

HIGH PERFORMANCE COMPUTING BY CONTEXT SWITCHING RECONFIGURABLE LOGIC

Jim Torresen
Knut Arne Vinger
Department of Informatics, University of Oslo
P.O. Box 1080 Blindern, N-0316 Oslo, Norway
E-mail: {jimtoer, karnevin}@ifi.uio.no

KEYWORDS

Computer systems, Electronics, Parallel methods, Performance analysis, Special-purpose processors.

ABSTRACT

Reconfigurable computing has grown to become an important and large field of research. It is based on using Field Programmable Gate Arrays (FPGAs). In this paper, this technology is introduced and it is shown how it can be applied for high speed computing. There is a large range of real-world applications for such systems, including image recognition and signal processing.

INTRODUCTION

Much of the focus on computer hardware is on the speed and performance of the latest processors. Moores law has so far been successful in predicting the progress in the technology. In parallel to this development, reconfigurable logic devices are getting faster and contain a larger amount of configurable logic as well. In this paper, such technology is applied for designing high performance computing systems. Whereas hardware normally is static at runtime, software processes are being swapped at a high rate. However, in this paper a system where the *hardware configuration* is swapped at runtime is proposed.

Reconfigurable computing has grown to become an important and large field of research. The present target technology is Field Programmable Gate Arrays (FPGAs). These devices consist of a number of logic blocks connected with interconnection lines as seen in Figure 1. Each logic block consists of configurable combinational logic together with one or a few flip-flops. The logic is implemented in Look-Up Tables (LUTs) in Random Access Memory (RAM). The configuration of the function of each logic block and its connections to other blocks are given by the configuration bitstream loaded from outside the device and stored in distributed RAM on the FPGA. It is this bitstream that is of interest in this paper when discussing hardware configuration swapping.

Work is conducted either by using commercial FPGAs or by having new FPGA-like devices developed. A survey of can be found in (Hartenstein, 2000). The benefit of the reconfigurable technology compared to processors is the inherent parallel architecture. That is, the devices consist of a num-

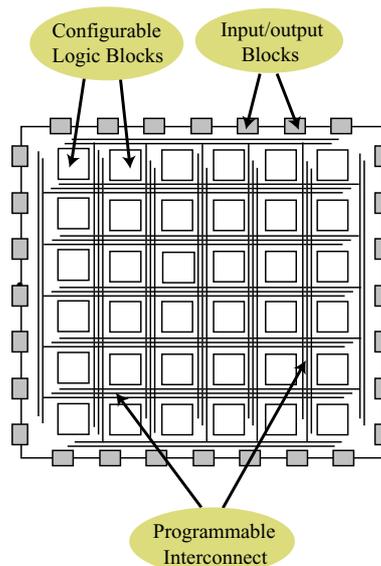


Figure 1: Illustration of a Field Programmable Gate Array (FPGA)

ber of simple processing units (logic blocks) with distributed communication links. The computation could be undertaken in a parallel scheme rather than by a single unit (processor) – as in a personal computer (PC).

Usually it is not interesting to have a *faster* solution if it results in *less* performance than some other system that still provides *sufficient* execution speed. However, for a set of applications, the execution speed is the limit for achieving high performance. Thus, the parallel reconfigurable technology would probably have benefits for problems with a parallel nature and when speed is a problem for other approaches.

The approach of applying reconfigurable logic for data processing has been demonstrated in some areas such as video transmission, image-recognition and various pattern-matching operations (handwriting recognition, face identification) (Villasenor and Mangione-Smith, 1997). Another area of interest is wireless systems, where tremendous computational capabilities are needed to allow for high data rates in the future (Rabaey, 2000).

Details about the most promising reconfigurable devices and how they can be applied for high performance computing are proposed in this paper. The next section of this paper describes the state-of-the-art within reconfigurable logic ap-

plied to adaptive system design. One of the most promising reconfigurable technologies is then presented in one section. This is followed by a proposal of a novel architecture with the potential of fast processing – by exploiting fast reconfiguration. Conclusions end the paper in the final section.

FPGA BASED ARCHITECTURES

This section includes an overview of the work undertaken on applying FPGAs for adaptive systems. To improve the overview, a set classification keywords are here presented (Torresen, 2002). They belong to the three different degrees of FPGA hardware adaptivity for providing configurable computing:

- **Static:** The configuration within the FPGA is the same throughout the lifetime of the system. This means no adaptivity at runtime.
- **Dynamic:** The configuration (or parts of it) is changed from time to time to introduce changes to the device. This represents rare adaptation.
- **Context switching:** A set of configurations are available in which the FPGA switch between using. This could provide computation speedup and represents frequent “adaptation” to the *best* configuration at a given time.

Most applications are implemented by applying the *static* approach – i.e. no adaptivity. However, *dynamic* systems have recently become more common. These allow the configuration to be upgraded when bugs are found *or* when the functionality of the system is to be changed.

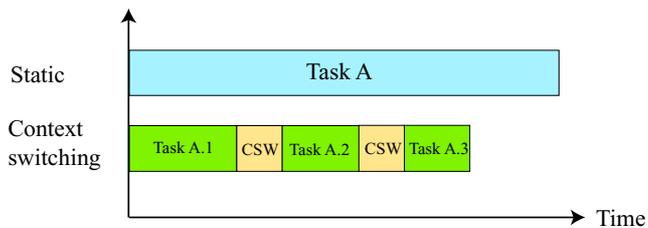


Figure 2: Illustration of a Context Switching FPGA Compared to a Static FPGA

The last degree – *context switching*, is applied in the work to be presented in this paper. The application area for context switching is in speeding up computation as depicted in Figure 2. Rapid swapping between successive configurations can give the FPGA-based system a considerable throughput. If a task A can be partitioned into a set of separate tasks (A.1, A.2 and A.3 in the example in the figure) to be executed one after the other, an FPGA configuration can be designed for each of them. Thus, each configuration is optimized for one *part* of the computation. During runtime, context switching (CSW) is undertaken and the total execution time for the task in the given example is reduced. The context switching time would have to be short to reduce the overhead of switching between the different configurations.

The fastest solution would be to have context switching FPGAs that are able to store a *set* of different configuration bitstreams and make the context switching in a single clock cycle. NEC has developed a prototype chip having this feature (Fuji et al., 1999). Further, in (Sidhu et al., 2000) a possible context switching FPGA architecture is proposed. This design is emphasized on both proving fast context switching as well as fast random access to the configuration memory. This is important, because you may want to change one or more of your configuration streams inside the FPGA at runtime. Still another device, capable of storing four configurations on-chip, is reported in (Scalera and Vazques, 1998).

Commercially available FPGAs do not yet provide configuration switching in one clock cycle and each new configuration would have to be downloaded externally. There has been undertaken some work based on context switching using such FPGAs. The main experience seems to be that today’s FPGAs are still requiring a (too) long reconfiguration time (Torresen, 2002). A task should be partitioned into *coarse* grained parts to reduce the overhead of switching between different configurations. Most devices require the complete configuration bitstream to be downloaded in one operation. The downloading time then increases with the size of the device. To reduce this overhead, Xilinx provides the family named Virtex, where the devices are *partially* reconfigurable. Then, the user can program a part of the device at a time and then reduce the time for switching to a new task to be executed. The Virtex family is among the newest introduced from Xilinx and these devices contain the largest amount of configurable logic on the market. Details about these devices and how they can be applied for context switching design are included below.

All commercial FPGAs are based on loading the configuration bitstream from outside the device. With their distributed processing architecture there could be interesting systems designed if an FPGA was able to *internally* write to its own configuration memory (Torresen, 1998). This would represent a self-reconfigurable FPGA.

XILINX VIRTEX ARCHITECTURE

The size of the devices in the Virtex family varies. However, the XCV1000-6 device (to be applied in this work) is one of the largest (the new Virtex II family contains devices about 10 times bigger than this device.). The architecture is based on a 64 by 96 array of Configurable Logic Blocks (CLBs) interconnected through a general routing matrix. Each CLB contains a routing matrix and two slices. Each slice consists of two look-up tables (LUTs) and two flip-flops (together with some control logic). The device contains 512 I/O pins. There are several features important for adaptability:

- Partial reconfiguration.
- Fast reconfiguration through the 8 bit SelectMAP interface.
- A large number of I/O-pins.

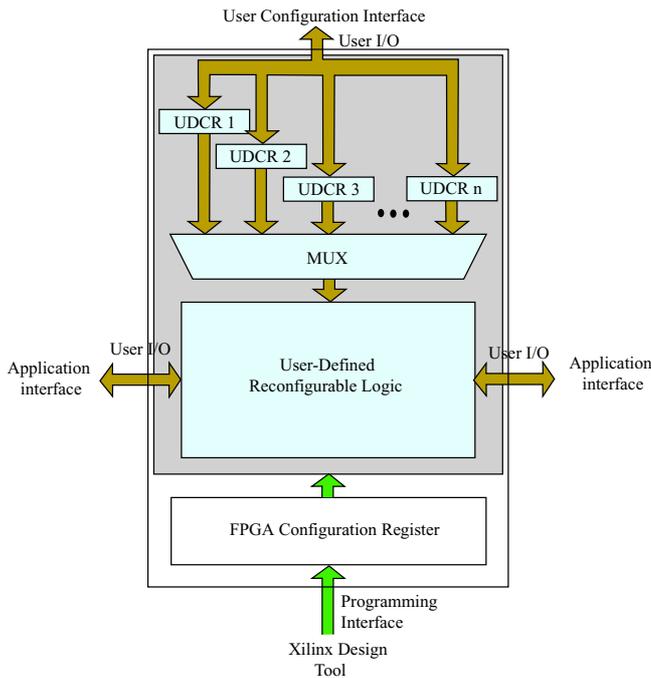


Figure 3: Implementing a “Custom” FPGA Within an FPGA Device

Look-Up Tables

The combinational logic is implemented in look-up tables. Each of them has four inputs and one output and is implemented as a 16 x 1 RAM. However, they can be used in several different modes:

- The two LUTs within a slice can be combined to create a 16 x 2 RAM, a 32 x 1 RAM or a 16 x 1 dual-port RAM.
- The LUT can be used as a 16-bit shift register.

Run-Time Performance

The Virtex devices are designed for high performance designs with synchronous system clock up to 200 MHz. This is from input pin to output pin. Several benchmarks (e.g. multiplier and parity tree) resulted in a performance of about 160 MHz. The devices are prepared for a PCI interface of 33 MHz or 66 MHz. Thus, for some applications, the interfacing may be limiting the performance.

IMPLEMENTING CONTEXT SWITCHING ON A VIRTEX FPGA

This section considers a novel approach to implement context switching on a Virtex FPGA device. It consists of implementing a novel structure in an FPGA which provides switching between a *set* of circuits in short time. This is based on earlier work of implementing a *user defined FPGA* inside an ordinary FPGA (Torresen, 1998; Sekanina and Ruzicka, 2000). However, it is extended to include a number of user defined configuration bitstreams as illustrated in Figure 3. Thus, in this scheme the “FPGA Configuration Reg-

ister” is *fixed* at runtime, representing the circuits on grey background given in the figure.

The design consists of a set of User Defined Configuration Registers (UDCRs). Only *one* is active at a time, and switching between them would be possible in one or a few clock cycles. The active UDCR configures the “User Defined Reconfigurable Logic” block. The drawback of this approach is that a certain amount of logic is required for defining this “virtual” FPGA for e.g. routing and representing the UDCRs. Thus, the amount of logic left as reconfigurable logic will be limited compared to the original amount of logic available. However, the number of logic blocks in the Virtex is large and each block can be configured in a set of ways.

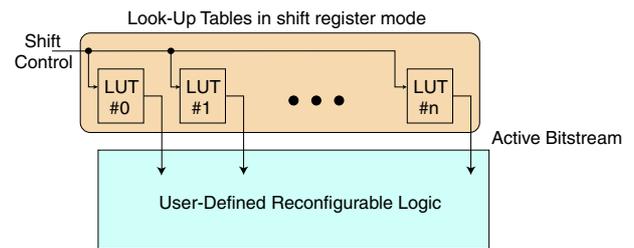


Figure 4: LUTs in a Virtex FPGA Used to Store a Set of Configuration Bitstreams

Each look-up table in a logic block can be configured into different modes as outlined in the previous section. The *shift* register mode is to be applied here. The proposed system – see Figure 4, applies a set of LUTs as shift registers to store a *set* of user defined configuration bitstreams. As illustrated in Figure 5, each LUT stores *one bit* of each bitstream and the maximum number of bitstreams is 16. The shift control is common for all the LUTs. Thus, when context switching between two bitstreams is to be undertaken, the LUTs shift one position and the next bitstream is presented on the LUT outputs. These outputs control the reconfigurable logic that is to be defined below.

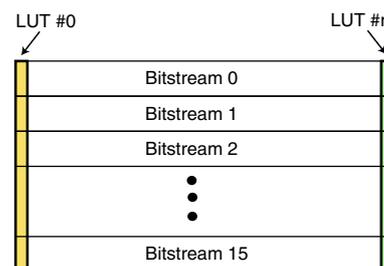


Figure 5: A Set of Bitstreams Stored in LUTs in a Virtex FPGA

To be able to store each configuration in the way described above, two LUTs are required as seen in Figure 6. The SR-LUT stores the 16 configuration bits and the LUT selects if the stored configuration is to be re-inserted into the SRLUT (*Executing*) or if a new configuration is to be loaded from outside (*Loading* by use of *New configuration* input line). Thus, the LUT is programmed to perform a multiplexer function – controlled by the *Executing/Loading* input. There are two LUTs in each slice, thus, one slice is used for each configuration bit line. This structure can be scaled up without

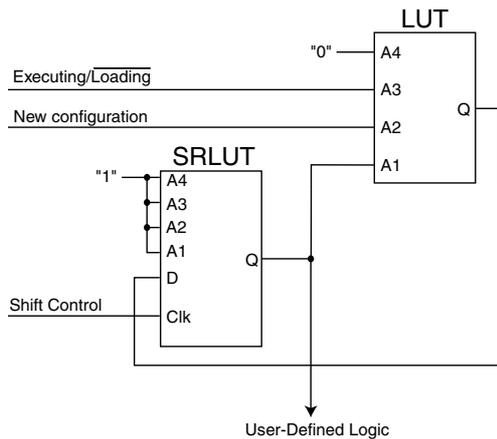


Figure 6: A Circuit For One Configuration Bit Line

additional time for routing by applying *hex lines* (routing to CLBs six CLBs away). Thus, we expect this system to run at a high clock rate.

The user-defined reconfigurable logic can be defined in several different ways. Here it is proposed one way which effectively utilizes the logic available in the FPGA. It is not possible to write to the LUTs from within the FPGA. Thus, they would have to be programmed before runtime (they can be configured from outside the device by re-writing a part of the configuration bitstream but this will be too slow for a fast context switching system.).

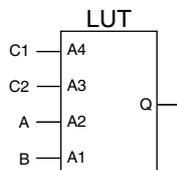


Figure 7: LUT Implementing Both Gate and MUX Function

In figure 7, one LUT is implementing both a logic function as well as routing (by a multiplexer function) of the two inputs A and B. C1 and C2 are used for control in the following way:

C ₁ C ₂	Function
00	NAND
01	B
10	A
11	NOR

By connecting such units into an array we are able to design a combinational circuit. By adding flip-flops and feedback connections, sequential circuits can be designed as well.

We are now able to design logic circuits optimized for a set of dedicated tasks. All configurations can be stored within an FPGA and context switching could be undertaken in one clock cycle by shifting the LUTs storing the configurations.

CONCLUSIONS

This paper has contained a presentation of reconfigurable logic and proposal of how it can be applied for high performance computation. This includes context switching in

real-time. The proposed architectures are expected to provide high speed processing in real-time systems.

ACKNOWLEDGEMENTS

The authors would like to thank Lukas Sekanina for inspiring and fruitful discussions during this work.

REFERENCES

- Fuji, T. et al. (1999). A dynamically reconfigurable logic engine with a multi-context multi-mode unified cell architecture. In *Proc. of Int. Solid-State Circuits Conf.*, pages 360–361.
- Hartenstein, R. (2000). A decade of reconfigurable computing: A visionary retrospective. In *Proc. of Int. Conference on Design Automation and Testing in Europe - and Exhibit (DATE)*. Munich, Germany.
- Rabaey, J. (2000). Silicon platforms for the next generation wireless systems - What role does reconfigurable hardware play?. In Hartenstein, R. et al., editors, *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, Lecture Notes in Computer Science, vol. 1896, pages 277–285. Springer-Verlag.
- Scalera, S. and Vazques, J. (1998). The design and implementation of a context switching FPGA. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 78–85.
- Sekanina, L. and Ruzicka, R. (2000). Design of the special fast reconfigurable chip using common FPGA. In *Proc. of Design and Diagnostics of Electronic Circuits and Systems - IEEE DDECS'2000*, pages 161–168.
- Sidhu, R. et al. (2000). A self-reconfigurable gate array architecture. In Hartenstein, R. et al., editors, *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, Lecture Notes in Computer Science, vol. 1896, pages 106–120. Springer-Verlag.
- Torresen, J. (1998). A divide-and-conquer approach to evolvable hardware. In Sipper, M. et al., editors, *Evolvable Systems: From Biology to Hardware. Second Int. Conf., ICES 98*, pages 57–65. Springer-Verlag. Lecture Notes in Computer Science, vol. 1478.
- Torresen, J. (2002). Reconfigurable logic applied for designing adaptive hardware systems. In *Proc. of the International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet (SSGRR'2002W)*. Scuola Superiore G. Reiss Romoli.
- Villasenor, J. and Mangione-Smith, W. (1997). Configurable computing. *Scientific American*, (6).