

# FoRTReSS tutorial

---

Version 1.0j

**Fabrice MULLER (Fabrice.MULLER@unice.fr)**  
**François DUHEM (Francois.DUHEM@unice.fr)**  
**2013-05-25**

<https://sites.google.com/site/fortresstoolbox/>

UNS - CNRS

Copyright 2013 All Rights Reserved.

These computer program listings and specifications, herein, are the property of Université de Nice Sophia-Antipolis (UNS) and Centre National de la Recherche Scientifique (CNRS), and shall not be reproduced or copied or used in whole or in part as the basis for manufacture or sale of items without written permission.

For a license agreement, please contact: [licensing@sattse.com](mailto:licensing@sattse.com)

## Table of contents

I.	Introduction.....	4
1.	About this document.....	4
2.	Pre-requisites & installation notes.....	4
3.	Installation checklist for minimal FoRTReSS execution.....	5
4.	About the example used in the tutorial .....	5
II.	Getting started .....	7
1.	Creating a project and a diagram .....	7
2.	FoRTReSS configuration .....	11
3.	Generation and simulation.....	13
•	Improving the SecureBox solution .....	15
III.	FoRTReSS preferences description.....	17
1.	FoRTReSS parameters .....	17
	Target device .....	17
	Step 1: RZ determination per task.....	17
	Step 2: RZ sort .....	17
	Step 3: RZ selection for simulation.....	17
	Step 4: Allocation.....	18
	Step 5: Partial reconfiguration cost model.....	18
	Step 6: Simulation .....	18
2.	Processor parameters .....	19
3.	Application parameters.....	20
4.	Task report/Netlist .....	20
IV.	FoRTReSS timing model.....	22
1.	Reconfiguration model.....	22
2.	Simulation time .....	22
3.	Transaction-Level Modelling .....	22
V.	Advanced features.....	24
1.	Describing FPGAs using XML .....	24
2.	Using an XML description of the task resources .....	26
3.	Using an XML description of the RZ set.....	27
4.	Diagram features .....	28
5.	Interfaces description and APIs.....	30

Task execution simulation .....	30
Changing testbench behaviour.....	30
Modifying scheduling algorithm.....	31
Changing the algorithm execution mode .....	32
6. Static & Interface constraints .....	32
Interfaces behaviour .....	32
Constraining interfaces.....	33
7. Design Space Exploration of the paper .....	36
8. Debugging a simulation (Windows only).....	37
VI. References.....	38
Change log .....	39

## I. Introduction

### 1. About this document

This document is a step-by-step tutorial on how to use FoTRReSS to perform design space exploration of partially reconfigurable systems. It is meant for designers that want to perform design space exploration of partially reconfigurable architectures quickly using FoTRReSS flow. More information about the flow can be found in the paper provided with the tool [1].

### 2. Pre-requisites & installation notes

First, you have to copy on your computer the right FoTRReSS directory depending on the operating system. For example, for Windows 32bits, copy the FoTRReSS directory contained in *win32.win32.x86* directory.

FoTRReSS requires the Java Runtime Environment 7 (JRE7) to be installed on the computer. The path to the virtual machine should be added to the Path environment variable. For Linux users, just unzip the JRE package into the FoTRReSS folder and name it *jre*. Note that JRE7 is included in "additional files" directory.

SystemC 2.3 must be installed on the computer in order to use FoTRReSS. They can be found here: <http://www.accellera.org/home/>. For Windows users, SystemC requires Visual Studio 2010 (32 bits only), so does FoTRReSS (versions 2010 and 2012 (32 bits), can be the free express version that can be found at this address: <http://www.microsoft.com/visualstudio/eng/downloads#d-2010-express>). SystemC installation notes ask the user to build the Debug version of SystemC. As FoTRReSS uses the Release version in order to speed up simulation, this configuration has to be built as well. Note that we directly provide SystemC 2.3 compressed file for Visual Studio 2010 and 2012, 32 bits. For Linux users, path to the dynamic library (*<path\_to\_systemc\_install/lib-{arch}>*), where *arch* is the system architecture, e.g. linux64) should be added to the LD\_LIBRARY\_PATH environment variable.

FoTRReSS also uses XercesC, a portable XML parser that can be found at this address: <http://xerces.apache.org/xerces-c>. Binary releases can be used for installation (the 32-bit version is required for Windows users, even on 64-bits operating systems, Linux users should use the version corresponding to their architecture). Please note that FoTRReSS needs access to Xerces dynamic library (.so or .dll depending on the operating system). Windows users should add *<path\_to\_xerces\bin>* to the PATH environment variable whereas Linux users should add *<path\_to\_xerces/lib>* to the LD\_LIBRARY\_PATH variable.

Modifying the LD\_LIBRARY\_PATH variable only works when FoTRReSS executable is launched using the same terminal (unless the variable is initialized in the .bashrc file). For users that want to launch FoTRReSS through the graphical user interface, the following permanent workaround is possible: create local.conf in the subdirectory /etc/ld.so.conf.d and add: *<path\_to\_xerces/lib>* as well as *<path\_to\_systemc/lib-{arch}>*. Then: *sudo ldconfig*.

Note that XERCES (compiled for Visual C++ versions 2010 and 2012, 32 bits) and SystemC 2.3 (compressed files) are provided in "additional files" directory. You just have to copy and extract them to the FoTRReSS installation path.

In order to access all FoTRReSS functionalities, some additional programs may be installed:

- Mentor Graphics ModelSim 6.6+ to view simulation results (free version is sufficient)

- Xilinx PlanAhead 12.x+ to view placement results and enable bitstream generation options in FoRTReSS flow

### 3. Installation checklist for minimal FoRTReSS execution

#### Linux Users

- Install Java Runtime Environment 7 (JRE7 32 bits or 64 bits)
- Unzip “*systemc-2.3.0-x86-and-x86\_64-linux.tar.gz*” OR compile SystemC 2.3
- Unzip XercesC binary release (“*xerces-c-3.1.1-x86-linux-gcc-3.4.tar.gz*” for linux 32 bits or “*xerces-c-3.1.1-x86\_64-linux-gcc-3.4.tar.gz*” for linux 64 bits) OR compile XercesC
- Update LD\_LIBRARY\_PATH and/or modify /etc/ld.so.conf.d/local.conf to include SystemC and XercesC *lib* directories

#### Example for Environment Variables (32 bits)

```
export SYSTEMC_PATH==/home/user/FoRTReSS/systemc-2.3.0
export SYSTEMC_LIB_PATH==/home/user/FoRTReSS/lib-linux
export XERCES_PATH=/home/user/FoRTReSS/xerces-c-3.1.1-x86-linux-gcc-3.4
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$XERCES_PATH/lib:$SYSTEMC_LIB_PATH
```

#### Example for Environment Variables (64 bits)

```
export SYSTEMC_PATH==/home/user/FoRTReSS/systemc-2.3.0
export SYSTEMC_LIB_PATH==/home/user/FoRTReSS/lib-linux64
export XERCES_PATH=/home/user/FoRTReSS/xerces-c-3.1.1-x86_64-linux-gcc-3.4
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$XERCES_PATH/lib:$SYSTEMC_LIB_PATH
```

#### Windows users

- Install Java Runtime Environment 7 (JRE7 32 bits or 64 bits)
- Install Visual C++ 32 bits only (2010 or 2012 Express/standard edition),
- Unzip “*systemc-2.3.0-x86-windows-vc-10.0-11.0.zip*” OR compile SystemC 2.3 (both Debug and Release configurations),
- Unzip XercesC binary release (32 bits only, “*xerces-c-3.1.1-x86-windows-vc-10.0.zip*” for Visual C++ 2010 or “*xerces-c-3.1.1-x86-windows-vc-11.0.zip*” for Visual C++ 2012) OR compile XercesC,
- Update the PATH environment variable to include XercesC *bin* directory

#### Example for Environment Variable, Visual C++ 2010 (32 bits only)

```
SET PATH = %PATH%;C:\eclipse\eclipse\xerces-c-3.1.1-x86-windows-vc-10.0\bin
```

#### Example for Environment Variable, Visual C++ 2012 (32 bits only)

```
SET PATH = %PATH%;C:\eclipse\eclipse\xerces-c-3.1.1-x86-windows-vc-11.0\bin
```

### 4. About the example used in the tutorial

We use a custom made example named SecureBox. It is responsible for securing video streams from a camera that are to be transmitted to a public domain (either transmission chain or a hard drive). For this purpose, the chain is composed of an MPEG-2 encoder, an AES encoder for data encryption and finally a Reed-Solomon (RS) encoder to prevent data corruption from transmission and/or storage.

The project resulting from this tutorial is also provided as an example. The source files are located in [FoRTReSS install]/tutorial/tutorial.zip.

## II. Getting started

### 1. Creating a project and a diagram

1. Launch FoRTReSS. Create a new project with *File* → *New* → *Project*. A dialog window pops up. Fill in the project name (for instance tutorial) and the project location (here, C:\app\FoRTReSS\_tutorial).

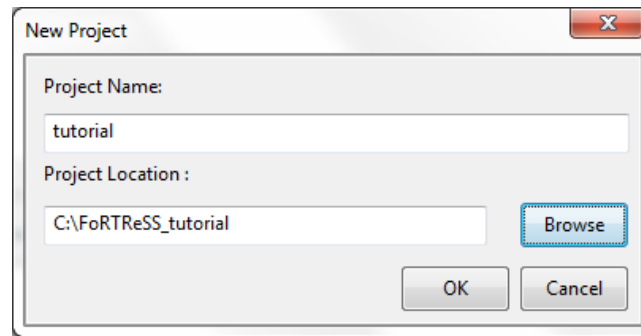


Figure 1 - Creating a project

2. Click *OK*. We now have to create a diagram that will represent the application. Click *File* → *New* → *Modulesystem Diagram*. A new window opens to ask the location of the diagram. Keep default location. Diagram name might be changed (here, from default to test\_application). Click *Next* and *Finish*.

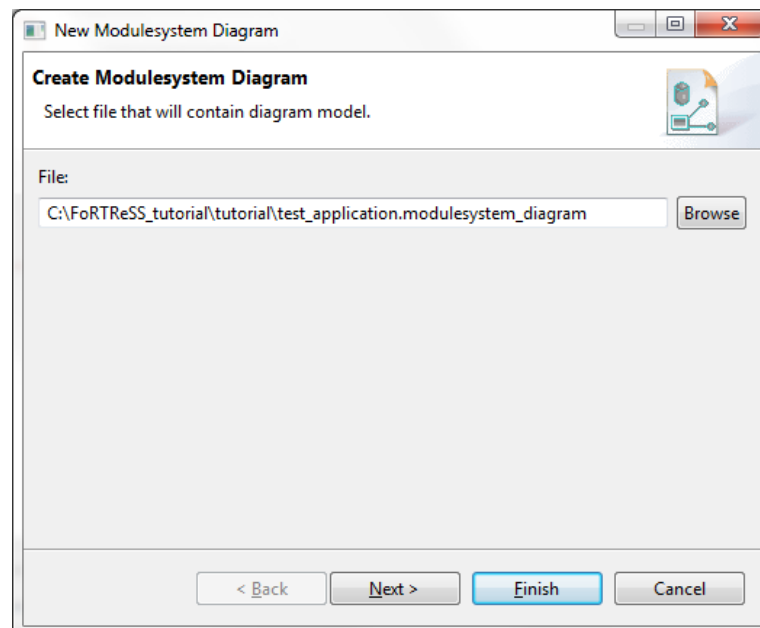


Figure 2 - Creating a diagram

The main view should now be displayed on the screen. It is composed of a diagram view, where diagrams can be edited using the palette, an outline that displays the entire diagram, a console to display information about the flow and finally a properties view that will be used to configure diagram components. Now, we will edit the diagram.

- Using the palette, insert two testbench (in and out) and six modules as pictured in the next capture. Also create connections between every component. Modules represent tasks of the Directed Acyclic Graph (DAG).

Now, we have to define implementations for each module. They can be either hardware (i.e. to be implemented on a reconfigurable zone) or software (i.e. to be implemented on a processor). In this use case, we will focus on hardware implementation.

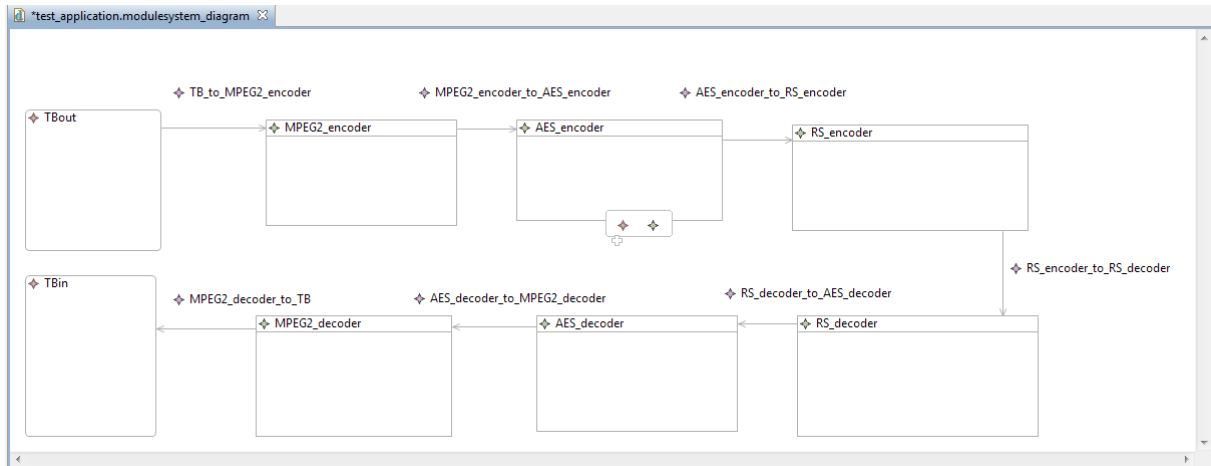


Figure 3 - SecureBox application block diagram

- Declare one hardware implementation for each module and call it "HWImpl".

At this point, the application has to be configured (resources, communication times and so on).

- Configure TBout according to the following capture. Keep TBin In with default values.

TestBench		
Core	Property	Value
Appearance	1. General	
	Period	33300
	Test Bench name	TBout
	Test Bench Thread	
	2. Results	
	Activate trace	true
	Enable configuration signals trace	true

Figure 4 - Testbench Out configuration

- Period:** Task periodicity. In the case of our custom application, we want to process 30 frames per second (fps) so that a frame should be sent to the system every 33.3 ms (all times considered in FoRTReSS are defined in microseconds).
- Testbench thread:** Thread associated with the module. FoRTReSS provides the user with a default thread that will be used in this tutorial. This default thread sends dummy packets to the system and verifies that the packets received after being processed by the chain contain the same data. Using a user-defined thread will be described in the second part of this tutorial.



- **Activate trace:** Do/don't trace module signals into the main trace. RecoSim generates a trace in VCD format (Value Change Dump) after each simulation that can be opened by GTKWave or ModelSim to observe the behaviour of the system during the simulation.
- **Enable configuration signals trace:** Do/don't trace signals related to configuration in the main VCD trace file.

6. Configure each connection the same way as depicted in the following capture.

Core	Property	Value
Appearance	1. General	
	Begin req	0
	Begin resp	0
	Connection name	TB_to_MPEG2_encoder
	End req	0
	End resp	0
	Packet size	16
	Target Block	Module MPEG2_encoder
	Transient channel	false
	2. Input	
	Input communication type	
	Input priority channel	false
	3. Output	
	Output communication type	
	Output priority channel	false

Figure 5 - Connections configuration

- **Begin req, Begin resp, End req and End resp:** Timing information used in TLM to delay communications between sockets (more information on the TLM Language Reference Manual [2]). In our case, they can be left to zero.
- **Packet size:** Size of the transactions exchanged between sockets in words.
- **Transient channel:** True if the channel is transient, i.e. the channel does not expect any transaction during the first hyperperiod (typically used when the DAG is cyclic). The use of this attribute will be demonstrated later.
- **Input/Output communication type:** interface type (e.g. AXI, FIFO, PLB ...). It should be left empty for this first approach as it will be explained in details in a dedicated section.
- **Input/Output priority channel:** channels that have priority on communication (i.e. not sharing a physical channel but rather use a dedicated channel on the reconfigurable zone). When there is only one connection on the module, this parameter does not change system behaviour.

7. Configure each module the same way:

Core	Property	Value
Appearance	1. General	
	Begin time	0
	Deadline	33300
	Module name	MPEG2_encoder
	Period	0
	Priority	0
	Static module	false
	3. Results	
	Activate Trace	true
	Enable configuration	false

Figure 6 - Modules configuration

- **Begin time:** Time at which the task can be executed for the first time (task offset)
  - **Deadline:** Task deadline. Here, it is 33300  $\mu$ s (33.3 ms) in order for the system to process 30 frames per second.
  - **Period:** Task period. When set to 0, it indicates that the system is considered as “full dataflow”, with no period consideration for the tasks. When set to another value, the system should also respect periodicity: there can be one and only one task execution per period.
  - **Priority:** Task priority, can be used by the scheduler.
  - **Static module:** If true, the task is always implemented on the FPGA and does not require reconfiguration.
8. Configure implementations as depicted in next capture. Adapt it to the different tasks: change field *Synthesis report netlist* to fit module’s name. MPEG2 tasks have an execution time of 5300  $\mu$ s where it is 8100  $\mu$ s for AES tasks and 10000  $\mu$ s for RS tasks.

ImplementationHard		
Core	Property	Value
Appearance	1. General	
	Implementation name	HWImpl
	Interface types	
	Netlist	AES_encoder
	Resource margin	0
	Task report	AES_encoder
	2. Execution	
	Best case execution time	8100
	Context swith time	0
	Energy consumption	500
	Frequency	100
	Use context switch	true
	Worst case execution time	8100
	3. Algorithm	
	Algorithm name	
	Nb preemption points	0
	Use algorithm	false

Figure 7 - Implementations configuration

- **Interface types:** Type of interfaces used by this particular implementations. An implementation might provide a minimal number of interfaces to reduce configuration time and/or resource utilization but it will also lower system performance as these physical

interfaces might be shared by several virtual channels (i.e. the different connections for a module in the DAG).

- **Netlist:** Netlist name (without extension). The netlist can be used, depending on FoRTReSS configuration, to build actual configuration bitstreams and estimate compression ratios. NGC and EDF netlists are accepted.
- **Resource margin:** margin between task requirements and reconfigurable zone resources. This way, it is possible to give the router extra resources to prevent some implementation failures. Xilinx recommends a 5 to 10 percent margin to ensure good behaviour during the implementation routing phase.
- **Task report:** Name of the file containing resource information for the task. It can be either a synthesis report generated by Xilinx ISE (.syr) or a generic XML file (.tsk). The XML file is described in a further section.
- **Best/Worst case execution time:** Distinction between BCET and WCET is made. The selection between both times is made by the reconfiguration manager at runtime. By default, it only uses the WCET.
- **Context switch time:** Time required to save/restore context for this implementation.
- **Energy consumption:** Energy consumption for one task execution.
- **Frequency:** Operating frequency.
- **Use context switch:** Determines if a context switch is required after configuring this task. See *FoRTReSS timing model* section for more details on this option.
- **Algorithm name:** Name of the algorithm that is used when the *Use algorithm* parameter is true. If *Use algorithm* is set to false or no algorithm name is specified, the default algorithm will only simulate the task execution time (simple *wait* instruction).
- **Nb preemption points:** Number of preemption points to consider when using default algorithm. They are placed so that the different execution segments are equals. User-defined algorithms also have access to this piece of information (more detailed explanation in the dedicated section).
- **Use algorithm:** If true, use the algorithm named *Algorithm name*. Else, simulate task execution by a *wait* statement.

Now, the application is completely specified, but FoRTReSS still has to be configured.

## 2. FoRTReSS configuration

1. Under the menu FoRTReSS, click *Preferences*. Select your PlanAhead version and, Visual C++ version and fill in the fields depending on your installation:
  - **PlanAhead Path (optional):** path to Xilinx PlanAhead executable
  - **SystemC path:** path to SystemC 2.3 root directory
  - **SystemCLib path:** path Release and Debug directories containing file systemc.lib. FoRTReSS uses the Release configuration for speed purpose.
  - **ModelSim (optional):** path to ModelSim executable
  - **Text Editor (optional):** Path to your favourite text editor. It is used to view/edit files used by FoRTReSS
  - **MSBuildPath (Windows only):** path to MSBuild.exe, a tool installed with Visual C++ that is needed for compilation. It can be found in the Microsoft.net folder. Version 4.x is for Visual C++ versions 2010 and 2012.

- **Visual C++ Path (Windows only, optional):** path to Visual C++ executable used to open the solution for Debug purposes.
- **Xerces Path:** Path to your Xerces library

In order to remember these paths for future use, click the *Save Template Path* button. Default paths can be restored later by clicking the *Restore Path* button.

2. Create a new configuration by clicking on *new* next to the configuration selector. Keep name *Config\_1* and click *OK*. Now the other tabs can be opened in the Preferences window.

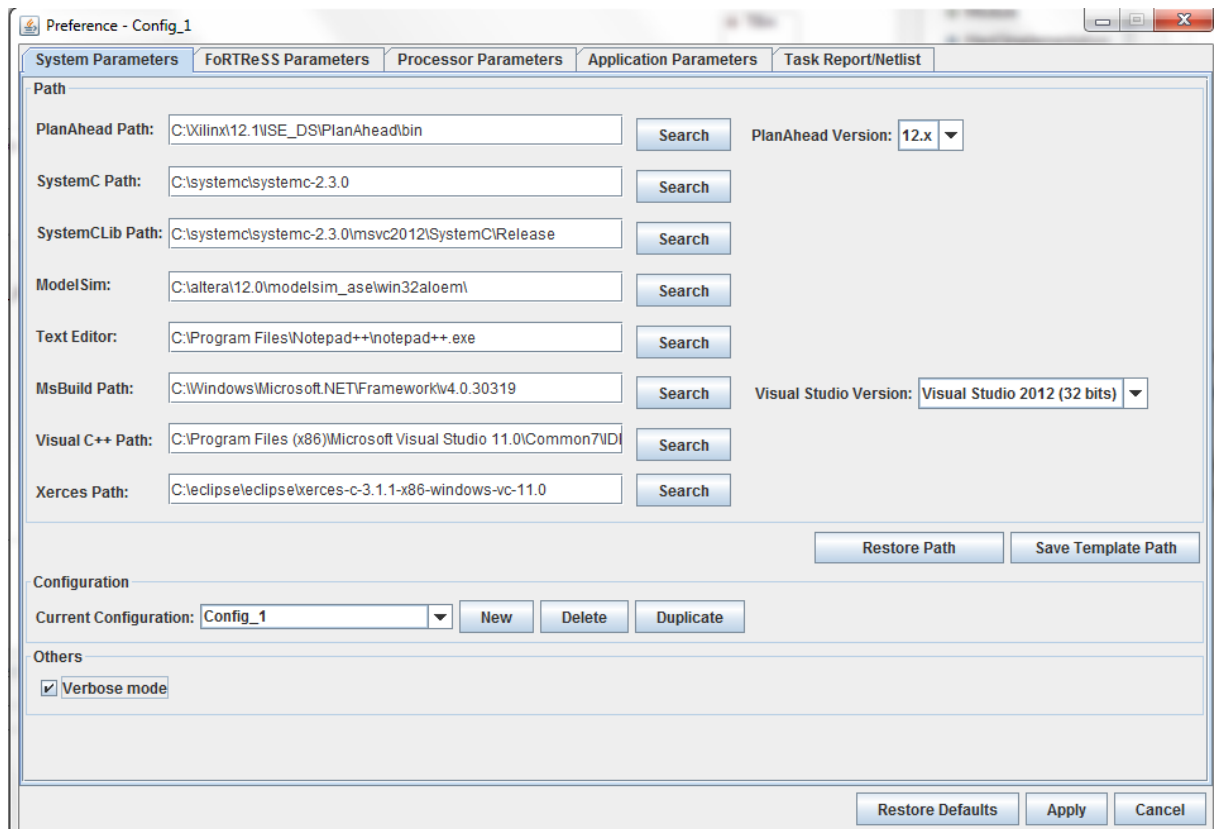


Figure 8 -- FoTReSS system parameters

3. Go to the *FoTReSS parameters* tab. Parameters for each step of the flow is set to a default value.
4. Go to the *Application Parameters* tab. Here, you can manage applications. For now, we only have defined one application (i.e. on diagram) but it is also possible to mix different applications with different requirements that should be implemented on a single FPGA. In such case, reconfigurable zones and processors are shared between applications.
5. Add the previously defined application by clicking on *Add*. Select *test\_application.modulesystem* that should be in the current folder if paths were respected since the beginning of this tutorial. This application is parameterized in such manner that it should achieve an optimum quality of service (100%, i.e. all tasks must respect their deadlines at all means) and with an energy consumption of 10 mJ. Note that applications might be easily duplicated by using the dedicated button.
6. Go to the *Task Report/Netlist* tab. Here, we will reference the synthesis reports and netlists that are used to describe implementations (NGC and EDF formats are accepted). They can be either

imported (copied to the working project directory) or referenced. Click *import* and then go to the folder netlists of the tutorial and select every file. Do the same for synthesis reports. Now, the window should look like this:

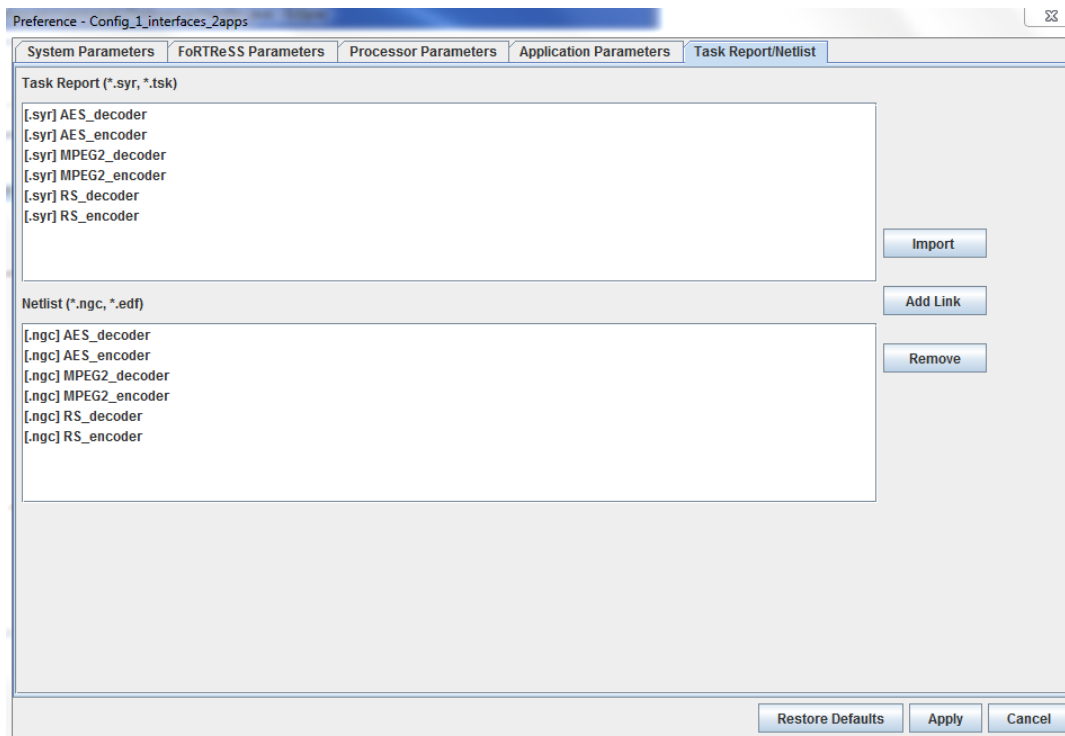


Figure 9 - Adding synthesis reports and netlists to the project

Note that once task reports and/or netlists have been imported or linked to the current project, it is possible to associate them to an implementation directly on the application diagram by right-clicking on the implementation and choosing *Select Netlist* (respectively *Select Task Report*).

7. Finally, click *Apply* to save changes made on FoTRReSS preferences.

### 3. Generation and simulation

At this point, everything required by FoTRReSS is described and the flow is entirely parameterized. Now, we have to generate the source code corresponding to the use case, compile it and simulate.

- Under menu FoTRReSS, choose *Generate* → *Current config*
- Under menu FoTRReSS, choose *Compile* → *Current config*
- Under menu FoTRReSS, choose *Simulate* → *Current config*

Note that launching simulation will automatically generate and compile code if it is not up-to-date. The simulation should have been successful after a few iterations.

We might now take a look at the generated solution:

- To see the trace file generated by RecoSim, under the menu *FoTRReSS*, choose *View Trace* → *Current config* and then choose which simulation trace to view (requires ModelSim).
- To see the placement generated by FoTRReSS, under the menu *FoTRReSS*, choose *View Placement* → *Current config* and choose a solution (requires PlanAhead).

- To view simulation results from a simulation compiled in a CSV file, under menu *FoRTReSS*, choose *View Statistics* → *Current config* and choose a solution. For instance, for the final solution (the one selected by FoRTReSS), results should be:

```

FoRTReSS Transcript
*****
View Statistics (configuration : Config_1)

Achieved Simulation Time : 3364477.045 us
Minimum Simulation Time : 80100.0 us
Application Offset      : 46800.0 us
Application Hyperperiod : 33300.0 us

=====
RZ number : 2 [RZ_353552, RZ_353569]
-----
Blank [0.0148612, 0.249397]
Reconfiguring [19.9489, 11.0044]
Mapped [5.78449, 19.0915]
Active [74.2517, 69.6547]
-----
Running [74.2517, 69.6547]
MPEG2_encoder [15.7528, 0.0]
HWImpl [15.7528, 0.0]
AES_encoder [20.9453, 3.26513]
HWImpl [20.9453, 3.26513]
RS_encoder [26.1556, 3.56668]
HWImpl [26.1556, 3.56668]
RS_decoder [6.24168, 27.0694]
HWImpl [6.24168, 27.0694]
AES_decoder [3.26593, 21.6073]
HWImpl [3.26593, 21.6073]
MPEG2_decoder [1.89034, 14.1462]
HWImpl [1.89034, 14.1462]
=====

Processor number : 0

Reconfiguration units occupation rates (in % of the simulation time)
  HW reconfiguration unit 0 [30.9533]
  SW reconfiguration unit 0 [0.0]

Scheduler
  Occupation rate (in % of the simulation time) [11.6957]
  Energy consumption (mJ) [0.0]

Achieved QoS factor for application (in %)
  test_application/App_0 [100.0]
|
Minimal QoS factor on an hyperperiod (in %)
  test_application/App_0 [100.0]

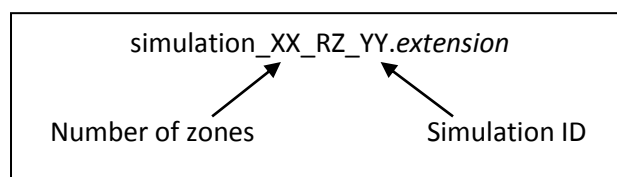
Static implementation versus PR implementation
  Resource type [Static area, PR area, Raw improvement (%), Total improvement (%)]
  BRAM [18.0, 24.0, -33.33, -33.33]
  Slice [3938.0, 2560.0, 34.99, 18.79]
  DSP [26.0, 32.0, -23.08, -23.08]

```

Figure 10 - Simulation statistics

### Naming conventions

Simulation log and trace files are named according to the following naming convention:



The number of reconfigurable zones is incremented until a first valid solution has been found. Then, the number of zones remains constant and the simulation ID is incremented with the number of allocation optimizations performed. When the simulation finally fails, FoTRReSS elects the previous solution as the final solution, named for which log and trace files are named `final_solution.extension`.

### • Improving the SecureBox solution

As we can see with the results from FoTRReSS execution, the PR solution introduces an area overhead compared to the static solution in order to satisfy the application real-time constraints, which clearly excludes using PR for this application. However, tuning some parameters might change the system behaviour and improve PR area occupation.

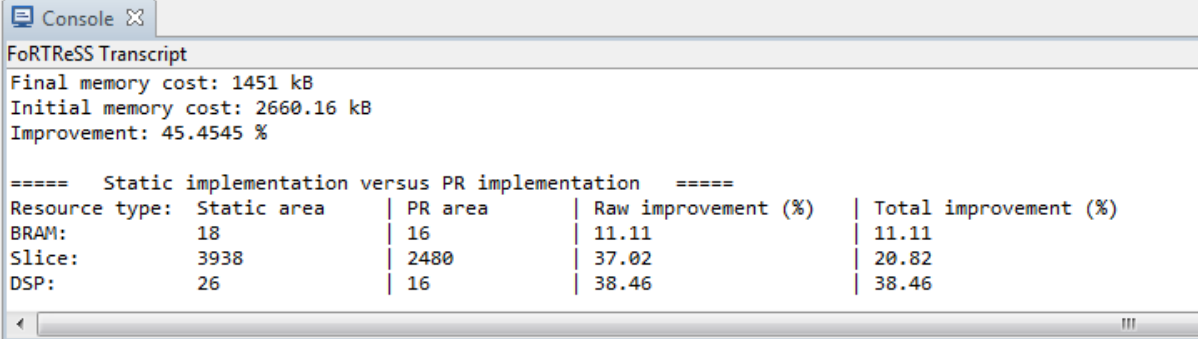
The console shows that FoTRReSS partitioning step failed. Launching another simulation in verbose mode gives us more information about this failure.

```
Partitioning RZ space...
Maximum task cost is for ID 0 (MPEG2_encoder.HWImpl)
Optimum task vector contains: MPEG2_encoder.HWImpl
Acceptable task vector contains: MPEG2_decoder.HWImpl
Unacceptable task vector contains: AES_encoder.HWImpl RS_encoder.HWImpl RS_decoder.HWImpl AES_decoder.HWImpl
Number of RZs hosting all the tasks: 1
Aggregate runtime for optimum task subset: 15.7524 (1 RZs)
Aggregate runtime for unacceptable task subset: 107.592 (2 RZs)
```

Figure 11 - Partitioning step

It seems that task partitioning is very heterogeneous as there are too many unacceptable tasks and too few optimum tasks for the partition to be efficient. Hence, we have to modify the corresponding trigger in step 3 parameters. We need to reduce both trigger values in order to have less unacceptable tasks and more optimum tasks. Decent values can be found by performing several iterations (and simulations). In our case, an acceptable trigger value of 12% and an optimum trigger value of 55% lead to a different PR solution which requires less area resources than a static implementation.

It is also possible to optimize the task allocation in order to reduce PR solution memory footprint (option that is disabled by default). This can be done by tuning both step 4 parameters and choosing one optimization strategy. For instance, choosing shortest runtime for both parameters lead to a 45% reduction of the overall bitstream size:



```
==== Static implementation versus PR implementation ====
```

Resource type:	Static area	PR area	Raw improvement (%)	Total improvement (%)
BRAM:	18	16	11.11	11.11
Slice:	3938	2480	37.02	20.82
DSP:	26	16	38.46	38.46

Figure 12 - Area results after optimization

This improved version is described within the configuration `Config_1_improved` in the project provided with the tutorial.





### III. FoRTReSS preferences description

#### 1. FoRTReSS parameters

In the first part of this tutorial, we kept default values for every FoRTReSS parameters. In this section, we will present parameters that can be tuned for each step of the flow.

##### Target device

- **Selected device:** Choose the FPGA device to consider. We only provide a limited set of FPGA descriptions (xc6vlx240t and xc5vfx70t). Other devices can be described by the user (Xilinx or any other device) via an XML description file. This description is explained in the corresponding section.
- **Static & Interface Constraints:** Add constraints to the FPGA. An entire zone of the FPGA can be defined as static (i.e. prohibiting reconfigurable regions to be placed in this area). Also, interfaces can be placed on the FPGA (for instance, defining a set of AXI4 interfaces on the FPGA to constrain placement of reconfigurable region using this type of interface). More on this subject in the dedicated section.
- **Default Costs:** Costs are defined for each type of reconfigurable resources. Basically, this means that wasting a column of CLB is less critical than wasting a column of BRAM because of the amount of resources available on the FPGA. Default costs already take this into account, but they can be overridden by the user.

##### Step 1: RZ determination per task

- **Use RZ set:** Instead of searching for reconfigurable zones, FoRTReSS can also use an RZ set described in an XML file by the user.
- **Purely reconfigurable RZ:** if selected, reconfigurable zones should not be designed over non-reconfigurable resources (even if they cannot be constrained).
- **Allow for non rectangular RZ shapes:** Allows generating non-rectangular zones with respect to some shape guidelines. RZ are optimal or near-optimal in terms of resources but in return, routing might sometimes be complicated.
- **Oversize RZ trigger:** in order to remove RZs for which internal fragmentation is too important. For instance, if trigger value is 10, it means that RZs that constrains at least 10% more resources than necessary will not be added to the working set.
- **Resources margin:** Global resources margin for determining the reconfigurable zone resources. Can be redefined individually for each hardware implementation (in the implementation properties).

##### Step 2: RZ sort

- **Shape cost, Compliance cost & Internal fragmentation cost factors:** Allows tuning the cost formula to emphasize either on RZ shape, task compliance or internal fragmentation.
- **Check for RZ redundancy:** for removing duplicates in the RZ list, improving exploration time.

##### Step 3: RZ selection for simulation

- **Implementation metric:** Only one metric implemented yet...
- **Trigger factors:** Refer to triggers used to partition the application. Please refer to the paper provided with the tutorial for more details.

- **Margin to reduce congestion:** Keep a certain amount of columns to separate reconfigurable zones and thus reduce congestion due to partition pins. This separation is only vertical (meaning that top and bottom of two reconfigurable zones might coincide).

#### Step 4: Allocation

- **RZ Allocation strategy:** Strategy to improve allocation.
  - *Shortest runtime:* the least used implementation is removed
  - *Highest fragmentation:* the task with the highest internal fragmentation on a reconfigurable zone is removed
  - *Highest memory cost:* The task with the highest memory cost is removed
- **Processor Allocation strategy:** Same as above, but for processors. Strategies include shortest runtime and highest memory cost.

#### Step 5: Partial reconfiguration cost model

- **Bitstream generation:** Determines whether bitstream generation has to be done to estimate compression. If selected *None*, the compression rate defined right after this parameter is used as an estimation. If *PlanAhead Implementation* is selected, a PlanAhead implementation is performed every time to estimate every compression ratio. This solution is very time consuming, considering that generating a bitstream can take up to 30 minutes depending on the device and that there might be a lot of estimations to do. However, the achieved solution is very close to reality. If *Final PlanAhead verification* is selected, only a final verification is done using actual bitstream compression ratios, which is a good compromise between time and efficiency. When a PlanAhead implementation is required for an implementation with no netlist defined, FoRTReSS will use the compression ratio defined below instead.
- **Compression rate:** Compression ratio estimation
- **Tbus:** Bus period
- **Ticap:** ICAP period
- **Nburst:** Burst accesses size
- **Tburst:** Time needed to perform such a burst
- **Latency:** Initial latency for each bus access
- **FIFO depth:** FaRM FIFO depth

More details about partial reconfiguration cost model can be found in [3].

#### Step 6: Simulation

- **Scheduler Queue Strategy:** Choose a strategy for sorting tasks inside the waiting queue. By default, uses an *Earliest Deadline First (EDF)* strategy.
- **Scheduler strategy:** Choose a strategy for the scheduler. By default, uses an *As Much As Possible (AMAP)* strategy (uses as much processing elements as possible). Both strategies can be modified and adapted to the application needs using an API. More details in the dedicated section.
- **Scheduler overhead:** Estimation of the time needed by the scheduler to decide
- **Scheduler Performance/Area/Energy Effort:** Specify which of the three design metrics should be promoted when searching for reconfigurable zone. Note that the default scheduler strategies only consider the performance metric, even though user-defined strategies can access the metric values.

- **Scheduler energy consumption:** Estimation of the energy needed for each scheduler decision.
- **Energy consumption limit:** Upper limit for energy consumption.
- **Hw Reconfiguration Units:** Number of hardware reconfiguration units that can run in parallel.
- **Sw Reconfiguration Units:** Number of software reconfiguration units that can run in parallel.
- **Maximum simulation time:** Guard for unknown behaviour of RecoSim. Should be enough for the testbench to execute properly.
- **Generate zero-filled lines in CSV:** Whether or not zero-filled lines should be included in CSV file.
- **Enable debug signals in trace:** Add some debug signals to the trace of each simulation
- **Enable configuration signals in manager's trace:** Add configuration signals in the manager's trace.

## 2. Processor parameters

As we already mentioned, it is also possible to perform HW/SW co-design by providing the simulator with a set of processors that can be used. These are tuned in the *Processor* tab of the *FoRTReSS preferences* window. In order to create a processor, click the *Add* button. Enter processor type (e.g. MicroBlaze, PowerPC and so on) and click OK. The processor is now created with default values for its parameters.

- **Instance name:** name of the processor
- **Active:** Activate/deactivate processor. Inactive processors are still in the processor list and can be reactivated. To effectively remove a processor, click on the *Remove* button while the processor to be removed is selected in the list. Processors might also be duplicated.
- **Context Switch Time:** time required to switch between software tasks.
- **Interfaces:** interfaces provided by the processor, used to see if tasks can fit.

Processors can be added multiple times, as the next capture illustrates.

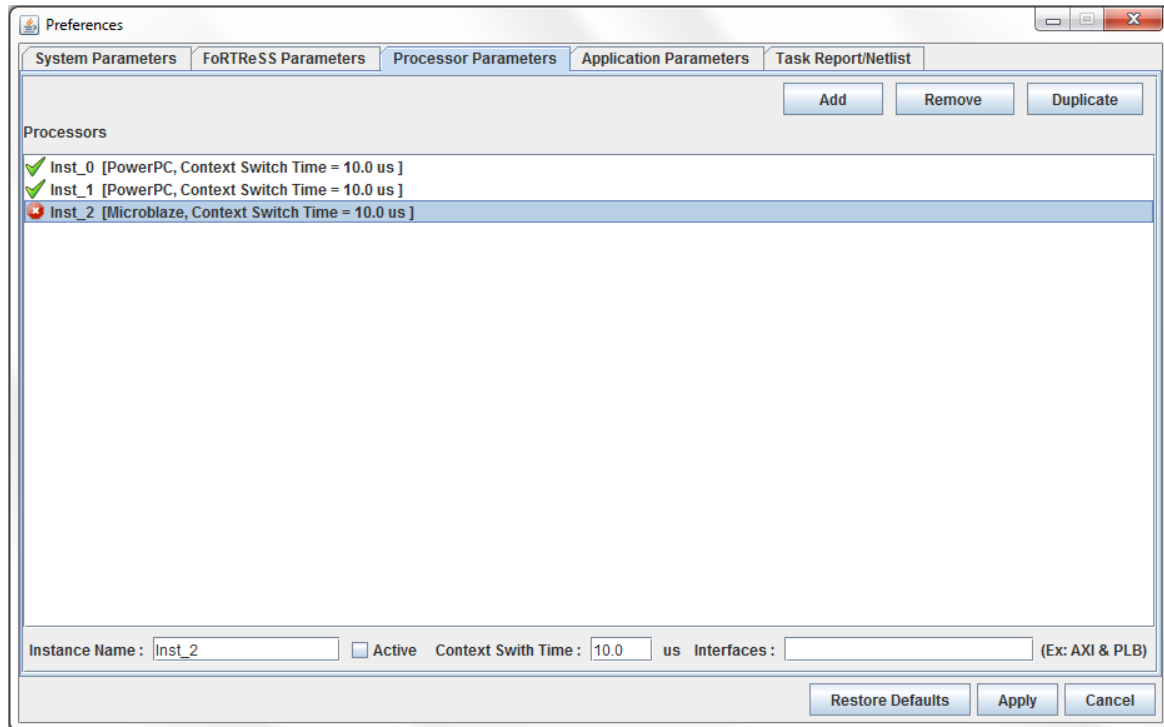


Figure 13 - Adding processors to the simulation

The, software implementations can safely be added to modules in the diagram view. Parameters available for such implementations are the following (parameters already defined for hardware implementations put aside):

- **Processor type:** Type of processor compatible with this implementation. If no value is set, the implementation is considered compatible with every processor in the design.
- **Binary loading time:** Time spent loading the binary (equivalent of the hardware reconfiguration).

### 3. Application parameters

The *Application parameters* tab of FoRTReSS preferences allows the user to manage applications that should be simulated. It is possible to add, remove and duplicate applications. Each instance can be separately activated/de-activated and parameters such as Quality-of-Service (QoS) and energy consumption limit can be tuned. Different applications (i.e. different diagrams) can be combined inside a single simulation. Several applications can be active at the same time: in such case, the applications have to run on a single FPGA at the same time, always respecting their timing constraints.

### 4. Task report/Netlist

In this tab, the user can list the task descriptions and netlists that will be used during FoRTReSS execution. Task description can be either an XML file of a synthesis report file issued by Xilinx synthesizer (.syr). Files can be either imported (i.e. copied into the tasks directory), producing a portable solution, or linked to the project (i.e. referenced by the project), reducing project size and allowing reuse of netlists and task descriptions.

Once task reports and/or netlists have been managed in this tab, it is possible to graphically assign them to an implementation by right-clicking on the implementation and choosing *Select Netlist* (respectively *Select Task Report*).

## IV. FoRTReSS timing model

### 1. Reconfiguration model

Figure 14 shows the execution timeline for two tasks sharing a single processing unit (reconfigurable zone or processor core) with preemption enabled for one of the task. First, task A is being configured and start its execution. After reaching preemption point P1, the manager decides that task B has a higher priority and hence should be placed on the processing unit. Then, context from task A is saved. Task B can safely be configured on the processing unit. At the end of its execution, task A can be ran again after configuring the task and restoring the context.



Figure 14 – Execution timeline during task preemption

There is a slight difference between processor cores and reconfigurable zones: for processor cores, context switch time does not depend from the executable (processor dependent) and is performed each time a binary is loaded (not only after a preemption). For hardware execution units, we assume that context switch is performed by reading/writing data from/to registers (another possibility consists in performing a readback on the configuration data, which is very unefficient). Thus, it is not mandatory to perform this context switch when the task is being configured for a new execution. The choice is given to the designer to perform this context switch or not with the parameter *Use context switch* for hardware implementations.

More detailed timelines for each operating mode and processing unit can be found within the FoRTReSS package in text file *context\_switch.txt* or directly through menu *FoRTReSS* → *View Information Report*.

### 2. Simulation time

FoRTReSS aims at providing architecture that will always satisfy the application timing constraints. Hence, the system must be simulated long enough to make this assumption. The minimal simulation time is based on two components:

- Time to fill the application pipeline: rounded up to the sum of all tasks offsets and periods (WCET in case of non-periodic applications)
- Hyperperiod: least common multiple (LCM) of task periods (WCET for non-periodic applications)

However, it is important to note that this value is just an approximation of the minimal simulation time, which can be altered by the fact that diagrams are CDFGs and not simple DFGs. The designer should be careful on simulation time when writing the application testbench.

### 3. Transaction-Level Modelling

RecoSim makes use of Transaction-Level Modelling (TLM) to abstract communication between modules. From the designer point of view, TLM is represented by timing information on the CDFG edges. Figure 15 shows interactions between modules as a sequence diagram.

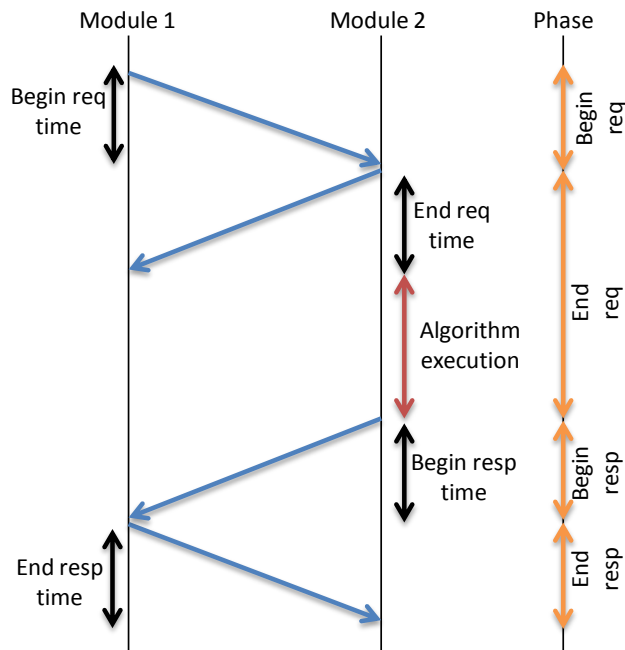


Figure 15 – Sequence diagram of a TLM transaction in RecoSim

Timing information can be seen as time required by data transfer to/from memory or directly between modules when considering point-to-point access. For instance, during request phase, *Begin req* time can refer to the time required to store the result of the algorithm into a shared memory or a FIFO. *End req* time might refer to the data retrieving phase. After this one, Module 2 is able to start the algorithm associated with the current implementation to process data. The response phase is used mostly for acknowledgements purpose, but it can still be annotated with timing information.

This figure also shows the TLM as depicted in the generated VCD trace files. Note that the END\_REQ phase also includes the algorithm execution since it lasts until the response is received.

In our example, we kept zero values everywhere since data transfers were already considered into the algorithm execution time.

## V. Advanced features

### 1. Describing FPGAs using XML

ForTRReSS comes with a limited number of devices described at this point (only the most popular reconfigurable FPGAs, typically the ones mounted on evaluation boards). However, it is possible to add other FPGAs or even describe new/future/virtual FPGAs.

ForTRReSS relies on a set of XML files for describing the devices. A sample example of such an XML file can be found on the next page. It is composed of 4 parts.

The first part consists in a brief description of the device with information such as device identifier or the number of lines and columns composing the device. Note that this description can be done once for several FPGA packages and/or speed grades.

Then, an exhaustive description of the cells used within the device is done under the **Cells** tag. Each cell is identified by its name and key parameters such as:

- **desc:** a short device description
- **cost:** Default cost for this resource type
- **nbCellsPerColumn:** number of such cells inside a column (also defined as the FPGA clock domain, the shortest height for a reconfigurable zone inside the FPGA)
- **reconfigurable:** whether the cell is reconfigurable or not
- **bitstreamSizePerColumn:** size of a bitstream that only constrains one column of this type of cell
- **defaultIncrement:** used only for generating UCF constraints (thus can be considered as used only by Xilinx devices, for resources that are reconfigurable and hence can be constrained in an UCF file), it represents the increment that is performed to identify two successive columns of the same cell type. For all we know, it does not depend on the device family and can be left as-is.
- **group:** Specify a group for this cell. Cells within the same group share the same counter for identifying them in the UCF constraint file. For instance, successive cells CLB $i$  and CLB $m$  will have ID  $n$  and  $n+1$ , even if they are not exactly the same type of cell.
- **marginSensitive:** If true, resource requirements of this type of cells will be subject to the resource margin defined for ForTRReSS step 1 (RZ determination per task). For instance, in Xilinx FPGAs, CLBs, that are routing resources, are sensitive to this resource margin whereas BRAM are not.
- **virtual:** This boolean attribute indicates whether the cell can be used for device description. Typically used for Xilinx FPGAs where Slices resources are not used for device description (as they cannot be constrained individually) but are still required for proper ForTRReSS execution since task resources can be described as Slices. If not specified, default value is false.
- **f\_red, f\_green, f\_blue:** RGB foreground color components for representing the cell in ForTRReSS toolbox.

For Xilinx devices, particular attention should be brought to describing CLBs and Slices. Even though slices cannot be constrained individually (it is only possible to constrain CLBs that are composed of



two Slices), it is necessary to describe them as well in the XML file to ensure proper behaviour of FoTReSS. For Xilinx users, we recommend using existing XML files as templates for describing new devices.

The third section, within the **Ressources** tag, is needed for accurately describing Xilinx FPGAs, since logic elements are alternatively described in terms of Slices or CLBs (*Configurable Logic Block*) depending on the context. Hence, a conversion is necessary to ensure proper behaviour. This information can be found in the *Configurable Logic Block User guide* corresponding to the device family (Virtex-6, Virtex-7...).

Finally, the last section actually describes the architecture of the FPGA within the **Description** tag. The description is made column by column. Each line within a column can be described individually using the **Lx** attribute (e.g. L1="CLB"). If the entire column is filled with a single type of cells, the **ALL** attribute is preferred (e.g. ALL="CLB"). When several successive columns share the exact same construction, the **REPEAT** attribute can be set to the number of repetitions. Also, note the use of the **X** attribute that is used to override the X-coordinate that can be inferred from the *defaultIncrement* parameter of the device (used only for proper UCF generation). For instance, we consider in this example a Virtex-6 LX240T FPGA. A look at its PlanAhead representation shows some reserved space at the centre of the device. This space is used in other lines to provide CLB resources, thus affecting the X-coordinate of the CLB cells. Another use case showing this attribute necessity is the presence of a hardwired PowerPC inside some Virtex-5 devices (such as the V5FX70T also described by the authors).

For Xilinx 7-series FPGAs and SoCs, it is impossible to constrain only one CLB column (up to this date, Xilinx tools in version 14.4) due to some interconnection restriction (see Xilinx Answer Record AR#53290 <http://www.xilinx.com/support/answers/53290.htm>). Some CLB columns can only be constrained at the beginning of a reconfigurable region (e.g. CLB\_L\_L with Xilinx naming conventions, meaning a CLBL that is placed on the left) or at the end of a region (e.g. CLB\_L\_R placed to the right). Therefore, we introduced a new keyword, **positionConstraint**, with three possible values: BEGIN, END and BOTH. By default, if this attribute is omitted, it is defaulting to BOTH (the column can either begin or end the region like for Virtex-5 and Virtex-6 devices, for instance).

#### */!\ Reserved keywords*

Because of the extra attention required by Xilinx FPGA (for handling both CLBs and Slices), some words have an important meaning in the context of describing devices. Thus, the words **CLB**, **CLBI**, **CLBm**, **SliceL**, **SliceM** and **Slice** are reserved to Xilinx components (unless a similar behaviour is expected).

```
<?xml version="1.0" encoding="UTF-8"?>
<Device>
  <DeviceName value="xc6vlx240t" />
  <Society value="Xilinx" />
  <Category value="Virtex6" />
  <LineNumber value="6" />
  <ColumnNumber value="102" />
  <DevicePackage>
    <DP value="ff1156" />
  </DevicePackage>
  <SpeedGrade>
    <SG value="-1" />
    <SG value="-2" />
  </SpeedGrade>

  <Cells>
    <Cell name="CLB1" desc="CLB1" cost="1" nbCellsPerColumn="40"
reconfigurable="true" bitstreamSizePerColumn="11808" defaultIncrement="2"
group="CLB" marginSensitive="true" f_red="24" f_green="116" f_blue="205" />
    <Cell name="Slice" virtual="true" cost="1" nbCellsPerColumn="40"
reconfigurable="true" bitstreamSizePerColumn="5904" defaultIncrement="2"
group="CLB" marginSensitive="true" />
    ...
  </Cells>

  <Ressources>
    <NB_LUT_SLICE value="4" />
    <NB_FF_SLICE value="8" />
  </Ressources>

  <Description>
    <CN ALL="IOB" />
    <CN ALL="CLBm" />
    <CN ALL="CLB1" />
    <CN ALL="BRAM" />
    <CN REPEAT="2" ALL="CLBm" />
    <CN ALL="DSP" />
    <CN REPEAT="6" L0 = "CLB1" L1 = "CLB1" L2 = "CLB_XXX" L3 = "CLB_XXX" L4 =
"CLB1" L5 = "CLB1" />
    <CN ALL="CLBm" X="84" />
    ...
  </Description>
</Device>
```

## 2. Using an XML description of the task resources

Task resource needs can be described by Xilinx synthesis reports (.syr files) or by generic XML files (.tsk extension). Here is a sample XML description.

```
<?xml version="1.0" encoding="UTF-8"?>
<Task>
  <DeviceName value="xc6vlx240t" />
  <DevicePackage>
    <DP value="ff1156" />
  </DevicePackage>
  <SpeedGrade>
    <SG value="-1" />
    <SG value="-2" />
    <SG value="-3" />
  </SpeedGrade>

  <Cell name="Slice" value="2200" />
  <Cell name="BRAM" value="16" />
  <Cell name="DSP" value="10" />
  <Cell name="UserCell0" value="12" />
  <Cell name="UserCell1" value="4" />
</Task>
```

A task is described within the **Task** tag. Tag **DeviceName** identifies the device for which this description is accurate. A single description can be the same for several packages and speed grades, as specified within the **DevicePackage** and the **SpeedGrade** tags respectively. Finally, required resources are described in the **Cell** tags, composed by the cell name and its value, i.e. the number of resources required by the task. This description should be done together with the device description in order for the cells to match names from the device description. For instance, the sample XML description defines a task which needs 12 cells from type *UserCell0* and 4 cells from type *UserCell1*. These cell types must be defined in the FPGA XML description otherwise FoTRReSS compilation will fail.

In order to use XML files rather than synthesis reports, simply import or link the .tsk file instead of the .syr file in the *Task report/netlist* tab of FoTRReSS preferences.

### 3. Using an XML description of the RZ set

FoTRReSS is able to infer a set of reconfigurable zones from the application tasks description. However, it is also possible to bypass this step by providing an XML file representing available reconfigurable zones with .rrd extension (*Reconfigurable Region Definition*).

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<RZSet>

  <DeviceName value="xc6v1x240t"/>
  <DevicePackage>
    <DP value="ff1156"/>
  </DevicePackage>
  <SpeedGrade>
    <SG value="-1"/>
  </SpeedGrade>

  <Description Column="70" Height="2" Line="3" RZ_ID="318101" Reverse="false">
    <CN L0="UNUSED" L1="USED"/>
    <CN ALL="USED" REPEAT="10"/>
  </Description>

  <Description Column="59" Height="2" Line="3" RZ_ID="318323" Reverse="false">
    <CN ALL="USED" REPEAT="6"/>
    <CN L0="USED" L1="STATIC"/>
    <CN ALL="USED" REPEAT="4"/>
  </Description>

</RZSet>
```

The first part of the XML description is composed of tags already described such representing the device package and speed grade. The description part contains the following attributes:

- **Column:** number of the first column composing the RZ
- **Line:** number of the first line composing the RZ
- **Height:** reconfigurable zone height (in numbers of clock domains)
- **RZ\_ID:** RZ identifier that can be either a string or a integer but must be unique
- **Reverse:** boolean to provide the possibility to describe columns from right to left rather than left to right

The description of the reconfigurable zone is quite similar to the device description. However, instead of defining the type of the cells, we prefer to define if their PR status, which can be one of the following (please refer to the publication for more details about RZ shapes and resource usage):

- **USED:** the cell is used by the RZ
- **UNUSED:** the cell is not used by the RZ
- **NOT\_RECONFIGURABLE:** the cell is not reconfigurable, hence unused by the RZ
- **STATIC:** unused resources that cannot be used by another reconfigurable zones

Creation and import of RRD files can be done through the *Customization* → *User RZ description* menu. Once a file has been created or imported, the parameter *Use RZ set* can be set to true in the FoTRReSS parameters tab. Then, the RRD file can be selected.

Note that XML descriptions of the chosen reconfigurable zones are also generated by FoTRReSS as an alternative to UCF files that may not be very adequate for non-Xilinx FPGAs or custom devices.

## 4. Diagram features

In this tutorial, we used only one testbench but it is possible to split it in two halves, one responsible for sending the data and the other one for receiving it. In addition, it is also possible to have multiple modules connected to a single module. For instance, diagrams like that are possible:

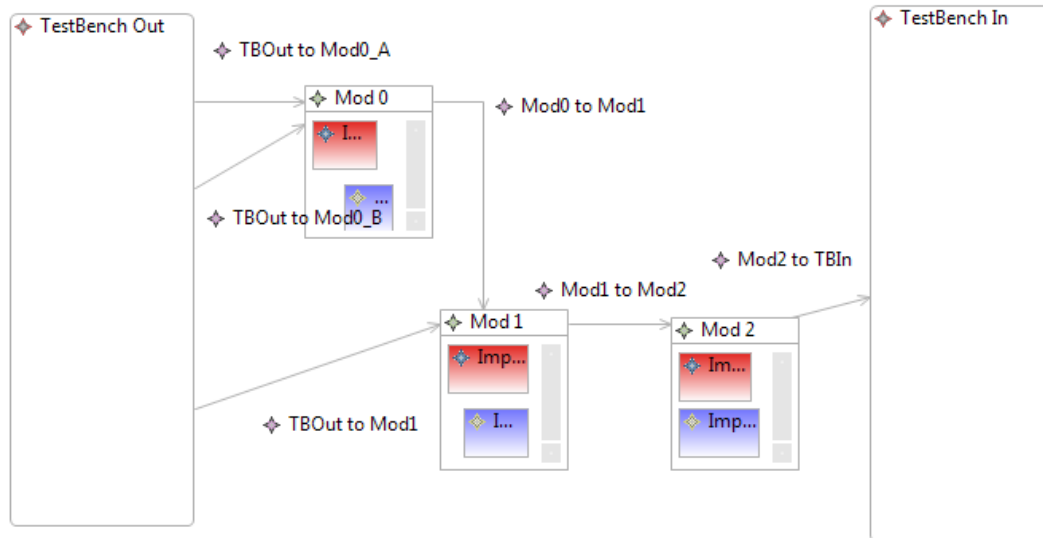


Figure 16 - Custom application diagram

**/!\ Important note:** there should be one and only one testbench responsible for sending data to the application as well as one testbench for receiving data.

As we can see, it is possible to provide multiple implementations for each module. In the previous capture, each task has both software and hardware implementation. More generally, it is possible to define any implementations for a task. Files for this application are provided with the tutorial in the *applications* directory, named *custom\_application*.

It is also possible to define cyclic diagrams. In the following capture, we can see that the link between modules Mod2 and Mod0 forms a cycle in the diagram. Hence, this connection has to be defined as transient, which means that Mod0 does not need to wait for data incoming from Mod2 on its first execution. This application is called *cyclic\_app* in the *applications* directory.



Figure 17 - Cyclic application diagram

## 5. Interfaces description and APIs

In order to modify components behaviour, FoRTReSS provides a set of APIs to help the designer with his task. The following figure shows interactions between components and how the designer may affect them.

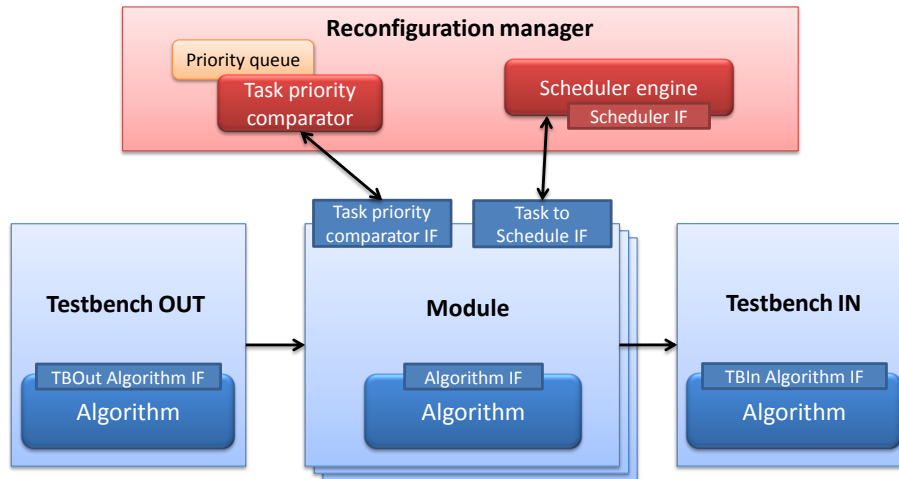


Figure 18 – APIs and interfaces inside RecoSim

### Task execution simulation

Up until now, we have not provided any execution scheme for the different implementations added to the design. Default behaviour consists in simulating the task execution using a single *wait* instruction. This behaviour can be sufficient for early estimations but may lack of precision. For this purpose, it is possible to define a custom behaviour for an implementation.

First, be sure that the text editor field of FoRTReSS system parameters has been set. In the diagram view, right-click on an implementation and choose *Algorithm* → *New* to create a new algorithm. Algorithms are defined within header files that should be located inside the project *algorithms* directory. Choose a file name, for instance *my\_custom\_algorithm*. The file is created and opened in your favourite text editor. The file provides a template for the definition of task algorithms and provides default behaviour. Default behaviour can now be replaced by SystemC code, authorized to access some module members through an API (documentation available in *help* → *FoRTReSS API* → *Algorithm* menu).

We recommend understanding the code beneath default behaviour before developing your own algorithm. It can be found within the *simulation/RecoSim/src* directory, file *module\_algorithms.h*

### Changing testbench behaviour

It is possible to change testbench behaviour as well, the same way it is done for implementation algorithms. In the diagram view, right-click on the testbench and select *Testbench Thread* → *New*. This will create a new file where testbench behaviour can be specified, using a set of dedicated APIs, depending of the testbench type. There are three types of testbench:

- **IN:** Testbench designed for receiving data only (API documentation file accessible through *Help* → *FoRTReSS API* → *Testbench In* menu)
- **OUT:** Testbench designed for sending data only (API documentation file accessible through *Help* → *FoRTReSS API* → *Testbench Out* menu)

- **INOUT:** Testbench designed for both sending and receiving data (API documentation file accessible through *Help* → *FoTRReSS API* → *Testbench In/out* menu)

Reference files for the implemented testbench algorithms can be found in file *RecoSim/src/testbench\_algorithms.h*.

### Modifying scheduling algorithm

At this point, we only provide the user with basic scheduling algorithms (EDF scheduling coupled with AMAP mapping). Since this strategy may not fit every design, it is possible to develop custom scheduling schemes.

Our scheduler is based on two blocks:

- A waiting queue with customizable priority policy: when the reconfiguration inserts a new task into the waiting queue, it is automatically sorted using a comparator. The comparator can access several Task class members (for instance, it is possible to retrieve a task deadline to build an Earliest-Deadline-First strategy (EDF, already implemented)). Custom strategies can be added to the FoTRReSS project by selecting *TaskPriorityComparatorFile* under menu *Customization*. Comparison is performed on pointers to *Task\_priority\_comparator\_interface* objects that allow the reconfiguration manager to access some module parameters. Functions provided by this interface are listed in the file *class\_manager\_\_interface.html* under the *Help* → *FoTRReSS API* → *Scheduler* → *Task Priority Comparator* menu.
- An API to modify scheduler core behaviour. The API documentation can be found under the *Help* → *FoTRReSS API* → *Scheduler* → *Scheduler* menu. To create a new file containing the scheduling algorithm, choose *SchedulingStrategyFile* under menu *Customization*. Some objects manipulation within the scheduler require the use of the *Task\_to\_schedule\_interface* API, designed for accessing functions helpful for scheduling tasks. Documentation can be found under the *Help* → *FoTRReSS API* → *Scheduler* → *Task to schedule* menu

The scheduler also relies on a set of task states for modelling purposes. Here is the state diagram with the relation between tasks.

- **Waiting:** The task has not been allocated to a reconfigurable zone
- **Queued:** The task is in the reconfiguration unit waiting queue
- **Configuration:** Task is being reconfigured (bitstream loading for hardware execution units, binary loading for software execution units).
- **Running:** Task is executed on the reconfigurable zone
- **Mapped:** Task has finished its execution and is still configured on the reconfigurable zone

Reference files for the implemented scheduler can be found in file *RecoSim/src/scheduling.cpp*.

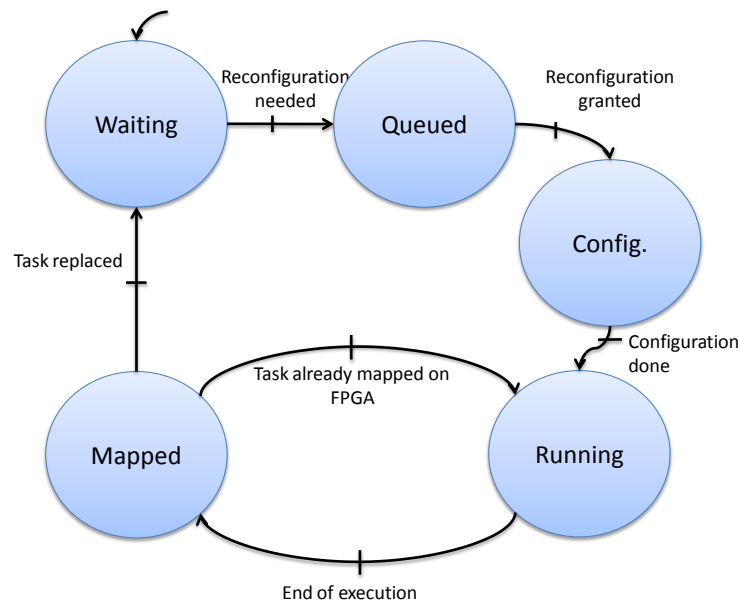


Figure 19 – Task state diagram

### Changing the algorithm execution mode

We mentioned earlier that it is possible to code module algorithms using a dedicated API. Simulating C++ code (or even VHDL code, through an adaptor) is slower than just emulating execution time with a single SystemC *wait* statement. FoTReSS provides the user with the possibility to change an algorithm execution mode at runtime. Therefore, it is possible to start a simulation with execution time simulation and switch to SystemC code for important/problematic sections.

This is done through the testbench by a call to the *TB\_IF\_add\_execution\_mode\_command\_to\_queue* function. The execution mode is represented as a string for more flexibility. The function also takes as parameters the targeted module (string) and the delay before applying the change (*sc\_time*). Note that it is the responsibility of the designer to handle different execution modes within the module algorithm.

## 6. Static & Interface constraints

### Interfaces behaviour

Interfaces described within FoTReSS are used in two ways: first, it restrains reconfigurable zones placement on the FPGA if similar interfaces constraints have been defined for the FPGA (as we will see in next section). It is also used by RecoSim to simulate their functional behaviour, introducing the concept of physical and virtual channels. A virtual channel represents the binding between two modules, i.e. edges of the CDFG. A physical channel is an interface that is provided by a task implementation. Hence, it is possible to define a smaller number of physical interfaces compared to virtual interfaces. In this case, virtual channels will have to share physical channels, thus reducing the overall resources utilization (by reducing the number of interfaces) but also reducing performance.

In this version of FoTReSS, we consider interfaces as a result of the flow: interfaces that should be connected to the reconfigurable zone are the aggregate of the interfaces from tasks mapped to this zone. However, this can seem quite inefficient for applications with very different interfaces. In further versions of the flow, FoTReSS would be able to optimize interfaces on reconfigurable zones.



Interfaces might be defined for processors as well to restrain task placement on these units.

### Constraining interfaces

It is possible to prohibit some columns from being used in a reconfigurable zone. In other words, it is possible to define a reconfigurable region and a static region. This can be done by editing FoTReSS parameters in the *FoTReSS preferences* window. Click the *Static & Interface constraints...* button. A new window opens to select which columns or zones of the FPGA must remain static on a graphical FPGA representation.

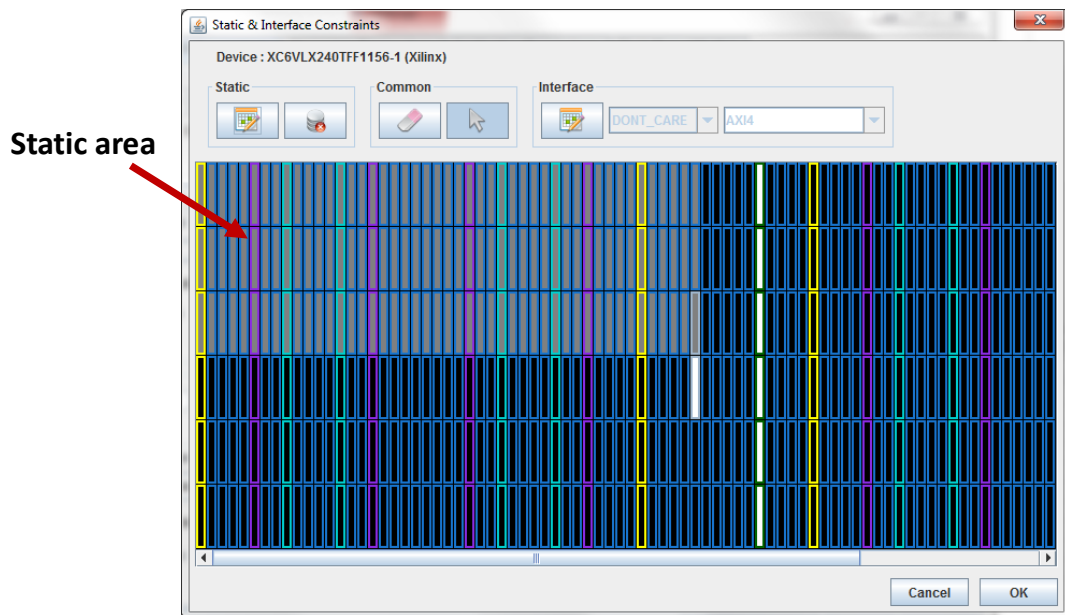


Figure 20 - Adding static constraints

In this new window, the first button on the left allows defining static areas while the second button deletes all static cells defined in the design. This results in the following floorplan (respectively for one and two applications):

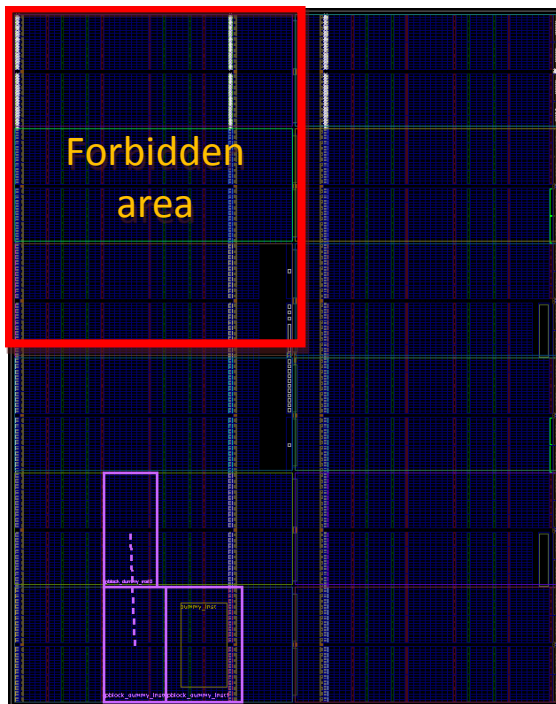


Figure 21 - Floorplan for one application

This is described within the configuration *Config\_1\_static\_area* in the project provided with the tutorial.

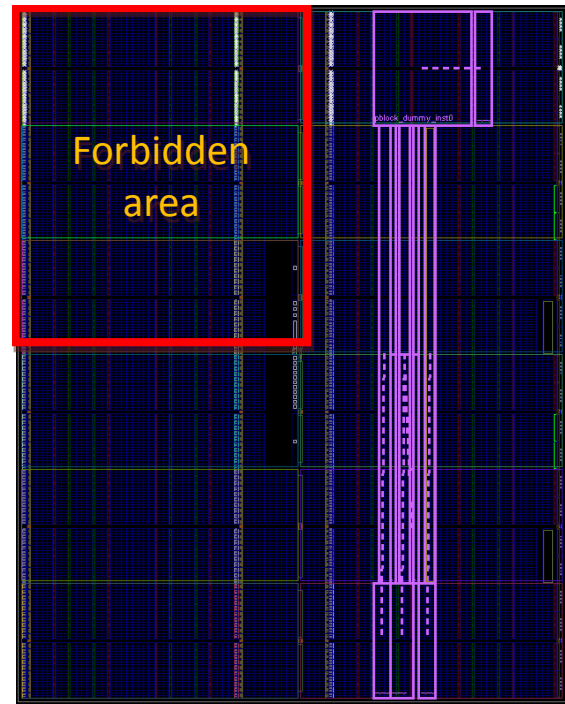


Figure 22 - Floorplan for two applications

In the same preferences window, the user can place interfaces on the FPGA. These interfaces are used by FoTRReSS to restrain placement of tasks on reconfigurable zones that provide the interfaces necessary for its execution. For instance, it can be used to place reconfigurable zones as close as possible to an interconnection. In order to use this functionality, we first need to specify interfaces required by the task. For all hardware implementations in the application, set the parameter *Interface Types* to AXI4, for instance. It is also necessary to specify the same interface for each connection (both *Input* and *Output communication type*). Go to the *Static & Interface constraints* window and insert a pool of AXI interfaces right under the static area previously defined. To do this, select the cells and click the *Add interface constraints* button. Select AXI4 as the interface type. The interface position inside the reconfigurable zone can be restricted as well following a compass point style. For example, interfaces tagged as *NORTHEAST* should be positioned in the upper right corner of the reconfigurable zone. In our case, we keep the value to *DONT\_CARE*. Figure 23 **Erreur ! Source du renvoi introuvable.** shows the constraints window after placing the AXI4 interfaces constraints. FoTRReSS gives a different solution from what we had before constraining interfaces, which is understandable considering that we had serious constraints which complicate RZ search. Indeed, it adds another zone to the simulation set in order to find a correct solution. When considering a single application, area results are still in favour of PR but with two applications running in parallel, we notice a BRAM overhead that might advantage static solutions.

This is described respectively within configurations *Config\_1\_interfaces* and *Config\_1\_interfaces\_2apps* in the project provided with the tutorial.

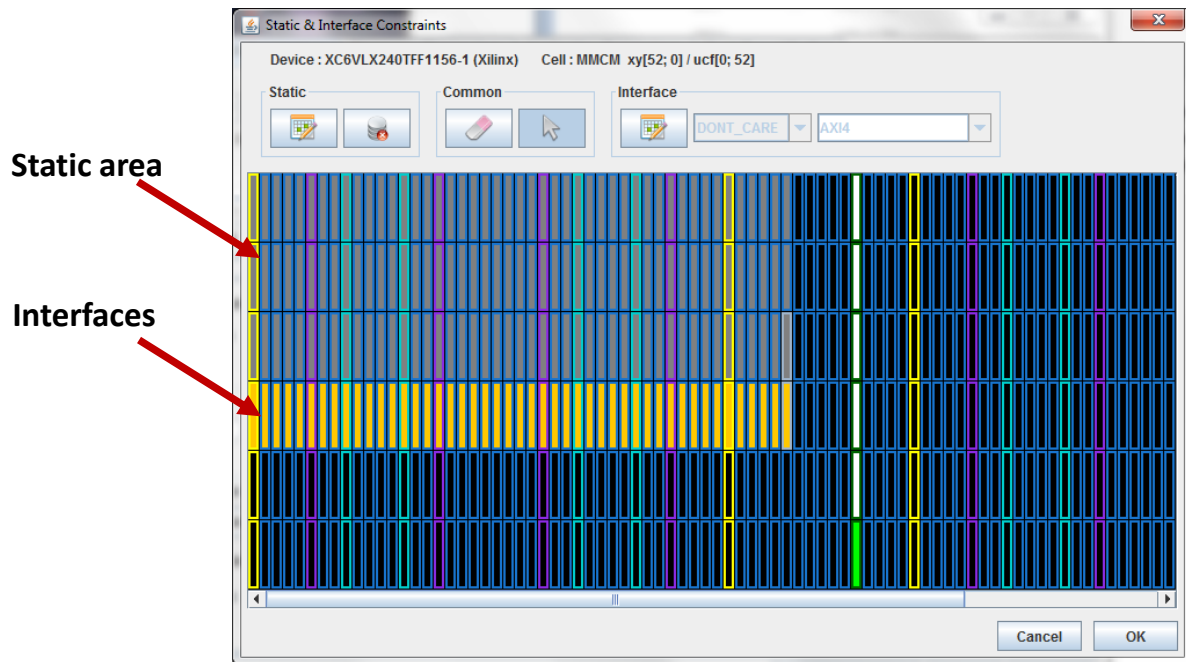


Figure 23 - Adding interface constraints

In this example, we only defined one AXI4 interface. It is possible to define more than one interface by using symbol '&'. For instance, a task implementation providing AXI4-Master, AXI4-Slave and FIFO interfaces would be parameterized with the expression: *AXI4-Master&AXI4-Slave&FIFO*.

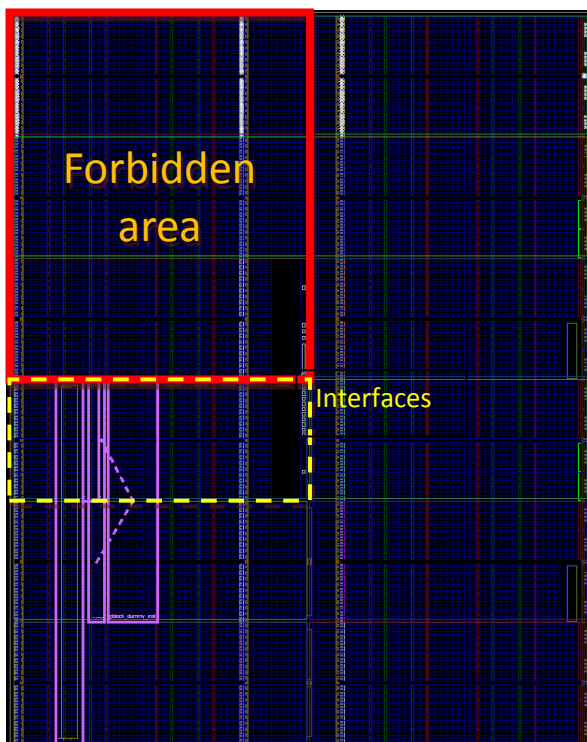


Figure 24 - Floorplan for one application with interface constraints

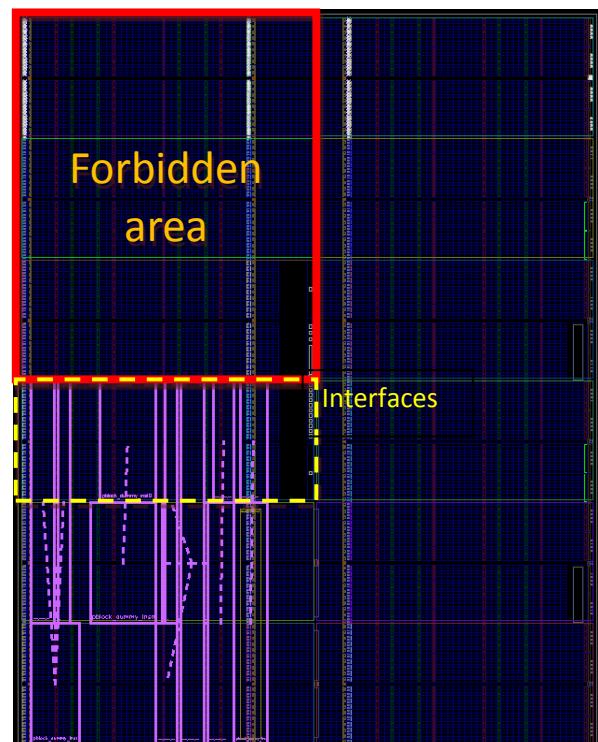
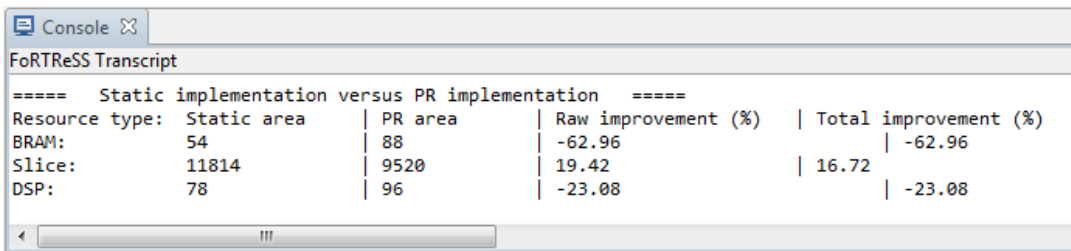


Figure 25 - Floorplan for two applications with interface constraints

## 7. Design Space Exploration of the paper

ForTReSS provides an easy way of managing applications that should be considered for simulation. It is done under the *Preferences* menu, *Application parameters* tab. As a result of the tutorial, there should be one application listed. The *Add* button adds application to the simulation list. The *Duplicate* button duplicates the selected application in the list. As stated in the paper provided with the tool, we went up to three applications in parallel. This case showed really poor results with default settings. In this section, we will show how it is possible to modify simulation parameters for Design Space Exploration purposes. Note that the results showed here might not be exactly the same as those in the paper since ForTReSS evolved a lot since the paper submission.



Console

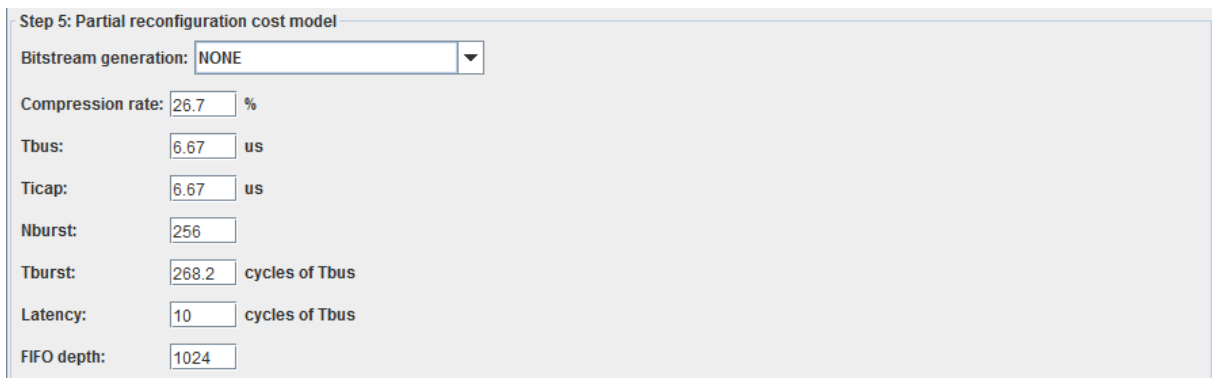
ForTReSS Transcript

```
===== Static implementation versus PR implementation =====
```

Resource type:	Static area	PR area	Raw improvement (%)	Total improvement (%)
BRAM:	54	88	-62.96	-62.96
Slice:	11814	9520	19.42	16.72
DSP:	78	96	-23.08	-23.08

Figure 26 - Area results for three applications

The paper shows that in order to improve the overall system performance, we need to speed up the configuration subsystem and to raise reconfiguration granularity (i.e. reconfiguring tasks after  $n$  executions rather than just one). The first improvement can easily be done via the *ForTReSS parameters* tab of the *Preferences* menu. The following figure shows the parameters used to clock the configuration subsystem at 150 MHz with 256 beats per burst access.



Step 5: Partial reconfiguration cost model

Bitstream generation: NONE

Compression rate: 26.7 %

Tbus: 6.67 us

Ticap: 6.67 us

Nburst: 256

Tburst: 268.2 cycles of Tbus

Latency: 10 cycles of Tbus

FIFO depth: 1024

Figure 27 - PR cost model parameters

Concerning the reconfiguration granularity, there is actually no way to properly prevent reconfiguration until a certain number of task execution is reached. A workaround is to modify timing information in order to simulate granularity modification. Indeed, reconfiguring only after two task executions can be seen as processing two packets at once, thus doubling execution times, deadlines, and periodicity.

This can be done in two ways. The first one (simple, but not scalable), consists in manually changing implementation execution times, task deadlines and testbench periodicity. The second (and preferred) one, consists in defining a constant that can be used inside the user interface and transmitted to ForTReSS via dedicated header and implementation files. First, create a header file:

under menu *Customization*, click *User Parameter Files* → *User Parameter Header File* (a text editor must be previously registered on FoTRReSS parameters). The file is then opened in your favourite text editor. Let us define a constant representing the number of task executions before reconfiguration. For instance: `#define NB_TASK_EXECUTIONS 2`. Then, change the timing information to accommodate this variable. For instance, set MPEG-2 encoder execution to `NB_TASK_EXECUTIONS*5300`. Once timing information has been changed, re-generate, compile and launch simulation. Now, area results should be in favour of a partially reconfigurable solution.

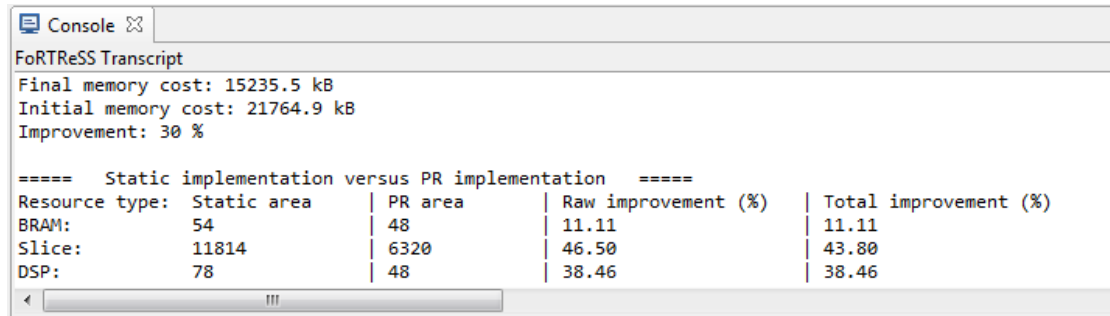


Figure 28 - Area results after design space exploration

## 8. Debugging a simulation (Windows only)

If, for any reason, an in-depth analysis of the execution code is required, it is possible to open the project in Visual Studio and switch to Debug configuration. This can be done via GUI menu *FoTRReSS* → *Run Visual C++*.

## VI. References

- [1] F. Duhem, F. Muller, W. Aubry, B. Le Gal, D. Négru et P. Lorenzini, «Design Space Exploration for Partially Reconfigurable Architectures in Real-Time Systems,» *Journal of Systems Architecture*, 2013.
- [2] Accelera Systems Initiative, IEEE Standard for Standard SystemC Language Reference Manual, 2011.
- [3] F. Duhem, F. Muller et P. Lorenzini, «Reconfiguration time overhead on field programmable gate arrays: reduction and cost model,» *IET Computers & Digital Techniques*, 2011.

## Change log

Version	Date	Description
0.9	2012-10-01	Initial version
1.0a	2013-01-31	Update to follow FoTReSS development, first stable release
1.0b	2013-02-01	Minor corrections Added details for scheduling modification, task state diagram
1.0c	2013-02-14	Minor corrections Pre-requisites update Added interfaces diagram Update to follow FoTReSS GUI
1.0d	2013-03-12	Added distinction between configuration and context switch Added SW implementation description XML task description clarified Add diagram explaining task switch and reconfiguration Pre-requisites update
1.0e	2013-03-25	Minor fixes Added interface expression description Added sequence diagram for module interactions with TLM Modified preemption diagram and explanations
1.0f	2013-04-05	Minor fixes Added positionConstraint definition for XML device description (7 series) Modification of the TLM transaction figure to show phases
1.0g	2013-05-07	Minor fixes
1.0h	2013-05-14	Minor fixes
1.0i	2013-05-15	Minor fixes, adding Visual C++ 2010
1.0j	2013-05-25	Minor fixes, deleting TLM path (only use SystemC 2.3)