# Optimize CASPER Development by "Black Boxing" Designs

David MacMahon

November 23, 2010

**Abstract**

The CASPER toolflow uses Xilinx System Generator as its top level design entry tool. This document describes how to use Xilinx System Generator's Black Box block to develop System Generator models (including CASPER based designs) more efficiently using versions 10.1 and later of the toolflow.

## 1 Introduction

Astronomy applications developed using the CASPER toolflow often contain multiple instances of the same heavyweight subsystem (e.g. a PFB). Generation and synthesis of these designs can take a long time since each instance of these heavyweight subsystems is generated and synthesized separately even though they are identical copies of each other. This document describes how to use System Generator's Black Box block to eliminate such unnecessarily repetitive steps, thereby greatly reducing development time and providing several other benefits that aid in collaborative development. The technique is an expansion upon an idea put forth by Chen Chang, the architect and designer of the BEE2 board. It uses two features of System Generator: the Black Box block and "compile to NGC netlist".

# 2 Benefits

Some of the benefits of using Black Box blocks include:

- Reduces development cycle time
  - Makes "Update diagram" much faster in Simulink
  - Makes generation much faster in System Generator
- Modularizes designs
  - Easier to work in parallel on large designs
  - Limits impact of design changes
  - Makes designs more amenable to version control systems
- Reduces potential for design entry errors

# 3 Procedure

A basic overview of the process is provided here. Full details of each step are given below. Repeat these steps for each heavyweight block type you want to use. Even if your model uses identical copies of a heavyweight component, you only need to generate and synthesize the netlist once. This is one way in which black boxing speeds up the development cycle.

1. Create a netlist for the heavyweight block
2. Extract the VHDL entity declaration from generated files
3. Use Black Box blocks to include the netlist in the top level model
4. Add one CASPER "pcore" block for the netlist

## 3.1 Create a netlist for the repeated heavyweight block

Creating a netlist is as simple as creating a model and invoking System Generator with certain options configured. These steps are detailed here.

### 3.1.1 Create a model that describes your component

To create the netlist for your heavyweight block, you can cut-and-paste from an existing design into a new model file or you can create the model from scratch. At a minimum your design must contain the following items:

- A System Generator block
- One instance of the heavyweight block (e.g. PFB) that you want to replicate in your design. Do not use CASPER "yellow blocks" in this model. If needed, you may instead use inputs and outputs to which yellow blocks can be connected in the higher level design files.
- Gateway In/Out blocks for the heavyweight block's inputs/outputs. Note that the Black Box port names are derived from these Gateway block names. Names that are VHDL reserved words will get a suffix appended to them automatically to avoid problems, but this may not be desirable.
- Simulink Sources/Sinks on the extremities. You can use simple constants and terminators (e.g. from `xlAddTerms`), but preferably you would use these to develop a test bench for your component.

Figure 1 shows an example System Generator model that contains a heavyweight FFT block that is being "black boxed". The name of the last output port of the green FFT block, `of`, would conflict with a VHDL reserved word. The Gateway Out for this port has been named `oflow` to avoid an unsightly suffix being added when generating the netlist.

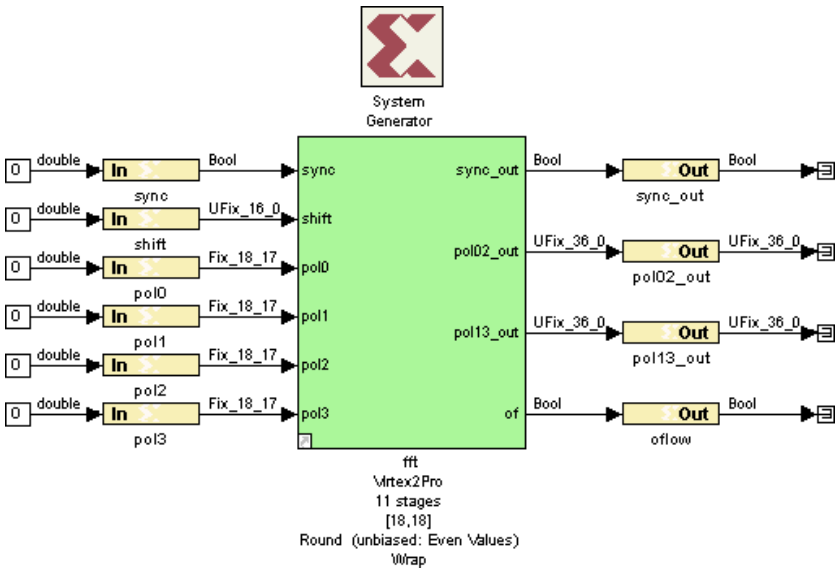


Figure 1: A System Generator model suitable for black boxing

### 3.1.2 Configure System Generator

Configure the System Generator block as described here.

- Select the "NGC netlist" compilation target
- Select the "Part" (i.e. FPGA type) you are targeting. This will ensure that the generated netlist is compatible with and optimized for the FPGA you will be using. Table 1 provides details about the FPGAs used on various CASPER boards.
- Click the "Settings" button and turn off the "Include Clock Wrapper" option
- Specify the target (i.e. output) directory. A recommended practice is to use an output directory of "`./`*name*`_core`" (e.g. `./fft_core`). For some reason, the leading "`./`" seems to be required.

Figure 2 highlights the System Generator settings required for generating an NGC netlist suitable for black boxing.

| Board | Family | Part | Speed | Package |
|---|---|---|---|---|
| ROACH | Virtex5 | XC5VSX95T | -1 | FF1136 |
| | Virtex5 | XC5VLX110T | -1 | FF1136 |
| BEE2 | Virtex2Pro | XCV2P70 | -7 | FF1704 |
| IBOB | Virtex2Pro | XCV2P50 | -7 | FF1152 |

Table 1: FPGA Details for CASPER Boards

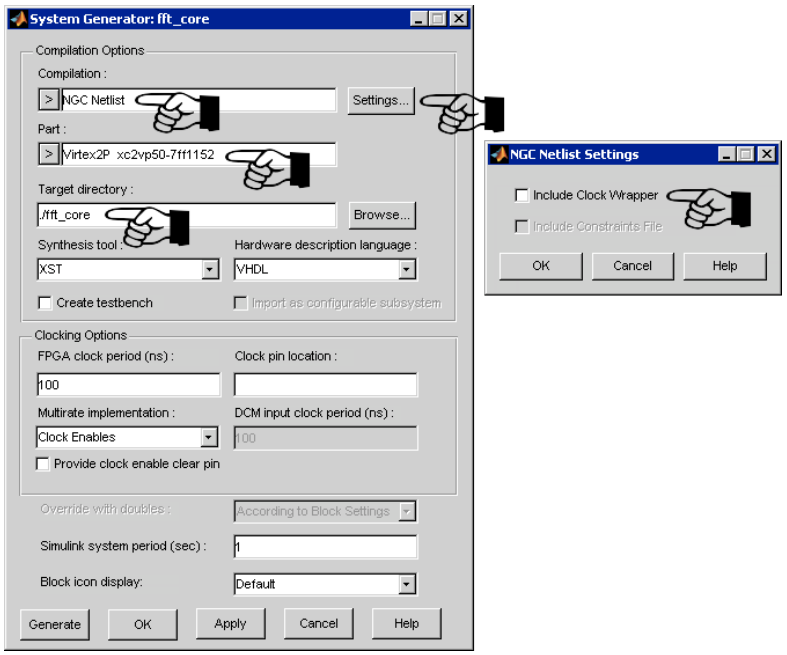


Figure 2: System Generator settings

### 3.1.3 Save the model using a descriptive name

Save the model using a descriptive name. A recommended practice is to save the model as "*name*_core.mdl" (e.g. fft_core.mdl).

### 3.1.4 Generate to a netlist

Click System Generator's "Generate" button to create the NGC netlist file (e.g. ./fft_core/fft_core.ngc). This will generate VHDL for the model and then synthesize the VHDL to an NGC netlist file using XST (Xilinx Synthesis Tool).

## 3.2 Extract VHDL Entity from Generated Files

To use the synthesized netlist with the Black Box block, you need to create a *core*_config.m file that describes the netlist in terms System Generator understands. The easiest way to do this is to extract the VHDL entity declaration from the generated VHDL (from which the netlist was synthesized) and let the Black Box interface generate the *core*_config.m file for you.

A Matlab function, extract_entity, has been created to facilitate the extraction of the entity declaration. In the Matlab command window, run extract_entity('*netlist_name*'), for example:

```
extract_entity('fft_core/fft_core.ngc');
```

This will create a .vhd file in the current directory with the same basename as the netlist file.

The Black Box interface uses this extracted entity declaration to generate the *core*_config.m file the first time the netlist is used in another design. This process is described below.

## 3.3 Use Black Box blocks in the top level model

When placing a Black Box block in a model, a file selection dialog will pop up asking for the VHDL entity describing the netlist so the tool can generate a *core*_config.m file for you. This is the easiest way to generate the

*core*_config.m file. The first time a netlist is used with a black box, select the name of the VHDL file generated by extract_entity. Note that your model file must be saved (i.e. it cannot be untitled) for this to work. The generated *core*_config.m file will open in the Matlab editor so you can modify it as desired. Some of the modifications you might want to perform are:

- Comment out (or delete) the tagAsCombinational line unless your core has a combinational path from input to output. It is unlikely that a subsystem being black boxed would have such a path.
- Change output signal type from UFix_1_0 to Bool. These two Simulink types both map to the same VHDL type. If you want/need the signals to be Bool, you must change them yourself.
- Reorder the addSimulinkInport and addSimulinkOutport lines since their order here will be the port order shown in the block itself. This can be especially important if you want the black box to be used as a drop-in replacement in an existing design that already expects a certain port order.
- Other things suggested by the comments in the *core*_config.m file as appropriate, but do **not** add the NGC netlist as an "additional source file".

Subsequent instantiations (i.e. uses) of the same netlist in other Black Box blocks just need to be told the name of the *core*_config function, so you can simply cancel the dialog box and enter the function name of the function (e.g. *name*_core_config) in the Black Box dialog.

## 3.4 Add one CASPER "pcore" block for the netlist

To make sure that the CASPER toolflow knows to copy the pre-synthesized netlist file into the appropriate build directories, you need to add a pcore block (from the BEE2 library) to your design. You only need one pcore block for the netlist regardless of how many time it is used in your design. In the pcore block's dialog, enter the name of the .ngc netlist file (e.g. ./fft_core/fft_core.ngc).

# 4 Simulation

Simulation of models containing Black Box blocks is possible, but not so convenient.[1] Despite this, black boxing can facilitate simulation and testing of the black boxed subsystems by treating the netlist-generating models as test benches. Additionally, the development cycle speedup provided by black boxing makes in-hardware testing more practical.

Here are some simulation techniques that can be useful with black boxed designs:

- If you want non-zero simulation data flowing out of a black box to facilitate testing/debugging of downstream logic, consider using the "Simulation Multiplexer" block with a "From Workspace" block.
- Using "To Workspace" blocks on the outputs of one netlist-generating model and "From Workspace" blocks on the inputs of another netlist-generating models allows the simulation output of one model to be the simulation input of another model.

# 5 Comments

This technique is most helpful if you use several of the same type(s) of block(s), but it can also be helpful even if you only have one instance of a heavyweight block because it makes it faster to make changes to the ancillary logic surrounding the heavyweight block(s), which is where most of the changes occur during the build/test/debug cycle.

The `extract_entity` function is a relatively new addition to `mlib_devel`. It was introduced in Git commit eab23d2. If your installed `mlib_devel` does not have that function, you will need to update it. For details, see http://casper.berkeley.edu/git.

This document is based on version 10.1.3.1386 of System Generator. Later versions may be slightly different, but the same basic concepts should apply.

[1] The same can be said of the large, complex models which would benefit most from black boxing.