Functionally Verifying State Saving and Restoration in Dynamically Reconfigurable Systems

Lingkan Gong and Oliver Diessel
School of Computer Science and Engineering
University of New South Wales
Sydney, NSW, Australia 2052
{lingkang,odiessel}@cse.unsw.edu.au

ABSTRACT

Dynamically reconfigurable systems increase design density and flexibility by allowing hardware modules to be swapped at run time. Systems that employ checkpointing, periodic or phased execution, preemptive multitasking and resource defragmentation, may also need to be able to save and restore the state of a module that is being reconfigured. Existing tools verify the functionality of a system that is undergoing reconfiguration. These tools can also be employed if state is accessed using application logic. However, when state is accessed via the configuration port, functional verification is hindered because the FPGA fabric, which mediates the transfer of state between the application logic and the configuration port, is not being simulated. We describe how to efficiently simulate those aspects of the fabric that are used in accessing module state. To the best of our knowledge, this work is the first to allow cycle-accurate simulation of a system partially reconfiguring both its logic and state and a case study shows that our method is effective in detecting device independent design errors.

Categories and Subject Descriptors

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

General Terms

Verification

Keywords

FPGA, Dynamically Reconfigurable Systems, Verification, State Saving and Restoration

1. INTRODUCTION

The exponential increase of hardware design costs and risks have driven the electronic industry to use reconfigurable devices such as FPGAs as computing platforms. Compared with customized chips, hardware/software systems implemented on reconfigurable devices achieve more flexibility for potential upgrades or bug fixes over the product life-cycle, while sacrificing acceptable tradeoffs in power, area and performance. Dynamically Reconfigurable Systems (DRS) extend such flexibility by allowing partial reconfiguration of hardware modules at run time. Recent examples of DRS include a networked multiport switch [13], a software-defined radio [10] and a video-based driver assistance system [1].

Apart from reconfiguring module logic, state saving and restoration (SSR) is a common and sometimes essential requirement of DRS. For example, to recover from an error, a module can be restored to a saved checkpoint [7]. For periodic/phased applications, the system configures the required computational module for each period of execution and copies the results across periods [5]. In hardware multitasking, a module can be preempted and later resume execution from the interrupted state [11]. To defragment FPGA resources, modules can be relocated from their original locations to other areas of available resources [6]. In each of these scenarios, module state need to be saved and restored during or after partial reconfiguration.

The architectural flexibility of DRS introduce challenges in functionally verifying the system because conventional verification methods assume that hardware circuits are time-invariant. Although each configuration of a design can be verified using traditional methods, new approaches are required to verify the design *while* it is undergoing reconfiguration, during which modules need to be properly swapped and module state needs to be saved and restored [3].

Register Transfer Level (RTL) simulation is the most common method for verifying hardware design functionality. By visualizing selected signals in a waveform viewer, RTL simulation assists in debugging a design without implementing it. In order to simulate partial reconfiguration of module logic, existing methods use multiplexers to interleave mutually exclusive modules [9], and use simulation-only bitstreams to capture the cycle-accurate behavior of module swapping [4]. This paper extends ReSim [4], our previous work, to support the simulation of a design reconfiguring *both* module logic and module state. In particular, we consider designs that utilize the Configuration Port (CP) to save and restore state (e.g., [6] [11]). The key contributions of this paper are:

- Extending the original use of simulation-only bitstreams (SimB) with state data to enable cycle-accurate simulation of CP-based SSR in DRS designs.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FPGA'12, February 22–24, 2012, Monterey, California, USA.
Copyright 2012 ACM 978-1-4503-1155-7/12/02 ...$10.00.

- Providing a case study showing how ReSim assists in simulating and debugging CP-based SSR.

It should be noted that for CP-based SSR, RTL simulation cannot detect device-dependent bugs. Running the DRS on the target device is the only way to accurately verify the system in real-world environments. For example, the locations of the state data on the FPGA fabric can't be checked until they have been determined by implementation. However, it is non-trivial to visualize signals of the implemented design for the sake of debugging because extra effort is required to insert probing logic using vendor tools such as Chipscope. Moreover, the probing logic can only visualize a limited number of signals for a limited period of time. As a result, debugging the design on chip involves costly iterations of probing the relevant signals and reimplementing the design. It is therefore desirable to perform RTL simulation to identify and fix as many *device-independent* bugs as possible in the early stage of the design cycle, and leave the *device-dependent* part of the design to be tested on the target FPGA once the implementation has been completed.

The rest of this paper is organized as follows. Section 2 outlines the background for CP-based SSR. We discuss the architecture of the ReSim library in Section 3. Section 4 illustrates the use of our tool on a case study, while the final section concludes the paper.

2. BACKGROUND AND RELATED WORK

For DRS designs, state data include storage elements such as flip-flops and memory cells such as Block RAM (BRAM) located on the FPGA fabric. To save and restore module state, the designer needs to create a datapath to access the state. There are currently two approaches for creating such a datapath. The first method adds design specific application logic to read/write storage elements (e.g., [7]), whereas the other approach utilizes the configuration port of the FPGA to access flip-flops and memory cells (e.g., [6], [11]). Comparing the two methods, CP-based SSR is more general as the configuration port defines a uniform access method for saving and restoring state anywhere on chip. Therefore, it does not require any changes to module logic, and does not introduce any side effect to the system such as extra resource usage or tighter timing requirements. In contrast, customized state access logic needs to be carefully inserted for each module in the system.

In a typical DRS design (see Figure 1), multiple reconfigurable modules (RM) are mapped to a Reconfigurable Region (RR) and communicate with the static circuitry to perform the required tasks. For CP-based SSR, the static part starts state saving with a synchronization operation, which copies the RM state to the configuration memory of the FPGA fabric. The user design then reads the state data by sending frame addresses, i.e., the addresses of the configuration memory, to the configuration port and sampling the returned data. In this process, requests from the user design and responses from the fabric are all delivered in the form of bitstreams, a device-specific, packet-like data structure containing commands (e.g., synchronize, read) and parameters (e.g., frame addresses). The restoration process reverses the state saving procedure. Please refer to the configuration guide (e.g., [14]) for further details of CP-based SSR.

The primary difficulty of simulating and verifying CP-based SSR is the conflict between simulation accuracy and

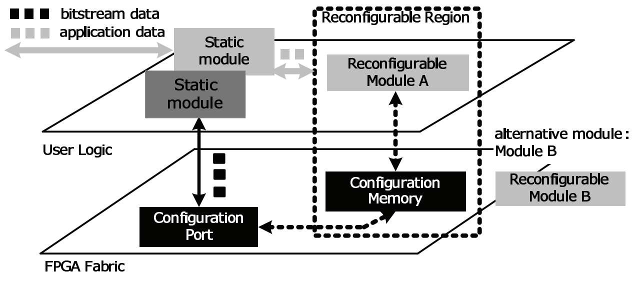


Figure 1: State saving and restoration in DRS

productivity. Because the state access datapath (i.e., the configuration memory and the configuration port depicted as black boxes in Figure 1) is part of the FPGA fabric, the most straightforward way to simulate CP-based SSR involves modeling the fabric. Unfortunately, vendor tools do not provide a simulation model for the fabric and even if such a model were available, simulating the design by modeling the fabric is performed at too low a level. For the sake of productivity, it is desirable that functional simulation should be independent of the FPGA fabric. This means it is better simulating user logic via HDL signals than simulating the bit settings in the configuration memory. It is also better to avoid the time-consuming implementation step in the iterative simulation-design cycle.

Alternatively, if accuracy can be sacrificed, CP-based SSR can be simulated without dependence on the FPGA fabric. The Dynamic Circuit Switch method modifies the RTL description of all storage elements in simulation so as to save and restore state [9]. By extending SystemC, ReChannel use call-back functions to access arbitrary variables in the modeled design [8]. However, the functionality added to the storage elements and the call-back functions do not exist on the target device and the simulated design therefore doesn't replicate what is implemented. Furthermore, these methods fail to model the interaction between user logic and the configuration port and thus only offer limited assistance in verifying CP-based SSR.

Although ReSim facilitates cycle-accurate simulation of configuration port accesses, it only supports the verification of a DRS reconfiguring its logic [4]. The simulation-only bitstream concept introduced in ReSim raises simulation accuracy by linking accesses to the simulated configuration port with module swapping. This work extends the application of the simulation-only bitstream by transferring module state via its contents. With this extension, ReSim is the first work to support cycle-accurate yet physically independent simulation of CP-based SSR.

3. RESIM LIBRARY

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 artifacts. It should be noted that the artifacts only model the aspects of the fabric that are essential to partial reconfiguration. In particular, the configuration bitstreams are replaced by simulation-only bitstreams, possible configuration ports are represented by an ICAP artifact, and the part of configuration memory to which each reconfigurable region (RR) is mapped is substituted by an Extended Portal.

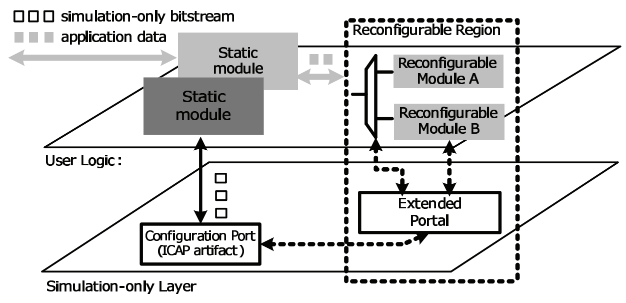
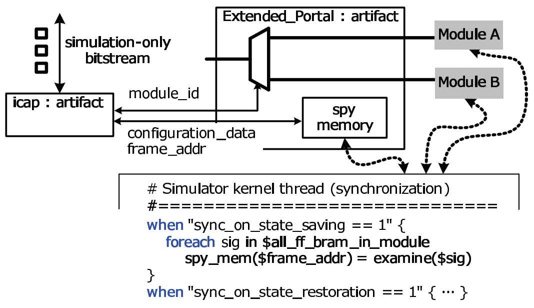


Figure 2: Using the simulation-only layer

To simulate partial reconfiguration of module logic, ReSim connects all RMs in parallel in the same way as existing approaches. However in ReSim, the selection of the active module is controlled by the Extended Portal, which is in turn triggered by writing a SimB to the ICAP artifact. SimBs are also used to simulate CP-based SSR. In particular, the static module transfers a SimB instead of a real bitstream to the ICAP artifact. By parsing the SimB, the ICAP artifact extracts the readback parameters, and controls the Extended Portal. The Extended Portal probes the RM state (typically modeled by HDL signals), and returns the state data to the ICAP artifact. Finally, the ICAP artifact returns the retrieved state data to the user design as a readback SimB. Restoring the state of a RM also utilizes the artifacts and SimBs to mimic the behavior of the FPGA fabric. However, a different SimB from the one for state saving is used, and the state data are copied back to the HDL signals. Using the SimB, ReSim mimics the behavior of the FPGA fabric during partial reconfiguration, and significantly improves the accuracy of simulating the user design.



```
# Simulator kernel thread (synchronization)
#=============================
when "sync_on_state_saving == 1" {
   foreach sig in $all_ff_bram_in_module
      spy_mem($frame_addr) = examine($sig)
}
when "sync_on_state_restoration == 1" { ··· }
```

Figure 3: Extended Portal

Figure 3 illustrates the architecture of the Extended Portal. Mimicing the FPGA fabric, the Extended Portal instantiates a spy memory as a substitute for the configuration memory to buffer the state data. To simulate state saving, for example, a Simulator-kernel thread (SKT) component probes RM signals when a synchronization operation is requested by the static module. The SKT uses simulator commands (e.g. ModelSim `examine`, `force` commands) to extract values of arbitrary HDL signals. The extracted signal values are then buffered to the spy memory, and are returned to the static module on each read from the ICAP artifact.

As the simulation-only layer doesn't rely on any physical details of the target device, the simulation of CP-based SSR is physically independent. The ReSim library is built upon the existing SystemVerilog language and Open Verification Methodology (OVM) [2], an open source SystemVerilog class library. As a result, ReSim is fully compatible with existing and mainstream EDA tools, although the current implementation of ReSim only supports ModelSim.

In order to avoid requiring the designer to create artifacts from scratch, ReSim automatically generates all artifacts based parameters defined in a script. The parameters include the name of each RM, the IO signals for each RR, the target FPGA family, etc. When executed, the script then creates parameterized artifacts by calling ReSim APIs.

4. A CASE STUDY

The hardware architecture of our case study is similar to a reference design from Xilinx [12] (see Figure 4). Although the original design only partially reconfigures the logic, we modified the software running on the microprocessor to support state saving and restoration as well. Our case study runs a periodic application. In each period of execution, the `xps_math` module is dynamically reconfigured with either an `adder` core or a `maximum` core as two alternative RMs. Apart from computation, each core maintains a `statistic` register, and its value is copied across configuration periods.

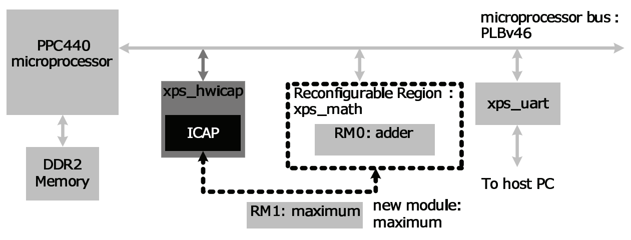
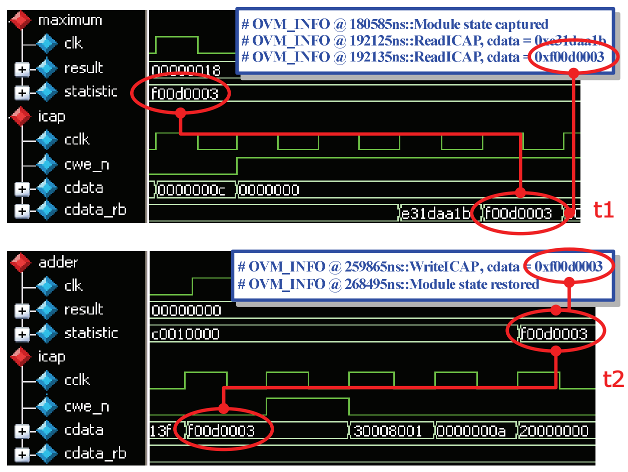


Figure 4: CP-based SSR case study, after [12]



```
# OVM_INFO @ 180585ns::Module state captured
# OVM_INFO @ 192125ns::ReadICAP, cdata = 0xe31daa1b
# OVM_INFO @ 192135ns::ReadICAP, cdata = 0xf00d0003
```

```
# OVM_INFO @ 259865ns::WriteICAP, cdata = 0xf00d0003
# OVM_INFO @ 268495ns::Module state restored
```

Figure 5: Waveform example

Figure 5 shows two waveform segments obtained by simulating the design using ModelSim 6.5g. Here, an old `maximum` core is reconfigured to a new `adder` core, and the value of the `statistic` register is copied across the two cores.

- **@t1**: Before reconfiguration, A readback SimB is written to the ICAP artifact requesting to save the value of the `statistic` register (0xf00d0003). The ICAP artifact is switched to read mode and returns the retrieved state data (see the `cdata_rb` signal).

- **@t2**: After reconfiguration, the `statistic` register of the new `maximum` core is initially zero. Then a restoration SimB containing the saved value is written to the ICAP artifact (see the `cdata` signal), after which the `statistic` register is restored to the desired value.

We detected dozens of bugs in our design using the above cycle-accurate simulation of CP-based SSR. For example, the restoration of the `statistic` register should have been skipped during the first round of execution as there was no previous execution period. The `statistic` register saved by the `xps_hwicap` driver was not properly returned to the periodic application, and was destroyed at the end of the function call. These bugs were detected as a consequence of incorrect values being restored to the `statistic` register. In the debugging process, it took less than a minute to complete one iteration of compiling and simulating the design on a Windows XP, Intel 2.53G Dual Core machine.

The simulated design was subsequently tested on an ML507 board with a Virtex-5 FX70T FPGA and we detected two device dependent bugs. As one example, the saved data were wrong because we failed to invert each bit of the saved state data as required by Virtex-5 FPGAs. In this cases, ReSim failed to mimic the exact behavior of the target device. Although this bug was trivial and would not have occurred with experienced designers, we used 3 iterations to trace the cause of this bug. Each iteration involved inserting new probing logic and re-implementing the design, and took 53 min to complete on the same machine used for simulation.

In this study, the workload for integrating ReSim with the testbench included: 50 lines of Tcl script for generating the artifacts, 10 lines of Verilog code to instantiate the artifacts, and 4 ModelSim commands to start the SKT. In contrast, the workload for modifying the `xps_hwicap` driver to support CP-based SSR included 1300 lines of C code. The simulation overhead of ReSim is proportional to the number of registers to be saved and restored. The overhead is also proportional to the frequency of SSR in a simulation run as each saving and restoration triggers a synchronization between the spy memory and the HDL signals. For our case study, the simulation overhead was negligible.

5. CONCLUSIONS AND FUTURE WORK

Functional verification has become a significant challenge in DRS designs. It is therefore essential to perform RTL simulation of the reconfiguration process, including the saving and restoration of state, as part of a full system simulation. This paper proposes extending ReSim [4] to simulate CP-based SSR. In particular, we use simulation-only bitstreams to correlate the accesses of a simulated configuration port with the state of a simulated module.

As ReSim abstracts out the details of the FPGA fabric, the simulation-only layer can be regarded as a vendor-independent device. Simulation can be thought of as functionally verifying a design on such a vendor-independent device. Although RTL simulation is only an approximation to the target device, our case study demonstrated that with negligible development and simulation overhead, simulation using ReSim captured a reasonable number of bugs and avoided most of the costly iterations of using Chipscope. Meanwhile, as ReSim only offers limited help in detecting device dependent bugs, it is desirable that a DRS be separated into a device dependent part and a device independent part. ReSim can then assist in verifying the device independent part, and the rest of the design should be tested on the target device.

6. REFERENCES

[1] Claus, C., Zeppenfeld, J., Muller, F., and Stechele, W. Using Partial-Run-Time Reconfigurable Hardware to accelerate Video Processing in Driver Assistance System. In *Design, Autom. Test in Europe* (2007), pp. 1–6.

[2] Glasser, M. *Open Verification Methodology Cookbook*. Mentor Graphics Corporation, 2009.

[3] Gong, L., and Diessel, O. Modeling Dynamically Reconfigurable Systems for Simulation-based Functional Verification. In *Field-Prog. Cust. Comput. Machines* (2011), pp. 9–16.

[4] Gong, L., and Diessel, O. ReSim: A Reusable Library for RTL Simulation of Dynamic Partial Reconfiguration. In *Field-Prog. Tech.* (2011).

[5] Jiang, Y.-C., and Wang, J.-F. Temporal Partitioning Data Flow Graphs for Dynamically Reconfigurable Computing. *IEEE Trans. VLSI Syst. 15*, 12 (2007), 1351–1361.

[6] Kalte, H., and Porrmann, M. Context Saving and Restoring for Multitasking in Reconfigurable Systems. In *Field-Prog. Logic and App.* (2005), pp. 223–228.

[7] Koch, D., Haubelt, C., and Teich, J. Efficient Hardware Check pointing: Concepts, Overhead Analysis, and Implementation. In *Field-Prog. Gate Arrays* (2007), pp. 188–196.

[8] Raabe, A., Hartmann, P. A., and Anlauf, J. K. ReChannel: Describing and Simulating Reconfigurable Hardware in SystemC. *ACM Trans. Design Autom. Electr. Syst. 13*, 1 (2008), 15.

[9] Robertson, I., and Irvine, J. A Design Flow for Partially Reconfigurable Hardware. *ACM Trans. Embedded Comput. Syst. 3*, 2 (2004), 257–283.

[10] Sedcole, P., Blodget, B., Anderson, J., Lysaght, P., and Becker, T. Modular Partial Reconfiguration in Virtex FPGAs. In *Field-Prog. Logic and App.* (2005), pp. 211–216.

[11] Simmler, H., Levinson, L., and Manner, R. Multitasking on FPGA Coprocessors. In *Field-Prog. Logic and App.* (2000), pp. 121–130.

[12] Xilinx Inc. *Partial Reconfiguration of a Processor Peripheral (UG744)*, 2009.

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

[14] Xilinx Inc. *Virtex-5 FPGA Configuration User Guide (UG191)*, 2010.