Secure Processing Using Dynamic Partial Reconfiguration

Christopher T. Rathgeb  Gregory D. Peterson
Electrical Engineering and Computer Science
University of Tennessee
Knoxville, TN 37996-2100
[crathgeb, gdp]@utk.edu

ABSTRACT
A popular research topic as of late has been dynamic partial FPGA (Field Programmable Gate Array) reconfiguration. This concept allows on-the-fly reconfiguration of digital systems where only parts of the circuit change, providing application acceleration and allowing static modules to continue processing unaffected by the dynamic elements. Design characteristics which benefit from this progressive approach include increased system flexibility, increased performance, and a reduction in circuit complexity. One characteristic receiving limited focus thus far has been the increased security that could result from these changing circuits. This benefit is innate to the design and makes reverse engineering of the system a much more ambitious task. In an effort to further enhance this passive security feature, a new partial reconfiguration technique has been proposed that changes connectivity between generic modules. This extended abstract introduces the method, model, and design flow for this dynamic partial FPGA reconfiguration technique and addresses the security implications of such a design.

Categories and Subject Descriptors
B.1.1 [Control Structures and Microprogramming]: Control Design Styles—Microprogrammed logic arrays
B.5.1 [Register-Transfer-Level Implementation]: Design—arithmetic and logic units, control design, styles
B.6.1 [Logic Design]: Design Styles—cellular arrays and automata, combinational logic, logic arrays, parallel circuits
B.7.1 [Integrated Circuits]: Types and Design Styles—gate arrays, advanced technologies, input/output circuits

General Terms
Design, Experimentation, Performance, Reliability, Security, Verification

Keywords
Partial Reconfiguration, Sandboxes, System Security, Field Programmable Gate Arrays

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, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CSIIRW '09, April 13-15, Oak Ridge, Tennessee, USA
Copyright 2009 ACM 978-1-60558-518-5...$5.00.

1. INTRODUCTION
Field Programmable Gate Arrays (FPGAs) are a class of integrated circuit that are designed to be quickly re-programmable. They are quite popular due to their high speeds and flexibility. Often times FPGAs are used for rapid prototyping of application specific integrated circuits that are to be fabricated. In order to reprogram one, a circuit layout must be converted into a bitstream that details the FPGA functionality by dictating the function and interconnections of primitive elements. This is similar to a C++ program being turned into machine language for a CPU. The majority of FPGA implementations are static and require the system to be completely erased before modifications can be loaded. However, some FPGAs allow for dynamic circuit modification to enable systems that are re-programmable at runtime. These systems are known as partially reconfigurable circuits. They are a very powerful type of circuit that results in heightened security, higher performance, lower circuit complexity, and a flexibility that is similar to software [8]. Other notable benefits include increased function density, reduced power dissipation [6], and a smaller required area for the entire circuit, which results in cost savings. Unfortunately, all of these benefits come at the cost of implementation complexity. The first hurdle is that the implementation requires bitstreams to be pre-synthesized and loaded into memory. Then, a controller needs to be accessed to load the partial bitstreams. The nature of this pre-synthesis opens up all kinds of manual floor-planning requirements. Next, the methodology for implementing a PRC(Partially Reconfigurable Circuit) is quite complicated. It requires the insertion of bus macros between modules, specific synthesis guidelines to generate the partially reconfigured netlist, proper floor-planning and placement of the bus macros, and adherence to specific PRC design rules; all of which require meticulous implementation in order to ensure success [11]. Due to design tool limitations and the lack of design flows, PRCs have, for the most part, remained in the realm of research. Existing tools require designers to perform many extra design steps to manipulate primitive routing and function elements. Many of these steps such as designing special connection points that must be manually inserted between static and dynamic regions [2], significantly increase the complexity of the implementation. Because of this, many designers have not had the proper guidance and experience to make PRCs practical [10]. In order to simplify this complexity and gain additional design security benefits, we developed a method that implements an array of generic modules known as sandboxes. Rather than modifying the bitstream for circuit reconfiguration, the connections between different sandboxes are modified, which essentially build the circuit at run time. This concept of reconfiguration at a more abstract level allows for security levels far higher than were previously possible. Areas where this
technique and heightened security may be applicable include data
cryptography, pattern recognition, power controlling, time
division multiple access applications, and data manipulation that
requires extreme security. This paper details the proposed
approach from the smallest element, the sandbox, up through the
design flow. An explanation of which methods can be
implemented to dynamically create fast, reliable, and secure
hardware functions will also be discussed.

2. THE SANDBOX APPROACH
In this system the sandbox is the smallest element of a larger
design. The sandbox is a generic isolated module that can
perform any one function, of a set of simple predefined functions.
A simple sandbox that performs some basic functions can be seen
in Figure 1 below:

Figure 1. Simple Sandbox That Performs Basic Arithmetic
Operations

A sandbox such as the one above might be capable of executing
different operations such as add, subtract, shift left, shift right,
and, nand, or, xor, nor, or more complex operations.
Combining lots of these basic sandboxes together, with different
operations on each, allows for the creation of a near limitless
number of circuits without modifying the bitstream. The idea is to
especially design an abstraction layer to be placed over the
FPGA, specifically for reconfiguration, that allows for circuit
modifications without modifying the actual hardware. Such a
system would allow for mobile hardware processes [13]. A
connection description could be passed to the FPGA along with
initial parameters. After the function is completed it can destroy
existing connections. The end result is a method of true virtual
hardware.
The sandbox model relies on a hierarchical design to allow easy
controllability. Connection bookkeeping is handled by a sandbox
controller that has many of these sandboxes attached to it. The
controller contains a port look up table and memory to store state
information for the hardware process.

Figure 2. A Sandbox Controller with Block RAMs

A hardware process would be implemented on the FPGA through
the following flow. First a computer system software process
would send a request to the daemon running on the machine,
which controls access to the FPGA. The daemon sends the
hardware configuration plus required inputs to the logic board.
Next the FPGA takes the hardware configuration, decodes it, and
sends it to the OTF (On-The-Fly) Layout Planner. This module
consults the resource manager to determine which sandbox
controllers are currently busy. From here the layout is determined
and the inputs and local layout configurations are passed to the
sandbox controllers that are needed for the process. The sandbox
controllers set their connections and execute the task based on the
inputs. A response is sent back to the daemon and relayed to the
requesting software process. The connections on the system are
then released and ready for the next operation.

Figure 3. The Complete Sandbox Model System

The expected benefits from this design include the benefits of
dynamic hardware systems with faster reconfiguration loading
due to no hardware being changed, and increased security which
will be discussed in the next section.

3. SECURITY
The driving motive that led to this new run-time reconfiguration
implementation was the need for secure systems. This section
will discuss the passive and active security features of this
implementation. The most notable passive security feature is that
any individual or group attempting to tamper with the circuit to
obtain valuable information would not see the true circuit. Other
security features include encrypted bitstreams, sensitivity control,
and quick connection flushing.

One of the major security features that the sandbox approach takes
advantage of is the anti-tamper protection. Any individual who
has the resources to extract the bitstream would have to decrypt it
—as FPGAs often allow their bitstreams to have triple DES
protection—and then reverse engineer it. Assuming they were
able to reverse engineer the circuit they would not have any
valuable information. The only thing that would be visible would
be the unconnected sandboxes. Essentially, there is no hardware
signature. This is due to the fact that the design is created and
destroyed at run-time. The hardware process is created through
the configuration, it executes, and then it dies through resetting
the sandbox interconnections.
Another prominent security feature of partially reconfigurable
FPGAs is that they contain SRAM controlled configurations.
This means that the pre-synthesized bitstreams are held in volatile
memory waiting to be swapped in at a nanosecond's notice. The
benefit of this is that when the system is powered off, the circuit
function disappears, which makes tampering much more difficult.
Moreover, the functions built out of tiled sandboxes require
dynamic instructions which are also held in volatile storage,
further protecting the design from tampering.
Next, several active security features will be added to the system to further increase the overall safety. The connection configuration and data that is passed to the FPGA can be encrypted before it is sent and then decrypted when received on the FPGA. Also, depending on the size and security requirements of the design, the FPGA can be split into different self-contained areas each of which contain different levels of security. This will allow multiple users, with different security requirements, to use the PRC elements, in a secure seamless manner. The regions in which the processes are run can be separated based on classification level or other measures. On very large systems or systems that require even further safety, these different regions may be implemented on completely different FPGA boards. An additional security feature that this approach makes easy to implement is a circuit self-destruct feature. A button or command could be added that instantly flushes all sandbox connections. This would destroy the circuit leaving it empty for anyone who tries to tamper with it later. One final security advantage to point out is that by design, sandboxes provide protection against many common attacks such as buffer overflows. All of these different passive and active security features make the sandbox model ideal for situations where security is of the utmost importance.

4. FUTURE WORK

A simple prototype is currently being developed to act as a proof of concept. It will implement the generic sandbox shown in Figure 1. After the proof of concept is completed, different sandboxes will be explored to create operational flexibility. Next, we will investigate a framework to maximize performance, reliability, security, and re-usability. A good tool to test the performance and reliability, is to run comparative analysis against static circuits with the same functionality. A verification metric would be suitable for future work as well to ensure reliability. As it stands, verifying a circuit model that constantly changes could be quite challenging. Creating a built in self-verification for dynamically loaded circuits would be a very cumbersome task, but one that needs to be explored.

5. CONCLUSIONS

The flexibility and performance of FPGAs have made them ideal for the rapid prototyping of custom ASIC (application specific integrated circuit) designs. One of the most intriguing research areas has been the practical application of dynamic partial reconfigurable FPGAs. These dynamic configurations have many benefits and limitations. They provide better performance and flexibility while taking up less area, which is ideal for ASICs. However, their limitations include lack of design tools and intensive manual floor planning. This is also assuming that the board chosen for design can even be dynamically reconfigured. This paper proposed a new solution for run-time reconfiguration limitations that enhances features such as security and reconfiguration load times. An added benefit is that such a system is generic enough that it can be implemented on FPGAs that have not been designed for dynamic reconfiguration. Security is one of the motivators behind this new implementation, and as such, notable security enhancements include no hardware signatures, bitstreams that do not reveal any sensitive information about the circuit, and the ability to quickly flush all circuit connections effectively destroying the circuit.

6. REFERENCES


