

WIRES ON DEMAND: RUN-TIME COMMUNICATION SYNTHESIS FOR RECONFIGURABLE COMPUTING

*P. Athanas, J. Bowen, T. Dunham, C. Patterson,
J. Rice, M. Shelburne, J. Surís*

Configurable Computing Lab
Dept. of Electrical & Computer Engineering
Virginia Tech
Blacksburg, Virginia 24061

M. Bucciario, J. Graf

Luna Innovations Secure Computing &
Communications Group
1703 South Jefferson St. SW, Ste 400
Roanoke, VA 24016

ABSTRACT

In systems typified by software defined radio, existing flows for run-time FPGA reconfiguration limit resource efficiency when constructing a variety of datapaths. Our approach allocates a sandbox region in which modules from a library can be flexibly placed and interconnected. An efficient run-time framework makes use of lightweight placement and routing techniques to respond on-demand to application requests. Compile time tools automate the task of adding interface wrappers to modules, insulating the designer from reconfiguration details.

1. INTRODUCTION

Contemporary computer engineering tries to develop systems with favorable tradeoffs between price, performance, power, adaptability and the effort (time and cost) required to use the technology. An axiom of reconfigurable computing research is that adding run-time adaptability to hardware can improve the three P's: price (by multiplexing the use of a smaller FPGA), performance and power efficiency (by specializing circuits to a problem instance). Even if the objectives are achieved, the significant increase in design effort works against the main attraction of FPGA technology. Reconfigurable application development remains daunting, largely because inter-module communication requires low-level physical design and is the responsibility of the designer.

Given the effort required to develop non-trivial, run-time reconfigurable (RTR) applications, the price / performance / efficiency return on investment needs to be substantial. The current approach to partial reconfiguration leads to an inter-module communication structure that remains fixed and often consists of one or more buses. However, the pervasive lesson in high-performance architecture is the importance of efficient communication. Because FPGAs are mostly uncommitted wires, we desire custom, point-to-point commu-

nication between dynamically instantiated modules in order to maximize communication efficiency.

RTR application design would be much easier if module communication circuitry was automatically synthesized. A relatively new research area, communication synthesis is an essential part of system-on-chip design productivity [1]. Commercial communication synthesis tools exist for ASIC design, such as Sonics' SMART Interconnect [2]. Designers need only provide a library of modules and memories (which often pre-exist as cores), and all connections and physical constraints are automatically generated. This degree of abstraction is sorely missing for RTR application development. As with software and static hardware design, reconfigurable applications should be insulated from rapidly evolving FPGA architectures. In contrast with existing approaches to communication synthesis, we partition the work into compile-time operations (preprocessing dynamically instantiated IP) and run-time operations (placing modules and completing connections).

This paper is structured as follows: Section 2 highlights previous efforts to implement RTR applications on FPGAs. The creation and use of a dynamic module library as well as inter-module routing options are described in Section 3. Section 4 illustrates the flow with a hardware/software implementation of an MP3 decoder. Finally, conclusions are summarized in Section 5.

2. THE EVOLUTION OF FPGA RUN-TIME RECONFIGURATION

Xilinx's efforts to promote RTR formed distinct phases that have some important lessons. The reconfiguration-friendly XC6200 architecture was the focus of the first phase [3]. Its commercial failure resulted from, among other things, poor support for reconfiguration in the associated tools, and a lack of architectural features (such as fast arithmetic) that designers were accustomed to. The second phase sensibly fo-

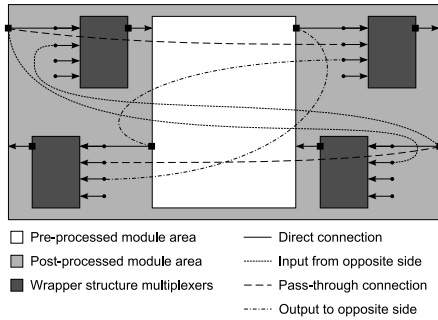


Fig. 1. Wrapper Structure Routing Options.

cused on reconfiguration tools for mainstream FPGA architectures, and resulted in the JBits Integrated Development Environment [4]. Run-time parameterized designs could be implemented without using the standard Xilinx tools by having a Java program configure all logic and connections in a structural manner [5]. However, most designers were not willing to forgo RTL design abstraction with familiar HDLs and timing-driven implementation tools.

Phase three has been in effect since 2002, and provides rudimentary support for partial reconfiguration in Xilinx’s mainstream implementation tools by adding constraints and special bus macros to the modular design flow [6]. In addition to the manual effort required to insert and place the bus macros, a number of limitations arise due to lack of a run-time environment. A set of reconfigurable regions may be allocated in a design, however they may not be stacked vertically because different configuration frames would be required for each combination of modules. Each region must be the size of the largest module that will occupy it. Inter-module routing resources are also fixed at design time. The constraints of this static approach result in the same inflexibility or resource waste as static array allocation in programs. As with software, the solution is dynamic allocation of reusable resources from a large pool.

3. IMPLEMENTATION OVERVIEW

3.1. Dynamic Module Library Preparation

The dynamic module library is composed of preprocessed IP blocks, stored in the form of partial bitstreams. Before compilation, blocks are encased in wrapper structures whose main function is to provide routing anchor points for block ports. A wrapper structure supporting horizontal dataflow is shown in Figure 1. Multiplexers allow run-time selection among same-side and opposite-side connections to module ports, and pass-through connections for signals unrelated to the module. As with the bus macros used in the Xilinx PR flow, wrappers consume a CLB row or column on each of a

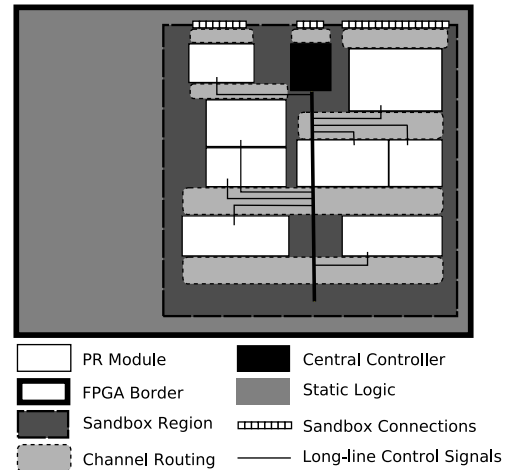


Fig. 2. Datapath Placement With Channel Allocation.

module’s port edges.

A module interface template describes the wrapper structure required by a particular IP block. Information in the template includes the port names and ordering, preferred block dimensions, dataflow direction, and routing options (such as the number of pass-through connections). IP block preprocessing takes as input the module’s port declarations and interface template, and produces HDL and constraints for a wrapped module. The mainstream tools are then invoked to generate one or more bitstreams for the module. Defining similar interface templates for a set of modules promotes port alignment when the modules are connected.

3.2. Module Placement and Channel Allocation

To reduce the time and memory requirements of the run-time placement process, placement occurs at the module level rather than at the gate level. This reduces the size of the problem from placing many thousands of cells to placing tens of blocks. Previous work often takes a naive view of the architecture by treating module placement purely as a packing problem and ignoring inter-module routing [7], or by considering only the architecture’s logic element grid, ignoring features such as BRAM [8].

The goal of datapath placement is to promote neighbor connections and reduce routing delays between blocks by minimizing the lengths of the connecting wires. Modules are first topologically sorted based on their connections. The precise placement of modules depends on the extra resources required, such as multipliers and BRAM. Datapaths are primarily horizontal or vertical with folds as necessary. Routing channels are allocated wherever modules do not connect strictly through abutment. Within the channel, delay estimation is performed based on wire lengths. Figure 2 shows an example of placement for vertical dataflow.

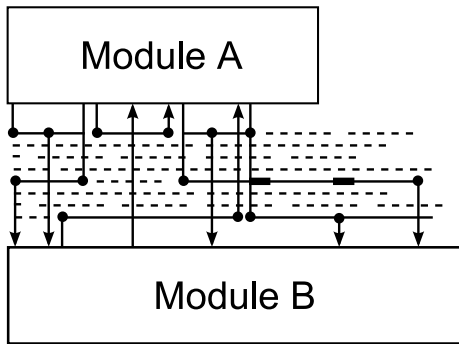


Fig. 3. Segmented Channel: Solid lines are used segments. Dark rectangles indicate joined segments. Dashed lines are unused segments.

3.3. Channel Router

Because contemporary FPGAs have a large amount of routing resources available, general routing is basically a graph search problem. By contrast, our inter-module routing requirements are limited to the channels reserved between the wrapper pins of adjacent modules. This approach permits routing with constructive algorithms based on templates that specify the sequence of wire segments to use. In fact, a simple greedy algorithm allows a good solution to be obtained [9]. Figure 3 shows how a channel incorporating jogs, loopbacks and multi-sink nets might be realized.

The router utilizes an abstract architecture that represents a subset of the wires and connections present in a CLB. By deriving the subset from resources common to two or more FPGA families, the abstract architecture allows channel routing to be treated in an architecture independent manner. The wires consist of unidirectional segments which span three CLBs and travel north, south, east or west. Each CLB contains the start, midpoint and end for ten segments in each direction. Connectivity in the abstract switch matrix is rich enough to support complex channels. After all signals have been routed in terms of the abstract architecture, routes are mapped to the corresponding resources in the actual architecture.

3.4. Long Line Allocation

In addition to the local wires used for channel routing, we consider the use of long lines for run-time connections. In the Virtex-II/Pro architecture, long lines span the entire chip width or height as continuous segments, while in the Virtex-4 and -5 families, they span 25 and 19 CLBs, respectively. Long lines are attractive in that they are not essential resources for routing within modules. Unfortunately, they suffer from sparse connectivity among CLBs, lean connectivity

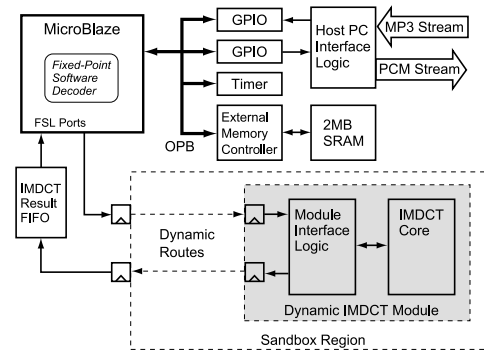


Fig. 4. MP3 Decoder Structure.

to other wires within a CLB, and low density.

While the use of long lines for general module I/O is illustrated in Section 4, the poor density of long lines may be prohibitive for modules having wide data ports. Long lines may be more useful for control signals related to run-time housekeeping, as depicted in Figure 2. Such communication might include a signal from a controller instructing a module to suspend or complete the current operation and prepare to be relocated or removed.

4. DESIGN EXAMPLE

This section presents a brief example that demonstrates the feasibility of flexible module placement and communication over dynamic routes. The choice of an MPEG-1 Layer 3 (MP3) audio decoder as the application was motivated by the algorithm's reliance on streaming data transfer between signal-processing stages. As shown in Figure 4, the decoder is a system-on-chip with a 36-point inverse modified discrete cosine transform (IMDCT) core. The module is faster than the software IMDCT function by a factor of 2.54 (including communication overhead), and speeds up the overall decoding process by a factor of 1.13.

Figure 5 shows the system implementation on a Virtex-II XC2V4000 FPGA. Because the module does not communicate with other dynamic modules, it uses long-line-specific anchor points rather than the wrapper described in Section 3.1. Through run-time-generated partial bitstreams, the IMDCT module is dynamically loaded, removed, and vertically repositioned within the sandbox region. By coordinating the reconfiguration with the software application, these changes can take place while other phases of the decoding process continue in software. Due to the long lines' sparse connection points, the module is restricted to nine positions within this sandbox, occurring at intervals of six CLBs. Streams are correctly decoded with the module absent or in any of the positions.

The IMDCT module utilizes two BRAMs and one 18x18 multiplier cell. Four distinct vertical alignments of these

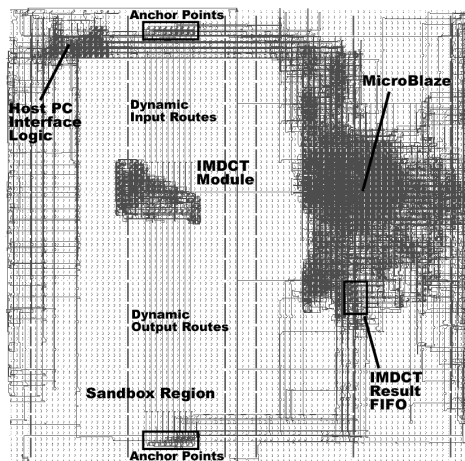


Fig. 5. MP3 Decoder Implementation.

cells can occur within the module, depending on its placement. To address alignment, the module is implemented and stored for all four possibilities at build time. When generating a partial bitfile for a particular vertical position, the run-time tools draw from the appropriate implementation. Note that, due to the six-CLB relocation restriction, only two distinct cell alignments occur in this design.

Dynamic route timing is managed with a simple, conservative approach. As Figure 4 shows, every dynamic net is “bookended” by registers on both end points. By establishing at design time that the worst-case dynamic route delay is less than one clock period, no timing consideration is required at run-time. The mainstream tools implement and verify timing for routes outside the bookend registers.

A data-push protocol accommodates the two-cycle latency introduced by the bookend registers without loss of throughput. The hardware and software interfaces guarantee that the receiver can always accommodate the number of data items to be transferred. This guarantee eliminates the need for handshaking signals from the receiver, allowing either sender to push one 32-bit sample per clock cycle.

5. CONCLUSIONS

Members of the reconfigurable computing research community need to build upon each other’s work by defining layers with agreed-upon interfaces and insulating applications from architectures. Software has inspired high-level abstractions such as extending operating systems to manage FPGA configuration, but the low-level details are often left as a JBits exercise. Careful partitioning is required between costly algorithms used to create efficient module implementations and run-time tasks. Efficient, algorithm-tailored module communication is as important as optimized modules to reap the benefits of configurable computing. The

Wires On Demand project acknowledges this by focusing on flexible, rapid and efficient module composition.

6. ACKNOWLEDGEMENTS

This work is supported by the United States Air Force under Contract Number FA8651-06-C-0126. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force.

7. REFERENCES

- [1] A. Pinto, L. Carloni, and A. Sangiovanni-Vincentelli, “Constraint-driven communication synthesis,” in *Proceedings of the Design Automation Conference 2002 (DAC’02)*, June 2002.
- [2] [Online]. Available: <http://www.sonicsinc.com>
- [3] C. Carruthers, B. Fawcett, C. Patterson, and B. Wilkie, “The XC6200: A microprocessor-oriented FPGA,” in *Proceedings of the Canadian Workshop on Field-Programmable Devices*, May 1996, pp. 91–96.
- [4] S. A. Guccione, D. Levi, and P. Sundararajan, “JBits: A Java-based interface for reconfigurable computing,” in *Second Annual Conference on Military and Aerospace Applications of Programmable Devices and Technologies (MAPLD)*, September 1999.
- [5] S. A. Guccione and D. Levi, “Run-time parameterizable cores,” in *Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications (FPL)*, August 1999, pp. 215–222.
- [6] Xilinx Inc., *XAPP290: Two flows for partial reconfiguration: Module based or difference based*, September 2004. [Online]. Available: <http://www.xilinx.com/bvdocs/appnotes/xapp290.pdf>
- [7] M. Jasiunas, “Combined run-time area allocation and long line re-routing for reconfigurable computing,” in *IEEE International Conference on Field-Programmable Technology*, 2003, pp. 407–410.
- [8] H. Walder, C. Steiger, and M. Platzner, “Fast online task placement on FPGAs: free space partitioning and 2D-hashing,” in *International Parallel and Distributed Processing Symposium*, 2003.
- [9] J. Greene, V. Roychowdhury, S. Kaptanoglu, and A. E. Gamal, “Segmented channel routing,” in *DAC ’90: Proceedings of the 27th ACM/IEEE Conference on Design Automation*. New York, NY, USA: ACM Press, 1990, pp. 567–572.