NoC System Generator

Version 2018 (Beta) - User's Guide

Contents

1 Overview

When you start the GUI, first some command windows appears and disappears in the background, then the start window appears:

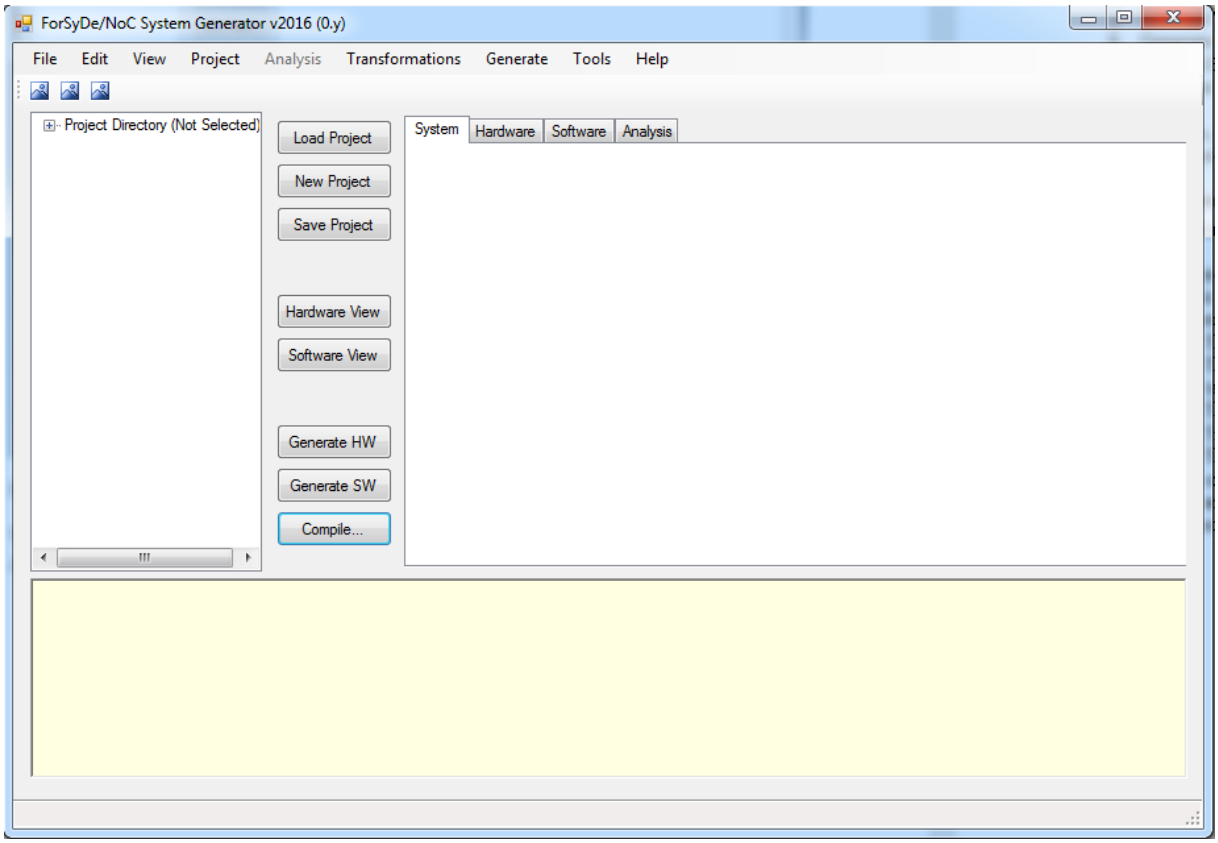


Figure 1.1. Main Window.

1.1 Main View Information Windows

Besides the Buttons and drop-down Menus, there are three windows displaying information:

1. The project directory window listing all files associated with the project
2. The main information window, having four tabs (the four tabs are explained further in chapter 6 - System View):
    a. The System Tab - displaying the content of the current active <system>.xml file
    b. The Hardware Tab – displaying information and properties (area estimates etc.) of the target HW implementation.
    c. The Software Tab – displaying information and properties (Worst Case Execution Times etc.) of the target SW implementation.
    d. The Analysis Tab – displaying information about the schedules of the software components. **NB! This window is not used at present. Instead it presents a mockup of future intended use.**
3. The console window – Displaying the result of the latest command.

1.2 Main window drop-down menus

A list of menus is available on the top. Their functionilty is described later in detail in Chapter 7 on page 33.

1. File
2. Edit
3. View
4. Project
5. Analysis
6. Transformations
7. Generate
8. Tools
9. Help

1.3 Quick Access Buttons

The main window has the following Quick Buttons that gives quick access to the most commonly used design steps:

1. Load Project – Opens an existing project
2. New Project – Creates a new project
3. Save Project – Saves the current project
4. Hardware View – Opens the Hardware View
5. Software View – Opens the Software View
6. Generate HW – Generates all necessary HW files for synthesizing the project using the backend FPGA tool vendor (Altera Quartus or Xilinx Vivado)
7. Generate SW – Generates all necessary SW files for compiling using the backend FPGA tools SDK
8. Compile/Generate Scripts – Opens the Script View for generating scripts for compiling SW on the target FPGA tool vendor. **NB! Since only Altera Quartus scripts is supported at present, this option is disabled in the beta release.**

2 Design Flow

To create a design using ForSyDe NSG, the following design flow should be followed:

1. Create Project
2. Select target technology
3. Decide NoC properties (Size, Topology, RNI type etc.)
4. Open Hardware View - Edit HW/node properties
5. Open Software View - Add, place and edit SW processes
6. Generate HW & SW for target platform
7. Synthesize platform for target technology
8. Use target tool SDK (or online system) to compile, download and debug SW.
9. If necessary, repeat steps 3-8 until system performance is satisfactory

2.1 Known Issues/“Features”

Step 6. The first time you press Generate HW to create a new project, you have to press Generate HW twice. The first time will generate the directory structure and the second will produce all requested files.

Step 5. Sometime, when connecting the communication lines between software components, the arrow pointer gets stuck on the dotted line separating different nodes.

Step 8. When creating the SW project directories for the first time in the SDK, the SDK requires that the directories do not exist beforehand. Therefore, you might have to delete the Software Node directories, create them using the SDK, and then run Generate SW to produce the required SW files.

3 Creating a New Project

To create a project, do the following steps:

1. Open a new project and select a working directory for storing your files and press Next...

2. Select target vendor, the FPGA board you want to use and the target directory that the NSG-tool should use for generating the FPGA project files.

3. Optional - Change NoC Settings (if defaults are not sufficient) – *NB! Only 3D and 2D Mesh Type NoCs of Nostrum type is currently available in the GUI.*

4. Save the project before you start with next step by pressing the <Save Project> button (if there is an asterisk in the System Tab – System* - your project has been modified and needs saving)

3.1.1 Set Project and Working Directory

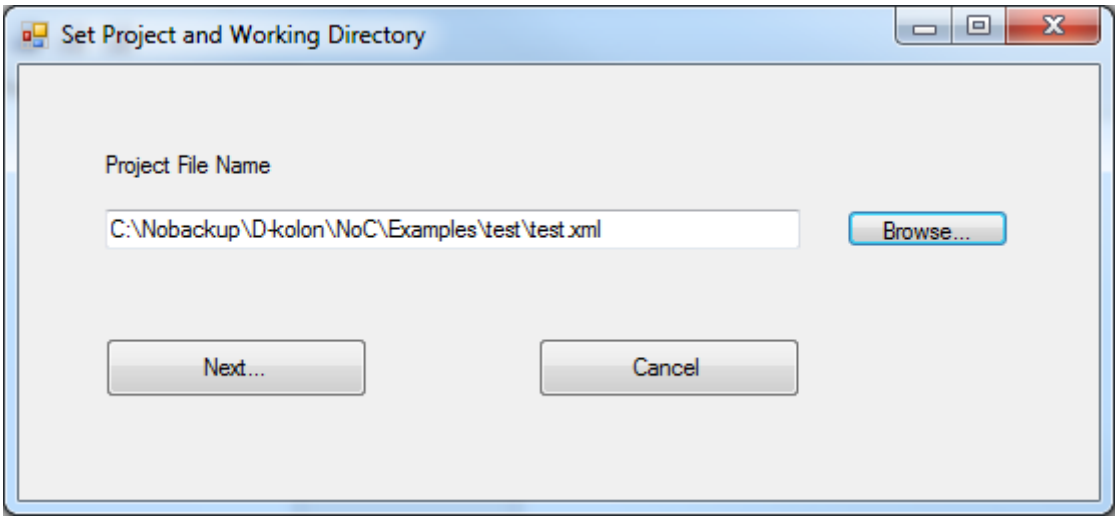


Figure 3.1. Set Project and Working Directory Menu

Browse to find the right directory working and then press *Next...*

3.2 Project Settings Dialog

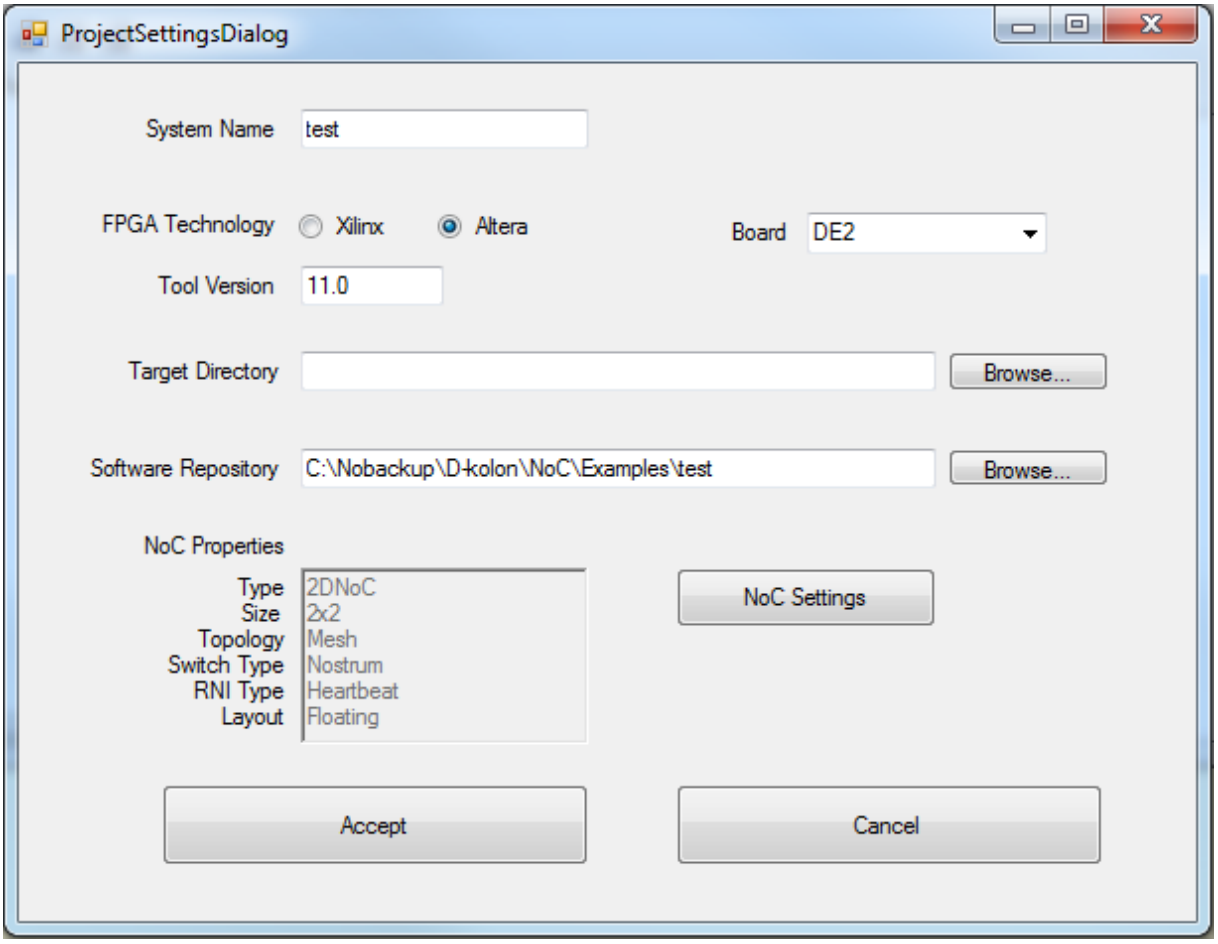


Figure 3.2. Project Settings Dialog Window

Restrictions

1. System Name – It is recommended that the system name have the same name as the project XML file to avoid problems during VHDL/FPGA backend compilation.
2. Board – You may only use one of the boards listed in the dropbox menu
3. Target Directory is the directory where the ForSyDe/NSG tool will store/generate all the necessary files to create a project for the target FPGA vendor
4. Software Repository is the directory from which the ForSyDe/NSG tool will search for the software files associated with the project

3.3 NoC Settings

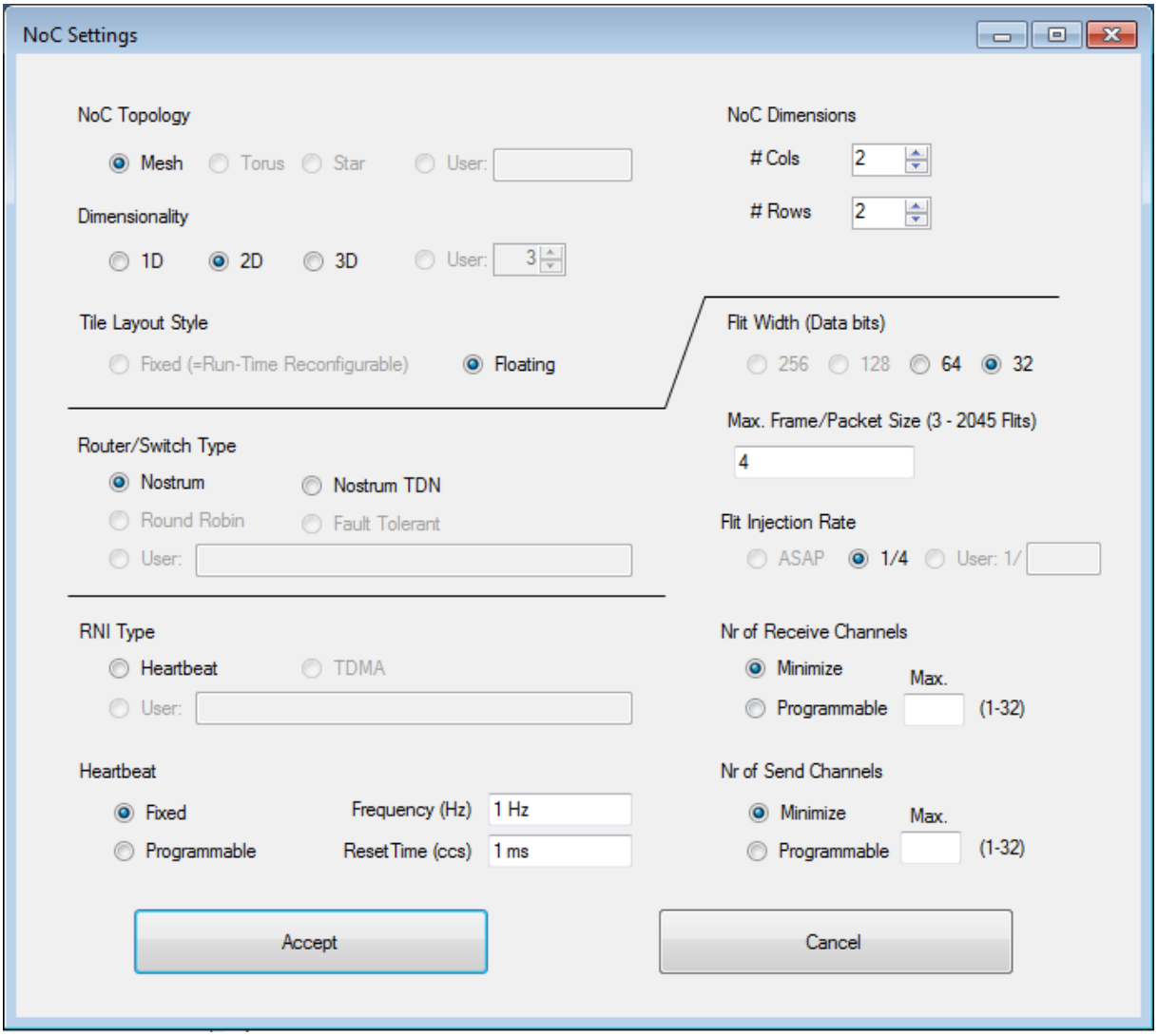


Figure 3.3. NoC Default Settings

Restrictions

1. NoC Type
    a. Topology. Sets which connecting pattern should be used between switches in the NoC. Currently, only the Mesh NoC is available.
    b. Dimensionality. Set how many dimensions should be used when connecting the Switches. Currently, 1D, 2D and 3D are available.
    c. Tile layout style is sets if the system can be runtime reprogrammable or not. If it is set to floating, i.e., there is no support for handling Run-Time Reconfigurable (RTR) Systems. If it is set to Fixed, the place of the tiles of the NoC System is fixed in the layout to certain regions, thus enabling Run-Time Reconfiguration.
2. Router/Switch Type – Currently, only the Nostrum with and without support for TDNs are available
    a. Flit Width – Here you select the number of data bits there are in the flits.
    b. Maximum FrameSize – The largest amount of information (in words) you ever intend to send over the NoC in this implementation.

The minimum value is 3. The Maximum value is 512. The maximum value must include the number of header flits/words. For 32 bits, two header flits are needed to transmit the setup information. For 64 bits and more, only one header flit is needed.
Use 64 (or 42 ☺) or some other large number if you do not know what value to set. As long as it is large enough to include the number of header flits, it's ok.

c. Flit injection rate – This parameter sets how often flits are injected into the NoC, provided that the path is clear. Only 1/4 (1 every fourth switch cycle) is available for Nostrum Switches. For Nostrum with TDNs, you set the parameter "TDN" in the Systems Window to the value "{x,y,z,w}", where the value 1 indicates that the NI should only inject in that TDN Switch Cycle Window. The value 0 indicates no transmission in the Switch cycle. If you set the parameter to "{0,0,0,1}" the NI will only inject packet flits in TDN 0, the value "{0,0,1,0}", TDN 1, etc. Thus, the value "{1,1,1,1}" indicates transmission in every TDN cycle, and the value "{0,0,0,0}" means no transmission at all.

d. Nr of Receive Channels. Here you set the number of receiver channels used in the NI. If you select minimize, the number of channels will be hard-wired and fixed in hardware to the number of channels used for connecting software components in the Software View. If you select programmable, the number of channels will be fixed to the number you select, and they will be run-time programmable. They will be initialized at start-up according to how you have connected Software Components in the Software View.

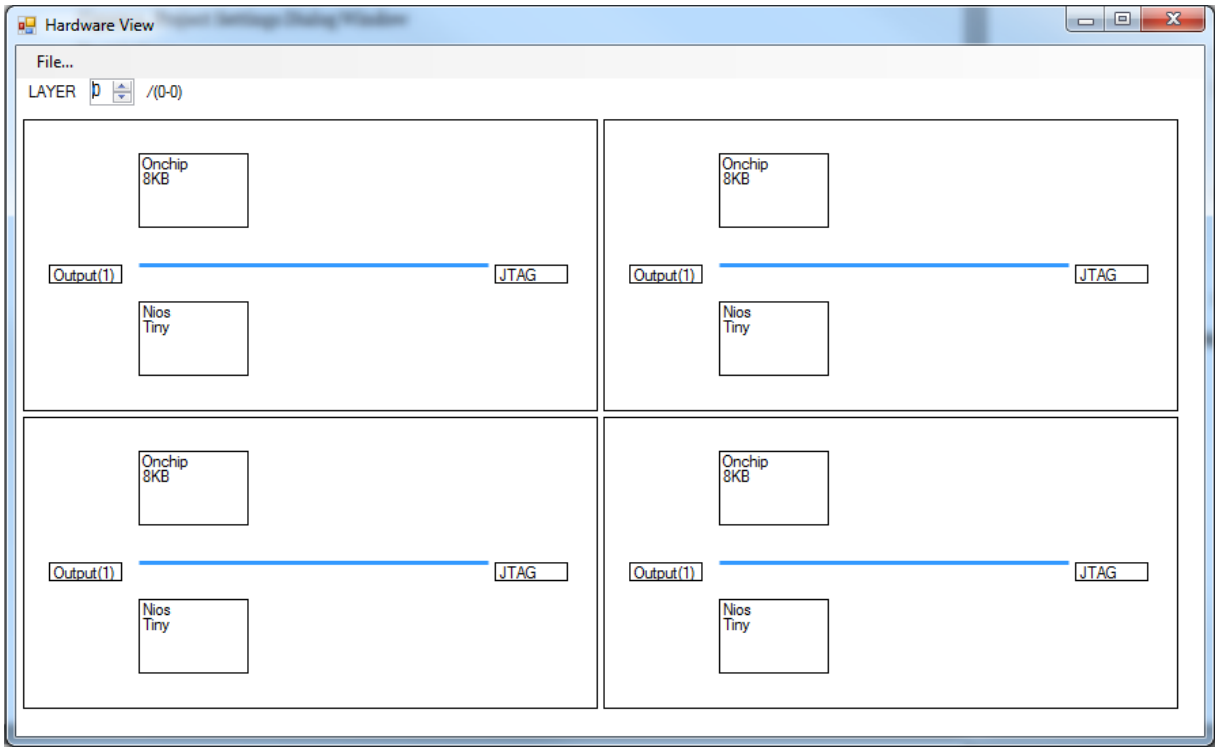
e. Nr of Send Channels. Here you set the number of Send channels used in the NI. If you select minimize, the number of channels will be hardwired and fixed in hardware to the number of channels used for connecting software components in the Software View. If you select programmable, the number of channels will be fixed to the number you select, and they will be run-time programmable. They will be initialized at start-up according to how you have connected Software Components in the Software View.

3. RNI Type – Selects which NI you are using. Currently, only the Synchronous Real-Time Heartbeat is available.

   a. Heartbeat Frequency sets the pulse for your system.
   A small value of 1 Hz is good for debugging onboard applications.
   A large value of ~10-100 kHz is good when debugging IPs using VHDL simulations.

   b. Reset sets the time to the first heartbeat. It should be set to a value that lets you boot the system safely.

4 The Hardware View



Figre 4.1. Default Hardware View

In the hardware view, you can configure each node of the NoC individually. Memories are displayed on the top, processors and caches on the bottom, IOs are displayed on the left, and JTAG debug modules to the right. IP modules are displayed together with the processors. The Layer selects which NoC layer you are in case you have selected a 3D NoC. If not, the only layer available will be set to 0.

The number of nodes on each layer is restricted in the GUI to maximum 4x4. The reason is that the NSG currently do not have a zoom function and that it is hard to view more than 16 Nodes simultaneously. The NoC generator backend does not have any restrictions.

If you click on a Tile, the Hardware Node Data Entry View for that node will appear.

4.1.1 Several CPUs

You can put more than one CPU in a node. CPUs in the same node share the bus inside and have one common NoC interface out of the node. The GUI is restricted to three nodes because of visibility reasons. The backend generator has no restrictions. However, it is recommended to put the number of CPUs on the shared bus low, since software components placed in the same node will compete for the resources inside it. The exception to this rules is the onchip memories, which are connected to each CPU node directly. To create a shared memory, external memory needs to be used.

4.1.2 Limitations with current NSG version

You have to use as many onchip memories as there are processors in the node. If you try to use more onchip memories than there are processors, the extra memories are ignored. Also, you can only connect one memory per processor. This means that currently, soft processors cannot use external memories, since it needs the onchip memory for storing its boot code.

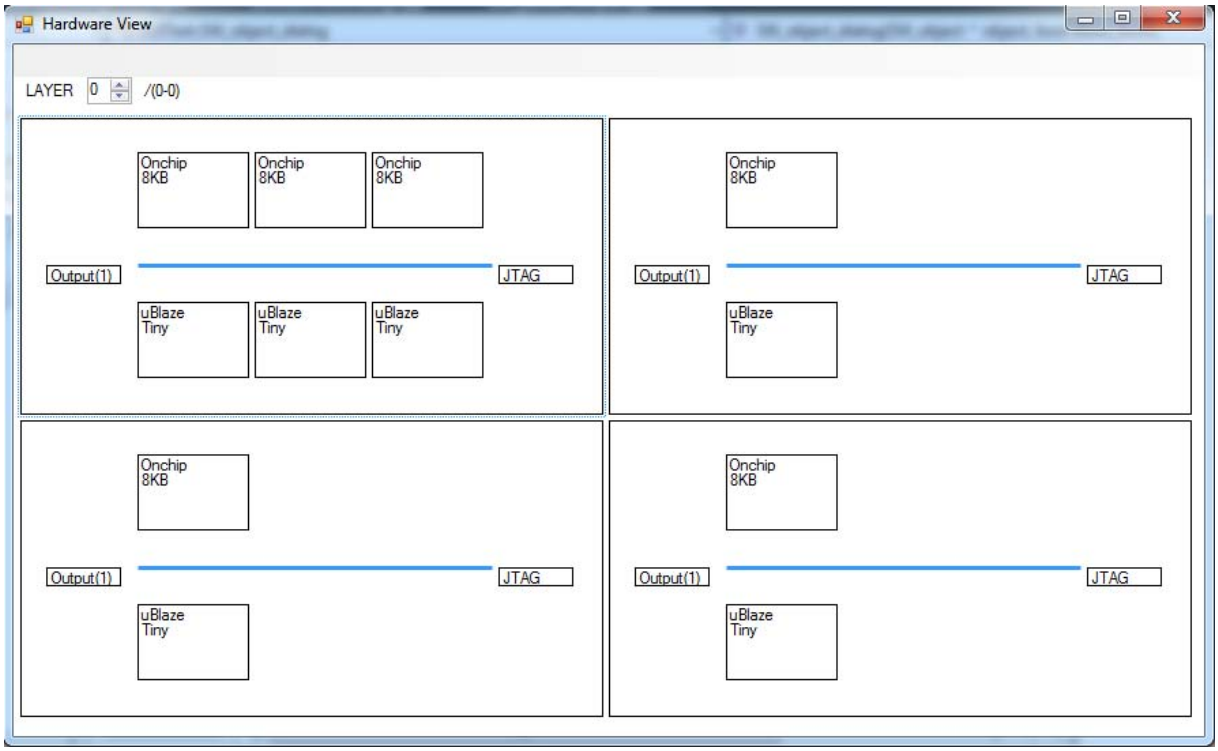


Figure 4.2. Several CPUs in one node

4.2 Hardware Node Data Entry View

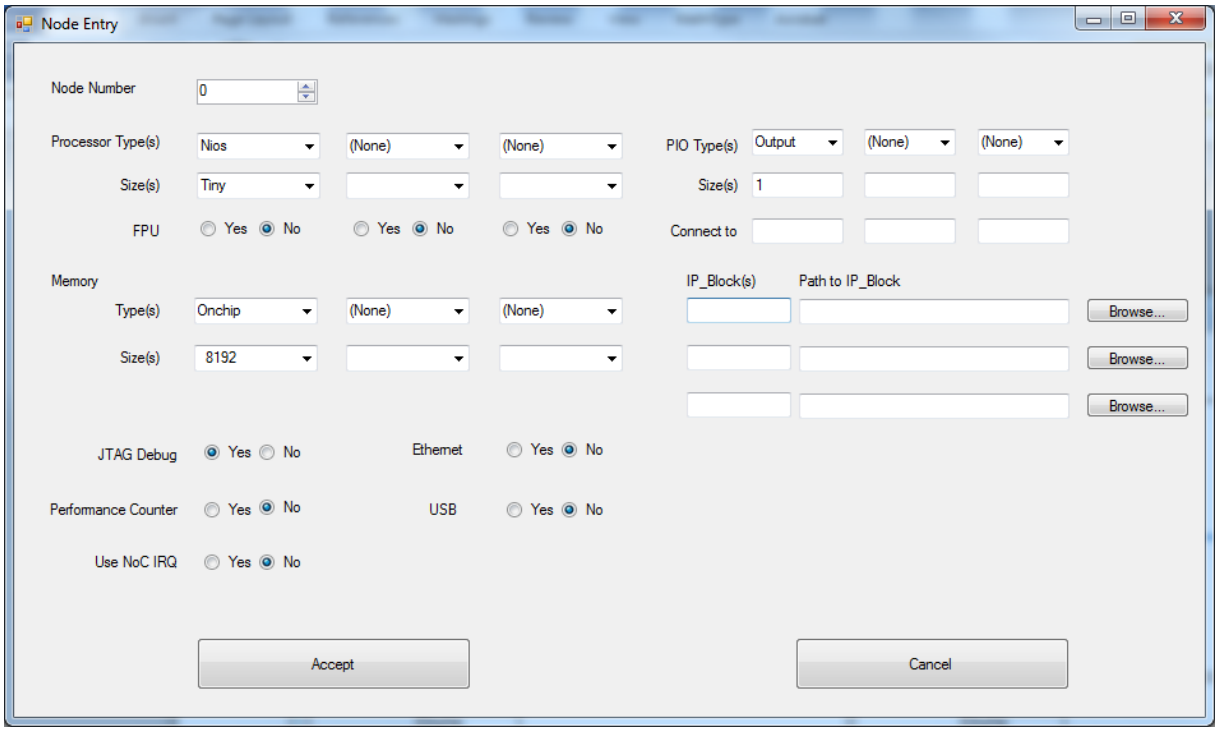


Figure 4.2. Hardware Node Entry

In the Hardware Node Data Entry, you configure the computation tile you selected in the Hardware View. You change the target Node number here if you selected the wrong tile. Each tile has one single common bus and one NI. All units on the bus use the same NI.

The number of entries in this view is restricted to three for two reasons. 1) Using more than three processors in the same tile a node does not make much sense if you are building a NoC system with tiles. 2) because of display reasons. Using more than three CPUs and memory entities per node does not display well in the Hardware View. The NoC System Generator itself does not have any restrictions, only the GUI. If you want to have more nodes, you have to edit the system xml description directly.

CPU

Processor Type. Here you select the type of soft processor to use for the Node. The value shown here depends on the target technology chosen for the target FPGA. If it is Altera, the value Nios is shown, if it is Xilinx, the value MicroBlaze will be shown. If your FPGA board support Hardcore ARM CPUs, you can select them if you are in Node 0. If you want to turn on the second ARM core, select an ARM CPU also on CPU1.

Size. Here you select the size of the processor type you have chosen. Three typical configurations sizes are available: Tiny, Small, and Large. The Tiny version does not use any cache, and is configured to be as small as possible. The Small version is using some acceleration, like Hardware multiplier and small caches, while the Large version is using all available hardware acceleration units and larger caches. If you select a Hardcore processor, this parameter has no effect.

FPU. Here you select if you want support of an external Floating Point Hardware Unit or not. If you select a Hardcore processor, this parameter has no effect.

Memory

Memory Type. Here you select the type of memory to use for the Node. Onchip memories are not shared, but rather connected to each CPU as scratchpad memories containing the boot code for that particular processor. SRAM memories and DRAM memories are shared memories and can be accessed by all CPUs in the node. If you select a Hardcore processor, one DRAM shared by all nodes is always generated. Currently, all memory is local non-shared memory. Shared-memories are not supported yet, but the plan is to allow CPUs to share some memories on the local bus in future versions of NSG.

Size. Here you select the size of the memory.

I/Os

PIO Type. Here you select whether the IO is an input, output or bidirectional.

Size. Here you select the width of the PIO.

Connect to. Here you select which pin the IO(s) should be connected to. The value is a comma-separated list of named pins. The IOs are assigned from bit 0 and upwards to the names in the list in the order they appear.

IPs

IP Block. Here you select which IP to use. The name of the IP Block is used to find the tcl-files to instantiate and connect the IP to the bus. If you have selected Altera Quartus as target, the tcl-files have to comply to the Altera standard. If you have selected Xilinx Vivado as target, the tcl-

files have to comply with the Xilinx standard. Exactly what are needed and how they lshould look internally is described in detail in section 9.4.

Path. Here you specify the path to the directory where the IP is stored. Depending on the target technology, the IP directory will have a different structure.

JTAG Debug unit.

Here you select whether you want a JTAG Debug unit or not. This option is valid for Altera designs, and is needed to communicate with the core and download programs. In Altera, one JTAG unit is instantiated for each core. In Xilinx, only one core is instantiated, but with one debug connection to each core. The debug core is a MicroBlaze Debug Module (MDM) unit, but works more or less in the same way as the JTAG debug units used for Altera Nios processors.

Performance Counter.

The performance counter is a remnant from the days when an external unit was needed in order to measure past time (clock steps) in the design. This function is nowadays incorporated in a register inside the Nostrum NI. However, for User NoCs, the ability to measure time by reading a register is not guaranteed to be there, so therefore this function is still there.

Ethernet.

Here you select whether the Ethernet IP should be used or not. **This function has yet to be included in the NSG. The function is not enabled in the beta release.**

USB.

Here you select whether the USB interface IP should be used or not. **This function has yet to be included in the NSG. The function is disabled in the beta release.**

ICAP

Here you select whether the ICAP interface should be used or not. This is a Xilinx Vivado only component, used for fault-tolerant designs. **This function has yet to be included in the NSG. The function is disabled in the beta release.**

5 The Software View

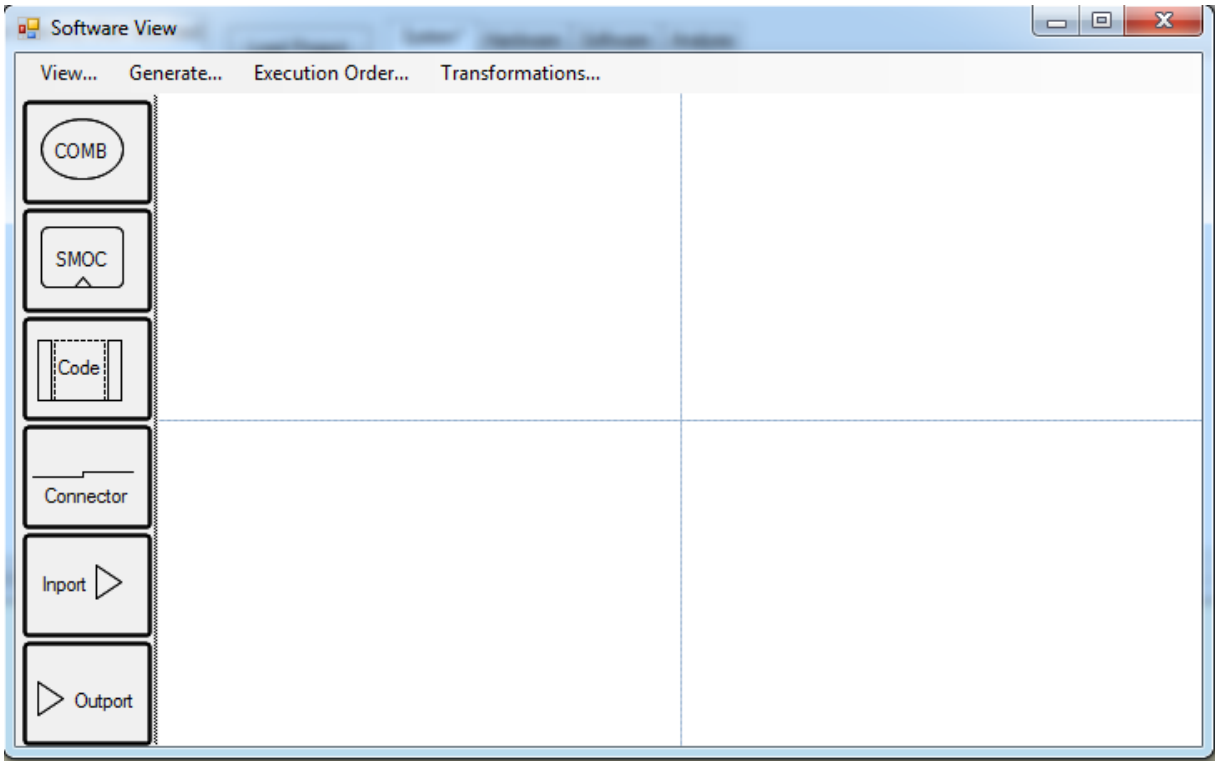


Figure 5.1.1. Default Software View

There are four drop-down menus in this view. They are described in detail in section 5.2 Drop Down on page 18:

1. View...
2. Generate...
3. Execution Order...
4. Transformations...

5.1 Adding/Editing Software/Components

In the software view, you can program/edit software components and assign them to CPUs in the NoC. You also assign communications channels between the software components. The NoC System Generator will generate a bare-metal OS template for you, and schedule your software components according to the System specification.

To add a software component, left-click on one of the component listed on the left. You place it in one of the squares to the right by positioning the cursor in the square and fix it by left-clicking again. To re-position it, right-click on the component and select move. Then left-click in the square where it should be. The name of the component is indicated above the function.

There are two types of software components to choose from. The type indicates which Model-of-Computation it uses, i.e., how and when the software is executed by the Bare-metal OS (BOS).

- **COMB** components are combinatorial functions that don't have a sense of time. They execute as fast as possible, one after the other, in the order they are listed in the square, at any time when all of their inputs are ready. The BOS system does not check the inputs before calling the software component, this has to be done in the software component itself. The order is also reflected by the order of the components in the system description.

- **SMOC** components are real-time functions that are executed only whenever there is a heartbeat, irrespective if the inputs have changed or not. Also, the recv buffer is pointing to the data received in the previous heartbeat, i.e., the input is delayed one heartbeat cycle, so they behave in essence like synchronous hardware, with a D-flipflip that is "clocked" on the rising edge of the heartbeat clock. The components are executed in the order they appear in the square. The order is also reflected by the order of the components in the system description.

You can change back and forth between SMOC and COMB operation in the edit window.

- The **Dropbox** function is used for importing functions from a third-party tool. Currently, the only tools supported by the NSG GUI tool is Simulink import and ForSyDe import. The default MoC operation is SMOC, but it can be changed later in the edit window after the function has been imported.

5.1.1 Connecting Software Components

You connect components by selecting the connector symbol and then fasten it by clicking on the left or right side of a software component. Inputs to the component are situated on the left-hand side and outputs are situated on the right-hand side of the component. You can only connect one input to one output or vice versa. When connecting, the tool traces your mouse position until you click on the target component.

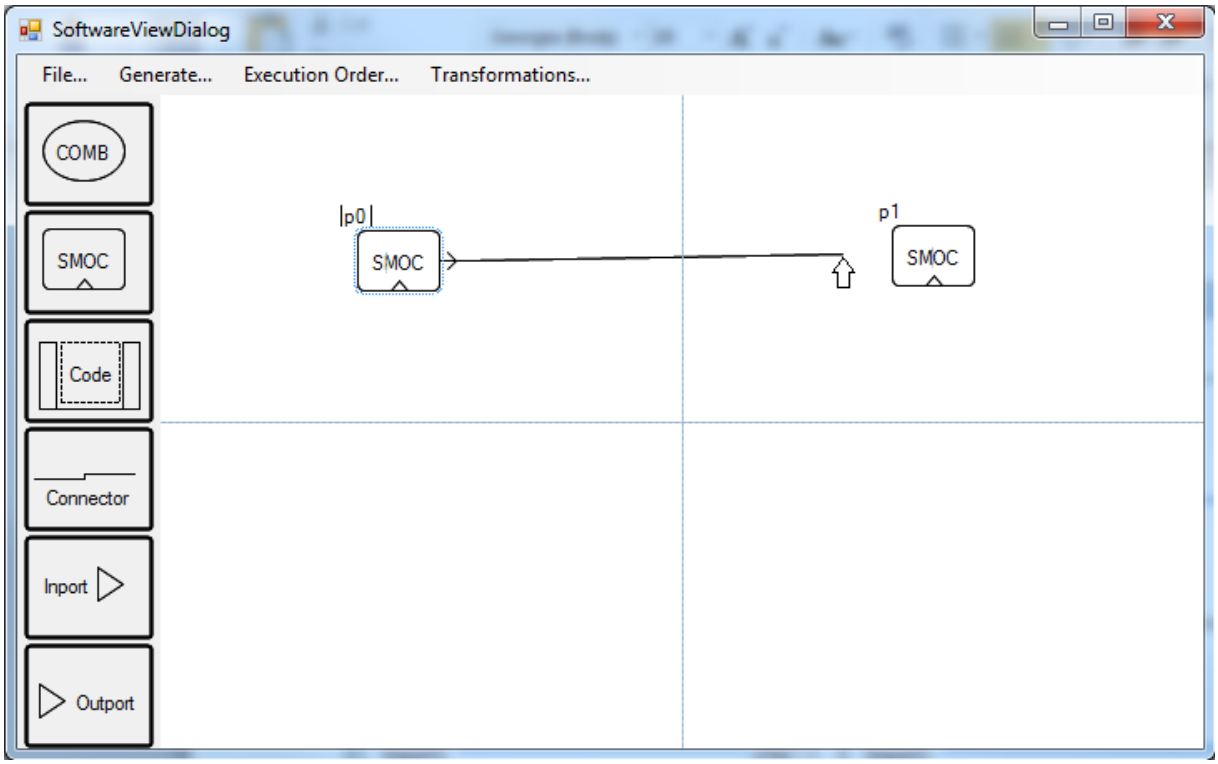


Figure 5.1.2. Adding Software Processes

- **Inport** functions are supposed to be associated with a software component reading from an input device. The inport cannot have any internal input connections. **This function/symbol is currently not implemented.**

- **Outport** functions are supposed to be associated with a software component writing to an output device. The outport cannot have any internal output connections. **This function/symbol is currently not implemented.**

If you select a component and right-click, a selection menu appears:

1. Select/Move
2. Edit
3. Clone
4. Delete

- **Select/Move -** let's you pick up and move around the node graphically and place it into other squares.

- **Edit** - let's you edit the properties of the Software Component in the Software Component Edit Window.

- **Clone** – let's you create a copy of the selected node. A new software component is created and placed into the current square. It has exactly the same code but no inputs, nor any outputs as the Software Component you copied from. You just have to connect it.

- **Delete** – deletes the selected Software Component, and all its connections.

If the component is Dropbox, one further item is seen in the menu, between select/move and edit:

- **Import** – let's you access the import menu for importing software from a 3rd party tool. The current supported tools are Simulink and ForSyDe.

5.1.2 Different CPUs

In case you have different CPUs in the same node, the CPUs are separated by a vertical line. The software component will be assigned to the CPU that resides in that place. Software components that are residing on the same CPU are displayed in schedule order, i.e., the software components of the same type will be scheduled in the order they appear on the screen. Software components residing on different CPUs will be displayed in the CPU box indicating the CPU they are associated to.

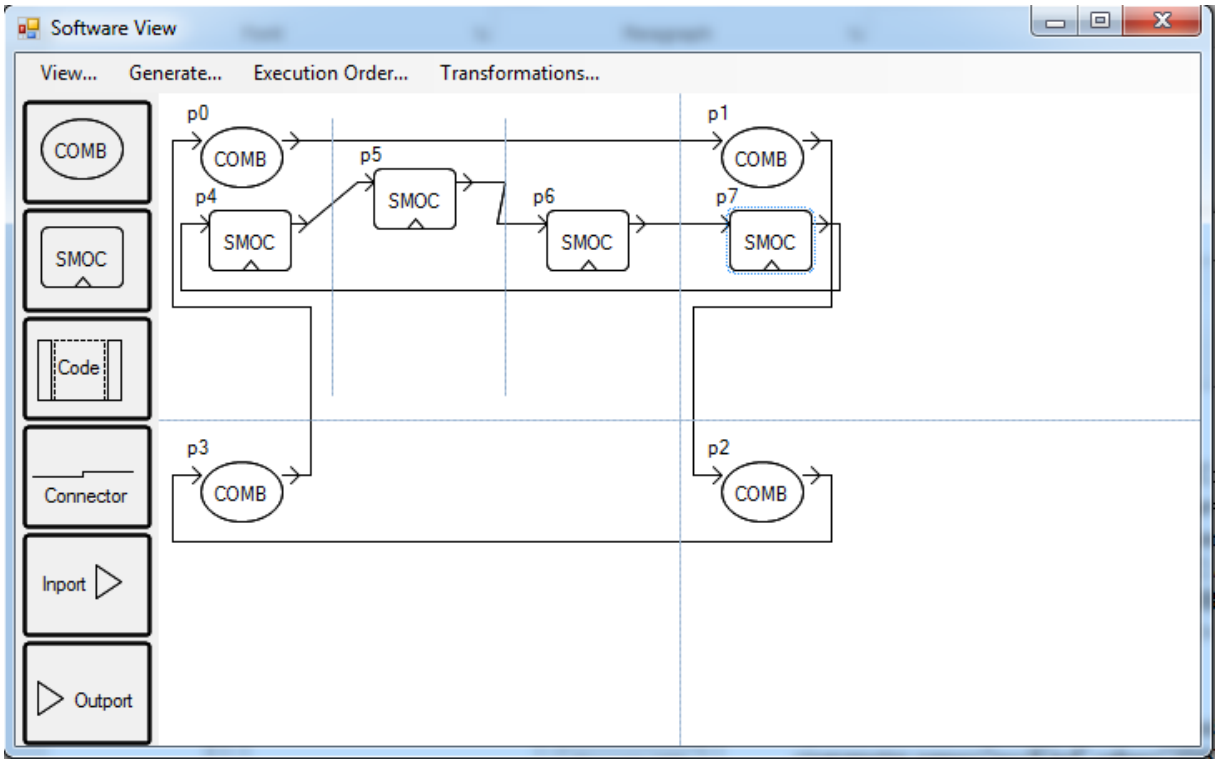


Figure 5.1.3. Example of Software residing on multiple CPUs inside a node.

5.1.3 3D View

In the Software view, there is no explicit Layer selection in case you have selected a 3D NoC. Instead, the node layers are superimposed on one another, so the node right above is in the same box as the one below. To visualize the difference from the software point of view, software components residing in different node layers are displayed with a 45 degree offset down to the right in the same box.

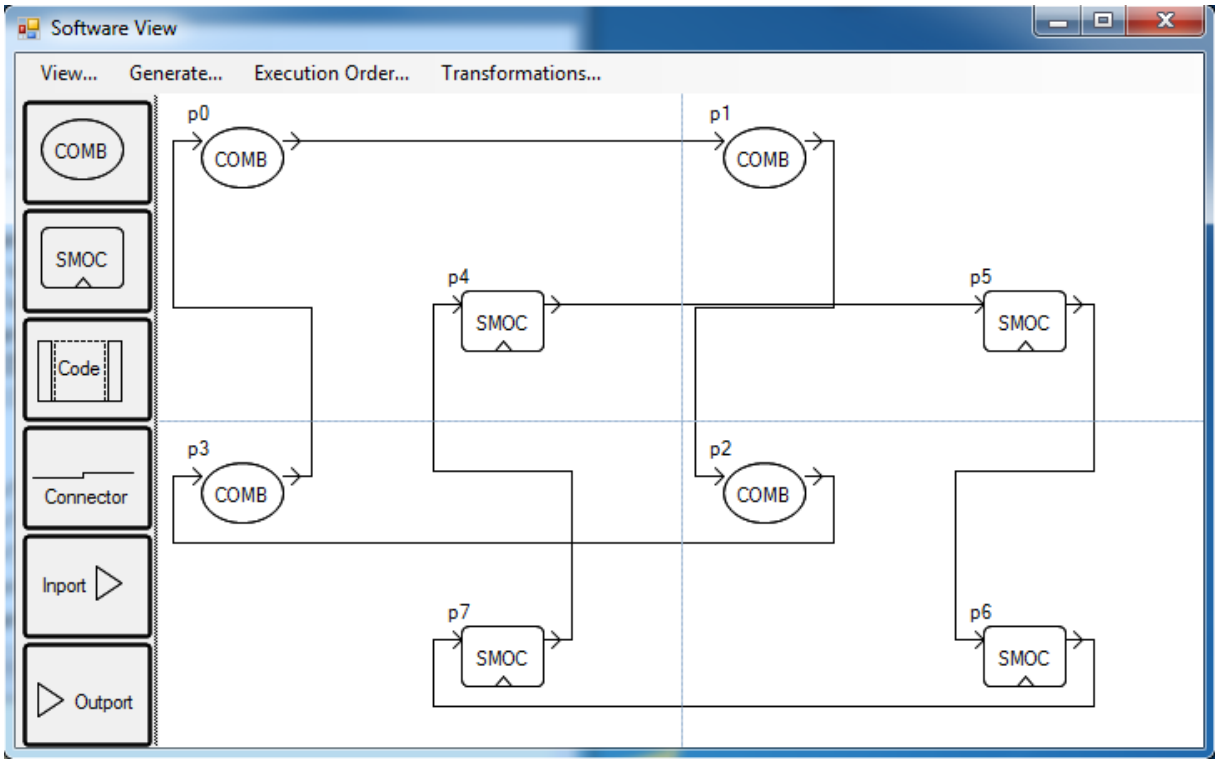


Figure 5.1.4. Example of a 3D Software View with 8 nodes.

5.1.4 Mixed 3D View with multiple CPUs.

Mixing 3D with multiple CPUs is also possible. However, currently it does not display well, since both offsets caused by the different CPU boxes and the 3D view applies. This makes the software components tend to end up outside the established boxes, see figure 5.4.

Also, it is important to remember that the CPUs in one node are sharing the RNI of the NoC node, i.e., access to the NI will be limited one node at a time.

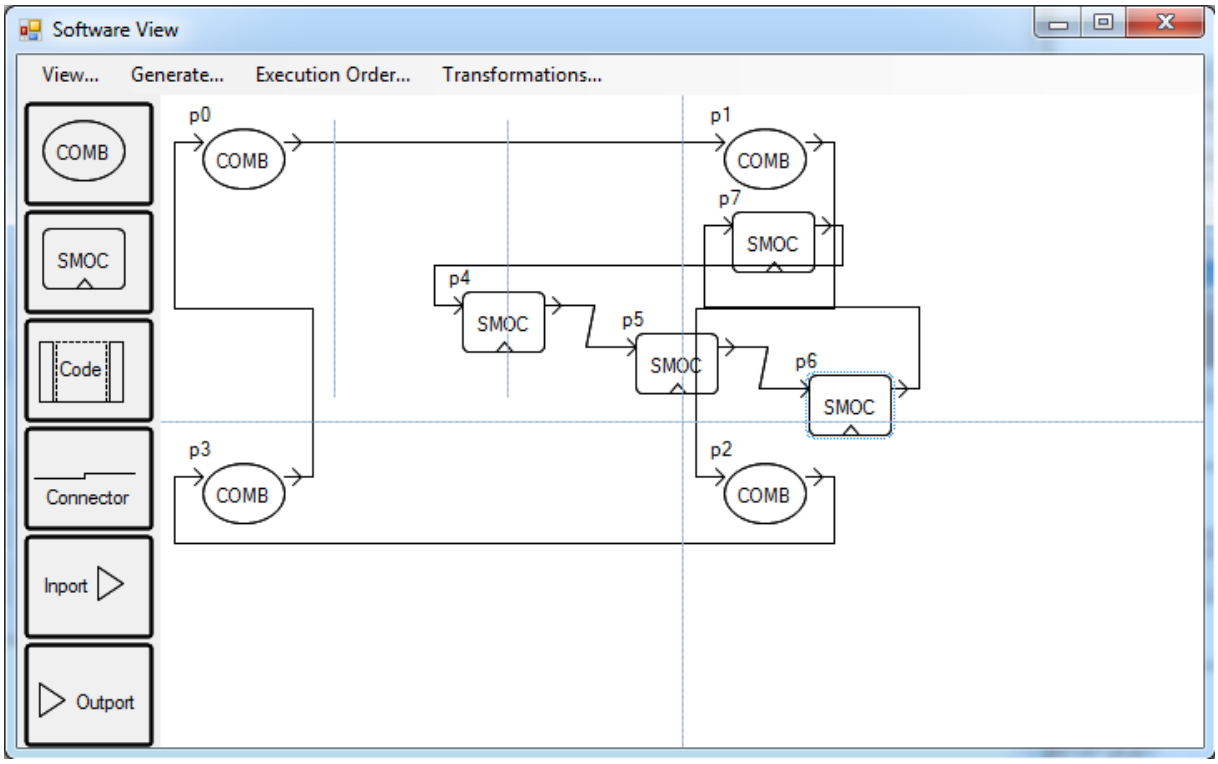


Figure 5.1.5. Example of Mixed 3D and multi CPU View. The software components on the CPUs of the second layer (node 4, CPU 0-2) tend to spill over into the second quadrant of the first layer (node 1).

5.2 Drop Down Menus

There are four drop-down menus in this view. They are:

1. View...
2. Generate...
3. Execution Order...
4. Transformations...

- **View...** In this menu there is currently only one selection: Redraw. It redraws the content of the current display window.

- **Generate...** In this menu there are two selections, but only one is currently active:

    - **Process Files...** This option renames and all #include-files in the c-code of the software components that are target technology sensitive, and change them to the correct name in the chosen target technology.

    - **ForSyDe SystemC...** This option generates ForSyDe SystemC-code from the c-code of the software components, for enabling an early transaction-level simulation of the system. **NB! This option is not active.**

- **Execution order...** This option opens the Schedule Processes view. In the view, you can select in which order the software components are executed. The higher number you put in the process order, the later it is scheduled in the window. If you then press the **Node Nr** bar, the **CPU Nr** bar, the **Process Order** bar, **WCET** bar, or **WCCT** bar, the lines in the view are reordered according to the selected criteria. The order in which the software components are listed in the Schedule Processes window, is the order they will appear in the Software View, after you have pressed the **Accept** button.

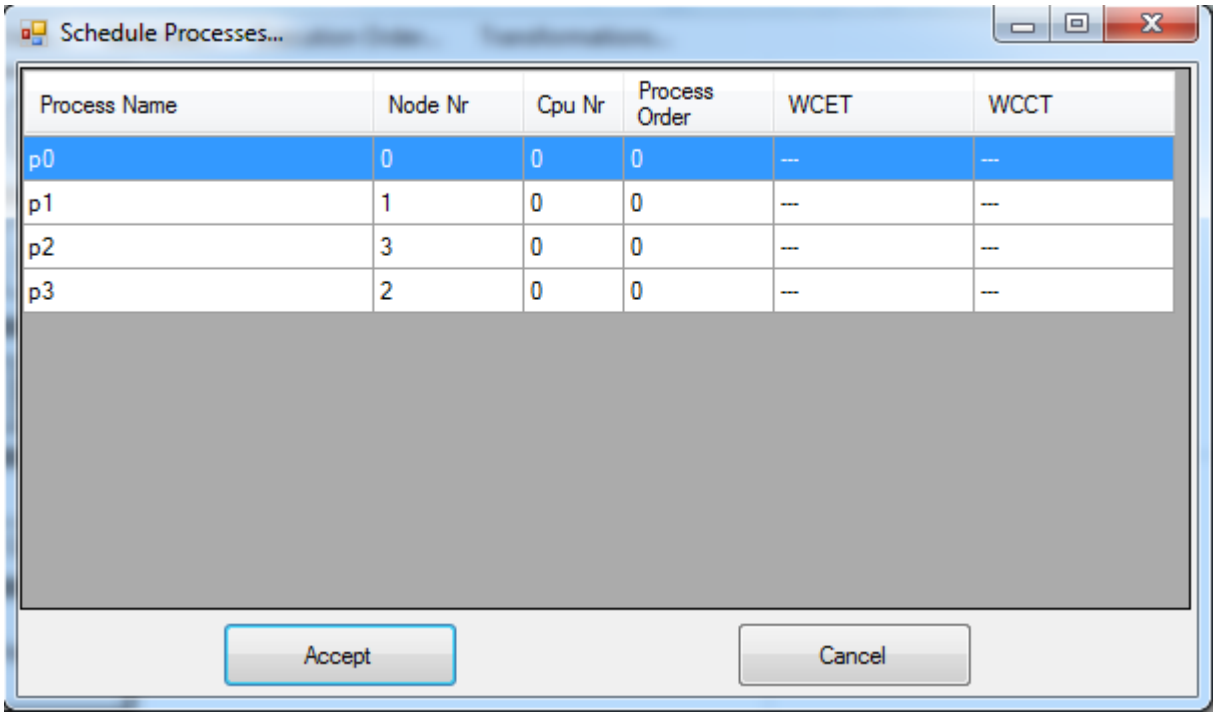


Figure 5.2.1. Schedule Processes Window.

- **Transformations...** In this Menu, you have three selections. Only one selection is currently active:
    - **Split...** This option is currently disabled. It is supposed to allow you to split a software component into several smaller ones, and redistribute the C-code inbetween them according to the designers directive.
    - **Merge...** This option is currently disabled. It is supposed to allow you to group and merge software components into bigger ones.
    - **Clone Process...** This option opens the Clone Process Dialog View. It allows you to make clones of the processes. It takes the C-code of the software component, copies it, and renames and replaces all references to the old components inputs and outputs to new ones.

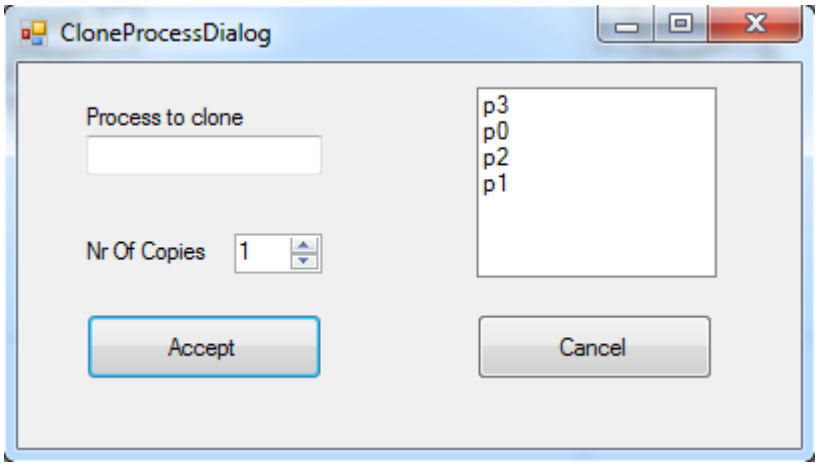


Figure 5.2.2. Clone Process Dialog Window

5.3 Software Component Edit Window

If you double-click on a component in a square, or right-click and select edit, the Software Component Edit Window appears.

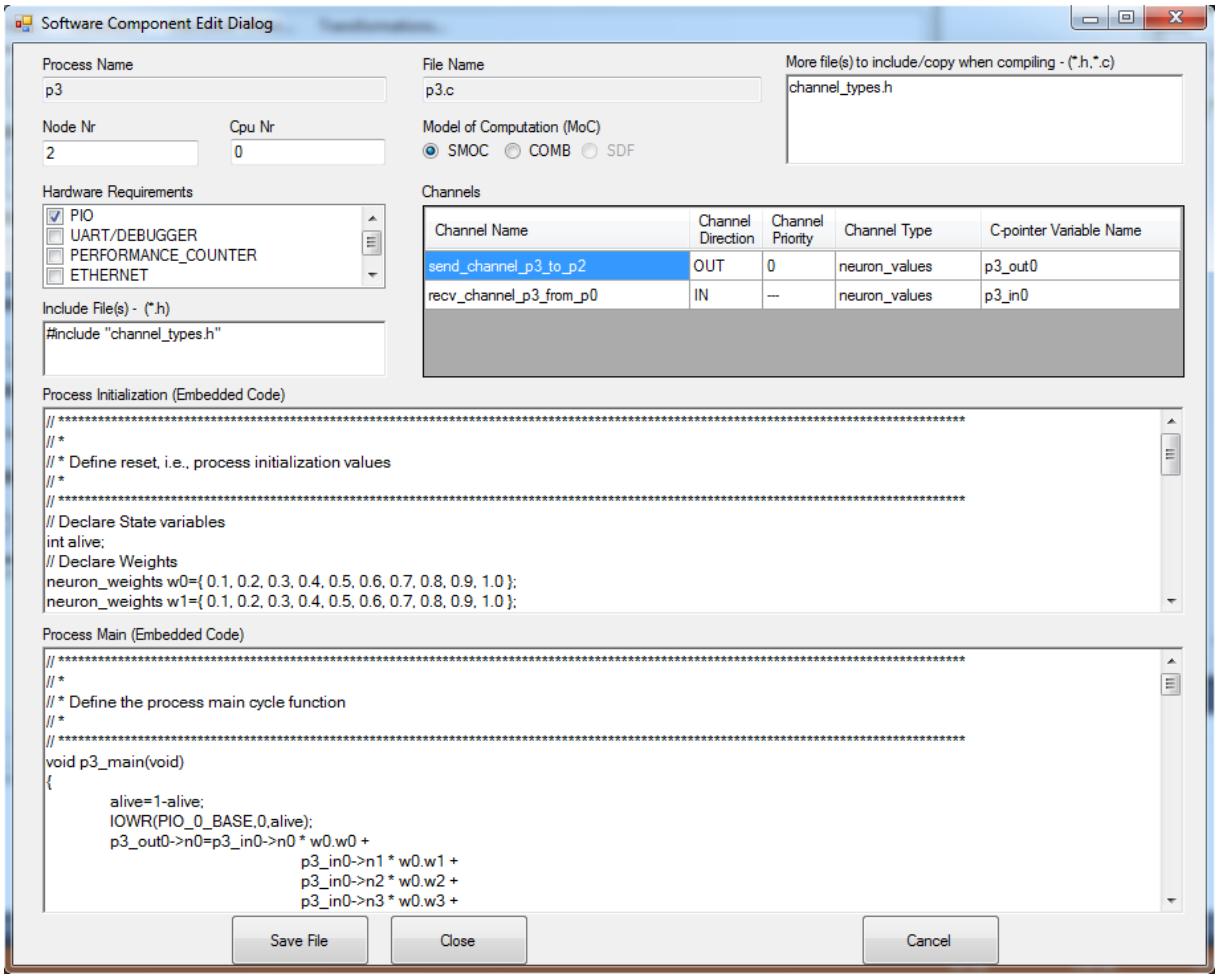


Figure 5.3.1. Software Component Edit Window

In the Software Component Edit Window, the details of the software component are listed. Here you can change the execution behavior by changing the MoC from SMOC to COMB or vice versa. The **Node Nr** and **Cpu Nr** where the software is placed can also be edited.

In the box Hardware Requirements, you can tick off which hardware units, and the corresponding device driver (.h-files) that the software component needs to function properly. The tool knows the proper name of the device driver .h-file needed for the chosen target technology, and it can detects and translate between when porting between technologies when it writes the software component's c-code to disk. This is also used as a constraint during scheduling. A software component cannot be placed on a node that does not have the corresponding hardware support. Else will an error be reported.

In the Include Files box, any other .h-file needed for the appropriate function of the Software Component is listed as normal include statements. These are not translated between technologies, but are assumed the same. If you use special datatypes or structs for the communication channels, the .h-file defining these should be listed here. These files should also be listed in the box **File(s) to include when compiling**.

In the **Channel Box**, the communication channels used by the software component is listed. The order of the channels comes from the order they are listed in the system description. Outputs are listed first, inputs later. The direction, data type, priority and the c-pointer used in the code to access the NoCs communication buffers can be edited here. To the c-pointer's name, a suffix should be added. <pointer_name>_out<nr> is used for outputs, and <pointer_name>_in<nr> is used for inputs. The <nr> field indicates the order in which it is listed in the system description, and thus in the graphical drawing of software component in the software view window. The name of the communication channel should not be changed; it is used by the tool during the compilation process. The name of the channel follows a naming standard:

Outgoing channels	send_channel_<from_process>_to_<target_process>
Incoming channels	recv_channel_<to_process>_from_<source_process>

Thus, the first of the process names should always be the process you are working in. If you want to add more channels, you have to connect them in the Software View Window first.

5.3.1 Process Initialization Window.

In the Process Initialization Window, the Embedded Code that should be executed during system boot/reset is written. Here you also declare locally shared variables that your software component needs to function properly.

The declaration of the init function follows a naming standard:

```
void <process_name>_init(void)
```

If you name it something else, the software template generated for the Bare-metal OS (BOS) will use the wrong function names.

SMOC functions should commit the reset value of the function into the transmission channel using the command

```
NOC_SEND(<c_pointer_variable_name>);
```

5.3.2 Process Main

In the Process Main Window, you write the Embedded Code that should be executed on every time the software component is called.

The declaration of the main function follows a naming standard:

```
void <process_name>_main(void)
```

If it is an SMOC function, the process is called exactly once every heartbeat, so no protection mechanism is needed. You send the next value by committing it to the transmission channel using the *NOC_SEND* command.

However, if it is a COMB function, its main function is executed as often as possible. Therefore, the inputs to COMB functions are stored in an input queue, to prevent that its inputs do not change value while it is executing. When the COMB function is finished executing, it should change its input buffer by executing the command:

```
NOC_RNI_CLEAR(NOC_RNI_BASE,<channel_name>);
```

This changes the message pointer in the queue to point on the next value.

The queue also contains a flag that indicates if a message is valid or not. This flag should be used to prevent the COMB from calculating the same value again. Therefore, the first thing that should be checked in a COMB function, is if all the data in COMB function's input channels have arrived yet. If so, the main body should be executed. If not, the main body should be skipped:

```
int input1=NOC_RNI_RECV_CHANNEL_STATUS(<NoC_base>,<channel_1>);
int input2=NOC_RNI_RECV_CHANNEL_STATUS(<NoC_base>,<channel_2>);
int input3=NOC_RNI_RECV_CHANNEL_STATUS(<NoC_base>,<channel_3>);
If (input1=1) and (input2=1) and ... (inputN=1)
{
  // Call/Execute Comb Main body Functionality
}
```

5.4 Generated Software – The BOS system

The NoC Generator generates the directory structure for all nodes under the Target Directory. The top level is call the *Software* directory. Under the Software directory, one directory for each node and processor is created, with the naming convention *Node_<node nr>_<cpu nr>*, and copies all files related to the software components of that Node to the Node directory. It generates the configuration information needed to setup the system in the software_configuration.h file, together with the device drivers needed to run the system, in the file named <noc_name>.h. It also generates a software template of a Bare-metal Operator System, called the BOS, in the file mixed_MoC_main.c, which invokes the software components in the right order so that the software behaves according to its Model of Computation (MoC).

5.4.1 Synchronous MOC BOS template

```
…
extern void p0_init(void);
extern void p0_main(void);

int main(void)
{
   // Send Channel, TDN, Target z, y, x, Dest PID, Source PID
   NOC_RNI_SEND_CHANNEL_INFO(NOC_RNI_BASE,send_channel_p0_to_p1,0,0,0,1,p1_pid,p0_pid);
   // Recv Channels - Dest MoC Type, Use Irq, Dest PID, Source PID
   NOC_RNI_RECV_CHANNEL_INFO(NOC_RNI_BASE,recv_channel_p0_from_p3,COMB,0,p0_pid,p3_pid);
   // Wait for first Heartbeat (=GlobalSynchronizer Flag)
   // to ensure that all SMOC processes initializes their output queues in the same sync frame
   int synchronize=0;
   while(synchronize==0)
   {
      synchronize=NOC_RNI_READ_SYNCHRONIZER_FLAG(NOC_RNI_BASE);
   };
   // Initialize all SMOCs
   P0_init();
   // Initialize all COMBs
   // Clear SMOC Synchronizer flag
   NOC_RNI_CLEAR_SYNCHRONIZER_FLAG(NOC_RNI_BASE,-1);
   while(1) // Loop forever
   {
      synchronize=0;
      while(synchronize==0)
      {
         synchronize=NOC_RNI_READ_SYNCHRONIZER_FLAG(NOC_RNI_BASE);
      // Execute COMBs' main function while waiting for the next Heartbeat
      };
      // Execute SMOCs' main function
      p0_main();
      NOC_RNI_CLEAR_SYNCHRONIZER_FLAG(NOC_RNI_BASE,-1);
   };
   return 0;
};
```

Figure 5.4.1. Genereted BOS for SMOCs

5.4.2 Asynchronous MOC BOS template

```
…
extern void p0_init(void);
extern void p0_main(void);

int main(void)
{
   // Send Channel, TDN, Target z, y, x, Dest PID, Source PID
   NOC_RNI_SEND_CHANNEL_INFO(NOC_RNI_BASE,send_channel_p0_to_p1,0,0,0,1,p1_pid,p0_pid);
   // Recv Channels - Dest MoC Type, Use Irq, Dest PID, Source PID
   NOC_RNI_RECV_CHANNEL_INFO(NOC_RNI_BASE,recv_channel_p0_from_p3,COMB,0,p0_pid,p3_pid);
   // Wait for first Heartbeat (=GlobalSynchronizer Flag)
   // to ensure that all SMOC processes initializes their output queues in the same sync frame
   int synchronize=0;
   while(synchronize==0)
   {
      synchronize=NOC_RNI_READ_SYNCHRONIZER_FLAG(NOC_RNI_BASE);
   };
   // Initialize all SMOCs
   // Initialize all COMBs
   p0_init();
   // Clear SMOC Synchronizer flag
   NOC_RNI_CLEAR_SYNCHRONIZER_FLAG(NOC_RNI_BASE,-1);
   while(1) // Loop forever
   {
      // Execute COMBs' main function while waiting for the next Heartbeat
      synchronize=0;
      while(synchronize==0)
      {
         synchronize=NOC_RNI_READ_SYNCHRONIZER_FLAG(NOC_RNI_BASE);
         p0_main();
      };
      // Execute SMOCs' main function
      NOC_RNI_CLEAR_SYNCHRONIZER_FLAG(NOC_RNI_BASE,-1);
   };
   return 0;
};
```

Figure 5.4.2. Genereted BOS for COMBs

5.4.3 Mixed MOC BOS template

```
…
extern void p0_init(void);
extern void p0_main(void);

int main(void)
{
   // Send Channel, TDN, Target z, y, x, Dest PID, Source PID
   NOC_RNI_SEND_CHANNEL_INFO(NOC_RNI_BASE,send_channel_p0_to_p1,0,0,0,1,p1_pid,p0_pid);
   // Recv Channels - Dest MoC Type, Use Irq, Dest PID, Source PID
   NOC_RNI_RECV_CHANNEL_INFO(NOC_RNI_BASE,recv_channel_p0_from_p3,COMB,0,p0_pid,p3_pid);
   // Wait for first Heartbeat (=GlobalSynchronizer Flag)
   // to ensure that all SMOC processes initializes their output queues in the same sync frame
   int synchronize=0;
   while(synchronize==0)
   {
      synchronize=NOC_RNI_READ_SYNCHRONIZER_FLAG(NOC_RNI_BASE);
   };
   // Initialize all SMOCs
   p0_init()
   p1_init()
   // Initialize all COMBs
   p2_init();
   p3_init();
   // Clear SMOC Synchronizer flag
   NOC_RNI_CLEAR_SYNCHRONIZER_FLAG(NOC_RNI_BASE,-1);
   while(1) // Loop forever
   {
      // Execute COMBs' main function while waiting for the next Heartbeat
      synchronize=0;
      while(synchronize==0)
      {
         synchronize=NOC_RNI_READ_SYNCHRONIZER_FLAG(NOC_RNI_BASE);
         p2_main();
         p3_main();
      };
      // Execute SMOCs' main function
      p0_main();
      p1_main();
      NOC_RNI_CLEAR_SYNCHRONIZER_FLAG(NOC_RNI_BASE,-1);
   };
   return 0;
};
```

Figure 5.4.3. Genereted BOS for Mix of COMBs and SMOCs

6 System View

In the System View, you can see four pages, marked System, Hardware, Software and Analysis. The first three has been implemented. The fourth one displays a mockup of a future analysis scenario.

1. **System View** – Contains the XML description of the system. A user can edit this window manually, change it and copy&paste text here.

   NB! If the system description is buggy, the code cannot be loaded. To correct a buggy system description code, you have to correct it using Notepad or some other text editor first.

2. **Hardware View** – Contains an area estimate of the Hardware Resources selected in the description. The estimate is based on the synthesised results of various design properties and is fairly accurate.

3. **Software View** – Contains a list of the Software associated with each NoC node, together with back-annotated sizes of the code size after compilation at the target node, including BOS.

4. **Analysis View** – Contains a mockup of a future analysis scenario. It should display WCET, and BCCT times of the Software Components used in the system.

6.1 System Description

```
<?xml version="1.0" encoding="UTF-8"?>
<system name="Asynchronous_demo" >
  <parameter name="targetDirectory" value="D:/FPGA_Designs/Asynchronous_demo" />
  <parameter name="targetManufacturer" value="Xilinx" />
  <parameter name="targetManufacturerVersion" value="2016.3" />
  <parameter name="targetToolPath" value="C:/XILINX" />
  <parameter name="boardType" value="ZEDBOARD" />
  <parameter name="PLL_ratio" value="1:1" />
  <hardware>
    <connect node="0" unit="(pio,0)" to="{ledg[0]}" />
    <connect node="1" unit="(pio,0)" to="{ledg[1]}" />
    <connect node="2" unit="(pio,0)" to="{ledg[2]}" />
    <connect node="3" unit="(pio,0)" to="{ledg[3]}" />
   <ip_block type="Xilinx" name="axi_uartlite" version="2.0"   directory="D:/NSG/Examples/Asynchronous_demo//Xilinx">
     <parameter name="bus" value="{axi,32,s }" />
     <parameter name="use_noc_irq" value="no" />
   </ip_block>
   <noc>
      <parameter name="nocType" value="Mesh" />
      <parameter name="nocKind" value="2DNoC" />
      <parameter name="nrofCols" value="2" />
      <parameter name="nrofRows" value="2" />
      <parameter name="LayoutMethod" value="Floating" />
      <parameter name="switchType" value="Nostrum_TDN" />
      <parameter name="rniType" value="Heartbeat" />
      <parameter name="NrOfRecvChannels" value="Minimize" />
      <parameter name="NrOfSendChannels" value="Minimize" />
      <parameter name="FrameSize" value="4" />
      <parameter name="FlitWidth" value="32" />
      <parameter name="FlitInjectionRate" value="1:4" />
      <parameter name="Heartbeat" value="1 Hz" />
      <parameter name="ResetTime" value="1 ms" />
    </noc>
    <node nr="0" mem_size="8192" jtag="yes" perf_counter="no" pio="{o,32}" noc_irq="no" cpu="{uBlaze,tiny}" />
    <node nr="1" mem_size="8192" jtag="yes" perf_counter="no" pio="{o,32}" noc_irq="no" cpu="{uBlaze,tiny}" />
    <node nr="2" mem_size="8192" jtag="yes" perf_counter="no" pio="{o,32}" noc_irq="no" cpu="{uBlaze,tiny}" />
    <node nr="3" mem_size="8192" jtag="yes" perf_counter="no" pio="{o,32}" noc_irq="no" cpu="{uBlaze,tiny}" />
  </hardware>
  <software>
    <parameter name="Repository" value="D:/NSG/Examples/Asynchronous_demo" />
    <process name="p0" node="0" cpu="0" moc="Asynchronous" sources="{p3}" targets="{p1}" files="{p0.c}" />
    <process name="p1" node="1" cpu="0" moc="Asynchronous" sources="{p0}" targets="{p2}" files="{p1.c}" />
    <process name="p2" node="3" cpu="0" moc="Asynchronous" sources="{p1}" targets="{p3}" files="{p2.c}" />
    <process name="p3" node="2" cpu="0" moc="Asynchronous" sources="{p2}" targets="{p0}" files="{p3.c}" />
  </software>
</system>
```

Figure 6.1.1. System Description

The System description is written in an XML-like language, which has been extended with the '#'-sign for marking a comment line, instead of the complicated way for describing comments that XML normally uses. It is a textual description of the system, which summarize all the choices you can specify in the GUI in a condensed way. You can change the system description directly, but you have to save it and reload it for the changes to be seen in the GUI.

First, the system name is given. Then comes a set of parameters that describe how the system should be synthesized:

- targetDirectory – tells which where to store the generated system and software files
- targetManufacturer – tells NSG which tool should be used as backend. Script generation will be adapted accordingly.

- targetManufacturerVersion – tells NSG which tool specific version that should be used, so that it can adapt script generation accordingly. **NB! Not all tool versions are supported**.

- targetToolPath – tells NSG in which directory it should look for the backend tool.

- boardType – tells NSG which board should be used as backend so that it can generate a proper pin placement.

- PLL_ratio – tells NSG if a PLL should be used, and in that case, what frequency ratio it should have.

6.1.1 Hardware section

Next comes a description of the hardware configuration. It starts with connection statements, that tell NSG how the PIOs' pins should be connected. If the system should contains IP blocks, they are declared here. Third comes the NoC configuration. It starts with a list of parameters that tells NSG how the NoC System should look like. Finally, the node configuration description is given.

6.1.1.1 Connection statements

The connection statements are supposed to be used to connect internal pin on the hw units in the nodes to each other. The only supported connection statement for the moment, is to connect the PIOs' pins to the external pins on the chip.

6.1.1.2 IP block declarations

If the system uses any IP blocks in the nodes, they need to be declared before they are used. The name of the IP block together with the path to where the files needed to import and generate them are given, together which type of bus the IP uses. The bus supported is AXI in case of Xilinx designs, and Avalon in case of Altera designs, and if the IP should be connected to the normal bus as a bus master or a bus slave. The alternative is connect the IP directly to the NoC. In that case, the DAP (direct access port) of the RNI should be used. Supported bus sizes are 32 for Altera and Xilinx designs, and 64 for DAP port designs. Multiple buses are allowed.

6.1.1.3 NoC configuration parameters

The NoC configuration tells how the VHDL code for the NoC template is stored, and which parameters to use to configure it. The NoC configuration parameters are:

- nocType – it tells NSG which topology to use for the NoC. The corresponding VHDL directory should contain the templates for this NoC type.

- nocKind – it tells NSG which dimensionality to use for the NoC. The corresponding VHDL directory should contain the templates for this NoC type.

- nrofCols – it tells NSG how many columns that the NoC should contain. If the dimensionality is 1D, this is the only parameter that needs to be given.

- nrofRows – it tells NSG how many rows that the NoC should contain. If the dimensionality is 1D, setting this parameter to anything other than 1 will generate an error. If the dimensionality is 2D, this parameter together with nrofCols is used.

- nrofLayers – it tells NSG how many 2D layers that the NoC should contain. If the dimensionality is less than 3D, setting this parameters to anything other than 1 will generate an error.

- LayoutMethod – this parameter tells NSG how the layout of the NoC should be generated. It can take two values, floating or fixed. Floating tells the backend to generate a layout using best

effort. However, for run-time reconfigurable implementations, a fixed layout is needed. Currently the parameter is ignored and it generates a floating implementation.

- switchType – this parameter tells NSG which switchtype to use when generating the NoC. If the switchType is not “Nostrum” or “Nostrum_TDN”, a user NoC is implied and NSG will start look for a user implementation.

- rniType – this parameter tells NSG which which RNI to use when generating the NoC. The only currently supported value is “Heartbeat”. If the rniType is set to something else, a user NoC is implied and NSG will start to look for a user implementation.

- NrOfRecvChannels – this parameters tells NSG what is the maximum amount of receive channels to use for all nodes. If the parameter is set to a numerical value, this value will be used in all nodes. If the parameter is set to “Minimize”, NSG will optimize the number of receive channels and set it to the number of input connections used for the HW node in question.

- NrOfSendChannels – this parameter tells NSG what is the maximum amount of send channels to use for all nodes. If the parameter is set to a numerical value, this value will be used for all nodes. If the parameter is set to “Minimize”, NSG will optimize the number of send channels and set it to the number of output connections used for the HW node in question.

- FrameSize – this parameter tells NSG what is the longest message that the system will ever have to manage. It needs to be large enough to contain both the setup flits and the message/data flits. For 32 bit data, 2 setup flits are needed. For 64 bit data and up, 1 setup flit is needed. The maximum value is 2048 bytes, or 512 (32-bit) words.

- FlitWidth – this parameter tells NSG how wide the data part of the flits should be. It will affect the bit width to the recv and send buffers, and the data width in the switch network accordingly.

- FlitInjectionRate – this parameter is only used by “Nostrum” networks. It tells Nostrum NoCs how long the RNI must wait before presenting a new a value to the NoC.

- TDN – this parameter is required by “Nostrum_TDN” networks. It tells Nostrum_TDNs in which switch cycle a message can be injected for all nodes. There are four slots which can be selected, and the number given should be in the format 2^slot, i.e.,

    - a value of 1 injects in slot 0
      a value of 2 injects in slot 1
      a value of 4 injects in slot 2
      a value of 8 injects in slot 3

  This value can be combined. Thus, a value of 5 injects data in slots 2 and 0, and a value of 15 injects in all slots, and a value of 0 turns sending off.

- Heartbeat – this parameter tells NSG with which frequency the signal “Heartbeat” should be set.

- Reset – this parameter tells the NoC how long it should wait before issuing the first Hearbeat.

- Programmable_Heartbeat. If this parameter is given and set to true, NSG will generate a NoC with a register for letting the processor Node 0 be able to reprogram the Heartbeat during run-time. The default for this parameter is “false”.

6.1.1.4 Node configuration parameters

The node connection statements tells NSG how the nodes should be configured. Each node has a similar set of parameters.

- Nr – this parameters contains the node number of the node to be configured.
- Mem_size – this parameter contains a list of the memory sizes for all memories used in the node. If multiple memory sizes are used, the memories are connected to the local CPU having the corresponding place in the list.
- Jtag – this parameter tells NSG if a JTAG unit should be used in the node or not.
- Perf_counter – this parameter tells NSG if a performance counter should be used in the node or not.
- Pio – this parameter contains a list of IO units that should be used in that node.
- Noc_irq – this parameter tells NSG if the rni's Irq-signal should be connected to the local processor or not.
- CPU – this parameter tells NSG how many CPUs that the node should have. It contains the list of CPU configurations for the node. The number of CPUs used and the number of memories that needs to be listed is correlated. Each node needs to have its own separate memory for containing that CPU's program code.
- Ip_block – this parameter contains a list with the names of all ip_blocks used in the node.

6.1.2 Software Section

This section tells NSG where, i.e., in which node, the software components should be stored and compiled. It also tells which MOC to use when generating the BOS system for the node, and how to configure the communication channels.

- Repository – this parameter tells NSG in which directory to search for the software that is listed in the process nodes.
- Name – this parameter tells NSG the name of the software component/process that should be executed.
- Node – this parameter tells NSG in which node the software component/process should be stored.
- CPU – this parameter tells NSG for which CPU in the node the software component/process should be compiled for.
- Moc – this parameter tells NSG which model of computation to use for the software component/process when the software for the BOS-system is generated.
- Sources – this parameter contains a list of the software component/process names that are providing input to the software component. The source must be matched with a corresponding target at the source node.
- Targets – this parameter contains a list of the software component/process names that the software component should communicate. The target must be matched with a corresponding source at the other target node.

6.2 Hardware Resources Used View

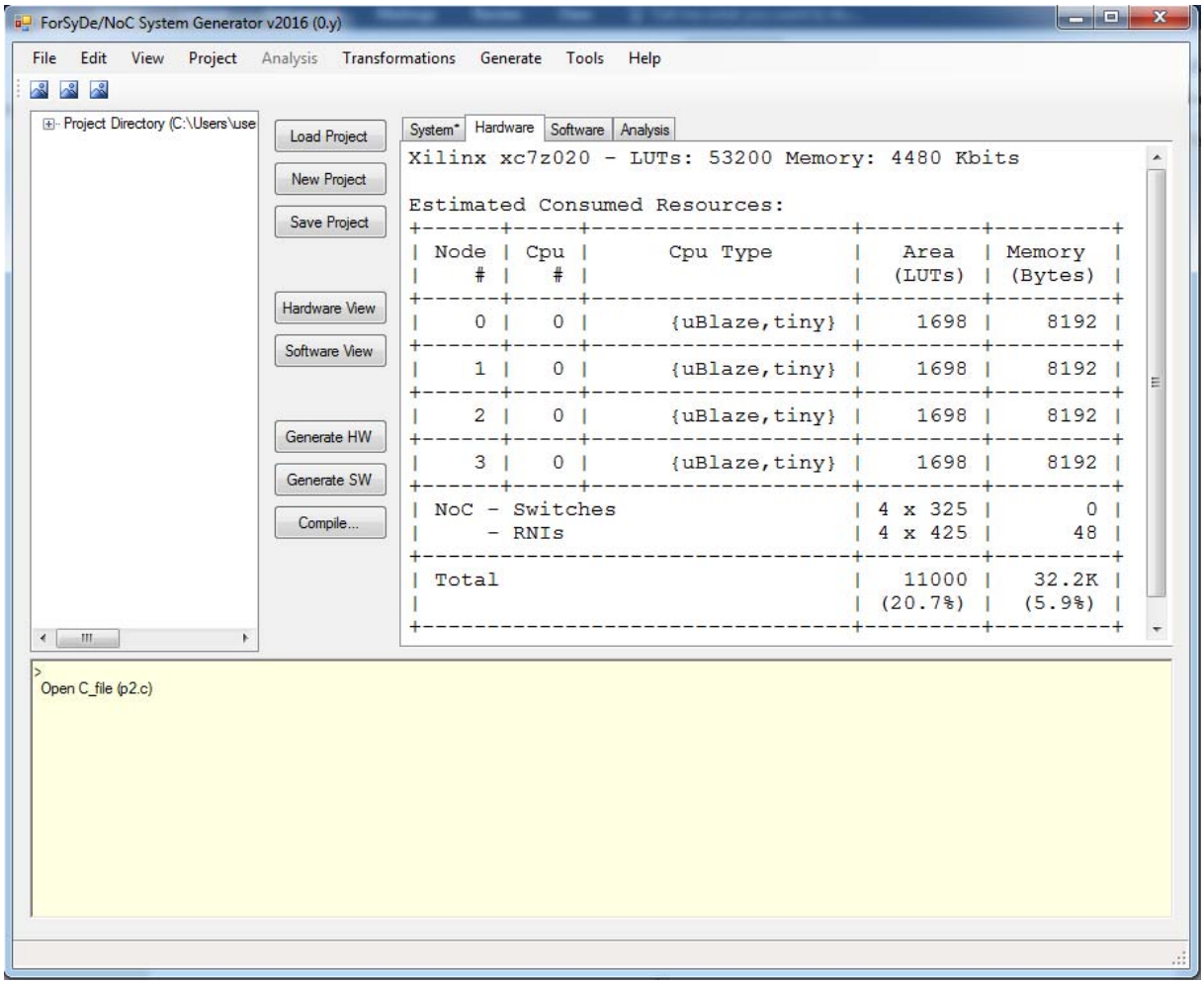


Figure 6.2.1. Hardware Resources Used view

This view present an estimate of the number of LUTs and memories needed for generating the system. The area estimate of the NoC is based on the measured area of a 2x2x1 NoC and has been hardcoded. The area of larger NoCs will be different from the number presented.

6.3 Software Resource Used View

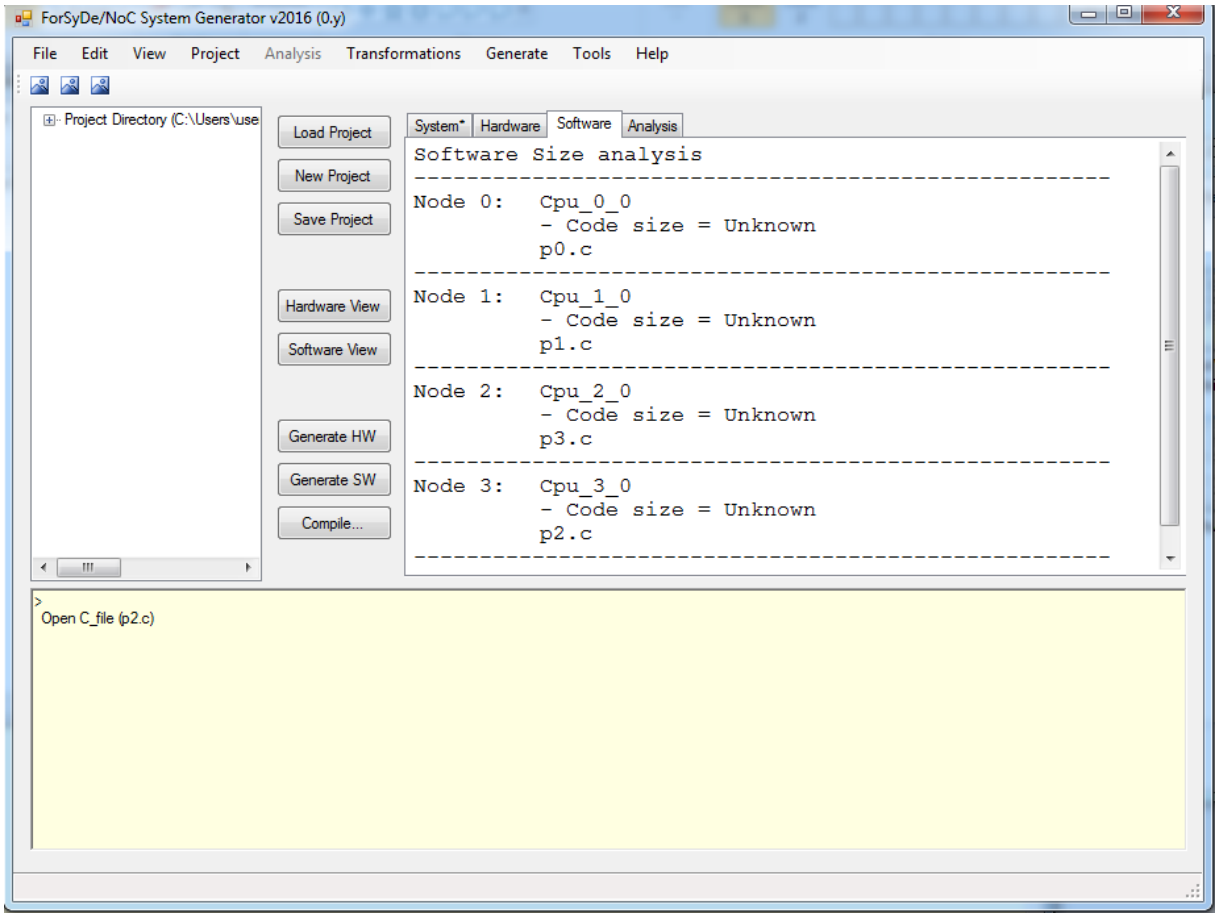


Figure 6.3.1. Software Resource Used view

This view contains a measurement of the amount of memory needed to compile the code. If can be obtained after the code has been compiled.

6.4 Timing Analysis View

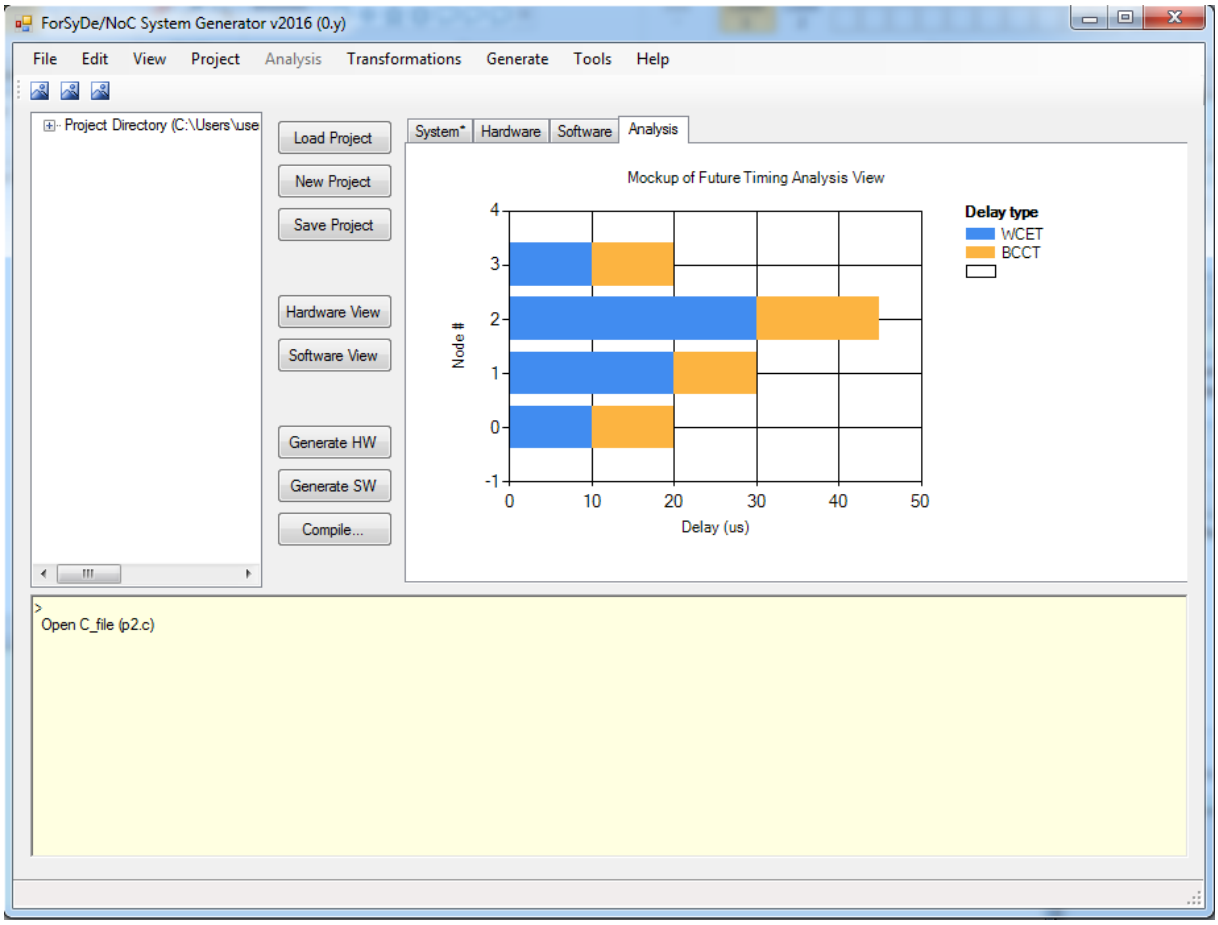


Figure 6.4.1. Timing Analysis view.

This tab contains a mock-up of the output of an envisioned future WCET (Worst Case Execution Time) estimation tool. It is currently not used.

7 Main View Drop down Menus

7.1 File

Restrictions:

The only working buttons are New, Open, Save and Save As. You can also Load, Open and Save a project using the Quick buttons on the Main Window.

7.1.1 New

7.1.2 Open

7.1.3 Save…

7.1.4 Save As…

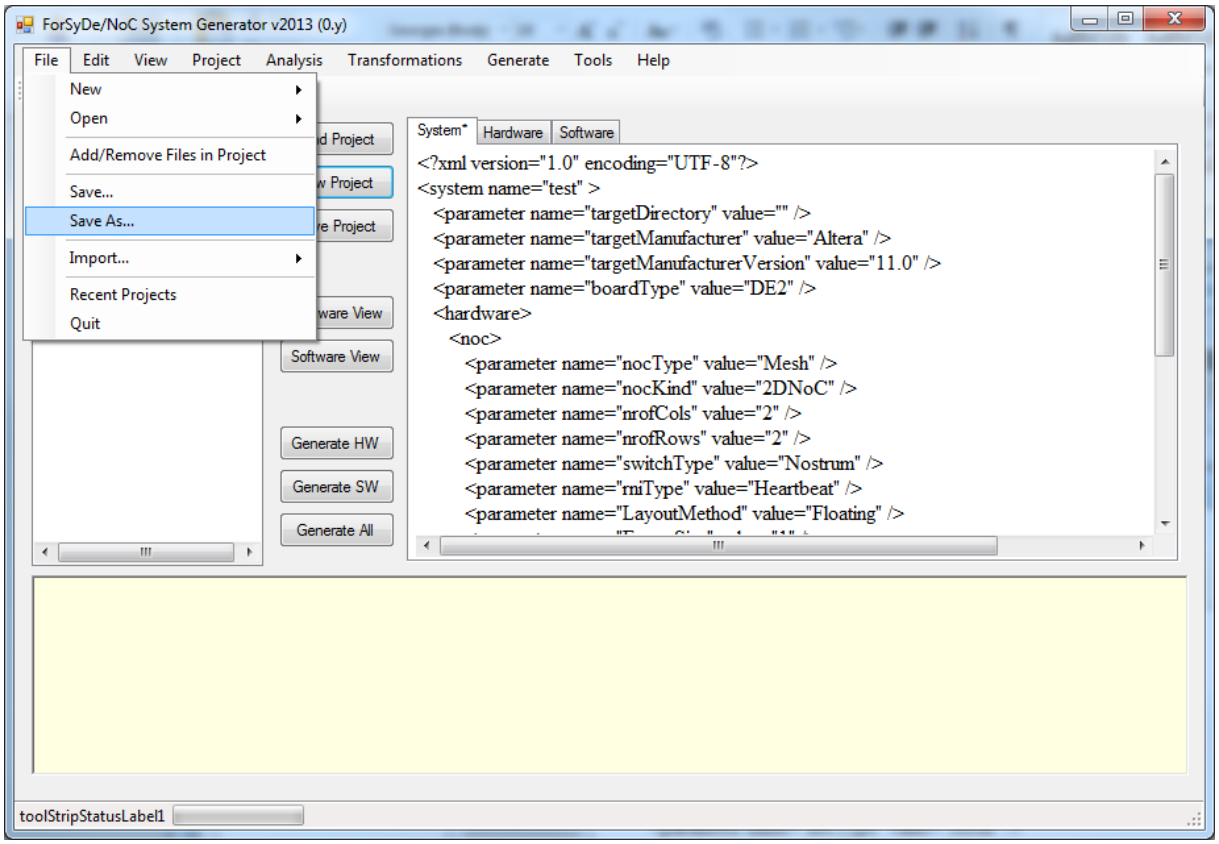


Figure 7.1. File Menu list

7.1.5 Import…

7.1.6 Recent Projects

7.1.7 Quit

7.2 Edit Menu

The Edit menu shows four choices, none of which has been implemented yet:

1. **Interconnect** – should let you create and connect an arbitrary NoC.
2. **Hardware** – should link you to the Hardware View**...**
3. **Software -** should link you to the Software View...
4. **Board –** should let you configure the pinning and components of your own FPGA board.

7.3 View Menu

The View menu shows two choices, none of which has been implemented yet:

1. **Hardware** – should link you to the Hardware View**...**
2. **Software -** should link you to the Software View...

7.4 Project Menu

The Project menu shows five choices, three of those have been implemented, two has not:

1. **Check XML** – If you manually have edited the system description in the System Description Window, the Check XML allows you to check if the code you have written is compliant with the System Description XML. **(This function have not been fully tested)**
2. **Timing Constraints...** - Not implemented yet. Is supposed to let you specify BCET, WCET, BCCT and WCCT in your descriptions.
3. **Pinning**... - Not implemented yet. Is supposed to let you check/specify the pinning of the description,
4. **NoC Settings...** - Opens the NoC Settings Window.
5. **Project Settings...** - Opens the Project Settings Window.

7.5 Analysis Menu

This window is currently disabled. It is supposed to let you do BCET, WCET, BCCT, and WCCT analysis.

7.6 Transformations Menu

This window has three choices. One has been implemented.

1. **Calc2HW...** – Opens the window to NSG's own HLS-tool Calc2HW.

2. **Clone process**... - This function is currently disabled. Use the Clone function in the Software View Window instead.

3. **Pipeline process...** - This function has not been implemented yet. It is supposed to break up the code for a COMB/SMOC into smaller SMOC pieces. This function will likely to move to the Software View in the future.

7.7 Generate Menu

This window has five choices:

1. **Hardware** – This is the same function as the Generate HW quick menu button.

2. **Software** – This is the same function as the Generate SW quick menu button.

3. **HW and SW** – This invokes the NoC System Generator and generates both Hardware and Software in the same call.

4. **HW and SW (verbose)** – This invokes the NoC System Generator and generates both Hardware and Software in the same call, but with the verbose switch turned on. Useful to get more information during debugging, especially if the NSG crashes.

5. **ForSyDe System-C...** - This choice opens the ForSyDe System-C conversion window. It converts the NSG system description into a simulated System C-description using the ForSyDe System-C libraries. **NB! This function crashes currently...**

7.8 Tools Menu

This window has six choices, of which only one is currently working:

1. Altera Quartus – This option should open the Altera Quartus tool. Currently it has to be opened manually, outside NSG.

2. Xilinx Vivado – This option should open the Xilinx/Vivado tool. Currently it has to be opened manually, outside NSG.

3. GaislerTech – This option should open the GaislerTech Leon III configuration tool. Currently it has to be opened manually, outside NSG.

4. ProGram Compiler – This option should open the KTH Program Compiler. This option has not been implemented yet.

5. ForSyDe System-C Simulator – This option should open the KTH ForSyDe System-C simulator. This option has not been implemented yet.

6. Default Settings... - Opens the Default Settings Window.

7.8.1 Default Settings Window

In this window, the default values for various selections that appear can be changed, for instance the default size of memory. When a new NoC is created, it uses the default settings to populated all created nodes. The defaults are saved and loaded when NSG starts.

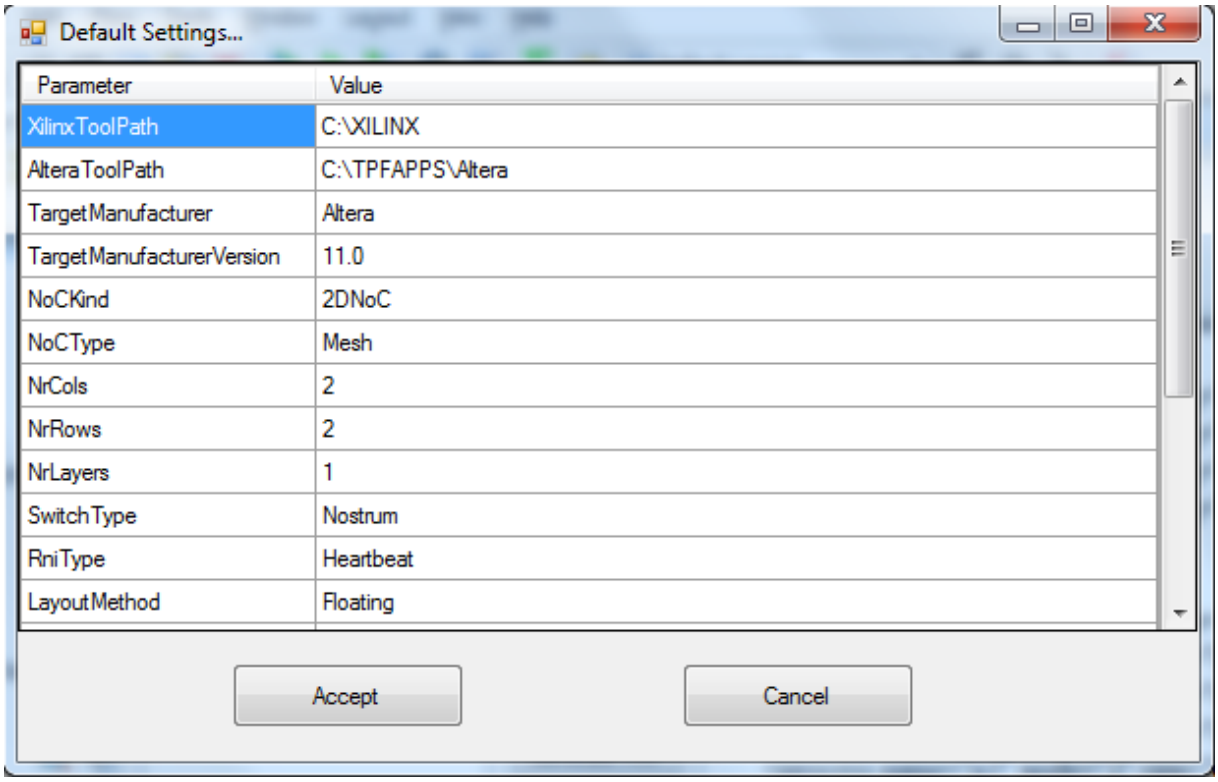


Figure 7.2. Default Settings

The settings that can be changed are:

- XilinxToolPath – The search path used by NSG where to search for Xilinx/Vivado versions
- AlteraToolPath – The search path used by NSG to search for Altera/Quartus versions
- TargetManufacturer – Default manufacturer when tool starts
- TargetManufacturerVersion – Default manufacturer version when tool starts
- NoCKind – Default NoC Dimension selected when tool starts
- NoCType – Default NoC Topology
- NrCols – Default number of columns
- NrRows – Default number of rows
- NrLayers – Default number of layers
- SwitchType – Default SwitchType
- RniType – Which RNI to use as default
- LayoutMethod – whether the layout on the FPGA should be floating or fixed. The latter is required for Run-Time-Reconfigurability. Accepted values *floating/fixed*.
- FrameSize – The default FrameSize in number of flit word to send. The number should include the header flits needed to set up a message frame.

- Heartbeat – The default frequency for the heartbeat signal.
- ResetTime – The default time before the first heartbeat.
- JTAG Debugger – Sets whether a JTAG Debugger should be generated or not by default when a new NoC is generated. Accepted values *true/false*.
- Performance Counter – Sets whether a performance counter should be generated or not by default when a new NoC is generated. Accepted values *true/false*.
  NB! This setting is obsolete since the NoC itself contains a counter that can be used for performance measurements.
- NoCIRQ – Set whether the NoCIRQ signal should be connected to the processor by default or not. Accepted values *true/false*.
- PIO – Sets the default type and size for the GPIO of a node. For instance *Output(1)*.
- Memory Type(Size) – Sets the default type and size of memory for a node. For instance *OnChip(8k)*.

7.9 Help Menu

This window has only one choice, which is not currently working:

1. **Version...** - This function is supposed to show information about the NSG version used. It has not been implemented. The version number is shown on the top of the Main window instead.

8 References

[1] Öberg, J., Robino, F., “A NoC System Generator for the Sea-of-Cores Era”, In Proc. of FPGAWorld 2011, Copenhagen, Stockholm, Munich, September, 2011, ACM Digital Libraries.

[2] Nowshad Painda Mand, Francesco Robino, Johnny Öberg, “Artificial neural network emulation on NOC based multi-core FPGA platform”, In Proc. of NorChip-2012, Copenhagen, Denmark, Nov. 12-13, 2012.

[3] F. Robino, J. Öberg, “The HeartBeat model: A platform abstraction enabling fast prototyping of real-time applications on NoC-based MPSoC on FPGA”, In Proc. of ReCoSoC-2013, Darmstadt, Germany, July 10-12, 2013.

[4] F. Robino, J. Öberg, “From Simulink to NoC-based MPSoC on FPGA”, In proc. of DATE-2014, Dresden, Germany.

[5] Hussein Ezzeddine, Johnny Öberg, Francesco Robino, “Validation of pipelined Double-precision Floating Point operations in a Multi-core environment Implemented on FPGA using the ForSyDe/NoC System Generator Tool Suite”, In Proc. of NorChip-2014, Tampere, Finland, Oct 2014.

[6] P. I. Diallo, S. H. Atterzadeh Niaki, F. Robino, I. Sander, J. Champeau, J. Öberg, “A Formal, Model-driven Design Flow for System Simulation and Multi-core implementation”, In Proc. of 10th IEEE International Symposium on Industrial Embedded Systems (SIES-2015), Siegen, 2015.

[7] A. Lenz, M. Azkarate-Askasua Blázquez, J. Coronel, A. Crespo, S. Davidmann, J.-C. Diaz Garcia, N. González Romero, K. Grüttner, R. Obermaisser, Johnny Öberg, J. Perez, I. Sander and I. Söderquist “SAFEPOWER project: Architecture for Safe and Power-Efficient Mixed-Criticality Systems”, In proc. of the Euromicro Conference on Digital Systems Design (DSD-2016), Limasol, Cyprus, 31 Aug.-2 Sept, 2016.

[8] R. Seyyedi, M. T. Mohammadat, M. Fakih, K. Grüttner, J. Öberg and D. Graham, “Towards Virtual Prototyping of Synchronous Real-time Systems on NoC-based MPSoCs”, In Proc. of 12th IEEE International Symposium on Industrial Embedded Systems (SIES-2017), Toulouse, France, 14-16 June, 2017.

[9] J. Öberg, “Synthesis of VLIW Accelerators from Formal Descriptions in a Real-Time Multi-Core Environment”, In Proc. of FPGAWorld 2017, Stockholm, Sweden, Sept. 2017, ACM Digital Libraries.

[10] K. Roswall, T. Mohammadat, G. Ungureanu, J. Öberg, I. Sander, “Exploring Power and Throughput for Dataflow Applications on Predictable NoC Multiprocessors”, In Proc of 21st Euromicro Conference on Digital System Design (DSD-2018), Prague, Czech Republic, 2018, pp. 719-726.

doi: 10.1109/DSD.2018.00011.
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8491891&isnumber=8491778

[11] Vivado Design Suite User Guide, Xilinx inc.

[12] Vivado Design Suite User Guide: Designing with IP (UG896), Xilinx inc.

[13] Quartus II Handbook Version 14.1.0, Intel inc.

[14] Introduction to Intel® FPGA IP Cores, Intel inc.

[15] Embedded Peripherals IP User Guide – Intel, Intel inc.

9 Appendices

9.1 Board Files

For NSG to work, the tool requires that there is a description of the board you are going to use. The board descriptions are stored according to the directory structure

<ForSyDe_path>/Boards/<target FPGA vendor>.

NSG currently supports two target FPGA Vendors, Altera and Xilinx. In each directory, there are examples of board files for the technology, together with a file containing the description of the supported devices. If you want to add your own board, you just have to write a board file, and add the FPGA device to the Device list, provided that you have to the license to access the target device in the target FPGA vendor's tool set.

9.1.1 Device file Syntax

The Devices.xml file, contains a list of devices, on the form

<device name="" LUTS="" MEMORY="" DSPS="" MAXIO="" />

An example is given below

<device name="EP2C35F672C6" LUTS="33216" MEMORY="473" DSPS="35" MAXIO="475" />

- The device name is the name of the FPGA
- LUTS is the number of available LUTs in the FPGA
- MEMORY is the number of available memory in the FPGA. Unit is in Kibits (1024 bits).
- DSPS is the number of available DSP units that can be used in the circuit.
- MAXIO is the maximum number of I/Os that can be used.

For devices that contain hard cpus, there is an additional parameter

- HARDCPU="type,number,size of 1st level Cache, size of 2nd level Cache". Unit is in KiBytes.

Example:

<device name="5CSEMA5F31C6N" LUTS="85000" MEMORY="4450" DSPS="174" MAXIIO="288" HARDCPU="Arm Cortex A9,2,32,512" />

9.1.2 Board file Syntax

A board file is name <Board name>.xml. It contains the declarations necessary for NSG to connect and associate IOs to external pins on the board. It also contains the necessary parameters that should be set to enable the target Vendor tool to synthesize a design.

The first parameter is the board name, e.g.

<board name="DE2-115">

Then follows a set of parameters with the overall properties of the board that needs to be set for correct synthesis. For Altera

- <parameter name="FAMILY" value="Cyclone IV E" />
- <parameter name="DEVICE" value="EP4CE115F29C7" />
- <parameter name="MIN_CORE_JUNCTION_TEMP" value="0" />
- <parameter name="MAX_CORE_JUNCTION_TEMP" value="85" />
- <parameter name="NOMINAL_CORE_SUPPLY_VOLTAGE" value="1.2V" />
- <parameter name="STRATIX_DEVICE_IO_STANDARD" value="2.5 V" />
- <parameter name="BOARDFREQUENCY" value="50MHz" />

For Xilinx:

- <parameter name="FAMILY" value="zynq" />
- <parameter name="DEVICE" value="xc7z020" />
- <parameter name="PACKAGE" value="clg484" />
- <parameter name="SPEEDGRADE" value="-1" />
- <parameter name="BOARDFREQUENCY" value="100 MHz" />

Next follows all connections. The '#'-sign is used for comments, according to the NSG extension to XML syntax.

The command <io_group is used to group together a set of inputs of the same type that have common properties. It should contain a group name, and list of common properties, in the form of a semi-colon separated list. A property is enclosed in curly brackets. Each property is in turn a list by themselves, in the form of a comma-separated list enclosed between the curly brackets, e.g.

Property="{<property one>,<property two>};{<property one>}"

For each pin, the pin name that should be associated with the pin placement is given, together with the direction of the pin.

Examples:

```
# Clock pins
<io_group name="sys_clk" property="{IOSTANDARD,LVDS_25}" type="clock">
        <pin name="sys_clk" value="Y2" direction="i"/>
</io_group>
        # I/O pins
            <io_group name="ledg" property="{IO_STANDARD,2.5 V}" type="pio">
            <pin name="ledg[0]" value="E21" direction="o"/>
            <pin name="ledg[7]" value="F17" direction="o"/>
</io_group>
```

NB! If you want to be able to change technology without changing the pin assignments, you should name the pins the same in the different board files. The pin name is case sensitive.

If you want to use an external ram, NSG requires you to group in one level more. The nr parameter tells which interface that is addressed. Three io_groups are required, "address", "CTRL", and "data":

```
<external_ram type="sram" nr="0">
        <io_group name="address" property="{}" type="address">
            <pin name="address[0]" value="T0" direction="o"/>
            ...
            <pin name="address[19]" value="T19" direction="o"/>
        </io_group>
        <io_group name="CTRL" property="{}" type="ctrl">
            <pin name="bw_n[0]" value="Q0" direction="o"/>
              ...
            <pin name="outputenable_n" value="R0" direction="o"/>
        </io_group>
        <io_group name="data" property="{}" type="data">
            <pin name="data[0]" value="D0" direction="io"/>
            <pin name="data[31]" value="D31" direction="io"/>
        </io_group>
</external_ram>
```

If your board contains a DRAM, it requires a different set of io_groups:

```
# DDR3 RAM
<external_ram type="DRAM">
        <io_group name="DDR_CLK" property="{IOSTANDARD,LVCMOS15}" type="CLOCK">
                <pin name="DDR_Clk_p" value="N4" direction="io"/>
                <pin name="DDR_Clk_n" value="N5" direction="io"/>
        </io_group>
        <io_group name="DDR_RESET" property="{IOSTANDARD,LVCMOS15};{TIG}" type="RESET">
                <pin name="DDR_DRSTB" value="F3" direction="io"/>
        </io_group>
        <io_group name="DDR_CTRL" property="{IOSTANDARD,LVCMOS15}" type="CTRL">
                <pin name="DDR_CKE" value="V3" direction="io"/>
                <pin name="DDR_CS_n" value="P6" direction="io"/>
                <pin name="DDR_RAS_n" value="R5" direction="io"/>
                <pin name="DDR_CAS_n" value="P3" direction="io"/>
                <pin name="DDR_WEB" value="R4" direction="o"/>
                ...
                <pin name="DDR_VRP" value="M7" direction="io"/>
        </io_group>
         # Bank Address pins
        <io_group name="DDR_BankAddr" property="{IOSTANDARD,LVCMOS15}"
        type="BANK_ADDRESS">
                <pin name="DDR_BankAddr[0]" value="L7" direction="io"/>
                <pin name="DDR_BankAddr[1]" value="L6" direction="io"/>
                 <pin name="DDR_BankAddr[2]" value="M6" direction="io"/>
        </io_group>
        <io_group name="DDR_Addr" property="{IOSTANDARD,LVCMOS15}" type="ADDRESS">
                <pin name="DDR_Addr[0]" value="M4" direction="io"/> # PS_DDR_A[0]
                <pin name="DDR_Addr[1]" value="M5" direction="io" /> # PS_DDR_A[1]
                ...
                <pin name="DDR_Addr[14]" value="G4" direction="io" /> # PS_DDR_A[14]
        </io_group>
        # data pins
        <io_group name="DDR_DQ" property="{IOSTANDARD,LVCMOS15}" type="DATA">
                <pin name="DDR_DQ[0]" value="D1" direction="io"/> # PS_DDR_DQ[0]
                <pin name="DDR_DQ[1]" value="C3" direction="io"/> # PS_DDR_DQ[1]
                ...
                <pin name="DDR_DQ[31]" value="Y1" direction="io"/> # PS_DDR_DQ[31]
        </io_group>
</external_ram>
```

The board file ends with

```
</board>
```

9.2 Using Altera Quartus as backend

9.2.1 Creating a design in Quartus

- Open Quartus
- Create a project in the target directory
- Open Qsys (or SoPC builder for older versions of the target tool). Select the file that NSG has created for system generation.
- Generate VHDL for the System (have a coffee break or do something else while Quartus generates the VHDL).

9.2.2 Working with the SDK

- Start Nios II building tools for Eclipse

The following window appears:

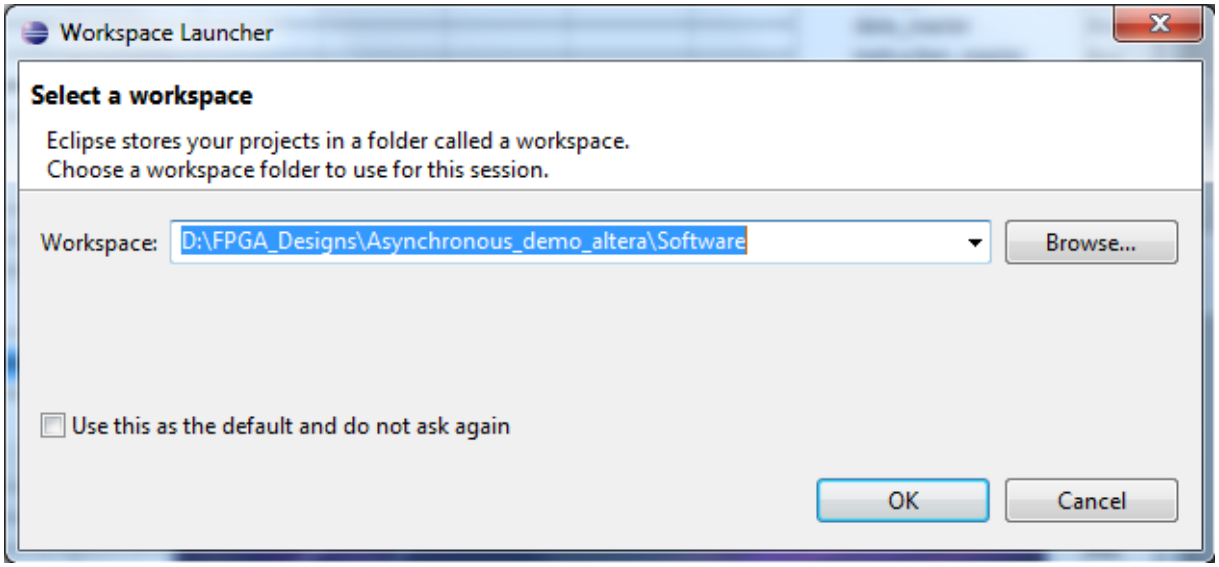


Figure 9.2.1. Eclipse Launch Window

- Set the workspace to the *Software*-directory under the NSG target directory.
- In Eclipse, start a new application project for every node that you have in the design (select File->New->Nios II Application and BSP from template). Fill in the name of the Project as Node_<node nr>_<cpu nr>. The associated BSP will be named after the project as Node_<node nr>_<cpu nr>_bsp. Select the CPU associated with the node CPU cpu_<node nr>_<cpu nr>. Select an empty template and press finish. The BSP name can be selected if you press <next>, but that can be left at default value. See figure 9.2.2 below. If Eclipse complains that the directory already exists on the disk, you have run the NSG software generation too early. Delete the directories and continue (press next or finish).

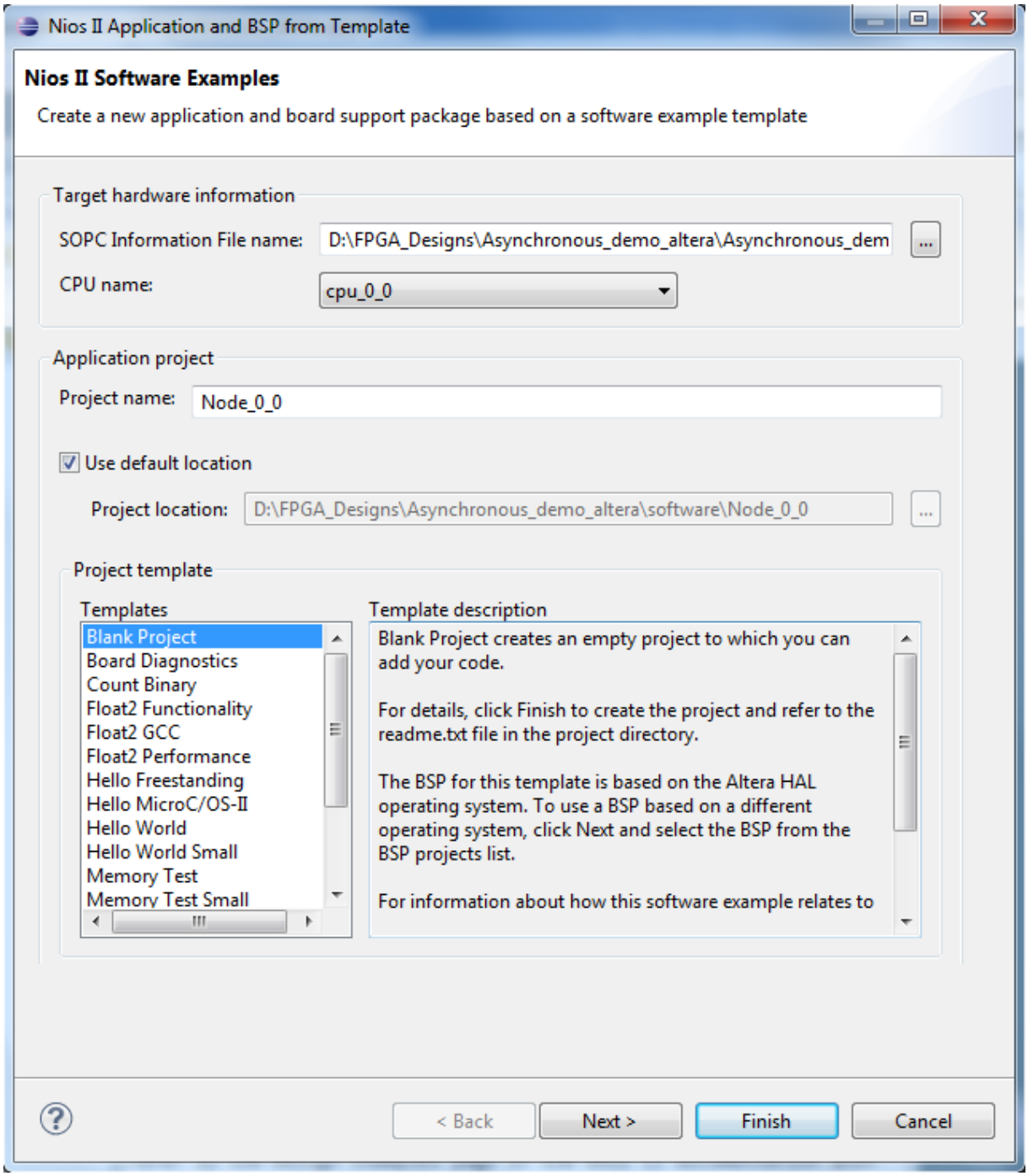


Figure 9.2.2. Eclipse create project menu.

Once you are finished, the Eclipse window should look like this

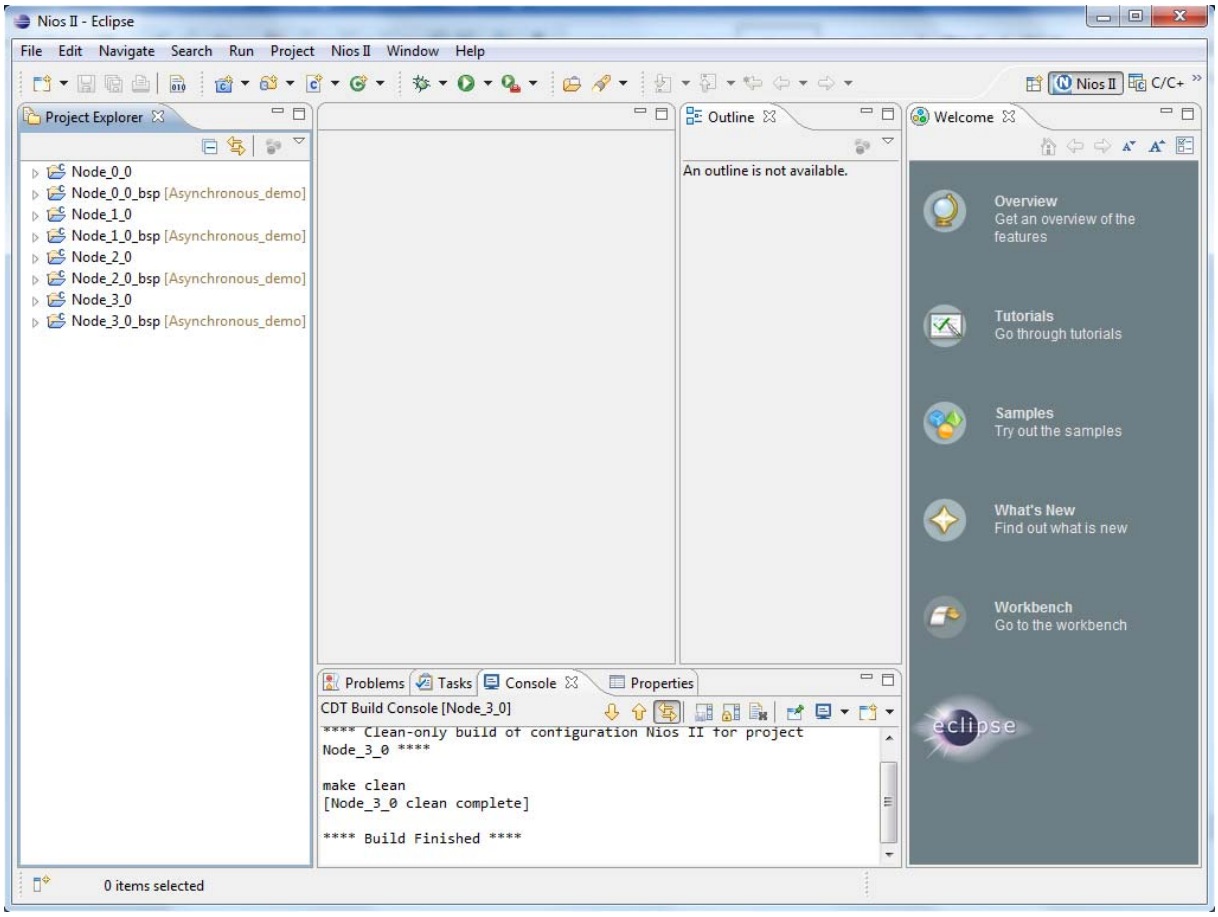


Figure 9.2.3. Eclipse View

Now you can work with the software like in a normal project.

9.2.3 Converting .elf-files to .hex-files

Once the SW has been built, you have to include it into the project. The genereted memories assume that the memory-files are .hex-files named according to “onchip_memory_<node_nr>_<cpu_nr>”. To convert the generated .elf-files to the correct format:

- Build all projects. This creates the required elf-files of all software projects.
- Right Click in the Project Explorer Window. Select Nios II->Nios II Command Shell...
- Go to the Software directory. Run the file create-hex-files (i.e., source create-hex-files). The scriot translate the elf-files to Quartus .hex-files, and copies them to the project folder.

9.2.4 VHDL Simulation

Once you are happy with the software, you can do a behavioural VHDL simulation of the entire design. NB! You cannot simulate designs that contains an ARM-core, only nios cores.

- In QSYS, when you generate HW, also remember to generate the VHDL for simulation and for generating a simple testbench.
- Convert the .elf-files to .hex-files, as explained in section 9.2.3.
- Copy and paste the generated hex-files to the simulation directory.

- Start Modelsim.
- Create a new project in the simulation directory and include all generated VHDL-files into the project.
- List the signals you want to look at in the simulation window.
- Run the simulation for the length of time that is required. A typical time to select here is a multiple of the Heartbeat time you have selected for the Synchronous MoCs, e.g., 200 us.

9.3 Using Xilinx Vivado as backend

9.3.1 Creating a design in Vivado

- Open Vivado
- Run the TCL-script CreateSystem.tcl that NSG created in the target directory
- Wait until Vivado has finished building the design (have a coffee break while Vivado is building the system)

9.3.2 Working with the SDK

- In Vivado, select File->Launch SDK

The following window appears:

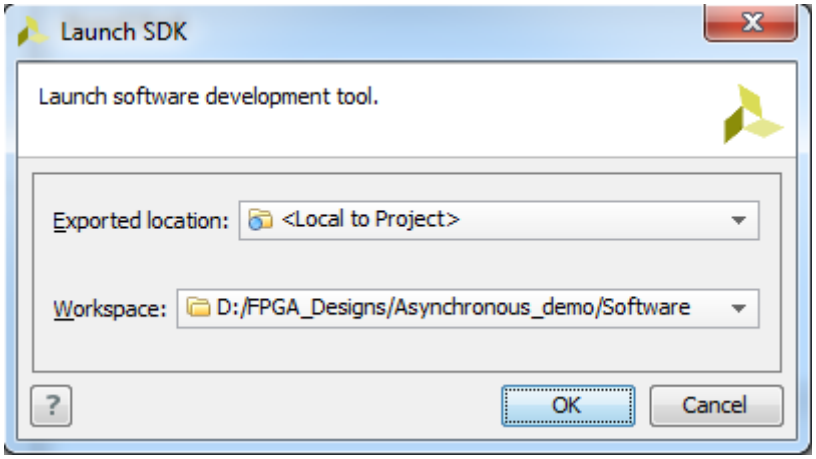


Figure 9.3.1. Launching SDK inside Vivado

- Set the workspace to the *Software*-directory under the NSG target directory. Exported location should be <Local to Project>.
- In the SDK, start a new application project for every node that you have in the design (select File->New->Application Project). Fill in the name of the Project as Node_<node nr>_<cpu nr>. The associated BSP will be named after the project as Node_<node nr>_<cpu nr>_bsp. Select the CPU associated with the node CPU mb_<node nr>_<cpu nr>. The rest can be left at default value. See figure 9.3.2 below. If the SDK complains that the directory already exists on the disk, you have run the NSG software generation too early. Delete the directories and continue (press next or finish).

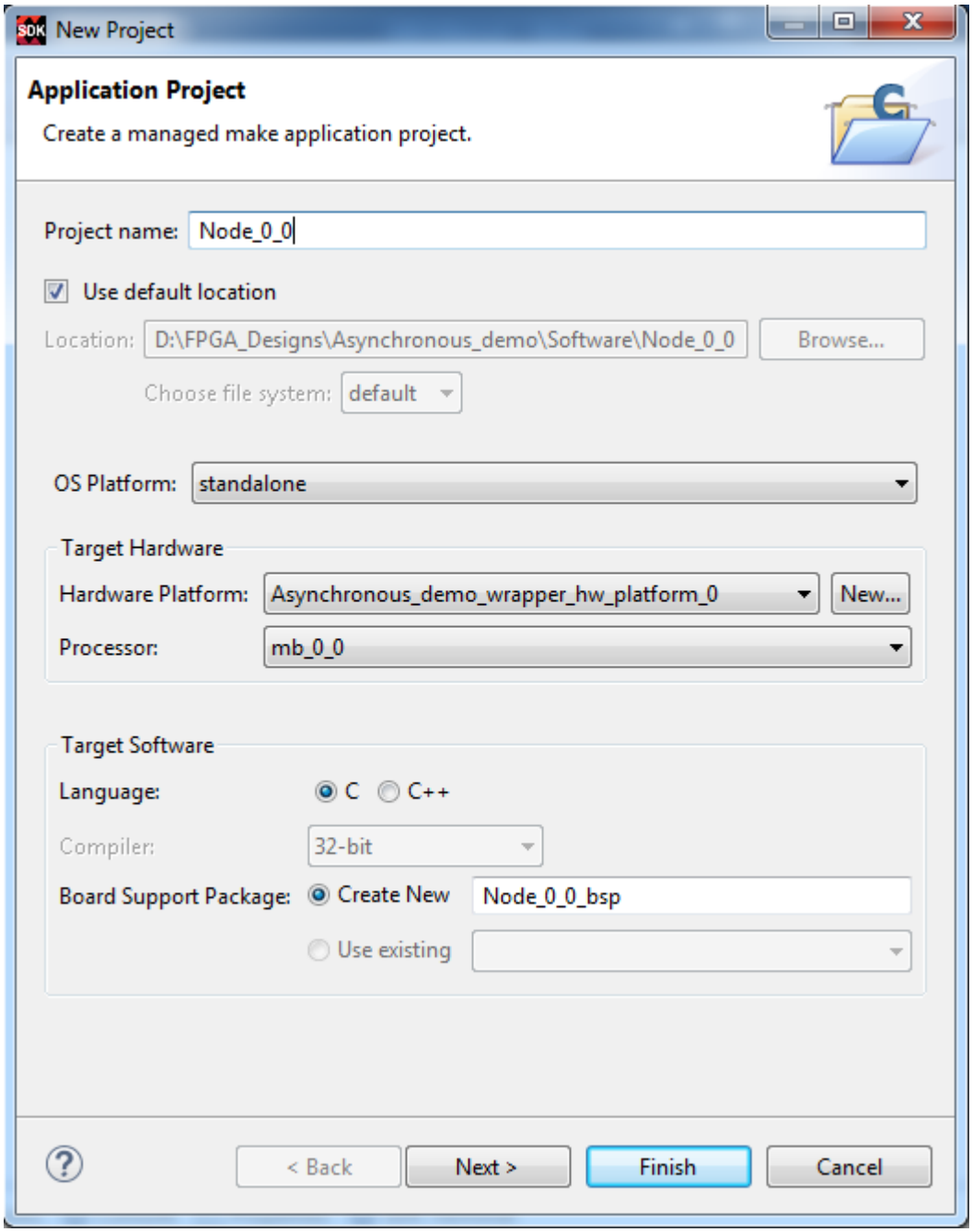


Figure 9.3.2. Creating a new Software Application project, and the corresponding bsp-file.

- If you press <next>, the software Template window appears, seen in figure 9.3.3. Select an Empy Application and press Finish.

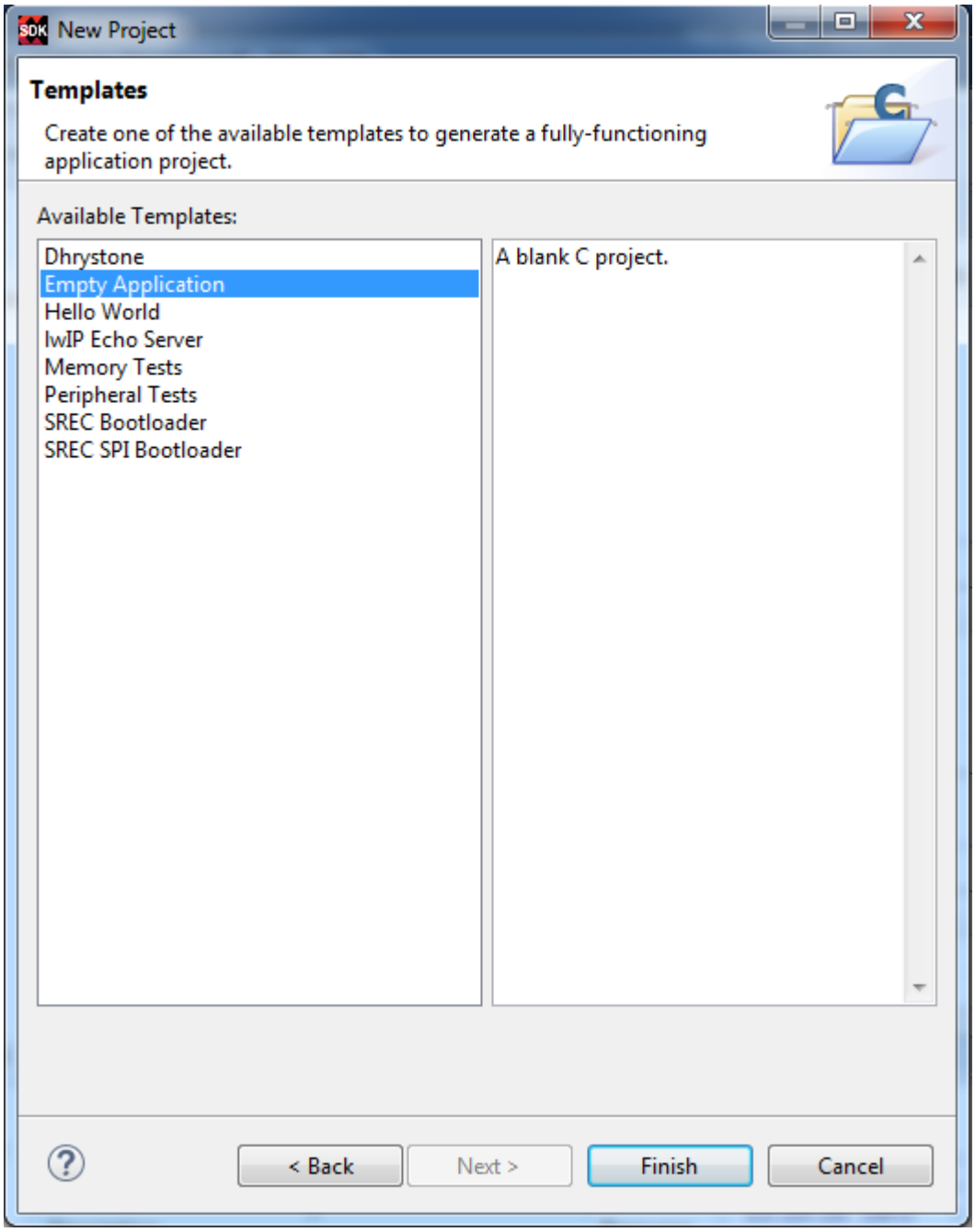


Figure 9.3.3. Software Template Selection in Vivado.

- Once you are finished, the SDK window should look like this

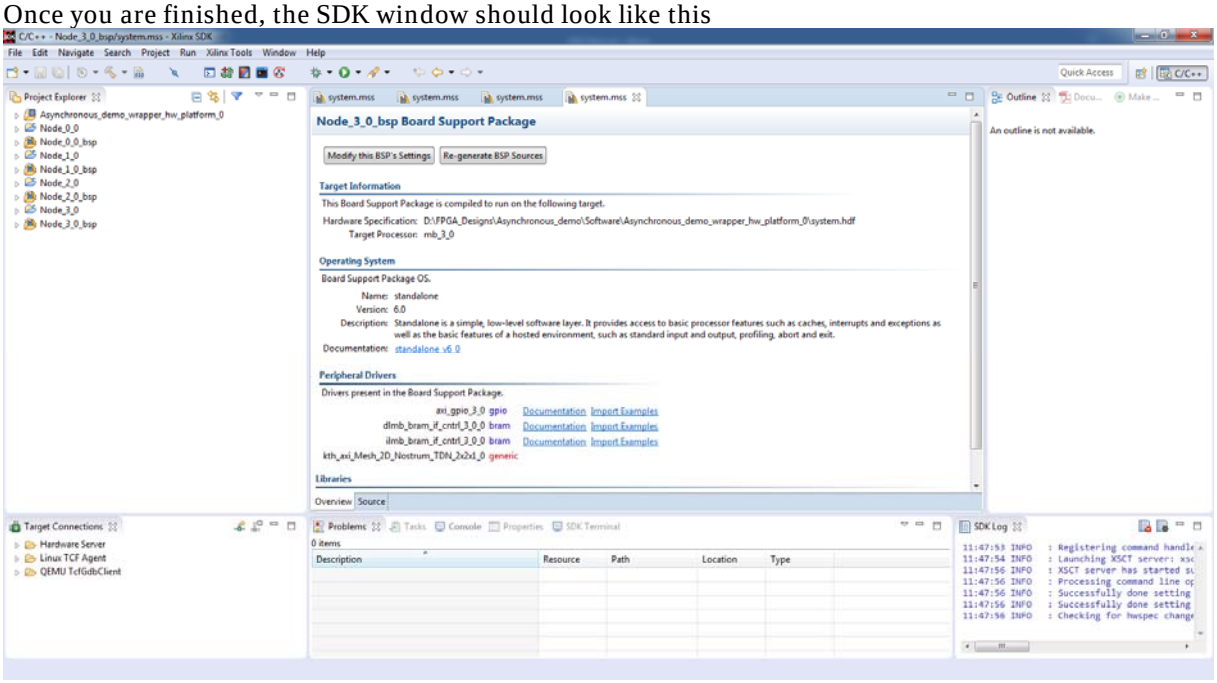


Figure 9.3.4. SDK Layout after project builds.

- In NSG – run Generate SW

- In SDK – refresh the directories Node_<node nr>_<cpu nr>/src by selecting the directory and pressing F5. The SDK will now automatically build the node software and tell you if there are any errors in it.

- If you change the SW in the SDK, please remember that it is not the original you are editing. The original is the one that is copied to this directory when you run Generate SW in NSG. Permanent changes should be done in the original C-file in the software repository.

- If you want to recompile your software, in SDK, press Project->Build All

9.3.3 VHDL Simulation

Once you are happy with the software, you can do a behavioural VHDL simulation of the entire design. NB! You cannot simulate designs that contains an ARM-core, only microBlaxe cores.

- In Vivado, run the tcl-script AssociateELFfiles.tcl.

- Under Quick-menu Simulation, press Run Simulation->Behavioural Simulation. The simulation window appears.

- On the Tcl console command line in the simulation window, type restart.

- Run the tcl-script SetupSimulation.tcl

- List the signals you want to look at in the simulation window.

- On the tcl console command line, type run <time> <time unit>. The typical time to write here is the Heartbeat time you have selected for the Synchronous MoCs, e.g., run 200 us.

9.4 Using IP blocks

IP Blocks are stored in sub-directories, preferably in a subdirectory right under your software repository.

An IP-block is declared in the hardware header using the <*ip_block*...> command:

```
<ip_block type="" name="" version="" directory="">
        <parameter name "bus" value="" />
        <parameter name "use_noc_irq" value="" />
</ip_block>
```

- The block type should be the target tool type Xilinx or Altera
- Name should be the name of the IP
- Version should be the IP version, e.g. "1.0" or "2.0".
- Directory is the path to the subdirectory where the IP is stored
- The parameter bus is a triplet that contains "{<bus_protocol>,<bus width>,<bus_type>}",
    - bus_protocol is the protocol that the IP speaks, i.e., Axi for Xilinx IPs, Avalon for Altera IPs, or DAP (direct access protocol) for IPs talking directly to the streaming 64-bit wide NoC RNI port.
    - Width of the bus, Axi and Avalon are 32 bits wide, DAP is 64.
    - Bus_type selects between a master ("m") or a slave ("s") port. You can have several master and slave ports. You indicate the port number in conjunction with the "m" or "s", i.e., "m#" or "s#".
      If you only have one slave or one master port and want it unnumbered, you add a space character instead of a number before the bracket, e.g., "{axi,32,s }".
- The parameter “use_noc_irq” sets whether the IP should be connected to the NoC_IRQ signal or not. Accepted values are "Yes"/"No".

e.g.,

```
<ip_block type="Xilinx" name="axi_timer" version="2.0"
    directory="C:/Users/user/Desktop/NoC_SafePower/Examples/Avionics_demo/Xilinx">
    <parameter name="bus" value="{axi,32,s }" />
    <parameter name="use_noc_irq" value="no" />
</ip_block>
```

9.4.1 Xilinx IPs

The subdirectory referred to in the IP_block declaration should follow the Xilinx standard for user repos.

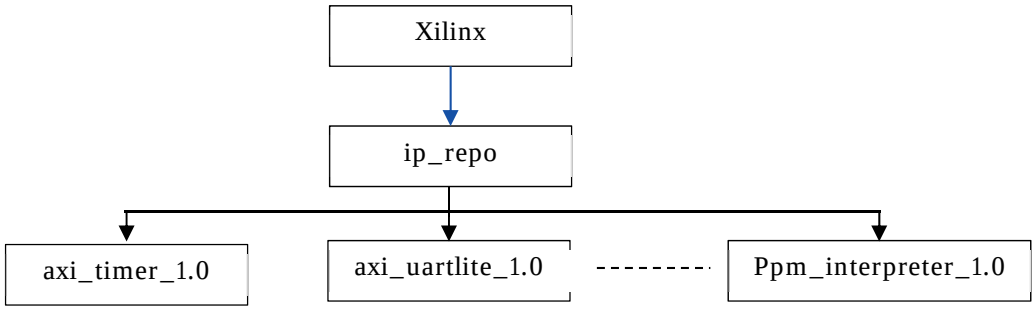


Figure 9.4.1. Xilinx IP subdirectory structure

9.4.1.1 Using Xilinx Vivado inbuilt IPs

Target tool IPs are IPs that come with the Vivado tool. They need only two files in their resp IP-directory to be used in NSG:

- CreateSubSystem.tcl
- ImportFiles.tcl

The file CreateSubSystem.tcl, contains the description for creating the IP using Vivado tcl-scripting, e.g.:

```
create_bd_cell -type ip -vlnv xilinx.com:ip:axi_timer:2.0 axi_timer_%d_%d
```

The IP is named axi_time_%d_%d, where the first "%d" indicate that it should be replaced with a node number and the second that it should be replaced with ip_number in that node.

The file ImportFiles.tcl should be left empty (or contain a comment) for target tool IPs, since no files need to be read to create it.

9.4.1.2 Creating User IP blocks for Xilinx Vivado

User IP blocks are created from VHDL-code and should be stored according to the IP_block standard of the target tool. In addition, the two files CreateSubSystem.tcl and ImportFiles.tcl should be stored at the root directory.

The CreateSubSystem.tcl contains the code needed to instantiate the ip, according to the description in the previous section, i.e.,

```
create_bd_cell -type ip -vlnv ppm:user:ppm_interpreter:1.8 ppm_interpreter_%d_%d
```

whereas the ImportFiles.tcl contains the code needed to use the vhdl code that is stored in IP directory, e.g.,

```
set_property  ip_repo_paths
   {C:\Users\user\Desktop\NoC_SafePower\Examples\Avionics_demo\Xilinx\ip_repo}
   [current_project]
update_ip_catalog
```

For more details on Creating own IPs, and the content of the files in the IP directories, we refer the interested reader to the Xilinx Vivado Design Manuals.

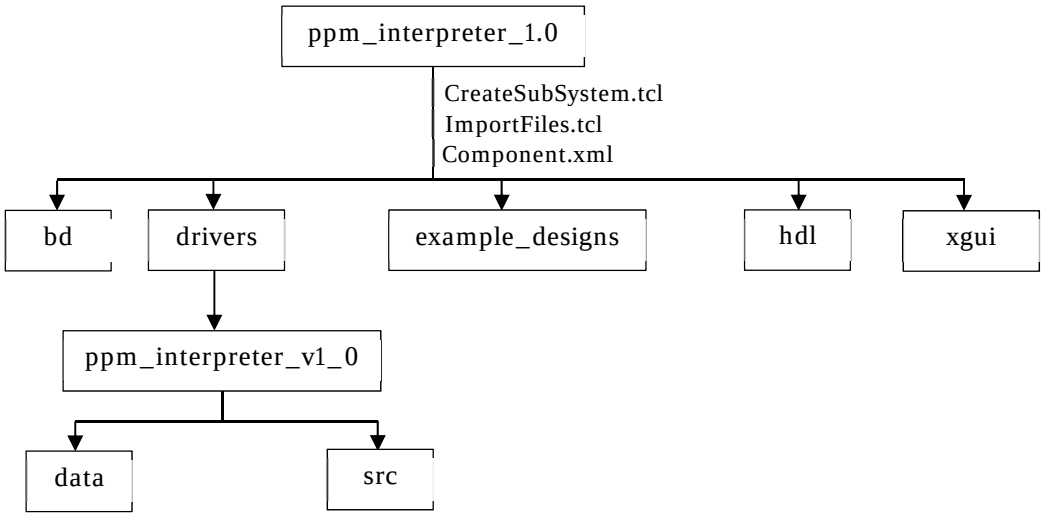


Figure 9.4.2. Xilinx User IP subdirectory structure

9.4.1.3 Example Xilinx User IP – CreateSubSystem.tcl

```
# TCL File Generated by NoC System Generator function Calc2HW version 2016
# Mon Nov 27 16:08:15 CET 2018
# DO NOT MODIFY

create_bd_cell -type hier calc2hw_p1_%d_%d
create_bd_pin -dir I -type clk calc2hw_p1_%d_%d/s_axi_aclk
create_bd_pin -dir I -type rst calc2hw_p1_%d_%d/s_axi_aresetn
create_bd_pin -dir I -from 63 -to 0 -type data calc2hw_p1_%d_%d/dap_readdata
create_bd_pin -dir O -from 13 -to 0 -type data calc2hw_p1_%d_%d/dap_address
create_bd_pin -dir O -from 63 -to 0 -type data calc2hw_p1_%d_%d/dap_writedata
create_bd_pin -dir O -type data calc2hw_p1_%d_%d/dap_read
create_bd_pin -dir O -type data calc2hw_p1_%d_%d/dap_write
create_bd_pin -dir O -from 7 -to 0 -type data calc2hw_p1_%d_%d/dap_byteenable
create_bd_pin -dir I -type data calc2hw_p1_%d_%d/noc_irq
create_bd_cell -type module -reference calc2hw_p1 calc2hw_p1_%d_%d/calc2hw_p1_0
startgroup
create_bd_cell -type ip -vlnv xilinx.com:ip:floating_point:7.1 calc2hw_p1_%d_%d/fp_add_0
set_property -dict [list CONFIG.A_Precision_Type.VALUE_SRC USER] [get_bd_cells
calc2hw_p1_%d_%d/fp_add_0]
set_property -dict [list CONFIG.Add_Sub_Value {Add} CONFIG.A_Precision_Type {Double} CONFIG.Flow_Control
{NonBlocking} CONFIG.Has_ARESETn {true} CONFIG.Result_Precision_Type {Double}
CONFIG.C_Result_Exponent_Width {11} CONFIG.C_Result_Fraction_Width {53} CONFIG.C_Accum_Msb {32}
CONFIG.C_Accum_Lsb {-31} CONFIG.C_Accum_Input_Msb {32} CONFIG.C_Mult_Usage {Full_Usage}
CONFIG.Has_RESULT_TREADY {false} CONFIG.Maximum_Latency {false} CONFIG.C_Latency {9} CONFIG.C_Rate
{1}] [get_bd_cells calc2hw_p1_%d_%d/fp_add_0]
set_property location {2 400 350} [get_bd_cells calc2hw_p1_%d_%d/fp_add_0]
endgroup
connect_bd_intf_net [get_bd_intf_pins calc2hw_p1_%d_%d/fp_add_0/M_AXIS_RESULT] [get_bd_intf_pins
calc2hw_p1_%d_%d/calc2hw_p1_0/S_AXIS_ADD_0_RESULT]
```

```
connect_bd_intf_net [get_bd_intf_pins calc2hw_p1_%d_%d/calc2hw_p1_0/M_AXIS_ADD_0_A] [get_bd_intf_pins
calc2hw_p1_%d_%d/fp_add_0/S_AXIS_A]
connect_bd_intf_net [get_bd_intf_pins calc2hw_p1_%d_%d/calc2hw_p1_0/M_AXIS_ADD_0_B] [get_bd_intf_pins
calc2hw_p1_%d_%d/fp_add_0/S_AXIS_B]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/s_axi_aclk] [get_bd_pins calc2hw_p1_%d_%d/fp_add_0/aclk]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/s_axi_aresetn] [get_bd_pins
calc2hw_p1_%d_%d/fp_add_0/aresetn]
startgroup
create_bd_cell -type ip -vlnv xilinx.com:ip:floating_point:7.1 calc2hw_p1_%d_%d/fp_mul_0
set_property -dict [list CONFIG.Operation_Type {Multiply} CONFIG.A_Precision_Type {Double}
CONFIG.Flow_Control {NonBlocking} CONFIG.Has_RESULT_TREADY {false} CONFIG.Maximum_Latency {false}
CONFIG.C_Latency {9} CONFIG.Has_ARESETn {true} CONFIG.Result_Precision_Type {Double}
CONFIG.C_Result_Exponent_Width {11} CONFIG.C_Result_Fraction_Width {53} CONFIG.C_Accum_Msb {32}
CONFIG.C_Accum_Lsb {-31} CONFIG.C_Accum_Input_Msb {32} CONFIG.C_Mult_Usage {Full_Usage}
CONFIG.C_Rate {1}] [get_bd_cells calc2hw_p1_%d_%d/fp_mul_0]
set_property location {2 400 450} [get_bd_cells calc2hw_p1_%d_%d/fp_add_0]
endgroup
connect_bd_intf_net [get_bd_intf_pins calc2hw_p1_%d_%d/fp_mul_0/M_AXIS_RESULT] [get_bd_intf_pins
calc2hw_p1_%d_%d/calc2hw_p1_0/S_AXIS_MUL_0_RESULT]
connect_bd_intf_net [get_bd_intf_pins calc2hw_p1_%d_%d/calc2hw_p1_0/M_AXIS_MUL_0_A] [get_bd_intf_pins
calc2hw_p1_%d_%d/fp_mul_0/S_AXIS_A]
connect_bd_intf_net [get_bd_intf_pins calc2hw_p1_%d_%d/calc2hw_p1_0/M_AXIS_MUL_0_B] [get_bd_intf_pins
calc2hw_p1_%d_%d/fp_mul_0/S_AXIS_B]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/s_axi_aclk] [get_bd_pins calc2hw_p1_%d_%d/fp_mul_0/aclk]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/s_axi_aresetn] [get_bd_pins
calc2hw_p1_%d_%d/fp_mul_0/aresetn]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/dap_readdata] [get_bd_pins
calc2hw_p1_%d_%d/calc2hw_p1_0/dap_readdata]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/dap_writedata] [get_bd_pins
calc2hw_p1_%d_%d/calc2hw_p1_0/dap_writedata]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/dap_read] [get_bd_pins
calc2hw_p1_%d_%d/calc2hw_p1_0/dap_read]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/dap_write] [get_bd_pins
calc2hw_p1_%d_%d/calc2hw_p1_0/dap_write]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/dap_address] [get_bd_pins
calc2hw_p1_%d_%d/calc2hw_p1_0/dap_address]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/dap_byteenable] [get_bd_pins
calc2hw_p1_%d_%d/calc2hw_p1_0/dap_byteenable]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/noc_irq] [get_bd_pins
calc2hw_p1_%d_%d/calc2hw_p1_0/noc_irq]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/s_axi_aclk] [get_bd_pins
calc2hw_p1_%d_%d/calc2hw_p1_0/Clk]
connect_bd_net [get_bd_pins calc2hw_p1_%d_%d/s_axi_aresetn] [get_bd_pins
calc2hw_p1_%d_%d/calc2hw_p1_0/Resetn]
save_bd_design calc2hw_p1_%d_%d
```

9.4.1.4 Example Xilinx User IP – ImportFiles.tcl

```
add_files -norecurse -scan_for_includes { \
 ip_repo/calc2hw_p1_1.0/hdl/memory.vhd \
 ip_repo/calc2hw_p1_1.0/hdl/calc2hw_p1.vhd \
}
```

```
import_files -force -norecurse {
 ip_repo/calc2hw_p1_1.0/hdl/memory.vhd \
 ip_repo/calc2hw_p1_1.0/hdl/calc2hw_p1.vhd \
}
```

9.4.2 Intel Altera IPs

The subdirectory referred to in the IP_block declaration should follow the Intel Altera standard for user repos.

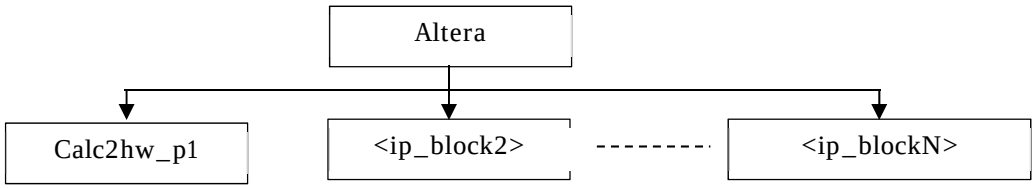


Figure 9.4.3. Intel Altera IP subdirectory structure

9.4.2.1 Using Altera Quartus inbuilt IPs

Instantiating Altera IPs is not supported in the Beta release.

9.4.2.2 Creating user IP blocks for Intel Altera Quartus

User IP blocks are created from VHDL-code and should be stored according to the IP_block standard of the target tool. The current file structure is show in Figure 9.4.4, below. In the file <ip_name>_hw.tcl, the tcl-commands needed by Quartus to instantiate the IP is stored. An example of a tcl-file is shown in section 9.4.2.3, below.

For more details on Creating own IPs, and the content of the files in the IP directories, we refer the interested reader to the Intel Altera Quartus Design Manuals.

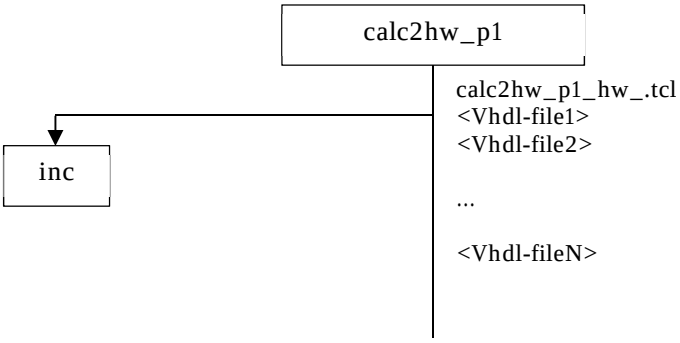


Figure 9.4.4. Intel Altera User IP subdirectory structure

9.4.2.3 Example Quartus User tcl-file

```
# TCL File Generated by NoC System Generator function Calc2HW version 2016
# Mon Nov 27 14:59:43 CET 2018
```

```
# DO NOT MODIFY

# +-----------------------------------
# |
# | p1 "p1" v10.1
# | null 2018-11-26.14:59:43
# |
# |
# | C:/Users/user/Desktop/NSG/Examples/Calc2HW_demo/Calc2HW/calc2hw_p1/calc2hw_p1.vhd
# |
# |    ./calc2hw_p1.vhd syn, sim
# |
# +-----------------------------------

# +-----------------------------------
# | request TCL package from ACDS 10.1
# |
package require -exact sopc 10.1
# |
# +-----------------------------------

# +-----------------------------------
# | module calc2hw_p1
# |
set_module_property NAME calc2hw_p1
set_module_property VERSION 10.1
set_module_property INTERNAL false
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property GROUP Calc2HW_Accelerators
set_module_property DISPLAY_NAME calc2hw_p1
set_module_property TOP_LEVEL_HDL_FILE calc2hw_p1.vhd
set_module_property TOP_LEVEL_HDL_MODULE calc2hw_p1
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE false
set_module_property ANALYZE_HDL TRUE
# |
# +-----------------------------------

# +-----------------------------------
# | files
# |
add_file std_logic_1164_additions.vhdl {SYNTHESIS SIMULATION}
add_file fixed_float_types_c.vhdl {SYNTHESIS SIMULATION}
add_file fixed_pkg_c.vhdl {SYNTHESIS SIMULATION}
add_file float_pkg_c.vhdl {SYNTHESIS SIMULATION}
add_file fp_package.vhd {SYNTHESIS SIMULATION}
add_file fp_mult.vhd {SYNTHESIS SIMULATION}
add_file fp_addsub.vhd {SYNTHESIS SIMULATION}
add_file fp_div.vhd {SYNTHESIS SIMULATION}
add_file fp_div.hex {SYNTHESIS SIMULATION}
add_file calc2hw_p1.vhd {SYNTHESIS SIMULATION}
# |
# +-----------------------------------

# +-----------------------------------
# | parameters
# |
# |
# +-----------------------------------

# +-----------------------------------
# | display items
# |
# |
# +-----------------------------------

# +-----------------------------------
# | connection point clk
# |
add_interface clk clock end
set_interface_property clk clockRate 0

set_interface_property clk ENABLED true

add_interface_port clk clk clk Input 1
add_interface_port clk reset reset Input 1
# |
```

```
# +-----------------------------------

# +-----------------------------------
# | connection point global
# |
# |
# +-----------------------------------

# +-----------------------------------
# | connection point m1
# |
add_interface m1 avalon start
set_interface_property m1 addressUnits SYMBOLS
set_interface_property m1 associatedClock clk
set_interface_property m1 burstOnBurstBoundariesOnly false
set_interface_property m1 doStreamReads false
set_interface_property m1 doStreamWrites false
set_interface_property m1 linewrapBursts false
set_interface_property m1 readLatency 0

set_interface_property m1 ENABLED true

add_interface_port m1 avm_m1_write write Output 1
add_interface_port m1 avm_m1_read read Output 1
add_interface_port m1 avm_m1_waitrequest waitrequest Input 1
add_interface_port m1 avm_m1_byteenable byteenable Output 4
add_interface_port m1 avm_m1_address address Output 32
add_interface_port m1 avm_m1_writedata writedata Output 32
add_interface_port m1 avm_m1_readdata readdata Input 32
# |
# +-----------------------------------

# +-----------------------------------
# | connection point noc_irq
# |
add_interface noc_irq interrupt start
set_interface_property noc_irq associatedAddressablePoint m1
set_interface_property noc_irq associatedClock clock
set_interface_property noc_irq associatedReset reset

set_interface_property noc_irq ENABLED true

add_interface_port noc_irq noc_irq irq Input 1
# |
# +-----------------------------------

# +-----------------------------------
# | connection point dap
# |
add_interface dap avalon_streaming start

set_interface_property dap associatedClock clock
set_interface_property dap associatedReset reset

set_interface_property dap ENABLED true

add_interface_port dap dap_address address Output 14
add_interface_port dap dap_read read Output 1
add_interface_port dap dap_write write Output 1
add_interface_port dap dap_byteenable byteenable Output 8
add_interface_port dap dap_writedata writedata Output 64
add_interface_port dap dap_readdata readdata Input 64
# |
# |
# +-----------------------------------
```

9.5 Defining User NoCs

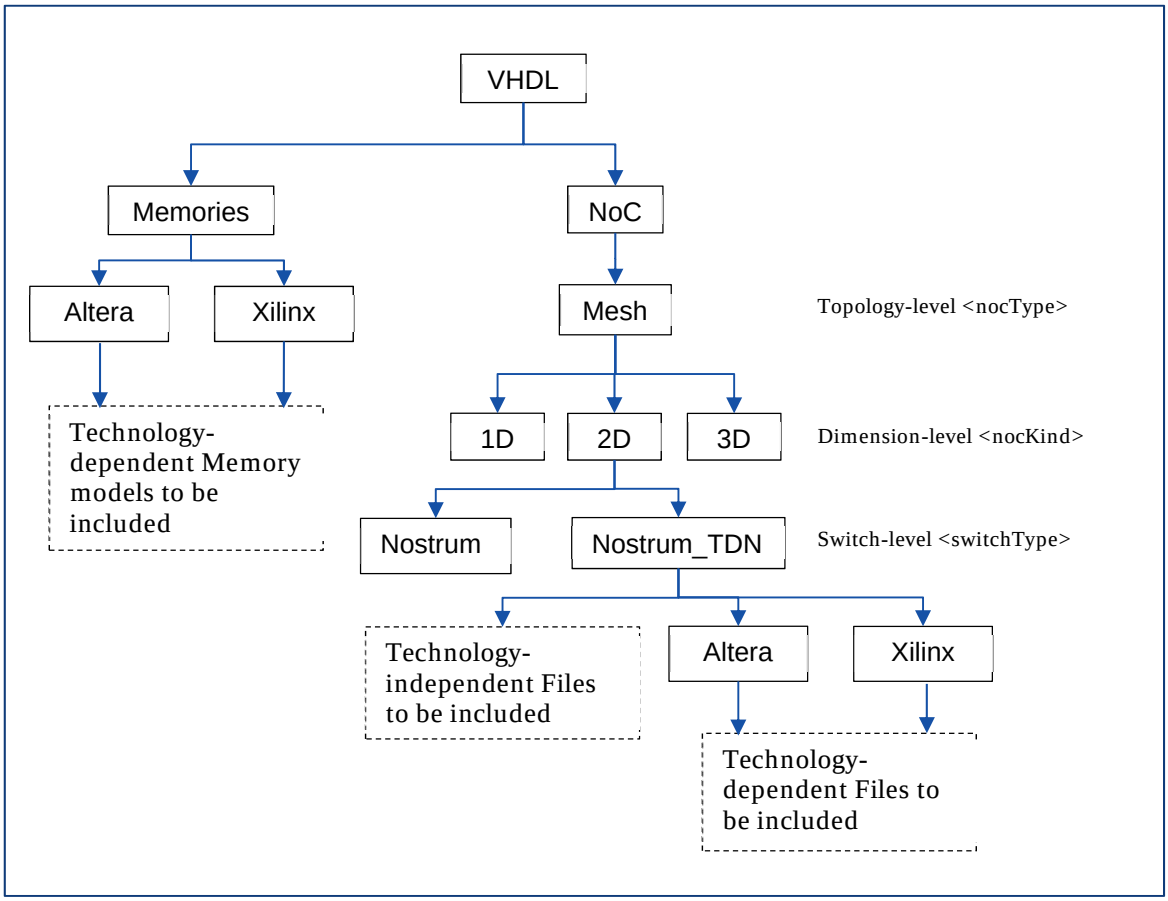


Figure 9.5.1. Directory structure for NSG NoC VHDL templates

To generate user NoCs, you first have to follow the NSG directory structure. The <nocType> is the name of the first level in the VHDL hierarchy. The dimension-level parameter <nocKind> is the name of the second level in the VHDL hierarchy. The name of the switch <switchType> indicates the third level, which is the level where all the technology-independent files are stored. NSG looks for the following files:

File	**Contents**
kth_interface_to_<switchType>_noc.vhd	This file contains the RNI ctrl interface
kth_<switchType>_rni.vhd	This file contains the RNI, and instantiates ctrl interface, and the memories for the send and recv buffers.
NoC_<nocType>_<nocKind>_<switchType>.vhd	This file contains the generic VHDL component, connected according to the

	dimensionality specified.
NoC_<nocType>_<nocKind>_<switchType>_package.vhd	This file contains the definitions required to build the NoC.
NoC_<nocType>_<nocKind>_<switchType>_Switch.vhd	This file contains the code for the switch.
NoC_<nocType>_<nocKind>_<switchType>_Switch_xmitter.vhd	This file contains the code for the xmitters in the switch.
NoC_<nocType>_<nocKind>_<switchType>_Switch_recv.vhd	This file contains the code for the receivers in the switch.
Altera specific files	
kth_avalon_<nocType>_<nocKind>_<switchType>_noc.vhd	This file contains the code for the generic top VHDL component, with technology specific interface for the Avalon bus
NoC_avalon_<nocType>_<nocKind>_<switchType>_avalon_wrapper.vhd	This file contains the definitions needed to build generic sized NoCs with Avalon bus interfaces.
Xilinx specific files	
kth_axi_<nocType>_<nocKind>_<switchType>_axi_wrapper.vhd	This file contains the code for the generic top VHDL component, with technology specific interface for the AXI bus
NoC_avalon_<nocType>_<nocKind>_<switchType>_avalon_wrapper.vhd	This file contains the definitions needed to build generic sized NoCs with Avalon bus interfaces.
kth_axi_<nocType>_rni.vhd	This file contains the mapping from the axi bus interface to the local bus interface.

Table 9.5.2 File names used during NoC generation

If the <switchType> is not set to “Nostrum” or “Nostrum_TDN”, the company part “kth” is replace with “user”.

In addition, the NSG tool generates two VHDL files:

noc_configuration.vhd	This file contains all parameters needed to configure the NoC. If any extra parameters are given that NSG does not recognize, they are replicated here with the same value given in the xml-template.
kth_<bustype>_<nocType>_<nocKind>_<switchType>_<nrCols>x<nrRows>x<NrLayers>.vhd	This file contain the top component defined in a way acceptable by the target tool. It has all bus interfaces unrolled, following the naming convention of the target tool’s standard interfaces for creating user components, with one bus-interfaces per node.

9.5.1 ImportFiles.tcl

If it is a user Noc, and the target tool is Xilinx Vivado, NSG also looks for the additional file “ImportFiles.tcl” in the technology-independent directory. In the file the required tcl-commands for importing the VHDL should be placed, in the order they should be analyzed. This gives the user more freedom in naming, since no naming convention is used. Rather, the name of the files are given in the tcl-script.

Examples of code that could be placed in the “ImportFiles.tcl”:

```
add_files -norecurse {
   ip_repo/$(NocName)_1.0/hdl/NoC_configuration_package.vhd
   $(sourceHWDirectory)/my_file1.vhd
   $(sourceHWDirectory)/xilinx/my_file2.vhd
}
```

Figure 9.5.3 Example tcl-code for ImportFiles.tcl

The *ip_repo* line refers to a subdirectory relative from to place where NSG places the generated files. All relative paths are relative to the project’s *targetDirectory*.

It is possible to replace parts of the paths with string macros that are derived during the synthesis process. The syntax is

$(<path_name>)

The valid path parameters that can be used are given in table 9.5.4. below

Parameter	**Value**
ForSyDe_Path	This parameter gives a directory string of the root directory where NSG is stored.

NoCName	This parameter produces a string with the name of the NoC being generated.
sourceHWDirectory	This parameter gives is a directory string of the source HW directory where the VHDL for the NoC is stored.
Memory_sourceHWDirectory	This parameter gives the directory string of the HW directory where the VHDL for the memory component is stored.

Table 9.5.4. Allowed macro path parameters that can be used in ImportFiles.tcl