

HIGH-LEVEL SPECIFICATION OF RUNTIME RECONFIGURABLE DESIGNS

Stephen D. Craven and Peter M. Athanas

Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
email: scraven@vt.edu, athanas@vt.edu

ABSTRACT

“C to Gates” compilers for FPGAs have been a topic of investigation for nearly two decades. Some of these endeavors have reached a point of viability. Impulse C, for example, enables an application developer to describe hardware using a large subset of standard C. While the Impulse C simulation and implementation tools provide an excellent high-level development environment for FPGA applications, no provisions exist for describing dynamic hardware. Through the addition of new functions and slight modifications to the behavior of the existing tools, the Impulse C language becomes a powerful development framework for dynamic reconfiguration of FPGA hardware. This modified language will constitute the input to an automated implementation flow.

Keywords: Field programmable gate arrays, development environment, partial reconfiguration

1. INTRODUCTION

FPGAs may be reprogrammed during operation — an ability known as dynamic reconfiguration or runtime reconfiguration. In some cases, sections of the FPGA can remain operational while only a portion of the device is reconfigured. This partial reconfigurability has been the subject of much research. By customizing part of a circuit to the problem at hand, significant performance benefits have been observed for encryption and pattern recognition. By replacing idle logic with new functionality, partial reconfiguration provides many of the benefits of a larger device. This virtual hardware approach may be used to fit a large design into a small part [1] or extend functionality to a deployed device [2].

In spite of the great potential that partial reconfiguration has repeatedly demonstrated, it has remained in the realm of research because of tool limitations. The tools provided by Xilinx, the largest FPGA vendor, as a patch to their standard ISE development suite, require the designer to perform many additional low-level steps [3]. Special connection points must be manually inserted and placed between

the static and dynamic regions. If internal configuration control is required, the designer must interface with the FPGA’s Internal Configuration Access Port (ICAP), which generally requires some knowledge of the lower level configuration details of the device.

Researchers have attempted to automate many of the time consuming and error prone low-level work. The JBits project provides a Java API into the FPGA’s configuration file, or bitstream [4]. Virginia Tech’s Janus project leveraged JHDL, a Java hardware description language, to describe partially reconfigurable designs [5]. Both of these projects, however, require that the design be described at a level lower than the RTL to which designers are accustomed.

In an attempt to increase the designer’s productivity by raising the level of abstraction, much research has gone into high-level synthesis techniques. The time-consuming RTL design and debugging can be avoided by describing a hardware design in a High Level Language (HLL) and directly translating the description to hardware.

Recognizing the benefits of high level synthesis for dynamically reconfigurable hardware, the Synthesis and Partitioning for Adaptive Reconfigurable Computing Systems (SPARCS) project permits an application to be described at a behavioral level in VHDL [6]. Another project, RT-C, leverages an augmented version of the C-to-gates language Handel-C to design dynamically reconfigurable hardware [7]. Both of these projects suffer severe limitations. SPARCS requires a macro library and does not permit partial reconfiguration. RT-C requires a host processor, manual translation steps, and JBits, limiting designs to older FPGA architectures.

Addressing the limitations of previous work, this paper presents extensions to the Impulse C language [8], commercially available from Impulse Accelerated Technologies, permitting the description and simulation of dynamic partial reconfiguration. Combined with an automated implementation flow and configuration controller generator, these language extensions enable the design, simulation, and implementation of dynamically self-modifying hardware all from a HLL description. This paper focuses on the language ex-

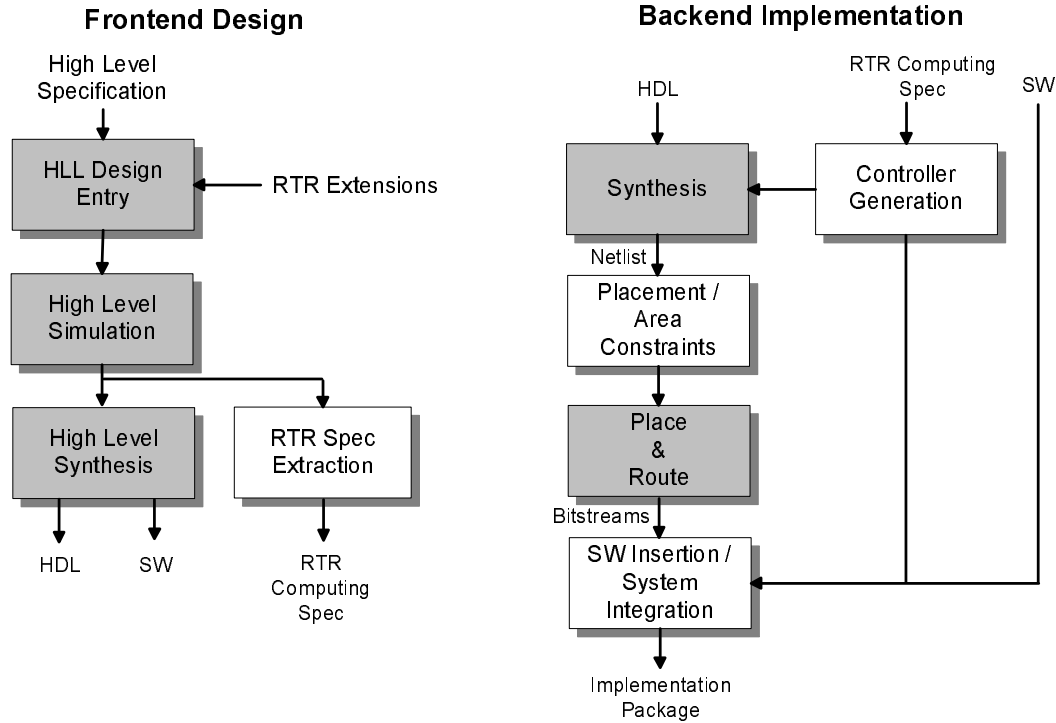


Fig. 1. Design and implementation flow

tensions while a previous paper provides an overview of the design and implementation flow [9].

2. HIGH-LEVEL DEVELOPMENT APPROACH

Complementing the Impulse C extensions is a complete design and implementation flow. The frontend design flow in Figure 1, abstracts the lower level configuration details from the designer. After design entry and high-level simulation of the design, the Impulse C compilation tool produces an HDL description of the design while scripts created for this project parse the Impulse C code to extract the Run-Time Reconfigurable (RTR) specifications, including information about configuration control and connections between the static and dynamic regions.

The architecture-specific backend implementation flow accepts the HDL, embedded software, and RTR Computing Specification. A configuration controller specific to the implementation platform is constructed. The results of synthesis are used to automate the creation of placement and area constraints for the reconfigurable modules. Finally, a system integration step combines the partial bitstreams, software, and initial bitstream into a package conducive to implementation on the specified platform (System ACE file, binary memory image plus initial bitstream, etc.).

3. IMPULSE C DESIGN DESCRIPTION

Impulse C is based on the Communicating Sequential Processes (CSP) model of computation. The application is divided into separate modules, called processes, which share no state. Communication occurs through blocking reads and writes to unidirectional point-to-point channels, called streams, constructed from FIFOs. Synchronization is achieved through the blocking that occurs during communication.

A simple application is shown in Figure 2 consisting of three processes connecting to form a datapath. Each Impulse C process can be implemented in hardware or software.

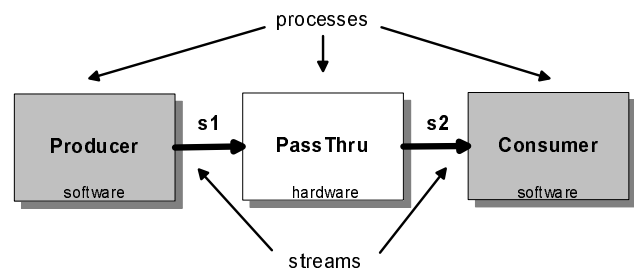


Fig. 2. CSP application description

Impulse C permits hardware processes to be described using a large subset of ANSI C, augmented with functions

for communication and system creation. As these functions are included in a C library, Impulse C can be used with any standard C development environment. Each process in Impulse C is modeled by a separate C function. The arguments to these functions are special stream data types that implement the FIFO connections between processes. A distinct configuration function instantiates instances of a process and connects the processes together. Impulse C processes are implemented in software unless marked for hardware implementation within the configuration function. Figure 3 presents the configuration function for the application from Figure 2.

```
void config>HelloWorld(void)
{
  co_stream s1, s2;
  co_process producer, consumer, dotext;

  // Create streams NAME, SIZE, DEPTH
  s1=co_stream_create("Stream1", 8, 2);
  s2=co_stream_create("Stream2", 8, 2);

  // Instantiate processes
  // Specify connections
  producer=co_process_create("Producer",
    (co_function) Producer, 1, s1);
  dotext=co_process_create("DoText", (co_function)
    DoText, 2, s1, s2);
  consumer=co_process_create("Consumer",
    (co_function) Consumer, 1, s2);

  // Mark process for hardware implementation
  co_process_config(dotext, co_loc, "PE0");
}
```

Fig. 3. Configuration function

4. IMPULSE C EXTENSIONS

Impulse C has no notion of dynamic hardware. Through the addition of a new data type and two new functions, partially reconfigurable designs can be developed and simulated from a HLL. The extended language is referred to as DR Impulse C.

To describe partially reconfigurable applications in DR Impulse C, the programmer defines sets of mutually exclusive Impulse C processes. Only a single process within each set may be resident in hardware at one time. At runtime, a configuration controller process can reconfigure the hardware, swapping one process in the reconfigurable set for another. New functions create these reconfiguration sets (`co_reconfig_create`) and select a new dynamic process to execute in hardware (`co_architecture_config`).

In simulations, each Impulse C process is spun off as a separate thread. Stream communication is modeled by reads and writes into circular buffers protected by semaphores.

Modifications have been made to the Impulse C simulation library, enabling the designer to model and simulate dynamic reconfiguration from a standard C development environment such as Microsoft's Visual Studio. Reconfiguration is simulated by cleanly killing the current executing thread and spawning a new thread of the active Impulse C process.

Figure 4 presents a simple DR Impulse C application. A producer process streams data to one of two processes, the specific process being selected at runtime.

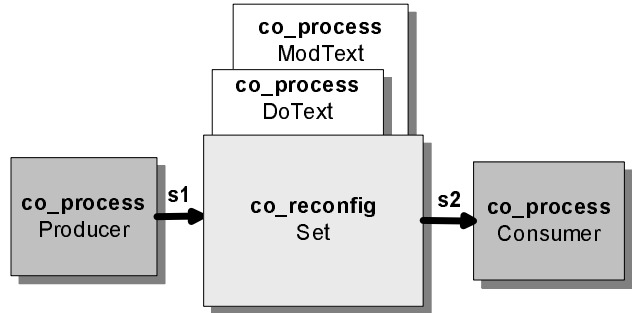


Fig. 4. Set of mutually exclusive processes

The configuration function for the example in Figure 4 is shown in Figure 5. A DR Impulse C configuration function closely matches standard Impulse C. Each process, including the dynamically reconfigurable processes, is instantiated as before. All processes sharing the same reconfigurable set must have the same I/O connections.

A new data type, `co_reconfig`, declares sets of mutually exclusive processes. The function call `co_reconfig_create` places Impulse C processes within a set.

Reconfiguration is controlled by a separate Impulse C process implemented in software running on an embedded MicroBlaze or PowerPC processor. Reconfiguration is initiated by the function `co_architecture_reconfig`. For simulations, this function stops and starts threads described by the Impulse C processes; however, when implemented in hardware this function interfaces with the ICAP found on Xilinx Virtex FPGAs. The partial configuration bitstream that implements the desired Impulse C process is pulled from an external memory and pushed into the ICAP to perform the dynamic reconfiguration.

The following command will stop the currently running Impulse C process in the Figure 5 example and replace it with the process `ModText`.

```
co_architecture_reconfig("test_reconfig_set",
  "ModText");
```

5. PROJECT STATUS

Modifications to the standard Impulse C simulation library to describe partial reconfiguration are complete. Simulation

```

void config>HelloWorldReconfig(void)
{
    co_reconfig Set;
    co_stream s1, s2;
    co_process Producer, Consumer, DoText, ModText;

    // Create streams NAME, SIZE, DEPTH
    s1=co_stream_create("Stream1", 8, 2);
    s2=co_stream_create("Stream2", 8, 2);

    // Instantiate processes
    // Specify connections
    Producer = co_process_create("Producer",
        (co_function) Producer, 1, s1);
    Consumer = co_process_create("Consumer",
        (co_function) Consumer, 1, s2);
    // Note that reconfigurable processes
    // have identical connections.
    DoText = co_process_create("DoText",
        (co_function) DoText, 2, s1, s2);
    ModText = co_process_create("ModText",
        (co_function) ModText, 2, s1, s2);

    // Place reconfigurable modules inside
    // a set. The first process listed
    // is the default.
    Set = co_reconfig_create("test reconfig set",
        2, DoText, ModText);
}

```

Fig. 5. DR Impulse C Configuration function

stress testing of partially reconfiguring designs, identified memory leaks that were rectified. Currently, the modified simulation library is limited to Windows computers because of the difference in thread implementations between Windows and Linux.

Work is progressing on the integration of the automated implementation flow. The Xilinx Early Access Partial Reconfiguration flow has been automated via makefiles. To permit self-reconfiguration, MicroBlaze ICAP drivers for the Xilinx Virtex-4 devices have been created and tested.

6. CONCLUSIONS

Modifications have been made to a commercial high level synthesis tool to permit the description and simulation of dynamic partial reconfiguration in support of the creation of a high level development environment for partially reconfigurable designs. This DR Impulse C language enables a designer to develop dynamic hardware from within any C development suite on Windows machines. Work is progressing on the integration of DR Impulse with an automated implementation flow. The completed project will significantly reduce the required skill set and development time for dynamic hardware.

7. ACKNOWLEDGMENT

The authors gratefully acknowledge the input and assistance of Impulse Accelerated Technologies, Inc.

8. REFERENCES

- [1] K. Chia, H.J. Kim, S. Lansing, W.H. Mangione-Smith, and J. Villasenor. High-performance automatic target recognition through data-specific VLSI. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):364–371, 1998.
- [2] John W. Lockwood, Naji Naufel, Jon S. Turner, and David E. Taylor. Reprogrammable network packet processing on the field programmable port extender (FPX). In *FPGA*, pages 87–93, 2001.
- [3] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford. Uenhanced architectures, design methodologies and cad tools for dynamic reconfiguration of xilinx fpgas. In *16th International Workshop on Field Programmable Logic and Applications, FPL*, pages 12–17, 2006.
- [4] S. Guccione, D. Levi, and P. Sundararajan. JBits: Java based interface for reconfigurable computing. In *2nd Annual Military and Aerospace Applications of Programmable Logic Devices Conference*, Laurel, Maryland, 1999.
- [5] David I. Lehn, Rhett D. Hudson, and Peter M. Athanas. Framework for architecture-independent run-time reconfigurable applications. volume 4212, pages 162–172. SPIE, 2000.
- [6] Iyad Ouais, Sriram Govindarajan, Vinoo Srinivasan, Meenakshi Kaul, and Ranga Vemuri. An integrated partitioning and synthesis system for dynamically reconfigurable multi-FPGA architectures. In *IPPS/SPDP Workshops*, pages 31–36, 1998.
- [7] T. K. Lee, A. Derbyshire, W. Luk, and P. Y. K. Cheung. High-level language extensions for run-time reconfigurable systems. In *Field-Programmable Technology (FPT). Proceedings. IEEE International Conference on*, pages 144–151, 2003.
- [8] D. Pellerin and S. Thibault. *Practical FPGA Programming in C*. Prentice Hall, Upper Saddle River, N.J., 2005.
- [9] S. Craven and P. Athanas. A high-level development framework for run-time reconfigurable applications. In *International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, Washington, D.C., 2006.