Opportunities and Challenges for Dynamic FPGA Reconfiguration in Electronic Measurement and Instrumentation

Oliver Diessel
School of Computer Science & Engineering
University of New South Wales, Sydney NSW 2052, Australia
Email: *odiessel@cse.unsw.edu.au*

Abstract *– Reconfigurable systems based on Field-Programmable Gate Arrays (FPGAs) can offer performance and power advantages over processor-based systems as well as cost and flexibility advantages over custom integrated circuit solutions. Dynamic reconfiguration, the ability to partially modify a circuit implementation at run-time, has the potential to extend the flexibility, utilization, and resilience of FPGA-based systems even further. In this paper we review the capabilities of this technology with respect to applications in electronic measurement and instrumentation and present results on testing and assuring the correctness of dynamically reconfigurable systems at design and run time.*

Keywords *– Dynamically reconfigurable systems, FPGA, modular reconfiguration, simulation, SEU mitigation.*

I. RECONFIGURABLE SYSTEMS

Digital computing systems are mainly implemented using one of two approaches. The desired function is either programmed as software into a processor, or it is realized in hardware as a digital circuit of an electronic device. The processor-based approach is relatively quick and easy to implement and modify, and is relatively cheap to deploy in small volumes. In contrast, the circuit-based approach is more complex and time-consuming to design and implement, may be impossible to modify, and only becomes cheap in very large volumes. On the other hand, relative to custom-designed circuits, processors expend more energy and have lower performance in operation.

Within the spectrum of implementation choices, Field-Programmable Gate Arrays (FPGAs), using static RAM (SRAM) as a configuration memory to implement application circuits post production, have created a niche for themselves between processors on the one hand and custom circuits on the other. FPGAs are *universal* digital devices in that the one device can be used to implement *any* digital circuit for which there are sufficient resources available. Fundamentally, an FPGA achieves this universality by providing a large number of configurable logic elements, composed of small look-up tables (LUTs) and flip-flops (FFs), that can be arbitrarily interconnected via a configurable wiring matrix. A specific function is realized by setting the desired function into the LUTs, and interconnecting these with FFs and input/output (IO) pins via wires that are selected by setting appropriate routing switches between signal sources and their destinations. The desired functionality is implemented by loading the FPGA's configuration memory with a configuration bitstream that contains the required LUT and routing switch settings. Changing the functionality implemented by the device simply involves loading a different configuration bitstream. Since FPGAs can be reconfigured to implement different circuits as readily as processors execute different programs, they bridge the gaps in flexibility, performance, energy consumption and costs between custom devices and processors.

FPGAs have proven themselves effective and competitive at a range of processing tasks. These include fundamental operations such as filtering, matching, sorting, switching and control. A prime application domain is signal processing, including image, video, and radar. Suitability for using FPGAs as core computational devices in these applications depends upon the performance requirements (more is better), available lead time (less is better), the need for processing flexibility (greater is better), and volume of the market (smaller is better). Another class of processing problems that is well matched to the capabilities of FPGAs is stream processing, which includes network traffic and video processing, cryptography, genomic sequence alignment, and database processing. FPGAs are also commonly found in very specialized, high-end simulation (e.g. N-body, cellular automata) and big science (e.g. the ALICE experiment of the Large Hadron Collider as well as the cross-point switches and correlators for the Australian Square Kilometre Array Pathfinder) projects.

II. DYNAMIC RECONFIGURATION

Since FPGA configurations are most commonly stored in volatile SRAM, they are typically loaded when the device is powered on from off-chip storage such as a PROM or flash device. Unless there is a need to change the functionality of the FPGA, it is not reconfigured while in operation. If a change in function is desired, the off-chip configuration store is overwritten so that when the FPGA is next re-started, the new configuration is loaded instead.

Certain FPGA device families provide a special operating mode in which *part of* the configuration memory can be reloaded while the unaffected user logic continues to operate. Because it occurs at run time, this is referred to as *dynamic*

978-1-4799-0759-5 /13/$31.00 ©2013 IEEE

reconfiguration (DR), and it has the potential to significantly enhance the functionality of the FPGA.

A common use of DR is to reuse FPGA area for different functions that are readily time-multiplexed, e.g. when different functions are needed in different operating "modes". An example of such a use is a so-called muxponder function in which a network switch is capable of supporting one of multiple communications protocols e.g. 10 GigE, OC48, and fibre, and is reconfigured between connections to implement the protocol needed for the next connection [36]. This type of use can be characterized by the design-time knowledge of the variety of circuit modules that may be swapped at run time. A generalization of the approach, described below, aims to provide an open system in which the modules that are swapped in at run time may not be known at design time.

A more refined usage of DR replaces existing circuits with variations of them to adapt the function to changing environmental conditions and requirements. For example, a receiver may dynamically adapt the number of CORDIC operators to the data rate and antenna number in a MIMO detector [32]; an email spam filter must be adapted to detect new threats as they emerge [10]; similarly, an electronic jammer needs to respond to rapidly changing characteristics of radio signals [7]. This use of DR complements circuit optimizations such as constant folding [34] and partial evaluation [30]. This type of use may need to rely on rapid circuit specialization at run time, which may constrain the complexity of the circuits it is applied to.

As alluded to in the first example of DR use above, another application of DR is to support "general-purpose" hardware acceleration as seen in experimental systems such as DISC [33], the Erlangen Slot Machine [23], and Intel's QuickAssist technology [19]. For this application, the FPGA's resources are organized so that they can be shared in space and time by multiple (potentially unrelated) tasks. Regions of the FPGA are reconfigured with new tasks, and may even have their tasks pre-empted, with the current state being saved before the task is unloaded. Independently loading an FPGA-based task while others are running is not possible without DR. This type of use relies upon the design of an effective framework and run-time manager for concurrently sharing IO, memory and wires among multiple independent FPGA-based circuits. Individual tasks need to be designed with these constraints in mind.

Finally, DR has also been used to provide fault tolerance in at least two ways. First, tasks can be migrated between hardware and software [29], or between resources on an FPGA [6]. A second use is to clear soft errors that have affected the configuration memory of an FPGA [3].

A. Possible uses of DR in EMI

The characteristics of DR that are potentially useful to electronic measurement, test and instrumentation applications are the ability to reuse hardware for different test circuits over time (time-multiplexed hardware) and the ability to rapidly specialize the test circuits to particular requirements.

Virtual instruments capable of meeting real-time processing constraints can be created using FPGAs that are dynamically reconfigured as the instrument is switched from one intended use to another. The potential advantage is that a partial dynamic reconfiguration takes less time than a complete static reconfiguration, and that the instrument can continue to operate while a subset of its functions are being reconfigured. Exploiting DR thereby enhances the flexibility and capability of the instrument.

VLSI testing can involve use of a large number of high-speed custom controllers. The specific controllers needed to test a particular chip could be implemented using FPGAs. The testing of complex chips may involve the creation of more control circuits than the test system can implement at once. Reconfiguration thus allows for an extensible hardware system to be created with the potential to implement a limitless number of specific tests. Dynamically reconfiguring the test system while it is operating has the potential to save considerable down time that would be incurred if the complete device were to be reconfigured statically while inactive.

When test equipment requires circuits to be specialized to meet performance constraints, DR might be exploited to adapt test circuits to specific needs, or to trial new approaches (such as searching the test circuit space) to perform a test.

It is also noteworthy that a number of specialized, high-speed instruments, as used in medicine [9], astronomy [17] and particle physics [11] have been constructed by employing DR to enhance flexibility, reduce power and improve radiation tolerance.

III. TESTING & ASSURING THE CORRECTNESS OF DYNAMICALLY RECONFIGURABLE SYSTEMS

Dynamic reconfiguration introduces additional complexity to the validation and testing of hardware. This additional complexity is due to the dynamic nature of the hardware. Hardware is traditionally considered static, and design and test methods have evolved around this central tenet. However, when hardware can change at run time, not only do we need to determine the correctness of the system before and after the change, we need new methods for testing that the system remains correct while the change is in progress. There is thus a need for tools and methods that assist in validating dynamically reconfigurable designs, in testing implementations, and in assuring correct operation at run time.

A. Design Time Validation

Apart from validating each configuration of a dynamically reconfigurable system, it is essential to test and debug the integrated DR design, including the behavior immediately before, during and after partial reconfiguration [13]. Unfortunately, FPGA vendors such as Xilinx do not provide methods for simulating the reconfiguration process [36].

DR is commonly simulated via the insertion of a multiplexer into the design to interleave the communication between reconfigurable modules connected in parallel [20]. This method is the basis for more recent efforts in simulating DR. However, it only models module swapping and fails to simulate other aspects of DR, such as module isolation and bitstream retrieval, which is increasingly handled on-chip.

The more recent Dynamic Circuit Switch method [21], [26], [27] improves the simulation accuracy of DR designs in various respects. However, like the virtual multiplexing approach, it assumes that the reconfiguration delay is zero or a constant and does not simulate bitstream traffic. Furthermore, reconfiguration is triggered by monitoring designer-selected signals in the Register Transfer Level (RTL) code, whereas on real FPGAs, module swapping is triggered by bitstream transfer.

ReChannel [25] is a SystemC-based, open source library to model DR at the transaction level. However, such extension only focuses on the high-level modeling of DR 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.

Our recent work, ReSim [14][15], 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 DR design.

B. On-chip Testing

Field testing runs the implemented design on the target device under real-world conditions. However, tracing the cause of a bug on the implemented design requires extra effort to insert probing logic using vendor tools (e.g., ChipScope [35] and SignalTap [1]) and the design needs to be re-implemented every time a different set of user design signals is to be probed. The debug turnaround time of on-chip debugging is therefore at least as long as the time-consuming implementation stage. Furthermore, since probing logic can only visualize a limited number of signals for a limited period of time, on-chip debugging typically involves more iterations to identify the source of a bug than simulation requires. Last but not least, collecting and analyzing coverage data on chip is not guided by quantifiable metrics. As a result, it is essential to perform thorough simulation-based functional verification to detect as many logic errors as possible before testing the implemented design on the target FPGA [37].

C. Run-time Checking

Checking designs are behaving as expected and that the conditions under which a design operates are as expected can be achieved using run-time assertions [31].

When physical faults, permanent or hard errors are detected, reconfigurable systems may make use of spare or redundant hardware [6].

When transient errors are to be tolerated, the most common approach is to use spatial redundancy, such as Triple Modular Redundancy (TMR), to detect and hide the errors [22]. In order to overcome transient errors leading to changes in datapath register values, the XTMR tool ensures all register inputs are voted upon [38].

The extensive configuration memory of FPGAs is a significant potential source of error, particularly when it is subjected to ionizing radiation, such as in space-based, airborne and very large-scale or radiation-exposed terrestrial applications. This is due to the fact that configuration memory bit flips can alter the function of a logic cell or the interconnection of components and thus present the same symptoms as hardware errors. These errors can be dealt with by reloading the configuration that has been corrupted. Traditionally, the complete configuration is "scrubbed", i.e. each configuration frame is read, checked and rewritten if found to contain an error [3]. More recently attention has turned to more fine-grained partial modular reconfiguration to reload a TMR module that is in error. This approach can be more responsive and use less energy, and is therefore attractive in real-time, space-based applications [2][24][4].

Dynamic reconfiguration has the potential to implement open-ended hardware-based systems, in which new hardware is loaded as needed and integrated with an operating system. One concern with such a vision is to ensure the acquired IP is safe for use and fit for purpose. Some work has been undertaken to study the potential for proof-carrying hardware to facilitate run-time validation [8]. However, real systems that demonstrate the potential for enhancing functionality in an on-line manner are yet to emerge.

IV. SIMULATING DYNAMIC RECONFIGURATION

Compared with traditional static FPGA designs, DR introduces additional flexibility for system designers but also introduces challenges to the verification of design functionality. Before reconfiguration, the static logic should properly synchronize with the circuit that is to be replaced to pause the ongoing computation. During reconfiguration, a reconfiguration controller transfers the configuration bitstream of the incoming circuitry to the configuration port. During this period, the static part must isolate the reconfigurable region (RR) to avoid the propagation of spurious outputs from partially configured circuits. After reconfiguration, the incoming circuit needs to be initialized to a known state before it starts execution [13].

Since traditional RTL simulation does not model characteristic features of DR, it only offers limited assistance in detecting DR-related bugs [14][15]. However, it is challenging to accurately model these characteristic features. In traditional FPGA-based hardware designs, the FPGA fabric is statically configured and does not interact with the user design. In DR designs, on the other hand, components of the FPGA fabric interact with the user design in the process of partial reconfiguration. In particular, bitstreams are transferred by the user design and the bitstreams subsequently

overwrite the configuration memory, thereby changing the functionality of the user design. Therefore, a completely accurate simulation of the reconfiguration process involves modeling these aspects of the FPGA fabric. However, because FPGA vendor tools do not provide a simulation model for the FPGA fabric, it is non-trivial for designers to accurately simulate the interactions between the user design and the FPGA. And in any case, fabric-accurate simulation includes a multitude of unnecessary details for verification. For the sake of productivity, it is desirable that functional verification remains physically independent. An effective simulation method therefore needs to strike a balance between simulation accuracy and verification productivity. Furthermore, the simulated design should be implementation ready. That is, the captured design should not be changed for simulation purposes.

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. ReSim uses a simulation-only bitstream (SimB) to model the bitstream traffic. A SimB captures the essence of a real bitstream in that it serves the purpose of and effects the mediation of module swapping. By summarizing the details of a real configuration bitstream, the size of a SimB is significantly reduced and verification productivity is thereby improved. Nevertheless, ReSim allows the bitstream transfer and its manipulation (decryption, decompression, etc.) to be tested. ReSim models the RR in order to exercise module swapping and its trigger conditions, as well as spurious outputs from the region while it is being reconfigured, and that the system correctly handles any initially undefined module state.

Since the simulation-only layer abstracts away the details of the FPGA fabric, it can be regarded as a fabric-independent FPGA device, and simulation can be thought of as functionally verifying the user design layer of a DR system on such a fabric-independent FPGA. Simulation using the simulation-only layer assists designers in detecting fabric-independent bugs of a DR design. These bugs include, but are not limited to: system integration bugs and software bugs; bugs in synchronizing the static region and the logic that is to be reconfigured before reconfiguration; bugs in the bitstream transfer logic; bugs in isolating the region undergoing reconfiguration; and bugs in initializing the newly configured circuitry. However, the simulation-only layer is not exactly the same as the FPGA fabric. The mismatches between the two can lead to bugs that remain undetected using the simulation-only layer and bugs that are incorrectly reported. These bugs, which could be categorized as being fabric-dependent, include errors in the bitstream itself, and errors in interpreting the content of the bitstream.

The ReSim library is built upon SystemVerilog [18] and the Open Verification Methodology (OVM) [12]. As a result, ReSim is fully compatible with existing and mainstream EDA tools. ReSim takes the functional specification and a set of reconfiguration strategies as inputs. The reconfiguration strategies include names, sizes and connectivity of RRs and reconfigurable circuitry. From these specifications, the designer creates RTL code for the user logic and describes the reconfiguration strategies using a Tcl script. ReSim automatically generates the simulation-only artifacts (including the SimBs and simulation models of the configuration port and RRs) based on the Tcl script. If required, the designer can edit the generated artifacts for design and/or test-specific needs.

Through 6 case studies (such as [16]), we have demonstrated that ReSim can be applied to a range of DR design styles including in-house designs, third-party designs, hardware-only designs, microprocessor-based HW/SW designs, a design that saves and restores module state via the configuration port, designs that use customized reconfiguration controllers and vendor IP, as well as designs that were mapped to different FPGA families. The extra development workload for using ReSim involved creating parameter scripts, which ranged from 50–150 lines of Tcl code for each of these case studies. Generally speaking, the workload of using ReSim is trivial compared to the effort spent creating a DR design and a testbench. For each case study, we used the ModelSim profiling tool to evaluate the simulation overhead of ReSim. We found that 0.3–20.9% of simulation time was spent in ReSim. The simulation overhead of ReSim is proportional to the number of signals crossing the RR boundary since all boundary signals are multiplexed as opposed to being connected to the static part directly. The overhead is also proportional to the frequency of reconfiguration in a specific simulation run, since each reconfiguration involves costs to swap logic and inject errors, and includes a scenario-dependent delay to transfer the SimB. In the course of these case studies, we found 69 DR-related bugs, 28 of which could only be detected using ReSim, and one of which could only be detected using on-chip debugging.

V. MITIGATING CONFIGURATION MEMORY UPSETS IN FPGAS

Off-the-shelf FPGAs are increasingly being considered as potential platforms for delivering the performance, flexibility, and power budgets required for space-based systems. However, space-based systems are exposed to ionizing radiation, which can cause undesirable signal transitions in the implemented circuits as well as in the configuration memory. In the configuration memory, such Single Event Upsets (SEUs) can have the effect of altering the contents of LUTs and the contents of routing switches. As a consequence, the implemented application circuits are affected by logic errors and stuck at faults.

A common approach used to hide the effects of SEUs is to implement application circuits using Triple Modular Redundancy (TMR) [22]. In such a scheme, three identical copies of a circuit operate in parallel, and the output is determined by majority vote. This scheme successfully hides transient and permanent errors that affect at most one of the triplicated modules since two erroneous modules can lead to

an erroneous output by majority decision. It is therefore desirable to eliminate "permanent" errors, in the form of configuration memory content errors and errors contained in datapath registers with cyclic dependencies, before an SEU affects a second module in a TMR system.

Configuration memory contents are corrected by rewriting the memory, i.e. reconfiguring it. Stopping the device to completely reconfigure it has the drawback of interrupting processing for a considerable period – perhaps on the order of a second – which is potentially unacceptable for real-time applications. Various dynamic reconfiguration approaches have therefore been suggested.

One approach, referred to as "scrubbing", periodically refreshes the entire configuration memory contents [3]. While the application can continue to operate while the scrub is in progress, the scrubbing process itself is slow and costly in terms of energy use because several MB of configuration data need to be transferred. Moreover, a separate mechanism is required to correct the corrupted state of datapath registers with cyclic dependencies. These can include checkpointing [28] and copying the state from a fault-free module.

Alternatively, the erroneous module can be reconfigured using dynamic partial reconfiguration, while its sibling modules in the TMR system continue to operate [2][24][4]. This approach is more responsive, takes less time to complete, and uses less energy as only that part of the device that contains the error is reconfigured. To overcome the problem of restoring the correct state to datapath registers with cyclic dependencies, a further set of approaches have been proposed. On the one hand, the state at which the reconfigured module is to be restarted can be predicted [24], or a voter is inserted into each feedback path [4]. In the latter case, the correct feedback is provided to the newly reconfigured module by its siblings.

In our work, we are investigating the dynamic partial reconfiguration approach as a means of designing circuits that are guaranteed to recover from a fault within a specified period. We would also like to show that we achieve similar, if not better, reliability than scrubbing while expending less energy. Our investigation is examining the trade-off between module size and recovery time, which is primarily determined by the reconfiguration delay. Nevertheless, a number of other factors need to be considered, including the layout of the TMR system, and the communication of reconfiguration requests to a centralized reconfiguration controller.

Recovery commences with the detection of a fault. When feedback signal paths are voted upon, the detection of consecutive errors in a module's output indicates the likely presence of a configuration memory content error for that module. The delay in detecting the error is bounded by the latency of the module. Interestingly, with all feedback being voted upon, this is also the time it takes to resynchronize the state of the reconfigured module with its siblings [4].

As a large and complex circuit or system will be partitioned into many TMR components, and requests for reconfiguration (initiated by voters) need to be communicated to a single reconfiguration controller, we have proposed the use of token ring network to convey the reconfiguration requests and completion signals. The delay of this network is proportional to the number of TMR components and the clock frequency of the slowest component [5].

The time to retrieve and write the partial configuration bitstream for the module undergoing reconfiguration is proportional to the size of the module's bounding box, normalized to the unit of reconfiguration (a 41-word configuration frame).

Combining the delays for detecting a fault, communicating the reconfiguration request, reconfiguring the faulty module, and resynchronizing the reconfigured module, allows us to choose module sizes that result in a maximum recovery time below the specified bound.

We have simulated and implemented on a Xilinx Virtex-5 XC5VFX70T FPGA a small system comprising a triplicated 16-bit, 21-tap single accumulator Finite Impulse Response (FIR) filter, a triplicated 8-to-3-bit, 256-sample Block Adaptive Quantizer (BAQ) circuit, a 3-wire token ring control network, and a flash-based reconfiguration controller. Our simulations, which assumed the maximum theoretical reconfiguration port bandwidth was available, indicated that the system could recover from an error in one of the three FIR modules in less than 50μs, and an error in a BAQ module in less than 80μs. In contrast, our implementation, which operated at half the simulated clock speed and had a reconfiguration bandwidth that was throttled by a slow flash controller, recovered from errors in these modules in under 2.0ms and 3.4ms, respectively. These recovery times correspond to FPGA regions covering 0.5% (100 reconfiguration frames) and 0.9% (180 reconfiguration frames), respectively, of the complete FPGA area. In comparison, a compete reconfiguration, or scrubbing, would have taken up to 100X longer than the time to reconfigure the BAQ module. Our results can also be contrasted with the peak average SEU rate of approximately 1 per second that has been estimated for an XC4VLX200 device located in a geosynchronous orbit. It should be noted that this device is twice as large as our test device, but is produced with a 90nm process rather than a 65nm process, which is more susceptible to radiation events. Without partitioning the circuits we implemented into smaller TMR modules, we were able to achieve maximum recovery times approximately 300X less than the average expected fault rate.

VI. CONCLUSIONS

Dynamic reconfiguration is a technique that enhances the flexibility, utilization, and reliability of high-performance FPGA-based systems. Potential applications in EMI include the construction of virtual instruments, extensible hardware-based test systems, and adaptive systems.

Dynamic reconfiguration introduces challenges for the validation, testing, and run-time checking of system behaviour. Our work has progressed the state-of-the-art in the functional simulation of DR systems by increasing the

accuracy with which such systems are modelled without significant loss of productivity. We are also investigating the use of DR to improve the reliability of FPGA-based systems that are exposed to ionizing radiation.

Future work includes devising improved techniques for assessing the correct and safe operation of dynamically reconfigurable systems at run time.

ACKNOWLEDGMENTS

The author wishes to acknowledge the primary contribution of Lingkan Gong towards the simulation work and the collaboration of Ediz Cetin, Lingkan Gong and Victor Lai on the SEU mitigation work reported in this paper. Donations received through the Xilinx University Program are also gratefully acknowledged.

REFERENCES

[1] Altera Corp., "Design Debugging Using the SignalTap II Embedded Logic Analyzer". 2012.
[2] C. Bolchini, A. Miele, and M. D. Santambrogio, "TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs". Defect and Fault-Tolerance in VLSI Systems, pp. 87–95, 2007.
[3] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration". Xilinx Corporation, 2000.
[4] E. Cetin, and O. Diessel, "Guaranteed Fault Recovery Time for FPGA-based TMR Circuits Employing Partial Reconfiguration". DAC Workshop on Computing in Heterogeneous, Autonomous 'N' Goal-oriented Environments, 2012.
[5] E. Cetin, O. Diessel, L. Gong and V. Lai, "Towards Bounded Error Recovery Time in FPGA-based TMR Circuits using Dynamic Partial Reconfiguration". Field-Programmable Logic and Applications, 2013.
[6] J. A. Cheatham, J. M. Emmert, and S. Baumgart. "A survey of fault tolerant methodologies for FPGAs". ACM Trans. Des. Autom. Electron. Syst., vol. 11, no. 2, pp. 501–533, 2006.
[7] T. W. Coleman, D. B. Gabriel, and B. Kogan, "Systems and methods for radio frequency hopping communications jamming utilizing software defined radio platforms". U.S. Patent Application No. 13/532,235, Publication No. 20130023201, Jan 24, 2013.
[8] S. Drzevitzky, U. Kastens, and M. Platzner, "Proof-carrying Hardware: Towards Runtime Verification of Reconfigurable Modules". Reconfigurable Computing and FPGAs, pp. 189–194, 2009.
[9] H. Eggers, P. Lysaght, H. Dick, and G. McGregor, "Fast Reconfigurable Crossbar Switching in FPGAs". Field-Programmable Logic, pp. 297–306, 1996.
[10] K. Eguro, "Automated dynamic reconfiguration for high-performance regular expression searching". Field-Programmable Technology, pp. 455–459, 2009.
[11] W. Gao, A. Kugel, R. Männer, N. Abel, N. Meier, and U. Kebschull, "DPR in high energy physics". Design, Automation and Test in Europe, pp. 39–44, 2009.
[12] M. Glasser, "Open Verification Methodology Cookbook". Mentor Graphics Corporation, 2009.
[13] L. Gong and O. Diessel, "Modeling Dynamically Reconfigurable Systems for Simulation-based Functional Verification". Field-Programmable Custom Computing Machines, pp. 9–16, 2011.
[14] L. Gong and O. Diessel. "ReSim: A Reusable Library for RTL Simulation of Dynamic Partial Reconfiguration". Field-Programmable Technology, pp. 1–8, 2011.
[15] L. Gong and O. Diessel. "Functionally Verifying State Saving and Restoration in Dynamically Reconfigurable Systems". Field Programmable Gate Arrays, pp. 241–244, 2012.
[16] L. Gong, O. Diessel, J. Paul and W. Stechele, "RTL Simulation of High Performance Dynamic Reconfiguration: A Video Processing Case Study". Reconfigurable Architectures Workshop, pp. 106–113, 2013.
[17] P. J. Hall, "Power considerations for the Square Kilometre Array (SKA) radio telescope". General Assembly and Scientific Symposium, XXXth URSI , pp. 1–4, 2011.
[18] IEEE, "IEEE Standard 1800-2005: SystemVerilog – Unified Hardware Design, Specification, and Verification Language". The Institute of Electrical and Electronics Engineers, Inc., 2005.
[19] Intel Corp., "Intel QuickAssist Acceleration Technology for Embedded Systems". http://www.intel.com/content/www/us/en/io/quickassist-technology/quickassist-technology-developer.html accessed 25/6/2013.
[20] W. Luk, N. Shirazi, and P. Y. Cheung, "Compilation tools for run-time reconfigurable designs". Field-Programmable Custom Computing Machines, pp. 56–65, 1997.
[21] P. Lysaght and J. Stockwood, "A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays". IEEE Trans. on VLSI Syst., vol. 4, no. 3, pp. 381–390, 1996.
[22] R. E. Lyons, and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability". IBM Journal of Research and Development, vol. 6, no. 2, pp. 200–209, 1962.
[23] M. Majer, J. Teich, A. Ahmadinia, and C. Bobda, "The Erlangen Slot Machine: A Dynamically Reconfigurable FPGA-based Computer". J. VLSI Signal Process. Syst., vol. 47, no. 1, pp. 15–31, 2007.
[24] C. Pilotto, J. R. Azambuja, and F. L. Kastensmidt, "Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications". Integrated Circuits and System Design, pp.199–204, 2008.
[25] A. Raabe, P. A. Hartmann, and J. K. Anlauf, "ReChannel: Describing and Simulating Reconfigurable Hardware in SystemC". ACM Trans. Des. Autom. Electron. Syst., vol. 13, no. 1, p. 15, 2008.
[26] I. Robertson, J. Irvine, P. Lysaght, and D. Robinson, "Improved Functional Simulation of Dynamically Reconfigurable Logic". Field Programmable Logic and Applications, pp. 541–574, 2002.
[27] I. Robertson and J. Irvine, "A Design Flow for Partially Reconfigurable Hardware". ACM Trans. Embed. Comput. Syst., vol. 3, no. 2, pp. 257–283, 2004.
[28] A. Sari and M. Psarakis, "Scrubbing-based SEU mitigation approach for systems-on-programmable-chips". Field-Programmable Technology, pp. 1–8, 2011.
[29] H. Simmler, L. Levinson, and R. Männer, "Multitasking on FPGA Coprocessors". Field-Programmable Logic and Applications, pp. 121–130, 2000.
[30] S. Singh, J. Hogg, and D. Mcauley, "Expressing Dynamic Reconfiguration by Partial Evaluation". FPGAs for Custom Computing Machines, pp. 188–194, 1996.
[31] T. Todman and W. Luk, "Runtime assertions and exceptions for streaming systems". Field-Programmable Logic and Applications, 2013.
[32] H. Wang, P. Leray, and J. Palicot, "Reconfigurable architecture for MIMO systems based on CORDIC operators". Comptes Rendus Physique, vol. 7, no. 7, pp. 735–750, 2006.
[33] M. J. Wirthlin, B. L. Hutchings, "A dynamic instruction set computer". FPGAs for Custom Computing Machines, pp. 99–107, 1995.
[34] M. J. Wirthlin and B. L. Hutchings, "Improving functional density through run-time constant propagation". Field-Programmable Gate Arrays, pp. 86–92, 1997.
[35] Xilinx Inc., "ChipScope Pro 12.1 Software and Cores (UG029)", 2010.
[36] Xilinx Inc. "Partial Reconfiguration User Guide (UG702)". 2010.
[37] Xilinx Inc., "Synthesis and Simulation Design Guide". 2010.
[38] Xilinx Inc. "Xilinx TMRTool product brief". (2009).