ReSim: A Reusable Library for RTL *Sim*ulation of Dynamic Partial *Re*configuration

Lingkan Gong and Oliver Diessel
School of Computer Science and Engineering
University of New South Wales
{lingkang,odiessel}@cse.unsw.edu.au

***Abstract*—Dynamic Partial Reconfiguration (DPR) enables software-like flexibility in hardware designs by allowing some of the logic on a Field Programmable Gate Array (FPGA) to be reconfigured while the rest continues to operate. However, such flexibility introduces challenges for verifying DPR design functionality because there is no straightforward way to simulate DPR at Register Transfer Level (RTL). This paper proposes the ReSim library to enable the RTL simulation of DPR. The library uses a simulation-only layer to hide the physically dependent details of DPR designs while providing sufficient accuracy for functional verification. The library is extensible and reusable. We assess the feasibility and demonstrate the value of our tool via two case studies of DPR designs.**

I. Introduction

Dynamic Partial Reconfiguration (DPR) allows some of the logic on a Field Programmable Gate Array (FPGA) to be reconfigured while the rest continues to operate. It enables software-like flexibility in hardware designs. Systems using such technology can save cost by time-multiplexing hardware resources and improve performance by adapting to changing execution requirements [1]. However, such flexibility introduces significant new challenges to ensuring the functional correctness of DPR designs.

Register Transfer Level (RTL) simulation is the most common method for verifying hardware design functionality. Although the various configurations of a DPR design are complete and fixed hardware designs in themselves, the actual process of reconfiguring a design can not be simulated by mainstream tools [2]. This inability to simulate the reconfiguration process poses significant difficulties for verifying DPR designs.

The reconfiguration process is prone to error and it is therefore necessary to perform cycle-accurate RTL simulation to detect potential bugs within the process [3]. More specifically, we have identified 14 potential sources of error BEFORE, DURING, and AFTER reconfiguration. For example, if application data and bitstreams are stored in the same storage device, the system may experience traffic contention DURING reconfiguration. Errors caused by such contention can not be detected without simulating the bitstream traffic. More generally, any reconfiguration activities on the *bitstream datapath*, such as bitstream transfer, compression, decryption and arbitration need to be verified by simulating the bitstream traffic. However, existing MUX-based methods such as [4] and [5] fail to provide the accuracy required to verify the design undergoing reconfiguration because they either swap modules instantaneously or assume a compile-time defined reconfiguration delay.

Simulating the reconfiguration process and the bitstream traffic improves the accuracy of functional verification. However, the bitstream contains physical information such as the frame addresses of the configuration data, thereby exposing the details of the FPGA fabric to the RTL. As a result, accurate simulation of DPR violates a general rule of functional simulation, which states that the simulated design should be physically independent.

Given the importance and difficulty of simulating the reconfiguration process, this paper proposes the ReSim library to address the issue. In particular, we consider the cycle-accurate functional verification of DPR designs but do not address physical implementation errors such as glitches and timing violations. The key contributions of this paper are

- Proposing the use of a simulation-only layer, a collection of physically independent components, to model the FPGA fabric with sufficient accuracy for functional simulation. We introduce a novel simulation-only bitstream to enable *cycle-accurate yet physically independent* simulation of a design undergoing partial reconfiguration.
- Providing a *reusable* library, ReSim, which integrates the simulation-only layer with commonly available tools and enables customization to various DPR design styles.
- Providing case studies showing how ReSim resolves the difficulties of simulating and verifying DPR designs.

The rest of this paper is organized as follows. Section 2 and 3 outlines related efforts and analyzes the challenges of simulating DPR. We discuss the architecture of the simulation-only layer and the ReSim library in Section 4. Section 5 illustrates the use of our tool on two case studies while the final section concludes the paper with a summary of its contribution and an indication of further work.

II. Related Work

FPGA vendors such as Xilinx claim that the various configurations of a DPR design can be individually simulated [2]. However, modern hardware verification practice indicates that

978-1-4577-1740-6/11 $26.00

the most costly bugs are encountered in the system integration stage, and correctly verified sub-systems are mandatory but far from enough to ensure the correctness of the integrated design [6]. Thus, in order to achieve full system simulation, it is essential to simulate the reconfiguration process.

The most common approach for simulating DPR has been to use a MUX to interleave the communication between reconfigurable modules connected in parallel [4]. This method is the basis for more recent efforts in simulating DPR. However, this approach only models module swapping and fails to simulate other aspects of the design undergoing reconfiguration.

The Dynamic Circuit Switch [5] was the first systematic study of simulating DPR. The idea is to add simulation-only artifacts to the RTL code of DPR designs so as to select and activate hardware tasks. Our work extends this idea into a more comprehensive simulation-only layer. One significant difference is that in DCS, reconfiguration is triggered by artifacts monitoring designer selected signals in the RTL code, whereas we use simulation-only dummy bitstreams to model the bitstream traffic and trigger the swapping of modules. As a result, ReSim emulates the behavior of the system on FPGAs more accurately.

ReChannel [7] is a SystemC-based, open source library to model DPR that was later extended in [3]. The original work extends SystemC with new classes such as `rc_reconfigurable` to encapsulate reconfiguration operations such as module swapping. However, such extension only focuses on the high-level modeling of DPR whereas the reconfiguration details of a design are not modeled or verified. The extended ReChannel in [3] adds new classes to capture the reconfiguration details and proposes a modeling methodology to assist functional verification of DPR designs at behavioral, TLM and RTL levels. However, the accurate simulation of the reconfiguration process was not discussed.

OSSS+R [8] is a design methodology for the high-level simulation and synthesis of DPR designs. The tool automatically generates synthesizable code for the reconfiguration controller. However, such automatic synthesis methodologies do not support the simulation and functional verification of manually specified or customized DPR designs.

There are other examples that extend SystemC to model DPR [9] [10]. These works focus on design space exploration, the primary concern of which is to partition the design into static and dynamic parts and hide the reconfiguration details. These are not accurate enough for functional simulation.

As mentioned in Section I, the primary drawback of the existing work is that it fails to provide the accuracy required to verify the design undergoing reconfiguration since the accesses to and the manipulations of the bitstream traffic are not simulated. This paper focuses on the RTL simulation of a DPR design. The following sections show how the simulation-only layer and the ReSim library improve the simulation accuracy while hiding the physical details of a design.

III. Challenges in Simulating DPR

As a general rule, functional simulation at RTL level should be both *cycle-accurate* and *physically independent*. However, the two requirements conflict with each other when simulating DPR. This section illustrates these conflicting challenges from temporal and structural perspectives. We then define the qualities of an ideal simulation method based on our analysis.

A. Temporal Analysis of Reconfiguration

From a temporal perspective, the reconfiguration process can be subdivided into three phases. BEFORE reconfiguration, the static part should *synchronize* with the outgoing reconfigurable module (RM) so that any ongoing computation is properly paused. DURING reconfiguration, the static part needs to *isolate* the reconfigurable region (RR) to avoid the propagation of spurious outputs from partially configured modules. At the same time, the *bitstream datapath* may be involved in complex operations such as decompression, decryption and arbitration, all of which should be verified. AFTER reconfiguration, the incoming RM needs to be *initialized* to a known state before it starts or resumes execution.

The logic for synchronization, isolation, initialization and the bitstream datapath are defined by designers and are therefore prone to error. The potential sources of error identified in [3] indicate the importance of accurately simulating this logic. However, it is difficult to provide the required accuracy because traditional RTL simulation doesn't model the essential features of the reconfiguration process. These features include:

- **Module Swapping:** HDL languages define the hierarchy of a design at compile time and thus don't support module swapping at run time. Moreover, switching modules instantaneously [4], or after a compile-time defined delay [5], hides the correct timing of the swapping.
- **Spurious Outputs:** The RTL code of simulated modules won't produce spurious outputs when being reconfigured. The isolation logic of the DPR design cannot therefore be tested. Moreover, modeling such spurious outputs as a constant such as the undefined "x" value in [5] lacks the flexibility required to thoroughly test the isolation logic.
- **Bitstream Traffic:** The RTL code of the bitstream datapath cannot be verified without simulating the bitstream traffic, which isn't available until after the time-consuming implementation step.
- **Triggering Condition:** Even when the real bitstreams are available, RTL simulation cannot make use of them to trigger module swapping without modeling the FPGA fabric and the effect of the bitstream upon it. This calls for far more modeling detail than is desirable and would only be feasible if the FPGA vendor were to provide such a model. The simulation of the triggering condition therefore doesn't replicate what is actually implemented. For example, [5] uses a non-synthesizable component called a Reconfiguration Condition Detector to trigger module

swapping, but there is no corresponding component in implemented designs.

- **Undefined Initial State:** The initial state of RMs are undefined AFTER reconfiguration. However in simulation, if, for example, an RM is swapped out and then swapped in again later, the internal signals of the RM are preserved by the HDL simulator as if the module had never been swapped out. The initialization logic of the DPR design cannot therefore be properly tested.

As FPGA vendor tools do not support the simulation of the reconfiguration process [2], it is non-trivial for designers to integrate the simulation of DPR with the environment used for verifying the rest of the system. Potential bugs, either arising from the reconfiguration process or from integrating DPR with the rest of the system, can not therefore be detected until the integrated design is tested on the target device.

B. Structural Analysis of Reconfiguration

Figure 1 illustrates three conceptual layers that can be identified for a DPR design: the application layer, the reconfiguration layer and the physical layer.

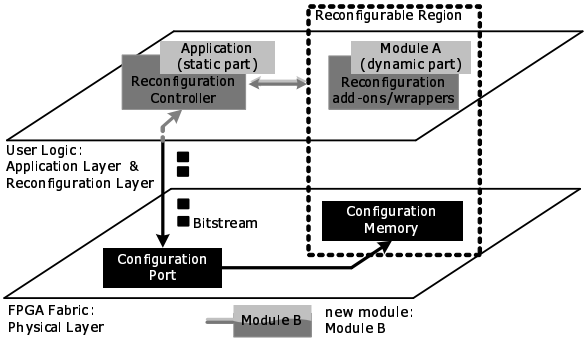

Figure 1. Structural analysis of DPR

- **Application and Reconfiguration Layer:** The application and reconfiguration layer comprise the user-defined logic of a DPR design. The application layer (lightly shaded blocks) implements the functional specification of the system. Its operation is highly dependent on the application. The reconfiguration layer (darkly shaded blocks) manages the reconfiguration process of the system. It typically includes a reconfiguration controller that transfers bitstreams from storage devices to the configuration port [11], or reads back configuration data for saving and restoring RM states [12]. This layer also includes reconfiguration add-ons, such as an RM wrapper that handshakes with the controller to properly pause the ongoing computation BEFORE reconfiguration [7].
- **Physical Layer:** The physical layer (black blocks) contains the FPGA resources such as the configuration port and the configuration memory. It also includes the bitstream which contains physically dependent information such as frame addresses. The details of this layer are described by vendor documentation such as [13] and are not captured by user-defined RTL. Therefore, they are usually excluded from functional simulation.

Given the layered DPR model of Figure 1, the reconfiguration process involves all three layers. It is therefore challenging to simulate user logic in the application and reconfiguration layer without relying on the physical layer. For example, the bitstream datapath belongs to the user logic in the reconfiguration layer, whereas the bitstream itself is in the physical layer. After it is written to the configuration port, the bitstream in turn affects user logic in the application and reconfiguration layer. Thus in simulation, hiding the physically dependent bitstream prevents the bitstream datapath from being verified and precludes the triggering condition of module swapping from being modeled. In short, hiding the physical layer in simulation sacrifices simulation accuracy.

However, physically independent simulation is essential for maintaining a high level of productivity when verifying design functionality. Firstly, as a general hardware design principle, functional verification should be independent of the results from the time-consuming implementation step. Moreover, physically independent simulation improves debugging productivity, during which designers are only interested in the correctness of user logic. Last but not least, physically independent simulation improves the throughput of the simulator because the design is modeled in less detail.

C. Ideal DPR Simulation Qualities

As mentioned above, hiding the physical layer sacrifices simulation accuracy, yet introducing physical information to simulation reduces productivity. The ideal simulation method should therefore *balance* between simulation-accuracy and physical independence. Furthermore, this balance is constrained by the desire for the simulated design to be *implementation ready*. Thus the simulated design should not make use of non-synthesizable components such as the Reconfiguration Condition Detector [5] to trigger module swapping.

The method should also be *reusable* for the sake of productivity. It should integrate well with mainstream tools, and it should be applicable to the large variety of DPR design styles.

IV. 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 redraws Figure 1 with all the physically dependent blocks (solid black boxes) replaced by the components of our simulation-only layer (open black boxes). These components are known as simulation-only artifacts or just artifacts. The solid boxes, on the other hand, represent user-defined real entities that are to be synthesized and mapped to FPGAs.

In ReSim, reconfiguration proceeds as follows: the user-defined reconfiguration controller transfers a simulation-only

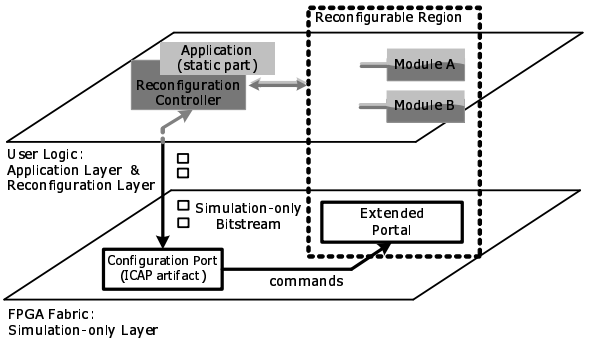

Figure 2. Using the simulation-only layer

bitstream (SimB, see Section IV-B) instead of a real bitstream from storage to the configuration port. The configuration port is replaced by the ICAP artifact, as a representative for possible configuration ports, to parse a SimB in simulation. By parsing the SimB, the ICAP artifact extracts RM and RR IDs, and sends commands to the Extended Portal, the substitute for the configuration memory. The Extended Portal swaps modules and injects errors according to the commands in the SimB (see Section IV-C). Reconfiguration is completed by sending the last word of the SimB to the ICAP artifact, which informs the Extended Portal to cease injecting errors and to connect the new RM to the static part. This process is physically independent and the rest of this section explains how our simulation architecture addresses the challenges of accurately simulating the reconfiguration process.

A. ReSim Overview

The ReSim library is built upon an existing High-level Verification Language (HVL) called SystemVerilog [14] and an open source SystemVerilog class library known as the Open Verification Methodology (OVM) [15]. As a result, ReSim is fully compatible with existing and mainstream EDA tools.

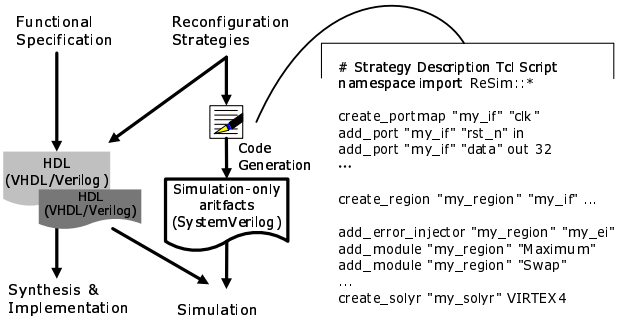

Figure 3. Development flow overview

Figure 3 provides an overview of the DPR design flow using ReSim. The proposed method takes the functional specification and a set of reconfiguration strategies as inputs. The reconfiguration strategies include names, sizes and connectivity of RRs and RMs. From both specifications, the designers create RTL code for the user logic and describe the reconfiguration strategies using a Tcl script. ReSim automatically generates the simulation-only artifacts based on the Tcl script. As an option, the designer can then edit the generated artifacts for design-/test-specific needs.

B. Simulation-only Bitstream

As RTL simulation cannot effectively make use of a real bitstream, ReSim substitutes a simulation-only bitstream to model the bitstream traffic and the triggering condition of module swapping. This SimB captures the essence of a real bitstream but its size is significantly reduced. Table I provides an example of a SimB that configures a new module.

Table I AN EXAMPLE OF SIMB FOR CONFIGURING A NEW MODULE

SimB	Explanation	Actions Taken
0xAA995566	SYNC Word	Start the "DURING Reconfiguration" phase
0x20000000	NOP	–
0x30002001 **0x01020000**	Type 1 Write FAR RRid=0x01 RMid=0x02 FAR=0x0000	Informs the Extended Portal to select the module id=0x02 to be the next active module in reconfigurable region id=0x01
0x30008001 **0x00000001**	Type 1 Write CMD WCFG	
0x30004000 **0x50000004**	Type 2 Write FDRI Size=4	
0x5650EEA7 0xF4649889 0xA9B759F9 0x4E438C83	Random SimB Word 0 Random SimB Word 1 Random SimB Word 2 Random SimB Word 3	Informs the Extended Portal to inject errors to both static and dynamic sides
0x30008001 0x0000000D	Type 1 Write CMD DESYNC	End the "DURING Reconfiguration" phase

Similar to a real bitstream, a SimB is composed of a header section, configuration data and a tail section. A SimB starts with a `SYNC` word (`0xAA995566`) and ends with a `DESYNC` command. In ReSim, these commands represent the start and end of the “DURING reconfiguration” phase.

A SimB contains commands to the ICAP artifact. The 6 consecutive words (bold in Table I) request that the current module in the RR with ID = 0x1 be replaced by the new module with ID = 0x2. Note that in a real bitstream the frame address (FAR) field is used to locate the physical postion of the target RR on an FPGA. In a SimB, on the other hand, the FAR field is replaced by physically independent numerical IDs for the new RM and the target RR.

The configuration data section of a real bitstream contains physically dependent information and is not relevant for functional simulation. So for SimB, this section is filled with random data and is reserved for further extension. Although the content is not used, writing the configuration data section to the ICAP model triggers the error injection operation (see Section IV-C). Thus the length of this section is determined by the designer for verification purposes and is typically significantly shorter than that of a real bitstream.

During a simulation run, the SimB mimics the bitstream traffic and helps the verification of the bitstream datapath, which may involve the transfer, compression, decryption and arbitration of bitstreams. Meanwhile, the commands in a SimB inform the simulation environment to swap modules and inject errors (see Section IV-C), thereby mimicing the triggering condition of a reconfiguration process.

C. Extended Portal

The Extended Portal is a physically independent substitute for the configuration memory of the emulated FPGA fabric (Figure 4). According to the principles of OVM, the Extended Portal is separated into a *module-based* part and a *class-based* part, which are connected through *SystemVerilog Interfaces* (shown as clouds, the vi symbol and the dashed link) [15].

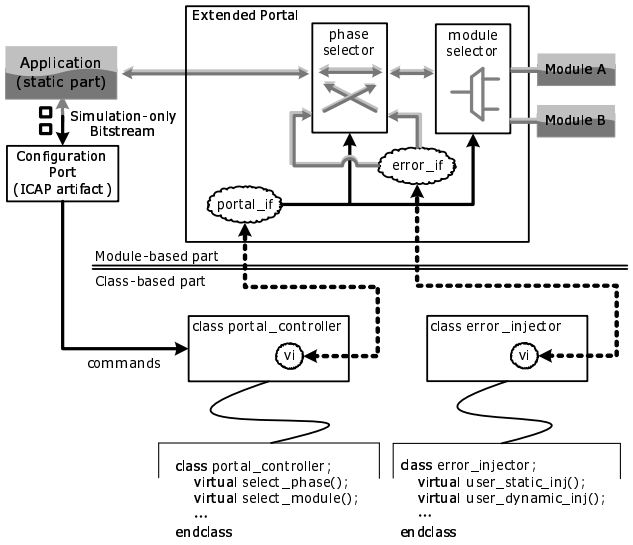

Figure 4. The Extended Portal

The module-based part instantiates the artifacts and is the infrastructure for modeling module swapping, spurious outputs and undefined initial states of a reconfiguration process. To model module swapping, the Extended Portal implements a MUX-like `module selector` to switch between RMs connected in parallel [4]. However in our work, the newly selected module is not connected to the static part until all configuration data of the SimB is written to the ICAP artifact. Thus the delay of reconfiguration is determined by the transfer of bitstreams instead of being zero or constant. The `module selector` (and the `phase selector` described below) is driven by the class-based part through a SystemVerilog Interface called `portal_if`.

The Extended Portal has two error injection mechanisms. Static error injection models the spurious outputs of the module undergoing reconfiguration [5] and helps to test the isolation logic of the DPR design. Dynamic error injection models the undefined initial state of the RM [3] and helps to test the initialization mechanism of the design. The class-based part drives both error sources through the `error_if`, and these errors are selected by the crossbar-like `phase selector`. In normal operation, the selected RM is connected to the static part. When the SimB is being written, the ICAP artifact informs the Extended Portal to connect error sources to both the static and dynamic parts. The error sources are disconnected and the normal static-to-RM connection is reconnected after the SimB is written.

The class-based part provides control and error sources to the module-based part of the Extended Portal. The default behavior of the `portal_controller` class is to control the `module selector` and the `phase selector` according to the commands in the SimB. The operations of the Extended Portal are thereby linked with the contents of the SimB. The default behavior of the `error_injector` class is to drive an undefined "x" value as an error source to both the static and dynamic parts of the system. All these default operations are defined by class member functions and can be overridden as desired, in the derived classes (see Section IV-D).

D. Reusability

In order to fulfill the reusability requirement of an ideal simulation method, ReSim allows users to parameterize artifacts for various DPR design styles. ReSim has three mechanisms to support extensions by users: code generation, class inheritance and factory overrides. As a result, ReSim can be flexibly customized to a user's design- or test-specific needs.

As already mentioned, the module-based artifacts are instantiated in the design hierarchy as if they were normal modules. In order to do that, the artifacts need to be parameterized with the hierarchy and the connectivity information of the user design. Code generation provides a set of Tcl APIs for users to parameterize the artifacts. By calling APIs in a Tcl script, the user describes the interfacing signals crossing the RR boundary, the affiliation of RMs and RRs, and the FPGA family used by the design (see Figure 3). ReSim generates the parameterized artifacts that can be correctly instantiated in the design hierarchy.

Class inheritance enables designers to change the default behavior of artifacts. For example, the `portal_controller` can be re-defined to ignore SimB and use a zero or constant reconfiguration delay if the designer finds it necessary to do so. Similarly, the designer can change the default "x" injection to design- or test-specific error sequences by extending the `error_injector` class. In contrast, the original works for static [5] and dynamic [3] error injection only support a constant "x" error value. These extensions can readily be implemented by redefining the *virtual* functions in the derived classes (see Figure 4). Class inheritance also improves the extensibility of ReSim. For example, the configuration port of a new FPGA family can be added to ReSim by defining a new class derived from `configuration_port_base`.

The last mechanism for customizing ReSim is called factory overrides [15]. After defining a derived class such as `my_error_injector`, ReSim allows the designer to replace all instances of the original `error_injector` class with the new one without editing any source code *within* the ReSim library. Designers can thus focus on defining the virtual functions in their derived classes without bothering about how ReSim is organized.

V. Case Studies

We demonstrate the value of ReSim via two case studies. The first is an in-house, general purpose DPR platform, known as XDRS. The second is a fast PCIe configuration reference design from Xilinx [16].

A. In-house XDRS system

The architecture of the XDRS system is similar to that of the Erlangen Slot Machine [17]. However, to focus on verification issues, we represent the essential elements of our platform as illustrated in Figure 5. This case study demonstrates that ReSim addresses the challenges of simulation accuracy without relying on the physical details of a design.

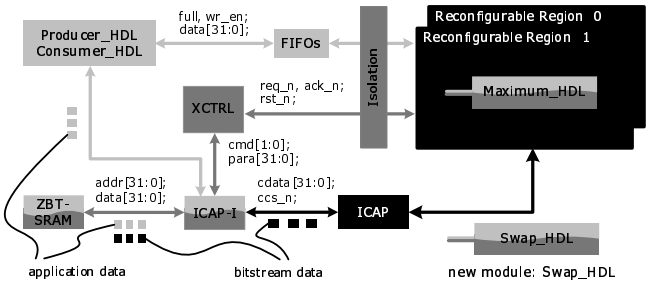

Figure 5. XDRS

The static part of the application layer is composed of `Producer` and `Consumer` modules, which keep feeding both reconfigurable regions with data from memory and copying output data back to memory. The reconfiguration controller, `XCTRL`, manages the reconfiguration process of the system. The memory (`ZBT-SRAM`) stores both bitstream and application data, and when one region is being reconfigured, the `ICAP-I` module [18] schedules the bitstream traffic and the application data of the other region. Thus the memory and the the `ICAP-I` forms the shared datapath between the application layer and the reconfiguration layer (shown as both lightly and darkly shaded boxes).

Figure 6 is a waveform obtained by simulating the XDRS system using ModelSim 6.5g. Here, an old `Maximum` module in Reconfigurable Region 1 of the XDRS system is reconfigured to a new `Swap` module using the SimB of Table I. The figure links the events on the waveform with the system diagram. The following discussion links the events with the DPR features mentioned in Section III-A (referred to using square brackets).

- **@t1**: `XCTRL` starts the reconfiguration and unloads the current module by asserting the `req_n`. As the `Maximum` module is busy, it blocks the `XCTRL` and doesn't assert `ack_n` until a few cycles later. Simulating these handshake signals assists the verification of the synchronization mechanism of the design.
- **@t2**: The first word of the SimB (`0xAA995566`) is transferred from the memory interface (the `A` & `DQ` signals) to the ICAP port (the `cdata` signal) [Bitstream Traffic], thereby marking the start of the "DURING reconfiguration" phase.
- **@t3**: After parsing the header section of the SimB, the ICAP artifact controls the Extended Portal to perform module swapping and error injection [Triggering Condition]. For module swapping, the new `Swap` module is selected by the `module selector`. Although such a selection is performed instantaneously, the new module is not connected to the static part until all configuration data of the SimB are written to the ICAP artifact. Thus the delay of such swapping is determined by the bitstream traffic [Module Swapping].
- **@t4**(1): When the configuration data section of the SimB is being written to the ICAP, the static part is connected to the error source by the `phase selector` so as to test the robustness of the isolation module. At the same time, the Extended Portal drives an undefined *"x" value* as an error source to all RR outputs (e.g. the `rd_en` signal in the `rr1` signal group) [Spurious Outputs]. Note that the 3 versions of IO signals (the `rr1`, `max_io` & `swp_io` signal groups) represent signals of the static side, the `Maximum` and the `Swap` modules respectively.
- **@t4**(2): Apart from the "x" injection to the static side, the Extended Portal also injects a sequence of *user-defined errors* to the `Swap` module so as to assist the verification of the initialization mechanism of XDRS AFTER reconfiguration (see t7). In particular, the Extended Portal starts the error injection sequence by de-asserting the `empty` signal. The `Swap` module responds to the `empty` signal by reading random data (`0xbfd505bd` on the `data` signal) into its internal register `indata` [Undefined Initial State]. This user-defined error sequence also demonstrates the reusability of ReSim.
- **@t5**: The `Consumer` module requests memory access. The `ICAP-I` scheduler therefore pauses the reconfiguration until the requested data are written to memory (see the long delay on the ICAP port). This example demonstrates that ReSim can integrate the simulation of DPR with that of the rest of the system.
- **@t6**: The `DESYNC` command is written to the ICAP thereby ending the "DURING reconfiguration" phase. All subsequent signal transitions on the ICAP interface until the next `SYNC` are ignored.
- **@t7**: `XCTRL` finishes the reconfiguration by asserting the reset signal (`rst_n`) to the design, which cleans all errors injected to the `Swap` module (the random data in the `indata` register) [Undefined Initial State].

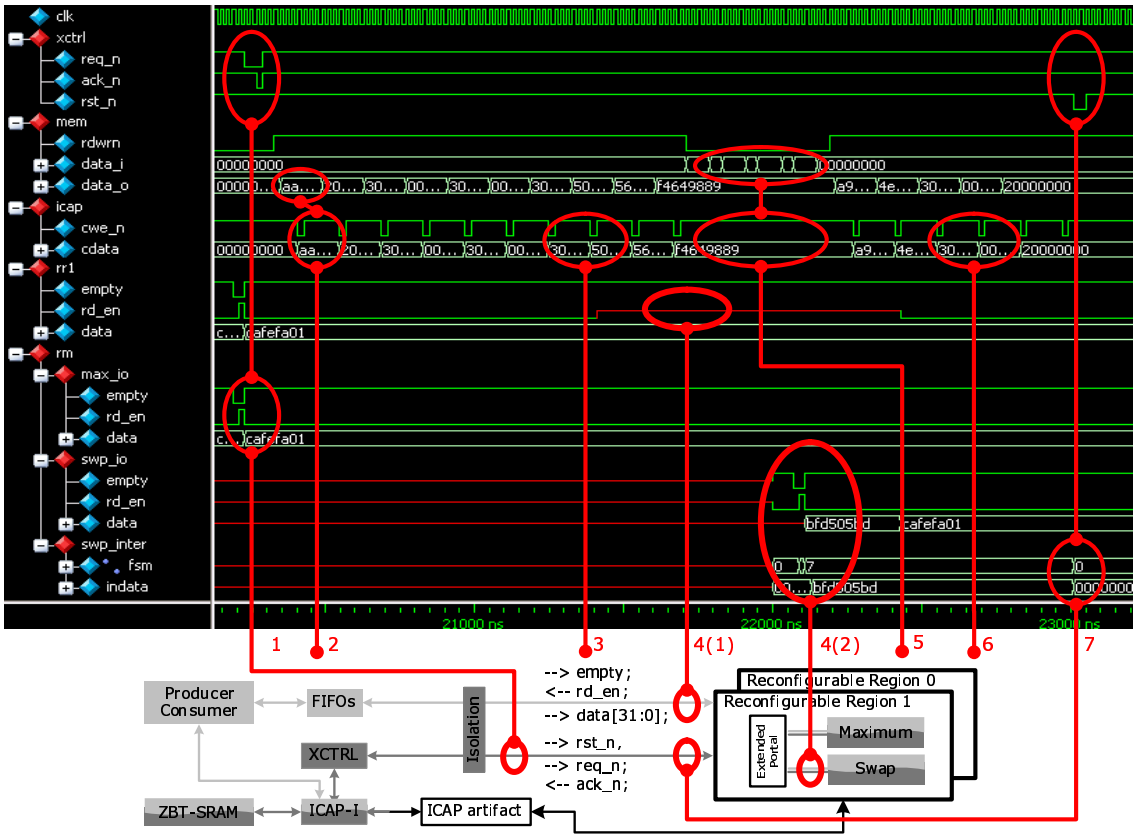

Figure 6. Waveform example

Simulation using ReSim was effective. By accurately simulating the synchronization, isolation and initialization mechanisms of the XDRS system BEFORE, DURING and AFTER reconfiguration, we detected dozens of timing errors in our design. For example, the `Isolation` module, which disconnects the RM DURING reconfiguration, resumes such connection one cycle too early AFTER reconfiguration. This error was easily identified because the "x" value injected by ReSim propagated to the static part in the erroneous cycle. Although such "x" injection is similar to [5], ReSim allows cycle-accurate simulation of the transition from DURING to AFTER reconfiguration, which is also essential to detect this type of error. The simulated and verified design was subsequently implemented without further error on an ADM-XRC-4 board with a Virtex 4 LX160 FPGA.

B. Fast PCIe configuration

The Fast PCIe configuration reference design applies DPR to address the tight PCIe startup timing specification [16]. At startup, the reference design only loads the light-weight PCIe endpoint logic as the static part to meet the PCIe timing requirements. The rest of the FPGA is then partially reconfigured with the core application logic through PCIe. The reference design targets an ML605 board with a Virtex 6 LX240T FPGA. This case study demonstrates that ReSim can readily be adapted to various design styles, especially designs of moderate complexity, and various FPGA families.

Figure 7 depicts a block diagram of the PCIe design. The PCIe endpoint receives and transmits PCIe transaction level packets (TLP) between the FPGA and the host. The TLPs include both application and bitstream data, and are dispatched to either the application or the reconfiguration controller by the `Switcher`. Thus the PCIe endpoint and the `Switcher` are shared by the application and reconfiguration layers. The application is a `Bus Master DMA` (BMD) design and is completely located inside the RR. The reconfiguration controller is the `PR Loader` module, which decodes the TLPs from the endpoint and writes the bitstream to the ICAP.

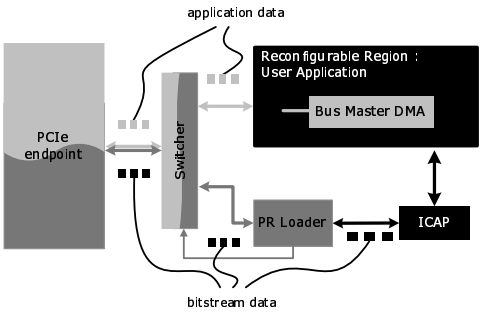

Figure 7. Fast PCIe configuration reference design, after [16]

We integrated the original simulation testbench with ReSim and simulated the design with ModelSim 6.5g.

- The complex bitstream datapath, including the shared PCIe endpoint, the shared `Switcher` and the dedicated `PR Loader`, was tested by simulating the bitstream traffic modeled by SimB.
- The `Switcher` isolates the RR from the static part, so that DURING reconfiguration, the undefined state of the BMD doesn't affect the PCIe endpoint. The robustness of the `Switcher` as an isolation module was tested against the errors injected to it.
- The `PR Loader` resets the BMD AFTER reconfiguration and the `Switcher` connects the BMD to the PCIe endpoint. This initialization process was tested against the errors injected to the dynamic side (i.e. the BMD) DURING reconfiguration.

The original testbench released along with the design also uses a dummy bitstream to verify the bitstream datapath. However, the contents of this dummy bitstream are meaningless and are therefore discarded after being written to the ICAP without triggering the swapping of the BMD module. As a result, the simulated BMD module operates whether or not the dummy bitstream is loaded correctly, and potential bugs on the bitstream datapath cannot be detected. ReSim models the triggering condition more accurately by swapping in the BMD module according to the SimB. Moreover, ReSim supports the error injection required to test the isolation and initialization mechanisms of the design.

The development workload for integrating ReSim with the original testbench includes: 150 lines of Tcl script for generating artifacts (120 of which are to describe the IO of the RR) and 10 lines of Verilog code to instantiate the artifacts in the original testbench. We left all class definitions at their defaults. Comparing with the original testbench, the simulation overhead of ReSim was proportional to the number of signals crossing the RR boundary as all boundary signals were multiplexed as opposed to being connected to the static part directly. The overhead was also proportional to the frequency of reconfiguration in a specific simulation run as each reconfiguration involves costs to swap modules and inject errors, and a scenario-dependent delay to transfer the SimB. For the PCIe example, the simulation overhead of ReSim was negligible.

VI. Conclusions and Future Work

As with modern ASIC designs, functional verification has become a significant challenge in large FPGA designs, and DPR adds a new dimension to the challenge. To perform simulation-based functional verification, it is essential to simulate the reconfiguration process as part of a full system simulation. This paper proposes using ReSim to simulate DPR by employing a simulation-only layer to model the physical layer of DPR designs. Our case studies show that RTL simulation of DPR using ReSim can achieve sufficient accuracy for functional verification without relying on the physical details of a design. Moreover, the ReSim library is reusable and can be adapted to various styles of DPR design.

Looking ahead, we plan to determine the optimal size for the SimB as a larger SimB may slow down the simulation and a small SimB may not achieve the desired level of test coverage. We are also looking at the simulation of DPR designs on Altera FPGAs. It is important to study how the idea of *cycle-accurate and physically independent* simulation can be adapted to FPGAs from various vendors.

References

[1] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, 2002.

[2] *Partial Reconfiguration User Guide (UG702)*, Xilinx Inc., 2010.

[3] L. Gong and O. Diessel, "Modeling Dynamically Reconfigurable Systems for Simulation-based Functional Verification," in *Field-Programmable Custom Computing Machines (FCCM), IEEE Symposium on*, 2011, pp. 9–16.

[4] W. Luk, N. Shirazi, and P. Y. Cheung, "Compilation tools for run-time reconfigurable designs," in *Field-Programmable Custom Computing Machines (FCCM), IEEE Symposium on*, 1997, pp. 56 – 65.

[5] I. Robertson, J. Irvine, P. Lysaght, and D. Robinson, "Improved Functional Simulation of Dynamically Reconfigurable Logic," in *Field Programmable Logic and Applications (FPL), International Conference on*, 2002, pp. 541–574.

[6] *The International Technology Roadmap for Semiconductors: 2009*, 2009. [Online]. Available: http://www.itrs.net/reports.html

[7] A. Raabe and A. Felke, "A SystemC Language Extension for High-Level Reconfiguration Modelling," in *Specification, Verification and Design Languages (FDL). Forum on*, 2008, pp. 55 – 60.

[8] A. Schallenberg, W. Nebel, A. Herrholz, and P. A. Hartmann, "OSSS+R: A Framework for Application Level Modelling and Synthesis of Reconfigurable Systems," in *Design, Automation and Test in Europe (DATE)*, 2009, pp. 970 – 975.

[9] A. Pelkonen, K. Masselos, and M. Cupak, "System-level Modeling of Dynamically Reconfigurable Hardware with SystemC," in *Parallel and Distributed Processing Symposium (IPDPS), International*, 2003, p. 8.

[10] A. V. Brito, M. Kuhnle, M. Hubner, J. Becker, and E. U. K. Melcher, "Modelling and Simulation of Dynamic and Partially Reconfigurable Systems using SystemC," in *VLSI (ISVLSI), IEEE Computer Society Annual Symposium on*, 2007, pp. 35 – 40.

[11] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hubner, and J. Becker, "A Multi-platform Controller Allowing for Maximum Dynamic Partial Reconfiguration Throughput," in *Field Programmable Logic and Applications (FPL), International Conference on*, 2008, pp. 535 – 538.

[12] H. Kalte and M. Porrmann, "Context Saving and Restoring for Multitasking in Reconfigurable Systems," in *Field Programmable Logic and Applications (FPL), International Conference on*, 2005, pp. 223 – 228.

[13] *Virtex-4 FPGA Configuration User Guide (UG071)*, Xilinx Inc., 2009.

[14] *IEEE Standard 1800-2005: SystemVerilog – Unified Hardware Design, Specification, and Verification Language*, The Institute of Electrical and Electronics Engineers, Inc., 2005.

[15] M. Glasser, *Open Verification Methodology Cookbook*, Mentor Graphics Corporation, 2009. [Online]. Available: http://www.mentor.com/cookbook

[16] S. Tam and M. Kellermann, *Fast Configuration of PCI Express Technology through Partial Reconfiguration (XAPP883)*, Xilinx Inc., 2010.

[17] C. Bobda, M. Majer, A. Ahmadinia, T. Haller, A. Linarth, and J. Teich, "The Erlangen Slot Machine: Increasing Flexibility in FPGA-based Reconfigurable Platforms," in *Field-Programmable Technology (FPT), International Conference on*, 2005, pp. 37 – 42.

[18] V. Lai and O. Diessel, "ICAP-I: A Reusable Interface for the Internal Reconfiguration of Xilinx FPGAs," in *Field-Programmable Technology (FPT), International Conference on*, 2009, pp. 357–360.