Introduction To Network Simulation With OMNET++ A case of PhoenixSim

Tutorial

Division of Computer Engineering
School of Computer Science and Engineering
University of Aizu

Contact: [d8151102, benab] @ u-aizu.ac.jp

Edition: June 1, 2015

Outline

- ❑ What Is OMNeT++?
- ❑ Programming model in OMNET++
- ❑ PhoenixSim:
  - PhoenixSim folder hierarchy
  - Install OMNET++
  - Build PhoenixSim
  - Configuration file
  - Run PhoenixSim
- ❑ PhoenixSim's Electronic Router
- ❑ RouterInport simple module C++ implementation

What Is OMNeT++?

- OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators.
- "Network" is meant in a broader sense that includes wired and wireless communication networks, on-chip networks, queueing networks. Etc.

- OMNET ++ framework includes:
  - ✓ Simulation kernel library
  - ✓ NED topology description language
  - ✓ OMNET++ IDE based on eclipse platform
  - ✓ GUI for simulation execution (Tkenv)
  - ✓ Command-line user interface for simulation execution (Cmdenv)

- OMNeT++ runs on Windows, Linux, Mac OS X, and other Unix-like systems.

- The OMNeT++ IDE requires Windows, Linux, or Mac OS X.

Programming model

- An OMNeT++ model consists of modules that communicate with message passing.
- The active modules are termed simple modules; they are written in C++, using the simulation class library.
- The whole model, called network in OMNeT++, is itself a compound module. Messages can be sent either via connections that span modules or directly to other modules.

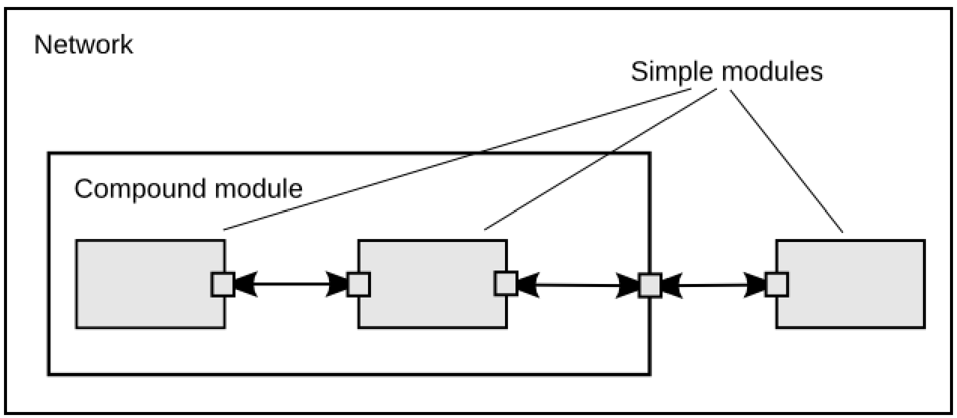
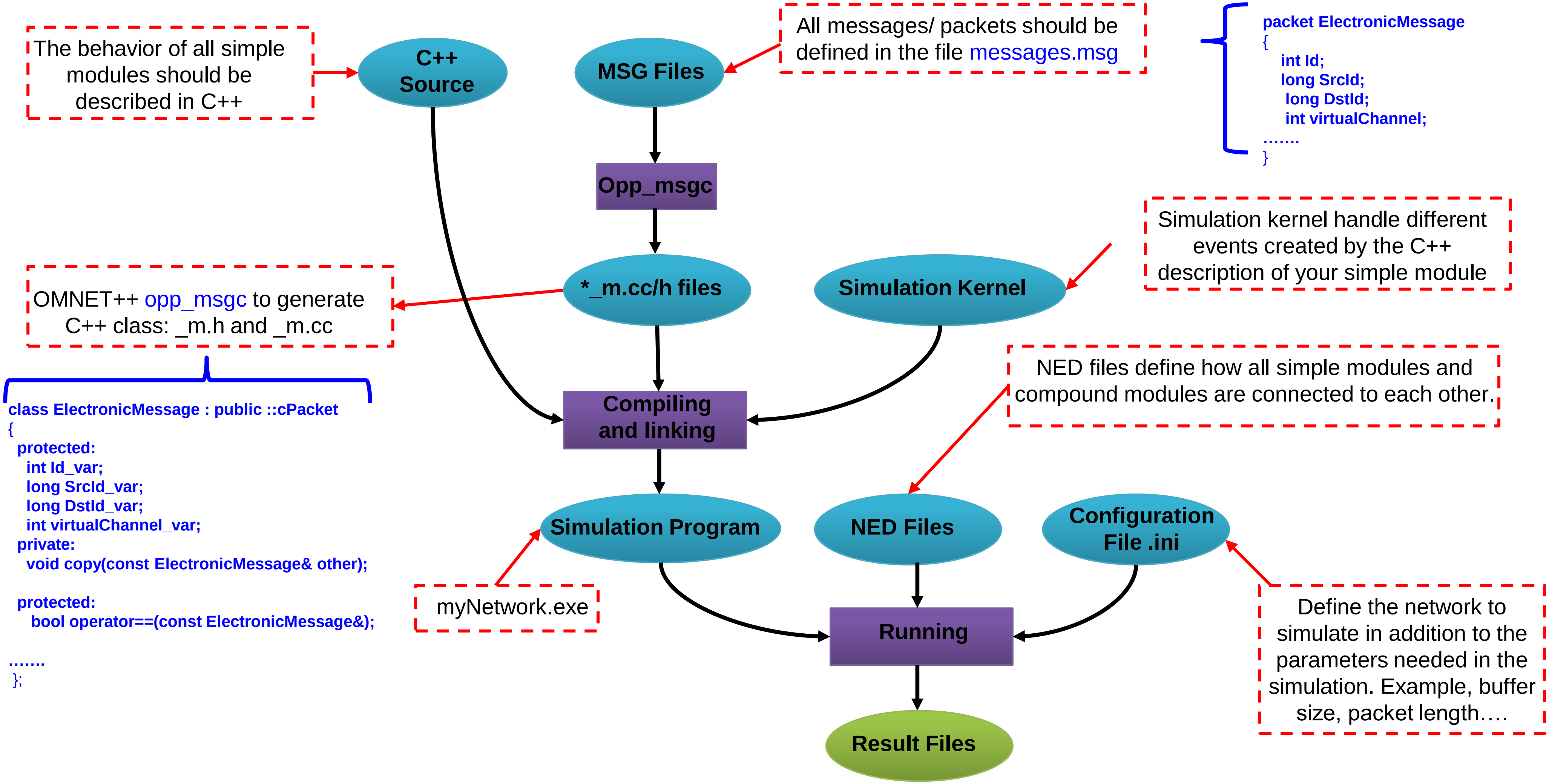


Figure: Simple and compound modules

Programming model



PhoenixSim 1.0

- PhoenixSim (Photonic and Electronic Network Integration and eXecution Simulator) was originally designed to allow the investigation of silicon nanophotonic NoCs taking into account component's physical layer characteristics.

- Because photonics often requires electronic components around it for control and processing, models of some typical electronic network components are incorporated.

- We ended up with a simulation environment that is suited to investigate both electronic and photonic NoCs.

PhoenixSim folder hierarchy

- The components you will find in the PhoenixSim code are organized into various folders depending on their function. The following enumerates and describes the contents of these folders:
- chipComponents - contains definitions of some useful components used in NoCs
- electronicComponents - contains the components used in electronic routers, including the ORION power model . Also contains all Arbiters, which are used to define new routing functions.
- ioComponents - contains components used in the IO PLANE, which is currently limited to DRAM
- parameters - contains default parameter values for all components
- photonic - contains all the models of the photonic devices including with PhoenixSim
- processingPlane - contains components necessary for modeling traffic generation
- simCore - contains core functions, such as message definitions and statistics
- topologies - contains the network topologies that come with PhoenixSim

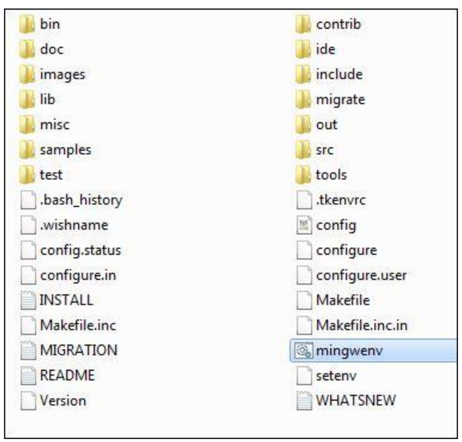
Install OMNET++

- Download the last version of OMNeT++ for windows from this link:

  OMNeT++ 4.6 win32 (source + IDE + MinGW, zip)

- Extract the zip folder into a directory that doesn't have space in its name, Otherwise you'll have problems with the makefiles.

- Go to the folder where you unzipped the sources and Start mingwenv.cmd as administrator. It will bring up a console with the MSYS bash shell, where the path is already set to include the omnetpp-4.6/bin directory.



Install OMNET++

- ❑ First check your gcc and g++ versions

```
/c/omnetpp-4.6-src-windows/omnetpp-4.6$ gcc --version
gcc.exe (Rev2, Built by MSYS2 project) 4.9.2
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

/c/omnetpp-4.6-src-windows/omnetpp-4.6$ g++ --version
g++.exe (Rev2, Built by MSYS2 project) 4.9.2
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- ❑ Then enter ./configure OMNET++ will check your system configuration

```
/c/omnetpp-4.6-src-windows/omnetpp-4.6$ ./configure
checking build system type... i686-pc-mingw32
checking host system type... i686-pc-mingw32
configure: ---------------------------------------------
configure: reading configure.user for your custom settings
configure: ---------------------------------------------
checking for icc... no
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.exe
checking for suffix of executables... .exe
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
```

```
Scroll up to see the warning messages (use shift+PgUp), and search config.log
for more details. While you can use OMNeT++ in the current configuration,
be aware that some functionality may be unavailable or incomplete.

Your PATH contains C:/omnetpp-4.6-src-windows/omnetpp-4.6/bin. Good!

/c/omnetpp-4.6-src-windows/omnetpp-4.6$
```

Install OMNET++

- Then enter make . The build process will create both debug and release binaries.

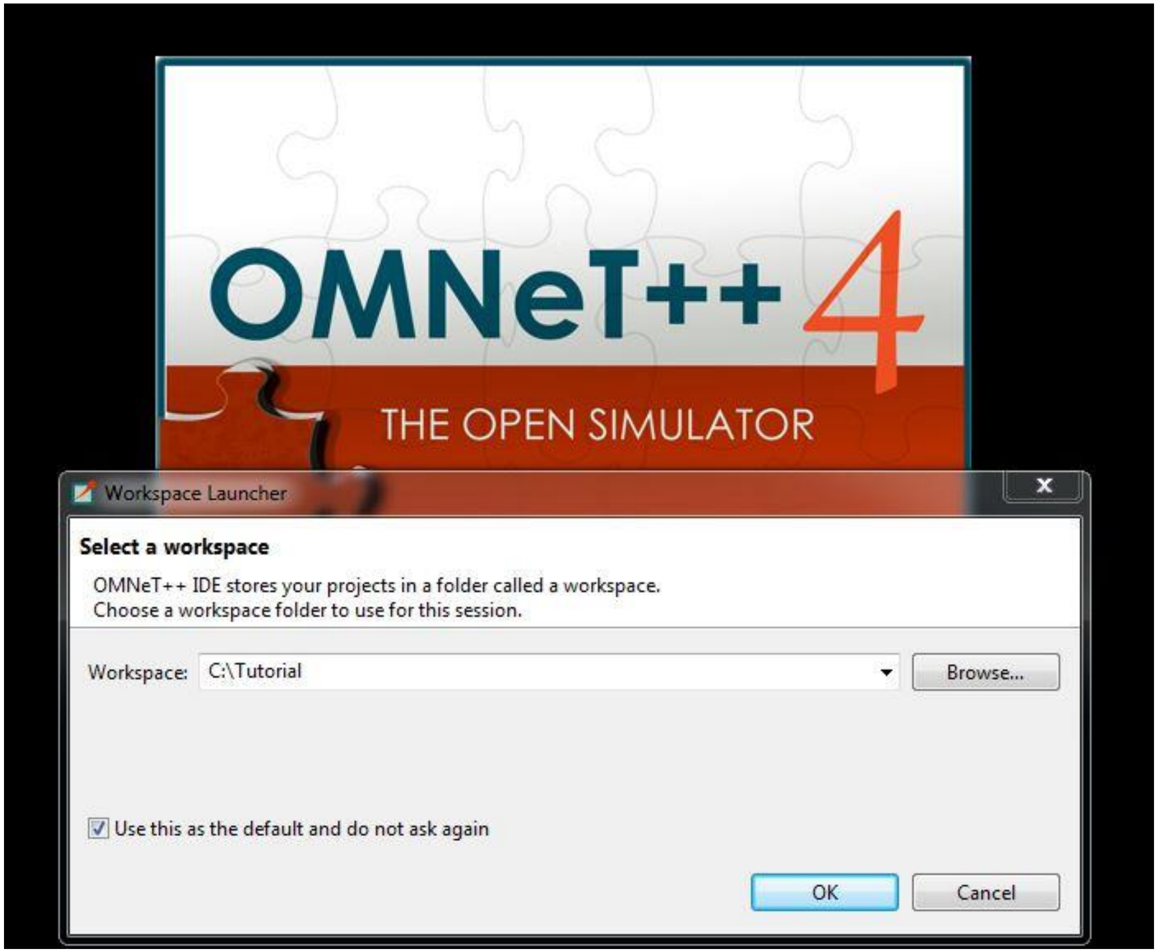
```
/c/omnetpp-4.6-src-windows/omnetpp-4.6$ make
make MODE=release
make[1]: Entering directory '/c/omnetpp-4.6-src-windows/omnetpp-4.6'
***** Configuration: MODE=release, TOOLCHAIN_NAME=gcc, LIB_SUFFIX=.dll ****
===== Checking environment =====
===== Compiling utils ====
make[2]: Entering directory '/c/omnetpp-4.6-src-windows/omnetpp-4.6/src/utils'
Copying scripts to bin directory...
make[2]: Leaving directory '/c/omnetpp-4.6-src-windows/omnetpp-4.6/src/utils'
===== Compiling common ====
make[2]: Entering directory '/c/omnetpp-4.6-src-windows/omnetpp-4.6/src/common'
Creating DLL: C:/omnetpp-4.6-src-windows/omnetpp-4.6/lib/gcc/liboppcommon.dll
make[2]: Leaving directory '/c/omnetpp-4.6-src-windows/omnetpp-4.6/src/common'
===== Compiling layout ====
make[2]: Entering directory '/c/omnetpp-4.6-src-windows/omnetpp-4.6/src/layout'
Creating DLL: C:/omnetpp-4.6-src-windows/omnetpp-4.6/lib/gcc/liboppLayout.dll
make[2]: Leaving directory '/c/omnetpp-4.6-src-windows/omnetpp-4.6/src/layout'
===== Compiling eventlog ====
```

- Now You should be able to start the IDE by typing: omnetpp

```
make[2]: Entering directory '/c/omnetpp-4.6-src-windows/omnetpp-4.6/samples/sockets
Creating executable: out/gcc-debug//sockets.exe
make[2]: Leaving directory '/c/omnetpp-4.6-src-windows/omnetpp-4.6/samples/sockets'
make[1]: Leaving directory '/c/omnetpp-4.6-src-windows/omnetpp-4.6'

Now you can type "omnetpp" to start the IDE

/c/omnetpp-4.6-src-windows/omnetpp-4.6$
```

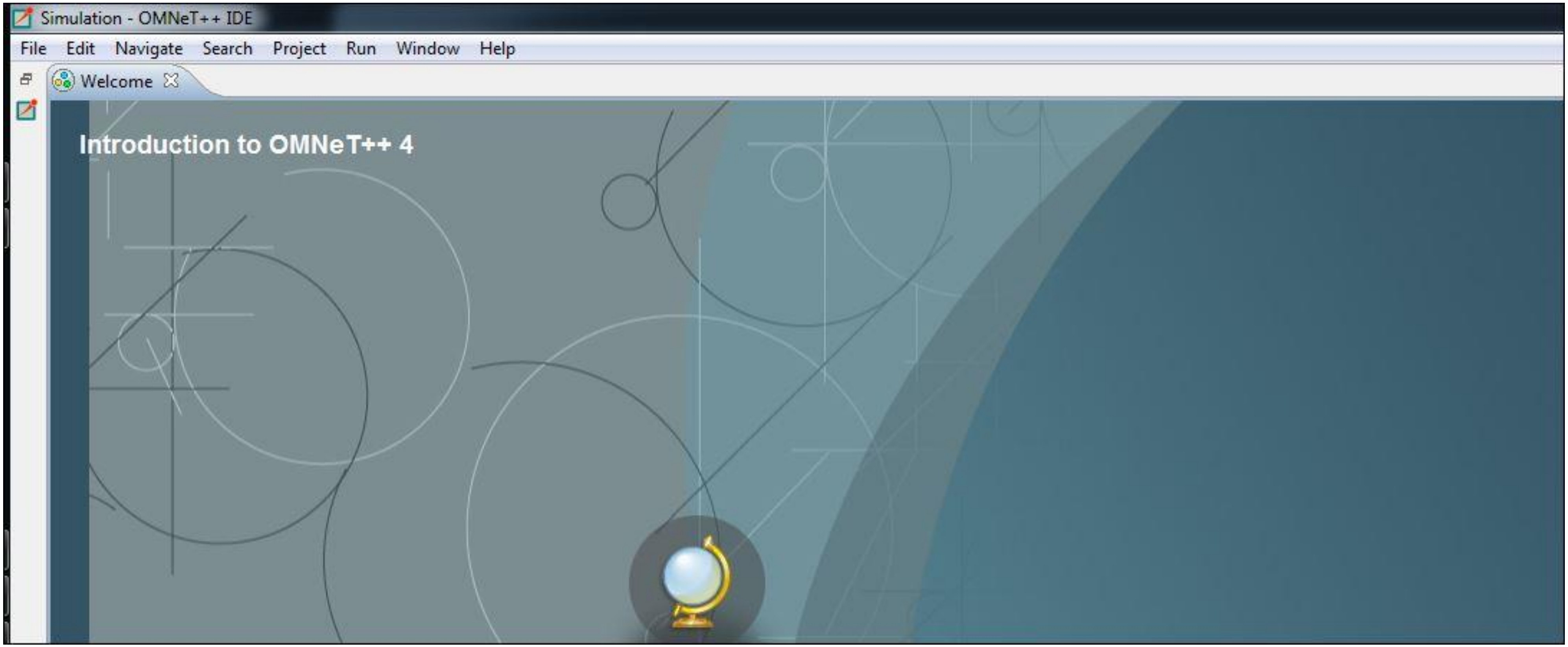
Install OMNET++

1- Choose your workspace directory



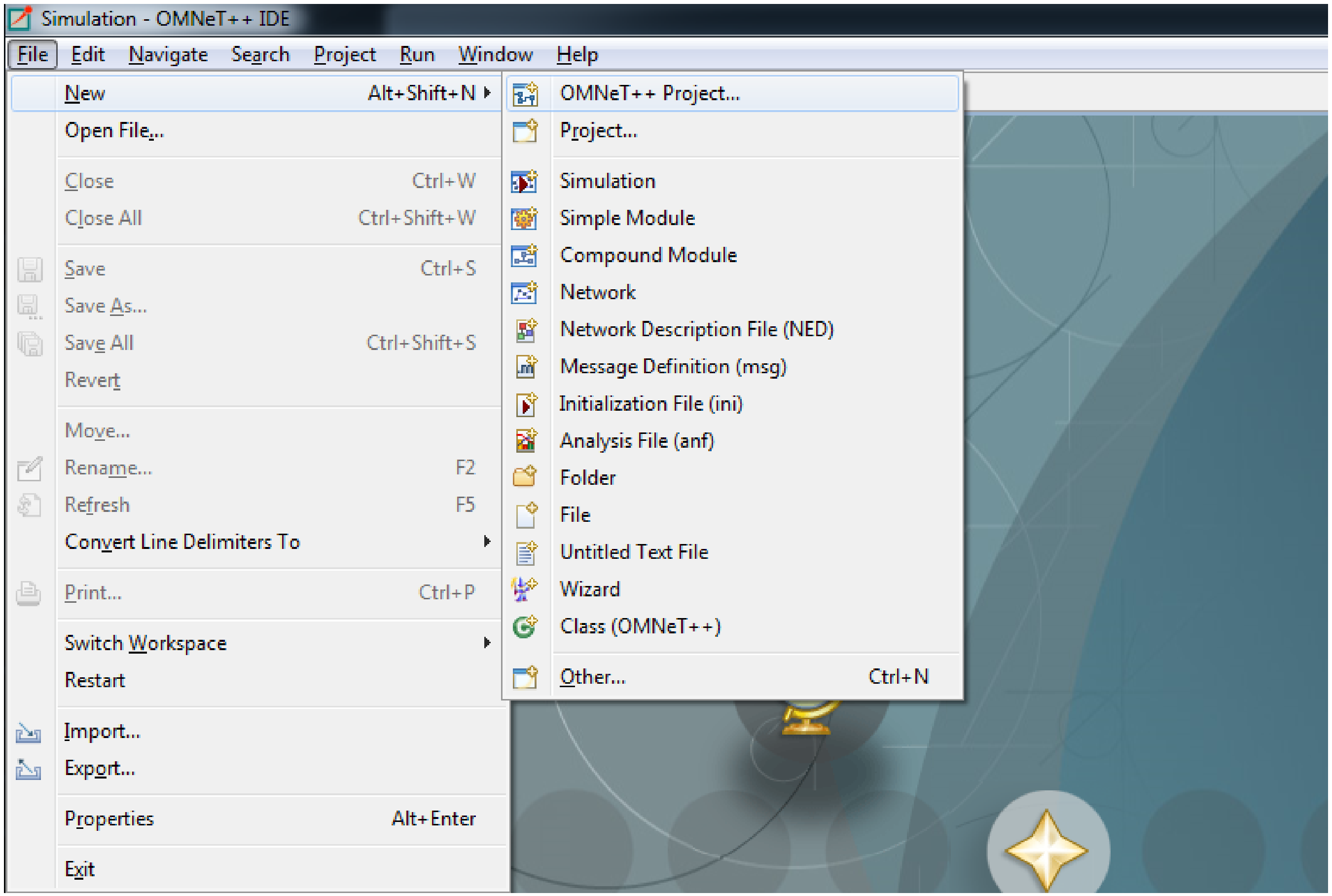
Install OMNET++

Welcome screen in OMNET++ 4.6



Build PhoenixSim

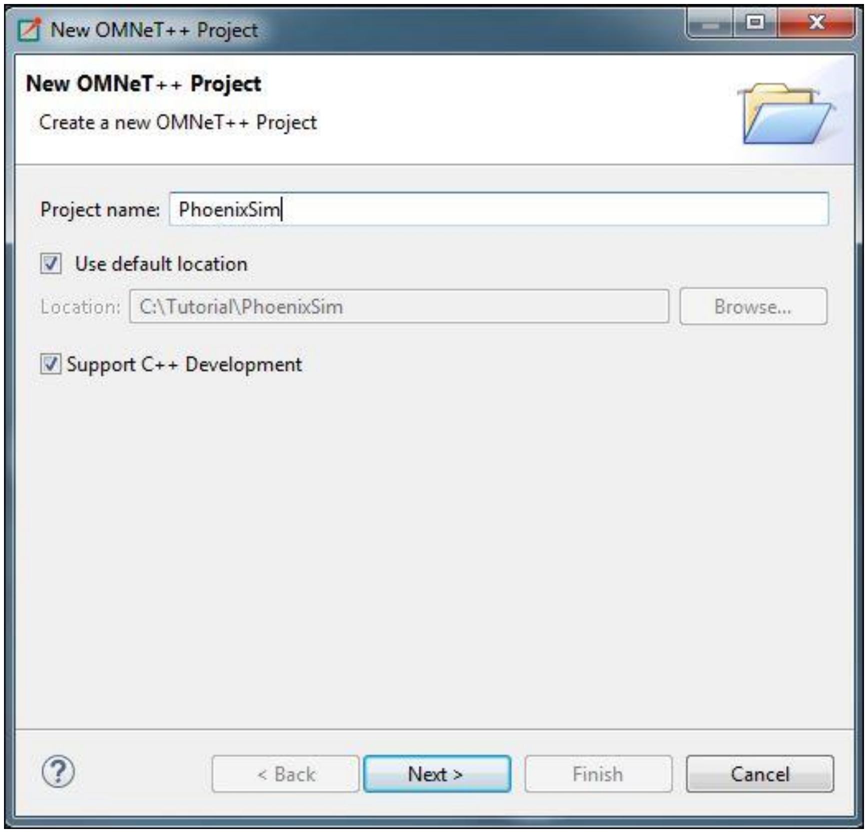
2- You need to create an empty project through File/New/OMNET++ Project

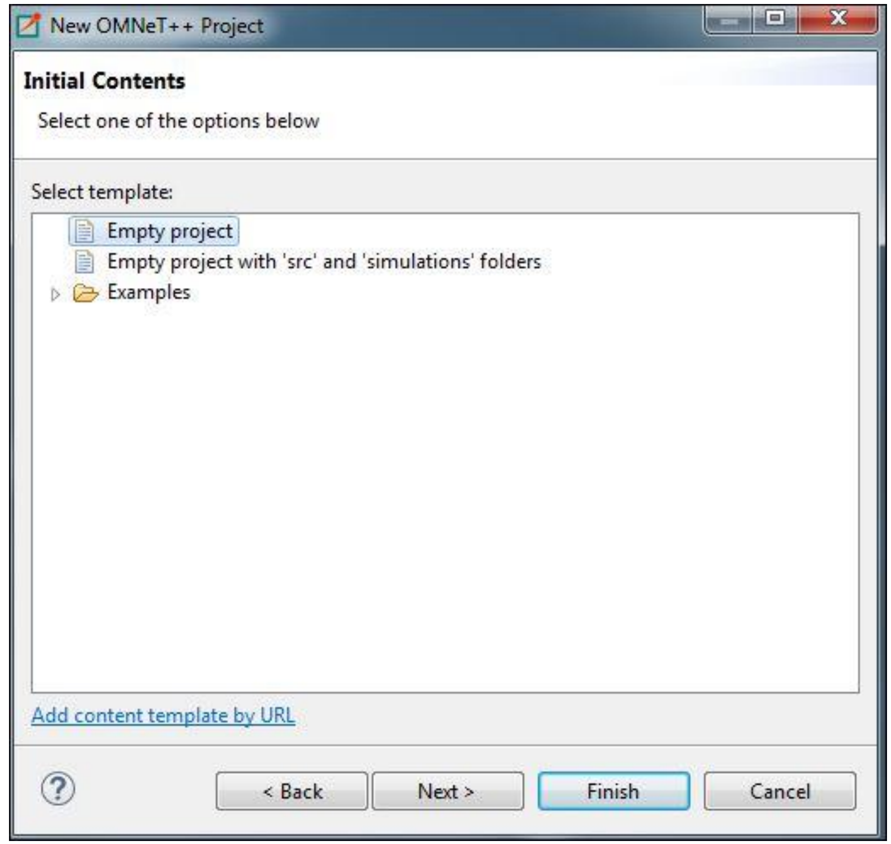


Build PhoenixSim

3- Name your project

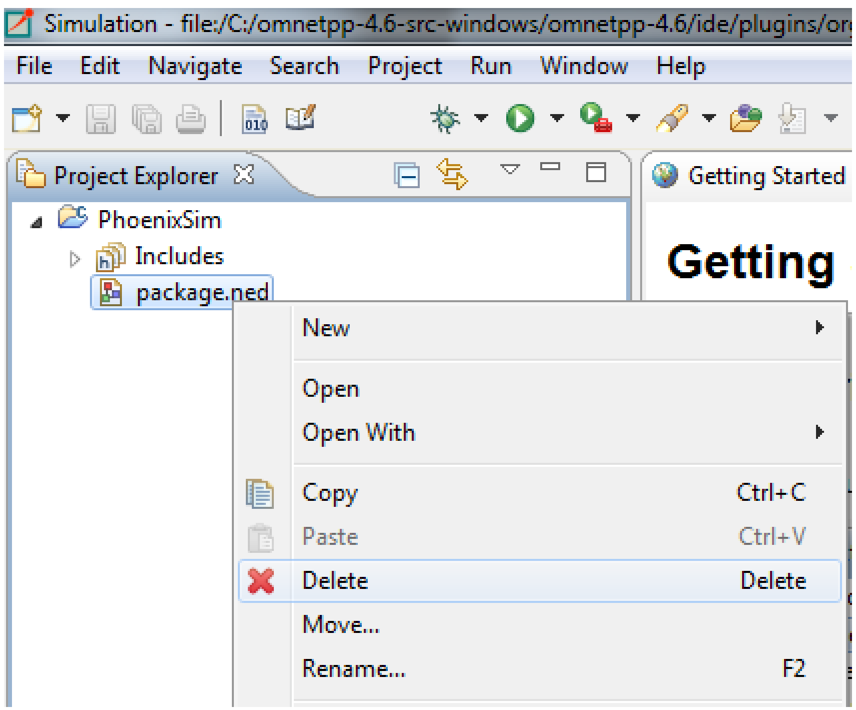
4- Chose empty project and click Finish



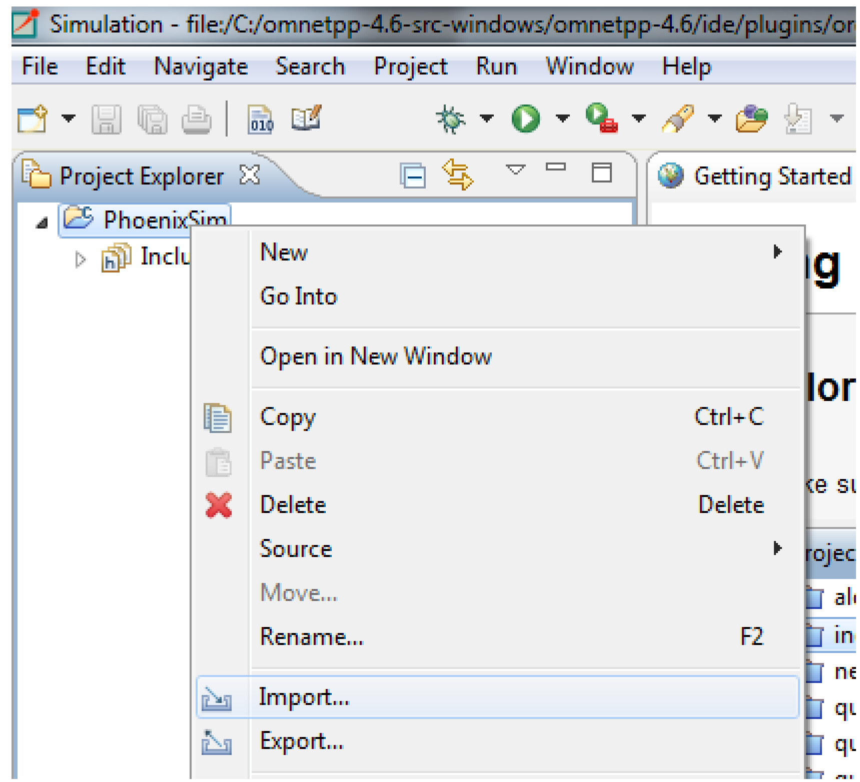


Build PhoenixSim

5- Do not forget to delete package.ned, otherwise you will get errors when you import PhoenixSim project

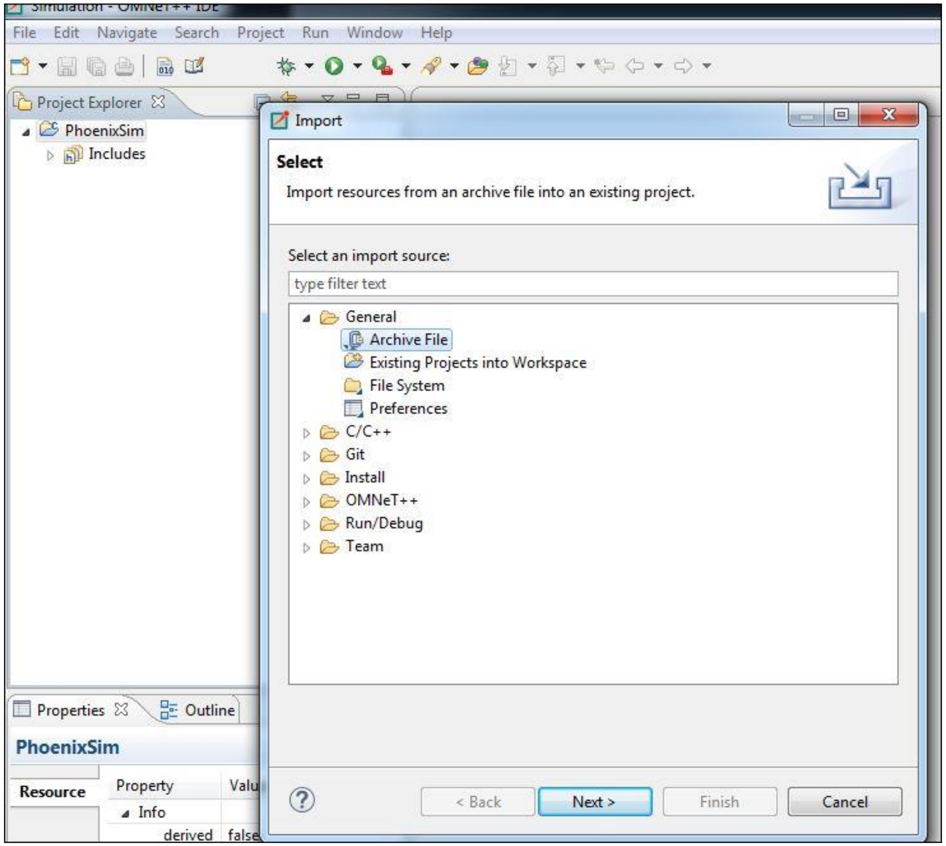


6- Right click on your project and choose Import

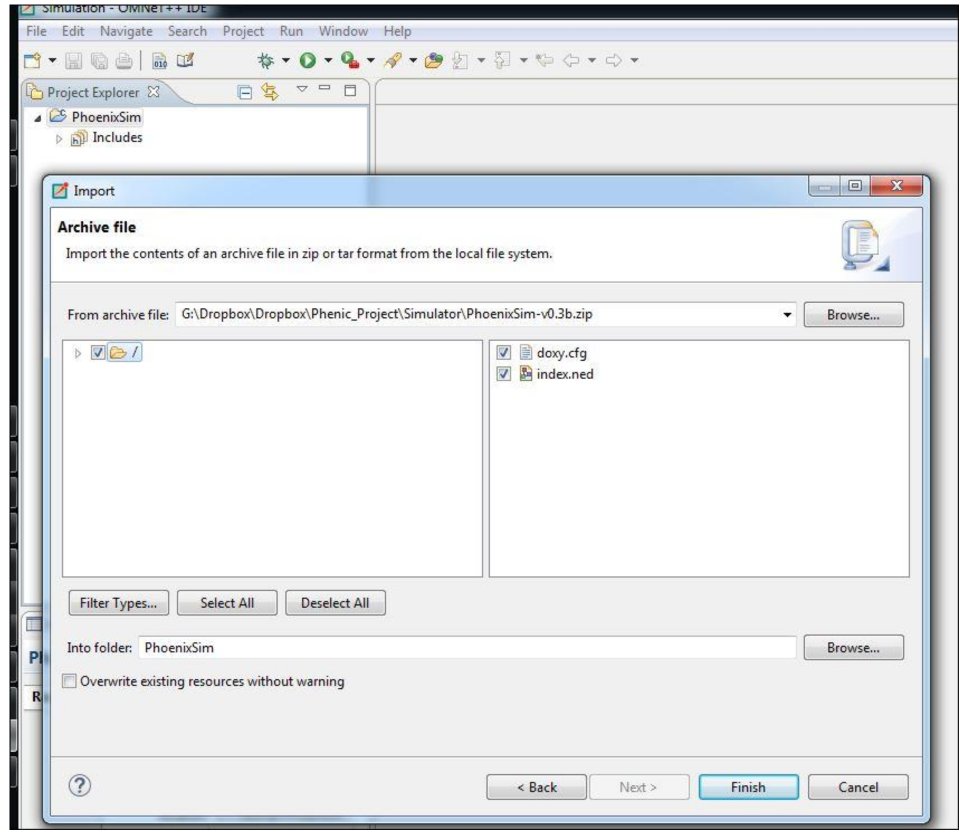


Build PhoenixSim

7- Choose General / Archive File

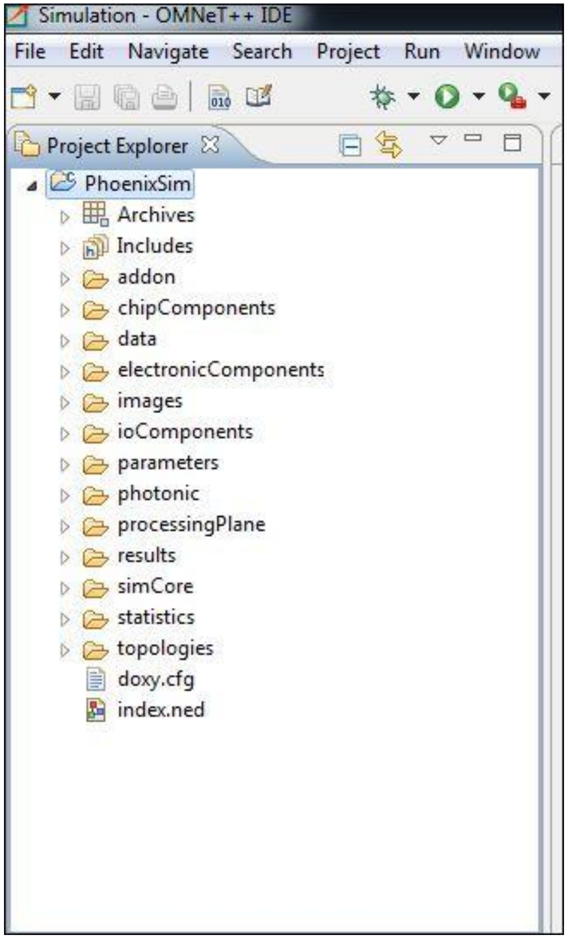


8- Go to where you saved PhoenixSim .zip file and select it (When you download PheonixSim DO NOT unzip it)

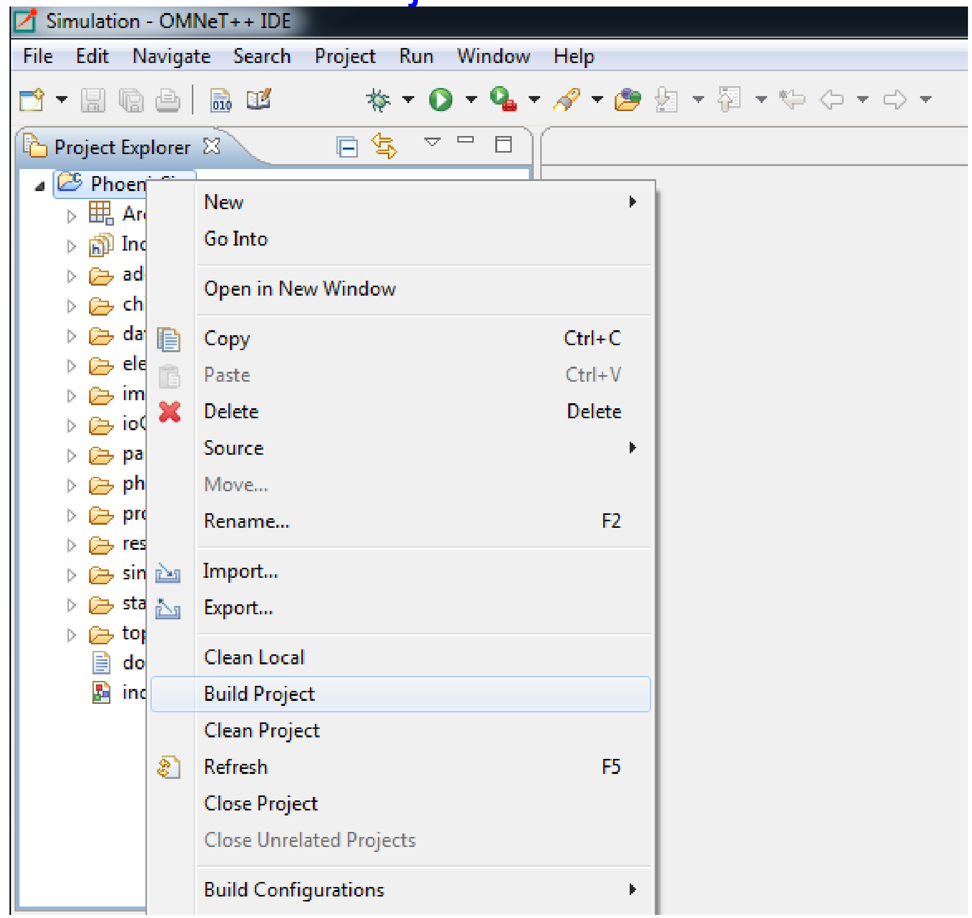


Build PhoenixSim

9- Now, your project is ready to be build



10- Right click on your project and click Build Project



Build PhoenixSim

11- If everything goes fine you will be able to see this message

Problems | Module Hierarchy | NED Parameters | NED Inheritance | Console

CDT Build Console [PhoenixSim]

```
In file included from topologies/photonicTDM/TDM/TDM_Switch_Controller.h:21:0,
                 from topologies/photonicTDM/TDM/TDM_Switch_Controller.cc:16:
simCore/sim_includes.h: At global scope:
simCore/sim_includes.h:22:12: warning: 'globalMsgId' defined but not used [-Wunused-variable]
 static int globalMsgId = 0;
            ^
simCore/messages_m.cc
Creating executable: out/gcc-debug//PhoenixSim.exe

20:26:53 Build Finished (took 2m:18s.765ms)
```

Configuration file

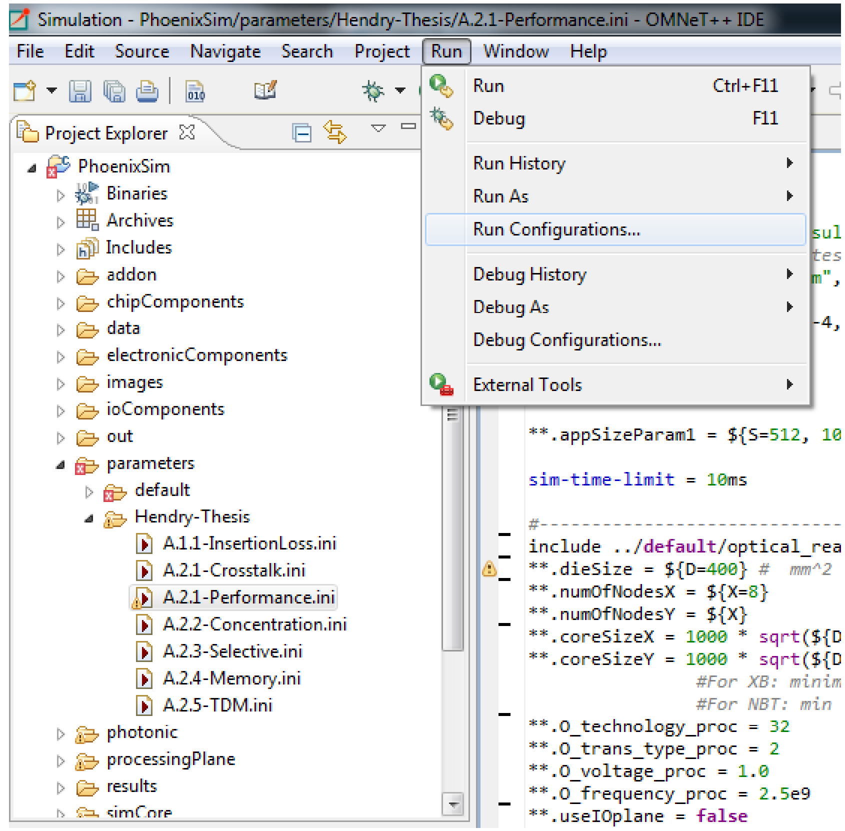
- PhoenixSim has multiple networks and topologies. You may find many configurations in <myProject>/parameters/Hendry-Thesis
- To run PhoenixSim you need a configuration file where you define all needed parameters for the simulation
- In this example the configuration file contains:
  1. Applications to run: “random”, “neighbor”..
  2. InjectionRate: 1E-3, 1E-4… (seconds)
  3. MessageSize: 512, 1024.. (bits)
  4. NetworkSize: X=8, Y=X
  5. ElectronicPara: Processor frequency= 2.5GHZ

*A.2.1-Performance.ini

```
[General]

**.logDirectory = "../../results/"
#-------- IL/functionality test -----------
**.application = ${P="random", "neighbor", "hotspot", "tornado", "bitreverse"}

**.appParam1 = ${A=1E-3, 1E-4, 1E-5, 1E-6, 1E-7}
**.appParam2 = 0
**.appParam3 = -1
**.appParam4 = 0

**.appSizeParam1 = ${S=512, 1024, 2048, 4096, 8192, 16384}

sim-time-limit = 10ms

#---------------------------------------
include ../default/optical_realistic_parameters.ini
**.dieSize = ${D=400} #  mm^2
**.numOfNodesX = ${X=8}
**.numOfNodesY = ${X}
**.coreSizeX = 1000 * sqrt(${D}) / ${X}   #; um     note: core size must be > 10
**.coreSizeY = 1000 * sqrt(${D}) / ${X}   #; um     used by Torus topology and E
               #For XB: minimum size is XxY=565x755 based on original 4x4 switc
               #For NBT: min size = 620x1034
**.O_technology_proc = 32
**.O_trans_type_proc = 2
**.O_voltage_proc = 1.0
**.O_frequency_proc = 2.5e9
**.useIOplane = false

**.processorConcentration = 1
#--------------------- Hybrid Circuit-Switched Photonic Networks ------------
[Config HybridPhotonicNetworks]

[Config PhotonicMesh]
extends = HybridPhotonicNetworks
network = topologies.photonicMesh.PhotonicMeshNetwork
**.networkName = "P-Mesh"
**.numOfWavelengthChannels = 64
**.networkProfile = "NET.;" + string(${X} * ${X}) + "."
**.meshTileVariant = 0 #NX style
**.switchVariant = "NonBlockingSwitch4x4New"
#----------------------------------------#
**.customInfo = ${P} + "_${S}_${A}"
include ../default/default_parameters.ini
```

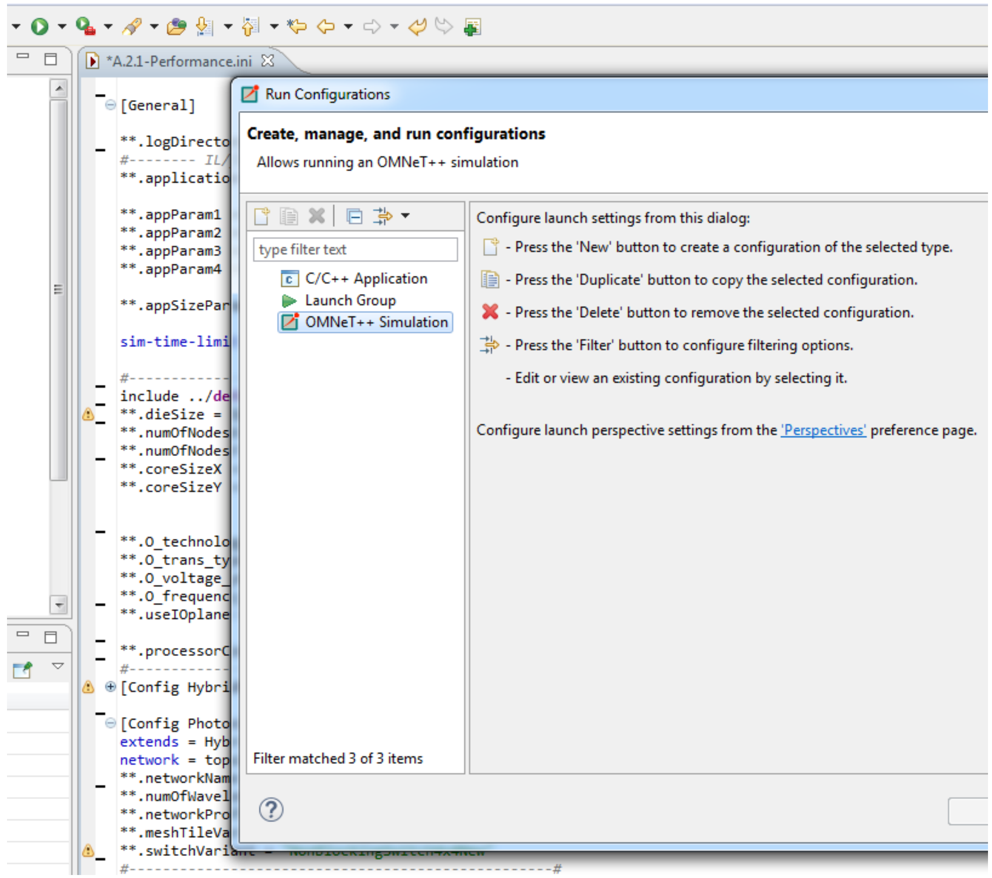
Run PhoenixSim

1- go to Run/Run Configurations



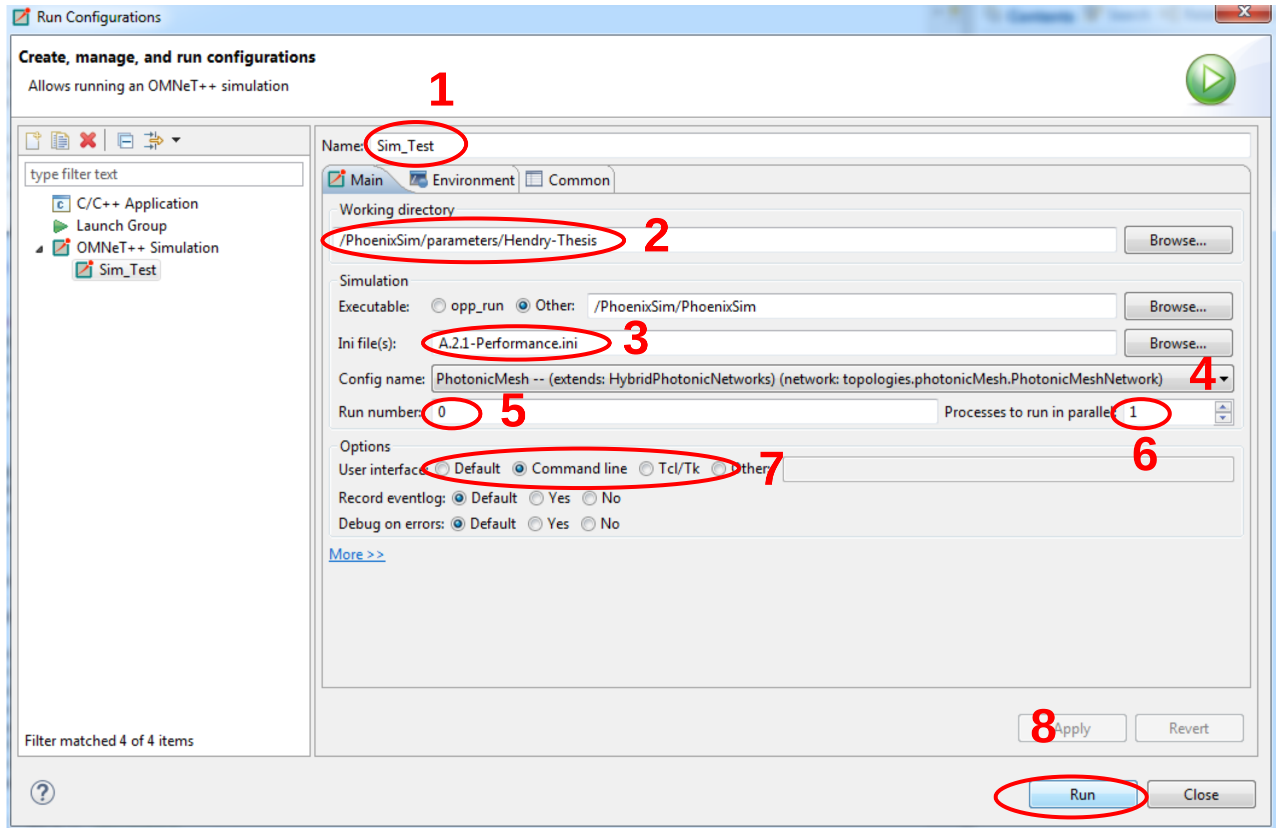
Run PhoenixSim

2- Create new configuration click on OMNET++ Simulation



Run PhoenixSim

1- Name your configuration.
2- Select the folder containing configuration files.
3- Select the configuration to run.
4- Select the network to simulate (you configuration file may have many networks).
5- Your configuration may have multiple iterations, you can run all of them by typing (*).
6- Select how many process run in parallel (depends on your processor).
7- choose default for GUI based simulation or Command line if you want to see the output on the console.
8- When you are ready click Run



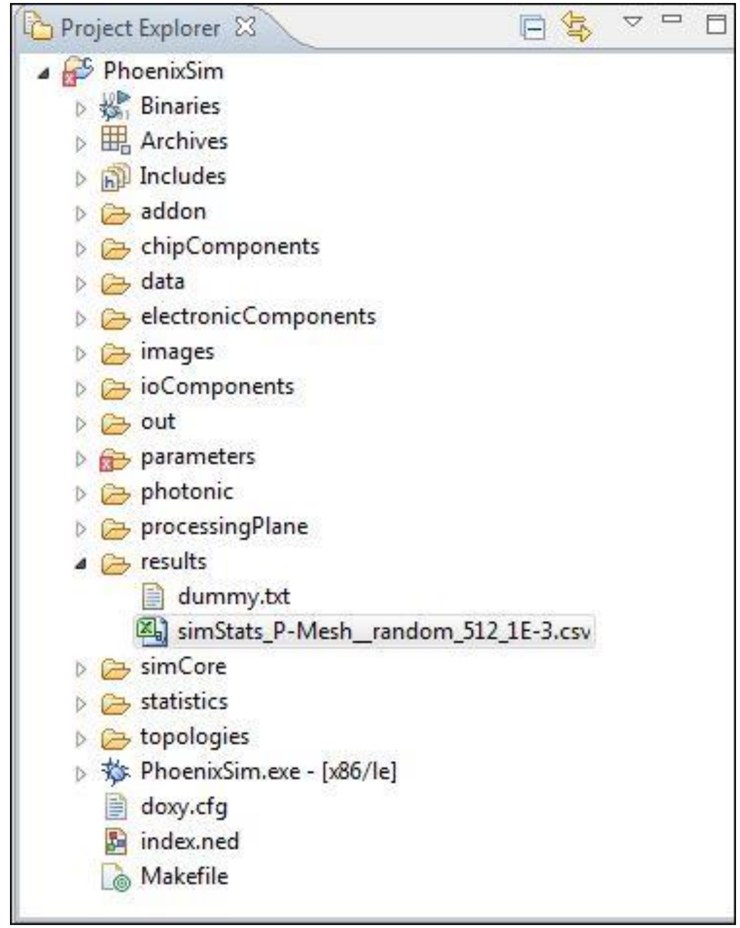
Run PhoenixSim

- When the simulation finish you will see on your console this output. In this example, only run number 0 is executed

```
Problems | Module Hierarchy | NED Parameters | NED Inheritance | Console
<terminated> Sim_Test [OMNeT++ Simulation] PhoenixSim.exe (5/28/15 8:33 PM - run #0)
Starting...

$ cd C:/Tutorial/PhoenixSim/parameters/Hendry-Thesis
$ ../../PhoenixSim.exe -r 0 -u Cmdenv -c PhotonicMesh -n ../.. --tkenv-image-path=../../images A.2.1-Performance.ini

OMNeT++ Discrete Event Simulation  (C) 1992-2014 Andras Varga, OpenSim Ltd.
Version: 4.6, build: 141202-f785492, edition: Academic Public License -- NOT FOR COMMERCIAL USE
See the license for distribution terms and warranty disclaimer
Setting up Cmdenv...
Loading NED files from ../..: 150

Preparing for running configuration PhotonicMesh, run #0...
Scenario: $P="random", $A=1E-3, $S=512, $D=400, $X=8, $repetition=0
Assigned runID=PhotonicMesh-0-20150528-20:33:45-12536
Setting up network `topologies.photonicMesh.PhotonicMeshNetwork'...
Initializing...
profile[0]: NET
profile[1]: DRAM
profile[2]: PROC

Running simulation...
** Event #1   T=0   Elapsed: 0.000s (0m 00s)  0% completed
     Speed:     ev/sec=0   simsec/sec=0   ev/simsec=0
     Messages:  created: 1728   present: 1728   in FES: 512
** Event #332630   T=0.010004523181   Elapsed: 1.449s (0m 01s)  100% completed
     Speed:     ev/sec=229558   simsec/sec=0.00690443   ev/simsec=3.32479e+007
     Messages:  created: 111382   present: 1283   in FES: 65

<!> Simulation time limit reached -- simulation stopped at event #332630, t=0.010004523181.


Calling finish() at end of Run #0...
undisposed object: (ProcessorData) PhotonicMeshNetwork.prPlane.tx[38].proc. -- check module destructor

End.
```

Run PhoenixSim

- ❑ The obtained result file is located in <myProject>/results
- ❑ The result file contains different statistics:
  - ✓ Electronic energy.
  - ✓ Photonic energy.
  - ✓ Performance including throughput and latency.
  - ✓ Physical Layer statistics including Crosstalk and Insertion Loss.
- ❑ For some configurations you will have hundreds of files. To speed up gathering statistics from these files, you need to write your own script to parse all files at once.



Run PhoenixSim

❑ Result file sample

Electronic Energy distribution

Photonic Energy distribution

Total component count in your network

A	B	C	D
Simulation Time:	0.010004523		
Electronic Energy			
	E_static (J)	E_dynamic (J)	
Electronic Arbiter	0.000154403	4.55E-09	
Electronic Clock Tree	0.0015029	0	
Electronic Crossbar	0.00540577	3.50E-09	
Electronic Inport	0.00300085	2.14E-08	
Electronic Wire	8.06E-06	1.20E-07	
Electronic Energy			
Sum:	0.0100721		
Photonic Energy			
	E_static (J)	E_dynamic (J)	
Detector	0.000819571	2.43E-07	
Detector static	0	0	
Modulator	0.000819571	1.35E-07	
PSE1x2	0.000102446	1.20E-07	
PSE1x2NX	5.12E-05	4.09E-08	
Photonic Energy			
Sum:	0.00179335		
All Energy			
Sum:	0.0118655		
Component Counts			
	Count		
Electronic Arbiter	64		
Electronic Crossbar	64		
Electronic Inport	320		
Electronic Wire	224		
PSE1x2	512		
PSE1x2NX	256		

Throughput

Latency

Latency breakdown

Physical layer stat (SNR)

Insertion Loss distribution

A	B	C	D	E	F	G
	TimeAvg					
Bandwidth App In (Gb/s)	0.0328044					
Performance: Application						
	Count	Min	Avg	Max	StdDev	
Latency (us)	641	0.038225	0.147966	0.368387	0.066776	
Msg Size (B)	641	64	64	64	0	
Performance: NIF						
	TimeAvg					
Bandwidth NIF Out (Gb/s)	0.0328044					
Blocking latency	641	0	2.87E-10	1.84E-07	7.26E-09	
Success setup latency	641	3.50E-08	1.45E-07	3.47E-07	6.64E-08	
Time in NIF Q	641	0	0	0	0	
Transmission latency	641	3.23E-09	3.34E-09	3.54E-09	6.82E-11	
Physical Layer Stats						
	Count	Min	Avg	Max	StdDev	
1 Laser Noise Power (mW)	641	1.63E-08	1.55E-07	4.07E-07	9.68E-08	
2 Message Noise Power (mW)	641	0	0	0	0	
3 Cross Wavelength Noise Power (mW)	641	0	0	0	0	
4 Total Optical Noise Power (mW)	641	1.63E-08	1.55E-07	4.07E-07	9.68E-08	
5 Johnson Noise Power	641	8.37E-13	8.37E-13	8.37E-13	3.53E-27	
6 Shot Noise Power	641	8.01E-10	8.01E-10	8.01E-10	1.01E-23	
7 Total Noise Power (dBm)	641	-60.8713	-60.212	-59.1761	0.425023	
8 Signal Power (dBm)	641	-25.0931	-16.2373	-11.1151	3.0085	
Electrical SNR	641	35.7782	43.9748	48.061	2.60847	
Optical SNR	641	52.7893	52.7893	52.7893	7.25E-13	
Insertion Loss						
	Count	Min	Avg	Max	StdDev	
Bending	641	0.035	0.150507	0.34	0.063689	
Crossing	641	0	1.60811	3.9	0.798554	
Drop Into Ring	641	1.5	3.91498	8	1.29448	
Pass By Ring	641	0.325	0.400031	0.515	0.0389	
Propagation	641	0.578323	2.10184	4.81431	0.911618	
Total	641	3.05332	8.17547	17.0313	3.0085	

PhoenixSim’s Electronic Router

- As we mentioned at the beginning of this tutorial, simple modules are the active components in the model. Simple modules are programmed in C++, using the OMNeT++ class library.

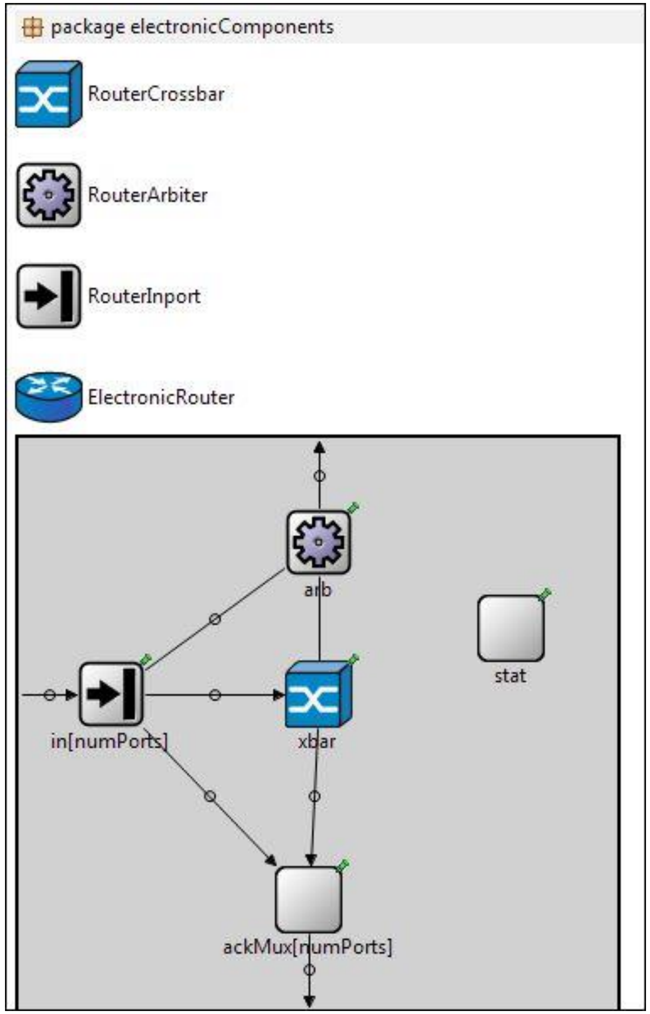
- In addition to the C++ implementation, simple module like all other modules in your network, needs to be defined with topology description language (NED).

- In the .ned file, the connections to other modules are defined. In addition, you can add all parameters that you need.

- In the following slides, and example of the electronic router and the input port in PhoenixSim is provided.

PhoenixSim’s Electronic Router

- The figure shows the .ned definition of the electronic router in PhoenixSim. You can find this file in <myProject>/electronicComponents/ElectronicRouter.ned
- ElectronicRouter.ned is a compound module having multiple simple modules connected with each other.
- The typical simple modules in the electronic router are: Input ports (in), Arbiter(arb), Crossbar (xbar).
- ackMux module is used to multiplex the ack from input and the data from the crossbar to be sent to the output port.
- Stat module is used for simulation purpose to gather statistics from the different modules.



PhoenixSim's Electronic Router

```
simple RouterInport
{
    parameters:
        int routerVirtualChannels;
        double clockRate;
        int electronicChannelWidth;
        int routerBufferSize;
        int myId;
        @display("i=block/arrival");
    gates:
        input in;
        inout req;
        output ackOut;
        output toXbar;
}
```

RouterInport
Simple module

Simple module needs C++ implementation:
RouterInport.h
RouterInport.cc

```
simple RouterCrossbar
{   parameters:
        int numPorts;
        double clockRate;
        int electronicChannelWidth;
        bool autounblock = default(true);
        bool isDataPlane = default(false);
        bool isProcPlane = default(false);
        @display("i=abstract/switch");
    gates:
        input in[numPorts];
        output out[numPorts];
        inout cntrl;
}

simple RouterArbiter
{
    parameters:
        string id;
        string level;
        string networkProfile;
        int processorConcentration;
        string addressTranslator;
        int routerMaxGrants;
        int numOfNodesX;
        int numOfNodesY;
        int numX;
        int numY;
        int type;
        int numPorts;
        int numPSE;
        int switchVariant;
        int routerVirtualChannels;
        int electronicChannelWidth;
        double clockRate;
        int routerBufferSize;
        @display("i=block/cogwheel");
    gates:
        inout req[numPorts];
        inout XbarCntrl;
        output pseCntrl[numPSE];
        input ackIn[numPorts];
}
```

Crossbar and Arbiter
Simple modules

```
module ElectronicRouter
{
    parameters:
        string id;
        string level;
        int numOfNodesX;
        int numOfNodesY;
        int processorConcentration;
        int numPorts;    //in plane
        int numPSEports;
        int type;
        double O_frequency_cntrl;
        int electronicChannelWidth;
        int routerBufferSize;
        int numX;
        int numY;
        int switchVariant = default(0);
        int dispValueX;
        int dispValueY;
        @display("i=abstract/router;bgb=355,340");
    gates:
        input portIn[numPorts];
        output portOut[numPorts];
        output toPSE[numPSEports];
    submodules:[]
    connections allowunconnected:
        for i=0..numPorts-1 {
            in[i].toXbar --> xbar.in[i];
            xbar.out[i] --> ackMux[i].in[0];
            in[i].ackOut --> ackMux[i].in[1];
            ackMux[i].out --> portOut[i];
            portIn[i] --> in[i].in;
            arb.req[i] <--> in[i].req;
        }
        arb.XbarCntrl <--> xbar.cntrl;
        for i=0..numPSEports-1 {
            toPSE[i] <-- arb.pseCntrl[i];
        }
}
```

Electronic Router
Component module

```
    connections allowunconnected:
        for i=0..numPorts-1 {
            in[i].toXbar --> xbar.in[i];
            xbar.out[i] --> ackMux[i].in[0];
            in[i].ackOut --> ackMux[i].in[1];
            ackMux[i].out --> portOut[i];
            portIn[i] --> in[i].in;
            arb.req[i] <--> in[i].req;
        }
        arb.XbarCntrl <--> xbar.cntrl;
        for i=0..numPSEports-1 {
            toPSE[i] <-- arb.pseCntrl[i];
        }
}
```

Connection between all modules

RouterInport C++ implementation

```
class RouterInport : public cSimpleModule {
public:
    RouterInport();
    virtual ~RouterInport();

protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void finish();
    void sendRequest(ElectronicMessage* emsg, int vc);
    int numVC;
    int flit_width;
    int buffer_size;
    double clockPeriod;

    StatObject* P_static;
    StatObject* E_dynamic;
    simtime_t lastTime;
    int myId;
    map<int, queue<ElectronicMessage*>* > buffs;
    cGate* inport;
    cGate* outport;
    cGate* reqportIn;
    cGate* reqportOut;
    cGate* ackport;

#ifdef ENABLE_ORION
    ORION_Array* power;
    ORION_Array_Info* info;

#endif
};

#endif /* ROUTERINPORT_H_ */
```

RouterInport.h

Defines different methods.

In addition, it contains the gates definitions.

```
void RouterInport::sendRequest(ElectronicMessage* emsg, int vc) {
    ArbiterRequestMsg* req = new ArbiterRequestMsg();
    req->setBcast(emsg->getBcast());
    if (req->getBcast()) {
        req->setStage(0);
    } else {
        req->setStage(10000);
    }
    req->setType(router_arb_req);
    req->setVc(vc);
    req->setDest(emsg->getDstId());
    req->setSrc(emsg->getSrcId());
    //req->setDestType(emsg->getDstType());
    req->setReqType(emsg->getMsgType());
    req->setPortIn(myId);
    req->setSize(emsg->getBitLength());
    req->setTimestamp(simTime());
    req->setMsgId(emsg->getId());
    req->setData((long) emsg->getEncapsulatedPacket());
    sendDelayed(req, clockPeriod, reqportOut);
}
```

RouterInport.cc

Example of sendRequest() method. This method implements how the input buffer prepares a request and sends it to the arbiter module through the predefined gate (reqportOut)

```
void RouterInport::initialize() {
    numVC = par("routerVirtualChannels");
    flit_width = par("electronicChannelWidth");
    buffer_size = par("routerBufferSize");
    clockPeriod = 1.0 / (double) par("clockRate");
    myId = par("myId");
    inport = gate("in");
    inport->setDeliverOnReceptionStart(true);
    outport = gate("toXbar");
    reqportIn = gate("req$i");
    reqportOut = gate("req$o");
    ackport = gate("ackOut");

    for (int i = 0; i < numVC; i++) {
        buffs[i] = new queue<ElectronicMessage*> ();
    }
```

RouterInport.cc

Each simple has initialize() method, where all gates are defined according to their first definition in the .ned file.

In addition, you can initialize the parameters used in this simple module.