreW = 0;
                        MoveUpW = 0;
                        MoveDownW = 0;
                        MoveRightW = 0;
                        break;

                case MoveNorthW:
                        MoveCoreW = 0;
                        MoveUpW = 1;
                        MoveDownW = 0;
                        MoveRightW = 0;
                        break;

                case MoveSouthW:
                        MoveCoreW = 0;
                        MoveUpW = 0;
                        MoveDownW = 1;
                        MoveRightW = 0;
                        break;

                case MoveEastW:
                        MoveCoreW = 0;
                        MoveUpW = 0;
                        MoveDownW = 0;
                        MoveRightW = 1;
                        break;

                case MoveLocalW:
                        MoveCoreW = 1;
                        MoveUpW = 0;
                        MoveDownW = 0;
                        MoveRightW = 0;
                        break;
                default:
                        MoveCoreW = 0;
                        MoveUpW = 0;
                        MoveDownW = 0;
                        MoveRightW = 0;
                }

        }

        void routing_logic::Data_Request(){
                if (Type_headerL.read() == "11")
                        Data_RequestL = 0;
                else
                        Data_RequestL = 1;

                if (Type_headerN.read() == "11")
                        Data_RequestN = 0;
                else
                        Data_RequestN = 1;

                if (Type_headerS.read() == "11")
                        Data_RequestS = 0;
                else
                        Data_RequestS = 1;

                if (Type_headerE.read() == "11")
                        Data_RequestE = 0;
                else
                        Data_RequestE = 1;

                if (Type_headerW.read() == "11")
                        Data_RequestW = 0;
                else
                        Data_RequestW = 1;
                }
```

1.4 Switch Allocator

When there are input data streams from various input port for the same output port, there is a collision. To avoid collision Arbiter or switching allocator is used. It arbitrates between various inputs. There are various

Arbitration techniques of which Round Robin is one of them. Here we have used Round Robin arbitration between the input ports at which data is available.
Credit based Flow control is implemented here, In this scheme Upstream router stores the count of each downstream router. It decrements the credit count when flit is forwarded and increments the counter when the buffer of downstream router is freed in which case it receives a credit in signal.
Note: When count =0 the buffer is full hence the data is not forwarded.
There are total 5 FSMs implemented for each output port. The FSM for Local output port is shown in fig. 3.1.
The block diagram for Switching Allocator is shown in fig. 3.2

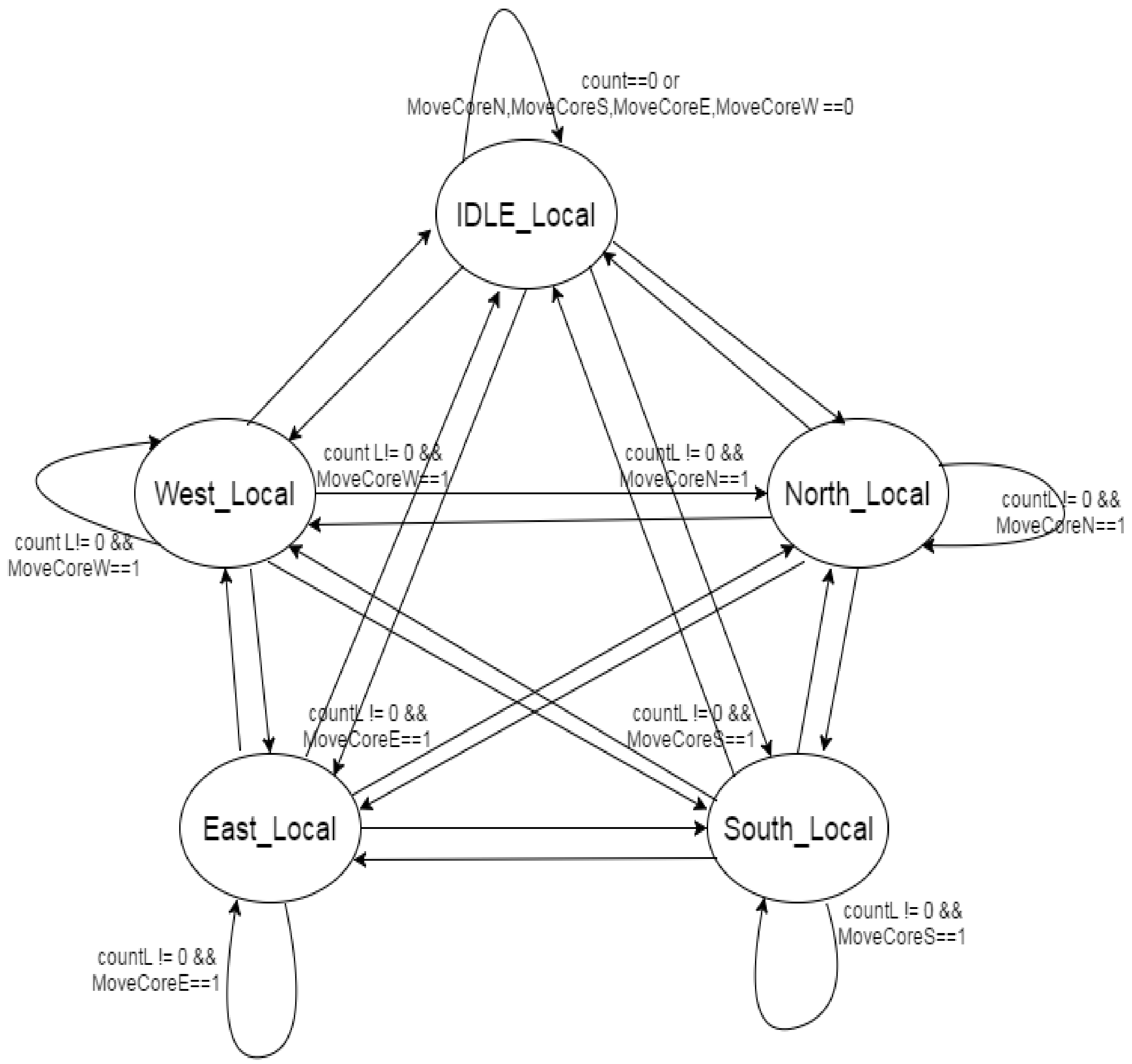


Fig. 3.1 FSM of Switching Allocator for Local output port

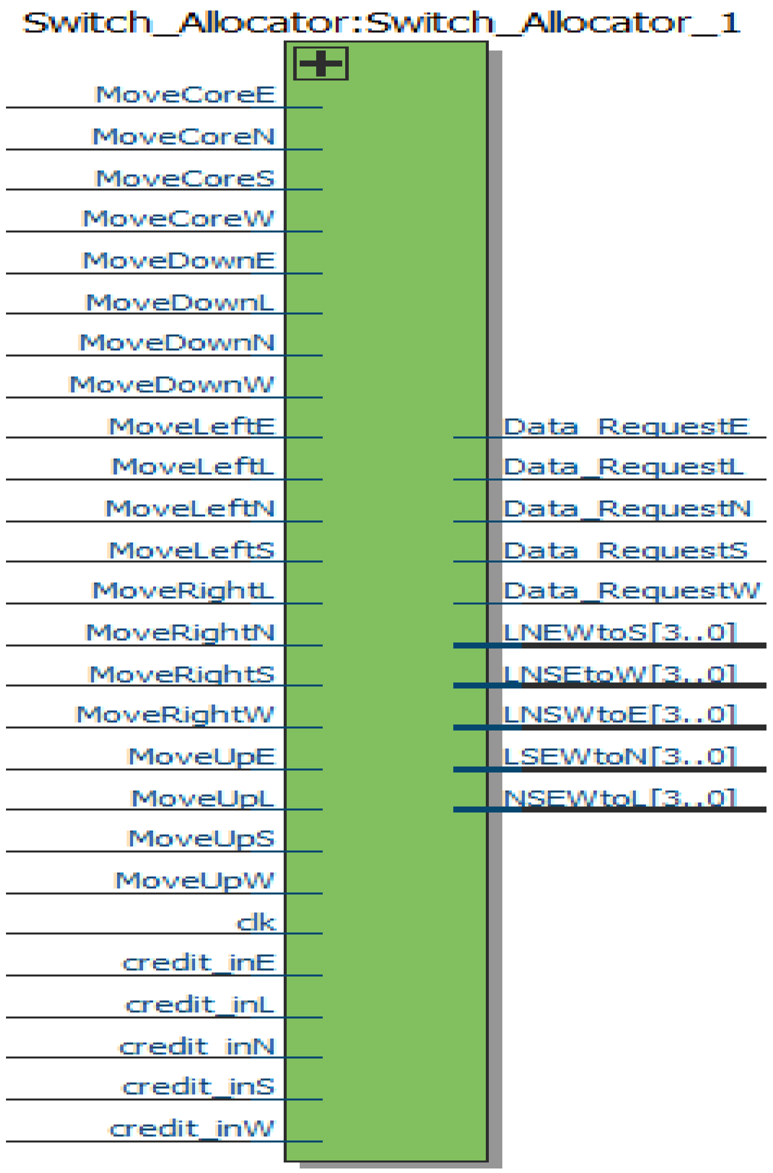


Fig. 3.2 Block Diagram of Switching Allocator

CODE:

//Switching_Allocator.h

```
#include "systemc.h"

enum vm5_state {
        IDLELocal, LocalN, LocalS, LocalE, LocalW
};
enum vm6_state {
        IDLEUP=5, UpCore, UpS, UpE, UpW
};
enum vm7_state {
        IDLEDN=10, DNCore, DNN, DNE, DNW
};
enum vm8_state {
        IDLER=15, RCore, RN, RS, RW
```

```
};
enum vm9_state {
        IDLELeft=20, LCore, LN, LS, LE
};

SC_MODULE(Switching_Allocator){
        sc_in_clk clock;
        sc_in<bool> credit_inL, credit_inN, credit_inS, credit_inE, credit_inW;
        sc_in<bool> MoveUpL, MoveDownL, MoveRightL, MoveLeftL;
        sc_in<bool> MoveCoreN, MoveDownN, MoveRightN, MoveLeftN;
        sc_in<bool> MoveCoreS, MoveUpS, MoveRightS, MoveLeftS;
        sc_in<bool> MoveCoreE, MoveUpE, MoveDownE, MoveLeftE;
        sc_in<bool> MoveCoreW, MoveUpW, MoveDownW, MoveRightW;

        sc_out<sc_bv<4> > NSEWtoL;
        sc_out<sc_bv<4> > LSEWtoN;
        sc_out<sc_bv<4> > LNEWtoS;
        sc_out<sc_bv<4> > LNSWtoE;
        sc_out<sc_bv<4> > LNSEtoW;
        sc_out<bool> Data_RequestL, Data_RequestN, Data_RequestS, Data_RequestE, Data_RequestW;

        sc_signal<bool> Data_RequestLN, Data_RequestLS, Data_RequestLE, Data_RequestLW;
        sc_signal<bool> Data_RequestNL, Data_RequestNS, Data_RequestNE, Data_RequestNW;
        sc_signal<bool> Data_RequestSL, Data_RequestSN, Data_RequestSE, Data_RequestSW;
        sc_signal<bool> Data_RequestEL, Data_RequestEN, Data_RequestES, Data_RequestEW;
        sc_signal<bool> Data_RequestWL, Data_RequestWN, Data_RequestWS, Data_RequestWE;

        sc_uint<3> countL = 6, countN = 6, countS = 6, countE = 6, countW = 6;

        sc_signal<vm5_state>state_signalL;
        sc_signal<vm5_state>next_stateL; //Local ouput state
        sc_signal<vm6_state>state_signalUp;
        sc_signal<vm6_state>next_stateUP; //UP ouput state
        sc_signal<vm7_state>state_signalDN;
        sc_signal<vm7_state>next_stateDN; //DN ouput state
        sc_signal<vm8_state>state_signalRight;
        sc_signal<vm8_state>next_stateRight; //Right ouput state
        sc_signal<vm9_state>state_signalLeft;
        sc_signal<vm9_state> next_stateLeft; //Left ouput state

        sc_trace_file *wf;

        void getnextstateL();
        void getnextstateUP();
        void getnextstateDN();
        void getnextstateRight();
        void getnextstateLeft();
        void setnextstate();
        void outputL();
        void outputUP();
        void outputDN();
        void outputRight();
        void outputLeft();
        void DataRequest();

        SC_CTOR(Switching_Allocator){
                wf = sc_create_vcd_trace_file("Switching_Allocator");
                sc_trace(wf, clock, "clock");
                sc_trace(wf, credit_inL, "credit_inL");
                sc_trace(wf, credit_inN, "credit_inN");
                sc_trace(wf, credit_inS, "credit_inS");
                sc_trace(wf, credit_inE, "credit_inE");
                sc_trace(wf, credit_inW, "credit_inW");

                sc_trace(wf, MoveUpL, "MoveUpL");
                sc_trace(wf, MoveDownL, "MoveDownL");
                sc_trace(wf, MoveRightL, "MoveRightL");
                sc_trace(wf, MoveLeftL, " MoveLeftL");

                sc_trace(wf, MoveDownN, "MoveDownN");
                sc_trace(wf, MoveRightN, "MoveRightN");
                sc_trace(wf, MoveLeftN, "MoveLeftN");
                sc_trace(wf, MoveCoreN, " MoveCoreN");

                sc_trace(wf, MoveUpS, "MoveUpSL");
                sc_trace(wf, MoveRightS, "MoveRightS");
                sc_trace(wf, MoveLeftS, "MoveLeftS");
                sc_trace(wf, MoveCoreS, "MoveCoreS");

                sc_trace(wf, MoveUpE, "MoveUpE");
                sc_trace(wf, MoveDownE, "MoveDownE");
                sc_trace(wf, MoveLeftE, "MoveLeftE");
                sc_trace(wf, MoveCoreE, " MoveCoreE");

                sc_trace(wf, MoveUpW, "MoveUpW");
                sc_trace(wf, MoveDownW, "MoveDownW");
                sc_trace(wf, MoveRightW, "MoveRightW");
                sc_trace(wf, MoveCoreW, "MoveCoreW");

                sc_trace(wf, NSEWtoL, "NSEWtoL");
                sc_trace(wf, LSEWtoN, "LSEWtoN");
                sc_trace(wf, LNEWtoS, "LNEWtoS");
                sc_trace(wf, LNSWtoE, "LNSWtoE");
                sc_trace(wf, LNSEtoW, "LNSEtoW");

                sc_trace(wf, Data_RequestL, "Data_RequestL");
                sc_trace(wf, Data_RequestN, "Data_RequestN");
                sc_trace(wf, Data_RequestS, "Data_RequestS");
                sc_trace(wf, Data_RequestE, "Data_RequestE");
                sc_trace(wf, Data_RequestW, "Data_RequestW");

                sc_trace(wf, state_signalL, "state_signalL");
                sc_trace(wf, state_signalUp, "state_signalUp");
                sc_trace(wf, state_signalDN, "state_signalDN");
                sc_trace(wf, state_signalRight, "state_signalRight");
                sc_trace(wf, state_signalLeft, "state_signalLeft");

                SC_METHOD(getnextstateL);
                sensitive << state_signalL << MoveCoreN << MoveCoreS << MoveCoreE << MoveCoreW;//
<< countL;
                SC_METHOD(getnextstateUP);
                sensitive << state_signalUp << MoveUpL << MoveUpS << MoveUpE << MoveUpW;// <<
countN;
                SC_METHOD(getnextstateDN);
                sensitive << state_signalDN << MoveDownL << MoveDownN << MoveDownE << MoveDownW;//
<< countS;
                SC_METHOD(getnextstateRight);
                sensitive << state_signalRight << MoveRightL << MoveRightN << MoveRightS <<
MoveRightW;// << countE;
                SC_METHOD(getnextstateLeft);
                sensitive << state_signalLeft << MoveLeftL << MoveLeftN << MoveLeftS <<
MoveLeftE;// << countW;
                SC_METHOD(setnextstate);
                sensitive << clock.neg();
                SC_METHOD(outputL);
                sensitive << state_signalL;
                SC_METHOD(outputUP);
                sensitive << state_signalUp;
                SC_METHOD(outputDN);
                sensitive << state_signalDN;
                SC_METHOD(outputRight);
                sensitive << state_signalRight;
                SC_METHOD(outputLeft);
                sensitive << state_signalLeft;
                SC_METHOD(DataRequest);
                sensitive << Data_RequestLN << Data_RequestLS << Data_RequestLE << Data_RequestLW
<< Data_RequestNL << Data_RequestNS << Data_RequestNE << Data_RequestNW << Data_RequestSL <<
Data_RequestSN << Data_RequestSE << Data_RequestSW << Data_RequestEL << Data_RequestEN <<
Data_RequestES << Data_RequestEW << Data_RequestWL << Data_RequestWN << Data_RequestWS <<
Data_RequestWE;
        }
        ~Switching_Allocator(){
                sc_close_vcd_trace_file(wf);
        }
        };
```

//Switching_Allocator.cpp

```
#include "Switching_Allocator.h"

void Switching_Allocator::getnextstateL(){
        switch (state_signalL){
        case IDLELocal:
                if (countL >1)  {
                        if (MoveCoreN.read() == 1)
                                next_stateL = LocalN;
                        else if (MoveCoreS.read() == 1)
                                next_stateL = LocalS;
                        else if (MoveCoreE.read() == 1)
                                next_stateL = LocalE;
                        else if (MoveCoreW.read() == 1)
                                next_stateL = LocalW;
                        else
                                next_stateL = IDLELocal;
                }
                else
                        next_stateL = IDLELocal;
                break;

        case LocalN:
                if (countL >1){
                        if (MoveCoreN.read() == 1)
                                next_stateL = LocalN;
                        else if (MoveCoreS.read() == 1)
                                next_stateL = LocalS;
                        else if (MoveCoreE.read() == 1)
                                next_stateL = LocalE;
                        else if (MoveCoreW.read() == 1)
                                next_stateL = LocalW;
                        else
                                next_stateL = IDLELocal;
                }
                else
                        next_stateL = IDLELocal;
                break;

        case LocalS:
                if (countL >1){
                        if (MoveCoreS.read() == 1)
                                next_stateL = LocalS;
                        else if (MoveCoreN.read() == 1)
                                next_stateL = LocalN;
                        else if (MoveCoreE.read() == 1)
                                next_stateL = LocalE;
                        else if (MoveCoreW.read() == 1)
                                next_stateL = LocalW;
                        else
                                next_stateL = IDLELocal;
                }
                else
                        next_stateL = IDLELocal;
                break;

        case LocalE:
                if (countL >1)  {
                        if (MoveCoreE.read() == 1)
                                next_stateL = LocalE;
                        else if (MoveCoreN.read() == 1)
                                next_stateL = LocalN;
                        else if (MoveCoreS.read() == 1)
                                next_stateL = LocalS;
                        else if (MoveCoreW.read() == 1)
                                next_stateL = LocalW;

                        else
                                next_stateL = IDLELocal;
                }
                else
                        next_stateL = IDLELocal;
                break;

        case LocalW:
                if (countL >1) {
                        if (MoveCoreW.read() == 1)
                                next_stateL = LocalW;
                        else if (MoveCoreN.read() == 1)
                                next_stateL = LocalN;
                        else if (MoveCoreS.read() == 1)
                                next_stateL = LocalS;
                        else if (MoveCoreE.read() == 1)
                                next_stateL = LocalE;
                        else
                                next_stateL = IDLELocal;
                }
                else
                        next_stateL = IDLELocal;
                break;

        default:
                next_stateL = IDLELocal;
                break;
        }
}

void Switching_Allocator::getnextstateUP(){
        switch (state_signalUp){
        case IDLEUP:
                if (countN >1){
                        if (MoveUpL.read() == 1)
                                next_stateUP = UpCore;
                        else if (MoveUpS.read() == 1)
                                next_stateUP = UpS;
                        else if (MoveUpE.read() == 1)
                                next_stateUP = UpE;
                        else if (MoveUpW.read() == 1)
                                next_stateUP = UpW;
                        else
                                next_stateUP = IDLEUP;
                }
                else
                        next_stateUP = IDLEUP;
                break;

        case UpCore:
                if (countN >1){
                        if (MoveUpL.read() == 1)
                                next_stateUP = UpCore;
                        else if (MoveUpS.read() == 1)
                                next_stateUP = UpS;
                        else if (MoveUpE.read() == 1)
                                next_stateUP = UpE;
                        else if (MoveUpW.read() == 1)
                                next_stateUP = UpW;
                        else
                                next_stateUP = IDLEUP;
                }
                else
                        next_stateUP = IDLEUP;
                break;

        case UpS:
                if (countN >1) {
                        if (MoveUpS.read() == 1)
                                next_stateUP = UpS;
                        else if (MoveUpL.read() == 1)
                                next_stateUP = UpCore;
                        else if (MoveUpE.read() == 1)
                                next_stateUP = UpE;
                        else if (MoveUpW.read() == 1)
                                next_stateUP = UpW;
                        else
                                next_stateUP = IDLEUP;
                }
                else
                        next_stateUP = IDLEUP;
                break;

        case UpE:
                if (countN >1) {
                        if (MoveUpE.read() == 1)
                                next_stateUP = UpE;
                        else if (MoveUpL.read() == 1)
                                next_stateUP = UpCore;
                        else if (MoveUpS.read() == 1)
                                next_stateUP = UpS;
                        else if (MoveUpW.read() == 1)
                                next_stateUP = UpW;
                        else
                                next_stateUP = IDLEUP;
                }
                else
                        next_stateUP = IDLEUP;
                break;

        case UpW:
                if (countN >1){
                        if (MoveUpW.read() == 1)
                                next_stateUP = UpW;
                        else if (MoveUpL.read() == 1)
                                next_stateUP = UpCore;
                        else if (MoveUpS.read() == 1)
                                next_stateUP = UpS;
                        else if (MoveUpE.read() == 1)
                                next_stateUP = UpE;
                        else
                                next_stateUP = IDLEUP;
                }
                else
                        next_stateUP = IDLEUP;
                break;

        default:
                next_stateUP = IDLEUP;
                break;
        }
}

void Switching_Allocator::getnextstateDN(){
        switch (state_signalDN){
        case IDLEDN:
                if (countS >1){
                        if (MoveDownL.read() == 1)
                                next_stateDN = DNCore;
                        else if (MoveDownN.read() == 1)
                                next_stateDN = DNN;
                        else if (MoveDownE.read() == 1)
                                next_stateDN = DNE;
                        else if (MoveDownW.read() == 1)
                                next_stateDN = DNW;
                        else
                                next_stateDN = IDLEDN;
                }
                else
```

```
                        next_stateDN = IDLEDN;
                break;

        case DNCore:
                if (countS >1) {
                        if (MoveDownL.read() == 1)
                                next_stateDN = DNCore;
                        else if (MoveDownN.read() == 1)
                                next_stateDN = DNN;
                        else if (MoveDownE.read() == 1)
                                next_stateDN = DNE;
                        else if (MoveDownW.read() == 1)
                                next_stateDN = DNW;
                        else
                                next_stateDN = IDLEDN;
                }
                else
                        next_stateDN = IDLEDN;
                break;

        case DNN:
                if (countS >1){
                        if (MoveDownN.read() == 1)
                                next_stateDN = DNN;
                        else if (MoveDownL.read() == 1)
                                next_stateDN = DNCore;
                        else if (MoveDownE.read() == 1)
                                next_stateDN = DNE;
                        else if (MoveDownW.read() == 1)
                                next_stateDN = DNW;
                        else
                                next_stateDN = IDLEDN;
                }
                else
                        next_stateDN = IDLEDN;
                break;

        case DNE:
                if (countS >1) {
                        if (MoveDownE.read() == 1)
                                next_stateDN = DNE;
                        else if (MoveDownL.read() == 1)
                                next_stateDN = DNCore;
                        else if (MoveDownN.read() == 1)
                                next_stateDN = DNN;
                        else if (MoveDownW.read() == 1)
                                next_stateDN = DNW;
                        else
                                next_stateDN = IDLEDN;
                }
                else
                        next_stateDN = IDLEDN;
                break;

        case DNW:
                if (countS >1){
                        if (MoveDownW.read() == 1)
                                next_stateDN = DNW;
                        else if (MoveDownL.read() == 1)
                                next_stateDN = DNCore;
                        else if (MoveDownN.read() == 1)
                                next_stateDN = DNN;
                        else if (MoveDownE.read() == 1)
                                next_stateDN = DNE;
                        else
                                next_stateDN = IDLEDN;
                }
                else
                        next_stateDN = IDLEDN;
                break;
        default:
                next_stateDN = IDLEDN;
                break;
        }
}

void Switching_Allocator::getnextstateRight(){
        switch (state_signalRight){
        case IDLER:
                if (countE >1) {
                        if (MoveRightL.read() == 1)
                                next_stateRight = RCore;
                        else if (MoveRightN.read() == 1)
                                next_stateRight = RN;
                        else if (MoveRightS.read() == 1)
                                next_stateRight = RS;
                        else if (MoveRightW.read() == 1)
                                next_stateRight = RW;
                        else
                                next_stateRight = IDLER;
                }
                else
                        next_stateRight = IDLER;
                break;

        case RCore:
                if (countE >1){
                        if (MoveRightL.read() == 1)
                                next_stateRight = RCore;
                        else if (MoveRightN.read() == 1)
                                next_stateRight = RN;
                        else if (MoveRightS.read() == 1)
                                next_stateRight = RS;
                        else if (MoveRightW.read() == 1)
                                next_stateRight = RW;
                        else
                                next_stateRight = IDLER;
                }
                else
                        next_stateRight = IDLER;
                break;

        case RN:
                if (countE >1) {
                        if (MoveRightN.read() == 1)
                                next_stateRight = RN;
                        else if (MoveRightL.read() == 1)
                                next_stateRight = RCore;
                        else if (MoveRightS.read() == 1)
                                next_stateRight = RS;
                        else if (MoveRightW.read() == 1)
                                next_stateRight = RW;
                        else
                                next_stateRight = IDLER;
                }
                else
                        next_stateRight = IDLER;
                break;

        case RS:
                if (countE >1){
                        if (MoveRightS.read() == 1)
                                next_stateRight = RS;
                        else if (MoveRightL.read() == 1)
                                next_stateRight = RCore;
                        else if (MoveRightN.read() == 1)
                                next_stateRight = RN;
                        else if (MoveRightW.read() == 1)
                                next_stateRight = RW;
                        else
                                next_stateRight = IDLER;
                }
                else
                        next_stateRight = IDLER;
                break;

        case RW:
                if (countE >1) {
                        if (MoveRightW.read() == 1)
                                next_stateRight = RW;
                        else if (MoveRightL.read() == 1)
                                next_stateRight = RCore;
                        else if (MoveRightN.read() == 1)
                                next_stateRight = RN;
                        else if (MoveRightS.read() == 1)
                                next_stateRight = RS;
                        else
                                next_stateRight = IDLER;
                }
                else
                        next_stateRight = IDLER;
                break;

        default:
                next_stateRight = IDLER;
                break;
        }
}

void Switching_Allocator::getnextstateLeft(){
        switch (state_signalLeft){
        case IDLELeft:
                if (countW >1) {
                        if (MoveLeftL.read() == 1)
                                next_stateLeft = LCore;
                        else if (MoveLeftN.read() == 1)
                                next_stateLeft = LN;
                        else if (MoveLeftS.read() == 1)
                                next_stateLeft = LS;
                        else if (MoveLeftE.read() == 1)
                                next_stateLeft = LE;
                        else
                                next_stateLeft = IDLELeft;
                }
                else
                        next_stateLeft = IDLELeft;
                break;

        case LCore:
                if (countW >1) {
                        if (MoveLeftL.read() == 1)
                                next_stateLeft = LCore;
                        else if (MoveLeftN.read() == 1)
                                next_stateLeft = LN;
                        else if (MoveLeftS.read() == 1)
                                next_stateLeft = LS;
                        else if (MoveLeftE.read() == 1)
                                next_stateLeft = LE;
                        else
                                next_stateLeft = IDLELeft;
                }
                else
                        next_stateLeft = IDLELeft;
                break;

        case LN:
                if (countW >1){
                        if (MoveLeftN == 1)
                                next_stateLeft = LN;
                        else if (MoveLeftL == 1)
                                next_stateLeft = LCore;
                        else if (MoveLeftS == 1)
```

```
                                next_stateLeft = LS;
                        else if (MoveLeftE == 1)
                                next_stateLeft = LE;
                        else
                                next_stateLeft = IDLELeft;
                }
                else
                        next_stateLeft = IDLELeft;
                break;

        case LS:
                if (countW >1){
                        if (MoveLeftS.read() == 1)
                                next_stateLeft = LS;
                        else if (MoveLeftL.read() == 1)
                                next_stateLeft = LCore;
                        else if (MoveLeftN.read() == 1)
                                next_stateLeft = LN;
                        else if (MoveLeftE.read() == 1)
                                next_stateLeft = LE;
                        else
                                next_stateLeft = IDLELeft;
                }
                else
                        next_stateLeft = IDLELeft;
                break;

        case LE:
                if (countW >1){
                        if (MoveLeftE.read() == 1)
                                next_stateLeft = LE;
                        else if (MoveLeftL.read() == 1)
                                next_stateLeft = LCore;
                        else if (MoveLeftN.read() == 1)
                                next_stateLeft = LN;
                        else if (MoveLeftS.read() == 1)
                                next_stateLeft = LS;
                        else
                                next_stateLeft = IDLELeft;
                }
                else
                        next_stateLeft = IDLELeft;
                break;

        default:
                next_stateLeft = IDLELeft;
                break;
        }

}

void Switching_Allocator::setnextstate(){
        state_signalL = next_stateL;
        state_signalUp = next_stateUP;
        state_signalDN = next_stateDN;
        state_signalRight = next_stateRight;
        state_signalLeft = next_stateLeft;

        if (next_stateL.read() == IDLELocal){
                if (credit_inL.read() == 1)
                        countL = countL + 1;
        }
        else {
                if (credit_inL.read() != 1)
                        countL = countL - 1;
        }

        if (next_stateUP.read() == IDLEUP) {
                if (credit_inN.read() == 1)
                        countN = countN + 1;
        }
        else {
                if (credit_inN.read() != 1)
                        countN = countN - 1;
        }

        if (next_stateDN.read() == IDLEDN) {
                if (credit_inS.read() == 1)
                        countS = countS + 1;
        }
        else {
                if (credit_inS.read() != 1)
                        countS = countS - 1;
        }

        if (next_stateRight.read() == IDLER) {
                if (credit_inE.read() == 1)
                        countE = countE + 1;
        }
        else {
                if (credit_inE.read() != 1)
                        countE = countE - 1;
        }


        if (next_stateLeft.read() == IDLELeft) {
                if (credit_inW.read() == 1)
                        countW = countW + 1;
        }
        else {
                if (credit_inW.read() != 1)
                        countW = countW - 1;
        }
}

void Switching_Allocator::outputL(){
        switch (state_signalL){
        case IDLELocal:
                Data_RequestLN = 0;
                Data_RequestLS = 0;
                Data_RequestLE = 0;
                Data_RequestLW = 0;

                NSEWtoL = "0000";
                break;

        case LocalN:
                Data_RequestLN = 1;
                Data_RequestLS = 0;
                Data_RequestLE = 0;
                Data_RequestLW = 0;

                NSEWtoL = "1000";
                break;
        case LocalS:
                Data_RequestLN = 0;
                Data_RequestLS = 1;
                Data_RequestLE = 0;
                Data_RequestLW = 0;

                NSEWtoL = "0100";
                break;
        case LocalE:
                Data_RequestLN = 0;
                Data_RequestLS = 0;
                Data_RequestLE = 1;
                Data_RequestLW = 0;

                NSEWtoL = "0010";
                break;
        case LocalW:
                Data_RequestLN = 0;
                Data_RequestLS = 0;
                Data_RequestLE = 0;
                Data_RequestLW = 1;

                NSEWtoL = "0001";
                break;
        default:
                Data_RequestLN = 0;
                Data_RequestLS = 0;
                Data_RequestLE = 0;
                Data_RequestLW = 0;

                NSEWtoL = "0000";
                break;
        }
}

void Switching_Allocator::outputUP(){
        switch (state_signalUp){
        case IDLEUP:
                Data_RequestNL = 0;
                Data_RequestNS = 0;
                Data_RequestNE = 0;
                Data_RequestNW = 0;

                LSEWtoN = "0000";
                break;

        case UpCore:
                Data_RequestNL = 1;
                Data_RequestNS = 0;
                Data_RequestNE = 0;
                Data_RequestNW = 0;

                LSEWtoN = "1000";
                break;

        case UpS:
                Data_RequestNL = 0;
                Data_RequestNS = 1;
                Data_RequestNE = 0;
                Data_RequestNW = 0;

                LSEWtoN = "0100";
                break;

        case UpE:
                Data_RequestNL = 0;
                Data_RequestNS = 0;
                Data_RequestNE = 1;
                Data_RequestNW = 0;

                LSEWtoN = "0010";
                break;

        case UpW:
                Data_RequestNL = 0;
                Data_RequestNS = 0;
                Data_RequestNE = 0;
                Data_RequestNW = 1;

                LSEWtoN = "0001";
                break;
        default:
                Data_RequestNL = 0;
                Data_RequestNS = 0;
                Data_RequestNE = 0;
                Data_RequestNW = 0;

                LSEWtoN = "0000";
                break;
        }

}

void Switching_Allocator::outputDN(){
        switch (state_signalDN){
        case IDLEDN:
                Data_RequestSL = 0;
                Data_RequestSN = 0;
                Data_RequestSE = 0;
                Data_RequestSW = 0;

                LNEWtoS = "0000";
                break;

        case DNCore:
                Data_RequestSL = 1;
                Data_RequestSN = 0;
                Data_RequestSE = 0;
                Data_RequestSW = 0;

                LNEWtoS = "1000";
                break;

        case DNN:
                Data_RequestSL = 0;
                Data_RequestSN = 1;
                Data_RequestSE = 0;
                Data_RequestSW = 0;

                LNEWtoS = "0100";
                break;

        case DNE:
                Data_RequestSL = 0;
                Data_RequestSN = 0;
                Data_RequestSE = 1;
                Data_RequestSW = 0;

                LNEWtoS = "0010";
                break;

        case DNW:
                Data_RequestSL = 0;
                Data_RequestSN = 0;
                Data_RequestSE = 0;
                Data_RequestSW = 1;

                LNEWtoS = "0001";
                break;

        default:
                Data_RequestSL = 0;
                Data_RequestSN = 0;
                Data_RequestSE = 0;
                Data_RequestSW = 0;

                LNEWtoS = "0000";
                break;
        }

}

void Switching_Allocator::outputRight(){
        switch (state_signalRight){
        case IDLER:
                Data_RequestEL = 0;
                Data_RequestEN = 0;
                Data_RequestES = 0;
                Data_RequestEW = 0;
                LNSWtoE = "0000";
                break;

        case RCore:
                Data_RequestEL = 1;
                Data_RequestEN = 0;
                Data_RequestES = 0;
                Data_RequestEW = 0;

                LNSWtoE = "1000";
                break;

        case RN:
                Data_RequestEL = 0;
                Data_RequestEN = 1;
                Data_RequestES = 0;
                Data_RequestEW = 0;

                LNSWtoE = "0100";
                break;

        case RS:
                Data_RequestEL = 0;
                Data_RequestEN = 0;
                Data_RequestES = 1;
                Data_RequestEW = 0;

                LNSWtoE = "0010";
                break;

        case RW:
                Data_RequestEL = 0;
                Data_RequestEN = 0;
                Data_RequestES = 0;
                Data_RequestEW = 1;

                LNSWtoE = "0001";
                break;

        default:
                Data_RequestEL = 0;
                Data_RequestEN = 0;
                Data_RequestES = 0;
                Data_RequestEW = 0;

                LNSWtoE = "0000";
                break;

        }
}

void Switching_Allocator::outputLeft(){
        switch (state_signalLeft){
        case IDLELeft:
                Data_RequestWL = 0;
                Data_RequestWN = 0;
                Data_RequestWS = 0;
                Data_RequestWE = 0;

                LNSEtoW = "0000";
                break;

        case LCore:
                Data_RequestWL = 1;
                Data_RequestWN = 0;
                Data_RequestWS = 0;
                Data_RequestWE = 0;

                LNSEtoW = "1000";
                break;
        case LN:
                Data_RequestWL = 0;
                Data_RequestWN = 1;
                Data_RequestWS = 0;
                Data_RequestWE = 0;

                LNSEtoW = "0100";
                break;

        case LS:
                Data_RequestWL = 0;
                Data_RequestWN = 0;
                Data_RequestWS = 1;
                Data_RequestWE = 0;

                LNSEtoW = "0010";
                break;

        case LE:
                Data_RequestWL = 0;
                Data_RequestWN = 0;
                Data_RequestWS = 0;
                Data_RequestWE = 1;

                LNSEtoW = "0001";
                break;

        default:
                Data_RequestWL = 0;
                Data_RequestWN = 0;
                Data_RequestWS = 0;
                Data_RequestWE = 0;

                LNSEtoW = "0000";
                break;
        }
}

void Switching_Allocator::DataRequest(){
        Data_RequestL = Data_RequestNL | Data_RequestSL | Data_RequestEL | Data_RequestWL;
        Data_RequestN = Data_RequestLN | Data_RequestSN | Data_RequestEN | Data_RequestWN;
        Data_RequestS = Data_RequestLS | Data_RequestNS | Data_RequestES | Data_RequestWS;
        Data_RequestE = Data_RequestLE | Data_RequestNE | Data_RequestSE | Data_RequestWE;
        Data_RequestW = Data_RequestLW | Data_RequestNW | Data_RequestSW | Data_RequestEW;
}
```

1.5 Crossbar Switch

The implementation of crossbar switch is pretty straight forward. It receives a total of 20 input pins from the Switching Allocator. Local_to_North, Local_to_South… and all the possible combination.
Based on this input signal it forwards the data from input to output port.

The Block diagram of crossbar switch is shown in fig. 4.1

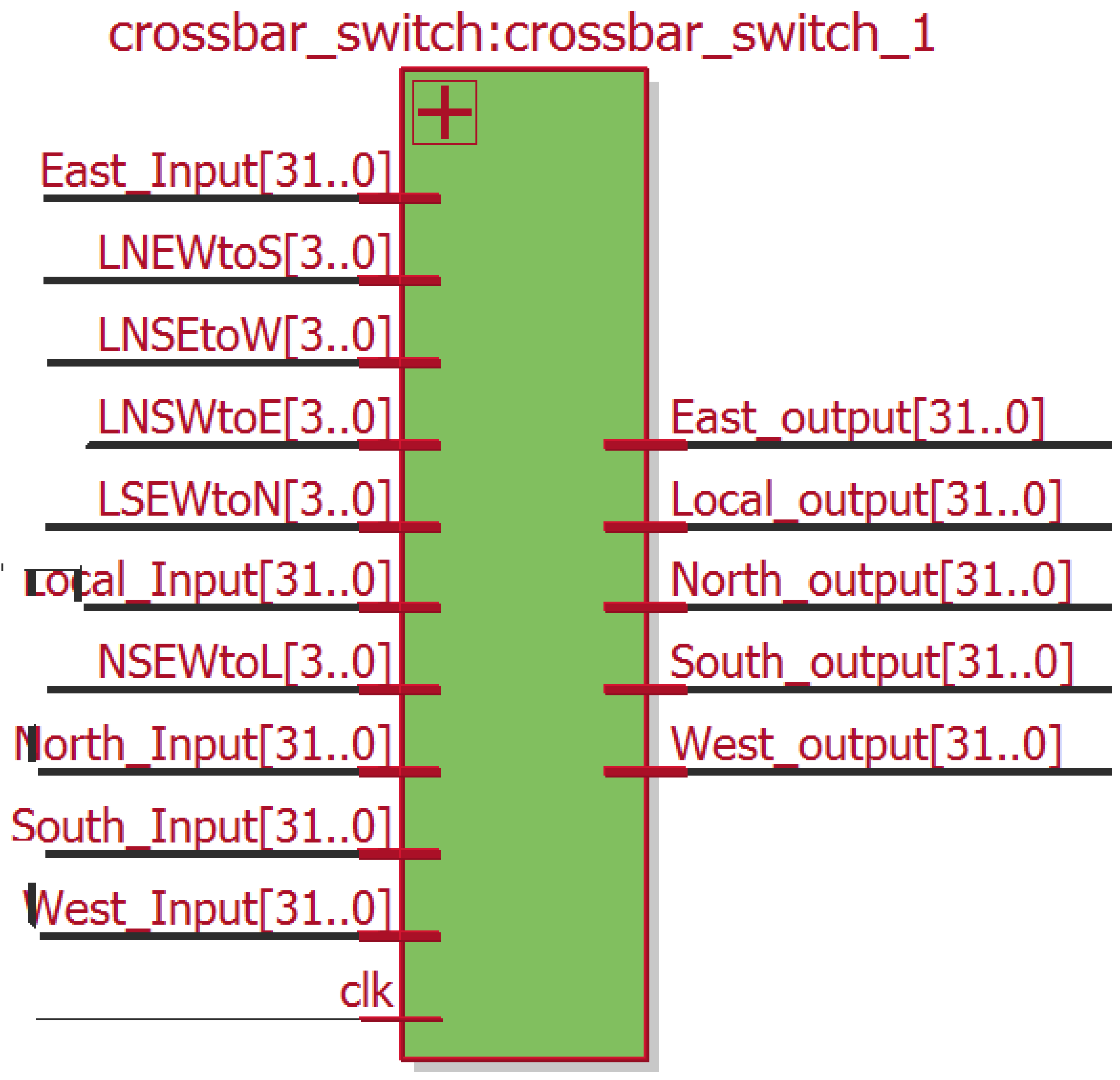


Fig. 4.1 Block Diagram of Crossbar Switch

CODE:

//crossbar_switch.h

```
#include "systemc.h"

SC_MODULE(crossbar_switch){
        sc_in_clk clock;
        sc_in<sc_bv<32> >Local_Input, North_Input, South_Input, East_Input, West_Input;
        sc_in<sc_bv<4> > NSEWtoL, LSEWtoN, LNEWtoS, LNSWtoE, LNSEtoW;
        sc_out<sc_bv<32> > Local_output, North_output, South_output, East_output, West_output;

        sc_bv<32> Local_output_temp, North_output_temp, South_output_temp, East_output_temp,
West_output_temp;
        sc_bv<32> temp1, temp2, temp3, temp4, temp5;

        sc_trace_file *wf;
        void cross_switch();
        void output_flits();

        SC_CTOR(crossbar_switch){
                wf = sc_create_vcd_trace_file("crossbar_switch");
                sc_trace(wf, clock, "clock");
                sc_trace(wf, Local_Input, "Local_Input");
                sc_trace(wf, North_Input, "North_Input");
                sc_trace(wf, South_Input, "South_Input");
                sc_trace(wf, East_Input, "East_Input");
                sc_trace(wf, West_Input, "West_Input");
                sc_trace(wf, NSEWtoL, "NSEWtoL");
                sc_trace(wf, LSEWtoN, "LSEWtoN");
                sc_trace(wf, LNEWtoS, "LNEWtoS");
                sc_trace(wf, LNSWtoE, "LNSWtoE");
                sc_trace(wf, LNSEtoW, "LNSEtoW");
                sc_trace(wf, Local_output, "Local_output");
                sc_trace(wf, North_output, "North_output");
                sc_trace(wf, South_output, "South_output");
                sc_trace(wf, East_output, "East_output");
                sc_trace(wf, West_output, " West_output");

                sc_trace(wf, Local_output_temp, "Local_output_temp");
                sc_trace(wf, North_output_temp, "North_output_temp");
                sc_trace(wf, South_output_temp, "South_output_temp");
                sc_trace(wf, East_output_temp, "East_output_temp");
                sc_trace(wf, West_output_temp, " West_output_temp");

                sc_trace(wf, temp1, "temp1");
                sc_trace(wf, temp2, "temp2");
                sc_trace(wf, temp3, "temp3");
                sc_trace(wf, temp4, "temp4");
                sc_trace(wf, temp5, "temp5");

                SC_METHOD(cross_switch);
                sensitive << clock.pos();
                SC_METHOD(output_flits);
                sensitive << clock.pos();
        }
        ~crossbar_switch(){
                sc_close_vcd_trace_file(wf);
        }
};
```

//crossbar_switch.cpp

```
#include "crossbar_switch.h"

void crossbar_switch::cross_switch() {
        if (NSEWtoL.read() == "1000"){
                Local_output_temp = North_Input;
        }
        else if (NSEWtoL.read() == "0100") {
                Local_output_temp = South_Input;
        }
        else if (NSEWtoL.read() == "0010") {
                Local_output_temp = East_Input;
        }
        else if (NSEWtoL.read() == "0001") {
                Local_output_temp = West_Input;
        }
        else
                Local_output_temp = 0x00000000;

        if (LSEWtoN.read() == "1000") {
                North_output_temp = Local_Input;
        }
        else if (LSEWtoN.read() == "0100") {
                North_output_temp = South_Input;
        }
        else if (LSEWtoN.read() == "0010") {
                North_output_temp = East_Input;
        }
        else if (LSEWtoN.read() == "0001") {
                North_output_temp = West_Input;
        }
        else
                North_output_temp = 0x00000000;

        if (LNEWtoS.read() == "1000") {
                South_output_temp = Local_Input;
        }
        else if (LNEWtoS.read() == "0100") {
                South_output_temp = North_Input;
        }
        else if (LNEWtoS.read() == "0010") {
                South_output_temp = East_Input;
        }
        else if (LNEWtoS.read() == "0001") {
                South_output_temp = West_Input;
        }
        else
                South_output_temp = 0x00000000;

        if (LNSWtoE.read() == "1000"){
                East_output_temp = Local_Input;
        }
        else if (LNSWtoE.read() == "0100") {
                East_output_temp = North_Input;
        }
        else if (LNSWtoE.read() == "0010") {
                East_output_temp = South_Input;
        }
        else if (LNSWtoE.read() == "0001") {
                East_output_temp = West_Input;
        }
        else
                East_output_temp = 0x00000000;

        if (LNSEtoW.read() == "1000") {
                West_output_temp = Local_Input;
        }
        else if (LNSEtoW.read() == "0100") {
                West_output_temp = North_Input;
        }
        else if (LNSEtoW.read() == "0010") {
                West_output_temp = South_Input;
        }
        else if (LNSEtoW.read() == "0001") {
                West_output_temp = East_Input;
        }
        else
                West_output_temp = 0x00000000;
}

void crossbar_switch::output_flits(){
        if (Local_output_temp.range(31, 30) == "01"){
                Local_output = temp1;
                temp1 = Local_output_temp;
        }
        else {
                Local_output = Local_output_temp;
                temp1 = 0x00000000;
        }

        if (North_output_temp.range(31, 30) == "01"){
                North_output = temp2;
                temp2 = North_output_temp;
        }
        else {
                North_output = North_output_temp;
                temp2 = 0x00000000;
        }


        if (South_output_temp.range(31, 30) == "01"){
                South_output = temp3;
                temp3 = South_output_temp;
        }
        else {
                South_output = South_output_temp;
                temp3 = 0x00000000;
        }

        if (East_output_temp.range(31, 30) == "01"){
                East_output = temp4;
                temp4 = East_output_temp;
        }
        else {
                East_output = East_output_temp;
                temp4 = 0x00000000;
        }

        if (West_output_temp.range(31, 30) == "01"){
                West_output = temp5;
                temp5 = West_output_temp;
        }
        else {
                West_output = West_output_temp;
                temp5 = 0x00000000;
        }
}
```

2. Router_16

It is used to connect the overall circuit, i.e. port map all the 16 NOC routers

CODE:

//router_16.h

```
#include "router.h"

SC_MODULE(router_16){
        sc_in_clk clock;
        sc_in<sc_bv<2> > XPresent1, YPresent1, XPresent2, YPresent2, XPresent3, YPresent3,
XPresent4, YPresent4, XPresent5, YPresent5, XPresent6, YPresent6, XPresent7, YPresent7,
XPresent8, YPresent8, XPresent9, YPresent9, XPresent10, YPresent10, XPresent11, YPresent11,
XPresent12, YPresent12, XPresent13, YPresent13, XPresent14, YPresent14, XPresent15, YPresent15,
XPresent16, YPresent16;
        sc_in<sc_bv<32> > Core_in_00, Core_in_10, Core_in_20, Core_in_30;
        sc_in<sc_bv<32> > Core_in_01, Core_in_11, Core_in_21, Core_in_31;
        sc_in<sc_bv<32> > Core_in_02, Core_in_12, Core_in_22, Core_in_32;
        sc_in<sc_bv<32> > Core_in_03, Core_in_13, Core_in_23, Core_in_33;

        sc_in<bool> Core_Credit_in_00, Core_Credit_in_10, Core_Credit_in_20, Core_Credit_in_30;
        sc_in<bool> Core_Credit_in_01, Core_Credit_in_11, Core_Credit_in_21, Core_Credit_in_31;
        sc_in<bool> Core_Credit_in_02, Core_Credit_in_12, Core_Credit_in_22, Core_Credit_in_32;
        sc_in<bool> Core_Credit_in_03, Core_Credit_in_13, Core_Credit_in_23, Core_Credit_in_33;

        sc_out<sc_bv<32> > Core_out_00, Core_out_10, Core_out_20, Core_out_30;
        sc_out<sc_bv<32> > Core_out_01, Core_out_11, Core_out_21, Core_out_31;
        sc_out<sc_bv<32> > Core_out_02, Core_out_12, Core_out_22, Core_out_32;
        sc_out<sc_bv<32> > Core_out_03, Core_out_13, Core_out_23, Core_out_33;

        sc_out<bool> Core_Credit_out_00, Core_Credit_out_10, Core_Credit_out_20,
Core_Credit_out_30;
        sc_out<bool> Core_Credit_out_01, Core_Credit_out_11, Core_Credit_out_21,
Core_Credit_out_31;
        sc_out<bool> Core_Credit_out_02, Core_Credit_out_12, Core_Credit_out_22,
Core_Credit_out_32;
        sc_out<bool> Core_Credit_out_03, Core_Credit_out_13, Core_Credit_out_23,
Core_Credit_out_33;

        sc_signal<sc_bv<32> > flit_00_10, flit_10_20, flit_20_30, flit_30_20, flit_20_10,
flit_10_00;
        sc_signal<sc_bv<32> > flit_01_11, flit_11_21, flit_21_31, flit_31_21, flit_21_11,
flit_11_01;
        sc_signal<sc_bv<32> > flit_02_12, flit_12_22, flit_22_32, flit_32_22, flit_22_12,
flit_12_02;
        sc_signal<sc_bv<32> > flit_03_13, flit_13_23, flit_23_33, flit_33_23, flit_23_13,
flit_13_03;

        sc_signal<sc_bv<32> > flit_00_01, flit_01_02, flit_02_03, flit_03_02, flit_02_01,
flit_01_00;
        sc_signal<sc_bv<32> > flit_10_11, flit_11_12, flit_12_13, flit_13_12, flit_12_11,
flit_11_10;
        sc_signal<sc_bv<32> > flit_20_21, flit_21_22, flit_22_23, flit_23_22, flit_22_21,
flit_21_20;
        sc_signal<sc_bv<32> > flit_30_31, flit_31_32, flit_32_33, flit_33_32, flit_32_31,
flit_31_30;

        sc_signal<bool>  Credit_00_10, Credit_10_20, Credit_20_30, Credit_30_20, Credit_20_10,
Credit_10_00;
        sc_signal<bool>  Credit_01_11, Credit_11_21, Credit_21_31, Credit_31_21, Credit_21_11,
Credit_11_01;
        sc_signal<bool>  Credit_02_12, Credit_12_22, Credit_22_32, Credit_32_22, Credit_22_12,
Credit_12_02;
        sc_signal<bool>  Credit_03_13, Credit_13_23, Credit_23_33, Credit_33_23, Credit_23_13,
Credit_13_03;

        sc_signal<bool>  Credit_00_01, Credit_01_02, Credit_02_03, Credit_03_02, Credit_02_01,
Credit_01_00;
        sc_signal<bool>  Credit_10_11, Credit_11_12, Credit_12_13, Credit_13_12, Credit_12_11,
Credit_11_10;
        sc_signal<bool>  Credit_20_21, Credit_21_22, Credit_22_23, Credit_23_22, Credit_22_21,
Credit_21_20;
        sc_signal<bool>  Credit_30_31, Credit_31_32, Credit_32_33, Credit_33_32, Credit_32_31,
Credit_31_30;

        //Dummy signals
        sc_signal<sc_bv<32> > dummy_input_00_North, dummy_input_10_North, dummy_input_20_North,
dummy_input_30_North;
        sc_signal<sc_bv<32> > dummy_input_00_West, dummy_input_01_West, dummy_input_02_West,
dummy_input_03_West;
        sc_signal<sc_bv<32> > dummy_input_30_East, dummy_input_31_East, dummy_input_32_East,
dummy_input_33_East;
        sc_signal<sc_bv<32> > dummy_input_03_South, dummy_input_13_South, dummy_input_23_South,
dummy_input_33_South;

        sc_signal<sc_bv<32> > dummy_output_00_North, dummy_output_10_North,
dummy_output_20_North, dummy_output_30_North;
        sc_signal<sc_bv<32> > dummy_output_00_West, dummy_output_01_West, dummy_output_02_West,
dummy_output_03_West;
        sc_signal<sc_bv<32> > dummy_output_30_East, dummy_output_31_East, dummy_output_32_East,
dummy_output_33_East;
        sc_signal<sc_bv<32> > dummy_output_03_South, dummy_output_13_South,
dummy_output_23_South, dummy_output_33_South;

        sc_signal<bool> dummy_Credit_in_00_North, dummy_Credit_in_10_North,
dummy_Credit_in_20_North, dummy_Credit_in_30_North;
        sc_signal<bool> dummy_Credit_in_00_West, dummy_Credit_in_01_West,
dummy_Credit_in_02_West, dummy_Credit_in_03_West;
```

```
        sc_signal<bool> dummy_Credit_in_30_East, dummy_Credit_in_31_East,
dummy_Credit_in_32_East, dummy_Credit_in_33_East;
        sc_signal<bool> dummy_Credit_in_03_South, dummy_Credit_in_13_South,
dummy_Credit_in_23_South, dummy_Credit_in_33_South;

        sc_signal<bool> dummy_Credit_out_00_North, dummy_Credit_out_10_North,
dummy_Credit_out_20_North, dummy_Credit_out_30_North;
        sc_signal<bool> dummy_Credit_out_00_West, dummy_Credit_out_01_West,
dummy_Credit_out_02_West, dummy_Credit_out_03_West;
        sc_signal<bool> dummy_Credit_out_30_East, dummy_Credit_out_31_East,
dummy_Credit_out_32_East, dummy_Credit_out_33_East;
        sc_signal<bool> dummy_Credit_out_03_South, dummy_Credit_out_13_South,
dummy_Credit_out_23_South, dummy_Credit_out_33_South;

        //pointers
        router *Router_00, *Router_10, *Router_20, *Router_30;
        router *Router_01, *Router_11, *Router_21, *Router_31;
        router *Router_02, *Router_12, *Router_22, *Router_32;
        router *Router_03, *Router_13, *Router_23, *Router_33;

        SC_CTOR(router_16){
                Router_00 = new router("Router1");
                (*Router_00)(clock, XPresent1, YPresent1, Core_in_00, dummy_input_00_North,
flit_01_00, flit_10_00, dummy_input_00_West, Core_Credit_in_00, dummy_Credit_in_00_North,
Credit_01_00, Credit_10_00, dummy_Credit_in_00_West, Core_Credit_out_00,
dummy_Credit_out_00_North, Credit_00_01, Credit_00_10, dummy_Credit_out_00_West, Core_out_00,
dummy_output_00_North, flit_00_01, flit_00_10, dummy_output_00_West);
                Router_10 = new router("Router2");
                (*Router_10)(clock, XPresent2, YPresent2, Core_in_10, dummy_input_10_North,
flit_11_10, flit_20_10, flit_00_10, Core_Credit_in_10, dummy_Credit_in_10_North, Credit_11_10,
Credit_20_10, Credit_00_10, Core_Credit_out_10, dummy_Credit_out_10_North, Credit_10_11,
Credit_10_20, Credit_10_00, Core_out_10, dummy_output_10_North, flit_10_11, flit_10_20,
flit_10_00);
                Router_20 = new router("Router3");
                (*Router_20)(clock, XPresent3, YPresent3, Core_in_20, dummy_input_20_North,
flit_21_20, flit_30_20, flit_10_20, Core_Credit_in_20, dummy_Credit_in_20_North, Credit_21_20,
Credit_30_20, Credit_10_20, Core_Credit_out_20, dummy_Credit_out_20_North, Credit_20_21,
Credit_20_30, Credit_20_10, Core_out_20, dummy_output_20_North, flit_20_21, flit_20_30,
flit_20_10);
                Router_30 = new router("Router4");
                (*Router_30)(clock, XPresent4, YPresent4, Core_in_30, dummy_input_30_North,
flit_31_30, dummy_input_30_East, flit_20_30, Core_Credit_in_30, dummy_Credit_in_30_North,
Credit_31_30, dummy_Credit_in_30_East, Credit_20_30, Core_Credit_out_30,
dummy_Credit_out_30_North, Credit_30_31, dummy_Credit_out_30_East, Credit_30_20, Core_out_30,
dummy_output_30_North, flit_30_31, dummy_output_30_East, flit_30_20);
                Router_01 = new router("Router5");
                (*Router_01)(clock, XPresent5, YPresent5, Core_in_01, flit_00_01, flit_02_01,
flit_11_01, dummy_input_01_West, Core_Credit_in_01, Credit_00_01, Credit_02_01, Credit_11_01,
dummy_Credit_in_01_West, Core_Credit_out_01, Credit_01_00, Credit_01_02, Credit_01_11,
dummy_Credit_out_01_West, Core_out_01, flit_01_00, flit_01_02, flit_01_11, dummy_output_01_West);
                Router_11 = new router("Router6");
                (*Router_11)(clock, XPresent6, YPresent6, Core_in_11, flit_10_11, flit_12_11,
flit_21_11, flit_01_11, Core_Credit_in_11, Credit_10_11, Credit_12_11, Credit_21_11,
Credit_01_11, Core_Credit_out_11, Credit_11_10, Credit_11_12, Credit_11_21, Credit_11_01,
Core_out_11, flit_11_10, flit_11_12, flit_11_21, flit_11_01);
                Router_21 = new router("Router7");
                (*Router_21)(clock, XPresent7, YPresent7, Core_in_21, flit_20_21, flit_22_21,
flit_31_21, flit_11_21, Core_Credit_in_21, Credit_20_21, Credit_22_21, Credit_31_21,
Credit_11_21, Core_Credit_out_21, Credit_21_20, Credit_21_22, Credit_21_31, Credit_21_11,
Core_out_21, flit_21_20, flit_21_22, flit_21_31, flit_21_11);
                Router_31 = new router("Router8");
                (*Router_31)(clock, XPresent8, YPresent8, Core_in_31, flit_30_31, flit_32_31,
dummy_input_31_East, flit_21_31, Core_Credit_in_31, Credit_30_31, Credit_32_31,
dummy_Credit_in_31_East, Credit_21_31, Core_Credit_out_31, Credit_31_30, Credit_31_32,
dummy_Credit_out_31_East, Credit_31_21, Core_out_31, flit_31_30, flit_31_32,
dummy_output_31_East, flit_31_21);
                Router_02 = new router("Router9");
                (*Router_02)(clock, XPresent9, YPresent9, Core_in_02, flit_01_02, flit_03_02,
flit_12_02, dummy_input_02_West, Core_Credit_in_02, Credit_01_02, Credit_03_02, Credit_12_02,
dummy_Credit_in_02_West, Core_Credit_out_02, Credit_02_01, Credit_02_03, Credit_02_12,
dummy_Credit_out_02_West, Core_out_02, flit_02_01, flit_02_03, flit_02_12, dummy_output_02_West);
```

```
		Router_12 = new router("Router10");
		(*Router_12)(clock, XPresent10, YPresent10, Core_in_12, flit_11_12, flit_13_12,
flit_22_12, flit_02_12, Core_Credit_in_12, Credit_11_12, Credit_13_12, Credit_22_12,
Credit_02_12, Core_Credit_out_12, Credit_12_11, Credit_12_13, Credit_12_22, Credit_12_02,
Core_out_12, flit_12_11, flit_12_13, flit_12_22, flit_12_02);
		Router_22 = new router("Router11");
		(*Router_22)(clock, XPresent11, YPresent11, Core_in_22, flit_21_22, flit_23_22,
flit_32_22, flit_12_22, Core_Credit_in_22, Credit_21_22, Credit_23_22, Credit_32_22,
Credit_12_22, Core_Credit_out_22, Credit_22_21, Credit_22_23, Credit_22_32, Credit_22_12,
Core_out_22, flit_22_21, flit_22_23, flit_22_32, flit_22_12);
		Router_32 = new router("Router12");
		(*Router_32)(clock, XPresent12, YPresent12, Core_in_32, flit_31_32, flit_33_32,
dummy_input_32_East, flit_22_32, Core_Credit_in_32, Credit_31_32, Credit_33_32,
dummy_Credit_in_32_East, Credit_22_32, Core_Credit_out_32, Credit_32_31, Credit_32_33,
dummy_Credit_out_32_East, Credit_32_22, Core_out_32, flit_32_31, flit_32_33,
dummy_output_32_East, flit_32_22);
		Router_03 = new router("Router13");
		(*Router_03)(clock, XPresent13, YPresent13, Core_in_03, flit_02_03,
dummy_input_03_South, flit_13_03, dummy_input_03_West, Core_Credit_in_03, Credit_02_03,
dummy_Credit_in_03_South, Credit_13_03, dummy_Credit_in_03_West, Core_Credit_out_03,
Credit_03_02, dummy_Credit_out_03_South, Credit_03_13, dummy_Credit_out_03_West, Core_out_03,
flit_03_02, dummy_output_03_South, flit_03_13, dummy_output_03_West);
		Router_13 = new router("Router14");
		(*Router_13)(clock, XPresent14, YPresent14, Core_in_13, flit_12_13,
dummy_input_13_South, flit_23_13, flit_03_13, Core_Credit_in_13, Credit_12_13,
dummy_Credit_in_13_South, Credit_23_13, Credit_03_13, Core_Credit_out_13, Credit_13_12,
dummy_Credit_out_13_South, Credit_13_23, Credit_13_03, Core_out_13, flit_13_12,
dummy_output_13_South, flit_13_23, flit_13_03);
		Router_23 = new router("Router15");
		(*Router_23)(clock, XPresent15, YPresent15, Core_in_23, flit_22_23,
dummy_input_23_South, flit_33_23, flit_13_23, Core_Credit_in_23, Credit_22_23,
dummy_Credit_in_23_South, Credit_33_23, Credit_13_23, Core_Credit_out_23, Credit_23_22,
dummy_Credit_out_23_South, Credit_23_33, Credit_23_13, Core_out_23, flit_23_22,
dummy_output_23_South, flit_23_33, flit_23_13);
		Router_33 = new router("Router16");
		(*Router_33)(clock, XPresent16, YPresent16, Core_in_33, flit_32_33,
dummy_input_33_South, dummy_input_33_East, flit_23_33, Core_Credit_in_33, Credit_32_33,
dummy_Credit_in_33_South, dummy_Credit_in_33_East, Credit_23_33, Core_Credit_out_33,
Credit_33_32, dummy_Credit_out_33_South, dummy_Credit_out_33_East, Credit_33_23, Core_out_33,
flit_33_32, dummy_output_33_South, dummy_output_33_East, flit_33_23);
	}
	~router_16(){
		delete Router_00; delete Router_10; delete Router_20; delete Router_30;
		delete Router_01; delete Router_11; delete Router_21; delete Router_31;
		delete Router_02; delete Router_12; delete Router_22; delete Router_32;
		delete Router_03; delete Router_13; delete Router_23; delete Router_33;
	}
};
```

3. Router testbench

Drives the router with appropriate inputs specially to check what happens when collision occurs, other such cases.

//router_16_tb.cpp

```
#include "router_16.h"

int sc_main(int argc, char* argv[]) {
	sc_clock  clock("clock", 80, SC_US);
	sc_signal<sc_bv<2> > XPresent1, YPresent1, XPresent2, YPresent2, XPresent3, YPresent3,
XPresent4, YPresent4, XPresent5, YPresent5, XPresent6, YPresent6, XPresent7, YPresent7,
XPresent8, YPresent8, XPresent9, YPresent9, XPresent10, YPresent10, XPresent11, YPresent11,
XPresent12, YPresent12, XPresent13, YPresent13, XPresent14, YPresent14, XPresent15, YPresent15,
XPresent16, YPresent16;
	sc_signal<sc_bv<32> > Core_in_00, Core_in_10, Core_in_20, Core_in_30;
	sc_signal<sc_bv<32> > Core_in_01, Core_in_11, Core_in_21, Core_in_31;
```

```
        sc_signal<sc_bv<32> > Core_in_02, Core_in_12, Core_in_22, Core_in_32;
        sc_signal<sc_bv<32> > Core_in_03, Core_in_13, Core_in_23, Core_in_33;

        sc_signal<bool> Core_Credit_in_00, Core_Credit_in_10, Core_Credit_in_20,
Core_Credit_in_30;
        sc_signal<bool> Core_Credit_in_01, Core_Credit_in_11, Core_Credit_in_21,
Core_Credit_in_31;
        sc_signal<bool> Core_Credit_in_02, Core_Credit_in_12, Core_Credit_in_22,
Core_Credit_in_32;
        sc_signal<bool> Core_Credit_in_03, Core_Credit_in_13, Core_Credit_in_23,
Core_Credit_in_33;

        sc_signal<sc_bv<32> > Core_out_00, Core_out_10, Core_out_20, Core_out_30;
        sc_signal<sc_bv<32> > Core_out_01, Core_out_11, Core_out_21, Core_out_31;
        sc_signal<sc_bv<32> > Core_out_02, Core_out_12, Core_out_22, Core_out_32;
        sc_signal<sc_bv<32> > Core_out_03, Core_out_13, Core_out_23, Core_out_33;

        sc_signal<bool> Core_Credit_out_00, Core_Credit_out_10, Core_Credit_out_20,
Core_Credit_out_30;
        sc_signal<bool> Core_Credit_out_01, Core_Credit_out_11, Core_Credit_out_21,
Core_Credit_out_31;
        sc_signal<bool> Core_Credit_out_02, Core_Credit_out_12, Core_Credit_out_22,
Core_Credit_out_32;
        sc_signal<bool> Core_Credit_out_03, Core_Credit_out_13, Core_Credit_out_23,
Core_Credit_out_33;

        // Connect the DUT
        router_16 DUT("Wave");
        DUT << clock << XPresent1 << YPresent1 << XPresent2 << YPresent2 << XPresent3 <<
YPresent3 << XPresent4 << YPresent4 << XPresent5 << YPresent5 << XPresent6 << YPresent6 <<
XPresent7 << YPresent7 << XPresent8 << YPresent8 << XPresent9 << YPresent9 << XPresent10 <<
YPresent10 << XPresent11 << YPresent11 << XPresent12 << YPresent12 << XPresent13 << YPresent13 <<
XPresent14 << YPresent14 << XPresent15 << YPresent15 << XPresent16 << YPresent16<< Core_in_00 <<
Core_in_10 << Core_in_20 << Core_in_30
                << Core_in_01 << Core_in_11 << Core_in_21 << Core_in_31
                << Core_in_02 << Core_in_12 << Core_in_22 << Core_in_32
                << Core_in_03 << Core_in_13 << Core_in_23 << Core_in_33
                << Core_Credit_in_00 << Core_Credit_in_10 << Core_Credit_in_20 <<
Core_Credit_in_30
                << Core_Credit_in_01 << Core_Credit_in_11 << Core_Credit_in_21 <<
Core_Credit_in_31
                << Core_Credit_in_02 << Core_Credit_in_12 << Core_Credit_in_22 <<
Core_Credit_in_32
                << Core_Credit_in_03 << Core_Credit_in_13 << Core_Credit_in_23 <<
Core_Credit_in_33
                << Core_out_00 << Core_out_10 << Core_out_20 << Core_out_30
                << Core_out_01 << Core_out_11 << Core_out_21 << Core_out_31
                << Core_out_02 << Core_out_12 << Core_out_22 << Core_out_32
                << Core_out_03 << Core_out_13 << Core_out_23 << Core_out_33
                << Core_Credit_out_00 << Core_Credit_out_10 << Core_Credit_out_20 <<
Core_Credit_out_30
                << Core_Credit_out_01 << Core_Credit_out_11 << Core_Credit_out_21 <<
Core_Credit_out_31
                << Core_Credit_out_02 << Core_Credit_out_12 << Core_Credit_out_22 <<
Core_Credit_out_32
                << Core_Credit_out_03 << Core_Credit_out_13 << Core_Credit_out_23 <<
Core_Credit_out_33;

        sc_start(0, SC_US);

        // Open VCD file
        sc_trace_file *wf = sc_create_vcd_trace_file("wave");
        // Dump the desired signals
        sc_trace(wf, clock, "clock");
        sc_trace(wf, Core_in_00, "Core_in_00");
        sc_trace(wf, Core_in_10, "Core_in_10");
        sc_trace(wf, Core_in_20, "Core_in_20");
        sc_trace(wf, Core_in_30, "Core_in_30");

        sc_trace(wf, Core_in_01, "Core_in_01");
        sc_trace(wf, Core_in_11, "Core_in_11");
        sc_trace(wf, Core_in_21, "Core_in_21");
        sc_trace(wf, Core_in_31, "Core_in_31");

        sc_trace(wf, Core_in_02, "Core_in_02");
        sc_trace(wf, Core_in_12, "Core_in_12");
        sc_trace(wf, Core_in_22, "Core_in_22");
        sc_trace(wf, Core_in_32, "Core_in_32");

        sc_trace(wf, Core_in_03, "Core_in_03");
        sc_trace(wf, Core_in_13, "Core_in_13");
        sc_trace(wf, Core_in_23, "Core_in_23");
        sc_trace(wf, Core_in_33, "Core_in_33");

        sc_trace(wf, Core_Credit_in_00, "Core_Credit_in_00");
        sc_trace(wf, Core_Credit_in_10, "Core_Credit_in_10");
        sc_trace(wf, Core_Credit_in_20, "Core_Credit_in_20");
        sc_trace(wf, Core_Credit_in_30, "Core_Credit_in_30");

        sc_trace(wf, Core_Credit_in_01, "Core_Credit_in_01");
        sc_trace(wf, Core_Credit_in_11, "Core_Credit_in_11");
        sc_trace(wf, Core_Credit_in_21, "Core_Credit_in_21");
        sc_trace(wf, Core_Credit_in_31, "Core_Credit_in_31");

        sc_trace(wf, Core_Credit_in_02, "Core_Credit_in_02");
        sc_trace(wf, Core_Credit_in_12, "Core_Credit_in_12");
        sc_trace(wf, Core_Credit_in_22, "Core_Credit_in_22");
        sc_trace(wf, Core_Credit_in_32, "Core_Credit_in_32");

        sc_trace(wf, Core_Credit_in_03, "Core_Credit_in_03");
        sc_trace(wf, Core_Credit_in_13, "Core_Credit_in_13");
        sc_trace(wf, Core_Credit_in_23, "Core_Credit_in_23");
        sc_trace(wf, Core_Credit_in_33, "Core_Credit_in_33");

        sc_trace(wf, Core_out_00, "Core_out_00");
        sc_trace(wf, Core_out_10, "Core_out_10");
        sc_trace(wf, Core_out_20, "Core_out_20");
        sc_trace(wf, Core_out_30, "Core_out_30");

        sc_trace(wf, Core_out_01, "Core_out_01");
        sc_trace(wf, Core_out_11, "Core_out_11");
        sc_trace(wf, Core_out_21, "Core_out_21");
        sc_trace(wf, Core_out_31, "Core_out_31");

        sc_trace(wf, Core_out_02, "Core_out_02");
        sc_trace(wf, Core_out_12, "Core_out_12");
        sc_trace(wf, Core_out_22, "Core_out_22");
        sc_trace(wf, Core_out_32, "Core_out_32");

        sc_trace(wf, Core_out_03, "Core_out_03");
        sc_trace(wf, Core_out_13, "Core_out_13");
        sc_trace(wf, Core_out_23, "Core_out_23");
        sc_trace(wf, Core_out_33, "Core_out_33");

        sc_trace(wf, Core_Credit_out_00, "Core_Credit_out_00");
        sc_trace(wf, Core_Credit_out_10, "Core_Credit_out_10");
        sc_trace(wf, Core_Credit_out_20, "Core_Credit_out_20");
        sc_trace(wf, Core_Credit_out_30, "Core_Credit_out_30");

        sc_trace(wf, Core_Credit_out_01, "Core_Credit_out_01");
        sc_trace(wf, Core_Credit_out_11, "Core_Credit_out_11");
        sc_trace(wf, Core_Credit_out_21, "Core_Credit_out_21");
        sc_trace(wf, Core_Credit_out_31, "Core_Credit_out_31");

        sc_trace(wf, Core_Credit_out_02, "Core_Credit_out_02");
        sc_trace(wf, Core_Credit_out_12, "Core_Credit_out_12");
        sc_trace(wf, Core_Credit_out_22, "Core_Credit_out_22");
        sc_trace(wf, Core_Credit_out_32, "Core_Credit_out_32");

        sc_trace(wf, Core_Credit_out_03, "Core_Credit_out_03");
        sc_trace(wf, Core_Credit_out_13, "Core_Credit_out_13");
        sc_trace(wf, Core_Credit_out_23, "Core_Credit_out_23");
        sc_trace(wf, Core_Credit_out_33, "Core_Credit_out_33");

        sc_trace(wf, DUT.flit_00_10, "flit_00_10");
        sc_trace(wf, DUT.flit_10_20, "flit_10_20");
        sc_trace(wf, DUT.flit_20_30, "flit_20_30");
        sc_trace(wf, DUT.flit_30_31, "flit_30_31");
        sc_trace(wf, DUT.flit_31_32, "flit_31_32");
        sc_trace(wf, DUT.flit_32_33, "flit_32_33");
        sc_trace(wf, DUT.flit_11_21, "flit_11_21");
        sc_trace(wf, DUT.flit_21_31, "flit_21_31");

        sc_trace(wf, DUT.Router_31->input_portW->input_flit, "input_flitW");
        sc_trace(wf, DUT.Router_31->input_portW->Virtual_buffer, "Virtual_bufferW");
        sc_trace(wf, DUT.Router_31->input_portW->output_flit, "output_flitW");
        sc_trace(wf, DUT.Router_31->input_portW->Data_Request_routing, "Data_Request_routingW");
        sc_trace(wf, DUT.Router_31->input_portW->Data_Request_switching,
"Data_Request_switchingW");
        sc_trace(wf, DUT.Router_31->input_portW->Data_Request, "Data_RequestW");
        sc_trace(wf, DUT.Router_31->routing->MoveDownW, "MoveDownW");
        sc_trace(wf, DUT.Router_31->Switching->Data_RequestSW, "Data_RequestSW");
        sc_trace(wf, DUT.Router_31->Switching->countS, "countS");
        sc_trace(wf, DUT.Router_31->crossbar->West_Input, "West_Input8");
        sc_trace(wf, DUT.Router_31->crossbar->South_output_temp, "South_output_temp8");
        sc_trace(wf, DUT.Router_31->crossbar->South_output, "South_output8");
        sc_trace(wf, DUT.Router_31->crossbar->North_Input, "North_Input8");

        sc_trace(wf, DUT.Router_31->input_portN->input_flit, "input_flitN");
        sc_trace(wf, DUT.Router_31->input_portN->Virtual_buffer, "Virtual_bufferN");
        sc_trace(wf, DUT.Router_31->input_portN->output_flit, "output_flitN");
        sc_trace(wf, DUT.Router_31->input_portN->Data_Request_routing, "Data_Request_routingN");
        sc_trace(wf, DUT.Router_31->input_portN->Data_Request_switching,
"Data_Request_switchingN");
        sc_trace(wf, DUT.Router_31->input_portN->Data_Request, "Data_RequestN");
        sc_trace(wf, DUT.Router_31->routing->MoveDownN, "MoveDownN");
        sc_trace(wf, DUT.Router_31->Switching->Data_RequestSN, "Data_RequestSN");

        sc_trace(wf, DUT.Router_32->input_portN->input_flit, "input_flitN12");

        sc_trace(wf, DUT.Router_32->input_portN->Virtual_buffer, "Virtual_bufferN12");
        sc_trace(wf, DUT.Router_32->input_portN->output_flit, "output_flitN12");
        sc_trace(wf, DUT.Router_32->input_portN->credit_out, "credit_outN12");
        sc_trace(wf, DUT.Router_32->routing->MoveCoreN, "MoveCoreN12");
        sc_trace(wf, DUT.Router_32->Switching->Data_RequestLN, "Data_RequestLN12");
        sc_trace(wf, DUT.Router_32->crossbar->North_Input, "cross_North_Input");
        sc_trace(wf, DUT.Router_32->crossbar->Local_output_temp, "cross_Local_output_temp");
        sc_trace(wf, DUT.Router_32->crossbar->Local_output, "cross_Local_output");




        // Initialize all variables
        XPresent1 = "00"; YPresent1 = "00";
        XPresent2 = "01"; YPresent2 = "00";
        XPresent3 = "10"; YPresent3 = "00";
        XPresent4 = "11"; YPresent4 = "00";
        XPresent5 = "00"; YPresent5 = "01";
        XPresent6 = "01"; YPresent6 = "01";
        XPresent7 = "10"; YPresent7 = "01";
        XPresent8 = "11"; YPresent8 = "01";
        XPresent9 = "00"; YPresent9 = "10";
        XPresent10 = "01"; YPresent10 = "10";
        XPresent11 = "10"; YPresent11 = "10";
        XPresent12 = "11"; YPresent12 = "10";
        XPresent13 = "00"; YPresent13 = "11";
        XPresent14 = "01"; YPresent14 = "11";
        XPresent15 = "10"; YPresent15 = "11";
        XPresent16 = "11"; YPresent16 = "11";


        Core_in_00 = 0x40FABCDE; //from 00 to 33
        Core_in_03 = 0x43CABCDE; //from 03 to 30
        Core_in_11 = 0x45EFFFAA; //from 11 t0 32
        Core_in_12 = 0x46012345; //from 12 to 00
        Core_in_23 = 0x00000000;

        sc_start(80, SC_US);

        Core_in_00 = 0x856789AB;
        Core_in_03 = 0x865677AC;
        Core_in_11 = 0x804FBBAA;
        Core_in_12 = 0x88912345;
        Core_in_23 = 0x4B2ABEFA; //from 23 to 02

        sc_start(80, SC_US);

        Core_in_00 = 0x91234567;
        Core_in_03 = 0x912377AC;
        Core_in_11 = 0x945FFAAA;
        Core_in_12 = 0x96123457;
        Core_in_23 = 0x884ABAFF;

        sc_start(80, SC_US);

        Core_in_00 = 0xAABCDEF1;
        Core_in_03 = 0xA67897AC;
        Core_in_11 = 0xC012FBCA;
        Core_in_12 = 0xCAAFF123;
        Core_in_23 = 0x944ABAAA;

        sc_start(80, SC_US);

        Core_in_00 = 0xB3456789;
        Core_in_03 = 0xB65677AC;
        Core_in_11 = 0x00000000;
        Core_in_12 = 0x00000000;

        Core_in_23 = 0xC213464A;
        Core_Credit_in_33 = 1;
        Core_Credit_in_32 = 1;
        sc_start(80, SC_US);

        Core_in_00 = 0xC7891412;
        Core_in_03 = 0xC65677AC;
        Core_in_23 = 0x00000000;


        sc_start(80, SC_US);

        Core_in_00 = 0x00000000;
        Core_in_03 = 0x00000000;


        sc_start(240, SC_US);


        sc_start(1600, SC_US);
        Core_Credit_in_33 = 0;
        Core_Credit_in_32 = 0;
        sc_start(720, SC_US);

        sc_close_vcd_trace_file(wf);
        return 0;// Terminate simulation

}
```