User-friendly FPGA Design with an Improved Cadence Opus - Xilinx XACT Interface

Andreas Koch

Technical University Braunschweig
Abteilung Entwurf integrierter Schaltungen (E.I.S.)
Gaußstr. 11, D-38106 Braunschweig, Germany
e-mail: koch@eis.cs.tu-bs.de

Abstract

This paper reports on various means to better integrate the Xilinx kit for FPGA design with the Cadence Opus framework. Weaknesses of the original implementation are exposed and methods to remedy them are suggested. The result is a system which presents similar user interfaces and standardizes procedures both for standard-cell based (e.g.. ES2) as well as FPGA designs. This simplifies the creation of exercises and tutorial materials for both environments

1 Introduction

When version 9301 of Cadence Opus with support for Xilinx FPGAs was delivered, we were looking forward to a marked increase in productivity and easier to use student labs, compared to our previous PC-based FPGA CAD system. We were aiming at the integration of the FPGA labs with the traditional semi-custom design labs using Opus. However, our expectations were not met, since the different subsystems failed to co-operate smoothly:

• The simulator, as delivered, was incompatible with the new licensing scheme and was lacking crucial features available in the previous version.

• The Xilinx design kit presented a user interface completely different from the standard Opus GUI, making the creation of comprehensive tutorials for students exposed to both semi-custom and FPGA-based design more difficult. Furthermore, the new interface was non-intuitive and error-prone, rendering it unsuitable for novice users.

• Features which had become cornerstones both for education and research applications, such as STL test pattern generation and combined simulation of schematic and functional views, were unavailable with the FPGA design kit.

Initial experiments with the most recent release of Cadence, 9401, suggest that the situation has not changed appreciably.

Thus, the following recount of our attempt to improve the integration of the FPGA design kit with the Opus framework 9301 might be helpful for others dissatisfied with the current state of the system.

2 Setting-up Verilog-XL

Verilog simulation initially did not work at all, since the 9301 version of Verilog already used the FlexLM licensing scheme, but was still called from the interface with the older passcode-file options. This was easily corrected by removing the option from all command invocations.

Furthermore, the Verilog provided on the 9301 CD did not contain the Simulation History Manager (SHM), used for communicating with other CAD tools (like waveform display and schematic monitors). In order to build a new version of Verilog with SHM, it was necessary to employ the SHM object file from the previous Verilog version, since an update was not provided on the 9301 CD, either. Despite these version disparities, the simulator works fine with the SHM features added on.

3 Revising the User Interface

The user interface presented during the Opus-Xilinx operations consisted of `xterm` windows popping up and prompting the user for manual command input. This was non-intuitive and error-prone, rendering it unsuitable for novice users.

Instead of running non-interactive commands externally in `xterm` windows, we now run the required commands from inside Opus as a `hiBatchProcess`, showing their output in Opus `view` windows. Apart from the better visual integration, this has the advantage that the user can now use menu commands to manipulate the window and its contents. For example, the user can use the mouse and menu-commands to read a log file reporting on the results of the step just executed.

For interactive operations, like the functional or timing simulation of a circuit using Verilog, we run the program using `hiBeginProcess` encapsulated in an Opus Window created with `hiEncap`. User commands entered in the window are transmitted to the simulator using `hiWriteChild`, output from the program is displayed using `hiSetEncapHistory`.

4 Unlocking the Power of ISE

While the preceding modifications lead to a functional and usable environment, simulation using the Xilinx design kit was still markedly different from other design kits (e.g.. ES2). Even the revised simulation procedure did not make use of the capabilities provided by the Cadence Interactive Simulation Environment (ISE) such as STL, simulation history data, simulation display using cWaves or schematic monitors and a fully encapsulated Verilog with complete menu controls. Furthermore, additional features like circuit hierarchy browsers also make use of ISE-generated data and thus also remained unavailable.

However, establishing a complete and stable ISE binding for simulation of Xilinx-based circuits proved more difficult than initially expected.

4.1 Modifications to the Verilog Models

First, in order for ISE to find the existing simulation models of the Xilinx library, a new symbolic link has to be established. The next step is the transformation of the file and module names of the models to be consistent with their symbol names inside the schematic editor Composer. This can easily be accomplished by a short `perl` program

which strips the `_[34]K` suffix from the names and folds them into lower case.

4.2 Modifications to the Composer Symbols

Now that the simulation models can be found by ISE, the next problems develop: The symbols inside Composer have their input ports labelled 1, 2, 3, ... but the simulation models use equivalent names A, B, C, ... This causes errors during compilation because the numeric labels are not valid Verilog port names and they don't match the port names in the models.

This difficulty was overcome by renaming all ports with numeric port names to their equivalent single-letter label. Two steps are necessary for the desired result. The first Skill procedure applied to the library-to-be-converted iterates over all schematic and symbol cells and changes their terminal names appropriately. The second step, also implemented in Skill, consists of recursively updating the connectivity information of each schematic cell and its subcells by extraction.

4.3 Handling Invisible Global Signals

While the steps described thus far allow ISE simulation of designs consisting only of simple combinational cells, sequential cells and tri-state pads have additional pitfalls

Both XC3000 and XC4000 chips use a chip-wide global reset signal to initialize all sequential elements on the chip. On XC4000 chips, an additional global signal to control the tri-stating of pads is also provided. These signals are not user-wireable and consequently not provided on the Composer symbols. However, since they are essential for correct simulation, they are contained in the port lists of the concerned cells' simulation models.

This arrangement leads to errors when netlisting circuits containing such cells. Since the ISE Verilog netlister uses only the Composer connectivity information, which is based on explicit wiring between ports on symbols, it misses the implicit global signals. When using the non-ISE simulation environment as provided by Cadence for Xilinx-based circuits, the missing signals are filled in during the EDIF to Verilog translation step.

Since ISE accesses the design database directly and the detour of an EDIF export is avoided, another way to add the global signals to the Verilog netlist has to be devised.

A solution was found in the form of a patch to the ISE Verilog netlister supplied by Cadence. The patch is applied by deploying a short Skill program with the name of the original Verilog netlister which loads the renamed original netlister and then substitutes the procedure `hnlVerilog-Print-ExplicitNetlist.` This procedure is responsible for writing connectivity information for a single cell instance to the Verilog netlist. By keeping a list of the cells which require the addition of the implicit global signals and checking the instances being netlisted against it, the appropriate signals can be added to the connectivity information. The global signals themselves are provided on the top level schematic as input pins with predefined names (GSR, GTS, GLOBALRESET_) and can thus be stimulated during simulation.

This step completes the integration into ISE. Xilinx-based circuits can now be easily stimulated using STL, smoothly simulated using the fully encapsulated ISE Verilog and comfortably observed using cWaves (instead of the rather spartan $gr_waves) and schematic monitors.

Furthermore, mixed simulation of designs containing schematic as well as Verilog cells is now possible.

5 Adapting Design Implementation Steps

The solution obtained thus far has the disadvantage, that different libraries are required for simula-

tion in ISE and design implementation using Xilinx XACT.

The reason is the renaming of ports in the ISE-compatible symbols. Since the Xilinx design translation tool `EDIF2XNF` depends on the numeric port names unsuitable for simulation, it cannot use the modified symbols.

However, we managed to work around this by providing the EDIF-to-Xilinx netlisting process with configuration files that transparently re-map names from ISE to XACT conventions, thus eliminating the need for separate libraries. These files, which have to be placed in the `edif[34]000` directories, are named `[34]000.map` and are read by `EDIF2XNF` during the conversion. The result is a legal XNF version of the design that can then be passed to the place and route tools.

6 Conclusions

These alterations achieve a complete integration of the Xilinx design kit into the Opus framework, enabling users to employ similar techniques both in semi-custom as well as FPGA design. We have since written a tutorial covering both design styles which introduces novice users to Opus. This tutorial (written in German) is available from us in LaTeX source.

In spite of our improvements, we are still not satisfied with the current state of the Cadence/Xilinx tools. For example, the EDIF export, which is part of the design implementation process, is highly error prone in 9301. This can range from corrupted EDIF files with missing signals to crashes of the program. Furthermore, the back annotation procedure frequently has problems restoring original signal names. The files created are thus incompatible with previously written test stimuli. Unfortunately, both tools are completely beyond our control.

We hope to find these fundamental problems corrected in the brand new (at least for EUROCHIP members) 9401 release but have not had sufficient time to conduct conclusive experiments.

Note

If you are interested in obtaining the sources to the modifications and programs mentioned in this paper, please contact the author at the e-mail address given at the beginning. We will attempt to port our environment to 9401 but are unable to give estimates on the timeframe required.