2013 IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum

RTL Simulation of High Performance Dynamic Reconfiguration: A Video Processing Case Study

Lingkan Gong [1], Oliver Diessel [1], Johny Paul [2], Walter Stechele [2]
[1] School of Computer Science and Engineering, University of New South Wales
{lingkang,odiessel}@cse.unsw.edu.au
[2] Institute for Integrated Systems, Technische Universität München
{johny.paul,walter.stechele}@tum.de

***Abstract*—Dynamically Reconfigurable Systems (DRS) allow hardware logic to be partially reconfigured while the rest of the design continues to operate. For example, the AutoVision driver assistance system swaps video processing engines when the driving conditions change. However, the architectural flexibility of DRS also introduces challenges for verifying system functionality. Using AutoVision as a case study, this paper studies the use of a recent RTL simulation library, ReSim, to perform functional verification of DRS designs. Compared with the conventional Virtual Multiplexing approach, ReSim more accurately simulates the AutoVision system before, during and after reconfigurations. With trivial development and simulation overhead, ReSim assisted in detecting significantly more bugs than found using Virtual Multiplexing. To the best of our knowledge, this paper is the first significant effort towards functionally verifying a cutting-edge, complex, real-world DRS application.**

I. 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 FPGAs as mainstream computing platforms. Compared with customized chips, hardware/software systems implemented on reconfigurable devices achieve shorter time-to-market and are more amenable to upgrades and bug fixes over the product life-cycle.

Dynamically Reconfigurable Systems (DRS) extend the flexibility of FPGAs by allowing partial reconfiguration of hardware modules at run time. By mapping multiple reconfigurable hardware modules to the same physical region of the FPGA, such systems are able to time-multiplex their modules at run time and adapt themselves to changing execution requirements. For example, the video-based driver assistance system of the AutoVision project swaps video processing engines to adapt to changing driving conditions (e.g., highway, countryside, urban traffic, tunnel) [1]. Recent development of the project extends the system flexibility to reconfiguring video engines during the processing of a single frame [2].

For modern hardware designs, either FPGA-based or ASIC-based, functional verification has become a significant challenge and IP reuse is one of the design methodologies that reduces verification effort [3]. By integrating or re-integrating proven IPs from previous projects, design productivity is significantly improved. However, while thoroughly verified sub-modules and IPs are essential, they do not guarantee the correctness of the integrated design [4]. For dynamically reconfigurable designs on FPGAs, apart from verifying each configuration of the system, it is therefore essential to test and debug the *integrated* DRS design, including the behavior immediately before, during and after partial reconfiguration [5]. Unfortunately, FPGA vendors such as Xilinx do not provide methods for simulating the reconfiguration process [6]. The difficulty of simulating partial reconfiguration thereby adds to the difficulties of simulating an *integrated* DRS design and, ultimately, to the application of IP-reuse design methodology.

Using the AutoVision driver assistance system [1], [2] as a case study, this paper aims to study the application of IP-reuse methodology to the design and verification of DRS designs. In particular, we reused proven IPs from the original design and optimized the design for better performance and resource utilization. We used Register Transfer Level (RTL) simulation to test and fix any potential bugs *relating to the reconfiguration process* in the re-integrated AutoVision system. The main contributions of this case study include:

- An analysis of the challenges and opportunities for applying the IP-reuse design and verification methodology to DRS designs; and
- An assessment of existing simulation approaches, Virtual Multiplexing [7] and ReSim-based simulation [8], in the functional verification of DRS designs.

To the best of our knowledge, this paper is the first significant effort towards functionally verifying a cutting-edge, complex, real-world DRS application. We aim to inspire researchers and designers to consider the value of RTL simulation to facilitate functional verification of DRS designs, thereby reducing, if not eliminating, reliance on costly on-chip debug cycles.

The rest of this paper is organized as follows. Section 2 outlines related efforts in functionally verifying DRS. Section 3 provides an overview of the Design Under Test (DUT), the Optical Flow Demonstrator, and formulates the verification goal of this paper. Section 4 illustrates the simulation environment and methods. Section 5 reports on the verification results and the last section concludes the paper.

978-0-7695-4979-8/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPSW.2013.79

II. Background and Related Work

The most common approach to simulating Dynamic Partial Reconfiguration (DPR) has been to insert a multiplexer into the design to interleave the communication between reconfigurable modules connected in parallel [7]. As the inserted multiplexer does not exist in the implemented design, it is known as a virtual multiplexer. This method is the basis for more recent efforts in simulating DPR. However, it only models module swapping and fails to simulate other aspects of DPR, such as module isolation and bitstream retrieval, which is increasingly handled on-chip.

The more recent Dynamic Circuit Switch method [9], [10], [11] improves the simulation accuracy of DRS designs in various aspects. It adds simulation-only artifacts to the RTL code of DRS designs in order to deactivate, switch and activate hardware tasks. It injects undefined "X" values to the static region to mimic the spurious outputs from modules undergoing reconfiguration. However, it still assumes that the reconfiguration delay is zero or a constant number and does not simulate bitstream traffic. Furthermore, reconfiguration is triggered by monitoring designer-selected signals in the RTL code whereas on real FPGAs, module swapping is triggered by bitstream transfer. Therefore, bugs introduced by the transfer of bitstreams and the triggering of module swapping can not be detected until the implemented design is tested on the target FPGA [5].

ReChannel [12] is a SystemC-based, open source library to model DPR. The 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 (e.g. module isolation, bitstream retrieval, accurate reconfiguration delay, triggering of module swapping) of a design are not modeled or verified.

The primary drawback of the existing work is that it fails to provide the accuracy required to verify the design undergoing reconfiguration since the interpretations and the manipulations of bitstreams are not simulated. Unfortunately, simulating the bitstream traffic involves interpreting the bit-level configuration memory settings for the module to be configured, which undesirably exposes the details of the FPGA fabric to the verification of the user design. Our recent work, ReSim [8], [13], improves the simulation accuracy by using simulation-only bitstreams as substitutes for the real bitstreams so as to accurately model the transfer of bitstreams and the timing of reconfiguration, and is the first work to support the cycle-accurate RTL simulation of the complete reconfiguration process of an *integrated* DRS design. This paper presents a case study of applying ReSim to the verification of a real-world DRS design created using various IPs, and compares ReSim with the traditional MUX-based simulation approaches.

It should be noted that DRS designs can also be tested and validated on the target FPGA under real-world conditions. On-chip debugging requires designers to insert probe logic using vendor tools such as Chipscope [14] and to re-implement the design every time a different set of user design signals need to be probed. The debug turnaround time is therefore at least as long as the time-consuming implementation stage. Moreover, since probing logic can only visualize a limited number of signals for a limited period of time, on-chip debugging typically requires many more iterations to identify the source of a bug than simulation requires. This paper compares on-chip debugging with simulation using our own development experience.

III. The Optical Flow Demonstrator

The AutoVision driver assistance system uses an Optical Flow algorithm to determine the speed and distance of moving objects (e.g. cars) on the road so as to identify potentially dangerous driving conditions [2]. As illustrated in Figure 1, the demonstrator uses reconfigurable video processing engines to accelerate the Optical Flow algorithm. In particular, each input video frame is first processed by a Census Image Engine (CIE) to generate a feature image. The reconfiguration controller (`IcapCTRL`) then reconfigures the CIE engine with a Matching Engine (ME), which compares two consecutive feature images and computes the motion vectors. Significantly, the hardware must be reconfigured twice per video frame to sustain the real-time throughput of the application. Finally, the embedded software running on an on-chip PowerPC processor draws the motion vectors and outputs the video frame. In the system, the engines and the reconfiguration controller transfer video data and bitstreams via a Processor Local Bus (PLB), whereas the software sets parameters of the engines and the reconfiguration controller via a Device Control Register Bus (DCR).

To perform the study reported here, both the hadware and the software of the Optical Flow Demonstrator were slightly modified from the original design, but can be viewed as a re-integration of the original design. In terms of hardware architecture (see Figure 1), the reconfiguration controller was modified to access the memory via the PLB bus instead of the original NPI interface. As far as the system software is concerned (see Figure 2), the processing flow was pipelined to better exploit the parallelism of the system. In particular, the PowerPC processor draws motion vectors for the previous frame while the engines are processing the current frame. The start, end and reconfiguration of the video processing engines are controlled by Interrupt Service Routines (ISR) that are independent of the main software functions.

The primary focus of this paper is the verification of the reconfiguration machinery (moderately shaded parts of Figure 1), which is defined as hardware and/or software that enables partial reconfiguration. In particular, the DUT instantiates a reconfiguration controller (`IcapCTRL`) to transfer bitstreams from memory to the FPGA's ICAP port. To avoid the propagation of erroneous signals from the region undergoing reconfiguration, an `Isolation` module was used to isolate

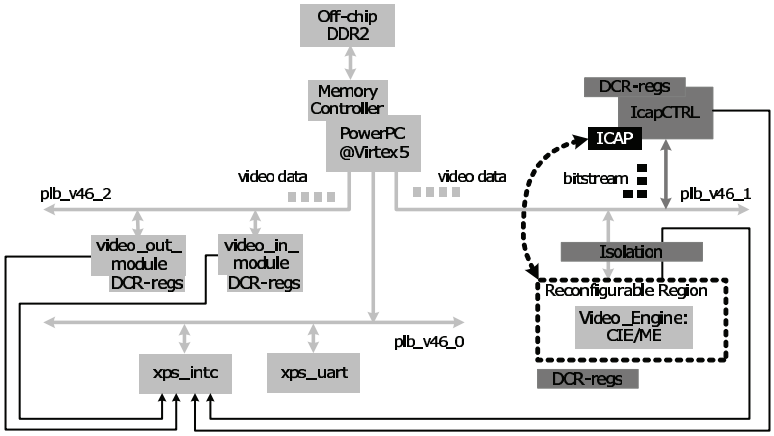

Figure 1. The hardware architecture of the Optical Flow Demonstrator

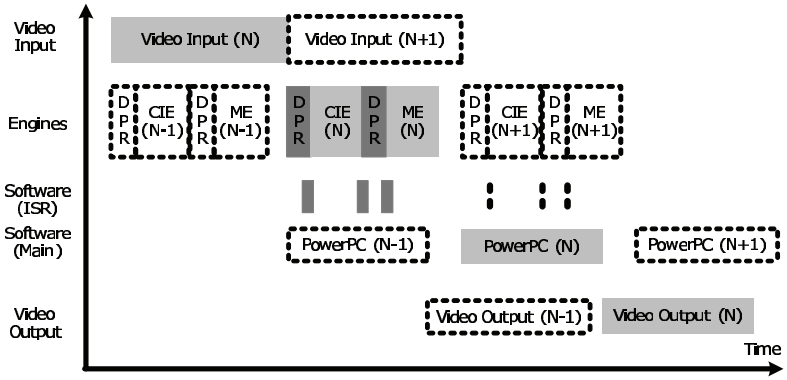

Figure 2. The processing flow of the Optical Flow Demonstrator

the engines during reconfiguration. Furthermore, the DCR registers were moved from inside the engines to the outside so as to avoid breaking the DCR daisy chain during reconfiguration. Since the DUT is an integration of various proven IPs from the original design, we focused on the verification of system *integration*. In particular, we aimed to verify that:

- The modified reconfiguration machinery (i.e., the PLB-based `IcapCTRL`) and its software driver were correct and were correctly *integrated* with the rest of the system hardware and software; and that
- The interrupt-driven reconfiguration sequence (i.e., the start and the end of partial reconfiguration in the pipelined processing flow) was working and was correctly *integrated* with the rest of the system hardware and software.

IV. Simulation Environment

The simulation environment we used had the same architecture as the DUT (see Figure 1), although some of the hardware modules were replaced with Verification IPs (VIP) [3]. A VIP differs from an IP since it only assists the verification of other modules and is therefore typically written in high-level languages such as C/C++/SystemC. For example, since the simulation environment does not have a camera or a display, the video input and output modules were replaced with SystemC VIPs to mimic the input/output video stream. In particular, Video VIPs read/write frames from/to the video files on disk and transfer to/from the simulated main memory via cycle-accurate PLB bus operations. On the other hand, since the processor netlist provided by Xilinx is too slow to run any realistic software, we used a PowerPC Instruction Set Simulator (ISS) [15] to emulate the operations of the target processor so that the software could run as if it were running on a real processor. We studied two methods for simulating DPR in the autoVision system: a traditional Virtual Multiplexing and a ReSim-based simulation method.

A. Virtual Multiplexing

We first used the traditional Virtual Multiplexing method [7] to simulate the reconfiguration process in the demonstrator. This method requires creating an `Engine_wrapper` to instantiate both engines (see Figure 3). A multiplexer inside the `Engine_wrapper` selects one active engine at a time and DPR is simulated by switching the multiplexer between engines. The selector of the virtual multiplexer is controlled by

an `Engine_Signature_Register`, which is controlled by software via the DCR bus.

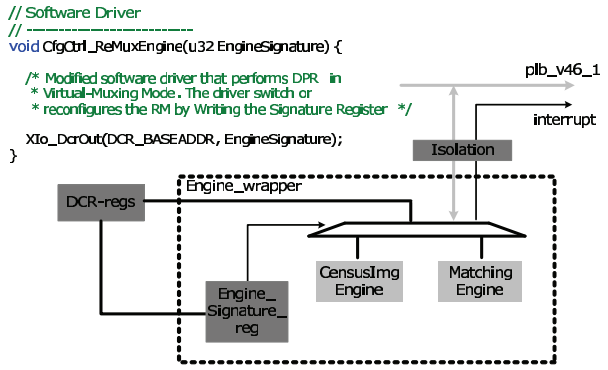

Figure 3. Virtual Multiplexing

Although Virtual Multiplexing is able to mimic the intra-frame reconfiguration of the AutoVision system, in simulation, DPR is triggered by software instead of by transferring a bitstream to the internal configuration port (`ICAP`). The `IcapCTRL` module is instantiated in the design but is not used in simulation. Therefore, the software and hardware tested in simulation do not match what is actually implemented. Furthermore, since multiplexing the simulated engines does not generate erroneous signals, as implemented designs might, the isolation mechanism (i.e., the `Isolation` module) is not tested in simulation. Using Virtual Multiplexing to simulate DPR therefore only provides limited assistance in debugging DPR of the AutoVision system. This study used it as a baseline simulation environment for simulation and debugging.

B. ReSim

ReSim-based simulation allows more accurate modeling of DPR compared with Virtual Multiplexing. As illustrated in Figure 4, the user design of ReSim-based simulation is composed of the RTL of the DUT (i.e., the engines and the reconfiguration machinery, illustrated as lightly and moderately shaded blocks in the figure), and is kept the same for both simulation and implementation. On the other hand, ReSim-based simulation uses simulation-only artifacts (i.e., open boxes in the left half of the figure) as substitutes that mimic the behavior of the corresponding features of the target FPGA (illustrated as darkly sharded boxes in the right half of the figure) during the reconfiguration process. In particular, simulation-only bitstreams (SimB) are substitutes for real configuration bitstreams, possible configuration ports are represented by an ICAP artifact, and the Extended Portal is an artifact that mimics the behavior of the part of configuration memory to which a reconfigurable region (RR) is mapped. These artifacts are not instantiated in the implemented design and adding these artifacts does not change the reconfiguration machinery of the design. In essence, these artifacts substitute for and abstract away the details of the corresponding device components.

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 FA=0x01020000	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	Word 0 starts error injection Word 3 ends error injection and triggers module swapping
0x30008001 0x0000000D	Type 1 Write CMD DESYNC	End the "DURING Reconfiguration" phase

In ReSim-based simulation, the reconfiguration controller `IcapCTRL` transfers a SimB to the ICAP artifact. A SimB mimics the impact of a real bitstream on the simulated user design. 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 (`0xAA995566`) and ends with a `DESYNC` command. However, instead of containing bit-level configuration memory settings for the module to be configured, as found in a real bitstream, a SimB contains numerical IDs for the module to be configured and the target reconfigurable region. For example, the SimB in Table I requests that the current module in the RR with ID = 0x1 be replaced by the new module with ID = 0x2 (see the bold SimB entries in the table). In addition, the length of a SimB is defined by the designer. For example, the designer can use a short (e.g. ~100 words) SimB to reduce the simulation-debug turnaround time, can adjust the length to test various scenarios of the bitstream transfer mechanism (e.g., FIFO overflow/underflow), and can set the length of a SimB to be the same as a real bitstream to achieve the maximum level of accuracy in simulating the reconfiguration process.

While the SimB is being written, an Error Injector artifact is connected to the static region and mimics spurious outputs from the region undergoing reconfiguration. By default, ReSim injects undefined “x” values to all outputs of the RR, which is similar to X injection proposed by DCS [11]. Furthermore, for advanced users, the error sources in ReSim can also be overridden for design-/test- specific purposes using the object-oriented programming techniques, thereby providing extra flexibility to the designer compared with [11].

The ICAP artifact interacts with the user design (i.e., the reconfiguration controller) and parses the SimB. After the SimB has been completely written to the ICAP artifact, the Extended Portal drives the multiplexer according to the engine ID extracted from the SimB, and connects the newly selected

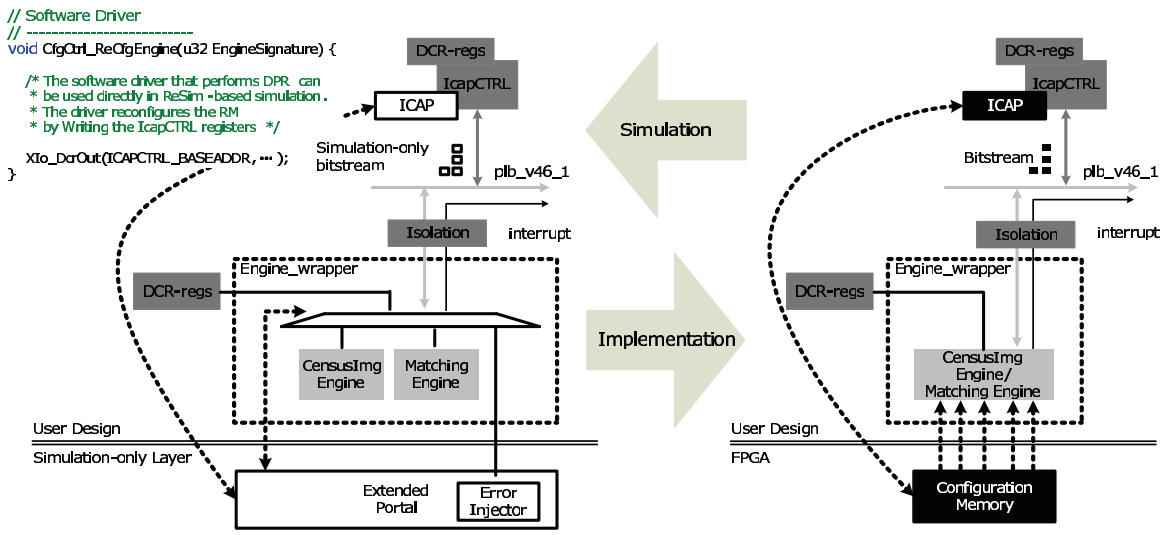

Figure 4. ReSim-based simulation

engine. Compared with Virtual Multiplexing, the benefits of using ReSim are as follows:

- Since ReSim uses a SimB to replace real bitstreams, the bitstream transfer datapath (e.g., the RTL code of the `IcapCTRL`) is verified in simulation. The delay of reconfiguration is determined by bitstream transfer instead of being zero or a constant. A bug in the bitstream transfer datapath prevents the new engine from being swapped in simulation.
- Although ReSim connects all engines in parallel, like Virtual Multiplexing, the selection of engines is triggered by the SimB. Thus, ReSim-based simulation does not use the indirect mechanism of the "`Engine_Signature`" register and the software driver that controls the reconfiguration process does not have to be changed for simulation purposes. Nor does ReSim rely on non-synthesizable components/mechanisms, such as the Reconfiguration Condition Detector found in [10], to trigger module swapping. Therefore, ReSim-based simulation verifies the real design intent.
- Since errors are injected into the static region when the SimB is being written to the ICAP, the isolation logic and the software driver that controls such logic is verified in simulation. If, for example, the designer failed to move the DCR registers out of the engines, the DCR daisy chain would break as a consequence of the injected errors propagating to the DCR bus of the static region.

As ReSim abstracts away the details of the FPGA fabric, the simulation-only layer can be regarded as a vendor independent device, and simulation can be thought of as functionally verifying a design on such a vendor independent device. ReSim-based simulation thereby captures the interaction between the user design and the FPGA fabric (e.g., the timing of reconfiguration events), and balances the need for accuracy with the requirement for physical independence [8]. However, mismatches between the simulation-only layer and the target FPGA can lead to bugs that would remain undetected by ReSim. For example, flipping a configuration bit of a real bitstream may cause incorrect logic to be reconfigured to the FPGA. Since a SimB does not model the individual configuration bits that are found in a real bitstream, it can not detect bugs that incorrectly change the configuration data of a bitstream. Generally speaking, since ReSim is physically independent, it can only assist in detecting bugs that are not related to the FPGA fabric of the target device.

V. Development Progress and Results

We ran the simulations using ModelSim 6.5g on a Windows XP, Intel 2.53GHz Dual Core machine. The simulation performance of the testbench is summarized in Table II. In the pipelined processing flow, the video engines are the bottleneck of the system throughput (see Figure 2). The time required to process one frame is determined by the video engines and is therefore the sum of the execution time of CIE (1.1ms), ME (1.4ms), 2 DPR intervals (<0.1ms) and 3 ISR intervals (0.5ms).

Overall, it took 11 minutes to simulate the processing of one frame. The Elapsed Time of each execution stage increases with the Simulated Time and also increases if the simulated design has more signal activities. For example, since the CIE engine has more signal fliping activities, the Elapsed Time to simulate 1.1ms of CIE operations (6min) is longer than the time to simulate 1.4ms of ME operations (4.5min). Since all bugs identified in this study (see Section V-A) were detected within the first 2-4 frames, the debug turnaround time for simulation was therefore at most 44 minutes per iteration. It should be noted that since the length of a SimB (4K words) is significantly smaller than that of the real bitstream

(129K words), the Simulated Time and the Elapsed Time for performing DPR can be ignored.

Table II TIME TO SIMULATE ONE VIDEO FRAME

	Simulated Time (ms)	Elapsed Time (min)
CensusImg Engine	1.1	6
Matching Engine	1.4	4.5
PowerPC Interrupt Handler	0.5	0.5
Dynamic Partial Reconfiguration	< 0.1	negligible
Overall	3.0	11

The simulation overhead of ReSim is trivial. Using ModelSim profiling tools, we found that 1.4% of simulation time was spent in the `Engine_Wrapper` multiplexer, which was triggered whenever the engine IOs toggled. Other simulation-only artifacts (e.g., Extended Portal, Error Injectors) consumed just 0.3% of the simulation time, but this would increase if a design were to perform DPR more frequently.

A. Development Workload and Bugs Detected

Figure 5 illustrates the progress of verification in terms of Lines of Code (LOC) changed and bugs detected. The LOC numbers were reported by a version control tool and included design source such as HDL, scripts, software (*.c, *.h), constraint files, development log files and project files. It should be noted that since we used the Embedded Development Kit (EDK) framework [16], some of the design source files were generated by the tool and also contributed to the LOC numbers. As a result, the LOC numbers should be thought of as reference data that indicate the relative development effort. The development had a few important milestones:

- By the end of Week 3, the designer finished assembling the modified Optical Flow Demonstrator (see Figures 1 and 2) as well as an initial testbench. Since design files of the Optical Flow Demonstrator and the legacy VIPs (i.e., the Video VIPs and the PowerPC ISS) were initially added to version control, the reported LOC number were very high. However, most design files were reused from previous projects, and the designer's workload involved re-integrating legacy components and simulating sanity checks such as a "hello world" program and a "camera to VGA display" application.
- The real verification work began in Week 4, when the designer started using Virtual Multiplexing to simulate the system. While not able to identify many DPR bugs, Virtual Multiplexing was helpful in detecting most of the bugs in the static design. In particular, between Week 6 and Week 9, the designer fixed 3 extremely costly bugs in the static region and Virtual Multiplexing-based simulation passed. Apart from bug fixes, the design itself was stable during this period. The LOC numbers were contributed to by changes to the testbench aimed at improving the simulation throughput and to reducing debug turnaround time.
- In the last 2 weeks, the designer used ReSim to simulate DPR and detected 2 software bugs and 6 DPR bugs in the system. The simulation passed at Week 11, after which no more bugs were detected.

The majority of the workload was incurred in setting up the baseline simulation environment and debugging the static part of the design (Week 1 to Week 9). Given the baseline simulation environment, the extra workload of using Virtual Multiplexing was small, which involved "hacking" the system hardware (250 LOC) and software (100 LOC) so as to add the `Engine_Wrapper` multiplexer. The workload involved in simulating the design with ReSim was also trivial. In particular, the designer needed to create a Tcl script (80 LOC) to generate simulation-only artifacts, and write HDL code (50 LOC) to instantiate the artifacts in the testbench. Since the modifications do not involve the design itself, ReSim-based simulation verifies design intent, covers all aspects of DPR, and tests system *integration*.

Table III lists a few DPR bugs detected in the DUT. 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.dpr.4) and software/hardware parameters (e.g., bug.dpr.5). Therefore, it was found to be highly desirable to have an integrated simulation environment to test the entire system.

Virtual Multiplexing was used at the beginning of the project. Since the `IcapCTRL` module was not exercised, we were unable to detect bugs (e.g. bug.dpr.6b) in the modified bitstream transfer datapath. On the other hand, module swapping was simulated by writing to the `engine_signature` register, which did not match the real module swapping operation on the target FPGA. Such a mismatch introduced some false positive bugs (e.g., bug.hw.2).

After the design was mature enough, we used ReSim-based simulation to test the reconfiguration machinery of the system. Apart from detecting bugs introduced by modifying the original design, we were able to identify 3 potential bugs in parts of the system that were the same as the original one. For example, the "engine reset bug" (bug.dpr.6b) was not exposed before because the original design used a faster configuratoin 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. The use of SimBs more accurately models the timing associated with partial reconfiguration.

B. Lessons Learnt

For safety-critical applications such as AutoVision, a bug could lead to data corruption (i.e., errors in pixel values) or system failure. For the example bugs listed, failing to transfer the bitstream or reset the new engine could lead to system

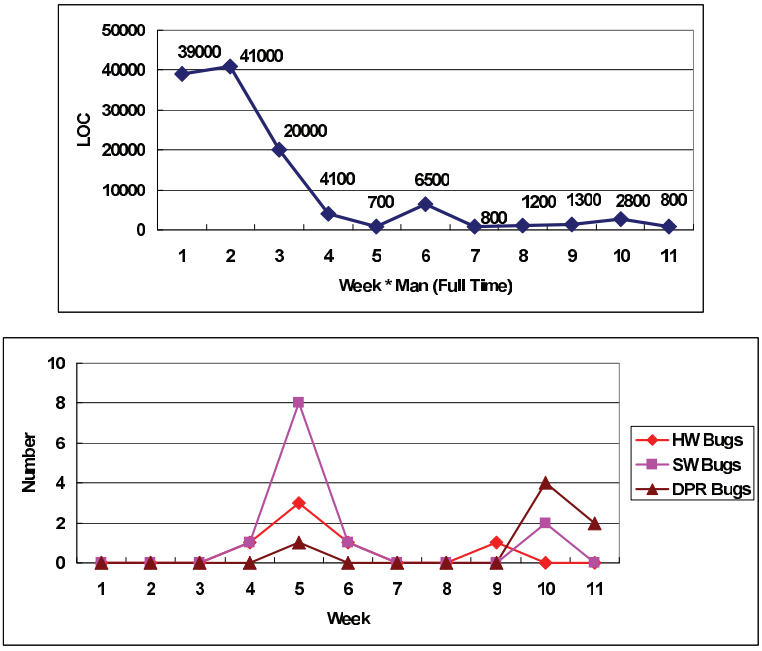

Figure 5. Development workload and bugs detected

Table III SELECTED LIST OF DETECTED BUGS

Bug Description	Bug Fixes	Comments
bug.hw.2: The CIE/ME was not reset correctly. This was because the `engine_signature` register was not correctly initialized and no engine was selected to be active.	Reset `engine_signature` at start up. Since the `engine_signature` register only exists in Virtual Multiplexing-based simulation, 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.
bug.dpr.4: The `IcapCTRL` module was used in point-to-point mode in the original system, and it failed to work with the shared PLB bus in the modified Optical Flow Demonstrator.	Changed some parameters of the `IcapCTRL` module so that it worked on shared buses. This bug was introduced by changing the way an IP was integrated.	Since ReSim models bitstream traffic, these bugs can ONLY be detected by ReSim-based simulation.
bug.dpr.5: After changing a parameter of the `IcapCTRL` module, the software driver was not updated accordingly and the SimB was not successfully transferred.	Updated the software driver of the `IcapCTRL` module to reflect the changes in the hardware. In particular, the calculation of bitstream size had to be changed. This bug was caused by a mismatch between hardware and software.	
bug.dpr.6b: The system software failed to wait until the completion of bitstream transfer before resetting the engines. This bug was introduced since the modified design used a different clocking scheme that slowed down the bitstream transfer, and the software was not updated to slow down the reset operations accordingly.	The bug could be fixed by modifying the design so that reset would only be triggered at the end of bitstream transfer. However, in order to avoid re-designing the third-party IP, we simply delayed the reset of engines by adding several dummy loops in the software.	Since ReSim more accurately models the timing of reconfiguration events, this bug can ONLY be detected by ReSim-based simulation

failure and cause serious catastrophes. It is therefore critical to thoroughly verify the design, especially the system *integration*, before using the system.

From the development progress and the bugs detected, we notice that system *integration* could be much more difficult than a designer might expect, even if the individual modules and software functions were FPGA-proven and reused. Although most design parts were reused, the designer of this study was new to the AutoVision project. As a result, much time was spent on understanding the hardware signals and software functions of legacy design parts, especially during debugging. This contributed to the relatively long time spent (i.e., 11 weeks) to complete the project. Furthermore, since the designer was not fully aware of the assumptions and constraints of legacy design parts, many bugs were introduced by integrating two correct modules with incorrect parameters

(e.g., bug.dpr.4, bug.dpr.5). These bugs can not be exposed in module-level tests and can be extremely time consuming to trace in the integrated system.

The original project tested and debugged the *integrated* Optical Flow Demonstrator using ChipScope [2]. For this case study and our host machine, the implementation and bitstream generation iteration took 52 minutes, and, as described in Section II, the debug turnaround time for on-chip debugging would therefore have been at least that much time per iteration, which is longer than the longest debug turnaround time for simulation (i.e. 44 minutes as described earlier). Furthermore, on-chip debugging typically requires several iterations to trace a bug. It could be anticipated that bugs such as the "engine reset bug" (bug.dpr.6b) would be extremely time consuming to trace using ChipScope.

For our case study, full system simulation significantly reduced the debug effort. In particular, since the Optical Flow Demonstrator is a complex software/hardware system, it is not possible, in general, to distinguish a software bug from a hardware bug when the system is not working (e.g., bug.dpr.6b). Full system simulation with both system hardware and software has the advantage of allowing the designer to move back and forth between the software running on the ISS and the RTL hardware. However, one significant overhead of full system simulation is that the designer has to build a simulation environment, which took 3 weeks in this case study. Fortunately, the investment in simulation such as Verification IPs can be reused across several projects and is still worthwhile. A full system simulation environment is therefore highly desirable to assist in debugging the integrated design.

Considering the development workload, 9 weeks out of 11 were spent on re-integrating the design and setting up the baseline simulation environment. To reduce such effort, it is recommended that the designers plan for verification, especially full system simulation, at the commencement of a DRS design project. For example, when some VIP is lacking, the designer needs to decide whether to build such IP or modify the design without compromising system performance. The HDL coding style should be simulation and synthesis friendly.

VI. Conclusions and Future Work

As with modern ASIC designs, functional verification has become a significant challenge for cutting-edge, complex DRS designs. Using AutoVision as a case study, this paper studied and analyzed the functional verification of complex DRS designs. The DUT of this study, a modified Optical Flow Demonstrator, can be viewed as an integration of proven IPs from various sources. From the development workload and the bugs detected, we identified that it is challenging and essential to verify an *integrated* system, including the behavior immediately before, during and after reconfiguration.

We used two methods, Virtual Multiplexing and ReSim-based simulation, to verify the DPR aspects of the modified Optical Flow Demonstrator. Virtual Multiplexing does not simulate an integrated design. In particular, the reconfiguration controller is not exercised and the isolation logic is not tested. Furthermore, the user software has to be "hacked" so as to model the module swapping. In contrast, ReSim-based simulation mimics the behavior of the FPGA fabric and enables cycle-accurate simulation of an integrated design immediately before, during and after reconfiguration. It is thereby able to effectively detect bugs missed by Virtual Multiplexing. For this case study, the development overhead of integrating ReSim into an existing simulation testbench was trivial compared with that of the testbench itself. The simulation overhead of ReSim was 1.7%, which was caused by selecting the engine IOs and using simulation-only artifacts such as error injectors. 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 .

References

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

[2] F. Altenried, "Time-sharing of Hardware Resources for Image Processing Accelerators using Dynamic Partial Reconfiguration," Bachelor's Thesis, Technical University of Munich, 2009.

[3] M. Keating and P. Bricaud, *Reuse Methodology Manual for System-on-a-Chip Designs*. Kluwer Academic Publishers, 2002.

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

[5] 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.

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

[7] 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.

[8] L. Gong and O. Diessel, "ReSim: A Reusable Library for RTL Simulation of Dynamic Partial Reconfiguration," in *Field-Programmable Technology (FPT), International Conference on*, 2011, pp. 1–8.

[9] P. Lysaght and J. Stockwood, "A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 3, pp. 381 – 390, 1996.

[10] 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.

[11] I. Robertson and J. Irvine, "A Design Flow for Partially Reconfigurable Hardware," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 2, pp. 257–283, 2004.

[12] A. Raabe, P. A. Hartmann, and J. K. Anlauf, "ReChannel: Describing and Simulating Reconfigurable Hardware in SystemC," *ACM Transactions on Design Automation of Electronic Systems*, vol. 13, no. 1, p. 15, 2008.

[13] L. Gong and O. Diessel, "Functionally Verifying State Saving and Restoration in Dynamically Reconfigurable Systems," in *Field Programmable Gate Arrays (FPGA), ACM/SIGDA International Symposium on*, 2012, pp. 241–244.

[14] *ChipScope Pro 12.1 Software and Cores (UG029)*, Xilinx Inc., 2010.

[15] *IBM PowerPC Multi-Core Instruction Set Simulator User's Guide*, International Business Machines Corp., 2010. [Online]. Available: http://www.ibm.com/developerworks/power/iss/

[16] *EDK Concepts, Tools and Techniques (UG683)*, Xilinx Inc., 2010.