Rapid Prototyping Tools for FPGA Designs: RapidSmith

Christopher Lavin [1], Marc Padilla, Philip Lundrigan, Brent Nelson [2], Brad Hutchings [3]

NSF Center for High-Performance Reconfigurable Computing (CHREC)
Dept. of Electrical and Computer Engineering
Brigham Young University
Provo, UT, 84602, USA
[1] chrislavin@byu.edu, [2] brent_nelson@byu.edu, [3] brad_hutchings@byu.edu

(Demonstration Paper)

***Abstract*—Designer productivity for FPGA design is significantly limited by the time-consuming nature of the FPGA compilation process (synthesis, map, placement, and routing). However, experimentation on alternative CAD tools for this purpose for Xilinx devices has been somewhat limited. This paper describes the development and distribution of RapidSmith, a software library to facilitate the manipulation of XDL designs and upon which a complete CAD system can be based. The demonstration portion of this paper will show prototypes of representative CAD tools which can be easily built on top of the RapidSmith system.**

I. Introduction

It is well-known that FPGA compilation times are many times slower than the rapid compile times of software systems. Further, faster FPGA compilation times would directly translate into improved productivity for designers because hardware engineers would then be able to complete more design and debug *turns per day*. Unfortunately, FPGA implementation times are not getting much faster, largely because devices keep getting bigger with every generation. In an FPGA design flow, the target hardware is also often available for design verification use during much of the development process. However, it often goes unused because there is no fast method to leverage it for verification during the design process. Thus, what is needed is a rapid mechanism for mapping a design onto an existing FPGA platform to perform rapid design and debug cycles, providing a more interactive development environment for FPGAs.

Some work has been done in efforts to obtain reusability using hard FPGA circuit cores to obtain faster FPGA compile times. Horta and Lockwood [1] demonstrated the creation of bitstream-based relocatable cores. Similar efforts were made in [2] where bitstream hard cores were used in a network on chip to provide accelerated logic emulation and prototyping. Unfortunately, using bitstream hard cores has restrictions in that they must reside between configuration boundaries and require matching bus macro interfaces to be present both in the core as well as in the existing FPGA configuration. Similar work by Tessier [3] shows use of pre-placed macroblocks to accelerate place and route by 2.6× over commercial tools. However, the macroblocks did not include any routing information.

There are two main challenges associated with trying to create a module-based FPGA design flow. First, there must be a fast and simple way to create modules that contain as much relative placement and routing information to provide reasonable speedup. The second challenge is that there exists no real framework for commerical FPGAs that would allow such a design flow to be built upon. To accurately report credible speedup and fast build times, we feel it necessary to perform such experiments on actual FPGAs. This work intends to overcome these two challenges by proposing a new hard macro-based design flow that, like software would create and use pre-compiled modules (or "hard macros") from a library that could be rapidly assembled, placed and routed for a rapid implementation. We also describe and demonstrate a new open source software framework to make such a design flow possible on Xilinx FPGAs. This framework is called RapidSmith.

In the remainder of this paper we first describe our proposed hard macro-based design flow and also a series of experiments to investigate the feasibility of the hard macro-based system. In the process of describing this we show the need for XDL-based design tools to support the creation of a new design flow. Finally, we describe such an XDL design tools framework called RapidSmith and outline the demonstrations which will be shown at the conference.

II. Hard Macro-Based Design: HM*Flow*

In order to accelerate the Xilinx FPGA design flow, we propose using hard macros as the basic design element for all designs. That is, designs are constructed by assembling together pre-placed and pre-routed hard macro blocks. We call this new FPGA design flow HM*Flow*.

One of the obvious benefits of using hard macros in a design flow is that there is no need to run synthesis, mapping, or packing when the final design is assembled. The removal of these three steps of the design flow represent a significant fraction of the overall runtime. For example, in a typical EDK

This work was supported by the I/UCRC Program of the National Science Foundation under Grant No. 0801876.

MicroBlaze design, these three steps can take more than 75% of the total implementation time [4].

An additional benefit of using hard macros as a method for faster design builds is that hard macros are relatively-placed and routed. Thus, only the hard macro needs to be placed instead of all of its individual components. This ultimately reduces the placement problem size significantly as a conventional design may have thousands of primitive instances to place, whereas a hard macro-based design may only have a few dozen hard macros to be placed. Furthermore, hard macros contain internal routing. Since the FPGA configuration routing fabric is generally homogeneous, pre-routed hard macros should be able to be placed nearly anywhere on the device. This also has the potential to significantly reduce the total number of routes to be routed in a design.

III. Initial Hard Macro Experiments

In order to understand how well such a hard macro-based tool flow is currently supported by the Xilinx tools, we ran a series of experiments which were previously reported in [4].

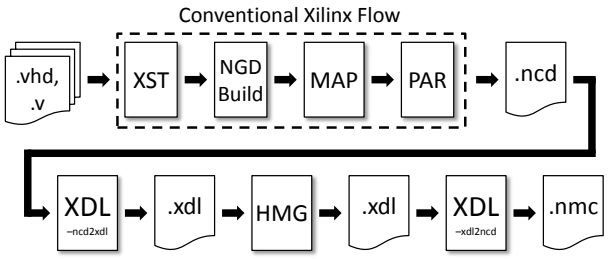

Fig. 1. Hard Macro Creation Flow

To create a set of hard macros for use in our experiments, we developed a Hard Macro Generator tool (HMG) and associated flow. Specific hard macro generation programs have been created before as in [5] and [6]. However, to our knowledge a tool to create general hard macros from arbitrary RTL has not been created before. All of our circuit manipulations were performed in XDL (XDL is a human-readable format equivalent to the more widely used NCD format) and was built on the RapidSmith framework detailed in Section IV. This flow leveraged the first four steps of the conventional Xilinx design flow as shown in Figure 1. Using the conventional Xilinx tool flow, we created a complete, placed and routed FPGA design which contained just the circuitry for the hard macro we wanted to create. The result was an XDL representation of the hard macro which was then converted to the Xilinx NMC format (the Xilinx hard macro counterpart format to NCD).

We then created structural VHDL which instanced these hard macros and then ran the Xilinx tool flow on this VHDL from synthesis through place and route. Since the design was an interconnected set of black boxes in the VHDL there was no real synthesis performed nor was there any mapping or packing done. In all, a number of such tests were completed with varying results. To summarize the results: (1) for some tests the design successfully placed and routed, but the resulting tool chain time was significantly longer than an equivalent process starting from a standard HDL-based design, (2) for others no valid placement for the hard macros in the design could be found and the placement phase failed, and (3) for other tests, placement would complete successfully but the router would fail with an error message that it could not route all the nets in the design. These results suggest that, in spite of the intuitive advantages that a hard macro-based design flow would seem to offer, Xilinx PAR does not work well with such a flow. The results further suggest that placement was the problem—either it failed to place the designs, or it seemed to create an un-routable placement.

Our second set of experiments was designed to bypass the Xilinx placement step to understand the router's handling of hard macros. To do so, the original VHDL-based structural design was processed as before but only up to mapping and packing. The result was an unplaced NCD file. The hard macros were then placed by hand using a custom created XDL macro placer tool. The resulting placed design was then passed to the Xilinx router (PAR) for completion.

The first design tested was a multiplier tree containing 15 identical instances of a 20×20 pipelined multiplier macro, organized as a binary tree of multipliers. The second design was a collection of five different cores: CORDIC, AES decryptor, Twofish (a symmetric key block cipher), FM Receiver, and Hilbert Analytic Filter. All cores were placed in the design and connected together to form a data-path with inputs and outputs connected to external IO pins. The results of these tests showed that the resulting circuits were implemented $3.1\times$ faster than a conventional VHDL-based Xilinx flow. The circuits produced were between 6% and 50% larger but had clock rates within 25% of the baseline designs. In all, this provided reasonable evidence that a speedup is possible for a hard macro-based flow. See [4] for a more complete description of the results.

IV. RapidSmith Overview

Based on the promising results derived from the experiments described in the previous section, we began to construct such a hard macro-based design flow. However, in order to do so, we found no adequate open research tool to build upon to utilize the XDL interface. There certainly have been several research projects using XDL such as [7], [8] and [9], but the source code associated with these projects is not publicly available. The implementation of our design flow required an efficient XDL manipulation tool that would facilitate a fast hard macro-based design flow and also allow for rapid development.

Since no such tool is available for XDL, we have developed our own framework and tools which we call RapidSmith (http://rapidsmith.sourceforge.net). RapidSmith is an XDL manipulation framework, written as a collection of Java packages which provides a software API for reading, writing, and manipulating Xilinx designs represented as XDL files. Our goal in creating RapidSmith was to create a foundation upon which to build a variety of FPGA CAD tool research projects. RapidSmith contains several data structures to represent the

primitives making up a design such as instances, nets, pins, switch matrix connections, tiles, etc.

In order to understand RapidSmith, a rudimentary understanding of the Xilinx Design Language (XDL) is needed. XDL is a human readable ASCII file format which is an alternative to the Xilinx proprietary NCD format for representing Xilinx designs. Xilinx distributes an unsupported `xdl` executable with its design tools that has the capability to convert NCD circuit description files to XDL and vice versa. The basic building blocks of XDL files are primitives such as SLICEs, DSP48s, BRAMs, and DCMs (there are a few dozen FPGA primitive types). In addition, XDL files describe how each of the primitives are configured (LUT contents, for example) and possibly placement information. XDL files can also describe both the logical as well as physical interconnections (nets and wires) between primitive instances. Therefore, NCD and XDL files can represent a design in essentially any developmental state from unplaced/unrouted designs to fully placed and routed designs.

In addition to being able to convert NCD files to and from XDL, the `xdl` executable has a `-report` mode that will create a detailed description of a particular Xilinx FPGA device. This information (an XDLRC file) provides the information required for RapidSmith to manipulate the XDL for a particular FPGA device. These XDLRC files can grow to several gigabytes in size because of the extreme detail they provide on the structure of a specific Xilinx device. Thus, an issue in the development of RapidSmith was optimizing its device representation to ensure a minimal footprint and fast load times. During the installation of RapidSmith a compact representation for the Xilinx devices of interest is created from XDLRC files. The largest of these is for an xc5vlx330 part (the largest Virtex 5 part) and requires only 154MB of heap space to hold at runtime which is significantly reduced from the 13GB XDLRC report file used to describe the part. This device file only consumes 1.04MB of disk space and can be loaded from disk in 1.87 seconds on a Windows XP SP3 workstation with a 3.0GHz Intel Core 2 Duo E6850, 4GB RAM and using the Java Sun JVM 1.6.0_21.

RapidSmith contains 9 different Java packages containing 70 classes and over 700 methods for design manipulation and serves as a foundation for rapid prototyping of FPGA CAD ideas and algorithms. Although there are several packages and classes, there are only two main abstractions that users need to understand about the API: the design and the FPGA device.

Designs in RapidSmith are composed of 4 main components as shown in Figure 2a: instances, nets, modules and module instances. Each of these elements have their own class abstraction in RapidSmith. These design building blocks can undergo a variety of transformations using the API such as creation, placement and routing, reconfiguration and repurposing, and logic transformation by changing attributes. Modules define collections of instances and nets with externally defined ports that act as macros. Module instances are actual realized copies of such macros in a design. These are powerful constructs that allow for radically different design creation methods on

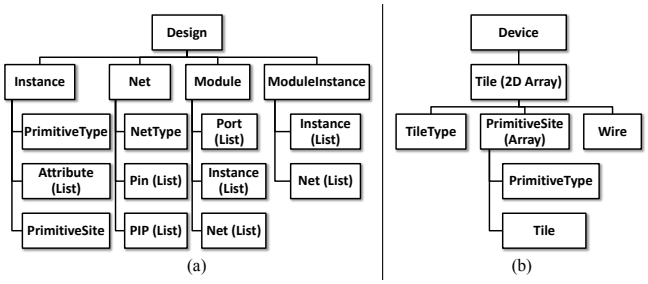

Fig. 2. RapidSmith class abstractions for designs (a) and FPGA devices (b).

FPGAs.

FPGA devices are the second abstraction in RapidSmith and represent the logic and routing fabric available to a designer (see Figure 2b). A Xilinx FPGA is conceptually divided into a grid of tiles which contain primitive sites and wires. A primitive site is a valid location for a design instance to be placed on the FPGA fabric. Most tiles also have programmable wires called PIPs as well as wires that extend beyond the tile boundary. All wires in RapidSmith are enumerated as the Java primitive type `int` to reduce memory usage. The device class provides all of the necessary information and APIs to build full placement and routing algorithms for Xilinx FPGAs.

Because of the abstractions and conveniences mentioned in RapidSmith, explorative design analysis and rapid prototyping of new ideas and algorithms can be very fast. The framework provides all of the necessary methods and classes to build fast FPGA placement and routing algorithms as well as implement on-the-fly hardware synthesis techniques. Further, the RapidSmith API can be easily integrated into a graphical environment for visualization of the FPGA hardware and design elements. Examples of these kinds of functionality are available with the RapidSmith distribution (http://rapidsmith.sourceforge.net) in the `examples` package.

V. Demonstration

As mentioned above, we believe RapidSmith provides an excellent foundation for the creation of FPGA design tools. In this section we will describe the demonstrations that will be performed at the conference.

A. Design Analysis Tools

Given a design in the XDL format, RapidSmith provides an easy-to-use platform for creating design analysis tools. One example of such a tool is shown in Figure 3. This is a simple design browser tool. In the figure you can see primitive instances (SLICEL for example), pins, and routing pips.

Another such tool is shown in Figure 4, which shows the ability to search for primitive instances in a loaded design.

Another such tool which will be demonstrated is a design analyzer (not shown) which tabulates detailed resource utilization statistics, such as might be useful for tuning an automatic placement tool.

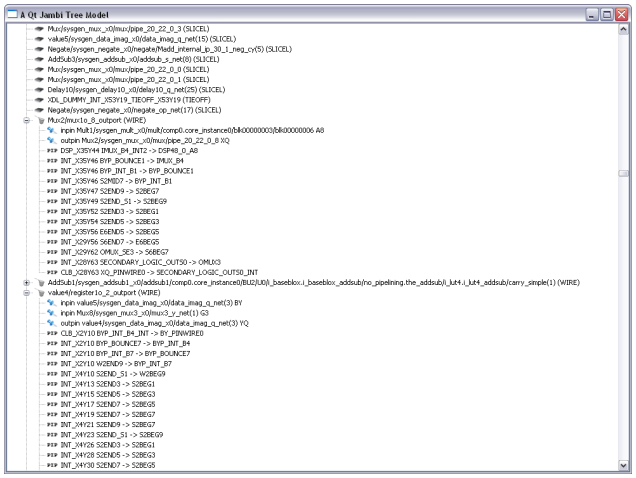

Fig. 3. Screenshot of a tree-based graphical list traversal of an XDL design.

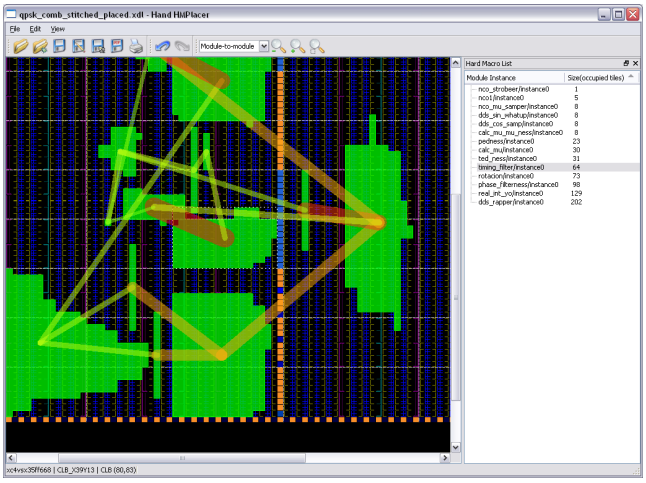

Fig. 5. Screenshot of an interactive hard macro placer built on RapidSmith.

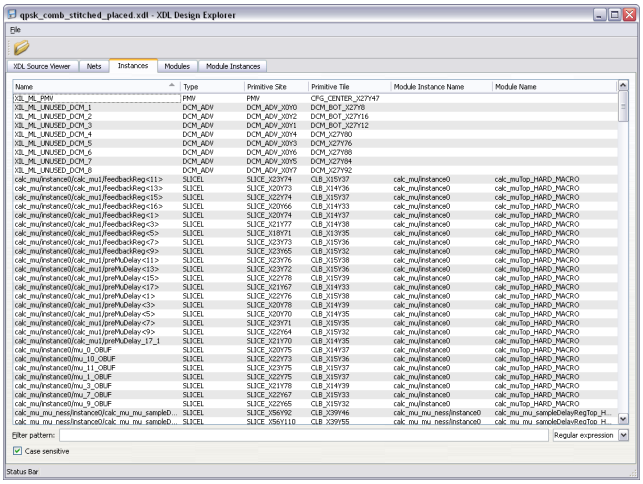

Fig. 4. Screenshot demonstrating a program using RapidSmith to interactively search XDL Instances.

B. Design Creation Tools

Not only can RapidSmith read and write XDL files, its API provides a mechanism to create circuitry in XDL format. API routines exist to create primitive instances, customize those instances, and create logical connections between primitive instance pins. A simple design creation tool like this will be demonstrated.

C. Physical Design Tools

Finally, RapidSmith provides an excellent framework for the creation of physical design tools such as placement and routing tools. Figure 5 shows the screenshot of a manual hard macro placement tool. We have used this tool in the creation and testing of various automatic placement tools, one or more of which will be demonstrated.

In conclusion, RapidSmith is an XDL manipulation system which provides a framework upon which to build a complete FPGA design tool suite. The demonstrations which will be shown will illustrate the variety of circuit manipulations that can be performed. As mentioned above, RapidSmith is being distributed as an open source tool at http://rapidsmith.sourceforge.net. It is our hope that others will adopt RapidSmith for their research which will help further develop its capabilities and provide new opportunities for research in FPGA CAD tools. One example is our current HM*Flow* project, designed to demonstrate the benefits of hard macro-based design for Xilinx FPGAs.

References

[1] E. L. Horta and J. W. Lockwood, "Automated Method to Generate Bitstream Intellectual Property Cores for Virtex FPGAs," in *Proc. Field Programmable Logic.2004*, 2004.

[2] Y. E. Krasteva, F. Criado, E. d. l. Torre, and T. Riesgo, "A Fast Emulation-Based NoC Prototyping Framework," in *RECONFIG '08: Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 211–216.

[3] R. Tessier, "Fast Placement Approaches for FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 7, no. 2, pp. 284–305, 2002.

[4] C. Lavin, M. Padilla, S. Ghosh, B. Nelson, B. Hutchings, and M. Wirthlin, "Using Hard Macros to Reduce FPGA Compilation Time," in *Proceedings of the 20th International Workshop on Field-Programmable Logic and Applications*, 2010, to appear.

[5] C. Claus, B. Zhang, M. Huebner, C. Schmutzler, J. Becker, and W. Stechele, "An XDL-based Busmacro Generator for Customizable Communication Interfaces for Dynamically and Partially Reconfigurable Systems," in *Workshop on Reconfigurable Computing Education at ISVLSI 2007*, Porto Alegre, Brazil, May 2007.

[6] "Using Three-State Enable Registers in 4000XLA/XV, and Spartan-XL FPGAs (XAPP123 v2.0)," Xilinx Inc., Tech. Rep., January 2002.

[7] N. Steiner, "A Standalone Wire Database for Routing and Tracing in Xilinx Virtex, Virtex-E, and Virtex-II FPGAs," Master's thesis, Virginia Tech, August 2002.

[8] K. Kepa, F. Morgan, K. Kościuszkiewicz, L. Braun, M. Hübner, and J. Becker, "FPGA Analysis Tool: High-Level Flows for Low-Level Design Analysis in Reconfigurable Computing," in *ARC '09: Proceedings of the 5th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 62–73.

[9] P. Graham, B. Nelson, and B. Hutchings, "Instrumenting bitstreams for debugging fpga circuits," *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, vol. 0, pp. 41–50, 2001.