**Assignment 1**

SYSC 5104

**Network On Chip**

Samaneh Ahmadvand (101021682)

Navinder Kaur Brar (101026788)

Department of Systems and Computer Engineering

Carleton University

March 2016

**PART I**

Network on Chips (NoC) consist of operating units. These units are connected to each other through network interfaces. NoC concept was proposed to overcome System on Chip (SoC) problems. The main goal is providing appropriate communication between operating units to enhance the system performance. The key concept for reaching this goal is the separation of the communication units from the processing units. One of the influencing factors in energy consumption, latency and other performance parameters of the NoC design is topology. The basic problem in the NoC architecture is the ability of the designed chip to provide a suitable trade-off between different important parameters to achieve appropriate quality of service. Mesh architecture is the most famous architecture in the NoC. This architecture has more flexibility as far as scalability and implementation are concerned.

In this assignment, we want to use Discrete Event System Specification (DEVS) formalism to model a 2x2 Mesh architecture (Figure 1), and to test a simple routing algorithm in this architecture. This model consists of four levels. In the first level, we have a coupled model that is called Chip. This coupled model consists of four similar coupled models. These coupled models are in the second level of the DEVS model hierarchy, and they are called Node. Each Node composes of an atomic model (Interface) and a coupled model (Unit). Interface and Unit together form the third level of the DEVS model hierarchy. In the fourth level, we have two atomic models: Unit Buffer and Unit Processor. Overall, in this DEVS model, we have nine coupled models and twelve atomic models.



Figure 1: DEVS model of 2x2 Mesh architecture

**PART II**

According to Figure 1 the atomic models for different Nodes (N1, N2, N3, and N4) are Buffer (B1, B2, B3 and B4), Processor (P1, P2, P3, and P4), and Interface (I1, I2, I3, and I4). All the Buffers have same functionality. While P1, P2, P3 and P4 are same too. All the Interfaces have the same functions also. The coupled models include Units, Node and Top (Chip) level an area.

The detail of the connections between all the components and their hierarchically structure has been shown in Figure 2.



Figure 2: Detailed view of DEVS model for Network on Chip.

In the following the formal specification of Atomic Models and Coupled Models has been discussed. Also there are detailed descriptions of responsibility of each function.

1. **Atomic Models**

**I (Interface) =<X, Y, S, δint, δext, λ, ta>**

**X= {(In1, In2, In3)}**

**Y= {(Out1, Out2, Out3)}**

**S= {Idle, Forward}**

**δext (s, e, x)** =

{

if ( msg.port() == In1 ) *// Input from unit processor*

{

fromNet = 0;

InterfaceI <<"Interface" << id <<" received '" << msg.value() << "' from port 'In1' at " << msg.time() << endl;

}

if( msg.port() == In2 ) //*Input from Network*

{

fromNet = 1;

InterfaceI <<"Interface" << id <<" received '" << msg.value() << "' from port 'In2' at " << msg.time() << endl;

}

if( msg.port() == In3 ) *//Input from Network*

{

fromNet = 1;

InterfaceI <<"Interface" << id <<" received '" << msg.value() << "' from port 'In3' at " << msg.time() << endl;

}

temp = msg.value();

state = Forward;

holdIn( Atomic::active, ProcessTime);

return \*this;

}

}

**δint(s) =**

{

switch (state){

case Idle:

passivate();

break;

case Forward:

state = Idle;

break;

}

return \*this;

}

**λ=**

{

if (state == Forward){

if (fromNet == 1) *//send the received msg from the network to the buffer of the Unit*

{

InterfaceO <<"Node " << id <<" Interface sent '" << temp << "' to the buffer at " << msg.time() << endl;

sendOutput( msg.time(), Out1, temp);

}

else if (fromNet == 0) *//send the received msg from the buffer to the Network*

{

if (rand ()%2 == 0) *//select a port randomly*

{

InterfaceO <<"Node " << id <<" Interface sent '" << temp << "' to the Network through port 'Out2' at " << msg.time() << endl;

sendOutput( msg.time(), Out2, temp);

}

else{

InterfaceO <<"Node " << id <<" Interface sent '" << temp << "' to the Network through port 'Out3' at " << msg.time() << endl;

sendOutput( msg.time(), Out3, temp);

}

}

}

return \*this ;

}



Figure 3: DEVS model of Interface

**B(Buffer) = <X, Y, S, δint, δext, λ, ta>**

**X= {(In, Ask)}**

**Y= {(Out)}**

**S= {Idle, Push, Pop}**

**δext (s, e, x)** =

{

if( msg.port ( ) == In ) *//Input from In*

{

BufferI <<"Unit " << id <<" Buffer received '" << msg.value ( ) << "' from port 'In' at " << msg. time ( ) << endl;

elements. push\_back ( msg.value ( ) ) ; *//Store Input value in Queue*

state = Push;

Qlen++;

holdIn ( Atomic::active, ProcessTime);

}

else if (msg.port( ) == Ask){

BufferI <<"Unit " << id <<" Buffer received '" << msg.value() << "' from port 'Ask' at " << msg.time( ) << endl;

if (elements.size( ) > 0) *//Pop from Queue (Check that the Queue is not empty)*

{

Request = 1;

state = Pop;

holdIn( Atomic::active, ProcessTime);

}

else if (elements.size() == 0)

{ *//Here this two condition are True :(msg.port() == Ask) & (elements.size() == 0)*

Request = 1; *//Although Buffer is empty but remember that server is waiting for input.*

passivate(); *//If the Buffer is empty then passivate.*

}

}

return \*this;

}

**δint(s) =**

{

switch (state){

case Idle:

if (Request == 1) state = Pop;

else passivate();

break;

case Push:

state = Idle;

break;

case Pop:

elements.pop\_front();

Qlen--;

Request = 0;

state = Idle;

break;

}

return \*this;

}

**λ =**

{

if ((state == Pop) && (Request == 1)){

BufferO <<"Unit " << id <<" Buffer sent '" << elements.front() << "' to port 'Out' at " << msg.time() << endl;

sendOutput( msg.time(), Out, elements.front()); *//Send out data from Buffer*

}

return \*this ;

}



Figure 4: DEVS model of Buffer

**P(myProcessor) =<X, Y, S, δint, δext, λ, ta>**

**X= {(In)}**

**Y= {(Out, Ask)}**

**S= {Idle, Task, Generate}**

**δext (s, e, x)** =

{

ProcessorI <<"Unit " << id <<" Processor received '" << msg.value( ) << "' from port 'In' at " << msg.time ( ) << endl;

if( msg.port ( ) == In ){ *//Input from In*

state = Task;

holdIn ( Atomic::active, ProcessTime);

}

return \*this;

}

**δint(s) =**

{

switch (state){

case Idle:

passivate();

break;

case Task:

if (is Generator == 0 || msgCounter > 10){

state = Idle;

}

else {

state = Generate;

}

askForTask = 1;

break;

case Generate:

msgToSend = (rand() % 4) + 1; *//create a number 1 to 4*

msgCounter = msgCounter + 1;

if (msgCounter > 10) {

state = Idle;

}

else {

holdIn( Atomic::active, ProcessTime);

}

break;

}

return \*this;

}

**λ =**

{

if (state == Generate && msgCounter > 0){

ProcessorO <<"Unit " << id <<" Processor sent '" << msgToSend << "' to port 'Out' at " << msg.time() << endl;

sendOutput( msg.time(), Out, msgToSend); *//Send out data from myProcessor to Interface*

}

if (askForTask == 1){

askForTask = 0;

ProcessorO <<"Unit " << id <<" Processor sent 1 to port 'Ask' at " << msg.time() << endl;

sendOutput( msg.time(), Ask, 1); *//Send out data from myProcessor to Interface*

}

return \*this ;

}



Figure 5: DEVS model for Processor

1. **Coupled Models**

**Node 1 = <X, Y, D, { Md** ⋮**d ∈ D}, EIC, EOC, IC, select>**

**X = {(In2, N), (In3, N)}**

**Y = {(Out2, N), (Out3, N)}**

**D= {I1, U1}**

**Md = {MI1, MU1}**

**EIC = { (self. In2, I1.In2); (self.In3, I1.In3) }**

**EOC = { (I1.Out2, self,Out2) ; (I1.Out3, self.Out3) }**

**IC = { (U1.Out1, I1.In1) ; (I1.Out1, U1.In1) }**

**Node 2 = <X, Y, D, { Md** ⋮**d ∈ D}, EIC, EOC, IC, select>**

**X = {(In2, N), (In3, N)}**

**Y = {(Out2, N), (Out3, N)}**

**D = {I2, U2}**

**Md = {MI2, MU2}**

**EIC = { (self. In2, I2.In2); (self.In3, I2.In3) }**

**EOC = { (I2.Out2, self,Out2) ; (I2.Out3, self.Out3) }**

**IC = { (U2.Out1, I2.In1) ; (I2.Out1, U2.In1) }**

**Node 3 = <X, Y, D, { Md** ⋮**d ∈ D}, EIC, EOC, IC, select>**

**X = {(In2, N), (In3, N)}**

**Y = {(Out2, N), (Out3, N)}**

**D = {I3, U3}**

**Md = {MI3, MU3}**

**EIC = { (self. In2, I3.In2); (self.In3, I3.In3) }**

**EOC = { (I3.Out2, self,Out2) ; (I3.Out3, self.Out3) }**

**IC = { (U3.Out1, I3.In1) ; (I3.Out1, U3.In1) }**

**Node 4 = <X, Y, D, { Md** ⋮**d ∈ D}, EIC, EOC, IC, select>**

**X = {(In2, N), (In3, N)}**

**Y = {(Out2, N), (Out3, N)}**

**D = {I4, U4}**

**Md = {MI4, MU4}**

**EIC = { (self.In2, I4.In2); (self.In3, I4.In3) }**

**EOC = { (I4.Out2, self,Out2) ; (I4.Out3, self.Out3) }**

**IC = { (U4.Out1, I4.In1) ; (I4.Out1, U4.In1) }**

**Chip = <X, Y, D, { Md** ⋮**d ∈ D}, EIC, EOC, IC, select>**

**X = φ**

**Y = φ**

**D = { Node1, Node2, Node3, Node4}**

**Md = {MNode1, MNode2, MNode3, MNode4}**

**EIC = φ**

**EOC = φ**

**IC = { (Node1.Out2, Node3.In2) ; (Node1.Out3, Node2.In3) ; (Node2.Out2, Node1.In2) ; (Node2.Out3, Node4.In2) ; (Node3.Out2, Node4.In3) ; (Node3.Out3, Node1.In3) ; (Node4.Out2, Node3.In3) ; (Node4.Out3, Node2.In2) }**



Figure 6: Coupled model for unit

**Part III**

There are test bench for each atomic and coupled models which are attached in (NOCassignment1.zip) to this report. Each of the components has a different event file (ComponentName.ev:Interface.ev,Buffer.ev, myProcessor.ev, Node.ev,Unit.ev) which tries to check the correctness of its components. For each component this event file and its related outputs are in folder with same name as that of the Component name.

For all Ps we need to test their reaction to received data packet (they should create ask for it ) and they response to received ask packet. Also they should generate packet at each certain time. The results show that this component works properly.

The event file and output file for myProcessor are like this(Same things happened for other Ps)

myProcessor.ev

00:00:00:10 In 1

00:00:00:10 In 1

00:00:00:16 In 1

00:00:00:20 In 1

00:00:00:25 In 1

00:00:00:30 In 1

00:00:00:35 In 1

00:00:00:40 In 1

00:00:00:45 In 1

00:00:00:50 In 1

00:00:00:55 In 1

00:00:00:60 In 1

00:00:00:65 In 1

myProcessor.out

00:00:00:000 ask 1

00:00:00:002 out 4

00:00:00:004 out 4

00:00:00:006 out 1

00:00:00:008 out 1

00:00:00:012 out 4

00:00:00:012 ask 1

00:00:00:014 out 3

00:00:00:018 out 4

00:00:00:018 ask 1

00:00:00:022 out 4

00:00:00:022 ask 1

00:00:00:024 out 3

00:00:00:027 out 1

00:00:00:027 ask 1

00:00:00:029 out 4

00:00:00:032 ask 1

00:00:00:037 ask 1

00:00:00:042 ask 1

00:00:00:047 ask 1

00:00:00:052 ask 1

00:00:00:057 ask 1

00:00:00:062 ask 1

00:00:00:067 ask 1

In case of Is, they should be able to forward then receive the packet correctly and according to their test results they work properly as well. They receive packet and in 10ms later they forward in for packet destination.

Interface.ev

00:00:00:10 In1 1

00:00:00:16 In3 6

00:00:00:20 In2 5

Interface.out

00:00:00:011 out2 1

00:00:00:017 out1 6

00:00:00:021 out1 5

For buffer also we have the same way of testing as that of other files.

Buffer.ev

00:00:00:10 In 1

00:00:00:10 In 1

00:00:00:16 In 1

00:00:00:20 In 1

00:00:00:25 In 1

00:00:00:27 ask 1

00:00:00:30 In 1

00:00:00:35 In 1

00:00:00:40 In 1

00:00:00:45 In 1

00:00:00:50 In 1

00:00:00:55 In 1

00:00:00:60 In 1

00:00:00:65 In 1

BufferOut.out

00:00:00:027 out 1