Simulation-Based Functional Verification of Dynamically Reconfigurable Systems

LINGKAN GONG and OLIVER DIESSEL, University of New South Wales

Dynamically reconfigurable systems (DRS) implemented using field-programmable gate arrays (FPGAs) allow hardware logic to be partially reconfigured while the rest of the design continues to operate. By mapping multiple reconfigurable hardware modules to the same physical region of an FPGA, such systems are able to time-multiplex their modules at runtime and adapt themselves to changing execution requirements. This architectural flexibility introduces challenges for verifying system functionality. New simulation approaches are required to extend traditional simulation techniques to assist designers in testing and debugging the time-varying behavior of DRS. This article summarizes our previous work on ReSim, the first tool to allow cycle-accurate yet physically independent simulation of a DRS reconfiguring both its logic and state. Furthermore, ReSim-based simulation does not require changing the design for simulation purposes and thereby verifies the implementation-ready design instead of a variation of the design. We discuss the conflicting requirements of simulation accuracy and verification productivity in verifying DRS designs and describe our approach to resolve this challenge. Through a range of case studies, we demonstrate that ReSim assists designers in detecting fabric-independent bugs of DRS designs and helps to achieve verification closure of DRS design projects.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Verification*; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems

General Terms: Verification

Additional Key Words and Phrases: FPGA, dynamically reconfigurable systems, verification

ACM Reference Format:
Lingkan Gong and Oliver Diessel. 2014. Simulation-based functional verification of dynamically reconfigurable systems. ACM Trans. Embedd. Comput. Syst. 13, 4, Article 97 (February 2014), 23 pages.
DOI: http://dx.doi.org/10.1145/2560042

1. INTRODUCTION

Due to the exponential increase in hardware design costs and risks, the electronics industry has begun shifting towards the use of reconfigurable devices such as field-programmable gate arrays (FPGAs) as mainstream computing platforms. An FPGA is a special type of integrated circuit that can be repeatedly programmed to perform an arbitrary logic function. Traditionally, an FPGA device is programmed when the system is powered up. Using dynamic partial reconfiguration (DPR), the programming and reprogramming of the FPGA can occur at system runtime and can be controlled by the system itself (i.e., self-reconfiguration). In particular, dynamically reconfigurable systems (DRS) implemented on FPGAs can reprogram/reconfigure part of their circuits at runtime to adapt to changing execution requirements [Xilinx 2010c; Altera 2010].

The authors thank Xilinx for their generous donations.
Authors' addresses: L. Gong (corresponding author) and O. Diessel, School of Computer Science and Engineering, University of New South Wales; corresponding author's email: lingkan.gong@unswalumni.com.

DOI: http://dx.doi.org/10.1145/2560042

By mapping multiple reconfigurable hardware modules (RM) to the same physical reconfigurable region (RR) of the FPGA, the system can time-multiplex its submodules at runtime, and the design density of a DRS is increased [Wirthlin and Hutchings 1998].

Designing effective DRS poses several challenging research problems, and functionally verifying the correctness of such systems is one of them. Register transfer level (RTL) simulation is the most common method of verifying hardware (either ASIC- or FPGA-based) design functionality. Since DPR is the process of reprogramming the configuration memory of the FPGA fabric with a configuration bitstream, cycle-accurate simulation of the reconfiguration process involves modeling the configuration memory and the configuration bitstream. However, the organization of the FPGA fabric, including the configuration memory and the configuration bitstream, is proprietary to the FPGA vendors. Furthermore, even if the simulation models of the FPGA fabric were available, the simulation would be performed at too low a level of detail and would significantly reduce verification productivity. As a result, new simulation approaches need to extend traditional simulation techniques to assist designers in testing and debugging DRS designs while part of the design is undergoing reconfiguration.

ReSim [Gong and Diessel 2011a, 2011b, 2012; Gong et al. 2013] is a reusable simulation library for supporting cycle-accurate simulation of modular reconfigurable DRS designs. The core idea of ReSim is to use a simulation-only layer to emulate the physical fabric of FPGAs so as to assist designers[1] in testing/debugging/verifying the user design. The following compares ReSim with previous RTL simulation tools for DRS designs (e.g., [Luk et al. 1997; Robertson and Irvine 2004; Hansen et al. 2013]).

—ReSim-based simulation is cycle accurate. Using the simulation-only layer, the reconfiguration process, including the transfer of configuration bitstreams and the subsequent module swapping, can be emulated, except that simulation-only bitstreams instead of real bitstreams are used and interpreted [Gong and Diessel 2011b, 2012]. Although the simulation-only layer is not a completely accurate model of the FPGA device, the user design, which is the focus of verification, is simulated in the desired cycle-accurate manner.
—ReSim-based simulation is physically independent. Instead of modeling the configuration bits of the FPGA fabric, the simulation-only layer only utilizes user design parameters (e.g., a list of interfacing signals that crosses RR boundaries, the affiliation of RMs and RRs, and the target FPGA family) to model reconfiguration [Gong and Diessel 2011b]. Therefore, the productivity of verifying a DRS design is not compromised for the level of simulation accuracy. One significant extra benefit of physical independence is that FPGA vendors do not need to disclose the details of the FPGA device in order to support simulation of partial reconfiguration.
—The simulated design is implementation ready. Since the simulation-only layer emulates the target FPGA, the user design does not need to be changed for simulation purposes [Gong and Diessel 2011b, 2012]. As a result, a user design bug exposed by ReSim-based simulation reveals an actual bug in the implemented design, and the design as implemented instead of some variation of it is simulated and verified.

Our research project focuses on the effective modeling [Gong and Diessel 2011a], simulation [Gong and Diessel 2011b, 2012], and functional verification of DRS designs, and we use both simple in-house designs as well as cutting-edge real-world applications [Gong et al. 2013] as our case studies. This article achieves the following.

—It summarizes our research into functional verification of DRS designs.

[1]Since design engineers also spend significant time in testing and debugging, this document does not explicitly distinguish between design engineers and verification engineers but refers to both as designers.

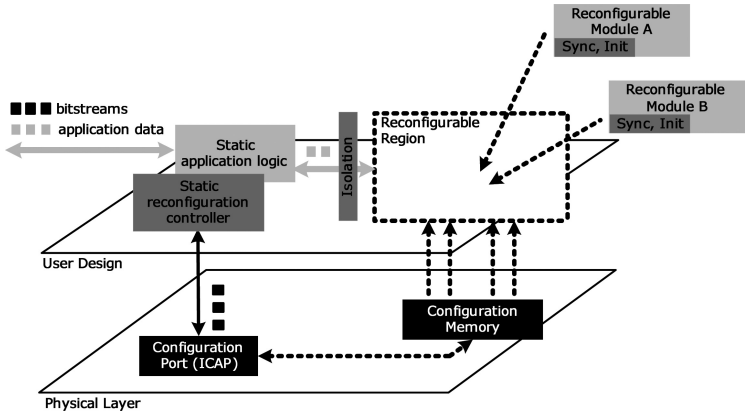

Fig. 1. Conceptual diagram of a DRS design.

—It provides a comprehensive set of case studies and presents a summary of the lessons learned.
—It presents a number of improvements to the ReSim library.
—It discusses possible future research directions.

The rest of this article is organized as follows. Section 2 summarizes verification challenges identified in our previous publications and illustrates the way ReSim resolves such challenges. We back up our claims in Section 2 with a comprehensive set of case studies in Section 3. Our case studies cover a range of DRS design styles, and we focus on real bugs detected in the development process. Section 4 outlines related work and forecasts future directions for the functional verification of DRS designs. Section 5 concludes.

2. RESIM-BASED SIMULATION OF DRS DESIGNS

2.1. Challenges in Verifying DRS Designs

Compared with traditional static FPGA designs, DPR introduces additional flexibility for system designers but also introduces challenges to the verification of design functionality. Figure 1 illustrates the design under test (DUT) to be simulated and tested. Before reconfiguration, the static logic should properly synchronize with the old RM to pause the ongoing computation. During reconfiguration, a reconfiguration controller transfers the configuration bitstream (depicted by a sequence of small black squares alongside communication links in the figure) of the new RM to the configuration port (e.g., Internal Configuration Access Port, ICAP).[2] During this period, the static part must isolate the RR to avoid the propagation of spurious outputs from partially configured RMs. After reconfiguration, the incoming RM needs to be initialized to a known state before it starts execution. We refer to the logic that performs the synchronization, isolation, and initialization of RMs as well as the bitstream transfer logic as the *reconfiguration machinery* (moderately shaded blocks in the figure).

Our previous work identified the potential bugs that could be introduced to DRS designs before, during, and after reconfiguration [Gong and Diessel 2011a]. Since traditional RTL simulation does not model characteristic features of DPR (i.e., module swapping and its triggering condition, bitstream traffic, bitstream content, spurious RM outputs, and undefined RM state); it only offers limited assistance in detecting

[2]Bitstream transfer can be performed by either an internal or an external reconfiguration controller.

DPR-related bugs [Gong and Diessel 2011b, 2012]. However, it is challenging to accurately model these characteristic features. This can be explained by considering the two conceptual layers identified in a typical DRS design (see Figure 1).

—*User Design Layer.* The user design comprises all user-defined modules. These include the application logic (lightly shaded blocks) performing the required processing tasks of the application, as well as the reconfiguration machinery (moderately shaded blocks) that manages the reconfiguration process.
—*Physical Layer.* The physical layer (darkly shaded blocks) represents the FPGA device which contains the logic/routing resources comprising the fabric of the device, the configuration memory controlling the function and interconnection of the fabric, the configuration distribution network commencing with the configuration port (e.g., ICAP, SelectMap), and the configuration bitstream used to program/reprogram the device. The implementation of the physical layer is proprietary to the FPGA vendor.

In traditional FPGA-based hardware designs, the physical layer is statically configured and does not interact with the user design. In DRS designs, on the other hand, the components of the FPGA fabric interact with the user design in the process of partial reconfiguration (see the links between the two layers in Figure 1). In particular, bitstreams are transferred by the user design, and the bitstreams subsequently overwrite the configuration memory, thereby changing the functionality of the user design. Therefore, a completely accurate simulation of the reconfiguration process involves modeling the FPGA fabric (i.e., *fabric-accurate simulation*).

As FPGA vendor tools do not provide a simulation model for the FPGA fabric, it is nontrivial for designers to accurately simulate the interactions between the user design and the physical layer. Potential bugs, either arising from the reconfiguration process or from integrating DPR with the rest of the system, cannot therefore be detected until the integrated design is tested on the target device.

On the other hand, even if the simulation model of the FPGA fabric were available, fabric-accurate simulation would include a multitude of unnecessary details for verification. Since the user design would be represented by configuration bits instead of RTL signals, the designer would not be able to focus on the desired user logic and verification productivity could thus be significantly reduced. For the sake of productivity, it is therefore desirable that functional verification is physically independent.

An effective simulation method therefore needs to strike a balance between simulation accuracy and verification productivity. Furthermore, this balance is constrained by the desire for the simulated design to be implementation ready, that is, the captured design should not be changed for simulation purposes.

2.2. The Simulation-Only Layer

The core idea of ReSim is to use a simulation-only layer to emulate the physical fabric of FPGAs so as to achieve the desired balance between accuracy and physical independence. Figure 2 redrafts Figure 1 with the physically dependent blocks (darkly shaded boxes) replaced by their corresponding simulation-only artifacts (open boxes). In particular, the configuration bitstreams are replaced by simulation-only bitstreams (SimB), possible configuration ports are represented by a CP artifact[3], and the part of configuration memory to which each RR is mapped is substituted by an RR artifact.[4]

ReSim uses a *simulation-only bitstream* to model the bitstream traffic. A SimB captures the essence of a real bitstream in that it serves the purpose of and effects the

[3]The CP artifact was called "ICAP artifact" in previous publications.
[4]The RR artifact was called "Extended Portal" in previous publications.

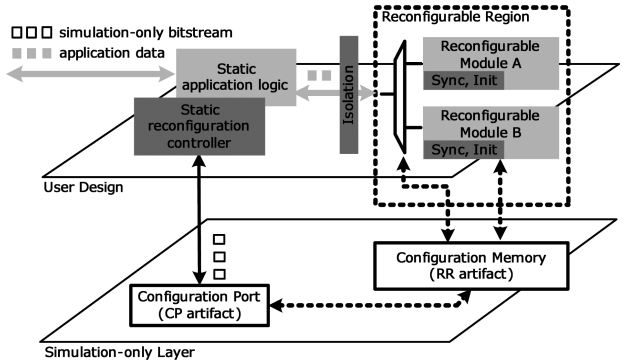

Fig. 2. Using the simulation-only layer.

Table I. An Example of SimB for Configuring a New Module

Entry	SimB	Explanation	Actions Taken
1	0xAA995566	SYNC Word	Start the **DURING Reconfiguration** phase
2	0x20000000	NOP	–
3	0x30002001 0x01020000	Type 1 Write FAR FA=0x01020000	Select the module id=0x02 to be the next active module in reconfigurable region id=0x01
4	0x30008001 0x00000001	Type 1 Write CMD WCFG	
5	0x30004000 0x50000010	Type 2 Write FDRI Size=16	Pad frame: not required Configuration data: 16 words (4 frames)
6	0x5650EEA7 0xF4649889 ... 0xA9B759F9 0x4E438C83	SimB Frame 0 Word 0 SimB Frame 0 Word 1 ... SimB Frame 3 Word 2 SimB Frame 3 Word 3	Module signature; Module state data; Enable/Disable error injection
7	0x30008001 0x0000000D	Type 1 Write CMD DESYNC	End the **DURING Reconfiguration** phase

mediation of module swapping. To improve verification productivity, the size of a SimB is significantly reduced. Table I provides an example of a SimB that configures a new module. Similar to a real bitstream, a SimB starts with a SYNC word (entry 1 in Table I) and ends with a DESYNC command (entry 7 in Table I). However, a SimB differs from a real bitstream in the frame address and the configuration data fields.

Instead of containing frame addresses for the configuration data, as found in a real bitstream, a SimB contains numerical IDs for the module to be configured and the target reconfigurable region. The example DRS design shown in Figure 3 has three RRs with RRID = 0,1,2 and RR0, for example, has 2 RMs with RMID = 0,1. In the example SimB, six consecutive words (entries 3–5 in Table I) request that the RR with ID = 0x1 be reconfigured with a module with ID = 0x2.

Instead of containing configuration bits of RMs, as found in a real bitstream, the configuration data section of a SimB contains simulation-only frames (entry 6 in Table I).

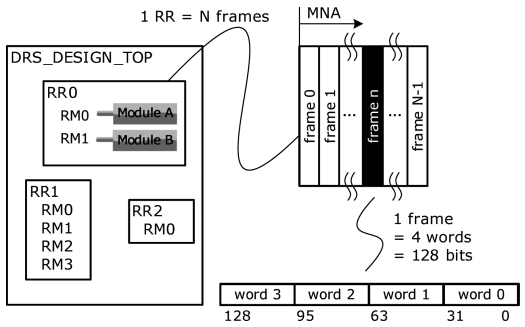

Fig. 3. Configuration memory in ReSim-based simulation.

In particular, each RR is composed of a user-defined number of simulation-only frames and each frame contains four words (see Figure 3). Our previous work [Gong and Diessel 2011b] used simulation-only frames to indicate the start and end of error injection. In Gong and Diessel [2012], words 1–3 of each frame are used to store the value of user-selected RTL signals, which represent the state data of the simulated RM. This article further extends word 0 of each frame with a sequence of signature data that are expected to be kept unchanged and are constantly checked throughout the simulation. If any bit of the signature data is corrupted because of a bitstream retrieval or transfer bug in the design, the signature check would fail during simulation.

Simulation-only bitstreams model two characteristic features of DPR.

—*Bitstream Traffic.* Since ReSim uses a SimB to replace real bitstreams, the bitstream transfer datapath (i.e., the transfer, compression, decryption, and arbitration of bitstreams) is exercised in simulation. Furthermore, the length of a SimB is significantly shorter than a real bitstream and can be adjusted to exercise various scenarios of the bitstream transfer mechanism (e.g., FIFO full/empty).

—*Bitstream Content.* The configuration data of a SimB contain the signature of the RM to be reconfigured. Checking the signature verifies that a correct bitstream is written to the configuration port. Furthermore, the configuration data of a SimB also contain data for state elements (i.e., flip-flops, memory cells). Since ReSim correlates the stored state data with the state of the simulated module, incorrect state data, typically caused by bugs in the design, can propagate to RMs of the simulated user design after state restoration and can therefore be quickly identified in simulation. Recent work [Hansen et al. 2013] uses real bitstreams to simulate the reconfiguration process. However, since the contents of real bitstreams cannot currently be interpreted by simulation models, they only provide limited benefits in terms of detecting bugs.

The *CP artifact* models the configuration port of a DRS design. During simulation, it interacts with the user-defined reconfiguration controller according to the interfacing protocol specified by the device configuration guide (e.g., [Xilinx 2009b, 2010d, 2010e, 2013]). Furthermore, a CP artifact takes a simulation-only bitstream as input [Gong and Diessel 2011b] or returns a readback SimB to the user design [Gong and Diessel 2012]. The library built-in CP artifact models the ICAP port of Xilinx FPGAs. To simulate DRS designs mapped to FPGAs from other vendors, the built-in CP artifact can be derived and extended using the object-oriented programming techniques supported by ReSim [Gong and Diessel 2011b].

The *RR artifact* is the placeholder for RMs in simulation and it models the following characteristic features of DPR.

—*Module Swapping and Triggering Condition.* ReSim-based simulation connects all RMs in parallel like most existing approaches do (e.g., [Luk et al. 1997; Robertson and Irvine 2004]). However, since the selection of the active RM is triggered by the SimB, any runtime-dependent events that delay the bitstream transfer also delay the module swapping. Instead of being modeled using an approximated constant value, the reconfiguration delay is more accurately modeled. Furthermore, since failing to transfer the SimB correctly prevents the new RM from being swapped in, a bug in the bitstream transfer controller or datapath can quickly be identified. Recent work [Hansen et al. 2013] extends our idea to extract reconfiguration timing information from the target FPGAs. Using the extracted timing, the simulation can more accurately model module swapping and its triggering condition.
—*Spurious Outputs and Undefined Initial State.* Since errors are injected into both the static and reconfigurable regions when the SimB is being written to the CP artifact, the isolation and the initialization logic can be exercised and tested in simulation. Failing to isolate the RR correctly can quickly be identified as a consequence of the injected errors propagating to the static region. Failing to initialize the newly configured module correctly can be detected, since the injected errors are not cleared. Compared with DCS, which only drives undefined "x" values to the static region [Robertson and Irvine 2004], our approach injects errors to both the static and the reconfigurable regions. Furthermore, in our approach, the start and end of error injection is also triggered by the SimB, which more accurately models the timing of the error injection operation.

Using a simulation-only layer, modular reconfiguration [Gong and Diessel 2011b] and configuration readback [Gong and Diessel 2012] can be simulated in the desired cycle-accurate manner. The user-defined reconfiguration controller reads/writes SimBs from/to the simulated CP artifact as if it were reading/writing a real configuration port. The RR artifact uses RRIDs/RMIDs instead of frame addresses to drive module swapping and to save and restore state data of simulated RMs.

Use of a simulation-only layer to emulate an FPGA device is analogous to the use of a bus functional model (BFM) to emulate a microprocessor when testing and verifying peripheral logic attached to a microprocessor bus [Xilinx 2010b]. Although the BFM is not a completely accurate representation of the microprocessor, it is accurate enough to capture the interactions between the microprocessor and the bus peripheral to be tested. Furthermore, since the BFM approach abstracts away the internal behaviors of a processor such as pipelines, BFM-based simulation is more productive than simulating a completely accurate processor model. Similarly, the simulation-only layer emulates the FPGA device and captures the interactions between the physical device and the user design to be tested. Furthermore, the simulation-only layer abstracts away the details of the FPGA fabric and significantly improves the verification productivity compared with fabric-accurate simulation.

2.3. Extended ReChannel

Use of the simulation-only layer can also be extended to transaction-level modeling (TLM). Our previous work proposed a top-down modeling methodology using ReChannel [Raabe et al. 2008], a SystemC-based library that models DPR to verify DRS designs at a high level. In this article, we extend the ReChannel library with new classes and data structures to model the bitstream traffic and triggering condition of DPR. Since TLM modeling abstracts away the signal-level details of individual modules, these extensions do not consider signal-level reconfiguration activities, such as spurious RM outputs and undefined initial RM state. With these new extensions, TLM can be used in the following example scenarios.

—For design space exploration or performance evaluation purposes, the bitstream traffic is simulated to decide whether to use a dedicated or a shared datapath for bitstream transfer.
—System-wide integration bugs relating to a sequence of reconfiguration events can be detected in the TLM model. For example, deadlocks caused by circular waiting of multiple swapped-out modules can be detected by accurately modeling the timing and triggering of module swapping. As another example, illegal requests to reconfigure a module still undergoing reconfiguration can be detected by more accurately modeling the reconfiguration delay.
—In order to test and debug any embedded software that could be in control of the reconfiguration process, the TLM model should be functionally equivalent to the implemented design from the programmer's point of view. The designer therefore needs to simulate the bitstream traffic byte by byte if the software buffers and transfers bitstream data byte by byte. Software bugs, such as setting an incorrect bitstream transfer size, could be detected in simulation.

2.4. Capabilities and Limitations

As the simulation-only layer abstracts away the details of the FPGA fabric, it can be regarded as a *fabric-independent* FPGA device, and simulation can be thought of as functionally verifying the *user design layer* of a DRS on such a fabric-independent FPGA. Simulation using the simulation-only layer assists designers in detecting fabric-independent bugs of a DRS design. These bugs include, but are not limited to the following:

—system integration bugs and software bugs described in Section 2.3;
—bugs in synchronizing the static region and RMs before reconfiguration, such as, failing to block a reconfiguration request until the RM is idle;
—bugs in the bitstream transfer logic, such as cycle mismatches, FIFO overflow, errors in driving the ICAP signals;
—bugs in isolating the region undergoing reconfiguration, such as isolating the RR too early or too late;
—bugs in initializing the newly configured module, such as resetting the RM before it is completely reconfigured, loading the RM pipelines with incorrect data.

However, the simulation-only layer is not exactly the same as the FPGA fabric. Table II lists the differences between a Virtex-5 FPGA, as an example of a target device, and our simulation-only layer, representing a fabric-independent FPGA. The mismatches between the two can lead to bugs that remain undetected using the simulation-only layer (i.e., False Negative bugs) and bugs that are incorrectly reported using the simulation-only layer (i.e., False Positive bugs). These bugs, which could be categorized as being fabric-dependent, include the following:

—errors in the bitstream itself (e.g., setting an incorrect frame address in the bitstream, single or multiple bit flips in the bitstream);
—errors in interpreting the content of the bitstream (e.g., accessing an incorrect state bit in a bitstream).

In theory, fabric-dependent bugs would not be introduced to modular reconfigurable DRS designs created by vendor tools, and ReSim can thereby provide assistance in verifying modular reconfigurable DRS designs. On the other hand, if a system is designed to directly modify or generate bitstreams at runtime, ReSim can only offer limited help to test and verify it. Furthermore, ReSim does not nor does it aim to provide assistance in verifying implementation-related bugs, such as the following.

Table II. Differences between the Virtex-5 FPGA Fabric and the Simulation-Only Layer

	Virtex-5 FPGA	Simulation-only Layer
Configuration memory	- Frame Address is composed of RA, CA and MNA - A frame has 41 words - Frame organization is not open	- Frame Address is composed of RRID, RMID, and MNA - A frame has 4 words - Word 0: signature data - Words 1–3: state data
Bitstream	- Normal Frame Address - Pad words and frames - Size is determined by the resources used	- Modified frame address - No pad word or frame - Reduced size
Logic allocation	- State bits are sparsely distributed in a frame	- State bits are grouped and stored contiguously - Has a bit-width field

—timing violation errors in the placed and routed design, and
—possible short or open circuits, if any, caused by partial reconfiguration [Beckhoff et al. 2010].

3. CASE STUDIES

We demonstrate the value of ReSim and ReSim-based functional verification via a number of case studies. The first is a generic DRS computing platform through which we aim to illustrate ReSim-based verification on an in-house, processor-based DRS design. The second, fault-tolerant application uses DPR to recover from circuit faults introduced by radiation, and we aim to demonstrate verifying an in-house, nonprocessor-based DRS system. Using a third-party design, a video-based driver assistance system [Claus et al. 2007], we then study the use of ReSim to perform functional verification of cutting-edge, complex, real-world DRS applications. Finally, we present the application of ReSim to vendor reference designs.

Overall, we aim to demonstrate that ReSim is flexible enough to simulate various DRS design styles. It should be noted that unless explicitly described, all bugs detected in our case studies were real bugs exposed during the project development. Furthermore, these bugs revealed design flaws in the user designs instead of in the simulation-only layer or in any simulation testbench. Therefore, our case studies can be used as examples to guide future designers in verifying their DRS designs.

3.1. Case Study I: In-House DRS Computing Platform

The DUT of our first case study is the XDRS computing platform (see Figure 4), which is similar to existing generic DRS platforms (e.g., [Bobda et al. 2005; Jara-Berrocal and Gordon-Ross 2010; Sedcole et al. 2007]. This case study aims to use XDRS as a representative system to study the verification of generic DRS computing platforms.

The XDRS computing platform has a control-centric processor and a computation-centric accelerator. It runs a demo streaming application in which the RRs of the `xps_xdrs` accelerator are reconfigured with simple mathematical computational cores (a `Maximum` or a `Reverse` module). Reconfiguration is managed by the in-house `xps_icapi` controller module and its software driver. It should be noted that the XDRS platform is similar to but not the same as the one illustrated in Gong and Diessel [2011a, 2011b]. In particular, in order to systematically study the top-down modeling and the coverage-driven verification of the platform, the system was redesigned from scratch, and the new platform targets a ML507 board instead of an ADM-XRC-4 board.

Using Extended ReChannel, the designer performed TLM modeling and co-simulated the hardware platform and the streaming application software. TLM modeling

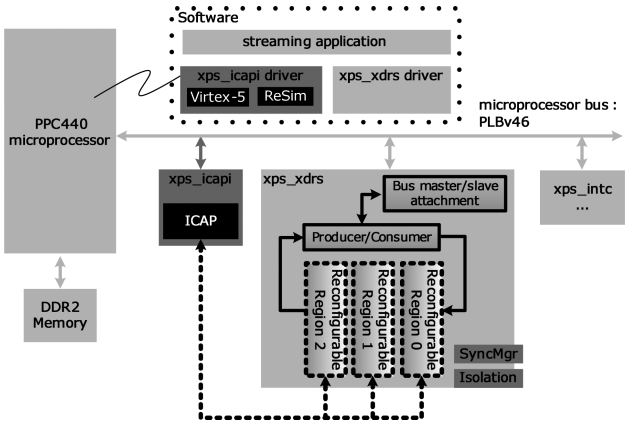

Fig. 4. The XDRS demonstrator.

identified potential design defects in the system hardware (e.g., BUG-Example.XDRS.1). For DPR bugs such as BUG-Example.XDRS.2, the source of errors was the software. For DPR bugs such as BUG-Example.XDRS.3, the bugs were caused by both system software and hardware. Therefore, it was found to be highly desirable to have a simulation environment to test the integrated SW/HW system, including when it is being reconfigured.

BUG-Example.XDRS.1. The `xps_icapi` module uses an `ICAPI_DONE` flag to indicate the end of bitstream transfer. Such a flag should be set to 1 at power up but was incorrectly initialized to 0. This bug was detected in the TLM model of the system, but it revealed a potential bug that could also exist in the RTL design.

BUG-Example.XDRS.2. To accelerate bitstream transfer, the system software buffers bitstreams in a fast DDR2 memory. However, the software failed to flush the bitstream data from the processor cache to the DDR2 memory and the `xps_icapi` module transferred incorrect bitstream data from the DDR2 memory during reconfiguration. The bug was easily identified since un-flushed SimB data was transferred to the simulated ICAP port but the new RM was not swapped in during simulation. This bug would not have been identified without modeling bitstream traffic.

BUG-Example.XDRS.3. The `SyncMgr` mistakenly requested a second reconfiguration before the first one had finished (i.e., an illegal reconfiguration sequence). The bug was detected because the delay associated with the first reconfiguration was accurately modeled by transferring the SimB using Extended ReChannel. The designer fixed the bug by modifying the hardware to report such illegal reconfiguration requests and by adding a `reconfiguration_in_progress` flag to the software driver.

ReSim-based RTL simulation accurately simulates the synchronization, isolation, and initialization mechanisms of the XDRS system. For example, the isolation bug, BUG-Example.XDRS.4, was easily identified, since ReSim models the spurious outputs

of RMs during reconfiguration. In order to thoroughly verify the design, we analyzed the test coverage data reported by the simulator and created tests to cover all possible reconfiguration scenarios. For example, BUG-Example.XDRS.5 was not exposed until we attempted to cover all possible reconfiguration requests scenarios. In both examples, ReSim allowed cycle-accurate simulation of the transition from before to during reconfiguration and from during to after reconfiguration, which are essential to detect bugs in RTL designs.

BUG-Example.XDRS.4. The `Isolation` module, which disconnects the RM during reconfiguration, resumed such connection one cycle too early after reconfiguration. The bug was identified because the errors injected by ReSim propagated to the static part in the mismatched cycle.

BUG-Example.XDRS.5. before reconfiguration, the RM should block a reconfiguration request until it finishes processing the current input sample. However, if a reconfiguration request arrived precisely when the RM had just started processing a sample, the RM failed to block the reconfiguration request.

We mapped a second DRS application to XDRS and performed ReSim-based verification. The second application periodically reconfigured one slot of the `xps_xdrs` accelerator with either an `adder` core or a `maximum` core. Apart from computation, each core maintained a `statistic` register, the value of which was copied across configuration periods, and the saving and restoration of the `statistic` register was performed via the ICAP. It should be noted that the periodic application was derived from Gong and Diessel [2012] and was redesigned in order to target the generic XDRS platform.

We detected five software bugs (e.g., BUG-Example.XDRS.6) while simulating the periodic application. The simulated design was subsequently tested on an ML507 board with a Virtex-5 FX70T FPGA, and we detected one fabric-dependent bug (i.e., BUG-Example.XDRS.7). Since ReSim failed to mimic the exact behavior of the target device, BUG-Example.XDRS.7 was missed by ReSim-based simulation and was identified by debugging the implemented design using ChipScope [Xilinx 2010a]. However, as ChipScope was only able to visualize a limited number of signals for a limited period of time, we used five iterations to trace the cause of one bug, and each iteration involved a 59-minute turnaround time to insert new probing logic and to reimplement the design. ChipScope-based debugging was therefore found to be time consuming.

BUG-Example.XDRS.6. The application software passed an incorrect pointer to the restoration routine of the software driver. The data dereferenced from this incorrect pointer was used to restore the `statistic` register. The bug was detected as a consequence of incorrect values being restored to the simulated `statistic` register.

BUG-Example.XDRS.7. The number of pad words returned from ICAP was not the same as the designer had expected. The software attempted to extract state bits from the wrong bit positions, and the extracted state value was therefore incorrect.

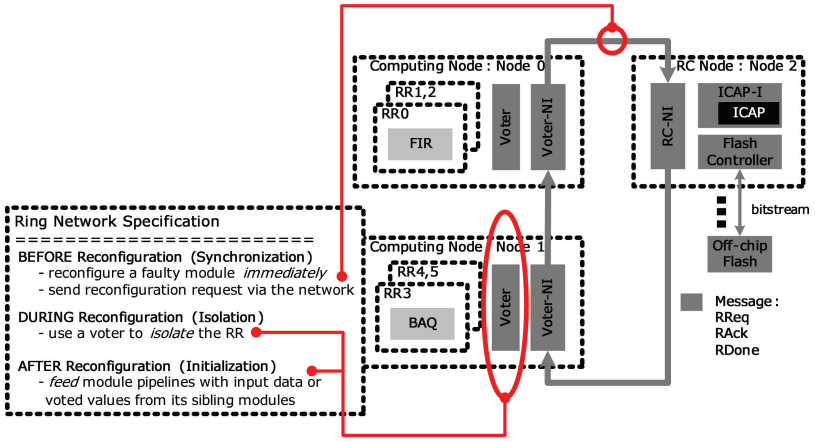

Fig. 5. The hardware architecture of the fault-tolerant DRS.

3.2. Case Study II: In-House Fault-Tolerant Application

The second case study involves the design of signal processing circuits to be used in space-based applications (e.g., Synthetic Aperture Radar) [Cetin et al. 2013]. In particular, a module suffering from permanent single event upsets (SEU) is recovered by reconfiguring it with a correct copy. DPR has been used in such fault-tolerant applications (e.g., [Paulsson et al. 2006; Ichinomiya et al. 2010; Cetin et al. 2013]), and this case study aims to apply ReSim and coverage analysis to verifying the DPR activities in fault-tolerant applications. We also compared the results of coverage-driven verification with ad-hoc on-chip debugging.

In the fault-tolerant DRS application (see Figure 5), voters that protect signal processing circuits are connected via a self-timed ring network. For each computing node (i.e., nodes 0 or 1), if the voter detects that one copy of the protected circuit is permanently affected by an SEU, it initiates a reconfiguration request by passing messages over the network. DURING reconfiguration, the module undergoing reconfiguration may produce spurious outputs, which are ignored by the voter. Thus, the voter works as an isolation module during reconfiguration and is therefore considered to be a component of the reconfiguration machinery (i.e., moderately shaded modules). After reconfiguration, the execution state of the newly reconfigured module is recovered by loading its pipelines with correct values either from the input data stream or from the checked feedback of its sibling modules [Cetin et al. 2013].

In order to compare simulation with on-chip debugging, we deliberately used ChipScope to test the RC-node (i.e., node 2) and detected four bugs. For example, the designer accidentally introduced BUG-Example.FT.1, and since the designer thought it was a bug in the logic, it took him five hours to trace the bug, which would have been identified very quickly in simulation. We then used traditional RTL simulation (i.e., without ReSim) to test the two computing nodes and detected ten bugs in the voter and the voter network-interface.

> *BUG-Example.FT.1.* The designer accidentally created a level-sensitive clock instead of an edge-sensitive one, as desired.

We used ReSim to test the integrated design (i.e., including all three nodes) and detected seven DPR-related bugs in the reconfiguration machinery. By analyzing the

causes of the bugs, we determined that four out of the seven DPR-bugs could have been detected by prior simulation methods, such as Virtual Multiplexing [Luk et al. 1997] or DCS [Robertson and Irvine 2004]. In particular, since bitstreams are transferred over a dedicated datapath, reconfiguration could be modeled with a constant delay. However, three bugs could only have been identified using ReSim, and we describe two of these bugs (i.e., BUG-Example.FT.2, BUG-Example.FT.3) in detail.

BUG-Example.FT.2. After reconfiguration, the system failed to feed enough data to the RM pipelines. This bug was exposed since the errors injected to the pipeline registers of the simulated RM were not properly flushed and propagated to the static region after reconfiguration.

BUG-Example.FT.3. Typically, reconfiguration is slower than transferring a message between two nodes. The expected operation of the RC-node is therefore: start reconfiguration; transfer an RAck message to a computing node; end reconfiguration; transfer an RDone message to a computing node. However, when the bitstream is very small and when the computing node is executing with a very slow clock, reconfiguration can finish before the RAck message is transferred. Under such a circumstance, the RC-node did not correctly send the RDone message.

Coverage analysis was applied to the ReSim-based simulation of the integrated design. For example, we randomized the SimB size so as to exercise the design with various SimBs and randomized the operating frequencies of nodes so as to cover various clock-frequency-related coverage items. We were able to identify a corner case bug (i.e., BUG-Example.FT.3), which is only exposed when the system reconfigures a very small RM whose bitstream is short. Unfortunately, such a scenario was not covered when testing the RC-node on the FPGA, and it is difficult to test since the size of a real bitstream cannot easily be adjusted for test purposes. Therefore, although on-chip debugging tests real-world conditions, it can sometimes only validate a limited number of the possible execution scenarios for a design. Previous simulation methods (e.g., [Schallenberg et al. 2009]) tend to annotate the reconfiguration delay according to the size of a real bitstream, which may also limit the possible scenarios that can be exercised. Since a SimB is not dependent on the FPGA fabric, the size of a SimB can easily be adjusted for test purposes. ReSim-based simulation is therefore better able to exercise some of the corner cases of a design.

3.3. Case Study III: Third-Party Video-Processing Application

The third case study involved the design and verification of a cutting-edge, dynamically reconfigurable driver-assistance system [Gong et al. 2013], which was modified from a recent design of the AutoVision project [Altenried 2009]. In particular, the DUT (see Figure 6) could be viewed as a re-integration of the original design modules. We thereby were able to study the application of an IP-reuse methodology to the verification of the system and analyze the bugs detected.

In order to compare ReSim-based simulation with virtual multiplexing, we deliberately used virtual multiplexing [Luk et al. 1997] to simulate the system at the beginning of the case study. Using virtual multiplexing, we inserted a multiplexer to switch the currently active engine with the selection being performed by writing to a software-accessible `engine_signature` register instead of by transferring a bitstream. With this simulation approach, the software and hardware tested in simulation

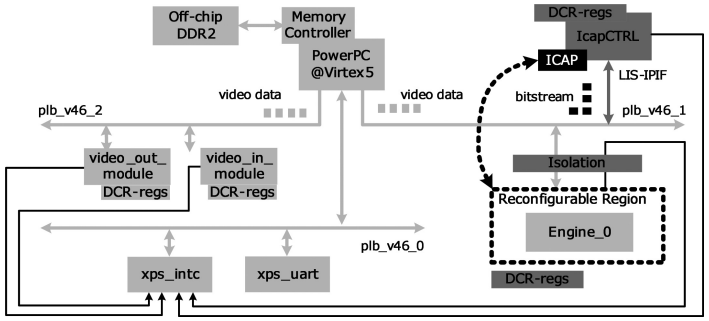

Fig. 6. The hardware architecture of the video-processing application [Gong et al. 2013].

did not match what was actually implemented, and we even detected a false positive bug (i.e., BUG-Example.AUTO.1) in the `engine_signature` register. Furthermore, since the `IcapCTRL` module was instantiated in the design but was not used in simulation, virtual multiplexing was unable to detect bugs in the bitstream transfer datapath (e.g., BUG-Example.AUTO.2). Last but not least, since multiplexing the simulated engines did not generate erroneous signals, as implemented designs might do, the isolation mechanism (i.e., the `Isolation` module) was not tested in simulation.

BUG-Example.AUTO.1. The `engine_signature` register was not correctly initialized, thus no engine was selected to be active. Since the `engine_signature` register only exists in Virtual Multiplexing, this bug was a *false alarm*. Since ReSim does not change the user design, this bug would not have been introduced with ReSim-based simulation.

We started using ReSim after the design had matured and detected six DPR-related bugs that were missed by virtual multiplexing (e.g., BUG-Example.AUTO.2, BUG-Example.AUTO.3, and BUG-Example.AUTO.4). Although individual engines and their software drivers were already FPGA-proven from the original design, bugs were introduced through mismatches between module parameters (e.g., BUG-Example.AUTO.2) and software/hardware parameters (e.g., BUG-Example.AUTO.3). Apart from detecting bugs introduced by modifying the original design, we were able to identify three potential bugs in parts of the system that were original. For example, BUG-Example.AUTO.4 was not exposed before because the original design used a faster configuration clock. This bug was identified because ReSim did not activate the newly configured module until all words of the SimB were successfully written to the ICAP. Therefore, the use of SimBs more accurately modeled the timing associated with partial reconfiguration, and ReSim-based simulation was effective in testing the integrated driving-assistance system.

3.4. Case Studies IV & V: Vendor Reference Designs

These case studies demonstrate the use of ReSim to verify two vendor reference designs. Since the reference designs had already been proven, it is not surprising that we were not able to detect bugs using ReSim-based simulation. However, simulating vendor reference designs demonstrates the robustness and the flexibility of ReSim.

Table III. The Effect of SimB Size on Verification Coverage

Test ID	SimB Size (B)	Individual Coverage (%)	Accumulated Coverage (%)	CPU Time (s)
1	32	56.7	56.7	253
2	48	68.62	68.85	251
3	80	69.01	69.64	257
4	144	69.01	69.64	264
5	272	69.01	69.64	310

BUG-Example.AUTO.2. The `IcapCTRL` module was directly connected to memory in the original system, and it failed to work with the shared PLB bus as used in the modified Optical Flow Demonstrator. Since ReSim modeled bitstream traffic, this bug was readily detected by ReSim-based simulation. This bug was introduced by changing the way the IP was integrated.

BUG-Example.AUTO.3. After changing a parameter of the `IcapCTRL` module, the software driver was not updated accordingly and the SimB was not successfully transferred. This bug was detected when a new engine was not swapped in due to the bug in the bitstream transfer process. The bug was introduced by a mismatch between hardware and software.

BUG-Example.AUTO.4. The system software failed to wait until the completion of bitstream transfer before resetting the engines. This bug was introduced when the design was modified to use a different clocking scheme that slowed down the bitstream transfer and the software was not updated to slow down the reset operations accordingly. Since ReSim more accurately modeled the timing of reconfiguration events, this bug could only be detected by ReSim-based simulation.

In the first case study, a Fast PCIe configuration (FPCIe) reference design loads a lightweight PCIe endpoint logic block within the required time at startup. The rest of the FPGA is then dynamically configured with the core application logic via the established PCIe link [Tam and Kellermann 2010]. In addition to performing cycle-accurate simulation on the reference designs, as reported in Gong and Diessel [2011b], we tuned the SimB to analyze the coverage of the Fast PCIe reference design. In particular, we adjusted the size of the SimB to exercise various scenarios of the bitstream datapath (see Table III). It should be noted that the following coverage analysis was performed on FPCIe design that used a 64-entry FIFO to buffer the bitstream instead of the 1024-entry one provided with the reference design.

To exercise the FIFO_FULL scenario, we increased the SimB size from 48 to 80 bytes, which is beyond the 64-byte FIFO capacity, and the coverage increased accordingly. The table also assists in selecting an optimal test set for regression. Considering Test 3, the individual coverage using 80 bytes of SimB was less than the accumulated coverage for Tests 1, 2, and 3. This increase in coverage was achieved through the addition of the shorter tests, which exercised scenarios that were missed by the longer test. Therefore, Tests 1, 2, and 3 form an optimal set of "golden" tests which achieve the highest accumulated coverage for the least simulation time.

In the last case study, a Reconfigurable Peripheral reference design instantiates the bus-based xps_hwicap IP core as a reconfiguration controller in order to swap peripheral modules (e.g., the xps_math module) attached to a bus [Xilinx 2009a]. By modeling the bitstream traffic, ReSim-based simulation was used to test the hardware logic and the software driver of the xps_hwicap IP core and to verify that the core was correctly integrated with the rest of the system. It should be noted that in order to simulate the system, we changed the reference design slightly. Since the reference design did not provide source code for the math cores, we designed our own simple math cores. Furthermore, since we did not have a simulation model for the provided SD card, we assume that bitstreams are correctly copied from the SD card to the DDR2 memory.

Although the design was proven, we were able to expose a potential isolation bug in the system (i.e., BUG-Example.UG744.1). This isolation bug was not exposed in the reference design because RMs were not accessed during reconfiguration. The bug was exposed after modifying the software to access the RM during reconfiguration and was identified during ReSim-based simulation of the modified design and when running the modified design on the target FPGA.

BUG-Example.UG744.1. The Reconfigurable Peripheral reference design did not isolate the RR undergoing reconfiguration. Spurious outputs could therefore propagate from the RR to the static part of the system

3.5. Summary and Lessons Learnt

Our case studies demonstrate that ReSim can be applied to a range of DRS design styles. In particular, we demonstrated the use of ReSim with in-house designs (i.e., the XDRS platform and the fault-tolerant DRS), a third-party design (i.e., AutoVision), and reference designs; with hardware-only designs (i.e., the FPCIe reference design and the fault-tolerant DRS) and with microprocessor-based HW/SW designs; with demo applications (i.e., streaming and periodic applications) and with real-world applications (i.e., a fault-tolerant application and a video processing application); a design that saves and restores module state via the configuration port (i.e., the periodic application on XDRS); with designs that use customized reconfiguration controllers (i.e., xps_icapi, IcapCtrl) and vendor IP (i.e., the xps_hwicap core); as well as with designs that were mapped to Virtex-5 (i.e., XDRS, fault-tolerant DRS, AutoVision) and to Virtex-6 (i.e., the two reference designs).

Table IV summarizes the case studies. We summarize these case studies in terms of the development workload[5], simulation overhead, and bugs detected. The bugs in the table were all exposed in verifying the designs, that is, it does not include the isolation bug (i.e., BUG-Example.UG744.1) that was deliberately introduced to the Processor Peripheral reference design. The table also does not include false positive bugs (i.e., BUG-Example.AUTO.1) identified in the case study.

3.5.1. Development Workload. The extra development workload for using ReSim involved creating parameter scripts [Gong and Diessel 2011b], which ranged from 50–150 LOC (Tcl) for these case studies. The parameters include a list of interfacing signals that cross RR boundaries, the affiliation of RMs and RRs, and the target FPGA family

[5]The lines of code (LOC) of designs only includes the reconfiguration machinery.

Table IV. Summary of Case Studies

Case Study	Complexity of the reconfiguration machinery (LOC)	Parameter Script (LOC)	Simulation Overhead (%)	DPR-related Bugs (ReSim/Others)
XDRS (Streaming Application)	1300 (Verilog, excluding EDK code) + 1150 (C)	50 (Tcl)	8.3	34/0
XDRS (Periodic Application)	1300 (Verilog, excluding EDK code) + 1750 (C)	50 (Tcl)	6.8	5/1
Fault-tolerant DRS	2150 (Verilog, excluding ICAP-I & Flash controller)	60 (Tcl)	20.9	18/4
AutoVision	1250 (VHDL) + 400 (C)	80 (Tcl)	1.7	7/0
Fast PCIe Configuration (XAPP883)	3500 (Verilog, excluding CoreGen code)	150 (Tcl)	0.3	–
Processor Peripheral (UG744)	2400 (VHDL) + 3200 (C)	50 (Tcl)	0.7	–

[Gong and Diessel 2011b]. Generally speaking, the workload of using ReSim is trivial compared to the effort spent creating a DRS design and a testbench.

3.5.2. Simulation Overhead. For each case study, we used the ModelSim profiling tool to evaluate the simulation overhead of ReSim. We found that 0.3–20.9% of simulation time was spent in ReSim. The simulation overhead of ReSim is proportional to the number of signals crossing the RR boundary, since all boundary signals are multiplexed as opposed to being connected to the static part directly. The overhead is also proportional to the frequency of reconfiguration in a specific simulation run, since each reconfiguration involves costs to swap modules and inject errors and includes a scenario-dependent delay to transfer the SimB. Overall, the simulation overhead was lower for more complex designs (e.g., AutoVision, FPCIe, etc.) and was higher for simpler designs (e.g., the fault-tolerant DRS).

3.5.3. Bugs Detected. From the bugs detected in the case studies, we notice that the reconfiguration process could be much more difficult to verify than a designer might expect. Correctly verifying the RMs and the static region is essential but does not guarantee the correct transition from one configuration to another. Table V lists all example bugs described in this section (including false positive bug(s) and deliberately introduced bug(s)). Apart from citing the Bug ID, we add to each bug a tag indicating the key words associated with each bug so as to remind readers of the details of these bugs. Bugs could be introduced immediately before (e.g., BUG-Example.XDRS.2, BUG-Example.XDRS.5, $\cdots$), during (e.g., BUG-Example.XDRS.4, BUG-Example.AUTO.2, $\cdots$), and after (e.g., BUG-Example.FT.2, BUG-Example.AUTO.4, $\cdots$) reconfiguration. Bugs could be introduced to hardware, software (e.g., BUG-Example.XDRS.2, BUG-Example.XDRS.6, BUG-Example.AUTO.3), and a mixture of hardware and software (e.g., BUG-Example.XDRS.3, BUG-Example.AUTO.4). Even when individual modules and software functions were FPGA-proven and reused, bugs could still be introduced due to mismatches between module parameters (e.g., BUG-Example.AUTO.2), inconsistencies between software and hardware (e.g., BUG-Example.AUTO.3), and using proven IP in a different scenario (e.g., BUG-Example.FT.3, BUG-Example.AUTO.4). Therefore, it is highly desirable to test and debug an integrated DRS

Table V. Summary of Example Bugs Described in Case Studies

Bug ID	Bug Tag	Section	Detected by
BUG-Example.XDRS.1	ICAPI_DONE	3.1	Extended ReChannel
BUG-Example.XDRS.2	Flush_Cache	3.1	**Extended ReChannel**
BUG-Example.XDRS.3	Multiple_Recon	3.1	**Extended ReChannel**
BUG-Example.XDRS.4	Isolation_Mismatch	3.1	**ReSim**
BUG-Example.XDRS.5	Block_Until_Idle	3.1	ReSim
BUG-Example.XDRS.6	Restoration_Pointer	3.1	**ReSim**
BUG-Example.XDRS.7	Wrong_Pad_Word	3.1	On-chip Debugging
BUG-Example.FT.1	Level_Clock	3.2	On-chip Debugging Could have been detected by ReSim
BUG-Example.FT.2	Feed_Pipeline	3.2	**ReSim**
BUG-Example.FT.3	Too_Quick_Recon	3.2	**ReSim**
BUG-Example.AUTO.1	Sig_Reg	3.3	Virtual Multiplexing
BUG-Example.AUTO.2	IcapCtrl_PLB	3.3	**ReSim**
BUG-Example.AUTO.3	Driver_Update	3.3	**ReSim**
BUG-Example.AUTO.4	Engine_Reset	3.3	**ReSim**
BUG-Example.UG744.1	Bus_Isolation	3.4	ReSim On-chip Debugging

design, including the process of reconfiguring the design with a range of configuration bitstream sizes that could be expected at runtime.

Bugs that can only be detected by ReSim/Extended ReChannel-based simulation are marked in bold in the "Detected by" column in Table V (see detailed descriptions of each bug in relevant sections). Other bugs could also be detected with previous methods, such as virtual multiplexing [Luk et al. 1997] and DCS [Robertson and Irvine 2004]. For example, BUG-Example.XDRS.1 and BUG-Example.XDRS.5 could have been detected by traditional RTL simulation, because exposing the two bugs did not require modeling any characteristic features of DPR. As another example, BUG-Example.UG744.1 could also be detected by DCS, which also injects errors to the static region while the RM is being reconfigured [Robertson and Irvine 2004]. However, since ReSim models the characteristic features of DPR, it is able to detect bugs that would have been missed by previous methods (see Section 3.3). Furthermore, since ReSim does not require changing the design for simulation purposes, it does not introduce false positive bugs (e.g., BUG-Example.AUTO.1). For a complete list of all DPR-related bugs detected in the case studies, please refer to Gong [2013].

Our case studies indicate that it is nontrivial to insert probe logic and to debug the implemented DRS design using ChipScope (see BUG-Example.XDRS.7, BUG-Example.FT.1). Furthermore, on-chip debugging does not collect coverage and can only validate a limited number of scenarios for the implemented design (see BUG-Example.FT.3). Even when a module is FPGA-proven, a bug can still be introduced during system integration (see BUG-Example.AUTO.2). However, on-chip debugging is completely accurate and can detect fabric-dependent bugs (e.g., BUG-Example.XDRS.7) which cannot be detected by ReSim-based simulation (see Section 2.4). Given the

difficulty of meeting both the requirements of simulation accuracy and verification productivity, we do not advocate replacing on-chip debugging with simulation approaches. Nevertheless, we believe that it is highly desirable to perform simulation to identify and fix as many fabric-independent bugs as possible in the early stages of the design cycle, and to leave the fabric-dependent part of the design to be tested on the target FPGA.

4. RELATED AND FUTURE WORK

4.1. Related Work

Most existing approaches for simulating DRS do not model the FPGA fabric and thereby compromise simulation accuracy. For example, at the RTL level, MUX-based methods such as Luk et al. [1997] and Robertson and Irvine [2004] instantiate all RMs in the simulation environment and select one RM at a time. High-level modeling techniques such as Raabe et al. [2008] and Schallenberg et al. [2009] extend SystemC with new classes or language constructs to model reconfiguration. These methods only model module swapping and fail to capture other characteristic features, such as bitstream traffic. Furthermore, since the design needs to be manually changed/modeled for simulation purposes (e.g., by inserting a virtual multiplexer), MUX-based methods verify a variation of the design instead of the implementation-ready design. Recent work [Hansen et al. 2013] extends ReSim by using real bitsteams and extracting reconfiguration timing information from the target FPGA devices. While slightly improving the modeling accuracy, the simulation is undesirably dependent on the FPGA fabric. The work did not study how many more bugs could be detected with such improvement in accuracy.

Formal verification mathematically proves that a hardware design meets its formal specification [Drechsler 2004]. Formal methods have been used to verify reconfigurable cores at runtime by running an online prover [Singh and Lillieroth 1999] or by checking the proof carried by the hardware task to be reconfigured [Drzevitzky et al. 2009]. Formal verification can also be used at system design time. The tiny pi-calculus approach applies a subset of pi-calculus to formally model the reconfiguration rules/directives (i.e., when to reconfigure what) of a DRS design [Seffrin et al. 2010]. Symbolic simulation has been used to verify reconfigurable streaming applications meet their executable software specification [Todman et al. 2012]. However, existing formal methods can only capture the module swapping operation (e.g., using the `reconfigure_if` conditional statement [Todman et al. 2012]) in the formal specification and fail to model other DPR-related properties, such as "a module should be swapped in after the transfer of a bitstream".

DRS designs can also be tested using vendor tools (e.g., ChipScope [Xilinx 2010a]). However, ChipScope-based debugging is a time-consuming step for FPGA-based designs. Furthermore, it requires expert knowledge of the ChipScope tool in order to probe signals of RMs [Xilinx 2011], which introduces additional complications to the on-chip debugging of DRS designs.

4.2. Future Work

Since using the simulation-only layer offers only limited assistance in detecting fabric-dependent bugs, we propose two possible directions to resolve such a verification gap but leave both directions as future work.

One direction is to further improve the simulation accuracy of the simulation-only layer by capturing more details of the FPGA fabric (e.g., placement information, configuration bits). Recent work [Hansen et al. 2013] is an example of such extension so as to simulate module relocation in slot-based DRS designs. As another example, instead

of simulating the entire FPGA fabric, it is possible to simulate a small portion of the design (e.g., the LUTs that are fine-grained reconfigurable) in a fabric-accurate manner, which can significantly improve the simulation accuracy for scenarios of interest. The more accurate the simulation-only layer is, the less mismatches exist between the simulation-only layer and the target FPGA, and the fewer false positive and negative bugs are introduced.

On the other hand, given the fundamental conflict between simulation accuracy and verification productivity, it is more or less inevitable to rely on on-chip debugging to identify fabric-dependent bugs. Therefore, the other way to resolve the verification gap is to improve the verifiability of a DRS design by changing the way a DRS is designed without compromising other design requirements. In particular, the following is desirable.

—The designer keeps the fabric-dependent part to a minimum or avoids using fabric-dependent features in the design. This can also improve the portability of the design while reducing verification difficulty.
—The designer reuses existing proven design tools and IP so as to avoid creating fabric-dependent components from scratch.
—The designer separates the fabric-dependent and fabric-independent parts so as to localize potentially unidentified bugs to the fabric-dependent part of the design. By carefully partitioning the design, the designers can maximize the number of fabric-independent bugs that could be exposed using the simulation-only layer and field testing can focus on debugging the fabric-dependent part, which requires less effort than debugging the entire design.

The simulation-only layer can also be extended to formal verification. For DRS designs, a formal specification may state, in formal language, that "a module should be swapped in after the transfer of a bitstream", and such a statement requires accurately modeling characteristic features of reconfiguration with acceptable levels of detail of the FPGA fabric. It would be possible to implement the simulation-only layer such that it can easily be used by formal verification tools. As a result, designers could then formally describe DRS design properties that involve both user design modules and components of the physical layer.

Looking ahead, dynamic reconfiguration has changed the way engineers design hardware and will also change the way hardware is verified. With dynamic reconfigurability, a DRS can be designed to be open-ended, which means that hardware RMs can be sourced externally to the system [Xilinx 2010c]. Since RMs are not known at system design time, a DRS needs the capability to verify its correct reconfiguration and execution at runtime. Despite limited research being done, runtime verification is a long-term trend for future DRS designs. Compared with design-time verification, runtime verification of open-ended DRS designs introduces two main challenges.

—Hardware modules/tasks sourced externally to the system may not have been thoroughly verified. Functional bugs could exist in the module itself or could be introduced when integrating the module with the rest of the system. Even for thoroughly verified modules, the reconfiguration process could introduce bugs to the system. If, for example, the frame addresses had been corrupted, reconfiguration could incorrectly change the static user logic.
—Apart from checking bugs that are unintentionally introduced to the design, the system also needs to identify malicious hardware tasks that could damage the system. Therefore, runtime verification needs to verify the credibility of the source of configuration bitstreams and reconfiguration requests. The verification problem thus morphs into a security problem.

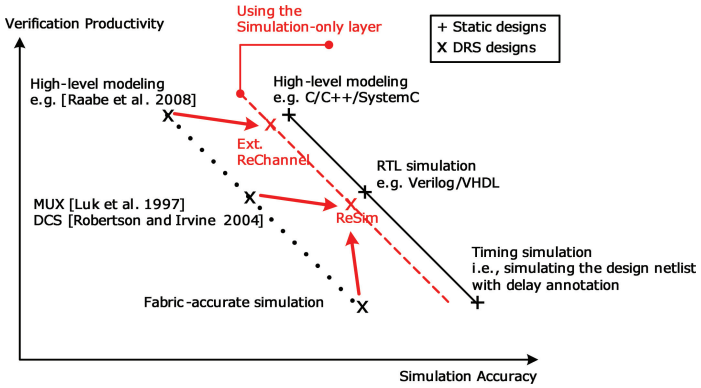

Fig. 7. Accuracy and productivity trade-off in simulating DRS designs.

Since runtime verification directly tests hardware tasks on the target FPGA, it is not necessary to provide designers with a simplified simulation model of the FPGA device and runtime verification should therefore be 100% accurate. However, runtime verification suffers from low visibility and controllability. Furthermore, the system may not be able to afford the resources to perform the verification. Existing runtime verification techniques are still ad-hoc (see Section 4.1), and significant effort is needed to address these issues.

5. CONCLUSIONS

As with modern ASIC designs, functional verification has become a significant challenge for DRS designs. This article has studied simulation-based functional verification of advanced, dynamic reconfiguration capabilities of modern FPGA-based designs. We summarize our work on the simulation library, ReSim, which enables a unified simulation environment to test and debug integrated DRS designs. The case studies demonstrated that ReSim can achieve sufficient accuracy for functional verification and that ReSim-based simulation assists in detecting bugs that would be missed using traditional methods. The development and simulation overheads of using ReSim are typically insignificant.

For the verification of hardware designs, designers need to find a balance between simulation accuracy and verification productivity (see Figure 7). For static designs, either FPGA-based or ASIC-based (the solid line in the figure) cycle-accurate RTL simulation is at the middle of the spectrum and is suitable for functional verification. However, designers have to sacrifice either accuracy or productivity in simulating DRS designs (the dotted line in the figure). Although fabric-accurate simulation is completely accurate, its productivity is too low for verifying design functionality. Without being dependent on the fabric, previous work using either RTL simulation (e.g., MUX-based method [Luk et al. 1997], DCS [Robertson and Irvine 2004]) or high-level modeling (e.g., ReChannel [Raabe et al. 2008]) maintains the desired level of productivity while sacrificing accuracy in simulating the reconfiguration process.

By mimicking the FPGA fabric using a simulation-only layer, ReSim significantly improves simulation accuracy over previous RTL-level approaches with negligible productivity penalties and significantly improves simulation productivity compared with fabric-accurate simulation. Using similar concepts, Extended ReChannel also improves the accuracy of high-level modeling without sacrificing much productivity. Extended ReChannel and ReSim are two representatives of simulation methods that use the simulation-only layer approach (the dashed line in Figure 7). Other work (e.g., [Hansen

et al. 2013]) has studied alternative trade-offs between simulation accuracy and verification productivity in the verification of DRS designs.

ReSim has been released as an open-source tool under the BSD license and is publicly available via `http://code.google.com/p/resim-simulating-partial-reconfiguration/` .

ACKNOWLEDGMENTS

The researchers would like to thank Mr. Jens Hagemeyer from the University of Paderborn for his guidance on state restoration on Virtex-5 FPGAs. We also thank Mr. Johny Paul and Prof. Walter Stechele from the Technical University of Munich for providing the AutoVision design as an important case study for assessing ReSim.

REFERENCES

Florian Altenried. 2009. Time-sharing of hardware resources for image processing accelerators using dynamic partial reconfiguration. Bachelor's thesis. Technical University of Munich. (2009).

Altera. 2010. Increasing design functionality with partial and dynamic reconfiguration in 28-nm FPGAs (WP01137). Altera Inc. http://www.altera.com/literature/wp/wp-01137-stxv-dynamic-partial-reconfig.pdf.

Christian Beckhoff, Dirk Koch, and Jim Torresen. 2010. Short-circuits on FPGAs caused by partial runtime reconfiguration. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*. 596–601.

Christophe Bobda, Mateusz Majer, Ali Ahmadinia, Thomas Haller, Andre Linarth, and Jurgen Teich. 2005. The Erlangen slot machine: Increasing flexibility in FPGA-based reconfigurable platforms. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*. 37–42.

Ediz Cetin, Oliver Diessel, Lingkan Gong, and Victor Lai. 2013. Towards bounded error recovery time in FPGA-based TMR circuits using dynamic partial reconfiguration. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*.

Christopher Claus, Johannes Zeppenfeld, Florian Muller, and Walter Stechele. 2007. Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system. In *Proceedings of the Design, Automation and Test in Europe (DATE)*. 1–6.

Rolf Drechsler. 2004. *Advanced Formal Verification*. Kluwer Academic Publishers.

Stephanie Drzevitzky, Uwe Kastens, and Marco Platzner. 2009. Proof-carrying hardware: Towards runtime verification of reconfigurable modules. In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. 189–194.

Lingkan Gong. 2013. ReSim case studies. http://code.google.com/p/resim-simulating-partial-reconfiguration/.

Lingkan Gong and Oliver Diessel. 2011a. Modeling dynamically reconfigurable systems for simulation-based functional verification. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 9–16.

Lingkan Gong and Oliver Diessel. 2011b. ReSim: A reusable library for RTL simulation of dynamic partial reconfiguration. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*. 1–8.

Lingkan Gong and Oliver Diessel. 2012. Functionally verifying state saving and restoration in dynamically reconfigurable systems. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*. 241–244.

Lingkan Gong, Oliver Diessel, Johny Paul, and Walter Stechele. 2013. RTL simulation of high performance dynamic reconfiguration: A video processing case study. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*.

Simen Gimle Hansen, Dirk Koch, and Jim Torresen. 2013. Simulation framework for cycle-accurate RTL modeling of partial run-time reconfiguration in VHDL. In *Proceedings of the International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*. 1–8.

Yoshihiro Ichinomiya, Shiro Tanoue, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga, and Toshinori Sueyoshi. 2010. Improving the robustness of a softcore processor against SEUs by using TMR and partial reconfiguration. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 47–54.

Abelardo Jara-Berrocal and Ann Gordon-Ross. 2010. VAPRES: A virtual architecture for partially reconfigurable embedded systems. In *Proceedings of the Design, Automation and Test in Europe (DATE)*. 837.

Wayne Luk, Nabeel Shirazi, and Peter Y. K. Cheung. 1997. Compilation tools for run-time reconfigurable designs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 56–65.

Katarina Paulsson, Michael Hubner, Markus Jung, and Jurgen Becker. 2006. Methods for run-time failure recognition and recovery in dynamic and partial reconfigurable systems based on Xilinx Virtex-II Pro FPGAs. In *Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*. 1–6.

Andreas Raabe, Philipp A. Hartmann, and Joachim K. Anlauf. 2008. ReChannel: Describing and simulating reconfigurable hardware in SystemC. *ACM Trans. Des. Autom. Electron. Syst.* 13, 1, 15.

Ian Robertson and James Irvine. 2004. A design flow for partially reconfigurable hardware. *ACM Trans. Embed. Comput. Syst.* 3, 2, 257–283.

Andreas Schallenberg, Wolfgang Nebel, Andreas Herrholz, and Philipp A. Hartmann. 2009. OSSS+R: A framework for application level modelling and synthesis of reconfigurable systems. In *Proceedings of the Design, Automation and Test in Europe (DATE)* 970–975.

Pete Sedcole, Peter Y. K. Cheung, George A. Constantinides, and Wayne Luk. 2007. Run-time integration of reconfigurable video processing systems. *IEEE Trans. VLSI Syst.* 15, 9, 1003–1016.

Andre Seffrin, Alexander Biedermann, and Sorin A. Huss. 2010. Tiny-n: A novel formal method for specification, analysis, and verification of dynamic partial reconfiguration processes. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (CSREA)*. 1–6.

Satnam Singh and Carl J. Lillieroth. 1999. Formal verification of reconfigurable cores. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 25–32.

Simon Tam and Martin Kellermann. 2010. *Fast Configuration of PCI Express Technology through Partial Reconfiguration (XAPP883)*. Xilinx Inc.

Tim Todman, Peter Boehm, and Wayne Luk. 2012. Verification of streaming hardware and software codesigns. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*. 147–150.

Michael J. Wirthlin and Brad L. Hutchings. 1998. Improving functional density using run-time circuit reconfiguration. *IEEE Trans. VLSI Syst.* 6, 2, 247–256.

Xilinx. 2009a. PlanAhead software tutorial: Partial reconfiguration of a processor peripheral (UG744). Xilinx Inc.

Xilinx. 2009b. *Virtex-4 FPGA Configuration User Guide (UG071)*. Xilinx Inc.

Xilinx. 2010a. ChipScope Pro 12.1 software and cores (UG029). Xilinx Inc.

Xilinx. 2010b. EDK concepts, tools and techniques (UG683). Xilinx Inc.

Xilinx. 2010c. *Partial Reconfiguration User Guide (UG702)*. Xilinx Inc.

Xilinx. 2010d. *Virtex-5 FPGA Configuration User Guide (UG191)*. Xilinx Inc.

Xilinx. 2010e. *Virtex-6 FPGA Configuration User Guide (UG360)*. Xilinx Inc.

Xilinx. 2011. Partial reconfiguartion - Can I insert chipscope cores within reconfigurable modules? Xilinx Inc. http://www.xilinx.com/support/answers/42899.

Xilinx. 2013. *7 Series FPGAs Configuration User Guide (UG470)*. Xilinx Inc.

Received May 2013; accepted October 2013