Advances and Trends in Dynamic Partial Run-time Reconfiguration[⋆]

Dirk Koch, Jim Tørresen
Department of Informatics
University of Oslo, Norway

1 Benefits of Runtime Reconfiguration

Partial runtime reconfiguration allows to fit circuits on an FPGA that would exceed the device capacity in a static only implementation. This implementation technique is only applicable if the system contains modules with mutual exclusive functionality or if the utilization of some modules allow time-multiplexing of the same FPGA resources. For example, as illustrated in Figure 1, in an FPGA-based network package inspection hardware, the same device resources (a reserved reconfigurable area on the FPGA) can be used for hosting different accelerator modules. Then, during runtime, this system can adapt to the current protocol load of the network traffic. As the total load is limited by the network itself, the variation among different protocols will lead to different demands among the corresponding accelerators. As one example, the different demands could been served by instantiating more or less accelerator instances.

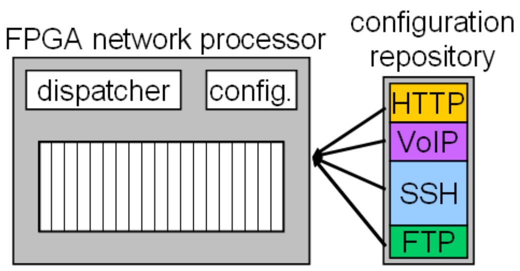

Fig. 1. Example of a reconfigurable network processor. Depending on the present protocol load, the system can instantiate different accelerator modules with the help of partial runtime reconfiguration.

⋆ This work is supported in part by the Norwegian Research Council under grant 191156V30. It summarizes results of the Dagstuhl Seminar 10281 *Dynamically Reconfigurable Architectures.*

Partial runtime reconfiguration can also be used to accelerate a system. By providing more area for a particular task and sharing the same area with the help of runtime reconfiguration, the execution time for each task can be reduced, as shown in Figure 2. Assuming that a task can start after the completion of its successor, the total execution time (latency) can be reduced time for all tasks. For example, in a system providing hardware acceleration for a secured SSL network data transfer, runtime reconfiguration can be used to switch between an accelerator module the asymmetric key exchange and a symmetric cipher module which is used for the entire data encryption.

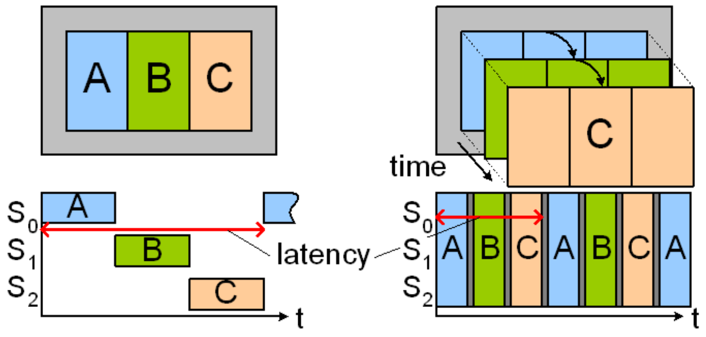

Fig. 2. Accelerating a system by partial runtime reconfiguration. By exploiting more parallelism (i.e. utilizing more resources), each module completes its execution faster. This can result in a smaller combined latency, even if we considering some configuration overhead.

It can be summarized that partial runtime reconfiguration can help to use smaller devices which helps to save monetary cost, power consumption, and space in a system. Furthermore, it can in some systems be used to reduce latency or throughput. However, despite this promising benefits, partial runtime reconfiguration is still exotic and is not popular in industrial systems. It has been shown its capabilities in, for example, software defined radio applications [1, 2]. Runtime reconfiguration is not widely applied, because not all systems are suitable for profitably applying this technique (e.g. if all modules are active at any time) and because a weak tool support for implementing corresponding systems. Furthermore, not all FPGA-vendors support partial runtime reconfiguration in their devices. As revealed in the following section, the latter issues will be solved by fulfilling customer demands regarding system reliability and safety as well as for fast system boot.

2 Support for Dynamic Partial Runtime Reconfiguration is Becoming Mainstream

The progress in silicon industry has resulted in a tremendous increase in device capacity of FPGAs. As illustrated in Figure 3, the smallest devices of the upcoming Altera Stratix 5 FPGAs as well as the announced Xilinx Virtex-7 FPGAs provide more than double the amount of logic and embedded memory as the flagship devices of the one decade old Stratix or Virtex-II series FPGAs. By passing the one million LUTs border, high density FPGAs are sufficient to host 250 softcore CPUs plus the required peripherals.

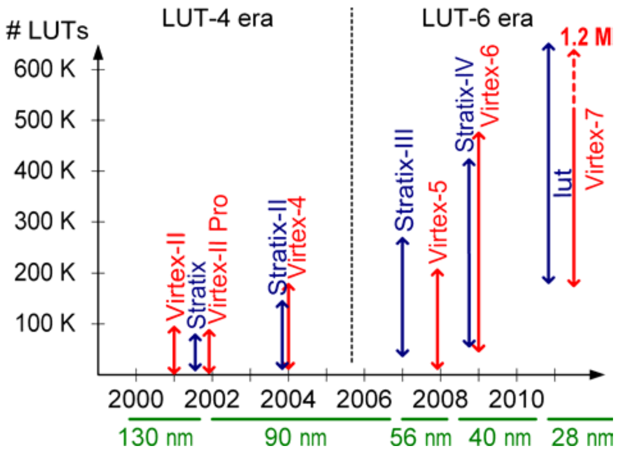

Fig. 3. One decade of FPGA evolution. The figure denotes the increase in logic density over time and over the corresponding process technology.

As the functionality of the here regarded FPGAs is defined by SRAM cells, the corresponding configuration data has also rose towards tens of megabytes (see Figure 4). Note that the highest capacity FPGAs typically provide more SRAM cells of what can be found in the largest SRAM memories released at the same time.

This progress in device density results in two drivers for introducing dynamic partial runtime reconfiguration in all future high-density devices: 1) increasing vulnerability to SEUs and 2) an increase in the configuration time.

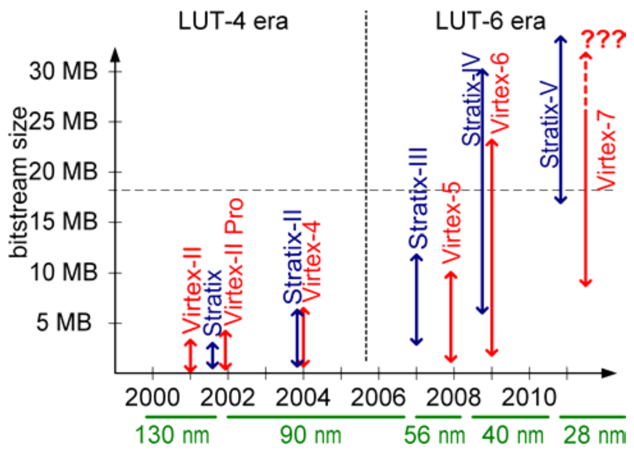

Fig. 4. One decade of FPGA evolution. The figure lists the increase in configuration memory size (total bitstream). By comparison, the largest currently available SRAM device provides 18 MB (Cypress 2009).

2.1 Increasing Vulnerability to SEUs

By dramatically increasing the total amount of configuration bits as well as by shrinking the configuration memory cells at the same time, FPGAs have become more vulnerable to single event upsets (SEU). As compared to ASICs, a single event upset may not only result in an error in the datapath but much more likely in a malfunction of the circuit. This results from possible SEUs in the configuration SRAM cells that describe the present circuit that has been loaded to the FPGA [3]. Not every SEU results necessary in a malfunction as, for example, not all resources of an FPGA (e.g., logic or routing resources) are used in a particular circuit. However, if SEUs are not corrected, multiple event upsets might occur and therefore dramatically increase the risk of a circuit failure [4].

For dealing with SEUs, different application and safety scenarios have to be considered [5]. If faults can be temporarily accepted (can be seen as noise), it is sufficient to permanently overwrite the existing configuration (or parts of it) while keeping the device in active operation mode. This process is called *configuration scrubbing*. Note that configuration scrubbing can be combined with other fault tolerant techniques, such as triple modular redundancy (TMR) [6, 7].

In other applications, the present configuration has to be validated (e.g., by reading back the configuration data) before committing the result, like for instance, in banking applications.

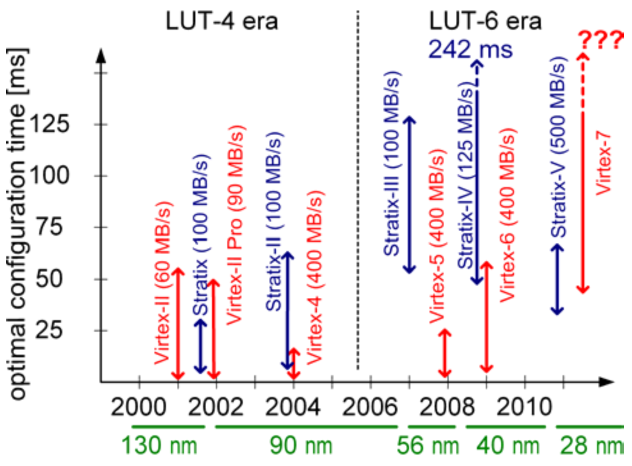

Fig. 5. One decade of FPGA evolution. The figure lists the configuration time of a full bitstream when considering the fastest possible configuration speed.

We can summarize that *partial runtime reconfiguration is required for implementing safety critical systems*.

2.2 Configuration Bootstrapping

As the configuration bitstream size has enormously increased (see Figure 4), FPGAs vendors have started to introduce faster configuration interfaces in order to avoid an exponential rise in configuration time (see Figure 5). This was mainly achieved by widening the configuration ports (e.g. from 8-bit in Virtex-II to 32-bit in Virtex-4 FPGAs).

However, the initial configuration is typically still provided by a relatively slow flash memory device [8]. This is critical for many applications that demand fast availability after power-up, like for example, a PCIe interfaces implemented on a FPGA. This issue can be solved with the help of *bootstrapping* where in an initial configuration step only the modules requiring fast availability will be loaded to the device while finishing the configuration in a further step. An example of this procedure is illustrated in Figure 6 for bootstrapping a system with an PCIe interface core and is inspired by application notes from Altera and Xilinx [9, 10]. Bootstrapping is based on partial reconfiguration for incrementally loading sections of the configuration bitstream at system start. Bootstrapping has also been demonstrated in a few academic systems (e.g., [11, 12]).

Reconfiguration is a general technique that can be used to speed-up the start of a system in manifold ways. For instance, in [13], readback

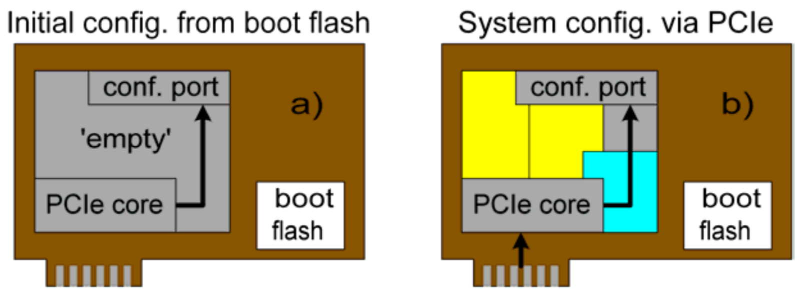

Fig. 6. Configuration bootstrapping example for PCIe solutions. a) After power-up, mainly the PCIe core is configured while leaving the rest of the device empty in order to fulfill fast PCIe core activation. b) Finally, the remaining system is loaded in a second non time-critical phase to the device.

and partial reconfiguration have been used to rapidly initialize a system with its initial state which has been captured once after normal system start. In [8], it is suggested to replace a HW-decompression module with user logic after it has decompressed the firmware of a system faster than a software only solution would perform this task.

We can summarize that *partial runtime reconfiguration allows to implement faster start-up times*. Together with the increasing vulnerability to SEUs, this forces FPGA vendors to include partial runtime capability in all their future devices. This can be identified be the vendor Altera Inc. that recently announced to provide full support for dynamic partial run-time reconfiguration in their Stratix-V series [14].

3 Context-Switching on FPGAs

With the widely introduction of partial run-time reconfiguration in all future high-density FPGAs, the technical basis for implementing context switching on FPGAs will be provided. In context switching FPGA-based systems, parts of the FPGA fabric will be shared by multiple modules over time. This is similar to multi-context task execution in software systems. However, in the hardware case, we have to distinguish between 1) the context of the FPGA (*device level*) and 2) the context represented inside the memories of the modules (*module level*). As depicted in Figure 7, the device level is given by the present module layout that can vary when (partially) reconfiguring the FPGA. Respectively, the module level is represented by the flip-flop values or the content of RAM blocks within the entire modules located on the device.

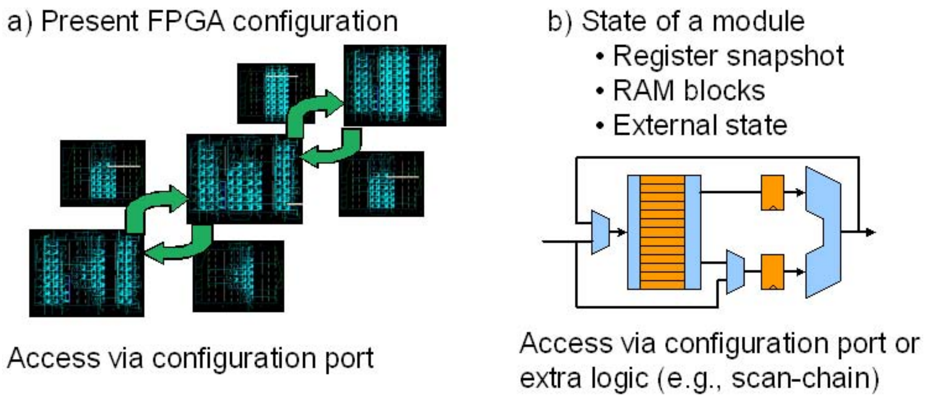
a) Present FPGA configuration
Access via configuration port
b) State of a module
• Register snapshot
• RAM blocks
• External state
Access via configuration port or extra logic (e.g., scan-chain)

Fig. 7. Context-switching on FPGAs. The context of an FPGA is twofold: on a) the *device level*, it is represented by the present FPGA configuration and on b) the *module level*, by the internal state of the modules.

3.1 COSRECOS: Making Context-Switching Available

Despite the technical basis and more than two decades of intensive research on exploiting partial run-time reconfiguration, there still exist a wide gap between the possibilities and what is currently available to implement reconfigurable systems. This can be seen by the little acceptance in industry for this topic.

With the new COSRECOS project (Context Switching Reconfigurable Hardware for Communication Systems) [15] we try to bridge this gap. By developing novel methodologies and advanced tools, we want to make the implementation of reconfigurable systems and their operation as simple as it is known from the software world. Consequently, we are investigating 1) *design-time aspects*, including models, analysis, debugging, and tools as well as 2) *run-time aspects* where we focus on high-speed reconfiguration, temporal module placement and interprocess communication. Moreover, we will 3) demonstrate our approaches on applications from the networking and general purpose domain.

In particular, we will enhance the capabilities of the tool ReCoBus-Builder (`www.recobus.de`) which already allows to implement reconfigurable systems with enhanced capabillities, including module placement in a very fine-grained two-dimensional resource grid, multi module instantiation, and module relocation. Note that these features are not available in the tools for implementing reconfigurable systems that are provided by the FPGA market leader Xilinx. Future versions of the tool will support latest FPGA devices and provide improvements in the implementation of an on-FPGA communication architecture that integrates reconfigurable modules. This will include tool handling (e.g., a comfortable GUI as well as a scripting interface), support for RTL simulation of the reconfiguration process, and efficiency in terms of latency and implementation cost.

For speeding up the reconfiguration process in the runtime system, we will evaluate the technical limits on existing FPGA platforms (e.g., by overclocking a configuration port) and by using advanced bitstream decompression techniques. For determining module placement and module scheduling, we are currently evaluating constraint programming which will be enhanced for heuristics required by the runtime system in the case, the temporal module placement cannot be predetermined.

As an application example, we are currently implementing accelerator modules for databases (e.g., sorting modules) which will be instantiated and combined dynamically with the help of partial runtime reconfiguration with respect to the present database operation (i.e., an SQL query).

The addressed topics are not only important for the implementation of reconfigurable systems, but also for a fully component based design flow where fully physically implemented modules can be directly integrated into a system. Our communication architecture will then be in charge to carry out the final top level routing among the pre-implemented components and our Framework will provide a placer for automatically computing module placement positions (by modeling the placement problem as a constraint program).

With this project we want to contribute to the research community with a strong emphasis on active collaborations as well as to enhance acceptance in industry for using dynamic partial run-time reconfiguration.

References

1. Kao, C.: Benefits of partial reconfiguration (2005) Xilinx Xcell Journal, vol. 2005, no. 55.
2. Kumar, R., Joshi, R., Raju, K.: A fpga partial reconfiguration design approach for rasip sdr. In: Annual IEEE India Conference (INDICON). (2009)
3. Caffrey, M., Graham, P., Johnson, E., Wirthlin, M.: Single-event upsets in SRAM FPGAs. In: Proceedings of the 5th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD). (2002)
4. Gebelein, J., Engel, H., Kebschull, U.: An approach to system-wide fault tolerance for FPGAs. In: Proceedings of the International Conference on Field Programmable Logic and Applications (FPL 2009). (2009) 467–471 ISBN: 978-1-4244-3892-1, DOI: 10.1109/FPL.2009.5272477.
5. Alderighi, M., Casini, F., D'Angelo, S., Pastore, S., Sechi, G.R., Weigand, R.: Evaluation of Single Event Upset Mitigation Schemes for SRAM based FPGAs using the FLIPPER Fault Injection Platform. In: DFT '07: Proceedings of the 22nd

IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, Washington, DC, USA, IEEE Computer Society (2007) 105–113

6. Heiner, J., Sellers, B., Wirthlin, M.J., Kalb, J.: FPGA Partial Reconfiguration via Configuration Scrubbing. In: Proceedings of the International Conference on Field Programmable Logic and Applications (FPL 2009). (2009) 99–104
7. Lima, F., Carro, L., Reis, R.: Designing fault tolerant systems into SRAM-based FPGAs. In: Proceedings of the 40th annual Design Automation Conference (DAC 03), New York, NY, USA, ACM (2003) 650–655
8. Koch, D., Beckhoff, C., Teich, J.: Hardware Decompression Techniques for FPGA-Based Embedded Systems. ACM Transactions on Reconfigurable Technology and Systems (TRETS) **2** (2009) 1–23
9. Altera Inc.: Introducing Innovations at 28 nm to Move Beyond Moore's Law (2010) available online: `www.altera.com/literature/wp/wp-01125-stxv-28nm-innovation.pdf` .
10. Xilinx Inc.: Partial Reconfiguration User Guide (UG702, v 12.2) (2010) `http://china.xilinx.com/support/documentation/sw_manuals/xilinx12_2/-ug702.pdf`.
11. Fong, R.J., Harper, S.J., Athanas, P.M.: A Versatile Framework for FPGA Field Updates: An Application of Partial Self-Reconfiguation. In: Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping (RSP'03), Washington, DC, USA, IEEE Computer Society (2003) 117–123
12. Hübner, M., Meyer, J., Sander, O., Braun, L., Becker, J., Noguera, J., Stewart, R.: Fast Sequential FPGA Startup Based on Partial and Dynamic Reconfiguration. In: Proceedings of the 2010 IEEE Annual Symposium on VLSI. ISVLSI '10, Washington, DC, USA, IEEE Computer Society (2010) 190–194
13. Schiefer, A., Kebschull, U.: Optimization of Start-up Time and Quiescent Power Consumption of FPGAs. In: International Conference on Field Programmable Logic and Applications (FPL 2005), IEEE Computer Society (2005) 551–554
14. Altera Inc.: Altera Unveils Innovations for 28-nm FPGAs (2010) available online: `http://www.altera.com/corporate/news_room/releases/2010/products-/nr-innovating-at-28-nm.html` .
15. University of Oslo: The COSRECOS project website (2010) `http://www.matnat.uio.no/forskning/prosjekter/crc/`.