Torc: Towards an Open-Source Tool Flow

Neil Steiner[1]
neil.steiner@isi.edu

Aaron Wood[1,2]
awood@isi.edu

Hamid Shojaei[1,3]
shojaei@wisc.edu

Jacob Couch[4]
jacouch@vt.edu

Peter Athanas[4]
athanas@vt.edu

Matthew French[1]
mfrench@isi.edu

[1]Information Sciences Institute
University of Southern California
3811 N Fairfax Dr, Ste 200
Arlington, VA 22203

[2]Department of Electrical Engineering
University of Washington
185 Stevens Way
Seattle, WA 98195

[3]Dept of Electrical and Computer Engr
University of Wisconsin-Madison
1415 Engineering Drive
Madison, WI 53706

[4]Dept of Electrical and Computer Engr
Virginia Tech
302 Whittemore Hall
Blacksburg, VA 24061

ABSTRACT

We present and describe Torc—*Tools for Open Reconfigurable Computing*—an open-source infrastructure and tool set, provided entirely as C++ source code and available at http://torc.isi.edu. Torc is suitable for custom research applications, for CAD tool development, and for architecture exploration.

The Torc infrastructure can (1) read, write, and manipulate generic netlists—currently EDIF, (2) read, write, and manipulate physical netlists—currently XDL, and indirectly NCD, (3) provide exhaustive wiring and logic information for commercial devices, and (4) read, write, and manipulate bitstream packets (but not configuration frame *contents*). Torc furthermore provides routing and *unpacking* tools for full or partial designs, soon to be augmented with BLIF support, and with packing and placing tools.

The architectural data for Xilinx devices is generated from non-proprietary XDLRC files, and currently supports 140 devices in 11 families: Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex4, Virtex5, Virtex6, Virtex6L, Spartan3E, Spartan6, and Spartan6L. We believe that Altera architectures and designs could be similarly supported if the necessary data were available, and we have successfully used Torc internally with custom architectures.

Categories and Subject Descriptors: J.6 [Computer Applications]: Computer-Aided Engineering

General Terms: Algorithms, Design, Standardization.

Keywords: C++, FPGA, place, route, unpack, EDIF, XDL, XDLRC.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FPGA'11, February 27–March 1, 2011, Monterey, California, USA.
Copyright 2011 ACM 978-1-4503-0554-9/11/02 ...$10.00.

1. INTRODUCTION

Modern FPGAs are complex devices, and the designs targeting them are often described in complex file formats. As a result, researchers either have to work with simplified files and models, or have to invest significant development effort into parsers, object models, and large routing graphs.

Special requirements on a large project forced us to develop custom tools for internal use, which we are now repackaging and releasing to the research community as an open-source project. One aim is to provide *real* device data, and thereby increase the relevance of the CAD tools that are frequently developed by researchers. Another aim is to provide a framework for device and design data, allowing researchers to focus on the unique and novel aspects of their work, instead of being waylaid by infrastructure development.

Device manufacturer data is often sensitive and proprietary, and we ensure that the data and capabilities underlying Torc—including exhaustive device databases—are derived from publicly available sources. As a result of this, Torc can manipulate bitstream packets and configuration frames, but does not understand frame *contents*.

We divide Torc into a collection of Application Programming Interfaces (APIs) and Tools: The *generic netlist* API supports unmapped netlists, most commonly EDIF. The *physical netlist* API supports mapped netlists, with or without placement or routing information, most commonly XDL. The *device architecture* API provides exhaustive wiring and logic information for supported commercial or experimental devices. And the *bitstream* API supports Xilinx bitstream packets and frames.

The *router* tool creates connections for anything from individual wires to entire designs. And the *unpacker* tool expands compound blocks such as SLICEs into constituent Look-Up Tables (LUTs), flip-flops, and muxes. Two tools still in development are the *packer* tool, which recombines what the unpacker has expanded, and the *placer* tool, which assigns physical locations to design logic elements.

We provide background information in Section 2, and then describe the Torc design in Section 3, the API in Section 4, and the associated CAD tools in Section 5. We finally consider applicability in Section 6 and conclude in Section 7.

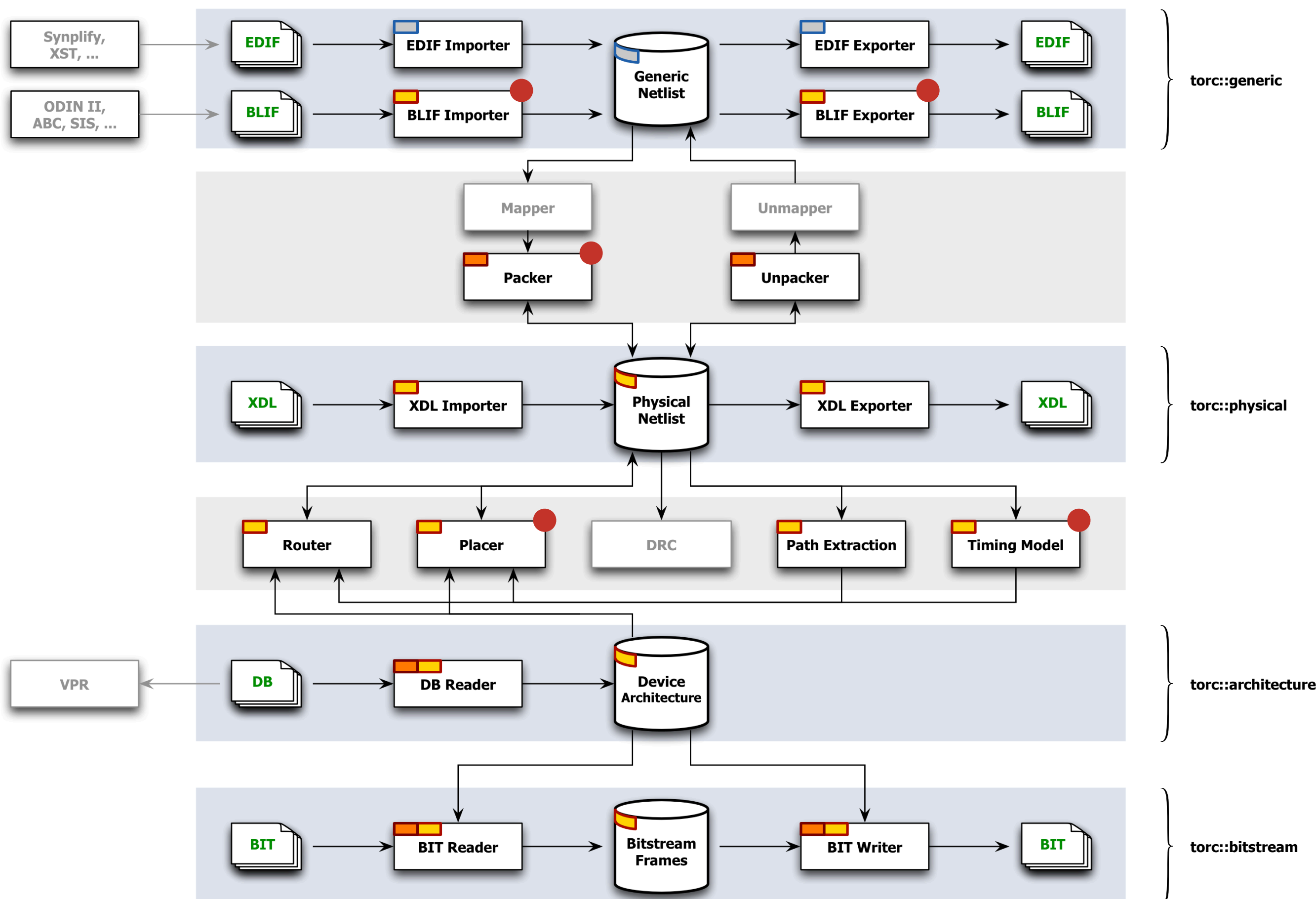

Figure 1: Torc block diagram. ***Red dots indicate components still under development.***

2. BACKGROUND

Torc functionality is heavily based on prior work, much of it previously unreleased for reasons beyond our control. Torc's Device Database is a direct descendant of ADB [11], originally developed in conjunction with the JBits [5] project, and later to become the foundation of a completely rewritten but unreleased version of JBits.

We know of no other API or tool set comparable to Torc, other than friendly competition from our collaborators at Brigham Young University (BYU) [6]. There are however a number of important related tools, some of which we hope to interface with to mutual advantage. In doing so, we believe we can usefully expand the number of research tool flows described on the fpgaCAD web site [4].

At the mapped netlist level, VPR [1] is the de facto place-and-route tool for research, and has been used for modeling in the development of recent Altera architectures [7]. At the mapped netlist level, many groups have dabbled with XDL designs or XDLRC device data, but few if any of those efforts have resulted in open toolsets.

Torc aims to be reasonably architecture independent, but is strongly influenced by the disparity in device and design information available for Xilinx and Altera architectures. While Xilinx provides insertion points and architecture data necessary for changes after synthesis, after mapping, after place-and-route, and even after bitstream generation, the available insertion points and architecture data for Altera devices are insufficient for our needs.

Despite the limited access to Altera wiring data and low-level implementation files, we believe that Torc's architecture API could represent Altera architectures, and that the physical netlist API could probably describe low-level Altera circuit implementations. The same is not true for the bitstream API, which is derived almost entirely from configuration information in Xilinx device user guides.

3. DESIGN

Torc's major components are depicted in Figure 1. The APIs are shaded in blue and are designated on the right according to their respective C++ namespaces. The tool sets are shaded in gray, and are positioned between the APIs that they depend upon. Input and output file types are labeled in green. The components still undergoing development or integration have red dots in their upper right corners, while the components not yet scheduled for development are grayed out. The remainder of the color coding reflects internal task assignments for ISI, Virginia Tech, and Interra Systems.

Also present on the left of the diagram are groups of tools that we hope to interface with, including commercial synthesis engines by way of EDIF files, and academic synthesis and optimization tools by way of BLIF files. We also hope

to supply routing graphs for real devices to VPR, further confirming and extending its usefulness.

4. API

The core functionality of Torc is encapsulated in the four databases depicted in Figure 1: The generic netlist API, the physical netlist API, the device architecture API, and the bitstream API.

4.1 Generic Netlist

The generic netlist API supports netlists that are not mapped to physical primitives in a target device. The API supports the ubiquitous EDIF 2 0 0 Level 0 [3], and can manipulate the NETLIST view type. Support for the academically popular BLIF format is still under development, and is intended to provide compatibility between Torc and existing research tools and flows.

The API includes EDIF importers and exporters, and is built around an internal *netlist object model*. The EDIF support and the object model were developed for us by Interra Systems, Inc., a company with significant expertise in front-end language analyzers, and with a customer base that includes some of the largest EDA vendors.

The generic netlist API can be used to access all of the circuit design elements, including libraries, cells, views, ports, instances, nets, and more. These elements can be queried, added, modified, or removed, and entirely new designs can be created from scratch. In addition to circuit manipulation, the API can also flatten netlists, and can provide a foundation for synthesis or mapping algorithms.

4.2 Physical Netlist

The physical netlist API supports netlists that have been mapped to physical primitives in the target device. Physical netlists may include partial or full placement and routing information, or may be devoid of any such information.

In a manner similar to the generic netlist API, the physical netlist API can be used to access all circuit elements, including designs, modules, nets, and instances, along with their configuration settings and any placement and routing information.

There are two reasons why physical netlist capabilities give the user exceptional control over their design: (1) Unlike the ISE place-and-route tool, Torc allows users to strictly enforce their requirements, making it possible to generate and retain arbitrary routes, or to reserve arbitrary resources or regions of the device. (2) There are no subsequent mapping or transformation steps performed before bitstream generation, so the user is guaranteed that any changes will be retained as applied. Comparable assurances are much more elusive at the generic netlist level.

4.3 Device Architecture

The device architecture API is built upon proven methods and representations for very large and irregular devices [11]. A precursor to this code was successfully used in an embedded system, allowing that system to autonomously place and route new circuitry within itself while continuing to run [10].

The Device Database includes exhaustive knowledge of the device wiring and logic, and makes all of that information available through the API. It also tracks wire and arc usage, to prevent contention with existing nets or logic, or simply to inform routers of resource availability. Furthermore, it provides the physical and bitstream APIs with tile maps, logic site maps, and usage information.

For router research, including tools such as VPR which may expect to work within a fully expanded routing graph, we note that one could expand the graph ahead of time, at the cost of significant memory overhead.

4.4 Bitstream Interface

The bitstream API supports reading, modifying, and writing bitstream packets and frames for supported Xilinx architectures. We note again that the API can work with configuration frames but lacks any understanding of frame contents, since that information is proprietary and undocumented. Unfortunately, no comparable information is available for Altera architectures.

5. TOOLS

Torc includes CAD tools to perform unpacking and routing, with packing and placing tools still being developed or integrated. The tools are provided as source code, rather than executables, and can serve as guides for working with the physical netlist API and the device architecture API. If compiled as standalone executables they can be substituted into the regular design implementation flow, with a few stipulations pertaining to timing and to multiple clock domains.

5.1 Unpacker and Packer

The configurable computing community generally speaks of *packing* in the sense of combining logic functions or gates into LUTs, or of combining LUTs and flip-flops into simple logic blocks or clusters, mindful of the impact on circuit performance, but with very few physical rules to constrain such operations. We use the term here to describe the process of combining LUTs, flip-flops, muxes, carry chains, and other elements into logic block primitives—Virtex SLICEs for example—while generating the corresponding configuration settings, and respecting device rules.

The value of a logic block packer is that it allows the user to work with circuitry much more naturally, in terms of LUTs, flip-flops, and other basic elements. In the common case of a user working from an existing design, it makes sense to first unpack it, modify the circuit as needed, and then incrementally re-pack, re-place, and re-route it. With the packer still under development, the unpacker nevertheless plays an important role by exposing synchronous and asynchronous circuit elements, to facilitate combinational path analysis. These combinational paths are then ordered according to logic level depth, and are used by the placer and router when prioritizing nets and resources.

5.2 Router

The router constructs paths that connect sources to sinks for every net, with final results that must meet timing requirements and be free from contention. The Torc routing capability includes an optional preliminary router, along with a global PathFinder [8] implementation to resolve net contention, and an underlying signal router based on an A* search [9]. We note that it is possible to bypass the preliminary router, and that it is also possible to invoke the signal router directly on nets or arbitrary device wires.

Regrettably absent from the available Xilinx architecture data is any timing information, and even ISE's timing analysis tool reports delays for nets rather than their constituent wires and arcs. Consequently neither the Device Database nor the router can accurately analyze or guarantee timing. We alleviate the issue in part by prioritizing long combinational paths and thus reducing the delay of the likely critical paths. We have also had considerable success modeling wiring delays based on technology process nodes, but have not yet integrated the resulting timing data into Torc.

6. APPLICATIONS AND FUTURE WORK

Torc is useful for any application that requires very fine-grained programmatic control over design implementation. Sometimes it is necessary to constrain a design in ways that are not possible with the vendor implementation tools. Taking the AREA_GROUP constraint as an example, although a user may wish to strictly constrain routing inside or outside of a specified boundary, the tools do not always produce the desired results. But Torc permits the user to allow, disallow, or even require the use of arbitrary routing resources. The user could also programmatically generate some portion of their design with the physical netlist API, and add highly structured placement in the manner of Lava [2].

Torc is also useful as a platform for CAD tool research. The possibility of broadly linking design information—at the HDL, generic netlist, and physical netlist levels—opens the possibility of tightly-coupled interactive incremental debug and development. We note growing interest in parallel tool research, but entirely new approaches may also emerge: It may be feasible to partition designs at the generic netlist level, and then to perform combined mapping, placing, and routing on each partition concurrently.

Torc tools still under development include the packer and placer. Additional capabilities that we hope to add in time include timing models, constraint file support, design rule checks, TCL scripting, parallelization, mapping, and support for other architectures.

7. CONCLUSION

Torc is an open-source C++ infrastructure for reconfigurable computing, suitable for custom research applications, for CAD tool development, and for architecture exploration. Its primary purpose is to promote and facilitate research, by providing a framework for device and design data, allowing researchers to focus on the truly novel and unique aspects of their work.

The Torc infrastructure can (1) read, write, and manipulate generic netlists—currently EDIF, (2) read, write, and manipulate physical netlists—currently XDL, and indirectly NCD, (3) provide exhaustive wiring and logic information for commercial devices, and (4) read, write, and manipulate bitstream packets (but not frame contents). Torc's use of standard file formats also allows complete simulation and timing analysis with standard commercial tools.

Torc furthermore provides unpacking and routing tools, with packing and placing tools still under development. Packing and unpacking allow users to more naturally work with basic architecture-independent elements, instead of working with more complex physical primitive instances. And though we are not yet deploying device timing models, we have had good initial success with our internal efforts.

The architectural data included with Torc is derived entirely from non-proprietary sources. We presently support 140 Xilinx devices in 11 families, and are interested in adding Altera support if the necessary data can be obtained.

Torc is available at http://torc.isi.edu.

8. ACKNOWLEDGMENTS

ISI wishes to thank our project sponsor. Thanks also to our collaborators, including the Virginia Tech Configurable Computing Lab; Brad Hutchings, Brent Nelson, and the Brigham Young University Configurable Computing Lab; and Vijeta Kashyap and the team at Interra Systems.

9. REFERENCES

[1] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In W. Luk, P. Y. K. Cheung, and M. Glesner, editors, *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications, FPL 1997, (London), September 1–3*, volume 1304 of *Lecture Notes in Computer Science*, pages 213–222. Springer Verlag, 1997.

[2] P. Bjesse, K. Claessen, M. Sheeran, and S. Singh. Lava: hardware design in Haskell. In *ACM SIGPLAN Notices*, volume 34, pages 174–184, New York, NY, 1999. ACM.

[3] Electronic Industries Association. *Electronic Design Interchange Format*, 1988.

[4] fpgaCAD. http://fpgacad.ece.wisc.edu.

[5] S. Guccione, D. Levi, and P. Sundararajan. JBits: Java based interface for reconfigurable computing. In *Proceedings of the Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference, MAPLD 1999, (Laurel, Maryland), September 28–30*, 1999.

[6] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, B. Hutchings, and M. Wirthlin. A library for low-level manipulation of partially placed-and-routed FPGA designs. Technical report, Brigham Young University, 2010.

[7] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, et al. The Stratix II logic and routing architecture. In *Proceedings of the 2005 ACM/SIGDA 13th Annual International Symposium on Field-Programmable Gate Arrays, FPGA 2005 (Monterey, California), February 20–22*, pages 14–20, 2005.

[8] L. McMurchie and C. Ebeling. PathFinder: A negotiation-based performance-driven router for FPGAs. In *Proceedings of the 1995 ACM 3rd International Symposium on Field-Programmable Gate Arrays, FPGA 1995, (Monterey, California), February 12–14*, pages 111–117, 1995.

[9] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Company, Palo Alto, California, 1980.

[10] N. Steiner and P. Athanas. Hardware autonomy and space systems. In *Proceedings of the 2009 IEEE Aerospace Conference, (Big Sky, Montana), March 7–14*, 2009.

[11] N. J. 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.