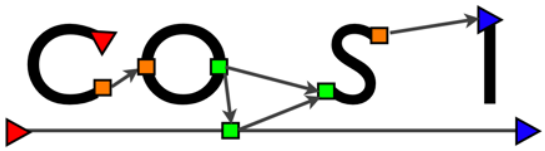

COSI-NoC User Manual
(Version 1.2)

Alessandro Pinto
University of California at Berkeley,
545P Cory Hall, Berkeley, CA 94720
apinto@eecs.berkeley.edu

September 23, 2007

Contents

Chapter 1

Introduction

COSI-NoC is an open source software for the automatic synthesis of networks-on-chip (NoC). COSI-NoC is part of the Communication Synthesis Infrastructure (COSI) under development at the University of California at Berkeley. The goal of the COSI project is to develop a formal framework, and a methodology, to address the problem of synthesizing embedded networks. A Network-on-Chip (NoC) is an embedded network that provides the communication service to a heterogeneous or homogeneous Multi-Processor System on Chip (MPSoC)..

The user of COSI-NoC specifies the communication synthesis problem by providing a description of: (1) the end-to-end communication requirements and (2) the performance/cost trade-off of routers and links.

An end-to-end communication requirement (also called point-to-point communication constraint) defines the minimum required bandwidth and the maximum allowed latency between a source IP core and a destination IP core.

The performance and cost of a point-to-point link are computed analytically. The user can specify the parameters of the model that include the physical characteristics of the silicon technology. For routers, we use Orion to estimate the power consumption and the area occupation.

To synthesize an optimal network, COSI-NoC 1.2 uses an algorithm based on an iterative application of a modified version of Dijkstra shortest path algorithm [Alg00]. The adoption of the simple shortest path algorithm has been advocated by many researchers in this field like in [MMA+06, HGR05]. The main difference with other approaches is the definition of a graph from which the shortest path is drawn. In particular, such graph, called the NoC *platform*, captures the set of all possible NOC implementations that satisfy the design specification constraints.

COSI-NoC takes into account the chip floor-plan, the limited area available for the communication infrastructure, the maximum distance that wires can span on a chip, the constraints on the input and output degree for routers and network interfaces, the constraints imposed by the flow control algorithm like deadlock avoidance, and the maximum throughput of wires and routers.

1.1 Content of the COSI-NoC (v1.2) Distribution

This distribution of COSI-NoC contains the following directory:

- `bin` contains the main executable called `CosiNoc.x`.
- `doc` contains this document.
- `systemc` contains a library of SystemC components to simulate a Network-on-Chip. The components provided with this library are: a model for sources and destinations, a model for routers, a model for point-to-point links implemented as bundle of wires, a simulation monitor.
- `techlib` contains performance and cost parameters of the on-chip communication components at 100,70 and 50 nm.
- `tests` contains three examples: `cmp2x2placed` used as a reference example in this manual, an advanced set-top-box (`ADSTB`), and a `tVOPD` example.

1.2 Installation

The installation of COSI-NoC is a straightforward process. We denote the current directory (where the COSI-NoC `.tgz` tar-ball resides) with `current`. Expand the tar-ball using the command:

```
current > tar -zxvf COSI-NoC.tgz
```

Directory `current` now contains a new directory `COSI-NoC` that contains the source code of the distribution. Change directory to `current/COSI-NoC` .

Before compiling COSI-NoC, you must define the environment variable `COSIROOT` to point to `current/COSI-NoC`. If you are using the bash shell, you can use the following command to set up the environment variable:

```
export COSIROOT=current/COSI-NoC
```

In order to compile the static libraries that are used by COSI-NoC and the executable file `CosiNoc.x`, simply execute the following command:

```
current/COSI-NoC> make
```

Two libraries are generated: `libcosi.a` that contains the objects files of COSI-NoC, and `libtinyxml.a` that contains the objects files of the XML parser that we use (TinyXML) [tin]. One executable is generated:

```
current/COSI-NoC/bin/CosiNoc.x
```

that is the main executable file.

This release contains already a compiled version of PARQUET [AM03]. We are not allowed to distribute the source code. This executable has been compiled under Fedora 6, with g++ 4.1.1. If you want to run COSI-NoC on a different platform you can obtain the source code of PARQUET from

`http://vlsicad.eecs.umich.edu/BK/parquet/`

Remark 1.2.1 (On floor-planning) If the placement of each IP core is known, the user can pass this information directly to COSI-NoC. Section 2.2 explains how to provide a placement of the IP cores as an additional input to the synthesis problem. Section 2.3 explains how to include the placement information in the specification of the communication constraints.

In these two cases, there is no need to floor-plan the chip before synthesizing the NoC. COSI-NoC needs PARQUET only if the chip floor-plan is not known or only partially known.

It is convenient to add the path where the COSI-NoC executable is to your `PATH` environment variable. If you use the `bash` shell, run the following command to add the path:

```
export PATH=$PATH:current/COSI-NoC/bin
```

Before running COSI-NoC, there is one more configuration step to complete. The file `cosiconfig.xml` that is in `current/COSI-NoC` is a configuration file that is read by COSI-NoC every time at the beginning of its execution. The configuration file has the following structure:

```
<?xml version="1.0" ?>
<Configuration>
  <COSI version="NOC 1.2"/>
  <TMP root="/tmp" />
  <SYSTEMC root="/usr/share/systemc-2.1.v1"/>
  <PARQUET root="/home/apinto/Projects/cosi/tools/PARQUET_050330"
                       exec="Parquet" />
</Configuration>
```

The `COSI` element defines the version of COSI-NoC that you are using (and you should not change it unless you make modifications to the infrastructure and you want to distinguish you release from the original one).

The `root` attribute of the `TMP` element defines the directory that shall be used for temporary files. Some independent modules of the COSI-NoC software release exchange information through temporary files. For instance, the communication constraints are parsed by the input module that writes the files needed by the PARQUET floor-planner in the temporary directory. The floor-plan module runs PARQUET that writes the chip placement file in the temporary directory. The placement parser module reads the placement from the temporary directory and initializes a data structure with the position of each IP core.

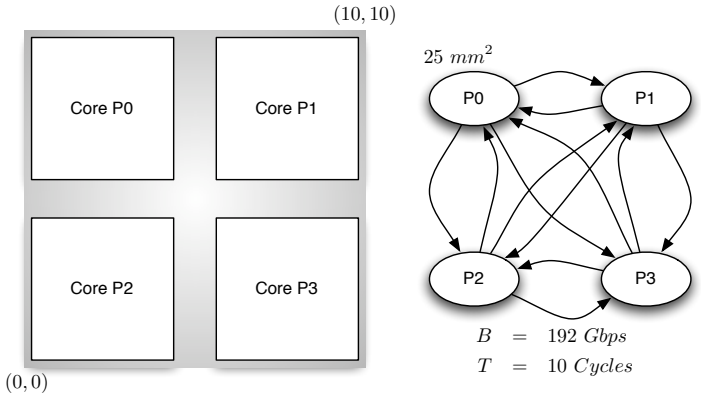

Figure 1.1: Example of a Chip-Multi-Processor (CMP) composed of four processors.

The `root` attribute of the `SYSTEMC` element defines the installation directory of the SystemC library. This information is used by the SystemC code generator of COSI-NoC that writes also a `Makefile` to compile an executable simulation of the synthesized NoC.

The `root` and `exec` attributes of the `PARQUET` element define the installation directory of the PARQUET floor-planner and the executable file to run, respectively. When PARQUET is compiled as a set of dynamically linked library plus an executable file, the directory defined by the `root` attribute is also used to set the `LD_LIBRARY_PATH` environment variable before runningPARQUET.

1.3 Getting Started

In this section we explain how to use COSI-NoC to synthesize an NoC and analyze the synthesis result.

We use a very simple example of a 2×2 Chip Multi-Processor (CMP). We start from the specification of the communication constraints of our chip as shown in Figure 1.1 (all the necessary files for this example can be found in the directory `COSIROOT/tests/cmp2x2placed`).

We assume that all the cores are already placed on the chip so that even if you don't have PARQUET installed you can still follow this example. Each core is 5×5 mm^2 and needs to communicate with all the other cores with an aggregate bandwidth equal to 192 *Gbps*. First, decide the path of two directories: the *input* directory that contains the specification of the communication constraints and the library of communication components and the *output* directory where the synthesis results will be saved. For instance, let these two directories be denoted by `/home/myname/project` and `/home/myname/project/results`. The project file (that we show in a moment), will be saved in `/home/myname/project` (this

is also the directory structure used for all the tests accompanying this release). Create a project file called `project.xml`. The content of a project file is divided in sections: the definition of properties related to the project, the definition of the constraints, the definition of the synthesis parameters and the definition of the type of results that you need. An XML project file starts as follows:

```
<?xml version="1.0" ?>
<Project
   name="cmp2x2"
   input="."
   output="./results" >

<!-- Other things to be added later -->

</Project>
```

The `Project` element defines a project called `cmp2x2`. The `input` and `output` attributes define the directories where the input files and output files are saved, respectively.

The communication constraints and the library of communication components are going to be described in two separate files. The name of such files are defined by two elements in the project file:

```
<?xml version="1.0" ?>
<Project
   name="cmp2x2"
   input="."
   output="./results" >

   <Constraints file="constraints.xml" />

   <Library
      name="Mylib"
      file="mylib.xml" />

<!-- Other things to be added later -->

</Project>
```

COSI-NoC parses the communication constraints from
`/home/myname/project/constraints.xml`
and the description of the library components from
`/home/myname/project/mylib.xml`.
The constraint file contains the specification of all the cores and all the end-to-end communication requirements among the cores. Create the file
`/home/myname/project/constraints.xml`
and start with the following elements:

```
<?xml version="1.0" ?>
<Constraints>
  <Core
     name="P0" type="Placed"
     xbl="0" ybl="0"
     xtr="5000" ytr="5000"
  />
  <Core
     name="P1" type="Placed"
     xbl="5000" ybl="0"
     xtr="10000" ytr="5000"
  />
  <Core
     name="P2" type="Placed"
     xbl="0" ybl="5000"
     xtr="5000" ytr="10000"
  />
  <Core
     name="P3" type="Placed"
     xbl="5000" ybl="5000"
     xtr="10000" ytr="10000"
  />

<!-- PtP constraints to be added here -->

</Constraints>
```

We have just defined the IP cores. Each core has a name and a type. In this case all the cores are already placed on the chip and they are characterized by the bottom left and top right corner coordinates in μm units.

Communication constraints are included as follows:

```
<?xml version="1.0" ?>
<Constraints>
  <Core
     name="P0" type="Placed"
     xbl="0" ybl="0"
     xtr="5000" ytr="5000"
  />
  <Core
     name="P1" type="Placed"
     xbl="5000" ybl="0"
     xtr="10000" ytr="5000"
  />
  <Core
     name="P2" type="Placed"
     xbl="0" ybl="5000"
```

```
    xtr="5000" ytr="10000"
  />
  <Core
    name="P3" type="Placed"
    xbl="5000" ybl="5000"
    xtr="10000" ytr="10000"
  />

  <Constraint name="P0P1"
              source="P0" dest="P1" bw="192000000000" T="10" />
  <Constraint name="P0P2"
              source="P0" dest="P2" bw="192000000000" T="10" />
  <Constraint name="P0P3"
              source="P0" dest="P3" bw="192000000000" T="10" />
  <Constraint name="P1P2"
              source="P1" dest="P2" bw="192000000000" T="10" />
  <Constraint name="P1P3"
              source="P1" dest="P3" bw="192000000000" T="10" />
  <Constraint name="P1P0"
              source="P1" dest="P0" bw="192000000000" T="10" />
  <Constraint name="P2P3"
              source="P2" dest="P3" bw="192000000000" T="10" />
  <Constraint name="P2P0"
              source="P2" dest="P0" bw="192000000000" T="10" />
  <Constraint name="P2P1"
              source="P2" dest="P1" bw="192000000000" T="10" />
  <Constraint name="P3P0"
              source="P3" dest="P0" bw="192000000000" T="10" />
  <Constraint name="P3P1"
              source="P3" dest="P1" bw="192000000000" T="10" />
  <Constraint name="P3P2"
              source="P3" dest="P2" bw="192000000000" T="10" />

<!-- Mutual exclusions to be added here  -->

</Constraints>
```

Each constraint is identified by a name, a source core, a destination core, a minimum bandwidth and a maximum latency requirements. The last (optional) set of elements are mutual exclusions among constraints. We know that each processor can only send packets to one other processor at the time. This implies that, for instance, constraints `P0P1` and `P0P2` are mutually exclusive (see Section 2.3 for an explanation of the meaning of mutual exclusions). The complete set of constraints is defined by the following xml file:

```
<?xml version="1.0" ?>
<Constraints>
```

```
<Core
   name="P0" type="Placed"
   xbl="0" ybl="0"
   xtr="5000" ytr="5000"
/>
<Core
   name="P1" type="Placed"
   xbl="5000" ybl="0"
   xtr="10000" ytr="5000"
/>
<Core
   name="P2" type="Placed"
   xbl="0" ybl="5000"
   xtr="5000" ytr="10000"
/>
<Core
   name="P3" type="Placed"
   xbl="5000" ybl="5000"
   xtr="10000" ytr="10000"
/>

<Constraint name="P0P1"
            source="P0" dest="P1" bw="192000000000" T="10" />
<Constraint name="P0P2"
            source="P0" dest="P2" bw="192000000000" T="10" />
<Constraint name="P0P3"
            source="P0" dest="P3" bw="192000000000" T="10" />
<Constraint name="P1P2"
            source="P1" dest="P2" bw="192000000000" T="10" />
<Constraint name="P1P3"
            source="P1" dest="P3" bw="192000000000" T="10" />
<Constraint name="P1P0"
            source="P1" dest="P0" bw="192000000000" T="10" />
<Constraint name="P2P3"
            source="P2" dest="P3" bw="192000000000" T="10" />
<Constraint name="P2P0"
            source="P2" dest="P0" bw="192000000000" T="10" />
<Constraint name="P2P1"
            source="P2" dest="P1" bw="192000000000" T="10" />
<Constraint name="P3P0"
            source="P3" dest="P0" bw="192000000000" T="10" />
<Constraint name="P3P1"
            source="P3" dest="P1" bw="192000000000" T="10" />
<Constraint name="P3P2"
            source="P3" dest="P2" bw="192000000000" T="10" />
```

```
  <Exclusion set="P0P1 P0P2 P0P3"/>
  <Exclusion set="P1P2 P1P3 P1P0"/>
  <Exclusion set="P2P3 P2P0 P2P1"/>
  <Exclusion set="P3P0 P3P1 P3P2"/>

</Constraints>
```

The library of communication components, that include wires and routers, are all defined in the library file. The library file contains three kinds of information: the parameters of the silicon technology, the characterization of the copper wires and the characterization of the on-chip routers as a function of the number of inputs and outputs.

```
<?xml version="1.0" ?>
<Library name="Mylib">

<Technology fclk="1.6e9" vdd="1.2" wmin="200e-9"
            ioff="0.15" isc="65e-6"
            r0="10.0e3" cp="2.5e-15"
            c0="1.5e-15"
            nlayers="7" />

<!-- Wires and Routers to be added here -->

</Library>
```

Each library is identified by a unique name. The first element that we have defined specifies many parameters of the silicon technology. The details about these parameters can be found in Section 2.4. The parameters here refer to a 100 *nm* technology.

The other elements that can be added to the library specification are the wires and the routers. For instance, we add the characterization of a global wire (Metal 6):

```
<?xml version="1.0" ?>
<Library name="Mylib">

  <Technology fclk="1.6e9" vdd="1.2" wmin="200e-9"
              ioff="0.15" isc="65e-6"
              r0="10.0e3" cp="2.5e-15"
              c0="1.5e-15"
              nlayers="7" />

  <Wire
     type="copper" layer="6"
     r="103.9e3" c="154.0e-12" pitch="460e-9" />
```

```
<!-- Routers to be added here -->

</Library>
```

For each wire, three attributes must be specified: the resistance and capacitance per unit length and the wire pitch that is used to compute the area occupied by a point-to-point link. The last component to characterize is the on-chip router. While the model that we use for wires is an analytical model, routers are characterized by tables that give the energy-per-flit and area of a router depending on the number of input and output ports.

```
<?xml version="1.0" ?>
<Library name="Mylib">

  <Technology fclk="1.6e9" vdd="1.2" wmin="200e-9"
              ioff="0.15" isc="65e-6"
              r0="10.0e3" cp="2.5e-15"
              c0="1.5e-15"
              nlayers="7" />

  <Wire
     type="copper" layer="6"
     r="103.9e3" c="154.0e-12" pitch="460e-9" />

  <Router maxin="3" maxout="3" maxbw="1.6e9"

    energy ="1.8e-11 1.2e-12  2e-11 2e-12       2.3e-11 2.7e-12
             3.3e-11 2e-12    4.7e-11 3.3e-12   5.2e-11 4.5e-12
             4.7e-11 3e-12    6.4e-11 4.6e-12   8.7e-11 6.2e-12"

    area= "7168   25600  34816
           32768  51200  69632
           49152  76800  104448"
  />
</Library>
```

The values are approximations of the real numbers that we previously derived using ORION [Wan02]. The energy consumption are expressed in *Joule* considering a switching factor equal to one. The area is expressed in μ^2. Entry (i, j) of these matrices characterize a router with i inputs and j outputs.

Remark 1.3.1 (On network interfaces.) In this example we are assuming that network interfaces have the same cost of the routers. We are aware of the fact that interfaces might have a different cost and we will add more accurate interface models in future releases.

Before running the synthesis we need to complete the project description by adding input parameters/constraints and specifying the desired outputs.

```
<?xml version="1.0" ?>
<Project
   name="cmp2x2"
   input="."
   output="./results" >

   <Constraints
      file="constraints.xml" />

   <Library
      name="Mylib"
      file="mylib.xml" />

   <Parameters
     switchingfactor="0.5"
     maxindegree="3"
     maxoutdegree="3"
     allowptp="0"
     density="50"
     powervsarea="1"
     />

<!-- Other things to be added later -->

</Project>
```

The parameters that we have just added define:

- the switching factor used to compute the power consumption;
- the maximum input degree and output degree of each router;
- a switch that does not allow point-to-point (one hop) connections between a source and a destination;
- the density of points that are considered as potential installation points for routers (expresses in number of points per mm^2);
- a parameter that balances the importance of minimizing power vs minimizing area (if P is the power consumption and A the area occupied by the network, the cost function used in this case is $\lambda P + (1 - \lambda)A$ where λ is defined by the `powervsarea` attribute. The cost function is not always defined in this way but depends on the optimization strategy chosen by th user).

All this parameters have defaults values, and other parameters can be specified (for a complete list of synthesis parameters see Section 2.5).

The last part of the project definition is used to ask COSI-NoC to generate some outputs (for a complete list of output options see Section 2.6):

```
<?xml version="1.0" ?>
<Project
   name="cmp2x2"
   input="."
   output="./results" >

   <Constraints
      file="constraints.xml" />

   <Library
      name="Mylib"
      file="mylib.xml" />

   <Parameters
     switchingfactor="0.5"
     maxindegree="3"
     maxoutdegree="3"
     allowptp="0"
     powervsarea="1"
     />
    <Output>
      <Svg name="cmp2x2.svg" />
      <SystemC name="cmp2x2.cpp" mk="cmp2x2.mk" />
      <Report name="cmp2x2.rep" />
      <Dot name="cmp2x2.dot" />
  </Output>

</Project>
```

We are asking COSI-NoC to generate an Svg graphical representation of the network that is going to be saved as `/home/myname/project/results/cmp2x2.svg`. We are also asking to generate other outputs like a textual report `/home/myname/project/results/cmp2x2.rep` and a SystemC simulator `/home/myname/project/results/cmp2x2.cpp`.

In order to run a synthesis you first need to make sure that the `COSIROOT` environment variable is set. If it is not, use the following command:

```
>export COSIROOT=current/COSI-NOC
```

also, you might want to add the directory where the COSI-NoC executable is located to your `PATH` environment variable:

```
> export PATH=$PATH:current/COSI-NOC/bin
```

In order to run the synthesis, use the following command:

```
/home/myname/project> CosiNoc.x project.xml
```

After the synthesis is completed, the results can be found in the directory `/home/myname/project/results`. To know how to use and interpret the results, please refer to Section 3.

With the project file that we have just set up, the synthesis ends with the following message:

```
The problem is not feasible,
please increase the number of input ports of a destination
Destination P0 input bandwidth 5.76e+11
            maximum input bandwidth 2.048e+11
```

Let's revisit the synthesis parameters that we specified. Among these, we didn't include two parameters that define the maximum outdegree of a source and the maximum indegree of a destination. We only specified these values for the routers. The default values for the source maximum output degree and for the destination maximum input degree is 1. Since our target clock frequency is $1.6GHz$, the capacity of a point-to-point connection is 1.6 $Gflitps$ that, for a flit with width equal to 128 corresponds to 204.8 $Gbps$. This means that each link can only carry one constraint. For a source, one output connection only is sufficient because the constraints out of a source are mutually exclusive. On the other hand, each destination receives three non exclusive inputs that cannot share the same link, thus requiring three distinct input ports. In order to account for this modify the parameters as follows:

```
<Parameters
     switchingfactor="0.5"
     maxindegree="3"
     maxoutdegree="3"
     destindegree="3"
     allowptp="0"
     powervsarea="1"
     />
```

and run the synthesis again. The number of points considered during synthesis is about 1000. This synthesis takes about 100 second to complete on an Intel Core Due @ $2GHz$. Step into directory `/home/myname/project/results`.

File `cmp2x2.rep` contains a textual description of the synthesis result. It contains many interesting properties of the network like the number average number of hops for all point-to-point connections, the power consumption and the area occupation. For instance, the power report for this example looks as follows:

```
Power report
  Wires :
    Dynamic power = 4.66758 W
    Static  power = 0.00787755 W
    Total   power = 4.67546 W
  Routers :
```

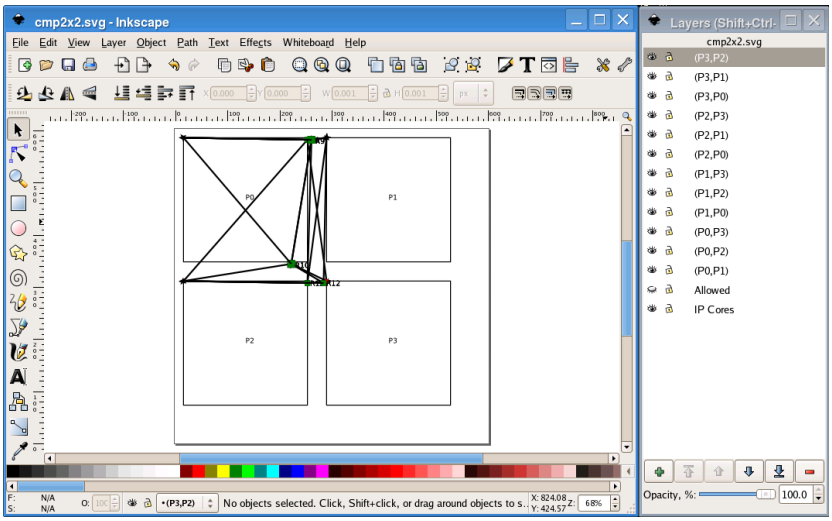

Figure 1.2: Visualization of the NoC using INKSCAPE.

```
  Dynamic power = 0.42825 W
  Static  power = 0.02736 W
  Total   power = 0.45561 W
                  ---------------------
Network dynamic power =   5.09583
Network power =   5.13107
```

Among the many file that you can in this directory, you can visualize Svg file containing the graphical representation of the synthesized network using INKSCAPE.

Figure 1.2 shows the graphical representation of the synthesis result using INKSCAPE. Each source to destination path is saved as a different layer in order to be able to visualize the choices taken by the algorithm. Routers (depicted as green rectangles) and links (depicted as black arrows) are drawn to scale. Notice that the representation of links is only logical in the sense that the real layout will route wires only along the x and y coordinates (i.e. Manhattan routing).

The logical structure of the network is saved in `cmp2x2.dot`. Figure 1.3 shows the graphical representation of the logical structure of the network using DOTTY. Each core is represented simply by a square. Routers are represented by records. Each record represents one entry of the routing table. The first row of the record corresponds to the source-to-destination flows that cross the router while the second row correspond to the next hop the packets should be forwarded to.

Finally, it is possible to generate an executable simulation of the NoC. Enter the directory `/home/myname/project/results` and type:

```
make -f cmp2x2.mk
```

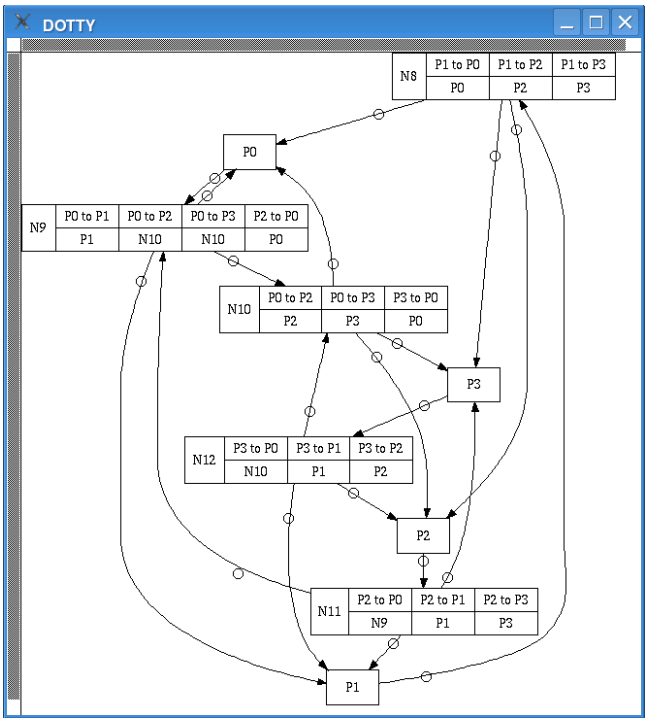

Figure 1.3: Visualization of the NoC using DOTTY.

an executable file called `cmp2x2.x` is generated. Running the simulation will print out the simulation results at run-time. In particular, the average bandwidth and delay for each constraint is printed to screen. The following snippet is the report relative to the destination core `P4`:

```
Bandwidth report for D4
        Average bandwidth from source 1 = 1.88975e+10
        Average bandwidth from source 2 = 1.88936e+10
        Average bandwidth from source 3 = 1.88923e+10
Delay report for D4
        Average latency from source 1 = 4.53227
        Average latency from source 2 = 5.0427
        Average latency from source 3 = 5.21759
```

The report shows the average bandwidth from each source and the average delay of packets from each source.

Chapter 2

The Input Format

COSI-NoC takes as input an XML [XML] file called *project*. A project contains the description of the communication synthesis problem, i.e. the input that are necessary to the COSI-NoC synthesis engine. The structure of a COSI-NoC XMLproject is shown in Figure 2.1.

The root element is the `Project` that is made of up to four elements. The elements that must be present are the `Constraints`, the `Library` and the `Parameters`. These three elements are necessary for the synthesis: the constraints define the problem in terms of the IP cores and the communication requirements among them; the library defines the available on-chip communication components and the rules to install and connect them; the parameters fix some constants and options that are needed by the synthesis algorithm.

The output element is optional. There are many possible outputs that a user can ask COSI-NoC to generate. Some of them are graphical or logical representation of the NOC and can be effectively used to have an idea of the NOC complexity and of the decisions that the algorithm has made. The SystemC [Sys] output is very useful to simulate the NOC under synthetic or real traffic conditions. Notice that if no output option is specified then no output is generated by COSI-NoC.

This chapter describes the COSI-NoC input format, more specifically: how a project is specified, how constraints are defined, how the library is described, and how the user specifies the output to be generated.

2.1 Configuration of COSI-NoC

The configuration file must be placed in the root directory of the distribution. This file is written in XML format and must be called `cosiconfig.xml`. An example of the content of the configuration files is the following:

```
<?xml version="1.0" ?>
<Configuration>
  <COSI version="NOC 1.2"/>
```

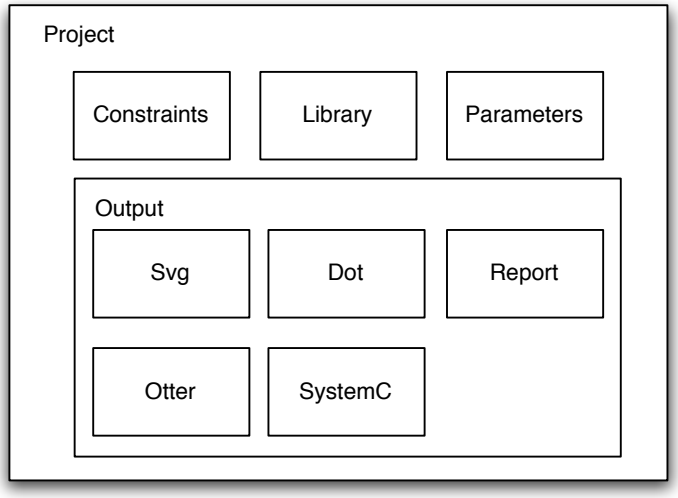

Figure 2.1: Xml structure of a COSI-NoC project.

```
  <TMP root="/tmp" />
  <SYSTEMC root="/usr/share/systemc-2.1.v1"/>
  <PARQUET root="/home/apinto/Projects/cosi/tools/PARQUET_050330"
           exec="Parquet.exe" />
</Configuration>
```

The `TMP` element is a temporary directory where COSI-NoC saves all temporary files used to communicated with external tools like PARQUET. The `SYSTEMC` element defines the installation directory of SystemC. This element is used in the generation of the makefile to compile the SystemC simulator that can be generated as one of the results of the NoC synthesis. The `PARQUET` element defines the installation directory of the PARQUET floor-planner and the name of the executable file to run.

A configuration file is already provided with the current distribution of COSI-NoC. The user needs only to change the pointers to the SystemC and PARQUET distributions.

2.2 Description of a COSI-NoC Project

A project is an xml description of a communication synthesis task. We use the simple example shown in Figure 1.1 to explain all the different parts that compose the synthesis task specification.

This example represents a Chip-Multi-Processor (CMP) composed of four cores. Each of them is a processor of specific area (25 mm^2 in our example).

Communication requirements among the processors are captured by a point-to-point graph. Each arc corresponds to an end-to-end communication constraint that must be satisfied. The constraints that we consider are bandwidth and latency. For instance, Figure 1.1 explicitly shows a communication constraint between `P2` and `P3`: the required minimum bandwidth is 192 *Gbps* and the maximum delay is 10 [1]. The project file for this example is the following:

The root element, specified by the tag `Project`, defines the project. Its attributes are the project `name`, the directory where all the input files are stored `inputdir` and the directory where all the output files should be saved `outputdir`. The attribute `inputdir` is considered a prefix for all input files (i.e the file containing the constraints and the file containing the library description), and `outputdir` is considered a prefix for all output files (i.e. any output that the user asks COSI-NoC to generate).

The `Project` element contains four elements:

- the specification of the point-to-point constrains marked with the tag `Constraints`. This information is stored in a separate XMLfile. The attribute of this elements is just a pointer to an external XMLfile that describes the communication constraints (refer to Section 2.3).
- The description of the library components marked with the tag `Library`. This information is also stored into a separate XMLfile. The attribute of this element is just a pointer to an external XMLfile that contains the description of the library elements (refer to Section 2.4).
- The set of synthesis parameters marked with the tag `Parameters` (refer to Section 2.5).
- The specification of the desired outputs marked with the tag `Output`. COSI-NoC can generate many different outputs for analysis, visualization, and simulation purposes (refer to Section 3).

2.3 Specification of the Communication Constraints

The specification of the communication synthesis problem comprises two parts: the communication agents and their interaction. In the case of NOC synthesis, the communication agents are intellectual property (IP) cores on the chip. Their physical sizes are treated as constraints to the communication synthesis engine. In fact, once the floor-plan has been decided, each IP core becomes a piece of logic on the chip that cannot be touched. This means that additional circuitry for the communication infrastructure can only be installed in those parts of the chip that are not already occupied by any IP core.

End-to-end communication constraints are captured by a set of arcs. Each arc connects a source IP core to a destination IP core. In the specification file,

[1] Currently, this number is interpreted as the maximum number of hops. For a different delay model (e.g. a cycle accurate delay model of routers) its meaning can change.

```
<?xml version="1.0" ?>
<Project name="cmp2x2"
 input="."
 output="./results" >

  <Constraints file="constraints.xml" useplacement="Soc_placed.pl" />

  <Library name="100nm4ch128" file="library100nm.xml" />

  <Parameters

     sparearea="0.25"
     step="0.01"
     flitwidth="128"
     switchingfactor="0.5"
     maxindegree="3"
     maxoutdegree="3"
     destindegree="3"
     sourceoutdegree="1"
     allowptp="1"
     allowtwohops="1"
     powervsarea="1.0"
     hopconstraints="0"
     areaconstraint="0"
     area="1"
     density="10"
  />

  <Output>
    <Svg name="cmp2x2.svg" />
    <SystemC name="cmp2x2.cpp" mk="cmp2x2.mk" />
    <Report name="cmp2x2.rep" />
    <TabAppend name="cmp.txt" />
    <Dot name="cmp2x2.dot" />
    <Otter name="cmp2x2.odf" />
  </Output>

</Project>
```

Figure 2.2: Example of COSI-NoCproject file.

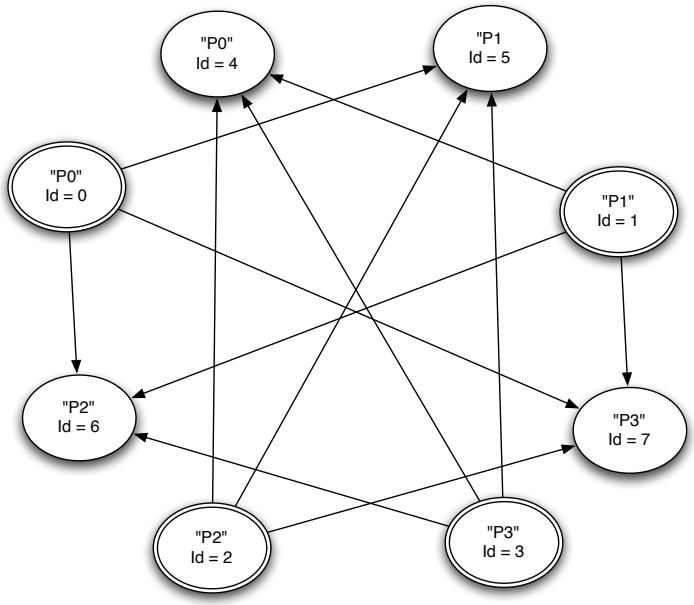

Figure 2.3: Bipartite graph representation of the communication constraints.

sources and destinations are denoted by string identifiers. At the specification level, each identifier denotes an IP core, therefore it is possible to assign the same identifier to a source and a destination. When the specification is parsed, a bipartite graph is generated where sources and destinations are distinct. Each node represents a directional interface (input or output but not bidirectional). Figure 2.3 shows the bipartite graph representation. Each node is an interface (sources are denoted by double circles) of an IP core that can have multiple inputs or multiple outputs but neither bidirectional nor mixed-type ports. By setting the synthesis parameters properly, the user can force the final implementation to have one port per core or many (see Section 2.5). Each node in the bipartite graph has a name (the same name of the IP core) and a unique integer identifier.

Figure 2.4 shows the XML description of the communication constraints for the 2×2 CMP. The root element is marked with the tag `Constraints`. It contains three type of elements: `Core`, `Constraint` and `Exclusion`.

A `Core` element specifies an IP core with the following attributes:

- `name` is a string that uniquely identifies the IP core.
- `type` refers to the type of the specification. It can be `Area` or `Placed`. If the value of this attribute is `Placed` then the core has a fixed position on the chip and a fixed horizontal and vertical dimension. In this case four additional attributes must be specified: `xbl` , `ybl` , `xtr` and `ytr`

```
<?xml version="1.0" ?>
<Constraints>
 <!--
 This test case contains four processors P0,P1,P2 and P3.
 Each processor is 25mm^2.
 Each processor needs to communicate with all other
 processors at 192Gb/s. Constraints having the same source
 are mutually exclusive.
 -->

 <Core type="Area" area="25000000" name="P0" />
 <Core type="Area" area="25000000" name="P1" />
 <Core type="Area" area="25000000" name="P2" />
 <Core type="Area" area="25000000" name="P3" />

 <Constraint name="P0P1"
   source="P0" dest="P1" bw="192000000000" T="10" />
 <Constraint name="P0P2"
   source="P0" dest="P2" bw="192000000000" T="10" />
 <Constraint name="P0P3"
   source="P0" dest="P3" bw="192000000000" T="10" />
 <Constraint name="P1P2"
   source="P1" dest="P2" bw="192000000000" T="10" />
 <Constraint name="P1P3"
   source="P1" dest="P3" bw="192000000000" T="10" />
 <Constraint name="P1P0"
   source="P1" dest="P0" bw="192000000000" T="10" />
 <Constraint name="P2P3"
   source="P2" dest="P3" bw="192000000000" T="10" />
 <Constraint name="P2P0"
   source="P2" dest="P0" bw="192000000000" T="10" />
 <Constraint name="P2P1"
   source="P2" dest="P1" bw="192000000000" T="10" />
 <Constraint name="P3P0"
   source="P3" dest="P0" bw="192000000000" T="10" />
 <Constraint name="P3P1"
   source="P3" dest="P1" bw="192000000000" T="10" />
 <Constraint name="P3P2"
   source="P3" dest="P2" bw="192000000000"T="10" />

 <Exclusion set="P0P1 P0P2 P0P3"/>
 <Exclusion set="P1P2 P1P3 P1P0"/>
 <Exclusion set="P2P3 P2P0 P2P1"/>
 <Exclusion set="P3P0 P3P1 P3P2"/>

</Constraints>
```

Figure 2.4: XML description of the communication constraints.

that are the coordinates of the bottom-left and top-right corner of the IP, respectively. If the value of this attribute is `Area` then the position of the IP core is not fixed and must be determined before synthesis using a floor-planner (we use PARQUET [AM03] to floor-plan a chip). If this is the case, an `area` attribute must be specified. Positions must be expressed in μm and areas in μm^2.

Remark 2.3.1 (Technical remark on the IP fixed placement) . When the type for an IP core is specified `Placed`, the IP core is placed in a specific position on the chip and the vertical and horizontal dimensions are also fixed. As every IP core in a specification is passed to the floor-planner, we need to make sure that a specific position gets assigned to the lower left corner of each placed IP and that the dimensions are also preserved. The way in which we achieve this is to tell the floor-planner that each placed IP is a hard rectilinear block defined by four vertices (the four corners of the IP). In order to place the IP in the desired position, we insert a pad for each placed IP in such position and assign a very high weight to the net connecting the pad and the IP.

When the prefix of a core name is `CosiComm`, the core is considered a spare block used only for the communication infrastructure. During floor-planning, these cores are treated as any other core: they can be placed or not. In many practical cases, the communication designer is assigned a fixed position for a few cores (typically the off-chip memory interface and the CPUs), and a fixed position for a few areas that can be used for communication. Our input format supports all such cases.

An end-to-end communication requirement is specified by a `Constraint` element, which has the following attributes:

- `name` is a string identifier. Each constraint should have a unique identifies that can be referenced later to define exclusion sets.
- `source` is the identifier of the IP core that is the source of this constraint.
- `dest` is the identifier of the IP core that is the destination of this constraint.
- `bw` is the bandwidth requirement in *bps*.
- `T` is the latency requirement in number of hops (recall that COSI-NoCcurrently supports only synchronous implementations of NOCs where each router operates at the same clock frequency).

For each constraint, both the source and the destination IP core identifiers must have been specified in the `name` attribute of some previously defined IP core.

Usually, a source IP core communicates with many other sources. To specify that some data must be broadcasted to several destinations it is equivalent to say that all the corresponding constraints are active at the same time. If broadcast communication is not supported, then the IP core can send data to one destination at the time. In order to model this situation, the user can define sets of constraints that are mutually exclusive using an `Exclusion` element.

The semantics of exclusion sets is tightly related to the dynamic behavior of the application. Two constraints that are in an exclusion set are never "active" at the same time. Since we don't explicitly capture the notion of time, this statement simply means that when two paths, that implement a pair of mutually-exclusive constraints, share the same resources only the one with tighter constraints determines the performance/cost trade-off of such resources. An `Exclusion` element has one attribute:

- `set` is a space-separated list of constraint identifiers.

Remark 2.3.2 (On exclusion sets) One might think that exclusion sets are not needed since it should be always possible to transform the specification into another one with average or worst case end-to-end constraints. It turns out that this is not always true. Consider the 2×2 CMP design in Figure 1.1. One might be tempted to transform that design into another one where each constraint has a bandwidth requirement equal to 192 *Gbps* without specifying any mutual exclusion. This is the case of worst case design. If the clock frequency is 1.6 *Ghz* and the flit-width is 128 the problem is unfeasible because both source and destination interfaces can only handle one constraints ($1.6e9 \times 128 = 192Gbps$). The other transformation is to assign a bandwidth requirement equal to 192/3 *Gbps* to each constraint. This is the case of average design where the solution could have links with an assigned bandwidth that is a fraction of 192 *Gbps* leading to a under utilization of the network capacity.

2.4 Specification of the Library Components

The library of communication components is also described in an XML file. A library file can contain more than one library each identified by a unique name. For instance, a library could contain the performance/cost characterization of on-chip routers and links for the same technology but for different clock frequencies of different flit-widths.

In COSI-NoC on-chip wires and routers are characterized in different ways. For wires, we use an *analytical model* [MBM04] assuming that they will be optimally buffered in the final implementation. For routers, we use Orion [Wan02] to derive an estimation of the area and energy per flit of each router. We run Orion for different configurations of a router in terms of number of inputs and outputs and we collect the results into a matrix. We are interested in two metrics: energy per flit and area. The energy per flit is further decomposed into two components: dynamic energy and static energy dissipation.

Figure 2.5 shows the typical description of a library of communication components that consists of the following elements:

- `Library` that contains the library description. The only attribute is the library name. The same XML file may contain many library descriptions that are distinguished by unique names.

```
<?xml version="1.0" ?>
<Library name="100nm1ch128">

  <Technology fclk="1.5e9"
              vdd  = "1.2"
      ioff = "0.15"
      wmin = "200e-9"
      isc  = "65e-6"
      r0   = "10.0e3"
      cp   = "2.5e-15"
      c0   = "1.5e-15"
      nlayers = "7" />

  <Wire type="copper" layer="6" r="103.9e3" c="154.0e-12" pitch="460e-9" />
  ...
  <Router maxin="10" maxout="10" maxbw="300e6"
          energy = "1.81013e-11 1.21089e-12   2.06545e-11 1.96132e-12 ...
                    3.26714e-11 2.098e-12   4.70244e-11 3.27506e-12 ...
                    ... "
           area="7168    25600 ...
                 32768   51200 ...
                 ..."
  />
  ...
</Library>
```

Figure 2.5: Example of the xml description of the library of communication components for a $100nm$ technology and operating frequency equal to $1.5Ghz$.

- `Technology` that contains the value of all the parameters needed to characterize the impact of the technology node on the performance and cost of the communication links. In this release we only consider copper wires but the library con be easily extended to include other types of interconnect. The attributes that must be specified are:
  - `fclk` that denotes the operating frequency. We characterize the energy dissipation and we derive the power consumption depending on the switching factor (see Section 2.5) and the actual bandwidth carried by a point-to-point link. The clock frequency is used to compute the leakage power that does not depend on the effective bandwidth carried by a link.
  - `vdd` that denotes the supply voltage.
  - `ioff` that denotes the transistor's off current.
  - `wmin` that denotes the width of the minimum sized inverter.
  - `isc` that denotes the transistor's short circuit current.
  - `r0` that denotes the transistor's output resistance.
  - `cp` that denotes the transistor's output capacitance
  - `c0` that denotes the transistor's input capacitance.
  - `nlayers` that denotes the number of metal layers. In this implementation of COSI-NoC we assume that the network links are implemented by global wires (i.e. metal layer 6).
- `Wire` that describe the properties of a link. The attributes that can be set are the `type` of the wire (only copper is supported by now) and all the important physical properties for that type. In the case of copper wires, the following data must be specified: the metal `layer`, the resistance per unit length `r` and the capacitance per unit length `c`.
- `Routers` that describes the energy dissipation and area of a router as a function of the number of inputs and number of outputs. These two quantities are given as two matrices where the value of the entry (i, j) defines that quantity for a router with i inputs and j outputs. The two attributes `maxin` and `maxout` are bounds on the maximum number of inputs and outputs of the model. The real bounds are set as attributes of the parameters of the project. The attribute `maxbw` is the maximum bandwidth that a router can handle. It represents the maximum throughput for each input/output port. *Notice that this library is very different from the one used in the* `cmp2x2` *example. To avoid a router to get congested, we limit the maximum bandwidth per input port to a flit rate that is much smaller than the clock frequency. This is a more realistic case since a the implementation of a router with flit traversal latency of one clock cycle is still hard.*

The same file can contain the specification of many libraries and each library can contain the specification of many types of wires and many types of routers. In this first release, though, we only handle one type of link and one type of router. The extension to many types does not change the approach that we follow for communication synthesis. The same algorithms can be reused while the size of the design space exploration increases.

Remark 2.4.1 (On the router's maximum bandwidth) Figure 2.5 shows a library where the maximum bandwidth supported by a router is equal to the clock frequency. Usually, the number of clock cycles necessary to transfer a flit from the input queues to the output is $N > 1$. In this case the effective maximum bandwidth of a router is less than the clock cycle. In particular, after the arbitration of the input queues has resulted in the input to output assignment, each flit is transferred in N cycles, therefore a packet is transferred at f_{clk}/N flits per second.

2.5 Specification of the Synthesis Parameters

The algorithm that is used by COSI-NoC to synthesis a network is very flexible and the result depends on the various parameters that must be set in the project file.

There are two sets of parameters that can be specified. One set contributes to the definition of the set of admissible implementations and consists of additional resource constraints. This set of parameters includes:

- `sparearea` (default 0.25%) defines the amount of area around the IP core that is considered available to install routers. Given an IP core with area A, a portion of area equal to `sparearea`$\cdot A$ is evenly distributed around the core and considered open space, i.e. available to implement the communication infrastructure (additional area can be reserved by defining `CosiComm` IP core, see Section 2.3).
- `step` (default 0.1) defines the minimum distance (in terms of critical sequential length) from the port of core and the closest interface/router.
- `flitwidth` (default 128) is the flit width. This value directly impacts the capacity of each link as the clock frequency is fixed.
- `switchingfactor` (default 0.5) is the percentage of bits that switch between two consecutive flits.
- `maxindegree` (default 2) is the maximum input degree for routers.
- `maxoutdegree` (default 2) is the maximum output degree for routers.
- `sourceoutdegree` (default 1) is the maximum number of outputs of a source network interface.

- `destindegree` (default 1) is the maximum number of inputs of a destination network interface.
- `allowptp` (default 0) is a switch to allow direct point-to-point connection between a source and a destination. If this switch is equal to 1 then such connections are allowed, if it is equal to 0 that point-to-point connections are not allowed.
- `allowtwohops` (default 0) is a switch to allow the connection between a source and a destination through a single interface. If the value of this switch is 1, each source and destination par is separated by a minimum of three hops.
- `powervsarea` (default 0.5) The cost of a network is the convex sum of power and area:
$$C = \lambda P + (1 - \lambda)A$$
The weighting constant λ is specified by this attribute.
- `hopconstraints` (default 0). If the value of this switch is 1 the algorithm tries to meet the input delay constraints, otherwise such constraints are disregarded.
- `areaconstraint` (default 0). If the value of this switch is 1, the synthesis algorithm tries to implement a network that occupies and area as close as possible to the value specified by the attribute `area`.
- `density` (default unspecified). This parameter specifies the number of points per millimeter square to be considered as installation points for routers.

2.6 Specification of the Required Output

COSI-NoC can generate a set of outputs to analyze the synthesis result. The synthesis result is guaranteed to satisfy all communication constraints and to belong to the set of admissible implementations. Yet, if you are interested in checking whether the implementation satisfies your expectations, COSI-NoC is able to generate a graphical representation of the implemented network (in Svg format [Svg]), a logical representation (in Dot format [Dot]), a SystemC simulation [Sys], an input file for the Otter network visualization tool [Ott] and a textual report.

In order to request the generation of a specific output, you can include an `Output` element in the XML project as follows:

```
<Output>
    <Svg name="cmp2x2.svg" />
    <SystemC name="cmp2x2.cpp" mk="cmp2x2.mk" />
    <Report name="cmp2x2.rep" />
```

```
    <TabAppend name="cmp.txt" />
    <Dot name="cmp2x2.dot" />
    <Otter name="cmp2x2.odf" />
</Output>
```

The `Output` element can contain one element for each required output. The recognized elements are:

- `Svg` to request the generation of an SVG file that depicts the network. The only attributes that can be specified is the name of the SVG file. The file name is relative to the `outputdir` parameter of the project (see Section 2.2).
- `Dot` to request the generation of a DOT file that represents the logical structure of the network. The only attribute that can be specified is the name of the DOT file. The file name is relative to the `outputdir` parameter of the project (see Section 2.2).
- `Otter` to request the generation of an OSDfile that can be read from the OTTER visualization tool. The only attribute that can be specified is the name of the osd file. The file name is relative to the `outputdir` parameter of the project (see Section 2.2).
- `SystemC` to request the generation of a SYSTEMC netlist of the implemented network. Two parameters can be specified: `name` is the netlist file name and `mk` is the name of the `makefile` to compile the SYSTEMC netlist and obtain an executable simulation. The file name and the `makefile` are relative to the `outputdir` parameter of the project (see Section 2.2). Make sure that the `COSIROOT` environment variable and the `SYSTEMC` element in the COSI-NOC configuration file are correctly specified.
- `Report` to request a report of the synthesis result in textual format. The only attribute that can be specified is the name of the text file. The file name is relative to the `outputdir` parameter of the project (see Section 2.2).
- `TabAppend` to request COSI-NOC to append the synthesis results as a single row in a tab separated file. This feature is very useful when COSI-NOC is used to synthesis many SoCs and the results must be then visualized using Excel.

If an element is not specified, the corresponding output is not generated by COSI-NOC. Please refer to Chapter 3 for an explanation of how to use the generated output.

Chapter 3

The Output Format

This chapter describes the various outputs tha can be generated by COSI-NoC and how these outputs can be used to analyze the network.

3.1 The Svg Output

The Svg output is the pictorial description of the on-chip network. An Svg file is an XML description that can be interpreted and rendered to create a picture. The XML elements that we use are simple circles and rectangles. Sources and destinations are represented by filled circles. Sources are colored in red while destination are colored in black. We assume that ports are located at the bottom left corner of each IP even if this information can be explicitly given to the floor planner and retrieved by parsing the floor-planning result. Note that the $(0, 0)$ corner of the Svg representation is the top-left corner.

Routers are represented by green squares and latches by orange squares. The distinction between routers and latches is the following: if, for all inputs, all and only the flows entering the router go to the same output then the router does not have to arbitrate among input queues or decide which output to send the input to. In this case, this piece of logic is reduced to be just a latch. Note that it is not possible to get rid of the latch because it was introduced by the synthesizer in order to break the wire that is longer than the critical sequential length.

Each link is represented by an arrow. Links are directional.

Sizes are actual physical sizes. When the Svg file is generated, the visible area is defined in μm and all dimensions are scaled appropriately in order to have a real idea of the area that is occupied by each block and the one taken by the wires.

Svg files can be opened with standard Svgenhanced browser or with tools like Inkscape [Ink]. If Inkscape is adopted as a viewer then two extra features are gained. The Svg file contains the definition of many layers. One layer is dedicated to the IP cores and it is the only one that is visible. The other layers

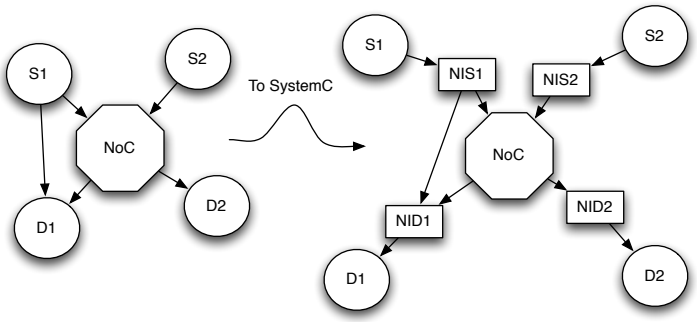

Figure 3.1: Transformation from the implementation graph to the SystemCnetlist.

contain the source-to-destination paths. There is one layer for each path and they can be visualized and superimposed using the layers menu of Inkscape. Moreover the rendered image can be saved in any other format.

3.2 The Dot Output

The Dot language was originally developed at AT & T Research Labs. It allows to describe a graph in a textual format independently from the final visualization. A set of rendering engine is available to read the graph description and automatically present a laid out graph to the user.

The Dot representation is very useful to understand the logical structure of the synthesis result. Each IP core is represented by a rectangle with an associated name (the name of the IP core). Each router is a record. The information stored in the record are the entries of the routing table. The first column of the record is the name of the router. By convention, the name of a router is obtained by the prefix R followed by a unique integer identifier. The following columns are the entry of the routing table. Each entry has two rows. The top row identifies a flow by its source and its destination. The bottom row denotes the next router to which the current router sends packets belonging to that flow.

3.3 The SystemC Output

COSI-NoC can generate a SystemC netlist of the implemented network. In order to generate an executable simulation, the netlist must be compiled together with the SystemC library of components provided in the distribution. For convenience, a `makefile` that compiles the netlist can be also generated.

When we generate the netlist, we transform the result of the synthesis as in Figure 3.1.

For each source and destination, we introduce a network interface. We connect each source and destination core to its network interface and then we connect the network interfaces to the rest of the network. Point-to-point links from source to destinations become point-to-point links between network interfaces. In this way, each core has at most one input and one output port. Notice that, if the `allowptp` parameter is set to zero, no point-to-point links are synthesized.

The SystemC library that is provided in the distribution implements a wormhole switching technique and a weighted fair queuing technique. COSI-NoC generates the routing tables and the weights for each router. The weight vector for a router R is a vector w_R that has an element for each input queue. The total bandwidth bw_R is distributed among the input queues such that $bw_i/bw_R = w_R[i]/\sum_i w_R[i]$.

The generation of a `makefile` requires the correct definitions of the `SYSTEMC` element in the COSI-NoC configuration file and of the `COSIROOT` environment variable. This variables are needed to point at the correct include and library directories. In this distribution, we target the Linux and Mac Os X platforms only.

Bibliography

[Alg00] *Introduction to Algorithms*. MIT Press, 2000.

[AM03] Saurabh N. Adya and Igor L. Markov. Fixed-outline floorplanning : Enabling hierarchical design. *IEEE Trans. on VLSI*, 11(6):1120–1135, December 2003.

[Dot] *Graphviz: Graph Visualization Software*. http://www.graphviz.org/.

[HGR05] Andreas Hansson, Kees Goossens, and Andrei Rǎdulescu. A unified approach to constrained mapping and routing on network-on-chip architectures. In *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 75–80, 2005.

[Ink] *Inkscape*. http://www.inkscape.org/.

[MBM04] Man Lung Mui, K. Banerjee, and A. Mehrotra. A global interconnect optimization scheme for nanometer scale vlsi with implications for latency, bandwidth, and power dissipation. *IEEE Transactions on Electron Devices*, 2004.

[MMA+06] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. De Micheli, and L. Raffo. Design of Application-Specific Networks on Chips with Floorplan Information. In *International Conference on Computer-Aided Design (ICCAD)*, 2006. to appear.

[Ott] *Otter: Tool for Topology Display*. http://www.caida.org/tools/visualization/otter/.

[Svg] *Scalable Vector Graphics (SVG)*. http://www.w3.org/Graphics/SVG/.

[Sys] SystemC *2.1 Language Reference Manual*. http://www.systemc.org/web/sitedocs/lrm_2_1.html.

[tin] *TinyXml*. http://www.grinninglizard.com/tinyxml/.

[Wan02] H. Wang. Orion: A power-performance simulator for interconnection networks, 2002.

[XML] *Extensible Markup Language (XML).* http://www.w3.org/XML/.