# CS/ECE 561: Hardware/Software Design of Embedded Systems
## Fall 2016

# Homework 3: NoC Simulation and Exploration

**Assigned:** 13 Oct, 2016
**Due:** 25 Oct, 2016

**Instructions:**

- Please submit your solutions via Canvas. Submissions should include source code files or text/word/pdf files in a single zip file, with separate folders designated for every problem.
- Some questions might not have a clearly correct or wrong answer. In such cases, grading is based on your arguments and reasoning for arriving at a solution.

---

**Prerequisites for this assignment**

Install SystemC (you should have done so for the first assignment). Download Noxim (in the attached file with the assignment) and install it. Look at the INSTALL.txt and MANUAL.txt in the doc directory of the Noxim tar file to familiarize yourself with installation procedure and learn how to use Noxim (i.e., with various command line options). Make sure to follow instructions to change the path of the SYSTEMC variable in the Makefile.defs before compiling. And, after you change the path of the SYSTEMC variable in the Makefile.defs, add the line *TARGET_ARCH = linux* (or linux64 if you are using a 64 bit machine) below SYSTEMC variable.

Also, you may need to set the $LD_LIBRARY_PATH variable to point to your SystemC library directory, if you get an error of the form: "error while loading shared libraries: libsystemc-2.3.1.so: cannot open shared object file: No such file or directory". On Ubuntu, the command to use would look like the following (without quotes): "export LD_LIBRARY_PATH /home/john/systemc-2.3.1/lib-linux:$LD_LIBRARY_PATH" where you should replace /home/john with your own path that contains your SystemC installation.

**Q1 [50 points].** Change the packet injection rate (pir) from 0.01 to 0.1 in increments of 0.01, and record the global average latency, global average throughput, total received packets, and total energy for each pir value in a table, for the default simulation environment. Use a poisson traffic distribution. Look at Manual.txt for help.

(a) What is the relationship between pir, average packet delay, throughput, and energy/packet?
(b) Repeat the above simulations for 100000 and 500000 cycles. What is the relationship between simulation cycles, pir, average packet delay, throughput, and energy/packet?

**Q2 [75 points].** During NoC design, it is often essential to explore a huge design space and select the best parameters (e.g., routing protocol, selection functions, etc). We will use the Noxim Explorer to help perform design space exploration. Switch to the /other directory, do a 'make' to build the 'noxim_explorer' tool, and then edit the sim.cfg file. In the sim.cfg file, change (i) the path of the simulator from '../noxim' to '../bin/noxim'; (ii) number of simulation cycles to 100000; (iii) number of repetitions to 1.

NOTE: If you are using a 64 bit machine, you may get an error such as:

```
terminate called after throwing an instance of 'std::out_of_range'
  what():  basic_string::substr: __pos (which is 4294967320) > this->size() (which is 0)
Aborted (core dumped)
```

To work around this error, comment out the two lines starting from line 536 in noxim_explorer.cpp (*uint pos; pos = line.find(RPACKETS_LABEL);*) and insert the following line in their place before you do make.

***std::size_t pos = line.find(RPACKETS_LABEL);***

(a) When we use adaptive routing algorithms (such as Odd-Even), the selection technique determines which of the multiple possible paths will be selected while a flit is being routed. Which selection technique is the best (i.e., lowest average packet latency) out of the ones being explored for the Odd-Even routing algorithm? To get the answer, you must run noxim_explorer with your modified sim.cfg file as an argument. Note that even if you do not have Matlab, you can easily access data in the generated output files.

(b) Perform an exploration that compares the following routing techniques: XY, negativefirst, North-last, Odd-even, and fully adaptive. Chose only the random selection technique (remove the other selection techniques), and keep all other exploration parameters the same as in (a). Which routing technique gives the best results (in terms of lowest average packet latency)? Hint: plot average latency vs. pir to get the answer.

(c) Now let's explore the impact of the type of traffic in the NoC. Which routing techniques from (b) are superior for random, shuffle, and transpose2 traffic (in terms of average packet latency)?

**Q3 [100 points].** In this question, you will be required to make modifications to the Noxim simulator source code, to add new functionality. Let's start with a tutorial on how to add a fixed priority arbiter to the NoC router. By default, Noxim implements an oblivious arbiter in NoC routers. In the implemented oblivious arbiter, the priority is rotated by one position for every successive arbitration cycle. A fixed priority router keeps the same priority across successive arbitration cycles.

Start by going to the 'src' directory and edit NoximRouter.cpp. Comment out 'start_from_port++'.

```
//start_from_port++;

// 2nd phase: Forwarding
for (int i = 0; i < DIRECTIONS + 1; i++) {
    if (!buffer[i].IsEmpty()) {
```

Save the file, and recompile the tool (go to bin directory and 'make'). Now your simulations will use a fixed priority arbiter, instead of the oblivious one. Let us make the system more flexible and provide command line options to select among the arbiters. Open NoximMain.h and add parameters for the arbiters.

```
// Arbiter types
#define ARBITER_OBLIVIOUS       0
#define ARBITER_FIXED_PRIORITY  1
#define INVALID_ARBITER        -1
```

Set the default arbiter as the oblivious arbiter.

```
#define DEFAULT_ARBITER              ARBITER_OBLIVIOUS
```

Add a parameter for the arbiter in the NoximGlobalParams struct

```
// NoximGlobalParams -- used to forward configuration to every sub-block
struct NoximGlobalParams {
    ...
    static int arbiter;
};
```

Save the file. Now open NoximMain.cpp to initialize this parameter.

```
// Initialize global configuration parameters (can be overridden with command-line arguments)
int NoximGlobalParams::arbiter = DEFAULT_ARBITER;
```

Now open NoximCmdLineParser.cpp to add command line options to select the arbiter. Add the following else if block

```
else if (!strcmp(arg_vet[i], "-arbiter")) {
    char *arbiter = arg_vet[++i];
    if (!strcmp(arbiter, "oblivious"))
        NoximGlobalParams::arbiter = ARBITER_OBLIVIOUS;
    else if (!strcmp(arbiter, "fixed"))
        NoximGlobalParams::arbiter = ARBITER_FIXED_PRIORITY;
    else
        NoximGlobalParams::arbiter = INVALID_ARBITER;
}
```

Finally, edit the NoximRouter.cpp file again to add support for these arbiter parameter options. Replace the line with 'start_from_port++' with the following

```
if (NoximGlobalParams::arbiter = ARBITER_OBLIVIOUS)
    start_from_port++;
else if (NoximGlobalParams::arbiter = ARBITER_FIXED_PRIORITY)
    start_from_port = DIRECTION_NORTH;
else
    assert(false);
```

Save the file. Recompile the tool by performing a 'make'. Congratulations – you just enhanced the Noxim simulator!

(a) Compare the fixed priority router with the oblivious arbiter by plotting pir against average packet latency. Explore for the following default parameters in the NoC:

- Traffic: random, shuffle, butterfly
- Routing: XY
- Selection: nop
- Packet size [4 12]
- Buffer size 4
- Repetitions 5
- Simulation time: 100000
- Warm up time 10000

(b) Implement the YX routing scheme, with command line support. As in our example for adding the fixed priority arbiter, this task will require modifications to NoximRouter.cpp as well as other files. Compare with the XY routing scheme. Use the environmental parameters from (a) for exploration.