Reconfigurable Logic Applied for Designing Adaptive Hardware Systems

Jim Torresen

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 adaptive system design. There is a large range of real-world applications for such systems, including network infrastructure.**

I. Introduction

Software upgrades have become a natural way to improve functionality and operation of a computer. Hardware upgrades have so far mainly consisted of a substitution of the complete system or at least a printed circuit board substitution. However, with the advent of reconfigurable hardware we are provided almost the same upgrade possibility as with software. Upgrades could be undertaken by substituting the configuration of the reconfigurable device. This could improve functionality or like software processes represent a switching between *different* processes.

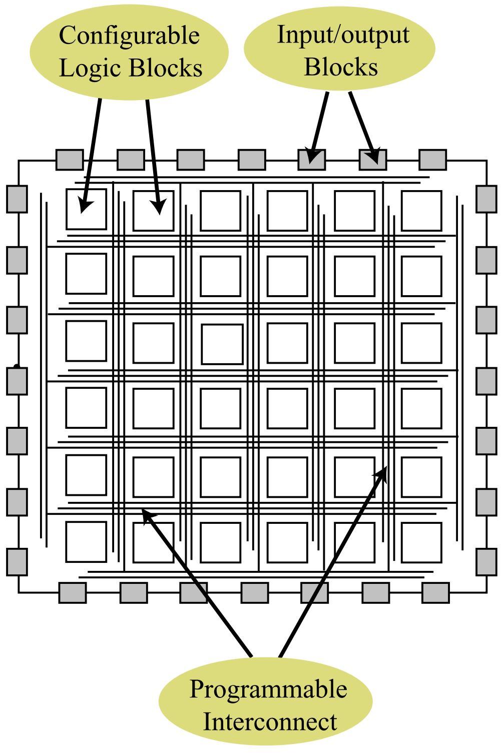

Fig. 1. Illustration of a Field Programmable Gate Array (FPGA).

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 Fig. 1. Each logic block consists of configurable combinational logic together with one or a few flip-flops. 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 SRAM on the FPGA. It is this bitstream that is of interest in this paper when discussing adaptivity.

Jim Torresen is with the Department of Informatics, University of Oslo, PO Box 1080 Blindern, N-0316 Oslo, Norway.
E-mail: jimtoer@ifi.uio.no, Web: http://www.ifi.uio.no/~jimtoer.

Work is conducted either by using commercial FPGAs or by having new FPGA-like devices developed. A survey of can be found in [1].

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) [2]. Another area of interest is wireless systems, where tremendous computational capabilities are needed to allow for high data rates in the future [3].

Details about the most promising reconfigurable devices and how they can be applied for adaptive system design are included in this paper. A part of this is how the large number of gates can be adopted in hardware system design.

The next section of this paper describes the state-of-the-art within reconfigurable logic applied to adaptive system design. In section III, a motivation for special hardware implementation is given. One of the most promising reconfigurable technologies is presented in Section IV. New architectures with the potential of fast processing – by exploiting fast reconfiguration, are proposed in Section V. Conclusions end the paper in Section VI.

II. 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 introduced. 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. However, several requirements of such a system exist [4]:

- Fallback to old configuration must be possible.
- Switching in ideally one clock cycle.
- On-line upgrade over Internet.
- Configuration bitstream must be encrypted to avoid reverse engineering.

In the future, automatic dynamic systems will probably arrive. These could autonomously upgrade the hardware as the environment (or data) changes or when bugs are detected in the system. One promising approach based on this idea is evolvable hardware [5].

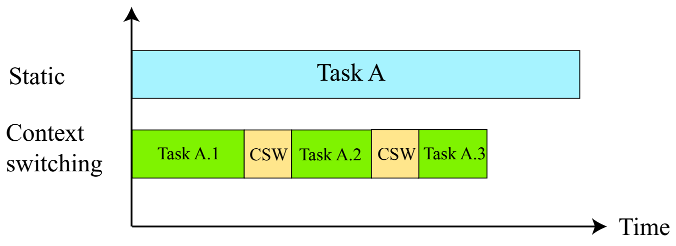

Fig. 2. Illustration of a context switching FPGA compared to a static FPGA.

The application area for *context switching* is in speeding up computation as depicted in Fig. 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 [6]. Further, in [7] 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 [8].

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. 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 (Virtex II contains larger devices than Virtex). Details about these devices and how they can be applied for adaptive system design are included below.

Gause *et al* have implemented a multimedia video-processing algorithm (2D SA-DCT) using several different FPGAs [9]. First the smallest (to have a shortest possible reconfiguration time) device in Altera FLEX 10K was used. It took 21 ms to reconfigure the device and with a processing time of 167 ns this gave a reconfiguration time 125,000 longer than the processing time! Somewhat lower overhead (2,500 times) was obtained with a Xilinx Virtex device having a reconfiguration time of 420 μs. Simmler *et al* are developing a multi-tasking FPGA coprocessor board and report a reconfiguration time of 12.4 ms for (probably the complete) Xilinx Virtex XCV400 [10].

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 [11]. This would represent a self-reconfigurable FPGA.

III. A Motivation for Special Hardware Implementation

This section includes an introduction to issues that would have to be considered when a given application is to be implemented. One of the most important decisions is to decide if the given application will benefit from special hardware implementation or not [12]. Basically, the two main platforms for system implementation are:

- Commercially available systems like a Personal Computer (PC) or an embedded system with appropriate interfaces. The latter kind is often called Commercial Off The Shelf (COTS) hardware.
- Specially designed hardware including microelectronic devices and microsystems.

With a focus on real-world applications, there is normally a demand for signal or image processing. Today, there is a number of possible target technologies to select among for a processing unit:

- PC microprocessor (CPU)
- Digital Signal Processor (DSP)
- Microcontroller
- FPGA
- ASIC (Application Specific Integrated Circuit)

The three first items implement an application as software, while the two latter ones normally implement it in dedicated hardware (not software). All these alternatives can be applied in special hardware implementations. However, PC and DSP-systems can also be purchased as COTS in many variants. Thus, this can often be the easiest and cheapest way to implement a system. However, there are several issues given in the list below showing when commercial systems are not applicable.

Memory and external interfaces would also be required. However, with the shrinking semiconductor technology, these units are in the future most likely to become integrated into the processing device. Such new kinds of devices are named System On Chip (SoC).

There are several issues that could make an application interesting for special hardware implementation (rather then using PC or COTS hardware):

- *Speed:* Real-time performance constraints. That is, specially designed hardware is required to run the application.
- *Size:* The hardware should be small to be used in mobile applications.
- *Power consumption:* Battery powered devices should minimize power consumption to extend the time between re-charging.
- *Reliability:* Safety critical systems could need special implementation to improve reliability issues.
- *(Analog) Interfacing devices not available:* When some parts of the system would have to be implemented in special hardware, it may not add much to include the complete hardware into this implementation.
- *Cost:* Special hardware implementation could prove to be cheaper than COTS for the number of units that is expected to be sold.

Several of these items are conflicting properties and priorities would have to be made. The most important issue for a real-world application is usually the performance (the correctness of its classification and/or control). Thus, the system designed should apply the technology that could provide the highest possible performance within the limits of speed, size, power consumption, reliability and cost.

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. Reconfigurable technology represents a parallel processing system with its many logic blocks. Thus, this technology would probably have benefits for problems with a parallel nature and when speed is a problem for other approaches.

IV. 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[1]. 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 LookUp 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.

A. Configuration

The maximum configuration bitstream programming rate is 66MHz. However, this requires handshaking. Configuration without handshake can be undertaken at 50MHz. The SelectMAP interface allows one byte at a time to be written and with a complete configuration data of 765,968 bytes, the configuration download time at 50MHz becomes:

$$T_{cdl} = 765,968 \cdot 20ns = 15.3ms$$

The smallest portion of the configuration memory that can be written or read is a *frame*. The number of configuration bits in a frame is 1248 bits (156 byte) for the XCV1000 – i.e. a fine grain partial reconfiguration. This requires a programming time of: $156 \cdot 20$ ns= 3.12 μs.

B. 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 PCI interface of 33 MHz or 66 MHz. Thus, for some applications, the interfacing may be the limitation in performance.

V. Implementing Context Switching on a commercial FPGA

This section considers various novel approaches to implementing context switching on a Virtex FPGA device. First we should distinguish two variants of context switching:

- Execution unit switching
- Data/template switching

The first item is used to speed up the "algorithm" itself. The second item is used to improve the processing speed by having all the needed data available in very short time. The two first sections below consider the first approach, while the last section introduces an architecture for data/template switching.

A. User Defined "FPGA" on FPGA

The method to be presented below 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 [11], [13]. However, it is extended to include a number of user defined configuration bitstreams as illustrated in Fig. 3. Thus, in this scheme the FPGA configuration register is *fixed* at runtime, representing the circuits on grey background given in the figure.

[1]The new Virtex II family contains devices about 10 times bigger than this device.

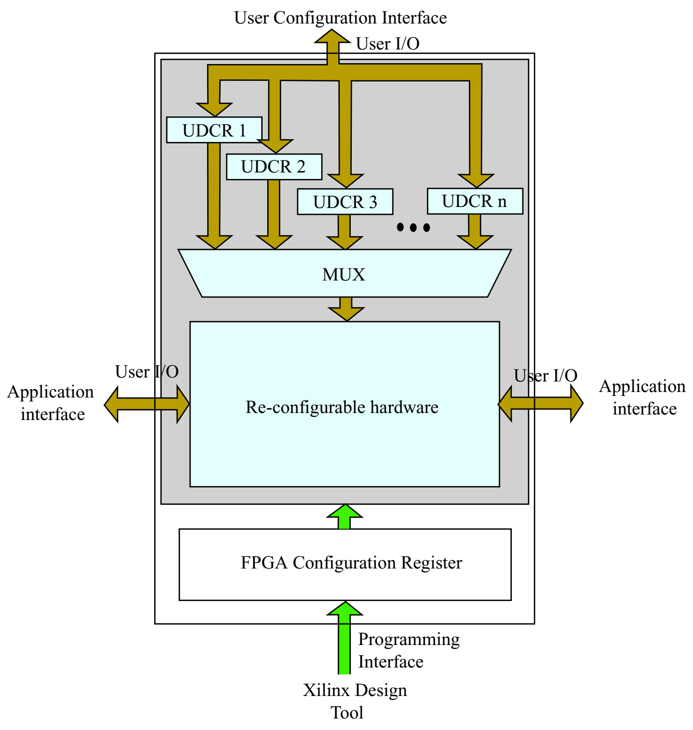

Fig. 3. Implementing a “custom” FPGA within an FPGA device.

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 “Reconfigurable hardware” 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. Thus, at the moment we are investigating how this can be exploited to limit the hardware overhead.

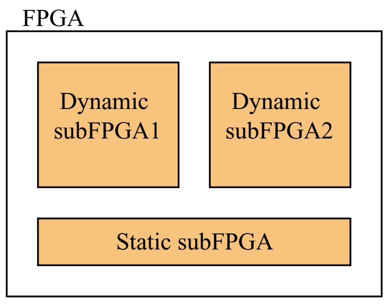

Fig. 4. An FPGA design partitioned into a set of subFPGAs.

B. Pipelined Configuration Bitstream Download

In this method, a part of the Virtex configuration bitstream is downloaded during runtime when context switching is to be undertaken. However, to reduce context switching time, the download is performed in a pipelined manner as seen in Fig. 4.

The FPGA is partitioned into a set of subFPGAs, and each is configured *separately* using partial reconfiguration. The static subFPGA is fixed and always active at runtime. This unit is managing the dynamic subFPGAs. These are to be reconfigured at runtime and are not always in operation. That is, only *one* of the dynamic subFPGAs is active at a given time. Thus, the one not being active can load its new configuration – by partial reconfigurations, while the other one is executing as depicted in Fig. 5. In this way, the FPGA becomes a pipelined unit where the configuration is pipelined with execution. The number of dynamic subFPGAs could be increased if the execution time is *shorter* than the configuration time.

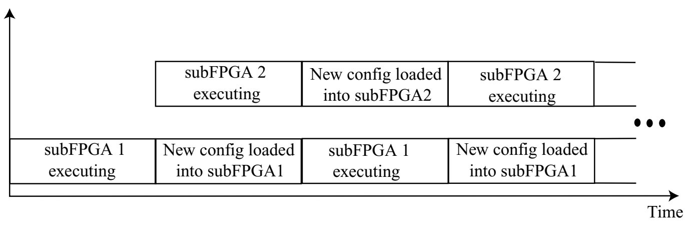
subFPGA 2 executing
New config loaded into subFPGA2
subFPGA 2 executing
subFPGA 1 executing
New config loaded into subFPGA1
subFPGA 1 executing
New config loaded into subFPGA1
Time

Fig. 5. Timing in a pipelined configuration of an FPGA.

C. Data/Template Switching Architecture

Many applications consist of a large amount of data to be processed. Data could be both information to analyze and/or templates used for matching.

A novel system for template matching is outlined in Fig. 6 where two buffers switch between *loading* and *matching*. Thus, while loading data into one buffer from the Data Input, the other buffer is applied for high speed matching with the Template Register. When the loading buffer is full, matching should be finished for the other buffer. Then, the buffers exchange operation.

The clock speed of the *loading* buffer is the same as that of the input data stream. The speed of the *matching* buffer is as high as possible and substantially higher than the input data stream. The larger the buffer, the longer time is available for matching. Matching is based on *shifting* the matching buffer to compare with the template stored in the Template Register. This is designed for matching consecutive units in the data stream. Depending on the template size, a number of templates may very well be matched in parallel (not shown in the figure). The set of templates are stored in external RAM and loaded into the template register through a multibit databus. With 512 I/O pins available we may suggest to use 3/4 of them for this bus. This results in loading of 48 bytes of data in parallel. Another option is to apply the built-in RAM of the Virtex device (Block SelectRAM).

C.1 An Example

To illustrate how this system could work we would like to study the matching in a 2 Mbit/s data stream. This is 262,144 bytes/s. Suppose that each buffer is of size 1

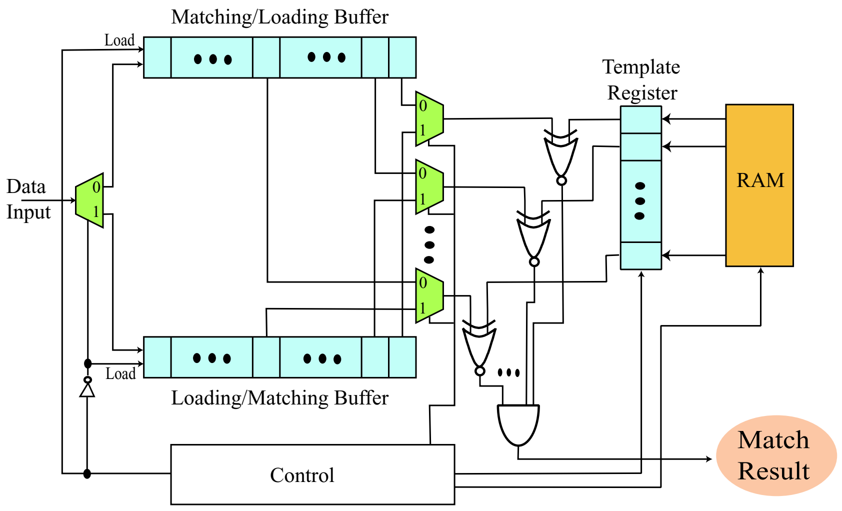

Fig. 6. An architecture for fast string matching.

Kbyte (1024 byte). The time for loading 1 Kbyte from the "Data Input" is:

$$t_l = \frac{1024}{262,144} = 3.91ms$$

Suppose that the matching is undertaken at 100 MHz (reasonable since benchmarks can run at 160 MHz). A full shift (shifting one byte at a time) of the matching buffer would take:

$$t_s = 1024 \cdot 10ns = 10.24\mu s$$

Ignoring the time for switching templates, there is time available for matching a large number of templates in sequence:

$$N_t = \frac{3.91ms}{10.24\mu s} = 381$$

If the data is in packet format, the architecture should be redesigned and one or a few packets could be stored in the buffer at a time. The buffers should then probably be implemented as a random access memory (RAM) structure to be able to match specific positions in a packet.

VI. Conclusions

This paper has contained a presentation of reconfigurable logic and proposal of how it can be applied for the design of adaptive systems. This includes context switching in real-time. The proposed architectures are expected to provide high speed processing in real-time systems.

Acknowledgements

The author would like to thank Dr. Oddvar Søråsen for inspiring and fruitful discussions during this work.

References

[1] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospectiv e," in *Proc. of Int'l.Conference on Design Automation and Testing in Europe - and Exhibit (DATE)*. 2000, Munich, Germany.

[2] J. Villasenor and W.H. Mangione-Smith, "Configurable computing," *Scientific American*, , no. 6, 1997.

[3] J.M. Rabaey, "Silicon platforms for the next generation wireless systems - What role does reconfigurable hardware play?.," in *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, R.W. Hartenstein et al., Eds., Lecture Notes in Computer Science, vol. 1896, pp. 277–285. Springer-Verlag, 2000.

[4] G. Prophet, "FPGAs + the Internet = upgradable product," *EDN Europe*, pp. 28–38, November 2000.

[5] J. Torresen, "Evolvable hardware as a new computer architecture.," in *Proc. of the International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet*, January 2002.

[6] T. Fuji et al., "A dynamically reconfigurable logic engine with a multi-context multi-mode unified cell architecture.," in *Proc. of Int. Solid-State Circuits Conf.*, 1999, pp. 360–361.

[7] R. Sidhu et al., "A self-reconfigurable gate array architecture.," in *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, R.W. Hartenstein et al., Eds., Lecture Notes in Computer Science, vol. 1896, pp. 106–120. Springer-Verlag, 2000.

[8] S.M. Scalera and J.R. Vazques, "The design and implementation of a context switching FPGA," in *IEEE Symposium on FPGAs for Custom Computing Machines*, 1998, pp. 78–85.

[9] J. Gause et al., "Static and dynamic reconfigurable designs for a 2D shape-adaptive DCT.," in *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, R.W. Hartenstein et al., Eds., Lecture Notes in Computer Science, vol. 1896, pp. 96–105. Springer-Verlag, 2000.

[10] Simmler et al., "Multitasking on FPGA coprocessors.," in *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, R.W. Hartenstein et al., Eds., Lecture Notes in Computer Science, vol. 1896, pp. 121–130. Springer-Verlag, 2000.

[11] J. Torresen, "A divide-and-conquer approach to evolvable hardware.," in *Evolvable Systems: From Biology to Hardware. Second Int. Conf., ICES 98*, M. Sipper et al., Eds., pp. 57–65. Springer-Verlag, 1998, Lecture Notes in Computer Science, vol. 1478.

[12] J. Torresen, "Possibilities and limitations of applying evolvable hardware to real-world application.," in *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, R.W. Hartenstein et al., Eds., pp. 230–239. Springer-Verlag, 2000, Lecture Notes in Computer Science, vol. 1896.

[13] L. Sekanina and R. Ruzicka, "Design of the special fast reconfigurable chip using common FPGA," in *Proc. of Design and Diagnostics of Electronic Circuits and Systems - IEEE DDECS'2000*, 2000, pp. 161–168.