Facultad de Informática
Universidad Complutense de Madrid

Proyecto de Sistemas Informáticos
Curso 2006/07

Diseño de una herramienta de generación de arquitecturas de interconexionado para una plataforma de emulación multiprocesador sobre FPGA

Ignacio Castiñeiras Pérez
Laura Guitiérrez Muñoz
Sergio Martín Moreno

Dirigido por:
David Atienza Alonso
Departamento de Arquitectura de Computadores y Automática

Los autores de este proyecto autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos no comerciales tanto la memoria, como el código, la documentación y el prototipo de la plataforma desarrollada.

Ignacio Castiñeiras Pérez Laura Gutiérrez Muñoz Sergio Martín Moreno

INDEX

KEYWORDS

NoC (Network on Chip)

Xpipes Compiler

Sunfloor

SoC (System on Chip)

Network

Switch

Core

Route

Link

AMBA

ENGLISH SUMMARY

In the last years, the growing of the Systems on Chip (SoC) has shown a problem with the interconnection with buses. It becomes a bottleneck due to the poor scalability of the buses. Because of that a new way of interconnection, Network-on-chip (NoC), has been developed. NoCs uses the idea of the computer network to implement a similar solution to make the interconnections in SoCs.

Unfortunately, NoC development is a hard task. In the last times have been developed some tools, such as SunFloor or XpipesCompiler that brings a great help to the NoC designer. But these applications only generate and compile topologies, and it was such a hard task. With NoCdificador we try to develop a tool that helps the NoC designer to modify his NoCs in a completely safe way.

But NoCdificador is not only a NoC modifier. It also controls the correctness of the network, can generate new networks from the beginning, can create or edit SunFloor input files and then invoke SunFloor. With NoCdificador, a topology can be simulated with XpipesCompiler and simulated with MPARM. And one of the most attractive features of this tool is the different statistics shown for the same network: simulated with AMBA (bus interconnection) and Xpipes (NoC interconnection), and even a graphic comparative between both simulations.

Our application has been developed in Java. One of the requirements of the program was that it had to be multiplatform, so we chose the multiplatform language we knew the best.

RESUMEN EN CASTELLANO

Nuestro proyectro consiste en la creación de una herramienta de generación de arquitecturas de interconexionado para una plataforma de emulación multiprocesador sobre FPGA.

1. Marco de trabajo

Comenzaremos orientando al lector sobre el marco en que se desarrolla nuestro trabajo y que objetivos persigue.

Actualmente los sistemas en chip (SoC) son cada vez más complejos, por la gran cantidad de componentes prediseñados que lo conforman.

Necesitamos que los diferentes componentes del SoC puedan enviarse información. La alternativa, llamémosla más "clásica", consistía en que los procesadores y los dispositivos de memoria se comunicaran mediante buses. Pero, a medida que el tamaño de los SoC crece, esta vía sufre problemas de escalabilidad de los buses, además de una longitud cada vez mayor de las conexiones.

Cobra pues sentido buscar una alternativa de futuro para la comunicación de los SoC.

Las redes en chip, NoC, representan una variante fiable de interconexión, extrapolando algunos de los conceptos de las redes de computadores a la interconexión de múltiples IP-Cores difundidos sobre un sustrato común.

Actualmente existen varias líneas de investigación trabajando sobre el diseño de NoC, tarea para nada sencilla.

Es en este punto donde cobra sentido nuestro proyecto, como soporte gráfico que dé un aspecto visual más amigable al arduo trabajo de elegir que diseño de red confeccionar, como estructurar esta red, y como observar de un plumazo algunos de los resultados obtenidos al simular de esta red.

2. Especificación "a vista de pájaro" de la funcionalidad que ofrece nuestra herramienta

- Construir una aplicación que sea capaz de implementar la siguiente funcionalidad:

I. Definir una red con todos sus componentes, partiendo de cero:

(Elementos interesados en comunicarse)
Procesadores
Dispositivos de memoria

(Construcción de vías de comunicación)

Elección de una topología de las estructuras de interconexión

Switches de interconexión

Links que definan los posibles caminos que puedan llevar los datos

(Comunicaciones establecidas)

Rutas que comuniquen proporcionando un sencillo entorno gráfico para la construcción de la red, así como la posibilidad de visualizarla en todo momento de su construcción.

II. Abrir una red ya existente y modificarla.

Mantener en todo momento un "conocimiento de la validez estructural de nuestra red", permitiendo al investigador **modificar a su antojo** los diferentes elementos de esta con la tranquilidad de poder ser informado cuando lo desee de situaciones que hagan la red inestable.

III. Interaccionar con el Programa Sunfloor, que entre su diversa funcionalidad se encuentra el, conociendo aspectos técnicos como:

a. Las aplicaciones que queremos tenga nuestra red.
b. La tecnología de las interconexiones de nuestra red.
c. Las características de los IP.

Confeccionar una red y un floorplan, o diseño interno del chip que contendrá a nuestra red.

3a) Dadas unas especificaciones a.b.c. Sunfloor generará su salida, y nuestra aplicación será capaz de utilizar la red proporcionada como salida por SunFloor y abrirla, para poder trabajar con ella, así como mostrar el FloorPlan.

3b) Proporcionar al investigador la posibilidad de definir él mismo los aspectos técnicos de una red a construir dentro de nuestro propio programa, a través de un interfaz gráfico sencillo y amigable. Después podrá llamar desde nuestro programa a SunFloor y trabajar con la red obtenida.

IV. Interaccionar con el programa XpipesCompiler, que entre su diversa funcionalidad es capaz de, tomando una red válida de entrada, compilarla a código SystemC. Ese mismo código SystemC, mediante MPARM es capaz de simular la red y ofrecer unos resultados sobre su rendimiento.

Nuestra aplicación será capaz de comprobar si la red que le enviamos a XpipesCompiler es válida, así como mostrar en un entorno gráfico los resultados de rendimiento obtenidos.

3. Especificación "con todo detalle" de la funcionalidad de nuestra aplicación

Nuestro sistema permitirá al usuario trabajar únicamente con una red en cada momento, si el usuario quiere abrir otra red será preciso que cierre previamente la red con la que estaba trabajando.

Lo que sí se permitirá será lanzar varias instancias de nuestra aplicación, para poder tratar con varias redes a la vez, cada una dentro de una ventana.

Nuestra aplicación permite al usuario elegir entre las distintas funciones que puede aplicar, navegando entre las distintas entradas de la barra de menú, o pulsando sobre los iconos de la barra de herramientas.

Los menús con los que puede interaccionar el usuario son:

a) File → Nueva red, abrir una ya existente, cerrar la red en uso, salvar, salvar como, salir de la aplicación

b) Edit → Deshacer, rehacer

c) View → Log, componentes incompletos de la red, parámetros de la red resultado de aplicar SunFloor.

d) Components → Crear, modificar y eliminar un componente

e) Simulate → Simular una ruta, simular la red, compilar la red, ver el código HDL resultante de nuestra red

f) Network → Tipo de red, cambiar su nombre

g) SunFloor → Cargar archivos para paso de parámetros, mostrar el floorplan, indicar las preferencias de visualización de las tablas

h) Graph → Mostrar y ocultar el grafo de nuestra red

i) Stats → Ver las estadísticas de rendimiento de nuestra red, según AMBA, según Xpipes, y mostrar una comparativa entre ellas

j) Help → Manual de usuario, "Acerca de" de la aplicación

Los iconos de la barra de herramientas son:

a) Open
b) Save
c) New File
d) Help
e) Undo
f) Redo
g) Simulate network
h) Simulate route

i) New Component
j) Modify Component
k) Delete Component
l) View Log
m) Show Graph
n) Close Graph

Como interfaz gráfica, se compone de dos paneles fundamentales, uno a la izquierda que permite visualizar un árbol con los componentes de la red, y ahondar un poco más en los parámetros que conforman la composición de nuestra red, si ésta es fruto del trabajo de SunFloor.
A la derecha un panel de mayor tamaño, donde visualizar la información más inmediata que quiera consultar el usuario sobre la red.

3.1. Definir una red "partiendo de cero"

Nuestro sistema esta preparado para soportar tres tipos de topologías de red, que son *Mesh* o *Torus*, dentro de las topologías regulares, u *Others*, como calificativo que engloba a las topologías irregulares.

Si se escoge una red de topología regular el sistema crea automáticamente los switches que servirán de interconexión, así como los links que comuniquen a estos.

Si se escoge una topología regular el sistema partirá de una red vacía, sobre la que se podrán comenzar a colocar los Procesadores y Dispositivos de memoria, así como los propios elementos de interconexión.

Es en esta construcción de la red donde se puede ver la importancia de esta herramienta NoCDificador, ya que su no existencia supondría necesitar un conocimiento profundo de los archivos que definen a las redes, de los atributos de cada elemento, de su orden, de las restricciones de interconexión.
Esto convertiría el diseño de las redes, en un trabajo oscuro, donde cada pequeño paso de diseño, cada pequeña alternativa de decisión a tomar se traduciría en una tarea muy compleja y oscura.
Con esta aplicación, el usuario irá construyendo su red a través de cuadros de dialogo sencillos y amigables, con un sistema que, por un lado, le permita definir y conectar los componentes a su antojo, y por otro le guarde bien las espaldas, garantizándole la validez estructural de su trabajo.

Nuestro sistema, en su aspiración de ir dirigido al trabajo de investigadores, distinguirá entre errores críticos y subsanables, no permitiendo los primeros por no tener sentido y advirtiendo de los segundos, para alertar al trabajador de los "efectos colaterales" que pueden estar suponiendo sus decisiones sobre un cierto componente determinado.

Este representa otro objetivo fundamental de nuestro trabajo, el garantizar una "constante visión de la consistencia de nuestra red". Así, cualquier pequeño cambio que el usuario haga sobre un determinado componente, **será controlado**, en el sentido de modificar todos los componentes que se vean afectados por este cambio.

Veamos por ejemplo, que si nuestro usuario elimina un switch, automáticamente nuestra aplicación modifica todos los componentes core que se encontrarán conectados a ese

switch, todos los link que conectarán ese switch y todas las rutas que en su encaminamiento se encontrarán con ese switch.

De nuevo otra ventaja para el diseñador con respecto a enfrentarse directamente con un archivo de texto que defina la red.

Otra funcionalidad que ofrece la aplicación es la posibilidad de deshacer y rehacer los últimos cambios efectuados sobre la red que se está tratando.
Uno puede por ejemplo, imaginar, desde el punto de vista de diseño, que supone el plantearse una alternativa, por ejemplo, cambiar un determinado core, conectándolo a otro switch diferente del que se encuentra en este momento. Con solo un par de clicks, nuestro usuario puede efectuar el cambio, puede visualizar su nueva red, puede ser informado de "los famosos efectos colaterales" que este cambio produciría, y puede *deshacer* la modificación del Core (y todos los cambios colaterales que ello supone).
Fácil, amigable, y rápido.

Pero no perdamos de vista lo que supone una red en chip, desde el punto de vista casi de definición como sistema que comunique los procesadores con los dispositivos de memoria y viceversa.

Nuestra aplicación ofrece un modo sencillo de visualizar todas las rutas de comunicación definidas en nuestra red así como todos y cada uno de los puntos por los que pasan.

Y si una imagen vale más que mil palabras, nuestra aplicación también permite visualizar en todo momento de un modo gráfico el aspecto de nuestra red, permitiendo observar uno a uno los cambios realizados, y haciendo que el usuario pueda acceder rápidamente a los componentes con un par de clicks en el gráfico sobre aquel que le interese modificar.

Ya en otro orden de cosas, nuestra aplicación interacciona con el programa XpipesCompiler, pero eso ya lo explicaremos más tarde. Pero incidir una vez más en que el usuario puede crear una red de cero, y llamar con esa red a XpipesCompiler y posteriormente a MPARM para poder optar a toda la funcionalidad de este programa, además de poder tomar parte de la salida de éste dentro de nuestra herramienta con el fin de observar algunos aspectos de rendimiento de la red que se ha definido. Por último, tan sólo indicar que MPARM ofrece este resultado de rendimiento en archivos de texto, por lo que necesitamos que nuestro sistema sea capaz de parsear estos archivos para mostrar resultados "gráficos" de rendimiento.

3.2. Abrir una red ya existente y modificarla

Ni que decir tiene que nuestro sistema no se ciñe exclusivamente a la creación de nuevas redes partiendo de cero, sino que pretende ser capaz de leer un archivo de definición de una red y construir esta misma con todos sus componentes dentro de nuestra aplicación.

Así, puede tomar archivos ya existentes, y construir la red "que se esconde" tras el texto. Del mismo modo, toda vez terminado el trabajo con nuestra aplicación podemos guardar nuestra red "gráfica" como archivo de texto.

Esto permite tomar un archivo de texto, traducir a nuestras redes que vemos y modificamos en nuestra aplicación, ver todos los aspectos de la red que enunciamos en

3.1., dejarla como nosotros queramos, y volver a trasformarla a archivo de texto, sustituyendo por completo la labor de modificación oscura y compleja de modificar las redes trabajando directamente sobre el archivo de texto.

Nuestro sistema está preparado para informar cuidadosamente al usuario de todo lo que ha ocurrido durante la lectura del archivo de texto mediante un *log* que monitoriza el proceso de lectura.

Además necesitamos ser capaces de leer estos archivos de texto para comunicarnos con el programa SunFloor, cuya salida nos da una red en formato de estos archivos.

3.3. Interacción con SunFloor

¿Puede nuestro trabajo tener el humilde objetivo de colaborar al menos un poquito en la labor de diseño de NoC ? Creemos que sí, en la medida de que sea capaz de comunicarse con SunFloor y Xpipes.

El objetivo final del diseño de NoC es obtener un chip que contenga la red "a medida" diseñada. Bien, de esto, en parte, se encarga SunFloor. SunFloor es un programa que, conociendo para qué tipo de aplicaciones será utilizada nuestra red, sobre qué tecnología será construida, ciertas características de su IPs, así como la frecuencia de reloj a la que funcionará el chip, es capaz de generar la red, y su distribución dentro del chip.

Bien, ¿pues por qué no colaborar nosotros en el diseño de las tablas que especificarán estos parámetros de entrada? Permitiremos que el usuario construya estas tablas dentro de nuestra aplicación partiendo de cero, o que elija unas ya construidas y las modifique a su gusto. Nuestra aplicación se encargará de llamar a SunFloor una vez que el usuario así lo indique.

¿Y por qué no utilizar también la salida de SunFloor? Nuestra aplicación recoge la red generada por SunFloor, permitiendo trabajar con ella como ya explicamos en 3.2. Además mostramos el dibujo del FloorPlan, para que el usuario pueda conocer su diseño.

3.4. Interacción con XpipesCompiler

Bien, ¿y qué mejor manera de conocer las caracerísticas de nuestro chip que simulándolo? Con ella podemos comprobar los resultados de nuestra red, completando un trabajo que partió de un sencillo archivo de texto que definía nuestra red, o de la nada en caso de construirla desde cero. Pero para poder simular, necesitamos código que pasarle al simulador.

Para ello, nuestra aplicación llamará a XpipesCompiler para obtener el código SystemC de nuestra red, y será capaz de mostrarlo. Así mismo llamará a MPARM para simular nuestra red con el código generado por XpipesCompiler, que no es otra cosa que comprobar su funcionamiento y rendimiento sobre una aplicación real. MPARM devolverá un archivo con el rendimiento de la red, y nuestra aplicación será capaz de traducirlo para mostrar aspectos de rendimiento de un modo "gráfico".

Mostraremos el resultado tanto si simulamos nuestra red sobre la conexión "clásica" de buses, como si lo hacemos sobre la nueva alternativa de las NoCs, y ofreceremos una comparativa de ambos rendimientos para observar con cuál funciona mejor.

4. Estructura de las clases

Las clases de nuestra aplicación pueden dividirse en tres grandes grupos según su función.

Por un lado tenemos las clases de tipo parse. Se encargan de leer los archivos de entrada, que tienen un formato concreto, y los transforman en datos legibles por nuestra aplicación.

Los principales parsers que tenemos son los de lectura/escritura de topologías, generado completamente a mano para adaptarse perfectamente a la estructura de dichos archivos; los de lectura/escritura de XML, generados usando la biblioteca Jdom; y por último el parse de lectura del fichero de estadísticas generado tras la simulación, mediante el que se obtienen los datos que se transforman posteriormente en tablas visibles para el usuario.

El segundo gran grupo de clases son las clases gráficas. Éstas a su vez podrían dividirse en dos subgrupos: las clases de visualización y las clases de modificación. Mientras las primeras son las que permiten ver el estado de la red y los componentes que la forman en todo momento, todo esto de dos formas distintas: mediante un grafo y un árbol; las segundas son las que muestran al usuario toda la información de las características de los componentes y rutas, y le permiten modificarla a su antojo.

El último gran grupo de clases lo forman las clases relativas a la lógica de la aplicación. Son las que mantienen los componentes, mantienen y modifican en estado de la red, comprueban la integridad de todas las operaciones que se realicen en la red. Son, en definitiva, las clases que se encargan de realizar las operaciones pedidas por el usuario.

5. Cómo está programado

Nuestra aplicación se encuentra programada íntegramente en Java. Elegimos este lenguaje de programación porque uno de los requisitos es que fuese una aplicación multiplataforma, y entre las diferentes opciones que se nos ofrecieron fue la que más nos gustó, por ser la que mejor conocíamos.

Salvo para la muestra del grafo, que utilizamos SWT, nuestra interfaz gráfica está programada enteramente con Swing, ya que nos resultaba visualmente agradable, además de ser ya conocido por nosotros y cubrir, en principio, todas las necesidades que íbamos a tener.

Un factor que creemos destacable es que nuestra aplicación, al igual que esta memoria, está desarrollada íntegramente en inglés, tanto la interfaz gráfica que verá el usuario, como el código y los comentarios que vería un programador. Esto es así porque nuestra aplicación tiene proyección internacional, y se espera que sea utilizada en entornos de investigación en diferentes países, así como que sus funcionalidades puedan ser ampliadas por aquella persona que lo necesite, ya sea en España o en cualquier otro lugar.

En cuanto a la organización del desarrollo, hemos utilizado Eclipse, ya que proporciona comodidades y ayudas al programador, pero sobre todo porque tiene un buen sistema de CVS, además de sencillo de usar. Esta última característica nos pareció importante, ya que la coordinación del desarrollo la realizamos a través de un CVS, para así poder tener en todo momento acceso a las últimas modificaciones, y que también el profesor pudiera comprobar el desarrollo de la práctica cuando deseara.

1. INTRODUCTION

As steady progress is being made in the miniaturization of chip features, embedded systems are quickly evolving towards complex devices, including a large set of computation engines, dedicated accelerators, input/output controllers and multiple memory buffers. MultiProcessor System-on-Chip (MPSoC) is a commonly used term to describe the resulting outcome. However, this feature- and performance-oriented evolution is not devoid of significant challenges, including mastering the increasing design complexity and minimizing power consumption.

Moreover, miniaturization itself is bringing its own set of design issues at the physical level, originated primarily by an increasing ratio of wire vs. logic propagation delay.

One of the most critical areas of MPSoC design is the interconnect subsystem, due to architectural and physical scalability concerns. The former is due to the performance pressure associated with several system cores that simultaneously demand communication resources, and subsequently relates to the need for providing adequate bandwidth and latency. The latter is due to the intrinsic issues due to the design of wires that must span across the whole chip area, namely, propagation delay, noise and crosstalk. Traditional shared bus interconnects are relatively easy to design, but do not scale well.

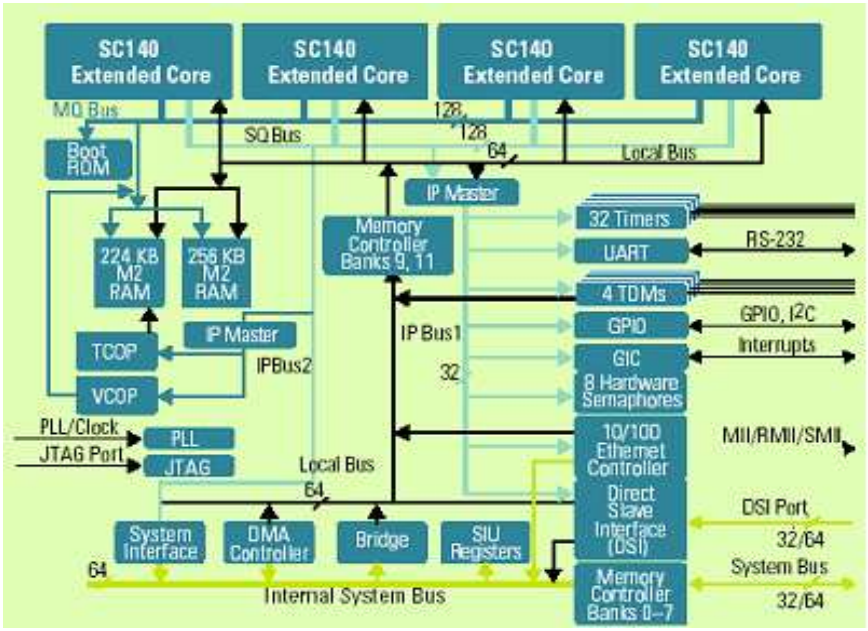


Fig 1.1 Motorola's SoC platform. Communication architecture becomes The major bottoneck.

Thus, evolutions have been conceived both from the protocol (e.g. outstanding transactions with out-of-order delivery) and the topology (e.g. bridges, crossbars) points of view. Nevertheless, scalability is still suboptimal, as protocol improvements still hit a bandwidth limit due to the available physical resources, and topological extensions require

the use of bridges (i.e. multiple buses or ``spaghetti-like'' design (Fig 1.1)) or large area overheads in routing structures (i.e. using crossbars).

Networks-on-Chip (NoCs) have been suggested as a promising solution
to the scalability problem. By bringing packet-based communication paradigms to the on-chip domain, NoCs address many of the issues of interconnect fabric design. Wire lengths can be controlled by matching network topology with physical constraints and bandwidth can be boosted by increasing the number of links and switches.

Furthermore, compared to irregular, bridge-based assemblies of clusters of processing elements, NoCs also help in tackling design complexity issues.

While these key advantages of NoCs have been largely accepted nowadays, the practical implementation of NoCs in latest technology nodes is a very open challenge. The crucial issue is again related to wiring and design automation. Even if capacitive loads and propagation delays can be controlled much better than in shared buses, issues such as manual wiring, definition of corrects buffers and switches to limit link power consumption, and the need for placement-aware logic synthesis still have to be explored to assess the feasibility of NoCs in forthcoming technology nodes.

In this introduction we present the design flow for NoCs, and the necessity of developing an application like NoCdificador. The remainder of the text is organized as follows. In the next paragraphs we overview previous work in the field on on-chip interconnects in general and, more particularly, focus on NoC synthesis and topology design and layout. Then, we introduce the NoC design flow, and finally we explain how our application works and how can it improve NoC's design.

1.1 Related work

The problem of high-performance or low-power synthesis of on-chip interconnects based on the bus paradigm has been studied extensively in the literature. The use of point-to-point links and bus design using floorplan feedback has been also explored. Recent research has focused on efficient synthesis methods for NoC-based interconnects and comparisons with bus-based SoCs. Relevant research in application-specific custom topology design has been proposed in earlier works on NoC design for well-behaved or regular traffics.

Floorplanning estimations and analytical models have been employed during the topology design process to obtain area and wirelength estimates, but these works are limited to libraries of standard topologies. A physical planner has been used to focus on minimizing power consumption on the links of NoC topologies. However, this method does not consider the area and power consumption of switches in the design. In, a flow that addresses the problem of full custom NoC topology design with early floorplan estimation is proposed. However, even though these works have considered the various problems of NoCs synthesis at the physical level, none of them has studied and covered extensively the possible consequences of different process technology nodes on complete NoC-based interconnects, as we present in this paper.

In addition, methods to build area and power models for various NoC components have been developed to enable system-level exploration of SoCs using NoC interconnects.

However, the existing area-power models of the NoC components, such as switches or network interfaces, were not targeting the 65 nm manufacturing technology and may need to be revised with the latest back-end tools to properly capture model requirements, as we illustrate in our results.

At a higher level of abstraction, different methods have been proposed to analyze traffic information and obtain models that can be utilized as inputs to bus and NoC design methodologies. These approaches are complementary to this work. In addition, the problem of supporting multiple applications has been studied. Also, methodologies that unify resource reservation, mapping and routing in NoC designs have presented. However, these works do not fully explore the topology design process.

Important research contributions have been presented on automatic code generation of NoC topologies for simulation and synthesis. These works complement the presented one, as their inputs are typically NoC designed topologies and even enable the use of post-synthesis timing libraries in their simulation models.

A very interesting study on the impact of technology scaling on the energy efficiency of standard topologies (such as meshes, tori and their variants) has been presented in techscaling-standard. Our work differs from this research in two ways: first, we consider the design of platform-specific NoC topologies and architectures. Second, we use a complete design flow that is integrated with standard industrial toolchains to perform accurate physical implementations of the NoCs.

1.2 Design flow

NoC designing is a hard, long task. The usual flow begins with the request of the different features expected for the NoC, such as power, area, core specifications, and application traffic. When these features have been fixed, the designer have to decide which topology is going to be used, the mapping, routes to use… all these decisions are really hard and depend on subjectivity of designer. Most of the times, the topology or routes chosen are not the best. To solve this problem, an automatic tool is used to design a more optimal topology than a hand-designed topology. This application is SunFloor.

With the topology generated by SunFloor, we can use another tool, xpipesCompiler, to compile this topology. After this step, RTL code will be generated for that NoC.

After compiling, the RTL code corresponding with our topology can be simulated with different applications, in order to get statistical data about the performance of the NoC designed by SunFloor. This simulations allows the designer improve the NoC, searching only the elements that brings problems to the NoC.

If all data are correct, NoC is ready for synthesis and emulation on a FPGA. Finally, if there are no problems after the emulation in FPGA, the NoC can be made on a chip.

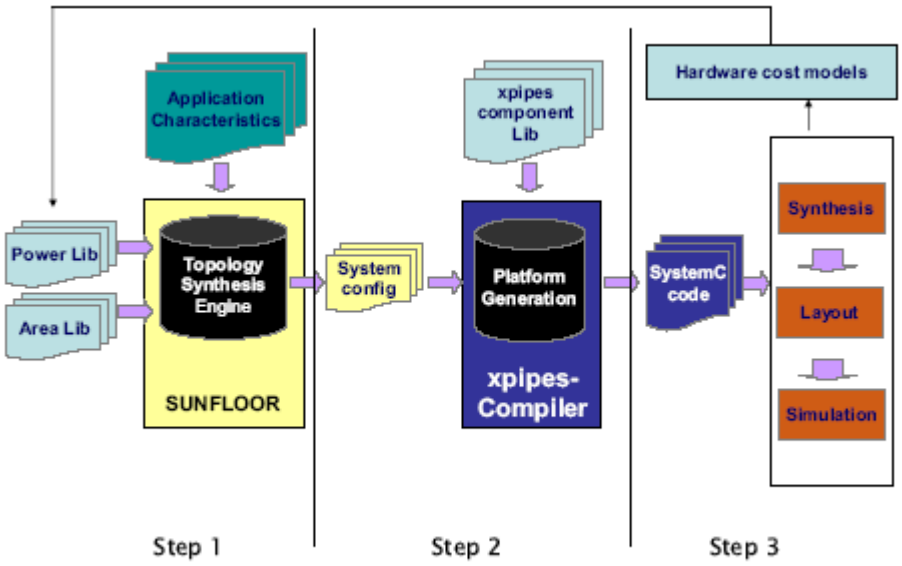


Fig 1.2 Usual design flow

1.3 ***NoCdificador***

NoCdificador is the tool we have developed in this project. It is an application that allows the designer create, modify, simulate… a NoC, assuring the correctness of the network.

Our application can interact with any phase of the design flow, except synthesis and emulation, so it has 3 different inputs:

a) Topology files:

The first input can be a topology file. Our application validates the topology, and then allows in a very easy, intuitive way to modify any feature or component in the NoC. In any moment, the NoC can be complied and simulated.

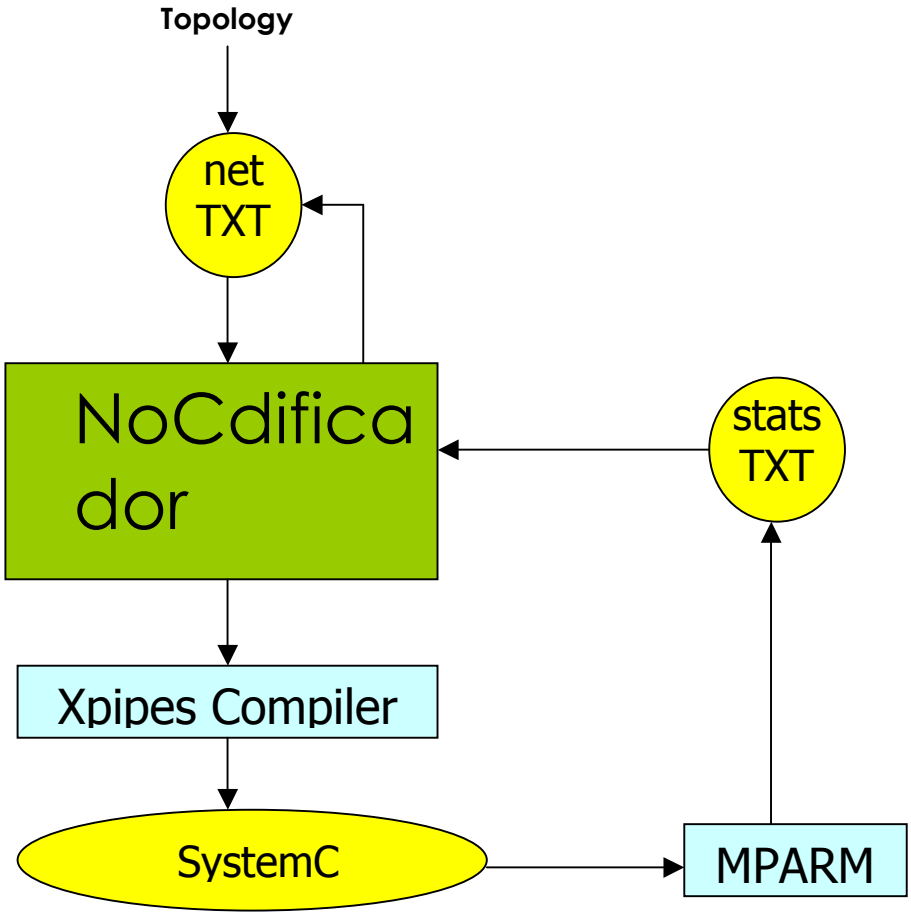


Fig. 1.3

b) Files with the characteristics wanted for the NoC:

In this case, NoCdificador can generate a xml file containing the data requested, or can open xml previously generated, and then generate the topology using SunFloor. This topology is taken as the input file in case a) (See Fig 1.3)

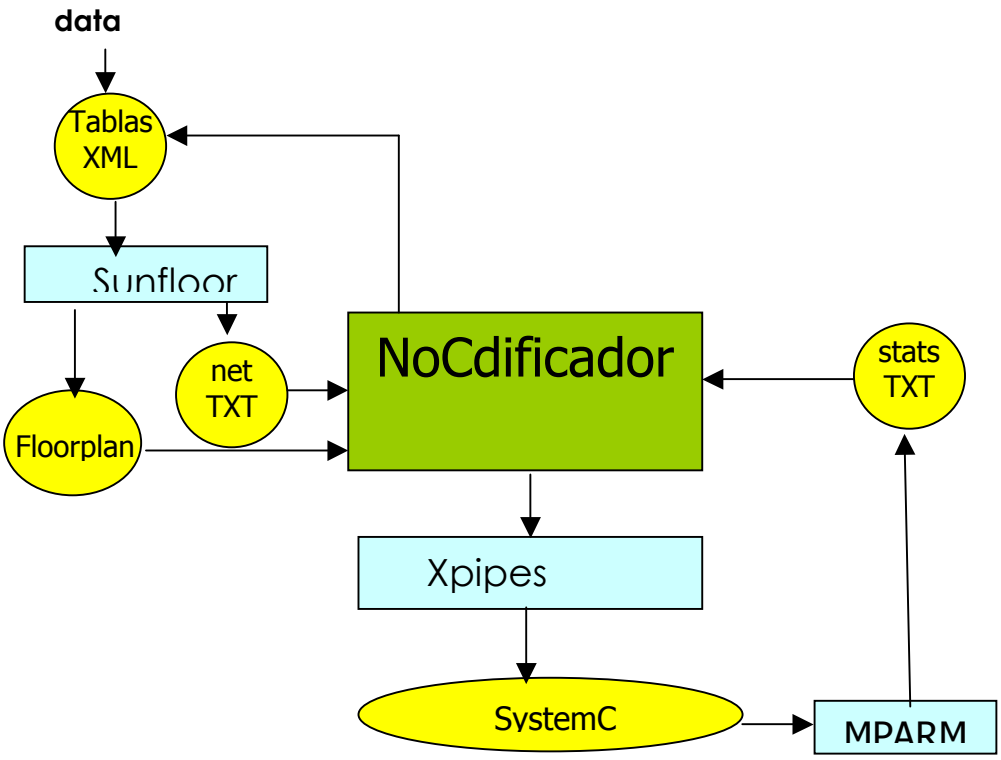


Fig. 1.4

c) New topology:

Finally, user can create a new topology, beginning from the very first step: give a name to an empty network.

Detailed features are detailed in chapter 2 (User manual). Anyway, some of them are going to be explained in this introduction.

Topology files text files, but they are not easy to modify. Any modification on any element of the NoC can have collateral effects on the rest of the elements in that NoC. A manual control of all this changes is a very long task, and designer can easily make a mistake. NoCdificador can make all this modifications, controlling collateral effects and assuring the correctness of the NoC, informing about incomplete routes or elements, and can make all this on an easy way thanks to his GUI.

A designer using NoCdificador can also have control of the elements on the NoC at any moment. NoCdificador have different ways to show all the elements of the NoC: a tree or a graph.

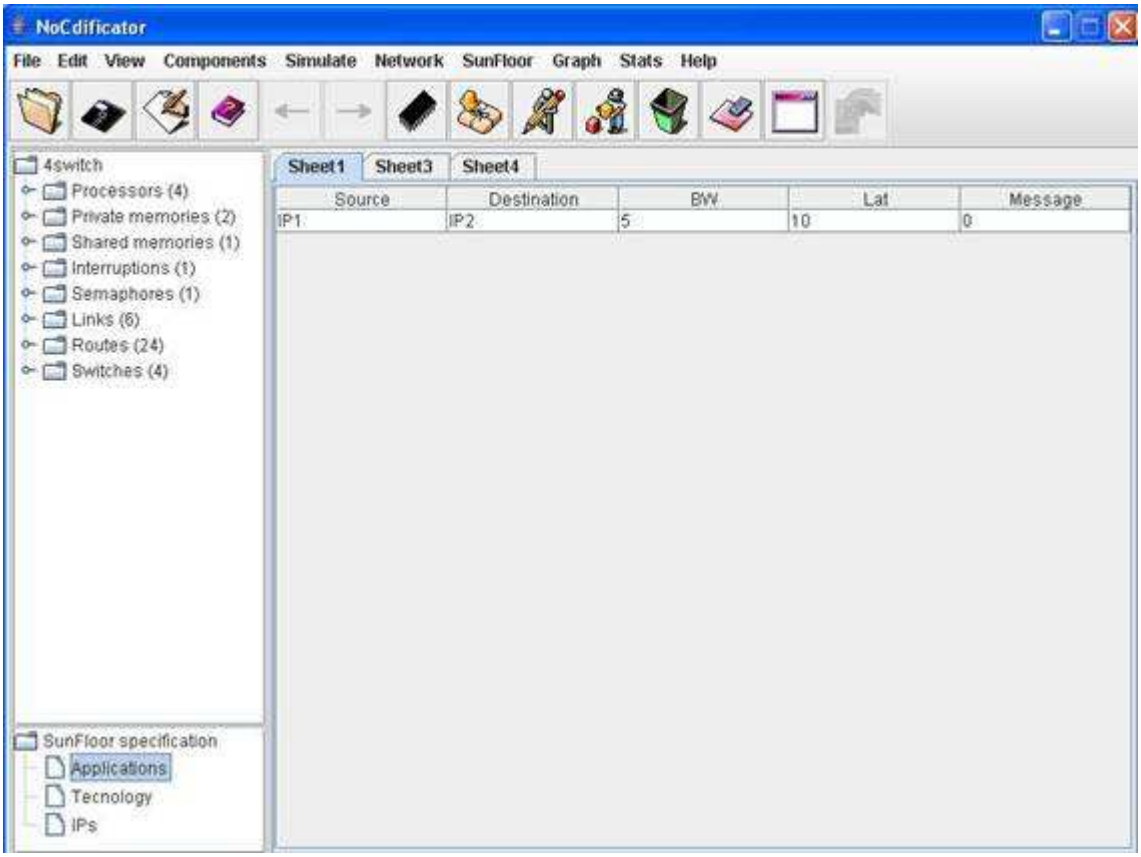
The user can also use XpipesCompiler and MPARM from NoCdificador to compile and simulate the network. After simulation, NoCdificador shows with a clear format the results of the simulation, even comparing different results. This visualization method is much more easy to read than txt tables generated by MPARM.

In conclusion, we think that our application is necessary due to the difficult method of designing NoC that is being used nowadays. NoCdificador can help designers to finish their NoC faster an easier.

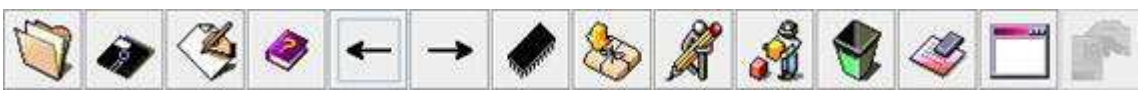
2. USER MANUAL

In this section we want to give a guide for those who want to learn how to use NoCdificador. We will begin on the main window, and then will explain the use of each menu.

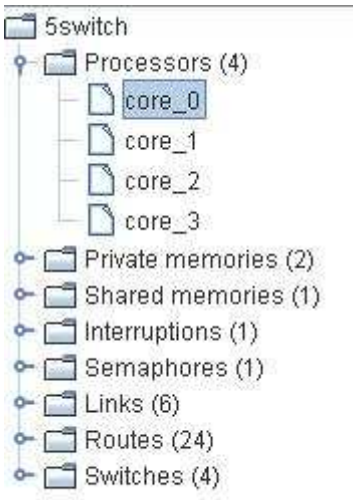
2.1 Main window



Buttons Bar: You get a ToolTip Text when you halts the pointer of the mouse over a button.



Network tree: On the tree you can see all the elements of the network, and make any modification by double-click on the element you want to modify.



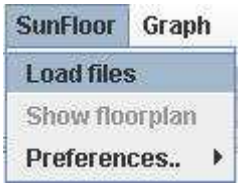
Sunfloor tree: Only active after invoking SunFloor. Allows watch the SunFloor specification table by double click on the table.

2.2 Open a network

There are two ways to open a network with NoCdificador:

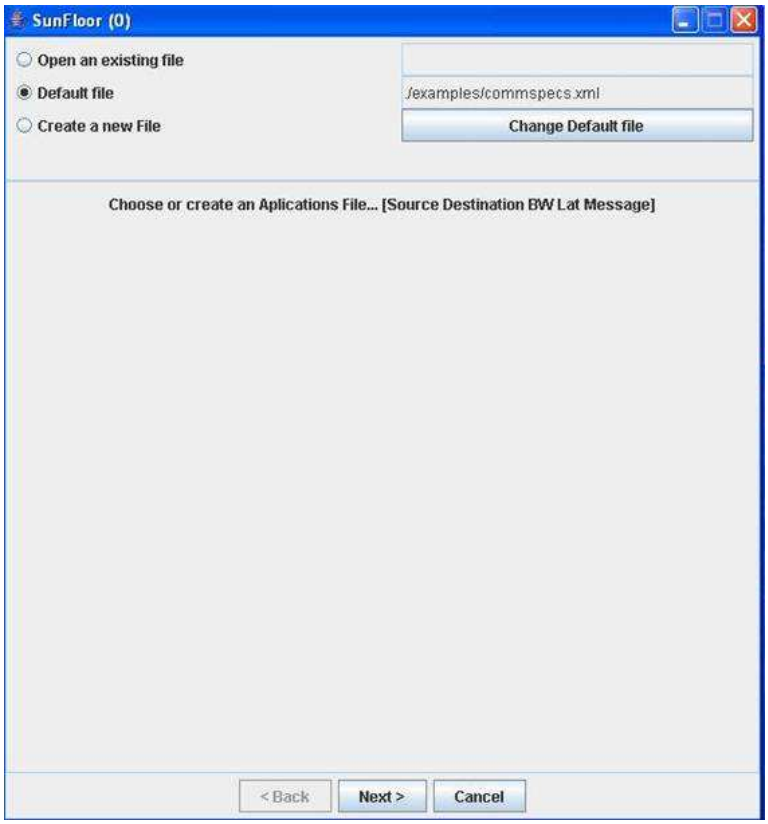
2.2.1 Using Sunfloor:

To choose this option, go to SunFloor→LoadFiles

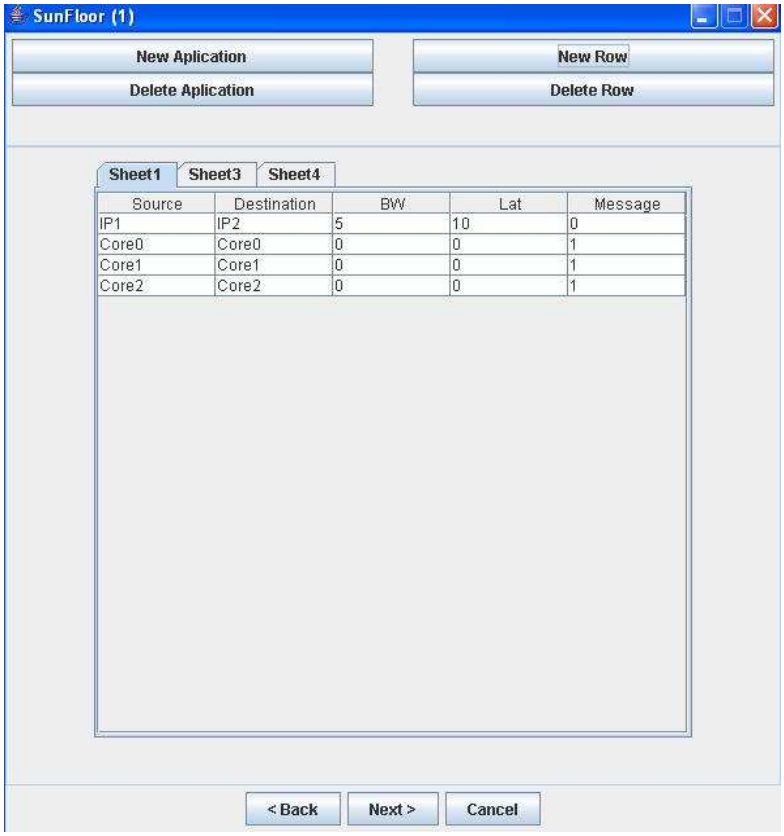


And follow these steps:

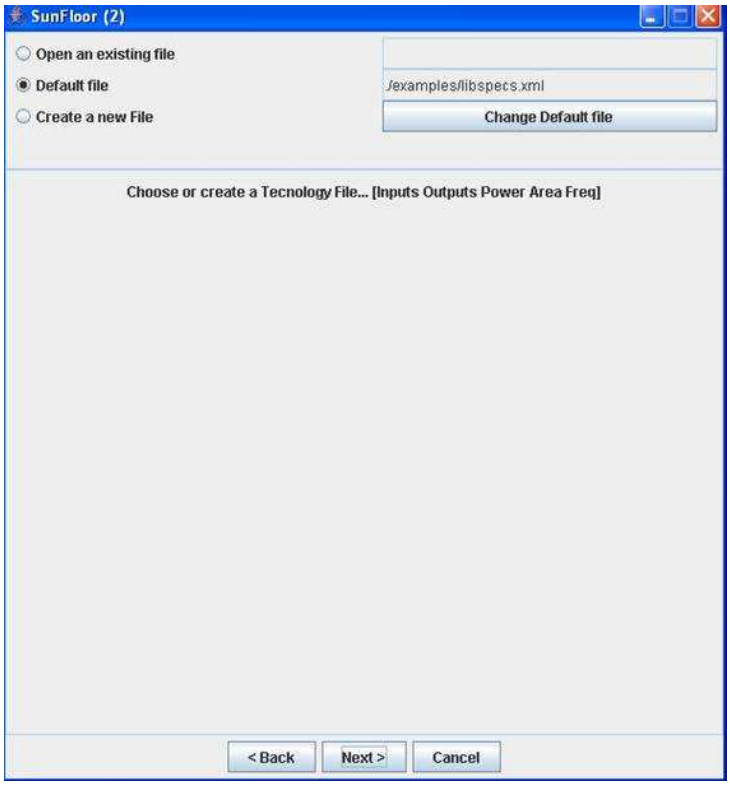
1. Choose create/load an application table. You can open a default one, choose an existing one or create a new one. Then press the next button.



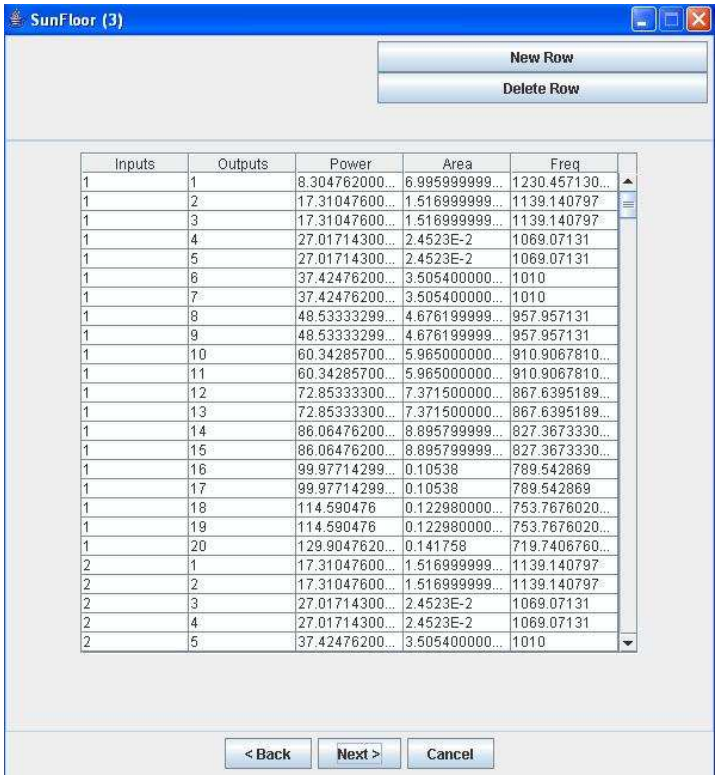
2. Fill the application table. You can create a new application or a new row into the active application, or delete the active application or the selected row. Then press the Next Button



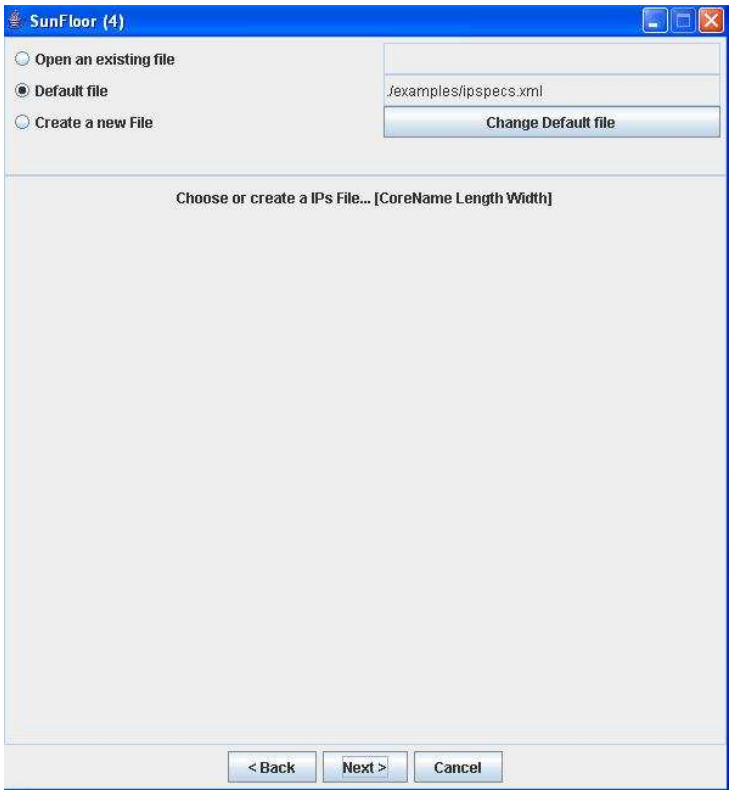
3. Choose create/load a technology table. You can open a default one, choose an existing one or create a new one. Then press the next button.



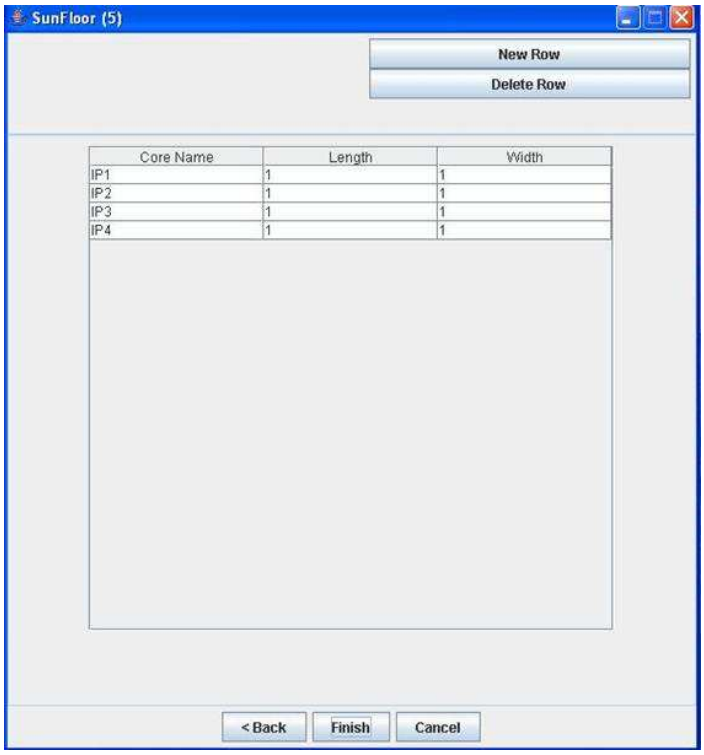
4. Fill the technology table creating or removing rows in the table. To go on, press the Next Button.



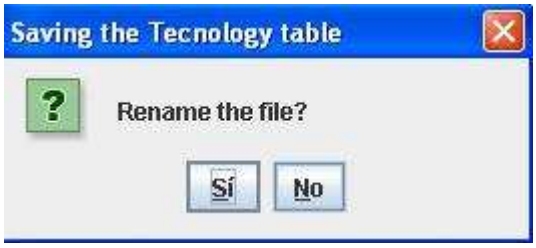
5. Choose create/load a technology table. You can open a default one, choose an existing one or create a new one. Then press the next button.

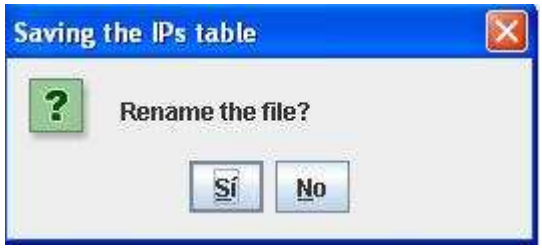


6. Fill the Ips table creating or removing rows in the table. Press Finish to save the tree tables.

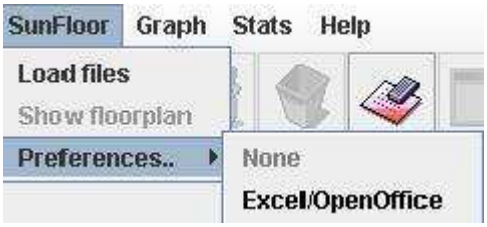


7. For each table, if it was opened you can rename or not rename it to save it with another name, but if the table was created you must save it with a name.



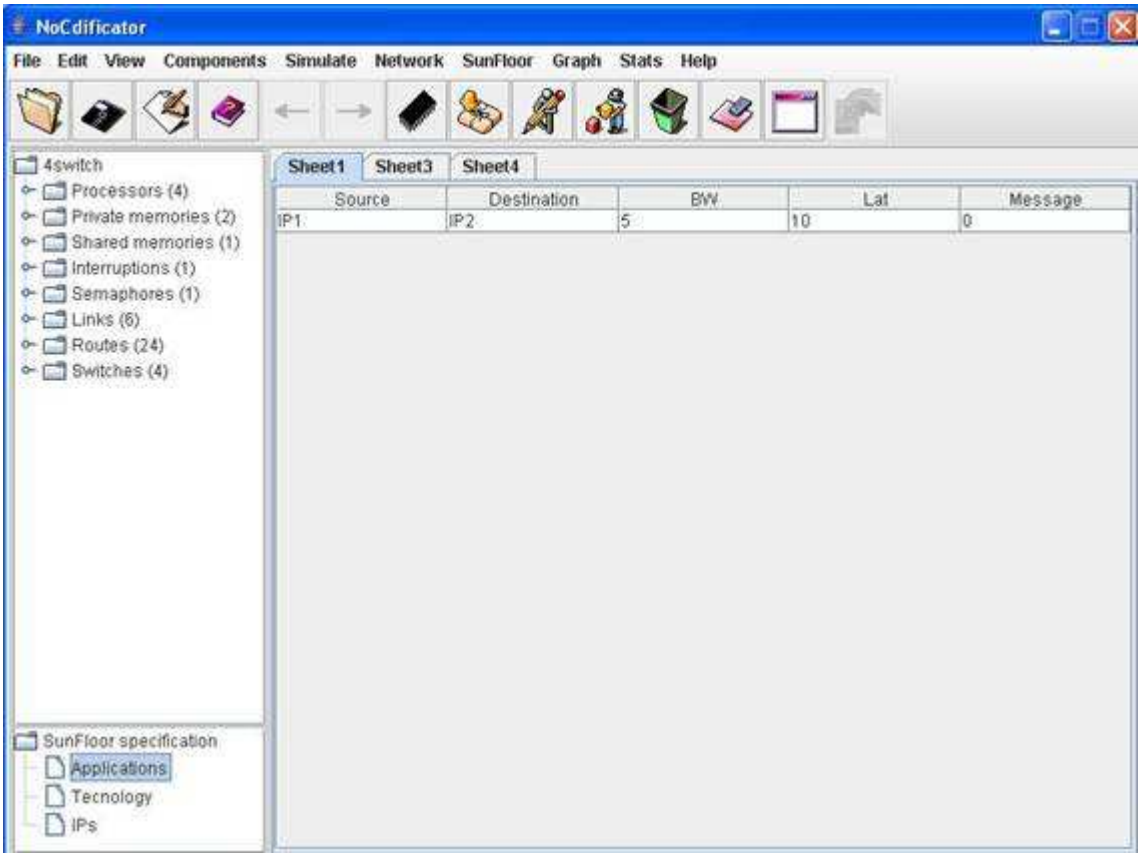


The tables will be saved into an XML format. There are two possibilities to save the tales: save the tables with an XML format that can be opened with excel or openOffice 2.0 or save the tables into an XML normal format.(they won't be able to open with excel or openOffice). For choose one of these options go to
SunFloor →Preferences →None/Excel

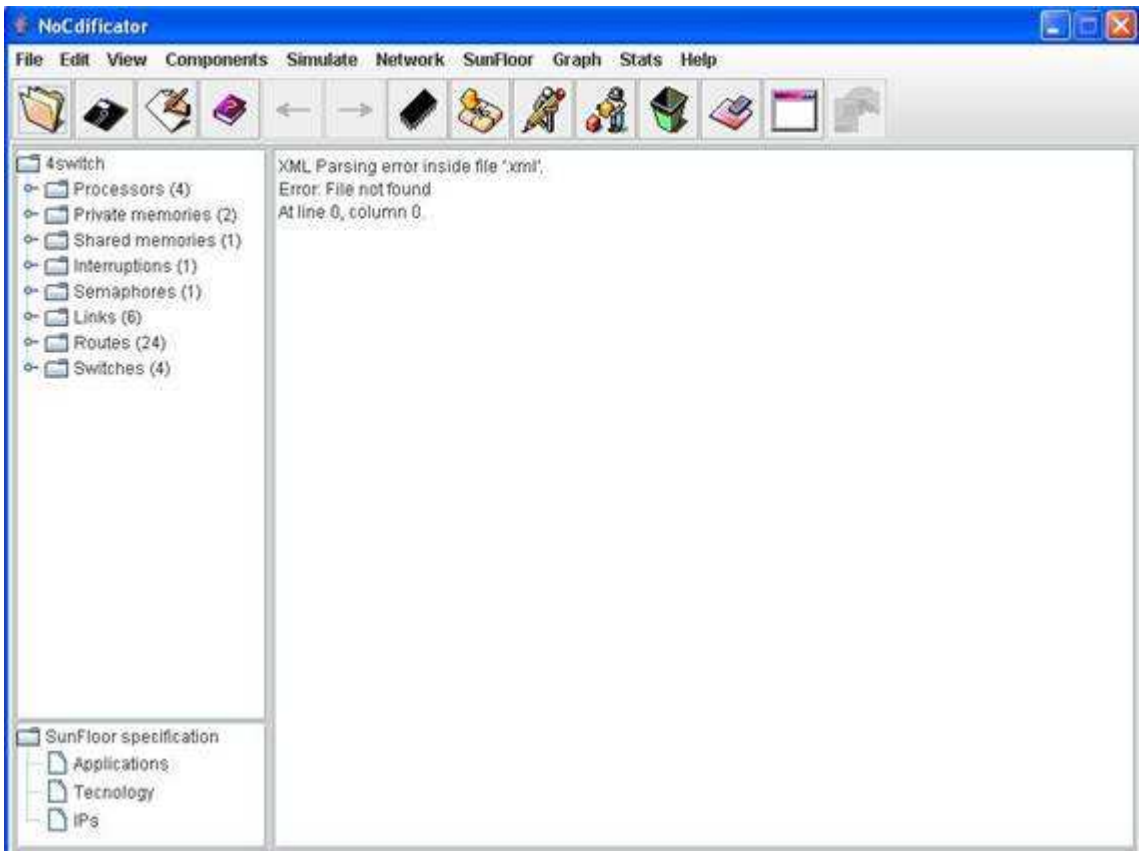


NoC frequency and the number of traffic generators must be specified now.

If all the tables are correct, Sunfloor generates a net. In the main window you will see the network tree, and below it there will be the Sunfloor tree with its three tables. You can see each table doing double click in its leaves.



If SunFloor has problems to generate the new net, warnings or errors will be shown into the main window.



You can cancel the Loading files operation in any moment. In this case, Sunfloor won't generate the new network.

More options you can do this way:

- You can change the default path of each table, to open a default table which is where you have chosen.

- You can go back to the previous step pressing the Back button.

- You can change the format of the XML files, to save the tables into an special XML format that allows to open the tables with excel or openOffice (SunFloor→Preferences→Excel/OpenOffice) or to save them into a normal XML format. (SunFloor →Preferences None). You have to choose it before Load files if you want to change it.

If SunFloor has success and the tables are correct, you can see the Floorplan generated by SunFloor (SunFloor →Show FloorPlan) in the main window.

You can see the Log file too, in these cases, because the network's file that Sunfloor generates, will be opened by the program.

To see the Log file go to: View →Log or press the Log button: 

2.2.2 Opening a TXT net file (View the Standard format)

There is a Standard TXT Format for define a network.(View the Standard format).

To open a net go to File→Open Network or press the Open button: 

To define the network into the TXT file:

First, define the Topology, with the networks name and the topology type:

topology(net_Name, topology_type**);**

where topology_type only can be : other, mesh or torus.

Then, declare the cores, private memories, shared memories, semaphores and interrupts with its correspondents attributes.

For cores:

core(core_Name, NI clock divider ,Input buffers depth, NI buffers, type_core, type_core2**);**

Where:

type_core only can be : **userdefined** or **requested** (if it is a specific one) and type_core2 only can be : **initiator** or **target**.

For private memories:

core(pm_Name, NI clock divider, Input buffers depth, NI buffers, type_core**, target :** mem_map**);**

Where:

type_core only can be : userdefined or requested (if it is a specific one) and mem_map has this format: 0x00, 0x01,....(is the memory mapping)

For shared memories:

core(shm_Name, NI clock divider, Input buffers depth, NI buffers, type_core, **target :** mem_map **-fixed);**

Where:

type_core only can be : userdefined or requested (if it is a specific one) and mem_map has this format: 0x00, 0x01,....(is the memory mapping)

fixed specifies that the memory position is fixed, only for shared and target cores.

For semaphores:

core(smm_Name, NI clock divider, Input buffers depth, NI buffers, type_core, **target :** mem_map **-fixed);**

Where:

type_core only can be : userdefined or requested (if it is a specific one) and mem_map has this format: 0x00, 0x01,....(is the memory mapping)

fixed specifies that the memory position is fixed, only for shared and target cores.

For interrupts:

core(int_Name, NI clock divider, Input buffers depth, NI buffers, type_core, **target :** mem_map **-fixed);**

Where:

type_core only can be : userdefined or requested (if it is a specific one) and mem_map has this format: 0x00, 0x01,....(is the memory mapping)

fixed specifies that the memory position is fixed, only for shared and target cores.

Then, declare the switches of the network and specifies its number of inputs, outputs and buffers, and Xcoor and Ycoor if the topology Type is mesh or torus:

switch(switch_Name, inputs_number, outputs_number, Input buffers depth, buffers_number**);**

switch(switch_Name, inputs_number, outputs_number, Input buffers depth, buffers_number,Xcoor,Ycoor**);**

To Continue define the links between switches (and cores, pms, shms, ... and swicthes).

You can't create a links between elements that don't exist. You have to declare them before.

link(link Name, element_source, element_target, number of repeaters**);**

Where:

element_source and element_target can be a switch or a type of core: **switch**_Name, **core**_Name, **pm**_Name, **shm**_Name, **smm**_Name or **int**_Name.

Links between two switches or a switch and a core type are allowed. But links between two types of core aren't.

If there are two links with the same Name and the element_source of one is the element_target of the other and the element_target of one is the element_source of the other, only a link with that Name will be created but it will be bidirectional.

To finish, declare routes:

A route has a source and a target(both must be core types) and the switches that take part in the route (in the rigth order). The source, the target and the switches have to be declared before.

route(coreType_Name, coreType_Name, **switches:** Name1, Name2,...NameN**);**

Where:

coreType only can be: core, pm, shm, smm or int.

And the Name of the switches can be its Name plus ".Letter" that means there is a subchannel in that switch.

Some Restrictions:

The black words are reserved words, and they mustn't change.

Two elements of the same type with the same Name are not allowed. (Except if they are links and the source and the target of one is the target and the source of the other)

The Name of each element can be an String without **-**, **(**, **)**, **;**, **:**, **.** and coma. The name "null" is a reserved name for the attributes of the incomplete elements. It means that the attribute with that doesn't exist yet.

It is not necessary to create some elements of each type. You can only create the topology to have a network file. But the topology must always be created in the network file.

For more information, view some network TXT examples:

2switches.TXT

```
// rules: specify a topology name (and possibly type);
// enter the cores first, then switches, then links

// assuming that first masters are entered, then the slaves in the //order as
// follows: masters, private memories, shared memory, semaphore, interrupt

// ----------------------------
// define the topology here
// name, mesh/torus specifier (mesh/torus/other)
// ----------------------------
// Topology:
topology(2switch, other);


// ----------------------------
// define the cores here
// core name and number, switch number, NI clock divider, NI buffers,
// initiator/target type, type of core (if a specific one is //requested),
// memory mapping (only if target), fixed specifier
// (only if target and of shared type)
// ----------------------------
// Cores:
core(core_0, 1,2, 6, userdefined, initiator);
core(core_1, 1,2, 6, userdefined, initiator);

// Private Memories:
core(pm_2, 1,2, 6, userdefined, target:0x00);
core(pm_3, 1,2, 6, userdefined, target:0x01);

// Shared Memories:
core(shm_4, 1,2, 6, userdefined, target:0x19-fixed);

// Semaphores:
core(smm_5, 1,2, 6, userdefined, target:0x20-fixed);

// Interrupts:
core(int_6, 1,2, 6, userdefined, target:0x21-fixed);
```

```
// ----------------------------
// define the switches here
// switch number, switch inputs, switch outputs, number of buffers,
// x coordinate (only if mesh/torus), y coordinate (only if //mesh/torus)
// coordinates are from upper left corner, counting from 0 included
// ----------------------------
// Switches:
switch(switch_0, 3, 3,2, 5);
switch(switch_1, 6, 6,2, 5);

// ----------------------------
// define the links here
// link number, source, destination
// ----------------------------
// Links:
link(link0, switch_0, switch_1,0);
link(link1, switch_1, switch_0,0);
link(link0c, core_0, switch_0,0);
link(link1c, core_1, switch_0,0);
link(link2pm, switch_1, pm_2,0);
link(link3pm, switch_1, pm_3,0);
link(link4shm, switch_1, shm_4,0);
link(link5smm, switch_1, smm_5,0);
link(link6int, switch_1, int_6,0);
// ----------------------------
// define the routes here
// source core, destination core, the order in which switches need to //be
// traversed from the source core to the destination core
// ----------------------------
// Routes:
route(core_0, pm_2, switches:0, 1);
route(core_0, shm_4, switches:0, 1);
route(core_0, smm_5, switches:0, 1);
route(core_0, int_6, switches:0, 1);
route(core_1, pm_3, switches:0, 1);
route(core_1, shm_4, switches:0, 1);
route(core_1, smm_5, switches:0, 1);
route(core_1, int_6, switches:0, 1);
route(pm_2, core_0, switches:1, 0);
route(pm_3, core_1, switches:1, 0);
route(shm_4, core_0, switches:1, 0);
route(shm_4, core_1, switches:1, 0);
route(smm_5, core_0, switches:1, 0);
route(smm_5, core_1, switches:1, 0);
route(int_6, core_0, switches:1, 0);
route(int_6, core_1, switches:1, 0);
```

Another example more complex is ***sunfloorNet.TXT*** :

```
topology(topologyname, other);
core(core_0,  1, 2, 6, userdefined, initiator);
core(core_1,  1, 2, 6, userdefined, initiator);
core(core_2,  1, 2, 6, userdefined, initiator);
core(core_3,  1, 2, 6, userdefined, initiator);
core(core_4,  1, 2, 6, userdefined, initiator);
core(core_5,  1, 2, 6, userdefined, initiator);
core(core_6,  1, 2, 6, userdefined, initiator);
core(core_7,  1, 2, 6, userdefined, initiator);
core(core_8,  1, 2, 6, userdefined, initiator);
core(core_9,  1, 2, 6, userdefined, initiator);
core(core_10,  1, 2, 6, userdefined, initiator);
core(core_11,  1, 2, 6, userdefined, initiator);
core(core_12,  1, 2, 6, userdefined, initiator);
core(core_13,  1, 2, 6, userdefined, initiator);
core(core_14,  1, 2, 6, userdefined, initiator);
core(pm_15,  1,2, 6, userdefined, target:0x00);
core(pm_16,  1,2, 6, userdefined, target:0x01);
core(pm_17,  1,2, 6, userdefined, target:0x02);
core(pm_18,  1,2, 6, userdefined, target:0x03);
core(pm_19,  1,2, 6, userdefined, target:0x04);
core(pm_20,  1,2, 6, userdefined, target:0x05);
core(pm_21,  1,2, 6, userdefined, target:0x06);
core(pm_22,  1,2, 6, userdefined, target:0x07);
core(pm_23,  1,2, 6, userdefined, target:0x08);
core(pm_24,  1,2, 6, userdefined, target:0x09);
core(pm_25,  1,2, 6, userdefined, target:0x0a);
core(pm_26,  1,2, 6, userdefined, target:0x0b);
core(shm_27,  1,2, 6, userdefined, target:0x19-fixed);
core(smm_28,  1,2, 6, userdefined, target:0x20-fixed);
core(int_29,  1,2, 6, userdefined, target:0x21-fixed);
switch(switch_0, 6, 5,2, 4);
switch(switch_1, 6, 6,2, 4);
switch(switch_2, 4, 4,2, 4);
switch(switch_3, 11, 8,2, 4);
switch(switch_4, 9, 9,2, 4);
switch(switch_5, 6, 8,2, 4);
switch(switch_6, 6, 8,2, 4);
link(link_0, core_0, switch_3, 0);
link(link_1, switch_3, core_0, 0);
link(link_2, core_1, switch_1, 0);
link(link_3, switch_1, core_1, 0);
link(link_4, core_2, switch_0, 0);
link(link_5, switch_0, core_2, 0);
link(link_6, core_3, switch_5, 0);
link(link_7, switch_5, core_3, 0);
link(link_8, core_4, switch_6, 0);
link(link_9, switch_6, core_4, 0);
link(link_10, core_5, switch_0, 0);
link(link_11, switch_0, core_5, 0);
link(link_12, core_6, switch_6, 0);
link(link_13, switch_6, core_6, 0);
link(link_14, core_7, switch_3, 0);
link(link_15, switch_3, core_7, 0);
link(link_16, core_8, switch_1, 0);
link(link_17, switch_1, core_8, 0);
link(link_18, core_9, switch_5, 0);
link(link_19, switch_5, core_9, 0);
link(link_20, core_10, switch_4, 0);
link(link_21, switch_4, core_10, 0);
link(link_22, core_11, switch_2, 0);
```

```
link(link_23, switch_2, core_11, 0);
link(link_24, core_12, switch_4, 0);
link(link_25, switch_4, core_12, 0);
link(link_26, core_13, switch_2, 0);
link(link_27, switch_2, core_13, 0);
link(link_28, core_14, switch_2, 0);
link(link_29, switch_2, core_14, 0);
link(link_30, pm_15, switch_3, 0);
link(link_31, switch_3, pm_15, 0);
link(link_32, pm_16, switch_1, 0);
link(link_33, switch_1, pm_16, 0);
link(link_34, pm_17, switch_0, 0);
link(link_35, switch_0, pm_17, 0);
link(link_36, pm_18, switch_5, 0);
link(link_37, switch_5, pm_18, 0);
link(link_38, pm_19, switch_6, 0);
link(link_39, switch_6, pm_19, 0);
link(link_40, pm_20, switch_0, 0);
link(link_41, switch_0, pm_20, 0);
link(link_42, pm_21, switch_6, 0);
link(link_43, switch_6, pm_21, 0);
link(link_44, pm_22, switch_3, 0);
link(link_45, switch_3, pm_22, 0);
link(link_46, pm_23, switch_1, 0);
link(link_47, switch_1, pm_23, 0);
link(link_48, pm_24, switch_5, 0);
link(link_49, switch_5, pm_24, 0);
link(link_50, pm_25, switch_4, 0);
link(link_51, switch_4, pm_25, 0);
link(link_52, pm_26, switch_2, 0);
link(link_53, switch_2, pm_26, 0);
link(link_54, shm_27, switch_4, 0);
link(link_55, switch_4, shm_27, 0);
link(link_56, smm_28, switch_4, 0);
link(link_57, switch_4, smm_28, 0);
link(link_58, int_29, switch_3, 0);
link(link_59, switch_3, int_29, 0);
link(link60, switch_0, switch_3, 1);
link(link61, switch_1, switch_3, 1);
link(link62, switch_1, switch_4, 1);
link(link63, switch_3, switch_4, 1);
link(link64, switch_3, switch_5, 1);
link(link65, switch_3, switch_6, 1);
link(link66, switch_4, switch_0, 1);
link(link67, switch_4, switch_1, 1);
link(link68, switch_4, switch_5, 1);
link(link69, switch_4, switch_6, 1);
link(link70, switch_5, switch_1, 1);
link(link71, switch_5, switch_3, 1);
link(link72, switch_5, switch_3, 1);
link(link73, switch_5, switch_4, 1);
link(link74, switch_6, switch_0, 0);
link(link75, switch_6, switch_3, 1);
link(link76, switch_6, switch_3, 1);
link(link77, switch_6, switch_4, 1);
route(core_0, pm_15, switches:3);
route(core_0, shm_27, switches:3, 4);
route(core_0, smm_28, switches:3, 4);
route(core_0, int_29, switches:3);
route(core_1, pm_16, switches:1);
route(core_1, shm_27, switches:1, 4);
route(core_1, smm_28, switches:1, 4);
route(core_1, int_29, switches:1, 3);
route(core_2, pm_17, switches:0);
```

```
route(core_2, shm_27, switches:0, 3, 4);
route(core_2, smm_28, switches:0, 3, 4);
route(core_2, int_29, switches:0, 3);
route(core_3, pm_18, switches:5);
route(core_3, shm_27, switches:5, 4);
route(core_3, smm_28, switches:5, 4);
route(core_3, int_29, switches:5.B, 3);
route(core_4, pm_19, switches:6);
route(core_4, shm_27, switches:6, 4);
route(core_4, smm_28, switches:6, 4);
route(core_4, int_29, switches:6.B, 3);
route(core_5, pm_20, switches:0);
route(core_5, shm_27, switches:0, 3, 4);
route(core_5, smm_28, switches:0, 3, 4);
route(core_5, int_29, switches:0, 3);
route(core_6, pm_21, switches:6);
route(core_6, shm_27, switches:6, 4);
route(core_6, smm_28, switches:6, 4);
route(core_6, int_29, switches:6.B, 3);
route(core_7, pm_22, switches:3);
route(core_7, shm_27, switches:3, 4);
route(core_7, smm_28, switches:3, 4);
route(core_7, int_29, switches:3);
route(core_8, pm_23, switches:1);
route(core_8, shm_27, switches:1, 4);
route(core_8, smm_28, switches:1, 4);
route(core_8, int_29, switches:1, 3);
route(core_9, pm_24, switches:5);
route(core_9, shm_27, switches:5, 4);
route(core_9, smm_28, switches:5, 4);
route(core_9, int_29, switches:5.B, 3);
route(core_10, pm_25, switches:4);
route(core_11, pm_26, switches:2);
route(core_12, pm_25, switches:4);
route(core_12, shm_27, switches:4);
route(core_12, smm_28, switches:4);
route(core_13, pm_26, switches:2);
route(core_14, pm_26, switches:2);
route(pm_15, core_0, switches:3);
route(pm_16, core_1, switches:1);
route(pm_17, core_2, switches:0);
route(pm_18, core_3, switches:5);
route(pm_19, core_4, switches:6);
route(pm_20, core_5, switches:0);
route(pm_21, core_6, switches:6);
route(pm_22, core_7, switches:3);
route(pm_23, core_8, switches:1);
route(pm_24, core_9, switches:5);
route(pm_25, core_10, switches:4);
route(pm_25, core_12, switches:4);
route(pm_26, core_11, switches:2);
route(pm_26, core_13, switches:2);
route(pm_26, core_14, switches:2);
route(shm_27, core_0, switches:4, 5.A, 3);
route(shm_27, core_1, switches:4, 1);
route(shm_27, core_2, switches:4, 0);
route(shm_27, core_3, switches:4, 5);
route(shm_27, core_4, switches:4, 6);
route(shm_27, core_5, switches:4, 0);
route(shm_27, core_6, switches:4, 6);
route(shm_27, core_7, switches:4, 5.A, 3);
route(shm_27, core_8, switches:4, 1);
route(shm_27, core_9, switches:4, 5);
route(shm_27, core_12, switches:4);
```

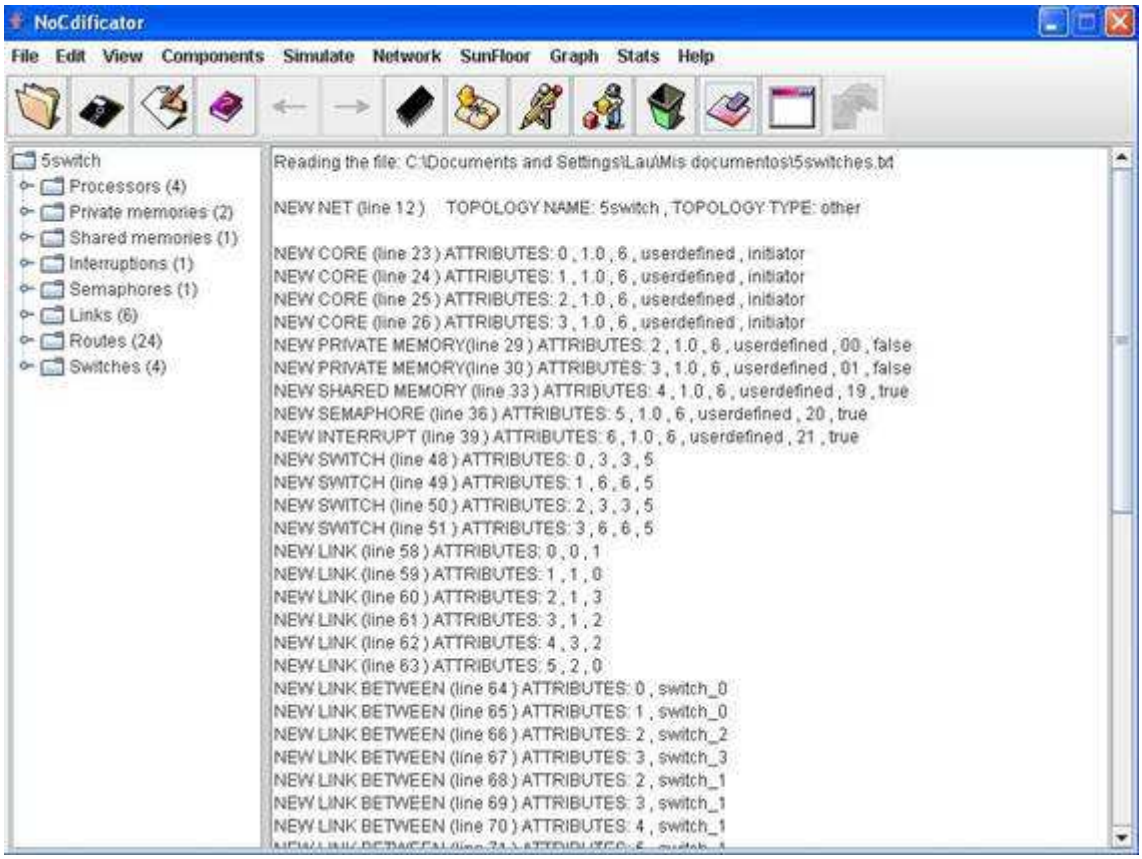
```
route(smm_28, core_0, switches:4, 6.A, 3);
route(smm_28, core_1, switches:4, 1);
route(smm_28, core_2, switches:4, 0);
route(smm_28, core_3, switches:4, 5);
route(smm_28, core_4, switches:4, 6);
route(smm_28, core_5, switches:4, 0);
route(smm_28, core_6, switches:4, 6);
route(smm_28, core_7, switches:4, 5.A, 3);
route(smm_28, core_8, switches:4, 1);
route(smm_28, core_9, switches:4, 5);
route(smm_28, core_12, switches:4);
route(int_29, core_0, switches:3);
route(int_29, core_1, switches:3, 5, 1);
route(int_29, core_2, switches:3, 6, 0);
route(int_29, core_3, switches:3, 5);
route(int_29, core_4, switches:3, 6);
route(int_29, core_5, switches:3, 6, 0);
route(int_29, core_6, switches:3, 6);
route(int_29, core_7, switches:3);
route(int_29, core_8, switches:3, 5, 1);
route(int_29, core_9, switches:3, 5);
```

The TXT file won't be opened if the file hasn't a defined topology in it. In other case the net will be opened. If there are errors in some element's declarations, these elements won't be created and NoCdificador advises you to check these errors looking the Log

(View→Log or with the Log button )

View Log File

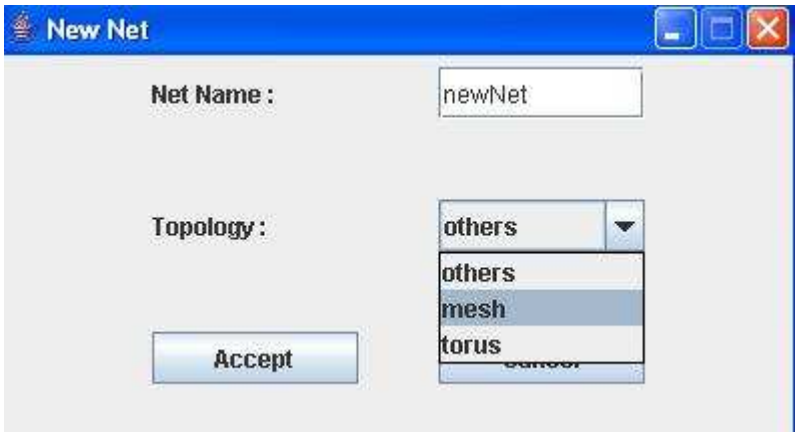
When a txt network file is opened a new File is created at the same path with the name of the first one plus "_Advise". This new file shows the possible errors or the actions that have been occurred during the opening of the net. You can see this file from the Nocdificador's main window (View →Log)



2.3. Create a new network

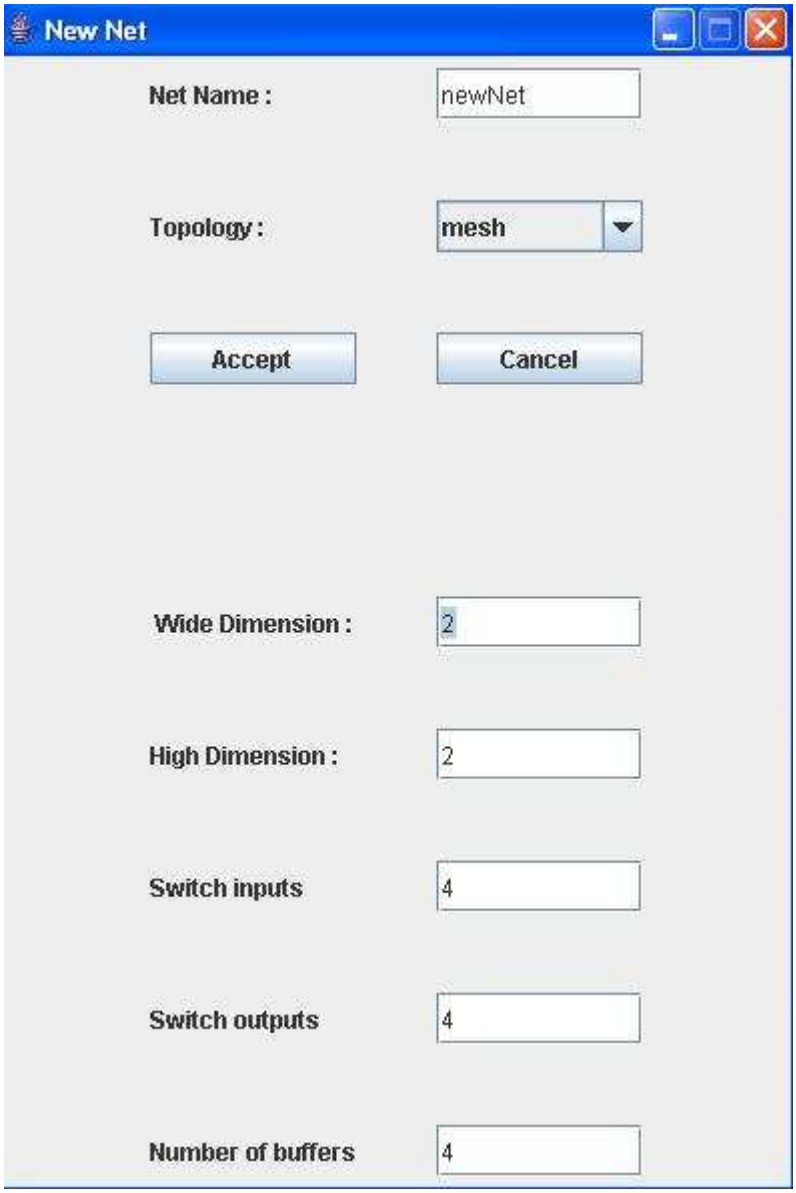
To create a new net go to File →New or press the New network button: 

It is necessary fill the network's name and choose the type of the topology (others, mesh or torus)

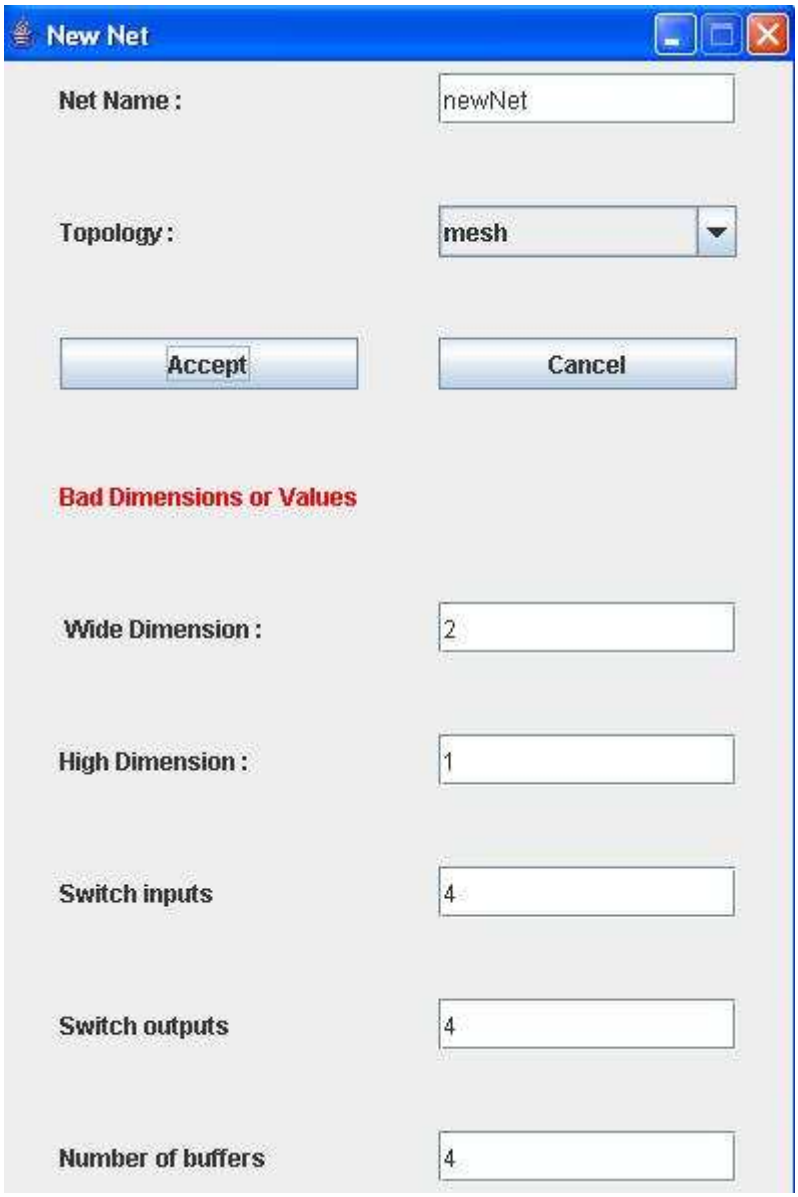


If the type is others the net will be created empty, only with that name and that topology type.

If the type is mesh or torus you must fill the Dimension gaps and the switch's attributes (number of inputs, outputs and buffers).



Minimum allowed dimension is 2 x 2 and minimum inputs and outputs for the switches are 4.



In this case the net will be created with that name, that topology type and only with the switches and the links between them (depends on the topology type, mesh or torus)

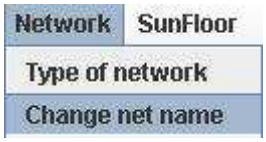
2.4 Modify the network

To modify the net, the net must be opened or created. There are many ways for modify the net:

- Change the network's name
- Create a new network's element
- Modify an existing network's element
- Delete an existing Network's element
- Undo and Redo actions

2.4.1 Change the network's name

To change the name of the net go to Network→Change net name.



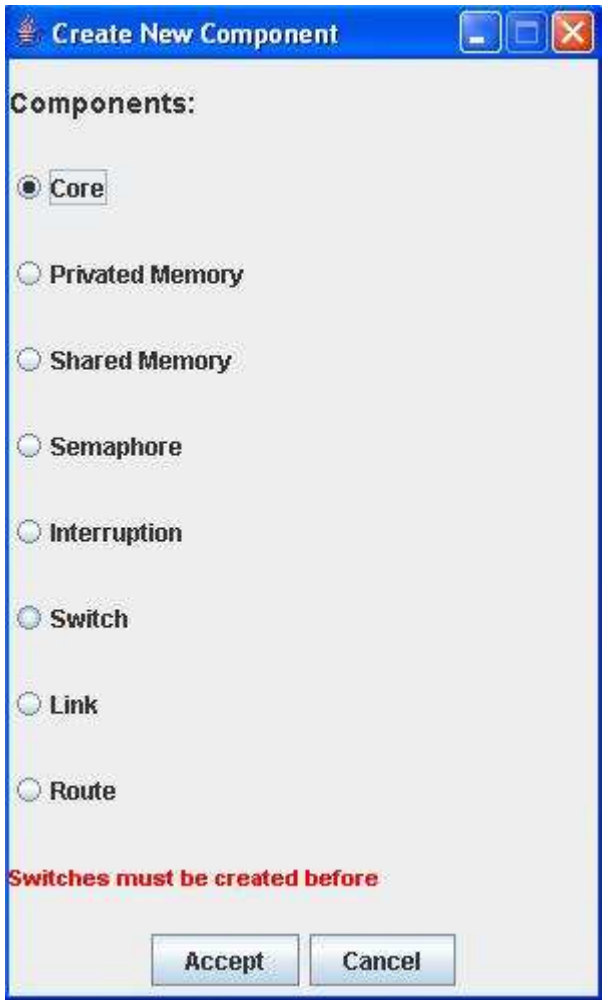
This is a simple modify and you couldn't undo this change, To undo it change the name again.

2.4.2 Create a new network's element

To create a new element in the net go to Components→New Component and choose the type of component you want create,

Or press the New Component Button: 

These are the component types: core, private memory, shared memory, semaphore, interrupt, switch, link and route.



If there aren't any switches in the net you will have to create them before create other elements.

Restrictions for the new components:

Don't use the same name for a new component if exists other of the same type with that name.

Fill all the gaps for the attributes of the component.

If the new component is a Route the source of it has to be initiator type and the target has to be target type. Also its switches list mustn't be empty. But it can be wrong connected, in this case the route will be created but it will be an incomplete route.

The used outputs and inputs of the switches can cause problems when a new core, private memory, shared memory, semaphore or interrupt is created with an associated switch that has all its connection busy. The same happens when a link is going to be created between two switches that have all its outputs and inputs full.

In all these cases if the new component has its attributes are correct, it will be created in the network. Then you can modify or delete it.

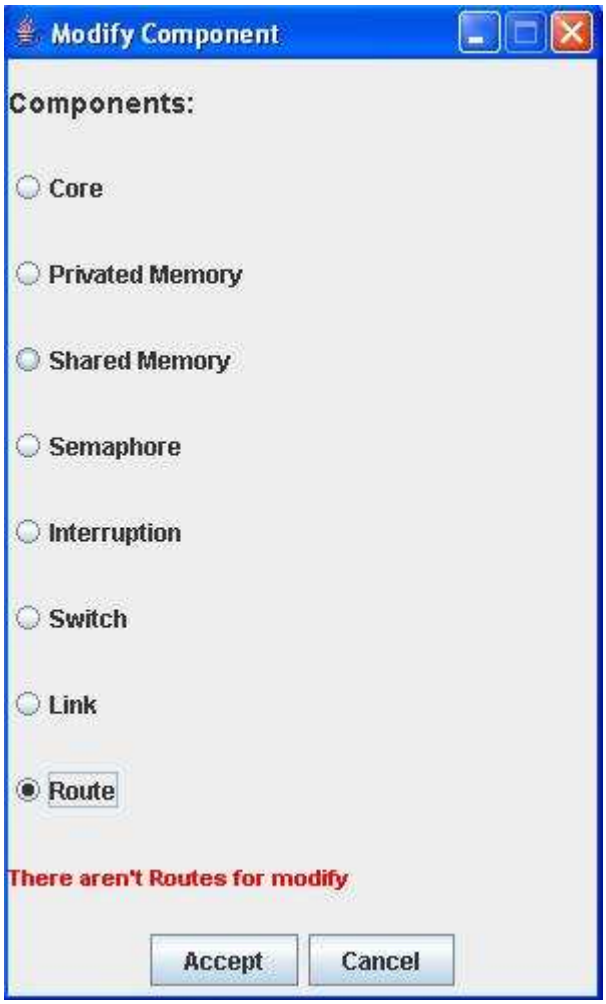
2.4.3 Modify an existing network's element

To modify an existing element in the net go to Components→Modify Component and choose the type of component you want modify.

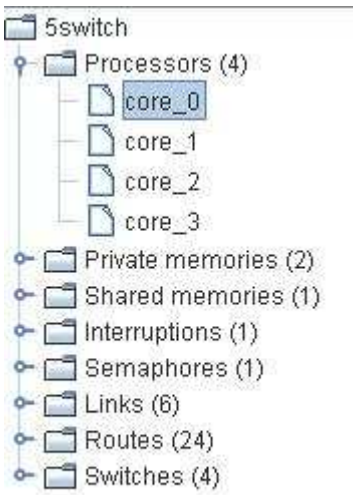
Or press the modify component button: 

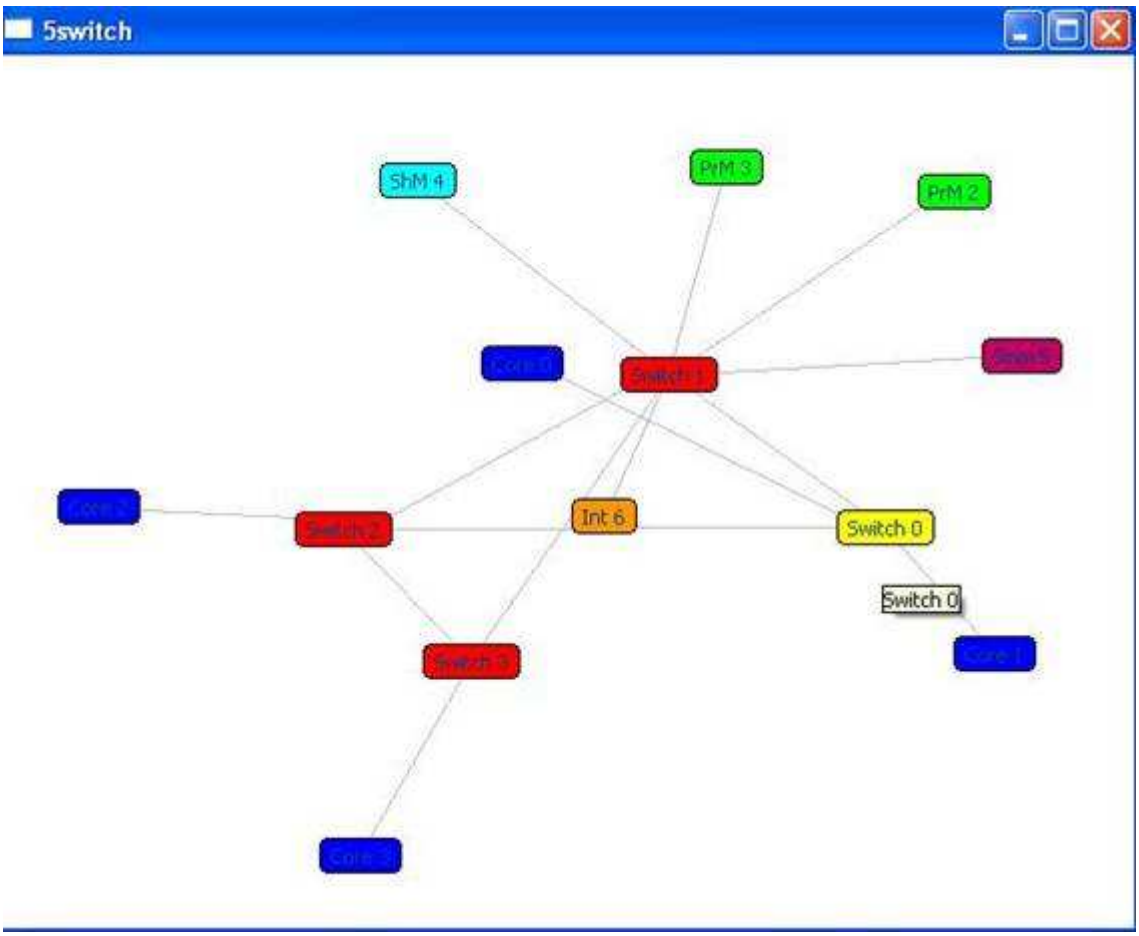
These are the component types: core, private memory, shared memory, semaphore, interrupt, switch, link and route.

You can't modify a type of component if there isn't any component of that type in the net. If any component of the chosen type exists you can modify it choosing it between all the components of the same type.



Other way more direct to modify it is doing double click on the component (On the component's leaf in the network's tree at the main window or on the component's node of the network's graph).





The restrictions for modify a component are the same that for create it. But if the component is incomplete, you could modify it even without specifying its incomplete attributes.

- Modify some attributes of the components can cause that other components turn into incomplete components:

- Modify the associated switch of a core, pm, shm, smm or int causes the routes where it took part turn into incomplete routes.

- Modify the target or the source of a link causes the routes where it took part turn into incomplete routes.

- Modify the switches list of a route can causes that route turns into incomplete route, if the switches list is a wrong list because of there isn't all the connections between its switches.

- Modify the number of inputs or outputs of a switch can cause problems if all its connections are used.

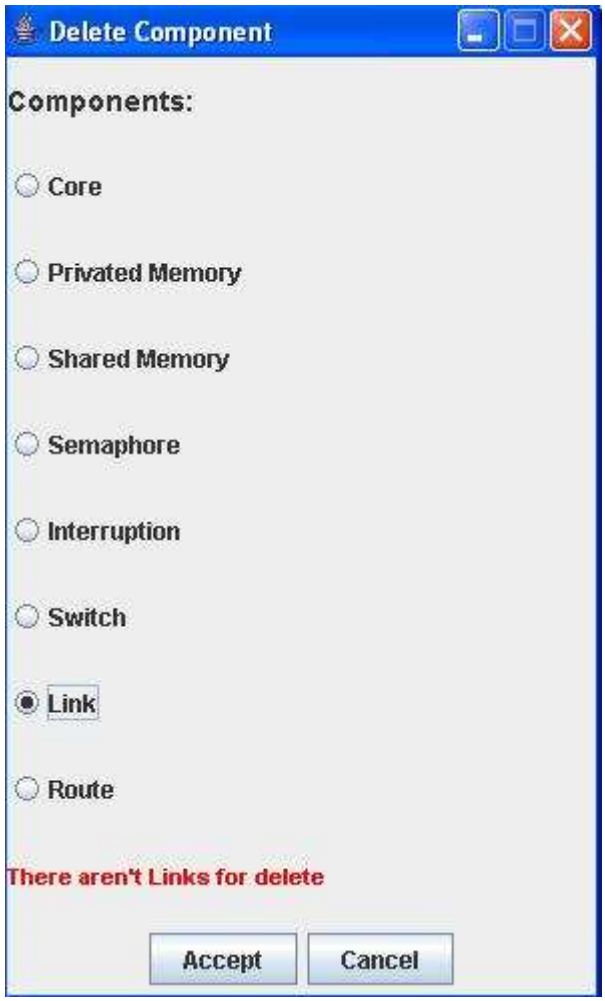
2.4.4 Delete an existing Network's element

To delete an existing element in the net go to Components→Delete Component and choose the type of component you want delete.

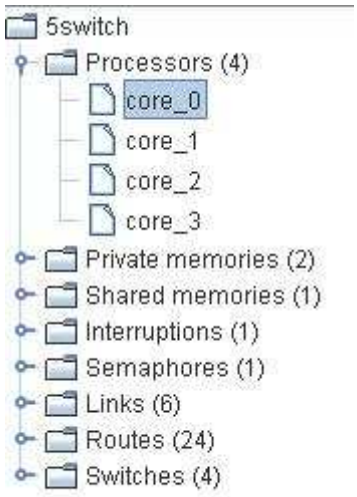
Or press the delete component button: 

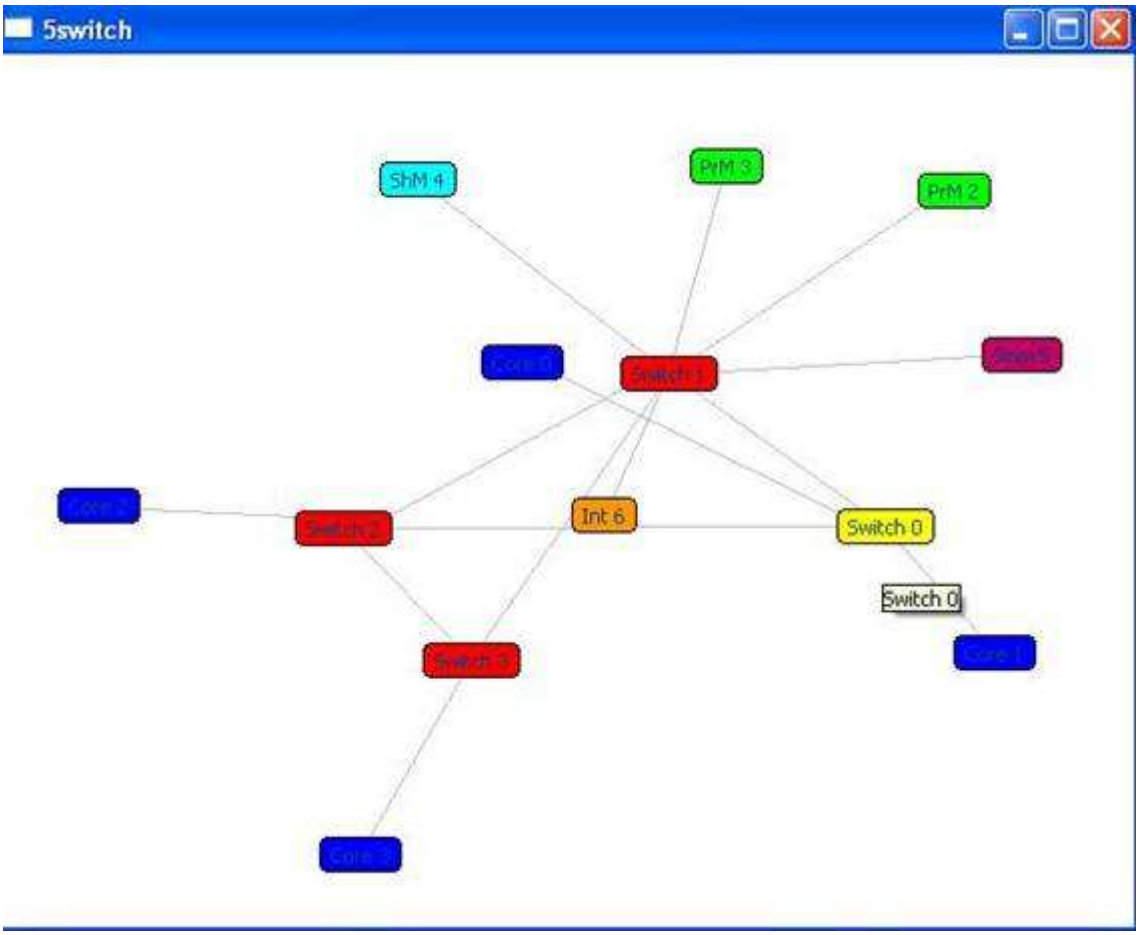
These are the component types: core, private memory, shared memory, semaphore, interrupt, switch, link and route.

You can't delete a type of component if there isn't any component of that type in the net. If any component of the chosen type exists you can delete it choosing it between all the components of the same type.



Other way more direct to delete it is doing double click on the component (On the component's leaf in the network's tree at the main window or on the component's node of the network's graph).





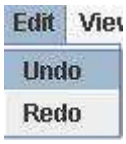
There isn't a restriction for delete an existing component, but it can cause that other components turn into incomplete components:

- Delete a core, private memory, shared memory, semaphore or interrupt makes the routes where it took part turn into incomplete routes.
- Delete a link makes the routes where it took part turn into incomplete routes.
- Delete a switch makes:
  - Routes where it took part turn into incomplete routes.
  - Links where it was the source or the target turn into incomplete links.
  - Cores, pm, shm, smm, int where it was the associated switch turn into incomplete components.
- Delete a route doesn't cause these problems.

2.4.5 Undo and Redo actions

You can undo and redo the last three changes of create, modify and delete components in the net.

To undo a change happened in the net go to Edit Undo or press the Undo Button:



To redo an Undo action before go to Edit→Redo or press the Redo Button :

Change the net name can't be undo or redo, but it is a simple change that can be resolved changing the name of the net again.

2.5 View information about the network

There is much information about the network that you can see with NoCdificador. To see it the net must exist, it has to have been created or opened before.

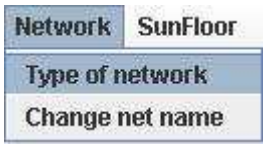
- View the network's topology
- View the Log
- View Information about each component
- View Incomplete elements

- View SunFloor tables
- View the FloorPlan
- Simulate the Routes
- View the network's Graph

- View Xpipes and AMBA Stats

2.5.1 View the network's topology

To see the topology type of the net go to Network→Type of network.



The topology type only can be: others mesh or torus. The last two ones make the net have a non-uniform structure, but with the fist one the net will have another structure, not uniform.

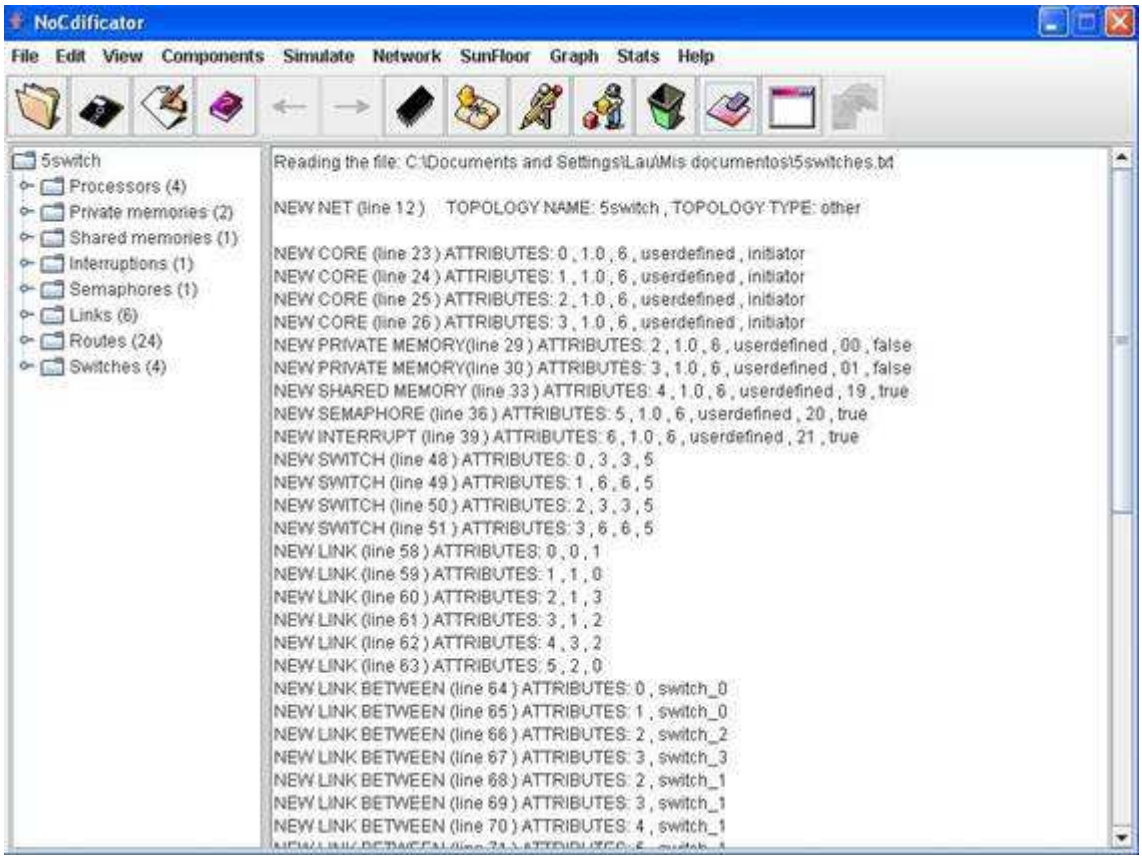
2.5.2 View the Log

The Log is a file that is created when a net is opened. The Log has all the information about the opened net and also the possible errors of its structure or format.

This file can be helpful for check the errors of the format network file because it advises of the errors, shows in where line of the file they are, and what type of error is. To see it go to View→Log.

Or use the Log button: 

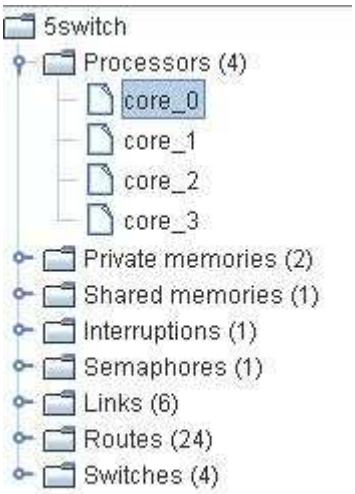
Even the file hasn't errors, looking the Log you can see in which line each element has been created and its attributes.

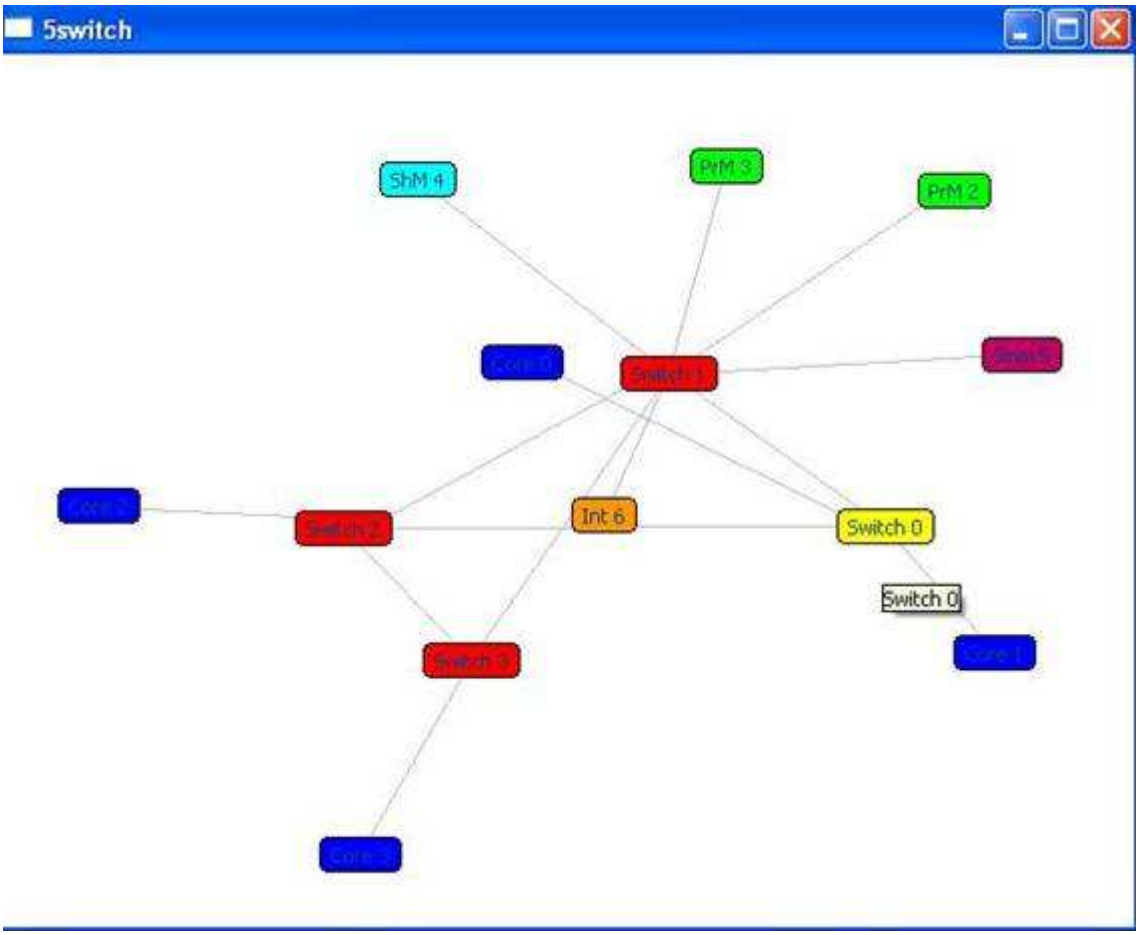


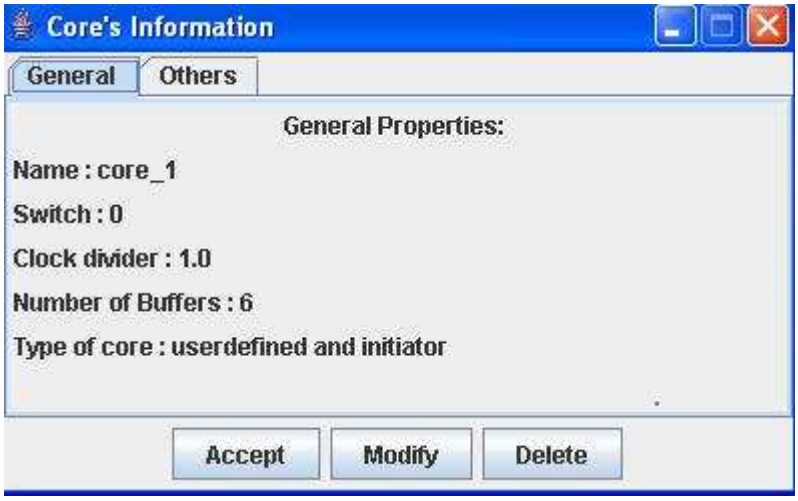
You only can see the Log if a net has been opened from a file or by SunFloor, not if it has been created by you from NoCdificador.

2.5.3 View Information about each component

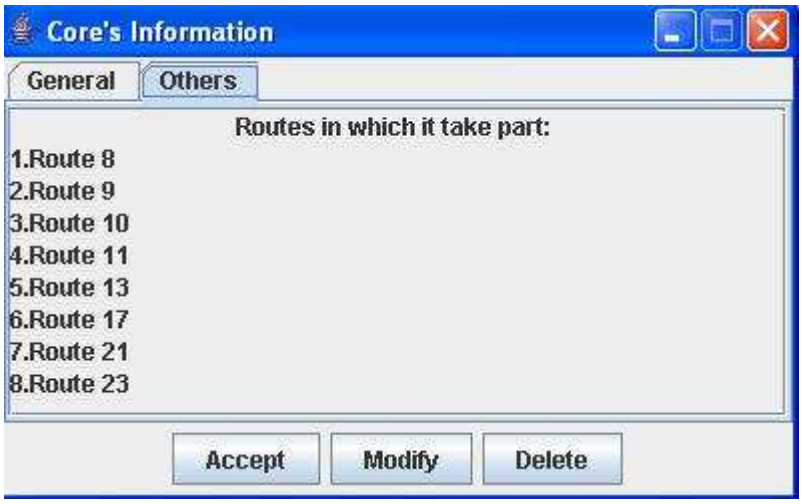
To see the attributes of each network's component do double click on it (On the component's leaf in the network's tree at the main window or on the component's node of the network's graph). A new window shows its characteristics.





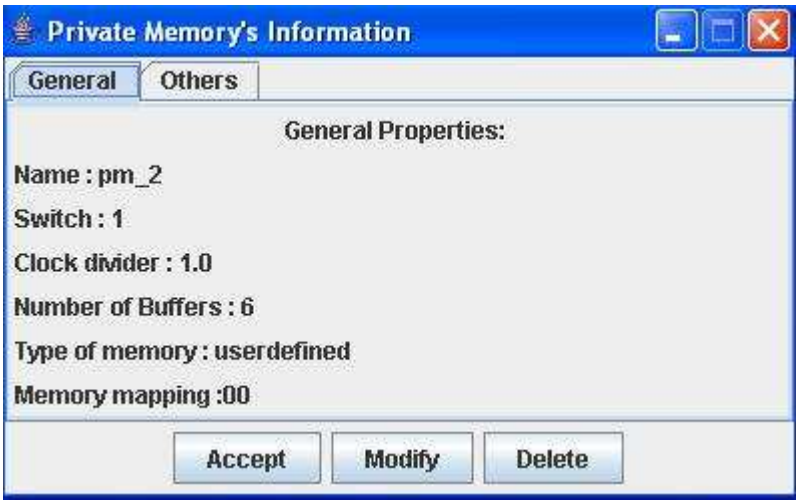


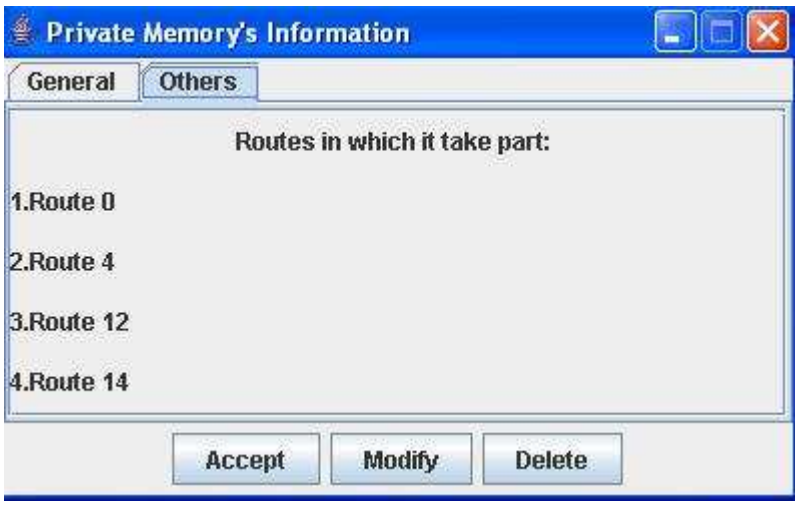
Also you can see other relational information about it in that window choosing "Other":

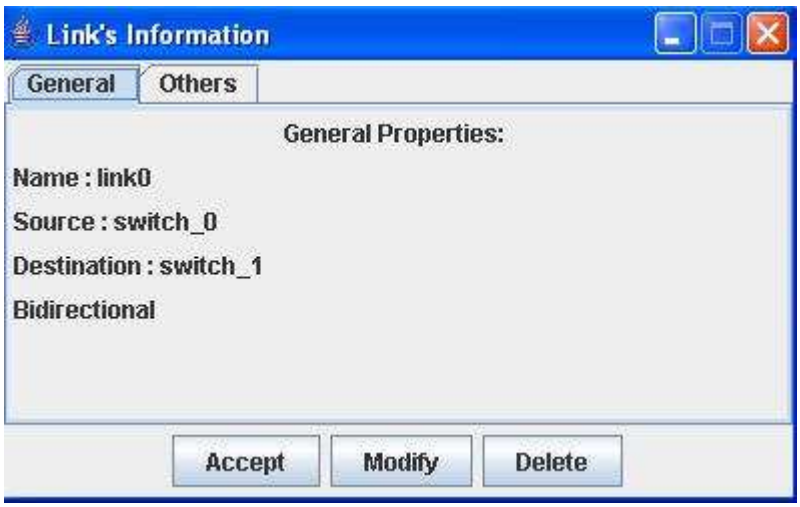


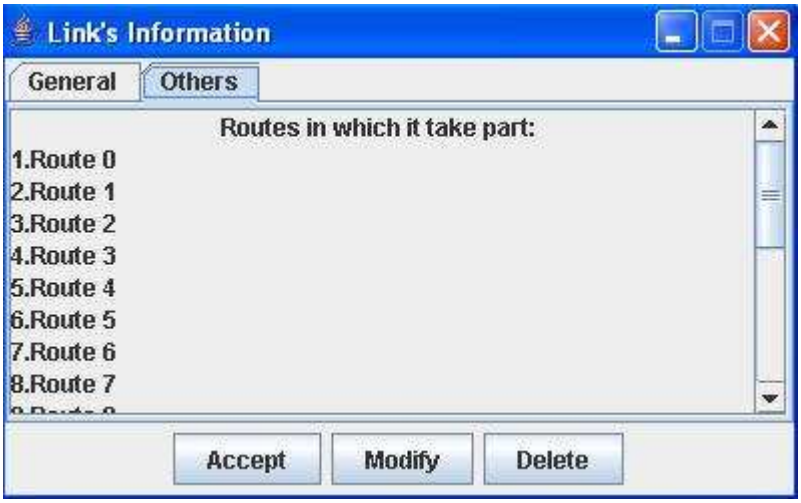
Other information about a:

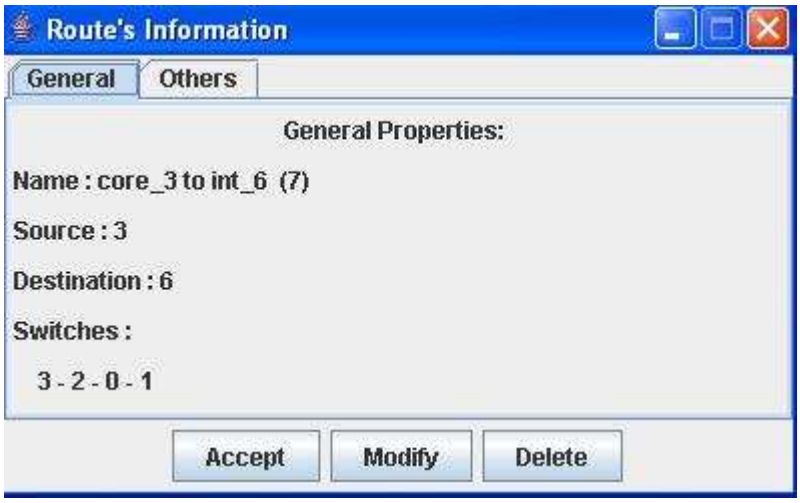
- Core, pm, shm, smm or int is the routes where it takes part.
- Switch is the routes where it takes part.
- Link is the routes where it takes part.
- Route is the links that take part on it.

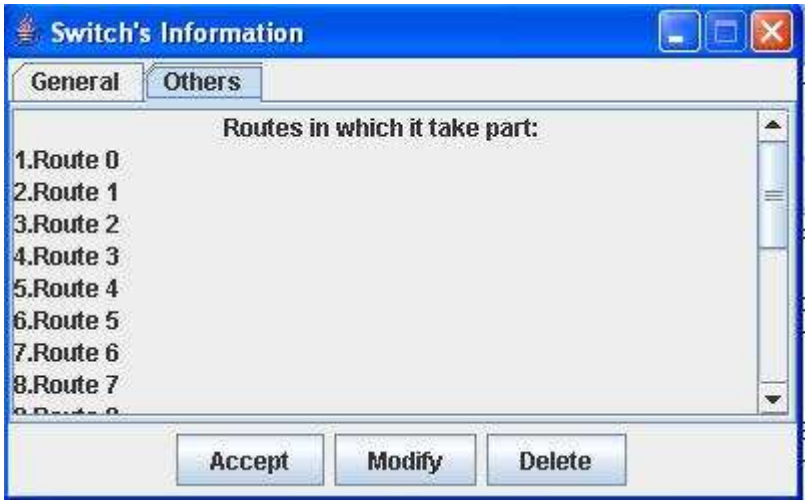





Link's Information
General
Others
General Properties:
Name : link0
Source : switch_0
Destination : switch_1
Bidirectional
Accept
Modify
Delete


Link's Information
General
Others
Routes in which it take part:
1.Route 0
2.Route 1
3.Route 2
4.Route 3
5.Route 4
6.Route 5
7.Route 6
8.Route 7
Accept
Modify
Delete


Route's Information
General
Others
General Properties:
Name : core_3 to int_6 (7)
Source : 3
Destination : 6
Switches :
3 - 2 - 0 - 1
Accept
Modify
Delete



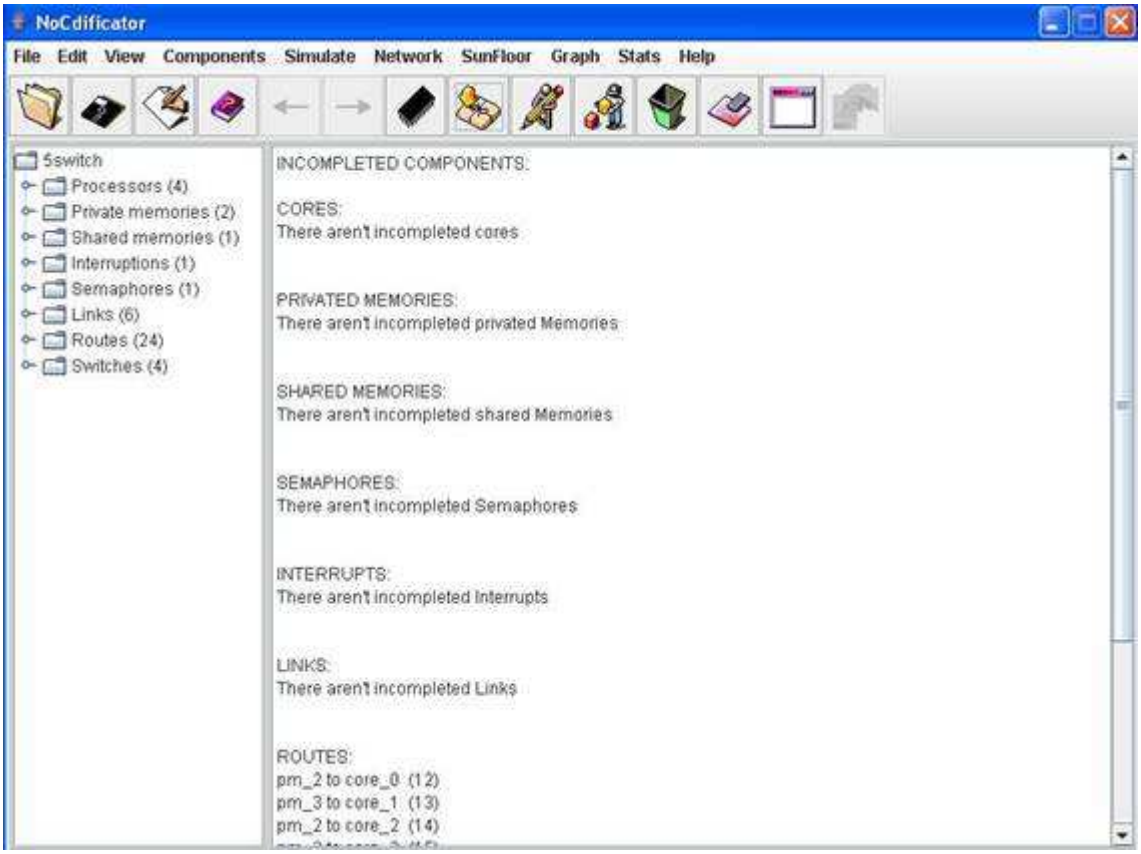
From that new window you can modify or delete the component too.

2.5.4 View Incomplete elements

The net can have incomplete elements that have any of its attributes undefined or wrong defined (like routes that have its switches list wrong connected).

You can see all these incomplete components go to View→Incomplete Components.

These components will be showed in the main window. It can be useful for known what components are incomplete in the net and complete them.



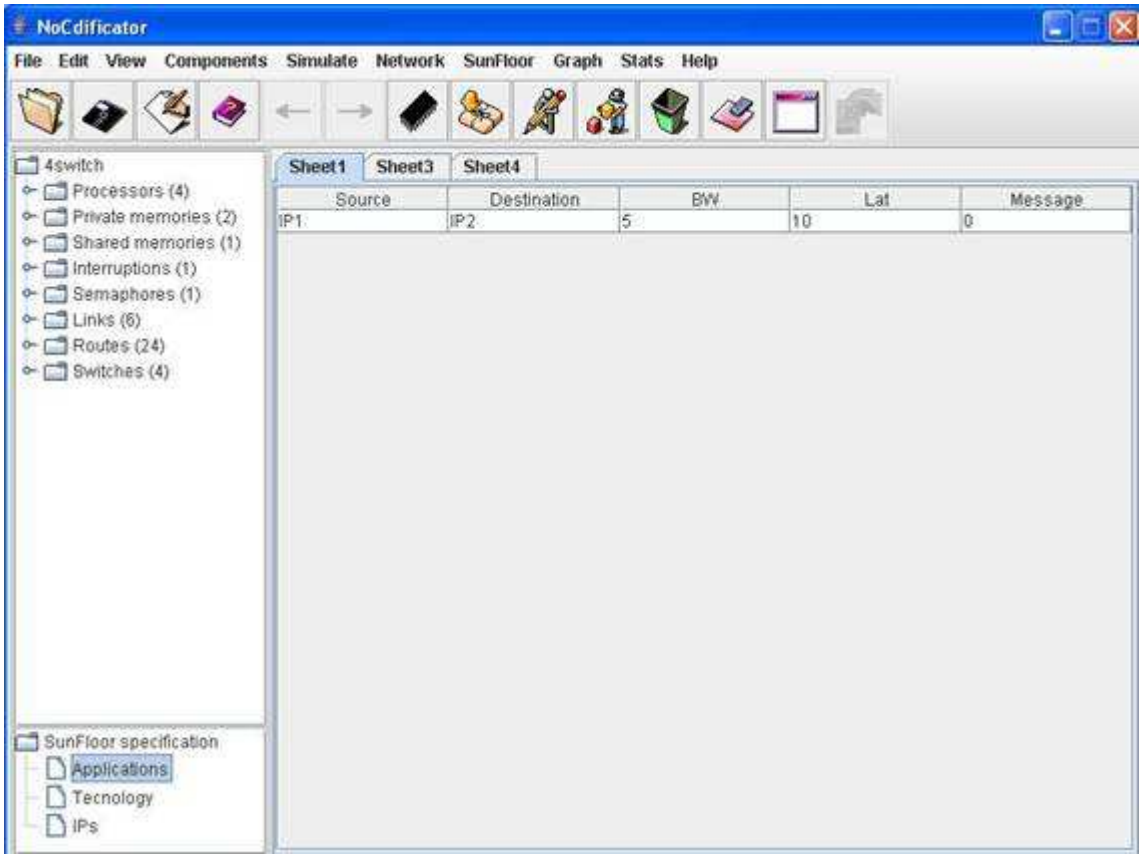
2.5.5 View SunFloor tables

When a net has been opened with SunFloor you can see its SunFloor Tables (application table, technology table and Ips tables).

To see them go to View→SunFloor applications, View→SunFloor technology or View →SunFloor Ips.

In each case you can see each table in the main window.

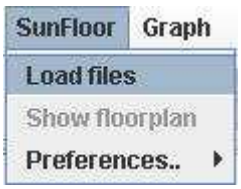
Other way to see them is doing double click on each leaf (applications, technology or IPs) of the SunFloor tree that is in the main window too, above the network's tree.



2.5.6 View the FloorPlan

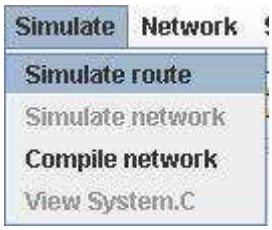
The FloorPlan is a graph generate by SunFloor that shows the structure of the net created. You only can see it if the current network has been generated by SunFloor.

To see this graph go to SunFloor →Show floorplan



2.5.7 Simulate the Routes

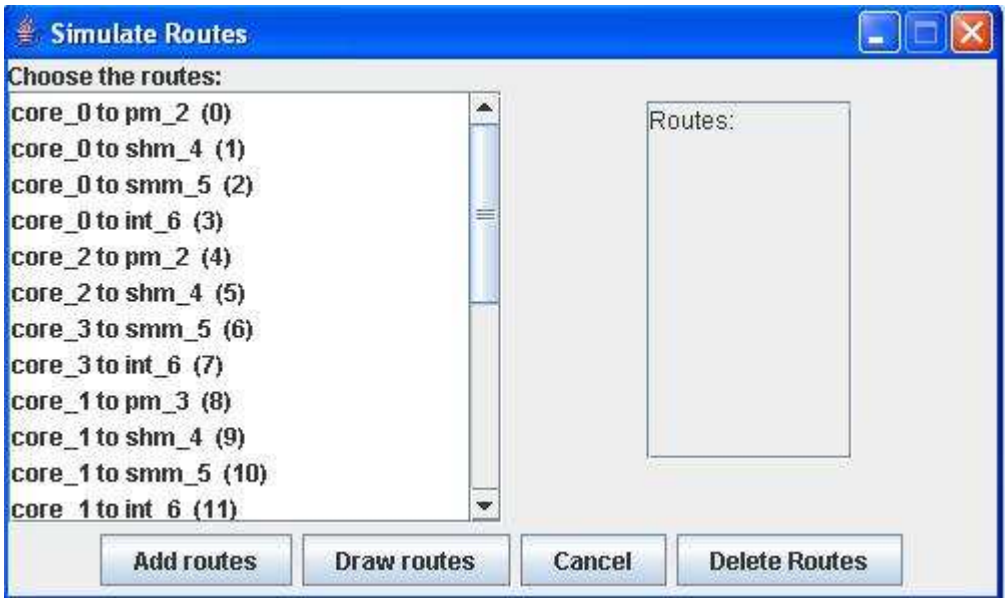
To see the route's paths (with all its switches and its links between them) go to Simulate→Simulate route.



Or press the simulate routes button: 

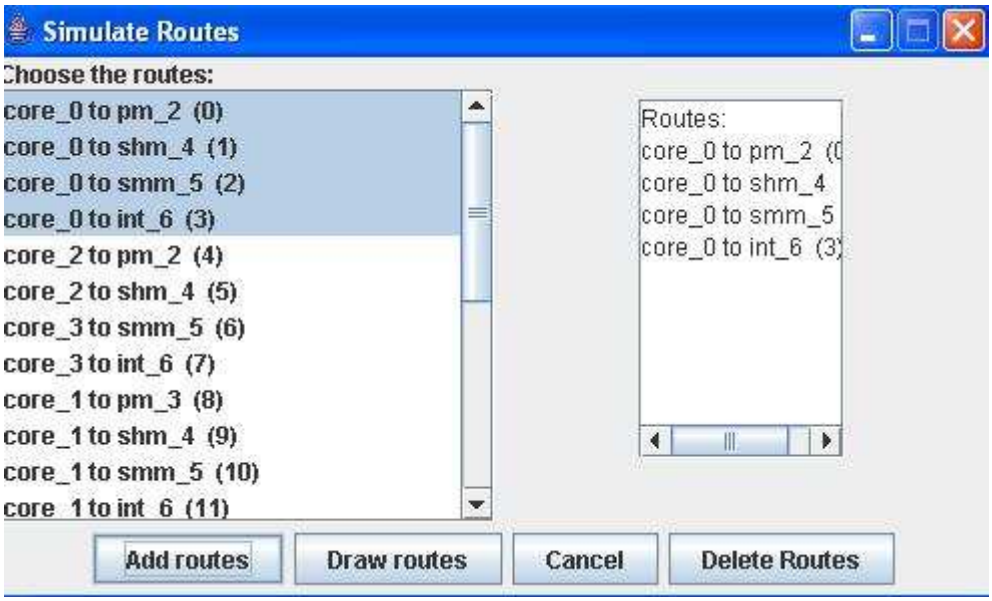
The net must have some routes to simulate them

A new window will appear where you can choose the routes of the net you want simulate following these steps:

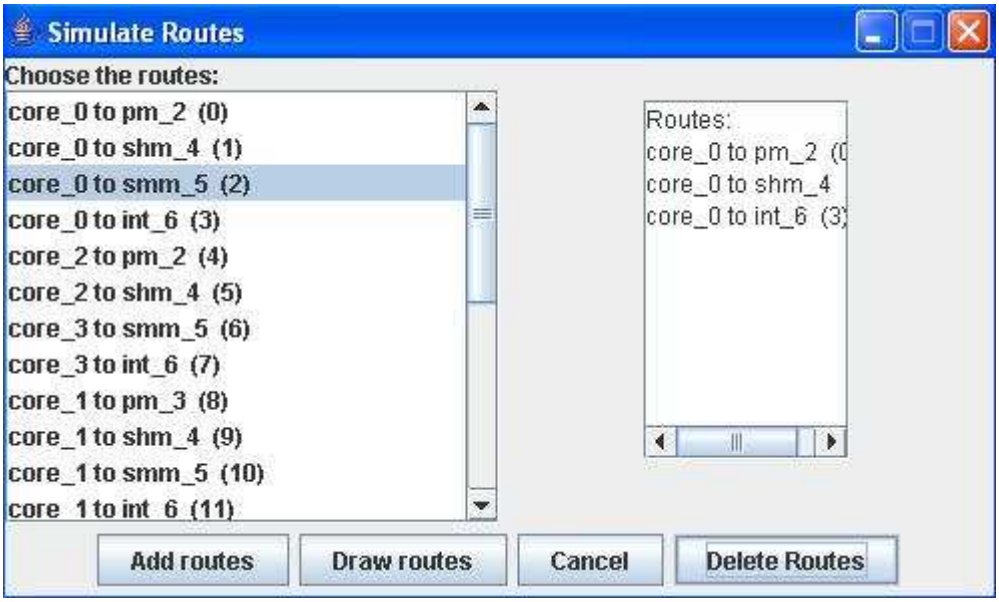


1) First select the routes you want simulate on the left list of the new window and press Add button.

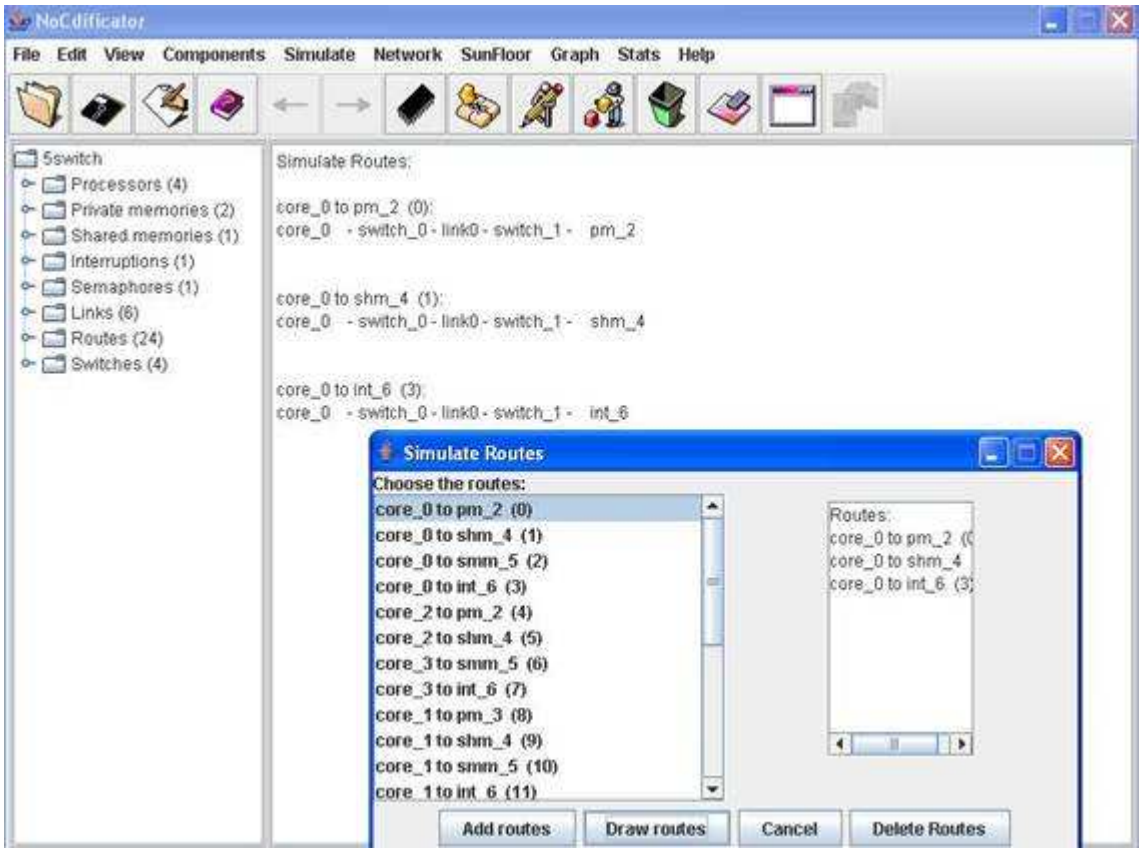
The selected routes will be added into the right list (that is the routes list that will be simulated)



2) You can delete the routes of the right list selecting them into the left list and press Delete button.



3) To simulate the routes of the right list press Draw routes button. All these routes will be simulated into the main window.



4) You can cancel the simulation of the routes pressing Cancel button.

Even the incomplete routes can be simulated. If they have wrong connections between switches of theirs paths the path will be showed without links between these wrong connections.

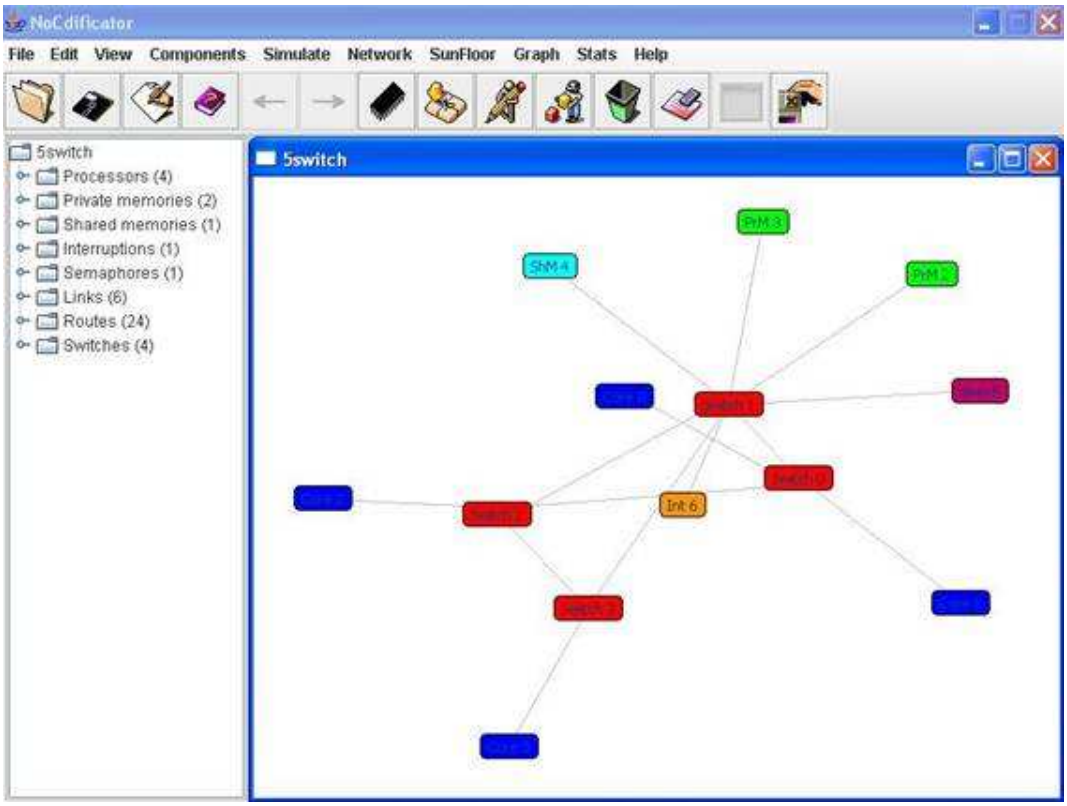
2.5.8 View the network's Graph

To see the network's graph, it has to be a network opened or created.

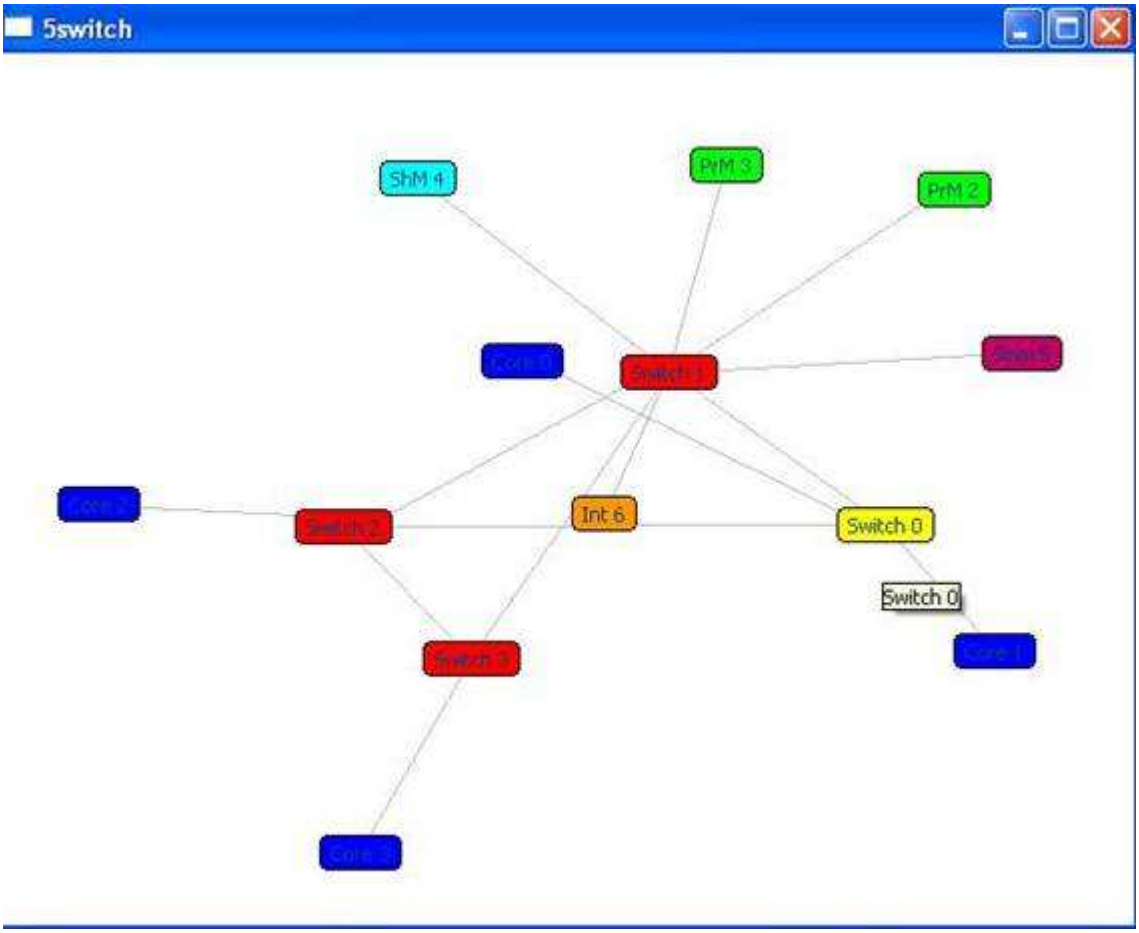
To see it go to Graph →Show Graph.

Or press: 

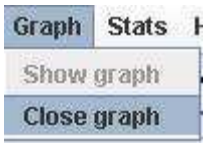
On the graph you can move the nodes and doing double click in those you can see the information about the node and modify or delete it.



The network's graph will be updated when any change in the net happens.



To close the graph window go to Graph→Close Graph



Or press: 

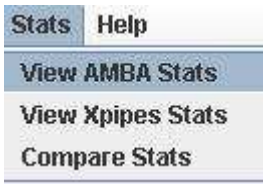
2.5.9 View Xpipes and AMBA Stats

When the net is compiled and simulated Xpipes and AMBA generates statistics files.

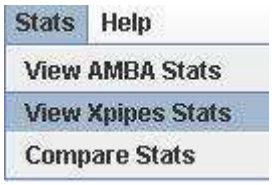
To see them first you have to compile the net and simulate it, if the net hasn't been compiled and simulated before.

If the simulate and compile process have been successful you could see both stats files. And also you can see the comparison between them.

To see the AMBA stats file go to Stats View AMBA Stats.

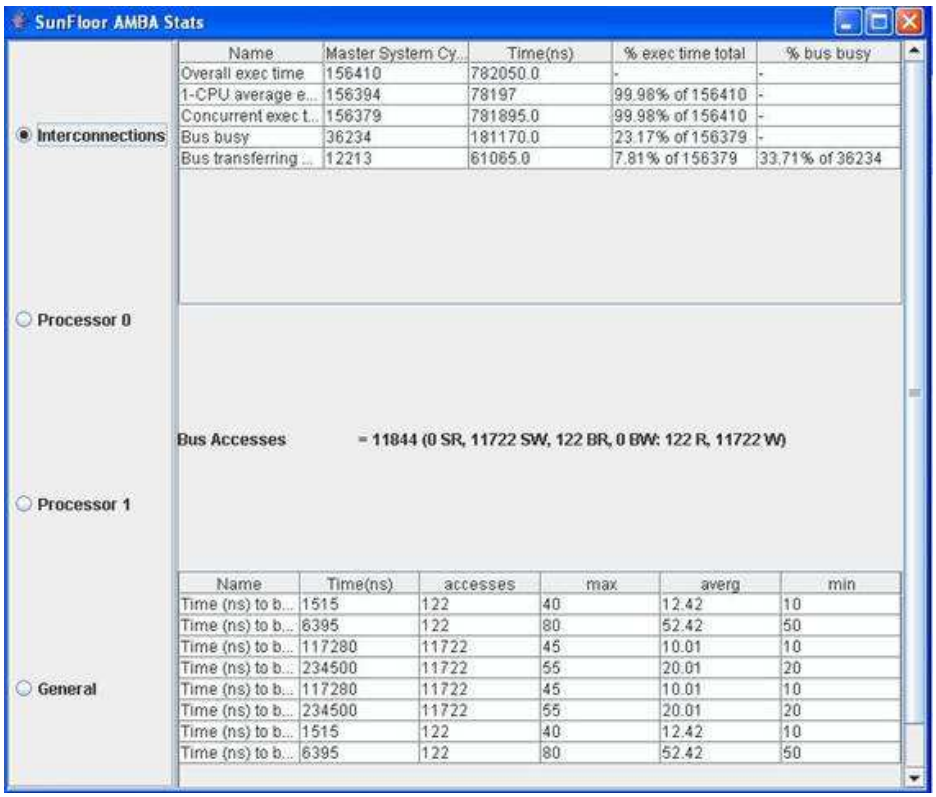


To see the Xpipes stats file go to Stats View Xpipes Stats.



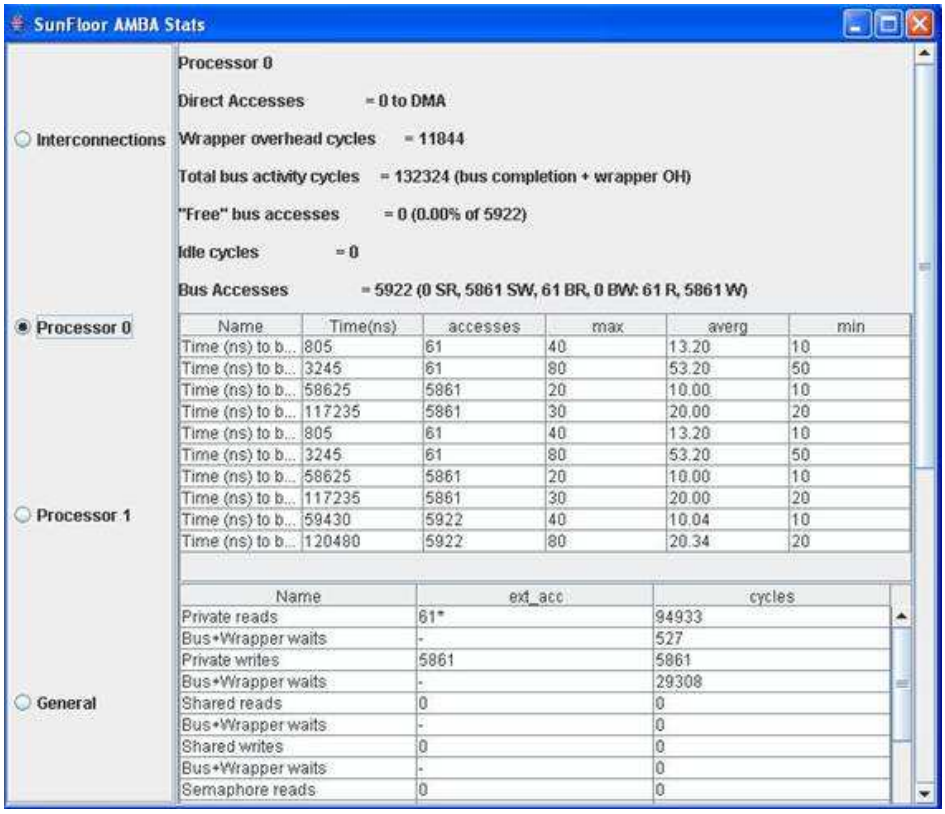
To see the Comparison between them go to Stats→Compare Stats.

If only simulate for AMBA has been successful you only could see the AMBA stats file.



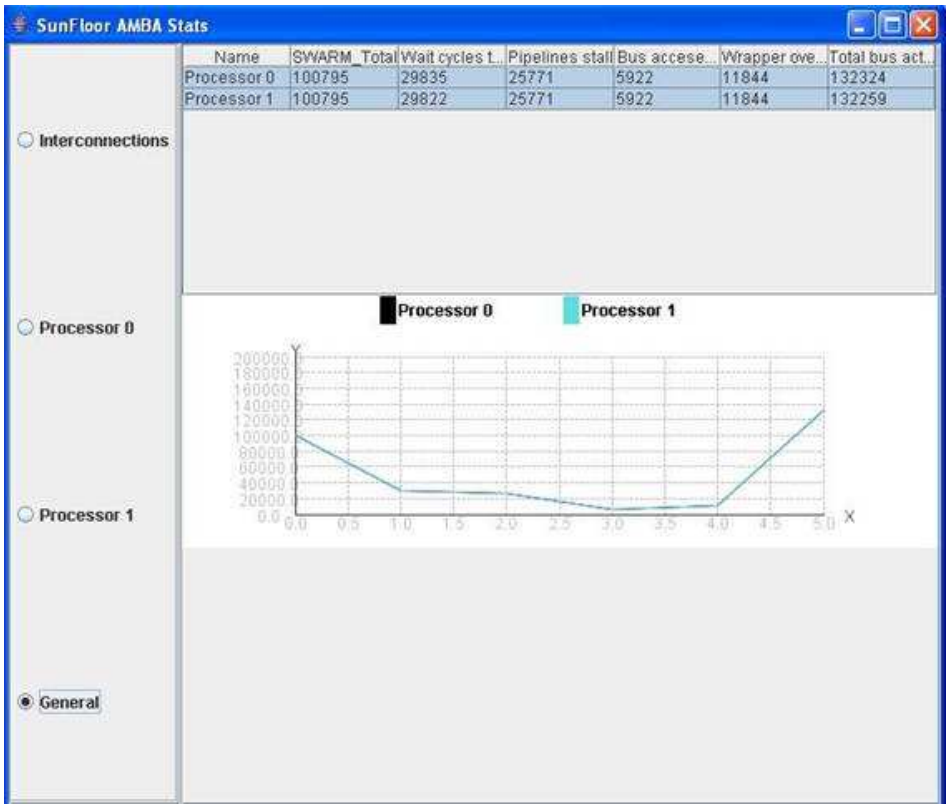
Name	Master System Cy..	Time(ns)	% exec time total	% bus busy
Overall exec time	156410	782050.0	-	-
1-CPU average e...	156394	78197	99.98% of 156410	-
Concurrent exec t...	156379	781895.0	99.98% of 156410	-
Bus busy	36234	181170.0	23.17% of 156379	-
Bus transferring ...	12213	61065.0	7.81% of 156379	33.71% of 36234

Name	Time(ns)	accesses	max	averg	min
Time (ns) to b...	1515	122	40	12.42	10
Time (ns) to b...	6395	122	80	52.42	50
Time (ns) to b...	117280	11722	45	10.01	10
Time (ns) to b...	234500	11722	55	20.01	20
Time (ns) to b...	117280	11722	45	10.01	10
Time (ns) to b...	234500	11722	55	20.01	20
Time (ns) to b...	1515	122	40	12.42	10
Time (ns) to b...	6395	122	80	52.42	50

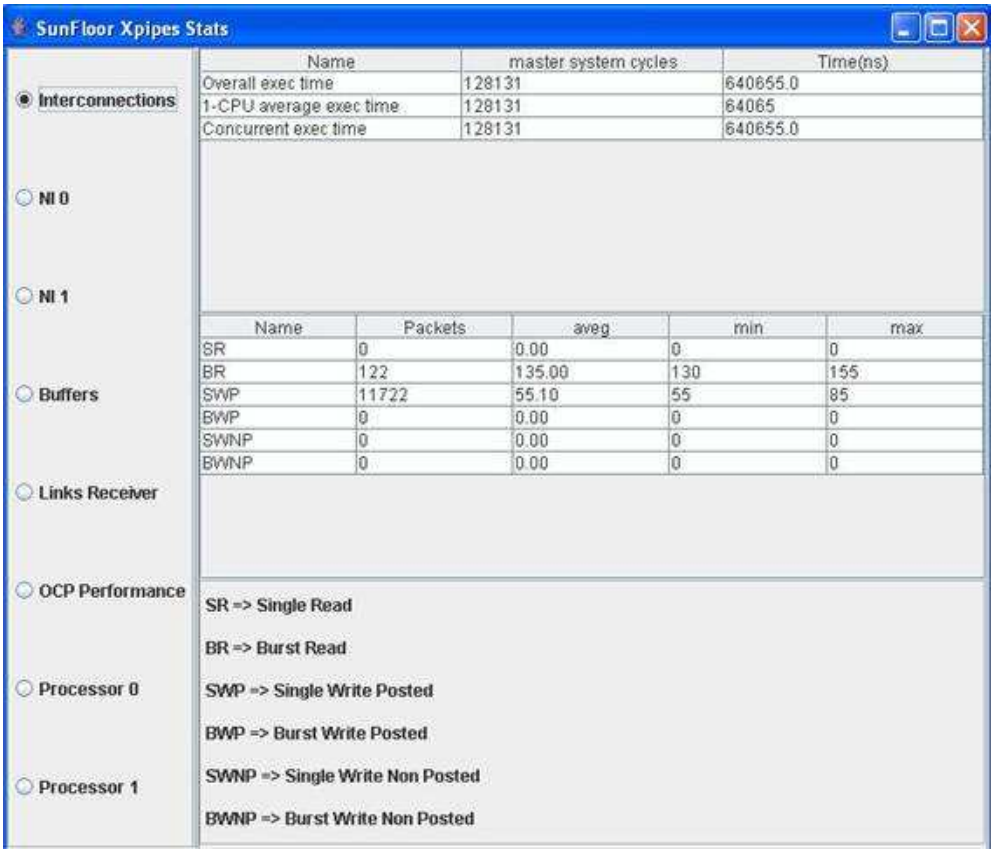


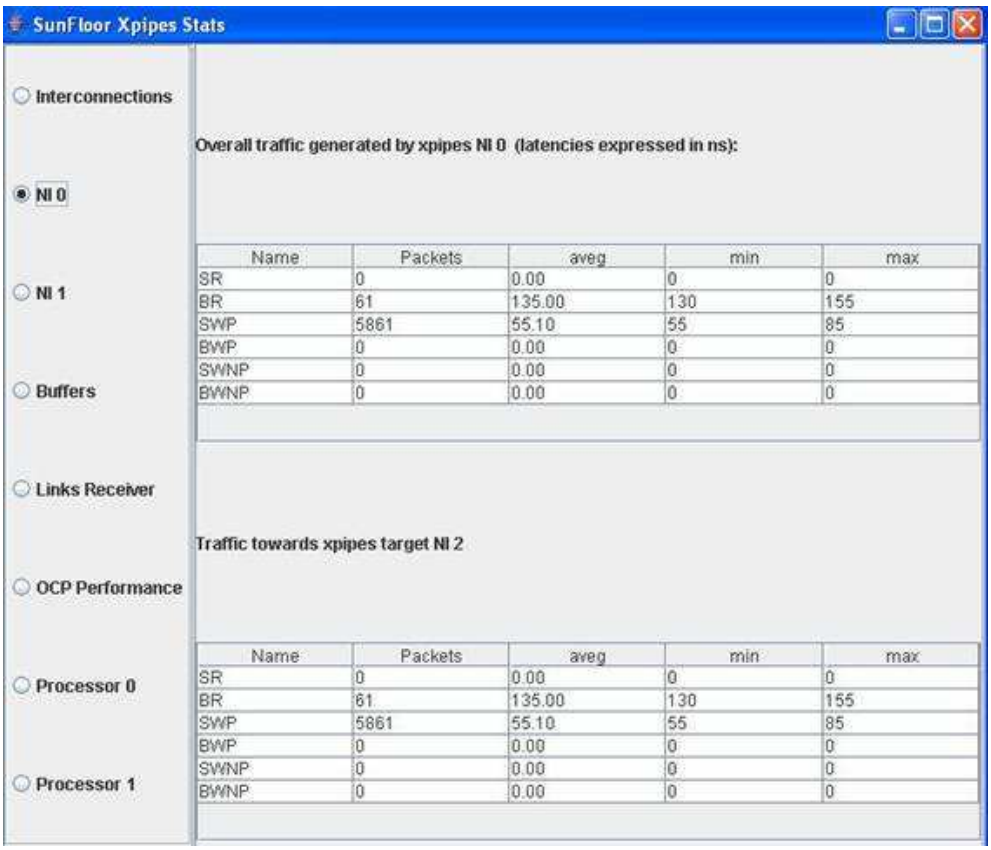
Name	Time(ns)	accesses	max	averg	min
Time (ns) to b...	805	61	40	13.20	10
Time (ns) to b...	3245	61	80	53.20	50
Time (ns) to b...	58625	5861	20	10.00	10
Time (ns) to b...	117235	5861	30	20.00	20
Time (ns) to b...	805	61	40	13.20	10
Time (ns) to b...	3245	61	80	53.20	50
Time (ns) to b...	58625	5861	20	10.00	10
Time (ns) to b...	117235	5861	30	20.00	20
Time (ns) to b...	59430	5922	40	10.04	10
Time (ns) to b...	120480	5922	80	20.34	20

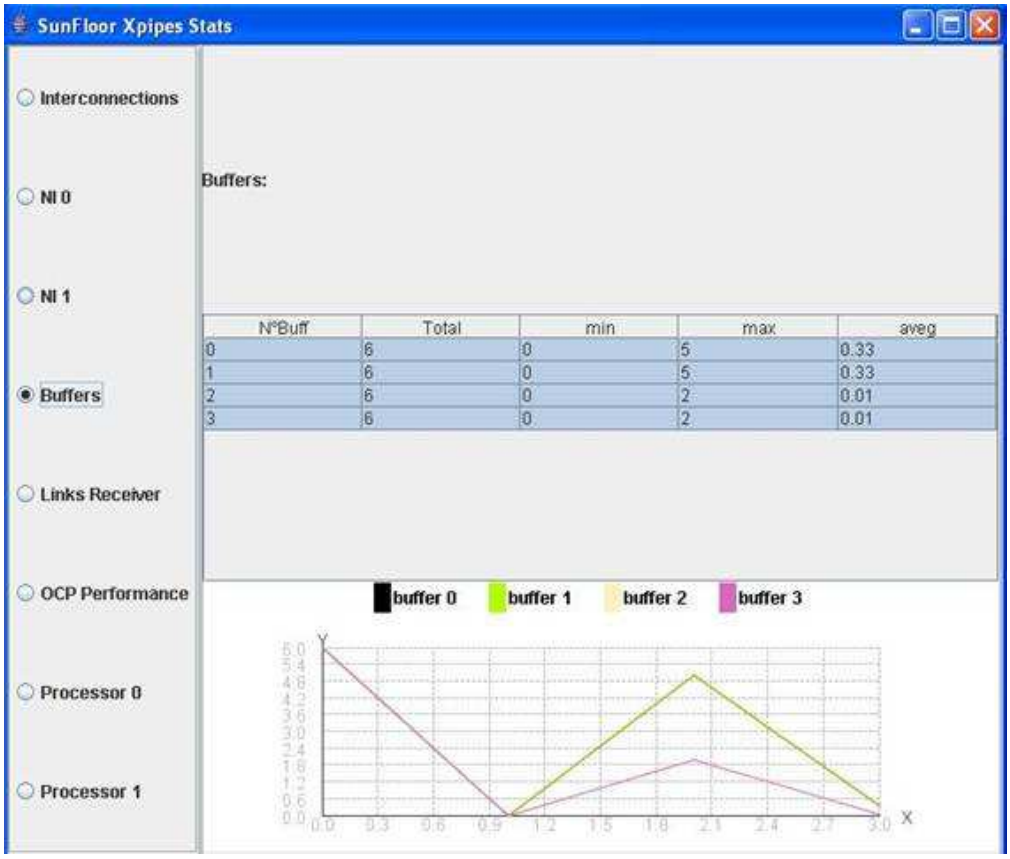
Name	ext_acc	cycles
Private reads	61*	94933
Bus+Wrapper waits	-	527
Private writes	5861	5861
Bus+Wrapper waits	-	29308
Shared reads	0	0
Bus+Wrapper waits	-	0
Shared writes	0	0
Bus+Wrapper waits	-	0
Semaphore reads	0	0

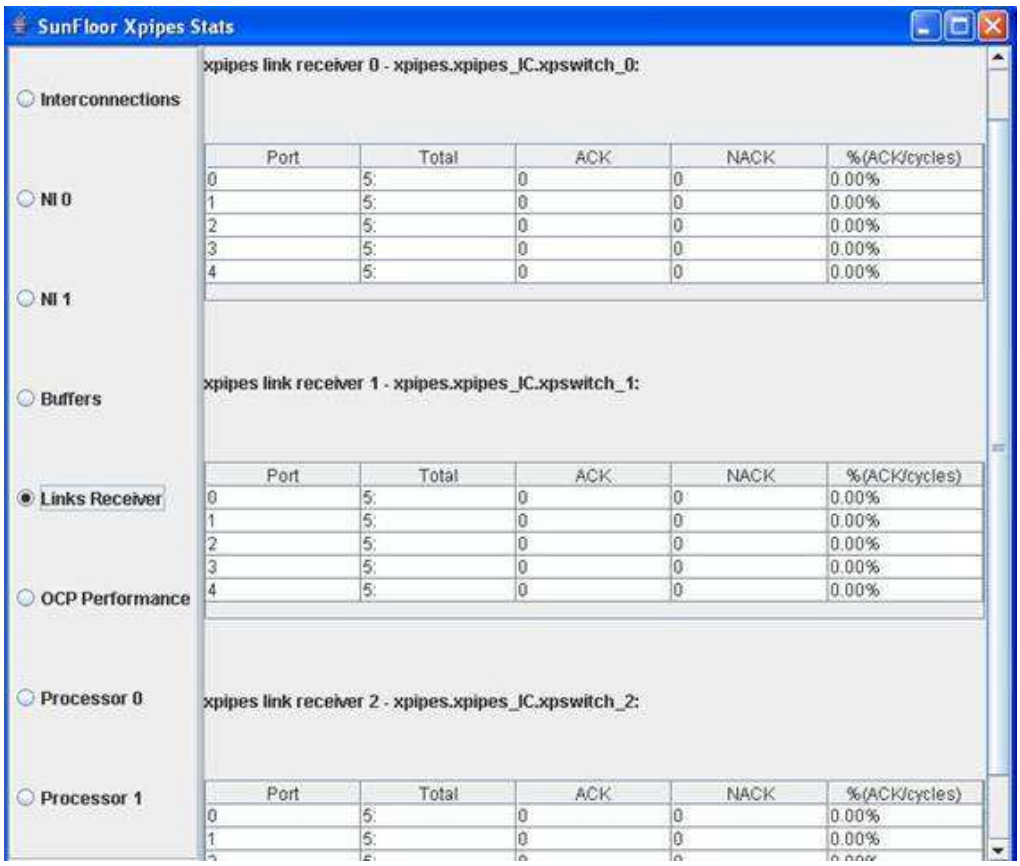


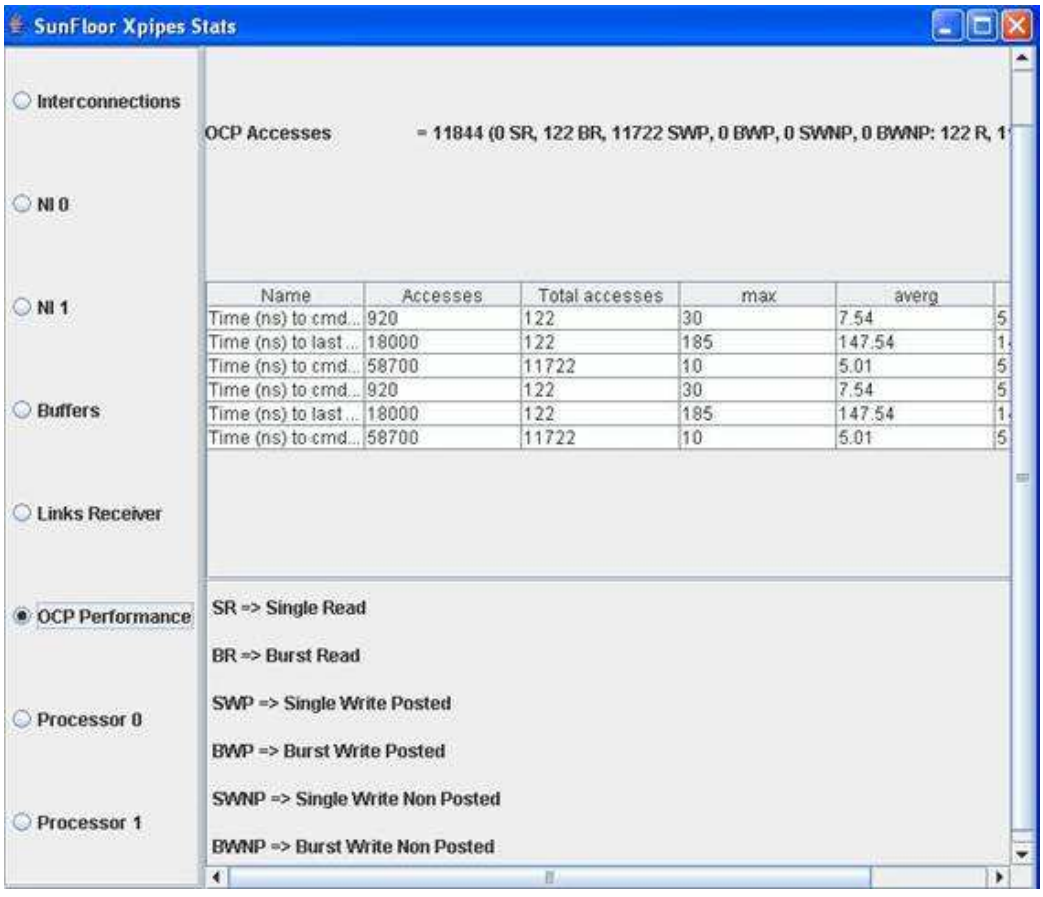
If only simulate for Xpipes has been successful you only could see the Xpipes stats file.

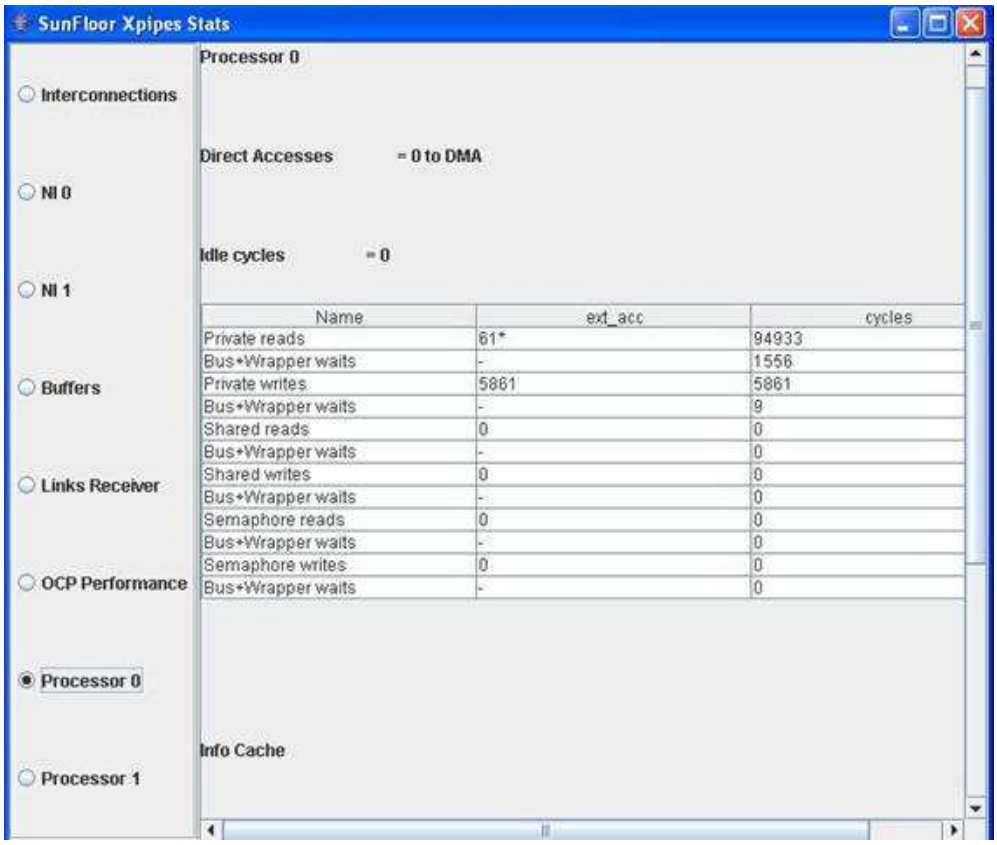



SunFloor Xpipes Stats
Interconnections
NI 0
NI 1
Buffers
Links Receiver
OCP Performance
Processor 0
Processor 1
Overall traffic generated by xpipes NI 0 (latencies expressed in ns):
Name Packets aveg min max
SR 0 0.00 0 0
BR 61 135.00 130 155
SWP 5861 55.10 55 85
BWP 0 0.00 0 0
SWNP 0 0.00 0 0
BWNP 0 0.00 0 0
Traffic towards xpipes target NI 2
Name Packets aveg min max
SR 0 0.00 0 0
BR 61 135.00 130 155
SWP 5861 55.10 55 85
BWP 0 0.00 0 0
SWNP 0 0.00 0 0
BWNP 0 0.00 0 0


SunFloor Xpipes Stats
Interconnections
NI 0
NI 1
Buffers
Links Receiver
OCP Performance
Processor 0
Processor 1
Buffers:
N°Buff Total min max aveg
0 6 0 5 0.33
1 6 0 5 0.33
2 6 0 2 0.01
3 6 0 2 0.01
buffer 0
buffer 1
buffer 2
buffer 3
Y
X

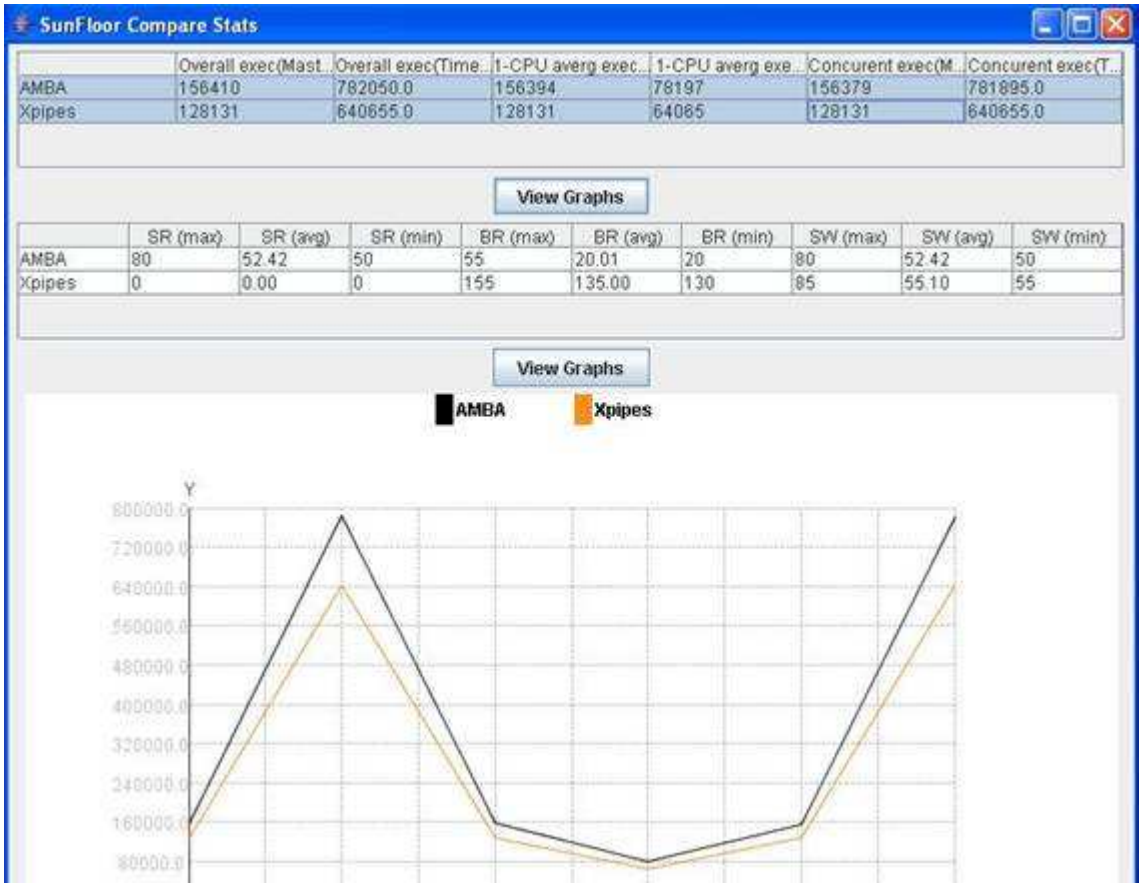

SunFloor Xpipes Stats
Interconnections
NI 0
NI 1
Buffers
Links Receiver
OCP Performance
Processor 0
Processor 1
xpipes link receiver 0 - xpipes.xpipes_IC.xpswitch_0:
Port Total ACK NACK %(ACK/cycles)
0 5: 0 0 0.00%
1 5: 0 0 0.00%
2 5: 0 0 0.00%
3 5: 0 0 0.00%
4 5: 0 0 0.00%
xpipes link receiver 1 - xpipes.xpipes_IC.xpswitch_1:
Port Total ACK NACK %(ACK/cycles)
0 5: 0 0 0.00%
1 5: 0 0 0.00%
2 5: 0 0 0.00%
3 5: 0 0 0.00%
4 5: 0 0 0.00%
xpipes link receiver 2 - xpipes.xpipes_IC.xpswitch_2:
Port Total ACK NACK %(ACK/cycles)
0 5: 0 0 0.00%
1 5: 0 0 0.00%


SunFloor Xpipes Stats
Interconnections
NI 0
NI 1
Buffers
Links Receiver
OCP Performance
Processor 0
Processor 1
OCP Accesses = 11844 (0 SR, 122 BR, 11722 SWP, 0 BWP, 0 SWNP, 0 BWNP: 122 R, 1
Name Accesses Total accesses max averg
Time (ns) to cmd... 920 122 30 7.54
Time (ns) to last ... 18000 122 185 147.54
Time (ns) to cmd... 58700 11722 10 5.01
Time (ns) to cmd... 920 122 30 7.54
Time (ns) to last ... 18000 122 185 147.54
Time (ns) to cmd... 58700 11722 10 5.01
SR => Single Read
BR => Burst Read
SWP => Single Write Posted
BWP => Burst Write Posted
SWNP => Single Write Non Posted
BWNP => Burst Write Non Posted



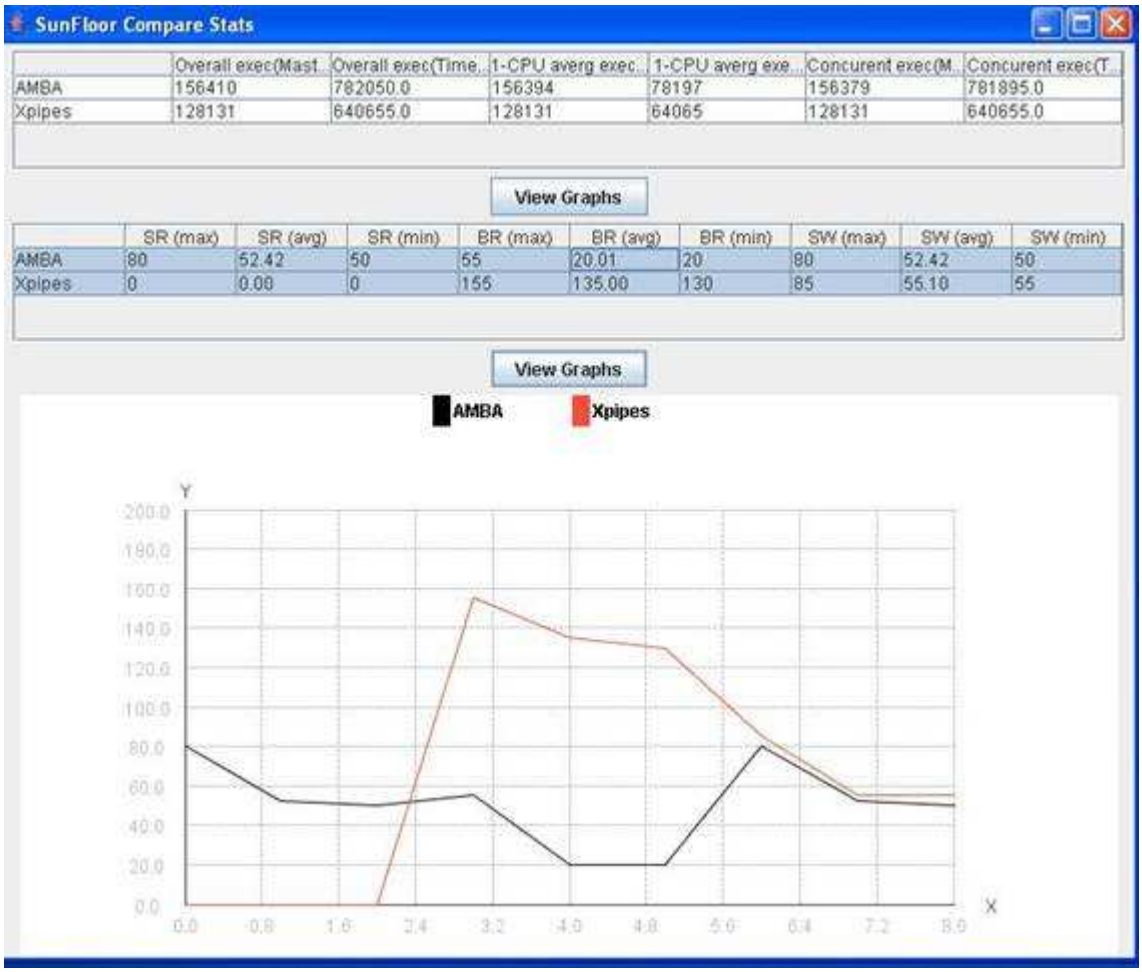
Name	ext_acc	cycles
Private reads	61*	94933
Bus+Wrapper waits	-	1556
Private writes	5861	5861
Bus+Wrapper waits	-	9
Shared reads	0	0
Bus+Wrapper waits	-	0
Shared writes	0	0
Bus+Wrapper waits	-	0
Semaphore reads	0	0
Bus+Wrapper waits	-	0
Semaphore writes	0	0
Bus+Wrapper waits	-	0

In the Stats Comparison you can selected the rows you what compare and press View graph button to see the rows in a graph.



	Overall exec(Mast..	Overall exec(Time..	1-CPU averg exec..	1-CPU averg exe..	Concurent exec(M..	Concurent exec(T..
AMBA	156410	782050.0	156394	78197	156379	781895.0
Xpipes	128131	640655.0	128131	64085	128131	640655.0

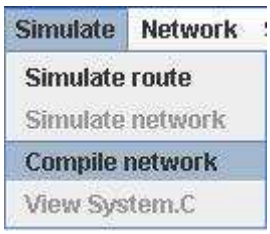
	SR (max)	SR (avg)	SR (min)	BR (max)	BR (avg)	BR (min)	SW (max)	SW (avg)	SW (min)
AMBA	80	52.42	50	55	20.01	20	80	52.42	50
Xpipes	0	0.00	0	155	135.00	130	85	55.10	55



2.6 Compile and Simulate the network

To compile and simulate the net, it is necessary to have an opened or created net in NoCdificador.

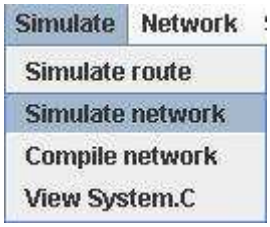
First compile the net, going to Simulate→Compile Network.



The results of the compile will be showed in the main window. You can see if any error has happened compiling the net in the results.

If the net has been compiled successful then you can simulate it. But if it has had error compiling the net you can see them in the main window but you can't simulate it.

To simulate the net go to Simulate→Simulate Network. It asks you for a number of initiators and a type of simulation.



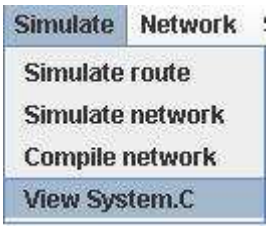
You must give a number of initiators and a type of application:

There are more application types:

This action simulates the net with Xpipes Compiler and AMBA. This process can take a long time, depends on the simulation type.

When the simulation is finished the results are showed in the main window. Even if the simulations haven't successful you can see the errors in the results.

You can see the systemC file generated going to Simulate→View SystemC



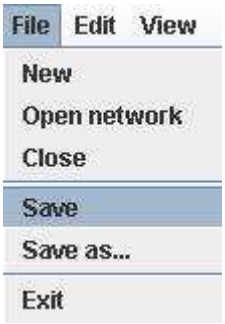
If the one or both simulations have successful you can see the Stats files generated by Xpipes and AMBA too, going to Stats menu.

You can do both actions pressing the Button:  that compile and then simulate the network.

2.7 Save a network

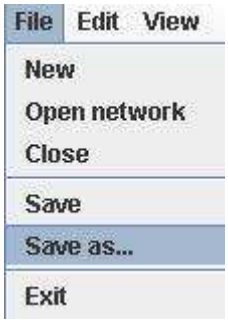
When you want to save the net you can save it with the save name if it has one because of it was opened before, or you can save it with another name.

To save the net with the same name it had go to File→Save. Or press the save Button 



If the net wasn't opened and it hasn't a save name yet, It ask you for a name to save the network txt file.

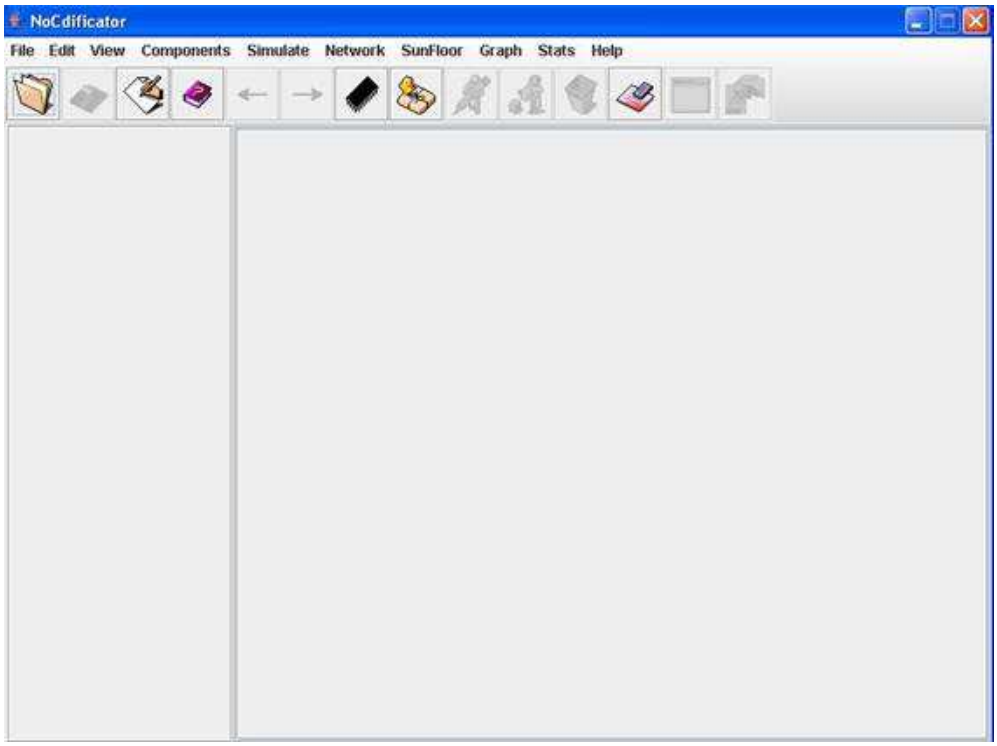
To save the net with another name (not with the name that has when it was opened) go to File→Save as...



2.8 Close a network

When you want to close a network go to File→Close.

If the net isn't saved yet NoCdificador asks you for save it. When the current net is closed its network tree disappears too.



3. DESCRIPTION OF THE CLASSES

NoCdificador is composed by 65 classes, which are divided in three groups:

Parse Classes:

Parse
Parse_Stats
Parse_Stats2

Graphic Classes:

ModifyNoC
Principal

- ***To modify***
  ComponentChooser
  CoreMenu
  CoreMenuMod
  LinkMenu
  SwMenu
  RouteMenu
  SunFloorMenu
  NewNetWindow

- ***To view information***
  ChooseRoutes
  CoreInforWindow
  LinkInforWindow
  RouteInforWindow
  SwInforWindow
  StatsWindow
  StatsWindow2
  StatsWindowCompare
  Grafo
  Help
  Principal_AboutBox

Logic Classes:

- ***For the net and its structure***

  Network
  Component
  Core
  PrivateMemory
  Semaphore
  SharedMemory
  Switches
  Route
  Interrupt
  Link
  Branch
  Tree
  TreeSunfloor
  ComponentPositionPair
  Save

- ***To undo and redo modifications***

  Modification
  Change

- ***For parse classes***

  TXTFilter
  ParseReturnPar
  XMLFilter

- ***For the tables of the program***

  Datos
  LinkReceiverElement
  ModeloTabla
  ModeloTablaStats
  ModeloTablaStats2
  NI_Element
  ObjetoTabla0
  ObjetoTabla1
  ObjetoTabla2
  ObjetoTabla3
  ObjetoTabla4
  ObjetoTabla5
  ObjetoTabla6
  ObjetoTabla7
  processor_stats
  processor_stats2
  SWARM_processor

- ***To syncronize the threads***

  Consumidor
  Productor
  Tuberia

ModifyNoC is the main class of the project and Principal is the interface of it. Now, we are going to detail each class:

ModifyNoC

This is the main class; its function is to create the main window.

Attributes:

Principal p

Methods:

Main: creates a new Principal object

Principal

This is the class that represents the main window. This class takes the control of the user actions and enable or disable buttons and options of the Menu Bar.

Attributes:

Int startOption
JPane contentPane
JMenuBar jMenuBar1
JPanel jPanel1
JPanel jPanel2
Int numSF
Help help
SunFloorMenu sunNuevo;
Parse parse
Save save
Network net
String saveName
ComponentChooser chCompo
Vector drawRoutes
Thread my_thread
Grafo g
Parse_Stats pa_Stats
Parse_Stats2 pa_Stats2
Tree jTree1
TreeSunfloor jTreeSunfloor
int xcoOK
int cplOKwithAMBA
int cplOKwithXpipes
int sunfloorOK
boolean isYetSave

Methods:

Main: creates a new Principal object

Branch

This class represents a branch of the net tree.

Attributes:

DefaultMutableTreeNode n

Methods:

Branch(Vector facts)

Change

This class represents one or more modifications in the net.

Attributes:

Vector whatModifications

Methods:

Change()

Vector getModifications()

ChooseRoutes

Window to choose the routes to be simulated.

Attributes:

Principal p

Methods:

ChooseRoutes(Principal p)

boolean esta(String s):

Returns true if the selected route is in the right list, else returns false.

addButton_actionPerformed(ActionEvent e):

This function adds a new selected route from the left list in the right list, if the selected route or routes are in the right list, they won't be added again.

drawButton_actionPerformed(ActionEvent e):

This function draws the routes of the right list into the main window. Routes are drawed with all its switches and links.

cancelButton_actionPerformed(ActionEvent e):

This function removes all the routes of the right list and closes the window.

deleteButton_actionPerformed(ActionEvent e):

This function deletes the selected route of the left list in the right list, if the selected route or routes aren't in the right list, they won't be deleted again.

deleteAllButton_actionPerformed(ActionEvent e):

This function removes all the routes of the right list.

Component

This class represents any network's element

Attributes:

String name
String prefix
String componentTypeName
boolean incompleted

Methods:

Component()

setName(String s)

String getName()

boolean getIncompleted()

setIncompleted(boolean b)

Grafo

Show the net with a graph window.

Attributes:

List selection
Principal p
Network n

Methods:

Grafo(Principal p_aux, Network n_aux):

Create the graph from the network

run()

chooseElement(List lista):

Open the correspondent information window of the element selected.

ComponentChooser

This class is a window to choose a type of network's element

Attributes:

Network r
Principal p

Methods:

ComponentChooser(int actionType,Principal p):

> Action type is 0 to create a new component, 1 to modify an existing component or 2 to delete an existing component.

acceptButton_actionPerformed(ActionEvent e):

> Open the correspondent window for each action type, where coreType is 0 for a core, 1 for a pm, 2 for a shm, 3 for a smm, 4 for an interrupt.
>
> Cases:
> - New core, pm, shm, smm, int => new CoreMenu(coreType,p);
> - Modify core, pm, shm, smm, int => new CoreMenuMod(coreType,1,-1,p);
> - Delete core, pm, shm, smm, int => new CoreMenuMod(coreType,2,-1,p);
> - New switch => new SwMenu(0,-1,p);
> - Modify switch => new SwMenu(1,-1,p);
> - Delete switch => new SwMenu(2,-1,p);
> - New link => new LinkMenu(0,-1,p);
> - Modify link => new LinkMenu(1,-1,p);
> - Delete link => new LinkMenu(2,-1,p);
> - New route => new RouteMenu(0,-1,p);
> - Modify route => new RouteMenu(1,-1,p);
> - Delete route => new RouteMenu(2,-1,p);

cancelButton_actionPerformed(ActionEvent e):

> Close the window

ShowWarnningMessage() :

> If there is any problem, show the warning message

ComponentPositionPair

This class is used to look for components in the net.

Attributes:

int position
Component object

Methods:

ComponentPositionPair()

setPosition(int i)

int getPosition()

setComponent(Component o)

Component getComponent()

Consumidor

Attributes:

Tuberia tuberia

Methods:

Consumidor(Tuberia t)

void run()

Core

It's a processor of the network

Attributes:

String sw
double divFrec
int numBuf
String arg5
String arg6
String corePrefix
boolean bi
int inputBufDepth

Methods:

Core(String i1, String i2, double d3, int i4,String a5, String a6,int inputBufDepth)

Core(String i1, String i2, double d3, int i4,String a5, String a6,boolean bidir,int inputBufDepth)

setSwitch(String i)
String getSwitch()

setDivFrec(double i)
double getDivFrec()

setNumBuf(int i)
int getNumBuf()

setArg5(String s)
String getArg5()

setArg6(String s)
String getArg6()

setIncompleted(boolean b)
boolean getIncompleted()

boolean isBi()
setBi(boolean bi)

int getInputBufDepth()
setInputBufDepth(int inputBufDepth)

CoreInforWindow

Show the properties of a core modify it or delete it.

Attributes:

Network r;
Principal p;

Methods:

CoreInforWindow(int type,String nom,Principal p)

Vector giveMeRoutes(int coeNumber, int type):

Returns a Vector of routes in which the core appears.

acceptButton_actionPerformed(ActionEvent e):

Close the window.

modifyButton_actionPerformed(ActionEvent e):

Close the window and show a new window to modify the core => new CoreMenuMod(typeC,1,number,p,this);

deleteButton_actionPerformed(ActionEvent e):

Close the window and show a new window to delete the core =>new CoreMenuMod(typeC,2,number,p,this);

CoreMenu

It is a window to create a core, private memory, shared memory, semaphore, or interrupt.

Attributes:

Network r;
Principal p;

Methods:

CoreMenu(int n,Principal p) :

Where coreType is: 0->Core, 1->PrivMem, 2->ShaMem, 3->Semap, 4->Int.

cancelButton_actionPerformed(ActionEvent e):

Close the window.

acceptButton_actionPerformed(ActionEvent e):

Make sure that all the attributes are correct and create the new element.

int value(char[] a):

Returns the int value of a char array of digits

double valorDouble(char[] a) :

Returns the double value of a char array of digits

CoreMenuMod

It is the window to modify or delete a core, pm, shm, smm or int.

Attributes:

Network r;
Principal p;

Methods:

CoreMenuMod(int n, int action, int direct,Principal p):

> Where coreType is: 0->Core, 1->PrivMem, 2->ShaMem, 3->Semap, 4->Int; actionType is: 1->modify or 2->delete; directOption is -1 if anyone is modified,if not, the number of the component to be modified (clicking)

cancelButton_actionPerformed(ActionEvent e):

> Close the window.

acceptButton_actionPerformed(ActionEvent e):

> Make sure all the attributes are correct and modify or delete the new element.

double doubleValue(char[] a) :

> Returns the double value of a char array of digits.

int value(char[] a):

> Returns the int value of a char array of digits

properties():

> Show the properties of the element selected

Datos

This is the structure of each row of the SunFloor tables.

Attributes:

String s1
String s2
String s3
String s4
String s5

Methods:

Datos(String s1,String s2, String s3, String s4, String s5)

Datos(String s1,String s2, String s3)

String get1()

String get2()

String get3()

String get4()

String get5()

set1(String s)

set2(String s)

set3(String s)

set4(String s)

set5(String s)

Help

Shows an html manual to help the user.

Attributes:

long serialVersionUID
JEditorPane panel

Methods:

Help()

change(URL url):

It changes the webpage. This will be invoked when a link is clicked.

addHyperlinkListener(HyperlinkListener listener):

Add a hyperlink to attend the events of click in a link.

Modification

This class represents modifications in any element of the net.

Attributes:

Network r
int actionType : (0->create, 1->modify, 2->delete)
int list
int pos
Component componentModify

Methods:

Modification()

Interrupt

This class represents an interrupt element in the net.

Attributes:

String sw
double divFrec
int numBuf
String arg5
final String preArg6
String posMem
boolean fixed
String CorePrefix
boolean bi
int inputBufDepth

Methods:

Interrupt(String i1, String i2, double d3, int i4,String a5,String i6, boolean b7,int inputBufDepth)

Interrupt(String i1, String i2, double d3, int i4,String a5,String i6, boolean b7,boolean bidir,int inputBufDepth)

setPosMem(String i)
String getPosMem()
setFixed(boolean b)
boolean getFixed()
setSwitch(String i)
String getSwitch()
setDivFrec(double i)
double getDivFrec()
setNumBuf(int i)
int getNumBuf()
setArg5(String s)
String getArg5()
setIncompleted(boolean b)
boolean getIncompleted()
boolean isBi()
setBi(boolean bi)
int getInputBufDepth()
setInputBufDepth(int inputBufDepth)

Link

This class represents a link element of the network.

Attributes:

String source
String target
boolean bidirectional
int numRepeaters

Methods:

Link(String i1, String i2, String i3, int i4)

Link(String i1, String i2, String i3,boolean bidir,int i4)

setSource(String i)
String getSource()

setTarget(String i)
String getTarget()

setIncompleted(boolean b)
boolean getIncompleted()

setBidir(boolean b)
boolean getBidir()

int getNumRepeaters()
setNumRepeaters(int numRepeaters)

LinkInforWindow

Shows the properties of a link modify or delete it.

Attributes:

Network r
Principal p

Methods:

LinkInforWindow(String nom,Principal p)

String[] readOnlySwitchName(String n) :

Returns the switch name, and the subchannel in the second position.

Vector giveMeRoutes(int n):

Returns a Vector of routes in which the core appears.

acceptButton_actionPerformed(ActionEvent e):

Close the window.

modifyButton_actionPerformed(ActionEvent e):

Close the window and show a new window to modify the link => new LinkMenu(1,number,p,this);

deleteButton_actionPerformed(ActionEvent e):

Close the window and show a new window to delete the link => new LinkMenu(2,number,p,this);

LinkMenu

Window that Create modify or delete a link.

Attributes:

Principal p
Network r

Methods:

LinkMenu(int n, int direct,Principal p)

CancelButton_actionPerformed(ActionEvent e):

Close the window.

acceptButton_actionPerformed(ActionEvent e):

Make sure all the attributes are correct and modify or delete the new element.

int value(char[] a):

Returns the int value of a char array of digits.

properties():

Show the properties of the element selected.

LinkReceiverElement

It is a LinkReceiver element of Xpipes Stats.

Attributes:

Vector tab1
String label

Methods:

LinkReceiverElement(String l, Vector tab1)

String getLabel()
void setLabel(String l)
Vector getTab1()
void setTab1(Vector tab1)

ModeloTabla

It represents the structure of the SunFloor Table.

Attributes:

int tipo
LinkedList datos
LinkedList listeners

Methods:

setTipo(int i)
int getTipo():

int getColumnCount():Returns the number of columns in the table

setDatos(LinkedList list) : Insert the dates in the table
LinkedList getDatos(): Returns the dates in the table

Object getValueAt(int rowIndex, int columnIndex):
Returns the value for the cell at <code>columnIndex</code> and <code>rowIndex</code>.

deleteRow (int r): Remove the row r of the table.

addDato(Datos newRow): Add a new row r into the table.

addTableModelListener(TableModelListener l): Add a listener in the table.

Class getColumnClass(int columnIndex):

Returns he class of the elements of the column c

String getColumnName(int columnIndex):
Returns the name of the column at <code>columnIndex</code>.

boolean isCellEditable(int rowIndex, int columnIndex)

removeTableModelListener(TableModelListener l)

setValueAt(Object aValue, int rowIndex, int columnIndex)

avisaSuscriptores (TableModelEvent evento)

ModeloTablaStats

It represents the structure of the AMBA Stats Table

Attributes:

int tipo
LinkedList datos
LinkedList listeners

Methods:

setTipo(int i)
int getTipo():

int getColumnCount():Returns the number of columns in the table

setDatos(LinkedList list) : Insert the dates in the table
LinkedList getDatos(): Returns the dates in the table

Object getValueAt(int rowIndex, int columnIndex):
Returns the value for the cell at <code>columnIndex</code> and <code>rowIndex</code>.

deleteRow (int r): Remove the row r of the table.

addDato(ObjetoTabla1 newRow)
addDato(ObjetoTabla2 newRow)
addDato(ObjetoTabla3 newRow)
addDato(ObjetoTabla4 newRow)
addDato(ObjetoTabla7 newRow)

addTableModelListener(TableModelListener l): Add a listener in the table.

Class getColumnClass(int columnIndex):
Returns he class of the elements of the column c

String getColumnName(int columnIndex):
Returns the name of the column at <code>columnIndex</code>.

boolean isCellEditable(int rowIndex, int columnIndex)

removeTableModelListener(TableModelListener l)

setValueAt(Object aValue, int rowIndex, int columnIndex)

avisaSuscriptores (TableModelEvent evento)

ModeloTablaStats2

It represents the structure of the Xpipes Stats Table

Attributes:

int tipo
LinkedList datos
LinkedList listeners

Methods:

setTipo(int i)
int getTipo():

int getColumnCount():Returns the number of columns in the table

setDatos(LinkedList list) : Insert the dates in the table
LinkedList getDatos(): Returns the dates in the table

Object getValueAt(int rowIndex, int columnIndex)

deleteRow (int r): Remove the row r of the table.

addDato(ObjetoTabla0 newRow)
addDato(ObjetoTabla1 newRow)
addDato(ObjetoTabla2 newRow)
addDato(ObjetoTabla3 newRow)
addDato(ObjetoTabla5 newRow)
addDato(ObjetoTabla6 newRow)

addTableModelListener(TableModelListener l): Add a listener in the table.

Class getColumnClass(int columnIndex):
Returns he class of the elements of the column c

String getColumnName(int columnIndex):
Returns the name of the column at <code>columnIndex</code>.

boolean isCellEditable(int rowIndex, int columnIndex)

removeTableModelListener(TableModelListener l)

setValueAt(Object aValue, int rowIndex, int columnIndex)

avisaSuscriptores (TableModelEvent evento)

Network

This class is the net. The net has all the component lists and can modify them.

Attributes:

String name
int topologyType; 0->OTHERS , 1->MESH , 2->TORUS
Change[] undo
int undoSize
Change[] redo
int redoSize
int numCreatedRoutes
Vector coresList
Vector privateMemsList
Vector sharedMemsList
Vector semaphoresList
Vector interruptsList
Vector switchesList
Vector linksList
Vector routesList

Methods:

addSwitchtoCore(int coreType,String CoreName,String swName)
conneSwitchesNet()
createNet(int wide,int high,int type,int inputs,int outputs, int buffers)
Network deleteNet()
deleteUncompletedCores()
deleteUncompletedInterrupts()
deleteUncompletedLinks()
deleteUncompletedPMem()
deleteUncompletedRoutes()
deleteUncompletedSemaphores()
deleteUncompletedSMem()
deleteUncompletedSwitches()
emptyRedo()
Modification executeModify(int i1, int i2, int i3, Component c4)
Boolean exist(String n,int list)
Core extractFromCoreList(int i)
Interrupt extractFromInterruptList(int i)
Link extractFromLinksList(int i)
PrivateMemory extractFromPrivateMemoryList(int i)
Route extractFromRoutesList(int i)
Semaphore extractFromSemaphoresList(int i)
SharedMemory extractFromSharedMemoryList(int i)
Switches extractFromSwitchesList(int i)
Change extractRedo()
Change extractUndo()
ComponentPositionPair findElement(String name, int list)
Modification getInverse(int i1, int i2, int i3, Component c4)
String getName()
setName(String s)

setTopologyType(int i)
int getTopologyType()
setNumCreatedRoutes(int c)
int getNumCreatedRoutes()
static Network getNet()
static Network getNet(String nam, int type)
boolean insertRedo(Change m)
boolean insertUndo(Change m)
boolean modifyCore(int actionType,int Pos,String i1, String i2, double d3, int i4,String a5, String a6,boolean bidir,int inBuf, Change c)

boolean modifyInt(int actionType,int Pos,String i1,String i2, double d3, int i4, String a5,String i6,boolean b7,boolean bidir,int inBuf, Change c)

boolean modifyLink(int actionType,int Pos,String i1, String i2, String i3,boolean isbidir,int inBuf,Change c)

boolean modifyNet(String n)

boolean modifyPrivMem(int actionType,int Pos,String i1, String i2, double d3, int i4, String a5,String i6,boolean b7,boolean bidir,int inBuf,Change c)

boolean modifyRoute(int actionType,int Pos,String i1, String i2, int sType, int dType, Vector v3,String name)

boolean modifyShMem(int actionType,int Pos,String i1, String i2, double d3, int i4, String a5,String i6,boolean b7,boolean bidir,int inBuf,Change c)

boolean modifySmm(int actionType,int Pos,String i1, String i2, double d3, int i4, String a5,String i6,boolean b7,boolean bidir,int inBuf, Change c)

boolean modifySw(int actionType,int Pos,String i1, int i2, int i3, int i4,int inBuf,Vector Channels, Change c)

boolean modifySw(int actionType,int Pos,String i1, int i2, int i3, int i4,int X, int Y, int inBuf,Vector Channels,Change c)

boolean routeCompleted(int i)
boolean hasSubChannel(String swName, String subcha)
String linkBetween(String a, String b)

NewNetWindow

This is the window to create a new net.

Attributes:

Principal pr
Network net

Methods:

NewNetWindow(Principal p)

ShowTextFields():

Depends on the topology type It will be necessary to fill more or less gap.

int value(char[] a):

Returns the int value of a char array of digits.

jButton1_actionPerformed(ActionEvent e):

Accept Button: Check the correct attributes and creates the new net.

jButton2_actionPerformed(ActionEvent e):

Close the window.

jComboBox1_actionPerformed(ActionEvent e):

Show the correct gaps when the selected element changes.

NI_Element

This represents a network interface element of Xpipes Stats

Attributes:

Vector tabGenBy
String labelGen
Vector labelTowards
Vector TowardsTabs

Methods:

NI_Element(String l, Vector tab1, Vector labelsT, Vector tabs)

String getLabelGen()
setLabelGen(String l)

Vector getTabGen()
setTabGen(Vector tab1)

Vector getLabelTowards()
setLabelTowards(Vector labelTowards)

Vector getTowardsTabs()
setTowardsTabs(Vector towardsTabs)

ObjetoTabla0

This is the structure of a row of the Interconnect AMBA

Attributes:

String name
String aux1
String aux2
String aux3
String aux4

Methods:

ObjetoTabla0(String s1, String s2, String s3, String s4, String s5)

String getAux1()
setAux1(String aux1)

String getAux2
setAux2(String aux2)

String getAux3()
setAux3(String aux3)

String getAux4()
setAux4(String aux4)

String getName()
setName(String name)

ObjetoTabla1

This is the structure of a row of the Interconnect statistics 1 table.

Attributes:

String name
String aux1
String aux2
String aux3
String aux4

Methods:

ObjetoTabla1(String s1, String s2, String s3, String s4, String s5)

String getAux1()
setAux1(String aux1)

String getAux2
setAux2(String aux2)

String getAux3()
setAux3(String aux3)

String getAux4()
setAux4(String aux4)

String getName()
setName(String name)

ObjetoTabla2

This is the structure of a row of the interconnect statistics 2nd tables.

Attributes:

String name
String aux1
String aux2
String aux3
String aux4
String aux5

Methods:

ObjetoTabla2(String s1, String s2, String s3, String s4, String s5, String s6)

String getAux1()
setAux1(String aux1)

String getAux2
setAux2(String aux2)

String getAux3()
setAux3(String aux3)

String getAux4()
setAux4(String aux4)

String getAux5()
setAux4(String aux5)

String getName()
setName(String name)

ObjetoTabla3

This is the structure of a row of the SWARM Processor table.

Attributes:

String name
String aux1
String aux2

Methods:

ObjetoTabla3(String s1, String s2, String s3)

String getAux1()
setAux1(String aux1)

String getAux2
setAux2(String aux2)

String getName()
setName(String name)

ObjetoTabla4

This is the structure of a row of the general statistics table.

Attributes:

String name
String aux1
String aux2
String aux3
String aux4
String aux5
String aux6

Methods:

ObjetoTabla4(String s1, String s2, String s3, String s4, String s5, String s6, String s7)

String getAux1()
setAux1(String aux1)

String getAux2
setAux2(String aux2)

String getAux3()
setAux3(String aux3)

String getAux4()
setAux4(String aux4)

String getAux5()
setAux4(String aux5)

String getAux6()
setAux4(String aux6)

String getName()
setName(String name)

ObjetoTabla5

This is the structure of a row of the buffers table.

Attributes:

String name
String aux1
String aux2
String aux3
String aux4

Methods:

ObjetoTabla5(String s1, String s2, String s3, String s4, String s5,)

String getAux1()
setAux1(String aux1)

String getAux2
setAux2(String aux2)

String getAux3()
setAux3(String aux3)

String getAux4()
setAux4(String aux4)

String getAux5()
setAux4(String aux5)

String getName()
setName(String name)

ObjetoTabla6

This is the structure of a row of the Xpipes Stats Interconnect table.

Attributes:

String name
String aux1
String aux2

Methods:

ObjetoTabla6(String s1, String s2, String s3)

String getAux1()
setAux1(String aux1)

String getAux2
setAux2(String aux2)

String getName()
setName(String name)

ObjetoTabla7

This is the structure of a row of the Second Compare statistics table.

Attributes:

String name
String aux1
String aux2
String aux3
String aux4
String aux5
String aux6
String aux7
String aux8
String aux9

Methods:

ObjetoTabla7(String s1, String s2, String s3, String s4, String s5, String s6, String s7, String s8, String s9, String s10)

String getAux1()
setAux1(String aux1)
String getAux2
setAux2(String aux2)
String getAux3()
setAux3(String aux3)
String getAux4()
setAux4(String aux4)
String getAux5()
setAux4(String aux5)
String getAux6()
setAux4(String aux6)
String getAux7()
setAux4(String aux7)
String getAux8()
setAux4(String aux8)
String getAux9()
setAux4(String aux9)
String getName()
setName(String name)

Parse

This class analyses the network TXT file for open the net.

Attributes:

Network net
int end
int barNumber
LineNumberReader input
BufferedWriter advise
Principal p

Methods:

Parse(Principal p)

analyse(String a):

This method returns an integer that indicates what component is being reading.

int createCore():

Create a core, pm, shm, smm or int component. It returns 0 if the component was created or 1 if not.

int createCore2():

Read a core sentence and create the core. It returns 0 if the component was created or 1 if not.

int createInterrupt():

Read an interrupt sentence and create the interrupt. It returns 0 if the component was created or 1 if not.

int createLink():

Read a linkj sentence and create the linkj. It returns 0 if the component was created or 1 if not.

int createPrivateMem():

Read a private memory sentence and create the pm. It returns 0 if the component was created or 1 if not.

int createRoute():

Read a route sentence and create the route. It returns 0 if the component was created or 1 if not.

int createSemaphore():

Read a semaphore sentence and create the smm. It returns 0 if the component was created or 1 if not.

int createSharedMem():

Read a shared memory sentence and create the shm. It returns 0 if the component was created or 1 if not.

int createSwitch():

Read a switch sentence and create the switch. It returns 0 if the component was created or 1 if not.

boolean createTopology():

Read a topology sentence and create it. It returns true if the topology was created or false if not.

double doubleValue(char[] a)
int value(char[] a)

ParseReturnPar readFile():

This method opens the file the user has chosen and interprets it.

ParseReturnPar readFile(String route):

This method opens the file of the route the user has chosen and interprets it.

String readNameWord():

This method reads words of the file which are names of components.

String readWord():

This method reads words of the file and returns them.

int whatCoreType(String tipo)

Parse_Stats

This class analyse the AMBA stats TXT file.

Attributes:

String file
Vector introduction
Vector interconnectionsTable1
String interconnectionsLabel
Vector interconnectionsTable2
Vector processors
Vector generalTable
Vector general
Vector inter1
Vector inter2
String interLabel
Vector pTable1
Vector pTable2
Vector procs

Methods:

Parse_Stats()

double doubleValue(char[] a)
int value(char[] a)

fileToVectors():

Obtain the vectors to fill the statistics tables.

fillTheGaps():

Obtain the values of the empty cells of each table.

boolean p_interTable1a(int i)
boolean p_interTable1b(int i)
boolean p_interTable1c(int i)

boolean p_interTable2(int position)

boolean processorTable1(int position, Vector v)
boolean processorTable2(int position, Vector v)

String pTableGeneral(String label)
readFile()

Parse_Stats2

This class analyse the Xpipes stats file.

Attributes:

String file
Vector introduction
Vector Int1
Vector Int2
Vector NIs
Vector Buffers
Vector LinkR
String OCPLabel
Vector OCPTable
Vector Procs

Methods:

Parse_Stats2()

fileToVectors()

p_BufTable(Vector v)

boolean p_interTable1(int i)

Vector p_interTable2(Vector v, int i)

p_linkTable(Vector v)

p_NITable(Vector v)

p_OCPTable(Vector v)

p_procsTable(Vector v)

ObjetoTabla3 processorTable_aux(int position, Vector v)

readFile()

Principal_AboutBox

This is the credits window

Attributes:

String product
String version
String copyright
String comments

Methods:

Principal_AboutBox(Frame parent)

actionPerformed(ActionEvent e):

Close the dialog on a button event.

PrivateMemory

This class represents a pm element of the network.

Attributes:

String sw
double divFrec
int numBuf
String arg5
final String preArg6
String posMem
String CorePrefix
boolean bi
int inputBufDepth

Methods:

PrivateMemory(String i1, String i2, double d3, int i4,String a5,String i6, boolean b7,int inputBufDepth)

PrivateMemory (String i1, String i2, double d3, int i4,String a5,String i6, boolean b7,boolean bidir,int inputBufDepth)

setPosMem(String i)
String getPosMem()
setSwitch(String i)
String getSwitch()
setDivFrec(double i)
double getDivFrec()
setNumBuf(int i)
int getNumBuf()
setArg5(String s)
String getArg5()
setIncompleted(boolean b)
boolean getIncompleted()
boolean isBi()
setBi(boolean bi)
int getInputBufDepth()
setInputBufDepth(int inputBufDepth)

processor_stats

This is the structure of any row in the general table of Stats.

Attributes:

Vector tab1
Vector tab2
String labelbus
String[] infolabels
Vector infoCache

Methods:

processor_stats(Vector cache, String labelbus, String[] labels, Vector tab1, Vector tab2)
Vector getInfoCache()
setInfoCache(Vector infoCache)
String getLabelbus()
setLabelbus(String labelbus)
String[] getLabels()
setLabels(String[] labels)
Vector getTab1()
setTab1(Vector tab1)
Vector getTab2()
setTab2(Vector tab2)

processor_stats2

This class represents any row of processor table of Xpipes Stats.

Attributes:

Vector tab1
String[] infolabels
Vector infoCache

Methods:

processor_stats2(Vector cache, String[] labels, Vector tab1)
Vector getInfoCache()
setInfoCache(Vector infoCache)
String[] getLabels()
setLabels(String[] labels)
Vector getTab1()
setTab1(Vector tab1)

Productor

This class is used by the Tube for synchronize the threads.

Attributes:

Tuberia tuberia
char ch

Methods:

Productor(Tuberia t, char cha)

run()

Tuberia

This is the tube that synchronizes the threads.

Attributes:

char buffer[]
int siguiente
boolean estaLlena
boolean estaVacia

Methods:

Tuberia()

boolean isEstaLlena():

Returns true if the tube is full.

setEstaLlena(boolean estaLlena)

boolean isEstaVacia():

If the tube is empty or full puts this variable = true or false.

synchronized char recoger():

Method that catches letters of the buffer (like a consumer)

synchronized void lanzar(char c):

Method that puts letters into the buffer (like a productor)

Route

This class represents a path between two elements of the network.

Attributes:

String source
String target
int sourceType (0 ->core_, 1->pm_ , 2->shm_, 3->smm_, 4->int_)
int targetType (0 ->core_, 1->pm_ , 2->shm_, 3->smm_, 4->int_)
Vector switchesList
String stringName
boolean incompletedlist
Network net

Methods:

Route(String i1, String i2,int tipo1, int tipo2, Vector v3)
Route(String i1, String i2,int tipo1, int tipo2, Vector v3,String n)

boolean getIncompleted()
setIncompleted(boolean b)

setIncompletedList(boolean b)
boolean getIncompletedList()

String getNameString ()
setNameString (String s, String d, int stype, int dtype)

String getSource()
setSource(String i)

int getSourceType()
setSourceType(int i)

String getTarget()
setTarget(String i)

int getTargetType()
setTargetType(int i)

Vector getVector()
setVector(Vector v)

RouteInforWindow

This class shows the attributes of the route.

Attributes:

Network net
Principal p

Methods:

RouteInforWindow(String nam,Principal p)

Vector giveMeLinks(int n):

Returns a Vector of links that appear in the route "n".

JComponent makeTextPanel(String text):

Fill the panels

acceptButton_actionPerformed(ActionEvent e):

Close the window.

deleteButton_actionPerformed(ActionEvent e):

Close the window and show a new window to delete the link => new LinkMenu(2,number,p,this)

modifyButton_actionPerformed(ActionEvent e):

Close the window and show a new window to modify the link => new LinkMenu(1,number,p,this)

String[] readOnlySwitchName(String n):

Retuns the switch name , and the subchannel in the second position.

RouteMenu

This window allows create, modify or delete a route.

Attributes:

Network r
Principal p

Methods:

RouteMenu(int n, int direct, Principal p)

Vector readList() :
Returns a string with the chain of the route(switches)

String write(Vector v):
Returns a string with the chain of the route(switches)

Vector analice(char[] n):
Returns a vector with the switches list like this: 0-1-2-…

int value(char[] a)

acceptButton_actionPerformed(ActionEvent e)

AddButton_actionPerformed(ActionEvent e)

cancelButton_actionPerformed(ActionEvent e)

DeleteAllButton_actionPerformed(ActionEvent e)

DeleteButton_actionPerformed(ActionEvent e)

ModifyButton_actionPerformed(ActionEvent e)

properties()

String[] readOnlySwitchName(String n):
Retuns the switch name , and the subchannel in the second position.

Object[] routeValue(String n):
Returns an integer 2 positions array, In position 0 there is the number and position 1 contains the type.

Save

This class saves the network into a TXT file.

Attributes:

Network r
BufferedWriter output

Methods:

Save()

String getPathWithoutExtension(String s):

Returns the path without the extenion (.txt)

String saveFile(Network net,String nametxt,boolean saveAs):

This method opens the file the user has choosed and writes in it.

String seeCoreType(int i):

This method returns the type of the coreType according with that number.

String seeTopologyType(int i):

This method returns the type of the topologyType according with that number.

Semaphore

This represents a semaphore (smm) element in the net.

Attributes:

String sw
double divFrec
int numBuf
String arg5
final String preArg6
String posMem
boolean fixed
String CorePrefix
boolean bi
int inputBufDepth

Methods:

Semaphore(String i1, String i2, double d3, int i4,String a5,String i6, boolean b7,int inputBufDepth)

Semaphore(String i1, String i2, double d3, int i4,String a5,String i6, boolean b7,boolean bidir,int inputBufDepth)

setPosMem(String i)
String getPosMem()
setFixed(boolean b)
boolean getFixed()
setSwitch(String i)
String getSwitch()
setDivFrec(double i)
double getDivFrec()
setNumBuf(int i)
int getNumBuf()
setArg5(String s)
String getArg5()
setIncompleted(boolean b)
boolean getIncompleted()
boolean isBi()
setBi(boolean bi)
int getInputBufDepth()
setInputBufDepth(int inputBufDepth)

SharedMemory

This represents a shared memory (shm) element in the net.

Attributes:

String sw
double divFrec
int numBuf
String arg5
final String preArg6
String posMem
boolean fixed
String CorePrefix
boolean bi
int inputBufDepth

Methods:

SharedMemory(String i1, String i2, double d3, int i4,String a5,String i6, boolean b7,int inputBufDepth)

SharedMemory (String i1, String i2, double d3, int i4,String a5,String i6, boolean b7,boolean bidir,int inputBufDepth)

setPosMem(String i)
String getPosMem()
setFixed(boolean b)
boolean getFixed()
setSwitch(String i)
String getSwitch()
setDivFrec(double i)
double getDivFrec()
setNumBuf(int i)
int getNumBuf()
setArg5(String s)
String getArg5()
setIncompleted(boolean b)
boolean getIncompleted()
boolean isBi()
setBi(boolean bi)
int getInputBufDepth()
setInputBufDepth(int inputBufDepth)

StatsWindow

This class shows the tables of the AMBA stats file.

Attributes:

JTable tablaInter1
JTable tablaInter2
JTable tablaProc1
JTable tablaProc2
JTable tablaGen

Methods:

StatsWindow(Vector proc,Vector in1,Vector in2, String interL,Vector general)

double doubleValue(char[] a)

deseleccionar():
It puts all the radioButtons with selected =false.

drawGraph():
Show a graphic between the selected processors.

drawTables():
Depends on each jradioButtons some tables will be showed.

graphsResults(Vector selected, int longGrafico, int[] numProcSelected):
Create the graph with the amounts.

jRadioButtonGeneral_actionPerformed(ActionEvent e):
Deselect all jRadioButtons, then Selects this radioButton and draw the tables.

jRadioButtonInterc_actionPerformed(ActionEvent e):
Deselect all jRadioButtons, then Selects this radioButton and draw the tables.

jRadioButtonProc_actionPerformed(ActionEvent e,int nump):
nump is the number of the processor. It deselects all jRadioButtons, then Selects this radioButton and draw the tables.

StatsWindow2

This class shows the tables of the Xpipes stats file.

Attributes:

JTable tablaInter1
JTable tablaInter2
JTable tablaProc1
JTable tablaNI1
JTable tablaLR
JTable tablaBuf
JTable tablaOCP

Methods:

StatsWindow2(Vector int1,Vector int2,Vector buffs, Vector LinkReceiv,Vector ocpT,String ocpL,Vector NItables,Vector proc)

double doubleValue(char[] a)

deseleccionar():
It puts all the radioButtons with selected =false.

drawGraph():
Show a graphic between the selected processors.

drawTables():
Depends on each jradioButtons some tables will be showed.

graphsResults(Vector selected, int longGrafico, int[] numProcSelected):
Create the graph with the amounts.

jRadioButtonBuffers_actionPerformed(ActionEvent e)

jRadioButtonInterc_actionPerformed(ActionEvent e)

jRadioButtonLR_actionPerformed(ActionEvent e)

jRadioButtonNI_actionPerformed(ActionEvent e,int numNI)

jRadioButtonOCP_actionPerformed(ActionEvent e)

jRadioButtonProc_actionPerformed(ActionEvent e,int nump)

StatsWindowCompare

This class shows diferences between AMBA and Xpipes Stats.

Attributes:

JTable tabla1
JTable tabla2

Methods:

StatsWindowCompare(Vector intX1,Vector intX2,Vector intA1,Vector intA2)

double doubleValue(char[] a)

deseleccionar():

It puts all the radioButtons with selected =false.

drawGraph():

Show a graphic between the selected processors.

drawTables():

Depends on each jradioButtons some tables will be showed.

graphsResults(Vector selected, int longGrafico, int[] numProcSelected):

Create the graph with the amounts.

jButtonGraphs_actionPerformed(ActionEvent e,int tipo)

SunFloorMenu

This class lets to load or create the tables that SunFloor needs.

Attributes:

int numApli
JTable tabla2
JTable tabla3
JTabbedPane jTabbedPane1
Vector tablas1
int SumnFloorOK
Principal p

Methods:

SunFloorMenu(int format, Principal pr)

String getPathWithoutExtension(String s):
Returns the path without the extenion (.txt)

saveFiles():
Save the three tables into XML files

saveXML(int type)

desactiveAll():
Deselcts all the radioButtons

dibuja():
draw the correspondent table or buttons in each step.

jButtonAccept_actionAdapter(ActionEvent e)
jButtonBack_actionPerformed(ActionEvent e)
jButtonCancel_actionPerformed(ActionEvent e)
jButtonChangeDefault_actionAdapter(ActionEvent e)
jButtonDeleteApli_actionPerformed(ActionEvent e)
jButtonDeleteFila_actionPerformed(ActionEvent e)
jButtonNewApli_actionPerformed(ActionEvent e)
jButtonNewFila_actionPerformed(ActionEvent e)
jButtonNext_actionPerformed(ActionEvent e)
jButtonSaveAs_actionAdapter(ActionEvent e)
jFileChooser1_actionPerformed(ActionEvent e)
jRadioButtonDefault_actionPerformed(ActionEvent e)
jRadioButtonNew_actionPerformed(ActionEvent e)
jRadioButtonOpen_actionPerformed(ActionEvent e)
muestraAbrirNuevo(boolean b)

SWARM_processor

This class contains all the Information about a processor in the statswindow.

Attributes:

Vector processorTable2
String processorAccesesLabel
Vector processorTable1
String processorInfoLabel1
String processorInfoLabel2
String processorInfoLabel3
String processorInfoLabel4
String processorInfoLabel5
Vector infoCache

Methods:

SWARM_processor(Vector v1, String s2, Vector v3, String s4, String s5, String s6, String s7, String s8, Vector v9)

SwInforWindow

This class shows the attributes of a switch.

Attributes:

Principal p
Network r

Methods:

SwInforWindow(String nom,Principal p)

Vector giveMeRoutes(int n):

Returns a Vector of routes in which the core appears.

JComponent makeTextPanel(String text)

acceptButton_actionPerformed(ActionEvent e):

Close the window.

deleteButton_actionPerformed(ActionEvent e):

Close the window and show a new window to delete the switch => menu=new SwMenu(2,number,p,this)

modifyButton_actionPerformed(ActionEvent e):

Close the window and show a new window to modify the switch => menu=new SwMenu(1,number,p,this);

String[] readOnlySwitchName(String n):

Retuns the switch name , and the subchannel in the second position.

Switches

This class represents another element of the network, the switch.

Attributes:

int inputs
int outputs
int numBuf
int coorX
int coorY
int inputsUsedConnections
int outputsUsedConnections
boolean XYconections
int inputBufDepth
Vector subChannels

Methods:

Switches(String i1, int i2, int i3, int i4,int i5)
Switches(String i1, int i2, int i3, int i4,int i5,Vector subcha)

Switches(String i1, int i2, int i3, int i4, int i5, int i6,int i7):
Switches(String i1, int i2, int i3, int i4, int i5, int i6,int i7,Vector subcha):

for topologyType= mesh or torus.

int getCoorX()
setCoorX(int i)
int getCoorY()
setCoorY(int i)
int getInputBufDepth()
setInputBufDepth(int inputBufDepth)
int getInputs()
setInputs(int i)
int getInputsUsedConnections()
setOutputsUsedConnections(int i)
int getNumBuf()
setNumBuf(int i)
int getOutputs()
setOutputs(int i)
int getOutputsUsedConnections()
setOutputsUsedConnections(int i)
Vector getSubChannels()
setSubChannels(Vector subChannels)

boolean hasSubchannel(String a)

SwMenu

This class shows a window to create modify or delete a switch.

Attributes:

Principal p
Network r

Methods:

SwMenu(int n, int directa,Principal p)

boolean estaChannels(String s) :

Returns true if the subchannel already exist in the switch.

CancelButton_actionPerformed(ActionEvent e):

Close the window.

acceptButton_actionPerformed(ActionEvent e):

Make sure all the attributes are correct and modify or delete the new element.

int value(char[] a):

Returns the int value of a char array of digits.

properties():

Show the properties of the element selected.

String[] readOnlySwitchName(String n)
update2()

updateSubcha_actionPerformed(ActionEvent e)

boolean esta(String n):

Returns the switch name , and the subchannel in the second position.

addButton_actionPerformed(ActionEvent e)

deleteButton_actionPerformed(ActionEvent e)

Tree

This class represents the Tree of the network.

Attributes:

Vector facts[]
Principal p
Network net
DefaultMutableTreeNode root
 DefaultMutableTreeNode branch, selection
 JTree tree
 DefaultTreeModel model

Methods:

Tree(Principal p)

updateTree(Network n):

Build the tree from the network.

TreeSunFloor

This class represents the Tree of Sunfloor.

Attributes:

Principal p
String facts[]
DefaultMutableTreeNode root
DefaultMutableTreeNode app, tecno, ip , selection
JTree tree
DefaultTreeModel model

Methods:

TreeSunfloor(Principal p)

public void updateTree():

Build the tree from the network.

4. EXTERN TOOLS

NoCdificador is an application that creates, modifies and control the correctness of a network, but it can't compile, simulate or create topologies automatically. To do all this tasks, it uses some external applications such as SunFloor, xPipesCompiler or MPARM.

4.1 SunFloor

SunFloor is a custom CAD tool used to synthesize the most power/performance efficient NoC topology that satisfies the application requirements. The application traffic characteristics, size of the cores, and the area and power models for the network components are obtained as inputs to the synthesis engine. The tool generates different NoC switches and maps the cores onto the switches. During the synthesis process, it determines deadlock-free routes for the different traffic flows of the application. The tool also produces the 2D floorplan of the synthesized NoC topology. This floorplan is generated on *gnuplot* format, so *gnuplot* is used to convert the floorplan generated by SunFloor to png, in order to be shown by NoCdificador. The output produced by the tool is a text file that defines the synthesized topology, which is fed to the next phase of the flow. In our case, it is fed to NoCdificador, in order to be checked or even modified before being compiled.

SunFloor is a fully automated solution that optimizes architectural parameters of the NoC tuning based upon applications requirements. It can generate both regular and custom-designed topologies, and choose it in an automatic way. Usually the type of topology is one of the most problematic decision in a hand-mapped design.

Designs generated by SunFloor are much more optimized than hand-mapped design for the same frequency. SunFloor design uses less switches and reduces the power consumption, although sometimes has a little increase of the area. But, maybe, the most important fact is that a design made with SunFloor can be ready in a few hours. A hand-mapped design could cost several weeks working on it.

4.2 XpipesCompiler

XpipesCompiler is another custom tool. It reads a topology definition file, generated by SunFloor, hand-mapped or created with NoCdificador, and instantiates the RTL description of the NoC components using Xpipes, a pre-designed SystemC RTL component library. The modules in the component library support a large number of instantiation parameters, such as different input/output ports, buffer sizes, etc. The tool also interconnects the RTL description of the processor/memory cores of the SoC (which are predesigned components, taken as user inputs) with the RTL code of the network components. The output of this step is the RTL design of the entire NoC that can be simulated and synthesized.

4.3 MPARM

MPARM is a multi-processor cycle-accurate architectural simulator. Its purpose is the system-level analysis of design tradeoffs in the usage of different processors, interconnects, memory hierarchies and other devices.

MPARM simulates different processor models, but in our case is SWARM the most important. SWARM (SoftWare ARM) is a modular simulation of an ARM 7 processor.

It also has different interconnection models. In our case, we use xpipes NoC to simulate the topology who has just been designed, but we also use AMBA model in order to compare NoC interconnection results with bus interconnection results.

5. CONCLUSION

As our project was aimed on the development of an application to configure NoCs, we have not been able to reach a great number of conclusions. Anyway, we could enumerate some of them, which can be divided in two groups: conclusions we reached during the development and conclusions we reached while using the application.

5.1 Development conclusions

Maybe the main conclusion we could remark is the design and implementation the application.

But this develop have made us learn much other things. For example, we have discover the world of NoC. We have learnt what a NoC is, how does them work and why they are necessary.

Another important thing we could detail is that we have got familiar with the investigation environment, that was completely unknown for us until we began this project.

Finally, we'd like to express how difficult was sometimes to develop an application that depends on other applications, when both are being developed at the same time. We have never worked like this before, and we have realized coordination is very important to avoid useless efforts.

5.2 User conclusions

Beside being developers, we have also been users of NoCdificador. As users, we have been able to conclude some remarkable facts about differences compiling SoC's with Xpipes (using NoC for the interconnection) and AMBA (using buses).

Using Xpipes provides better performance under congestion, and much better scalability. On the other hand, AMBA still gives a lower power consumption, and much better area integration.

To sum up, we could say that NoC's technology is growing up very fast, but still have to improve some aspects such as power consumption an integration.

6. BIBLIOGRAPHY

[1] A Complete Network-On-Chip Emulation Framework. N.Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, F. Actor.

[2] NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip. Davide Bertozzi, Antoine Jalabert, Srinivasan Murali, Rutuparna Tamhankar, Stergios Stergiou, Luca Benini, and Giovanni De Micheli.

[3] Diseño de redes en chip de propósito específico con información de rutado físico. David Atienza, Srinivasan Murali3, Federico Angiolini4, Luca Benini, Giovanni De Micheli2, José M. Mendías y Román Hermida.

[4] Networks on Chips: A New SoC Paradigm. Luca Benini University of Bologna, Giovanni De Micheli Stanford University.

[5] Designing Application-Specific Networks on Chips with Floorplan Information. Srinivasan Murali, Paolo Meloni, Federico Angiolini, David Atienza, Salvatore Carta, Luca Benini, Giovanni De Micheli, Luigi Raffo.

[6] Routing Aware Switch Hardware Customization for Networks on Chips. Paolo Meloni, Srinivasan Murali, Salvatore Carta, Massimo Camplani, Luigi Raffo, Giovanni De Micheli

[7] SunFloor: Application-Specific SunFloor: Application-Specific Design of Networks-on-Chip. S. Murali , D. Atienza, G. De Micheli - D. Atienza, G. De Micheli , F. Angiolini, L. Benini - F. Angiolini, L. Benini , P. Meloni, S. Carta, L. Raffo - P. Meloni, S. Carta, L. Raffo

[8] MPARM.Page
www-micrel.deis.unibo.it/sitonew/research/mparm.html