Journal of Advanced Computer Science and Technology, *1 (4) (2012) 337-346*

www.sciencepubco.com/index.php/JACST

Reconfigurable Architectures

Abida Yousuf, Roohie Naaz Mir, and Hakim Najeeb-ud-din (SMIEEE)

Department of Information Technology,
E-mail: abidayousuf.17@gmail.com
Department of Computer Sciences and Engineering
E-mail: naaz310@yahoo.co.in
Department of Electronics and Communication Engineering
E-mail: najeeb@nitsri.net

Abstract

In the area of computer architecture, designers are faced with the trade-of between flexibility and performance. The architectural choices span a wide spectrum, with general-purpose processors and application specific integrated circuits (ASICs) at opposite ends. General-purpose processors are not optimized to specific applications, they are flexible due to their versatile instruction sets that allow the implementation of every computable task. ASICs are dedicated hardware devices that can achieve higher performance, require less silicon area, and are less power-consuming than instruction-level programmable processors. However, they lack in flexibility. Reconfigurable computer architectures promise to overcome this traditional trade-off and achieve both, the high performance of ASICs and the flexibility of general-purpose processors.

Keywords: *ASICs, Compile Time Reconfiguration, FPGA, Reconfigurable Computing, Run Time Reconfiguration.*

1 Introduction

The concept of reconfigurable computing has existed since 1960, when Gerald Estrin's land mark paper proposed the concept of a computer made of a standard processor and an array of reconfigurable hardware. The main processor would control the behavior of the reconfigurable hardware. Once the task was done, the hardware could be adjusted to do some other task. This results in hybrid computer

structure combining the flexibility of software with speed of hardware. But unfortunately this idea was far ahead of its time in needed electronic technology.

Reconfigurable devices like field-programmable gate arrays (FPGAs) contain an array of computational elements whose functionality is determined through multiple programmable configuration bits. For a given application, at a given time, the spatial structure of the device will be modified such as to use the best computing approach to speed up that particular application. If a new application has to be computed, the device structure will be modified again to match the new application. Contrary to the Von Neumann computers, which are programmed by a set of instructions to be executed sequentially, the structure of reconfigurable devices are changed by modifying all or part of the hardware at compile-time or at run-time, usually by downloading a logic onto the device. In reconfigurable systems, Flexibility is possible because 'the application must always adapt to the hardware' in order to be executed and the Performance is possible because 'the hardware is always adapted to the application.' If we consider two scales, one for the performance and the other for the flexibility, then the General Purpose Computers (GPP) can be placed at one end and the Application Specific Integrated Circuits (ASICs) at the other end as illustrated in Fig. 1. Reconfigurable computing is intended to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware [1] [2].

Ideally, we would like to have the flexibility of the GPP and the performance of the ASIC in the same device. We would like to have a device able 'to adapt to the application' on the fly. We call such a hardware device a *reconfigurable hardware* or *reconfigurable device* or *reconfigurable processing unit (RPU)* in analogy to the Central Processing Unit (CPU) [2].

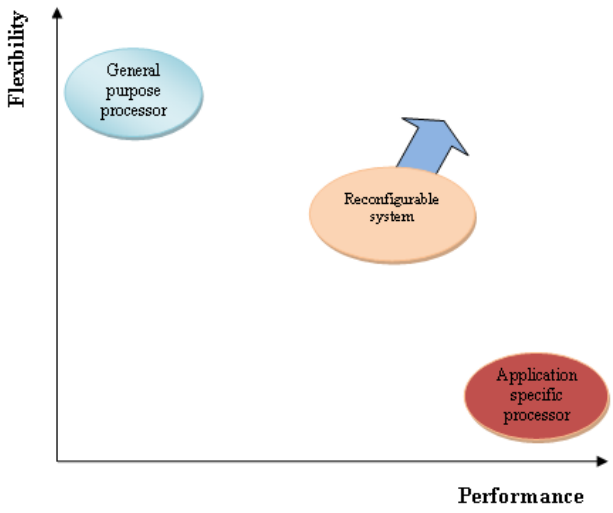

Fig.1: Flexibility v/s performance of processor classes.

2 Vision for Reconfigurable Computer

Reconfigurable computing is defined as a computer having hardware that can be reconfigured to implement application-specific functions and is typically shown in Fig. 2. The reconfigurable computer systems combine microprocessors and programmable logic devices, typically field programmable gate arrays (FPGAs), into a single system. Reconfigurable computing allows applications to map computationally dense code to hardware. This mapping often provides orders of magnitude improvements in speed while decreasing power and space requirements.

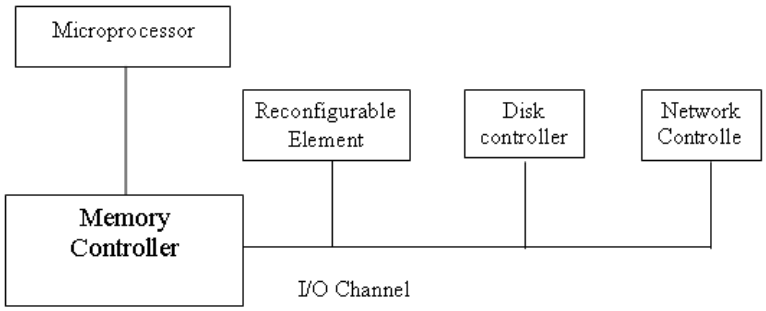

Fig. 2: A typical schematic of a reconfigurable computer.

As mentioned above, RC uses programmable FPGAs. Current FPGA technology provides more than 10 million logic elements, internal clock rates over 200MHz, and pin toggle rates approaching 10 Gb/s. With these large devices, the scope of applications that can target an FPGA has dramatically increased. The challenges for using FPGAs effectively fall into two categories: ease of use and performance [3] [4] [5].

Ease of use issues includes the following:

1. Methodology of generating the 'program' or bit stream for the FPGA.
2. Ability to debug an application running on both the microprocessor and FPGA simultaneously.
3. Interface between the application and the system or Application Programming Interface (API).

Performance issues include the following:

1. Data movement (bandwidth) between microprocessors and FPGAs.
2. Latency of communication between microprocessors and FPGAs.
3. Scalability of the system topology.

Performance is the fundamental reason for using RC systems. By mapping algorithms to hardware, designers can tailor not only the computational components, but also perform data flow optimization to match the algorithm.

Today's FPGAs provide over a terabyte per second of memory bandwidth from small on chip memories as well as tens of billions of operations per second. Transferring data to the FPGA and receiving the results poses a difficult challenge to RC system designers. In addition to bandwidth, efficient use of FPGA resources requires low latency communication between the host microprocessor and the FPGAs. Achieving low latency in the presence of high bandwidth communication is one of the most difficult obstacles facing a system designer. When low latency is achieved, scaling and optimization across multiple computational elements can occur, often called load balancing [6] [7] [8].

2.1 Coupling with Host Processor

There are several ways in which reconfigurable element may be coupled with the host processor as illustrated in Fig. 3 [9] [10].

Initially, reconfigurable hardware can be used solely to provide reconfigurable functional units within a host processor. This allows for a traditional programming environment with the addition of custom instructions that may change over time. Here, the reconfigurable units execute as functional units on the main microprocessor data-path, with registers used to hold the input and output operands. Then a reconfigurable unit may be used as a coprocessor. A coprocessor is in general larger than a functional unit, and is able to perform computations without the constant supervision of the host processor. Instead, the processor initializes the reconfigurable hardware and either sends the necessary data to the logic, or provides information on where this data might be found in memory.

Next, an attached reconfigurable processing unit behaves as if it is an additional processor in a multiprocessor system. The host processor's data cache is not visible to the attached reconfigurable processing unit. Therefore, there is a higher delay in communication between the host processor and the reconfigurable hardware, such as when communicating configuration information, input data, and results. However, this type of reconfigurable hardware does allow for a great deal of computation independence, by shifting large chunks of a computation over to the reconfigurable hardware.

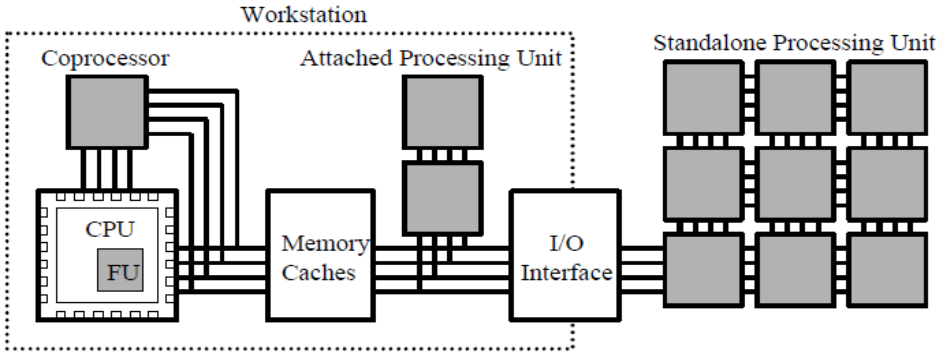

Fig. 3: Different levels of coupling in a reconfigurable system, Reconfigurable logic is shaded.

Finally, the most loosely coupled form of reconfigurable hardware is that of an external standalone processing unit. This type of reconfigurable hardware communicates infrequently with a host processor. This model is similar to that of networked workstations, where processing may occur for very long periods of time without a great deal of communication.

3 Reconfigurable Architecture Systems

The research efforts in reconfigurable computing are divided into two groups according to the level of the used hardware abstraction or *granularity* [11] as follows:

1. Fine-grained reconfiguration System.
2. Course-grained reconfiguration System.

3.1 Fine-Grained Reconfiguration System

The hardware abstraction in this approach is the gate and register level. By reconfiguration of registers, gates, and their interconnections, the internal structure of typical functional units is changed. This approach has been enabled by the development of SRAM-based *field-programmable gate arrays* (FPGAs).

3.1.1 Basic System Architecture

The basic architecture for fine-grained reconfigurable systems is shown in Fig. 4. The reconfigurable hardware (FPGA) is connected to a host processor by a memory bus. In many systems, the FPGAs have separate data-paths to the memories which require dual-ported RAMs or arbitration logic.

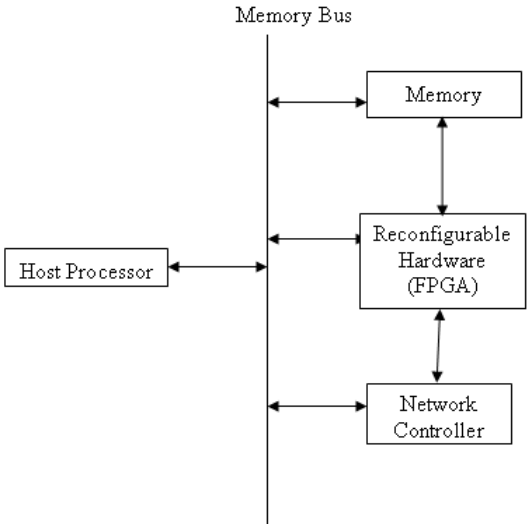

Fig. 4: Basic system architecture for fine-grained reconfiguration.

Reconfigurable architectures can be differentiated with *coprocessors* and *attached processors*. The former contain only a small number of FPGAs and are mostly found in embedded applications. Attached processors denote architectures that employ arrays of FPGAs. These architectures often use programmable switches to interconnect FPGAs with FPGAs or memories and are targeted to general-purpose processing. For many applications, FPGA-based computing machines achieve sound speedups compared to general-purpose processors. The sources of these speedups are manifold. First, dedicated function units and data paths are specialized to the actually required operations and number of bits. Second, parallelism is exploited at the operation level. Third, customized memory architectures provide the memory bandwidth required to keep the parallelized function units busy. The control is specialized, and hard-wired instruction fetching and decoding are eliminated. Typical of FPGAs with their highly regular array-like structure are deep pipelines, e.g. bit-serial and systolic designs. Pipelining is often used for digital signal processing (DSP) applications, where latency is traded in for high throughput [12].

3.1.2 The main Characteristics of Fine Grained Systems are as under:

1. Maps small groups of instructions onto the reconfigurable device.
2. A lot of communication is needed between those parts.
3. Low granularity implies greater flexibility.
4. Needs more power and more space, However higher delays occur (more routing is required).
5. Logic cells have at least 1 bit.
6. A wide variety of applications benefits from this paradigm with speed ups between 2 and 5.

3.2 Coarse-Grained Reconfiguration System

The hardware abstraction in this approach is based on a set of fixed blocks, like functional units, processor cores, and memory tiles. Reconfigurability is achieved by reprogramming the switches that interconnect processor cores, ALUs, and memory tiles. It is often argued that these architectures should be called adaptive rather than reconfigurable. In fact, the distinction between reconfigurability and programmability is blurred. All research efforts in this area target general-purpose computing and the organizational issues of future generation processors. This work is driven by the following two trends of IC technology and application demands respectively [12].

- The trend from IC technology is the increased importance of wire delay in the future one billion transistor single-chip processors. Along with decreasing silicon feature sizes, the speed of gates doubles every 5 years, but the wires slow down by a factor of 4 in the same time. This 8-fold increase in the relative wire-to-gate delay limits the scalability of contemporary general-

purpose processors that use centralized control and register files. Future processor architectures will have to consist of some sort of local computing clusters that are interconnected.

- The application trend is the increased importance of data-streaming workloads. The term data-streaming describes DSP algorithms that process large amounts of data. Examples are video compression or graphics, as used in multi-media applications. The support of multimedia applications already drives the instruction-set design for current processors. However, to achieve higher and scalable performance for future generation processors, parallelism will have to be exploited to a much greater extent and at more levels than it is done now.

The main Characteristics of Coarse-grained Systems are as under:

1. Maps large, computationally-intensive parts to the reconfigurable array.
2. Potential to reduce communication between internal blocks.
3. Better optimized for standard data path applications.
4. Every CLB (configurable logic block) has more than 1 Bit (2, 4 or more bits are usually).
5. Optimized for large bit-width computation.
6. More power efficient, lower delays (less routing is required).
7. Suffers for 1 bit-width instructions.

4 Reconfiguration Models

There are two reconfiguration models: compile-time reconfiguration (CTR) and run-time reconfiguration (RTR), as shown in the schematic 5. CTR is the most often used model in fine-grained reconfigurable computing. RTR is a rather new technique and only few application examples have been reported yet.

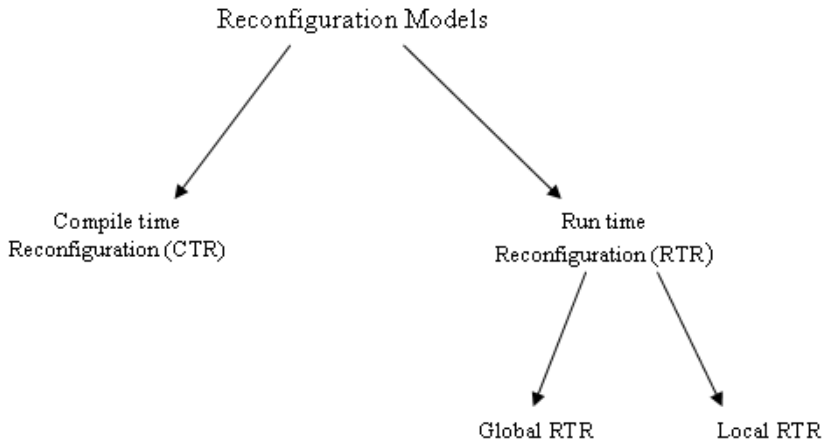

Fig. 5: Reconfiguration Model.

4.1 Compile-Time Reconfiguration (CTR)

The reconfigurable hardware system is configured prior to the application's run-time, i.e. at compile-time, and remains static during the run-time. The main goal of CTR architectures is to achieve high performance as hardware accelerators. The critical, i.e. most time-consuming portions of an application are extracted and synthesized to run on the reconfigurable hardware. The extraction of critical program portions is either done automatically or by user assistance. For applications that contain long integer operations, e.g. cryptography, or linear systolic applications, these machines have achieved a higher performance than conventional supercomputers.

Reconfigurable architectures for emulation and prototyping of digital systems belong also to the category of CTR systems. They are used during the design process of ASICs and general-purpose designs to verify by emulation. As digital systems are often described in *hardware description languages* (HDLs), these emulation and prototyping architectures can be seen as HDL accelerators.

4.2 Run-Time Reconfiguration (RTR)

In RTR, also called *dynamic reconfiguration*, the hardware is reconfigured during the application's run-time. An application is split into segments that are executed successively, utilizing the same reconfigurable hardware. RTR can increase the *functional density* of FPGAs. The functional density is defined as

$$D = 1 - AT$$

This metric includes the area A in some unit hardware resources and the execution time T for a task. Generally, successfully applied RTR increases T slightly by the additional reconfiguration time, but decreases A to a much greater extend. RTR systems are used in two cases.

In the first case, RTR architectures substitute larger CTR systems or systems with several ASICs in embedded systems. Here, the goal of RTR is to reduce the cost. The second case is applications where CTR systems are not feasible due to excessive hardware requirements. Compared to CTR, RTR involves three additional problems: *temporal partitioning*, *reconfiguration overhead*, and *inter-configuration communication*.

In temporal partitioning, an application is split into time-exclusive segments. For each of these segments dedicated hardware is designed. Many applications break down into segments or operational phases quite naturally. As the reconfiguration takes places during the application's run-time, the reconfiguration time becomes an overhead. This reconfiguration overhead, i.e. the ratio of the reconfiguration time to the execution time must be kept as small as possible. Therefore, FPGAs with short reconfiguration times are of utmost importance. The problem of inter-configuration communication arises, when one FPGA configuration produces

results that are used by other configurations. To avoid this, results are stored temporarily in a memory or in FPGA registers/RAM cells that are not destroyed during reconfiguration.

RTR can be further divided into *global* and *local* RTR. In global RTR, the complete FPGA is reconfigured. To achieve an acceptable FPGA utilization, the application must be partitioned into segments with roughly equally-sized hardware requirements. An application class for RTR is pattern matching, where huge amounts of data have to be compared with different templates. For each template dedicated hardware is designed, leading to high speedups compared to software solutions. If the number of templates is high, CTR systems become infeasible. RTR systems however, have been built in these cases for free-text database searching or scanning the human genome database. In local RTR, only a part of the FPGA is reconfigured, while the other parts of the FPGA remain active. This gives greater flexibility than global RTR, because the segment sizes are not required to be equal. However, the partitioning and design process for local RTR systems is very complicated, as the correct interfacing of the different concurrent designs on the FPGA must be assured. Examples for local RTR systems have been reported for artificial neural networks.

5 Conclusion

Reconfigurable computing is becoming an integral part of research in computer architectures and software systems. By placing the computationally intense portions of an application onto the reconfigurable hardware, the overall application can be greatly accelerated. An architectural paradigm shift is required to meet the complex problems facing high-performance computing (HPC). Although challenges abound, RC systems allow us to explore solutions that are not viable in today's limited computing environments. The benefits in size, speed and power alone make RC systems a necessity.

This paper surveys reconfigurable computing, architecture and design issues. The main trends in architectures are fine grained and course grained systems and the main trends in design methods are compile time and run time reconfiguration models

References

[1] Christophe Bobde, "*Introduction to Reconfigurable Computing*," Springer, (2007).
[2] K .Hwang, "*Advanced computer architecture*," McGraw-Hill, (1993).
[3] R. Hartenstein, "*Basics of reconfigurable computing*," Springer, (2007), pp. 451-501.

[4] Stephen Brown, and Jonathan Rose, "Architecture of FPGAs and CPLDs: A Tutorial,"Available at *http://www.eecg.toronto.edu/~jayar/pubs/brown/survey pdf.*

[5] T. J. Todman, G.Constantinides, O.Mender, and P. Cheung, "Reconfigurable computing: architectures and design methods," *Proceedings of the IEE - Computers and Digital Techniques*, Vol. 152, No. 2, (2005), pp. 193-207.

[6] D.A Buell and Kenneth, L. Pocek, "An introduction to custom computing machines," *The Journal of Super Computing*, Vol. 9, No. 3, (1995), pp. 219-230.

[7] R. Hartenstein, "A decade of reconfigurable computing," *Proceedings, Conference on Design, Automation and Test in Europe*, Munich, Germany, (2001), pp. 642-649.

[8] R. Hartenstein, H. Grünbacher, "The Roadmap to reconfigurable computing ," *Proceedings of the 10th International Conference on Field-Programmable Logic and Applications, Springer Verlag, Heidelberg, Lecture Notes in Computer Science*, Villach, Austria, Vol. LNCS 1896, (FPL 2000).

[9] R. Hartenstein, "*Morphware and Configware*," Springer, (2006).

[10] K. Compton, S. Hauck, "Configurable computing: A survey of systems and software," *Northwestern University, Dept. of ECE Technical Report*, (1999), pp. 1-2.

[11] Stamatis, Vassiliadis, D. Soudris, "*Fine and course grained reconfigurable computing*," Springer, (2007).

[12] S. Hauck, "The role of FPGAs in reprogrammable systems," *Proceedings of the IEEE* Vol. 86, No. 4, (1998), pp. 615-638.

[13] R. Hartenstein, "Course grained reconfigurable architectures," *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, Yokohama, Japan, (2001), pp. 564-570.

[14] S. Hauck and A. Dehon, "*Reconfigurable Computing: The theory and practice of FPGA-Based computation*," Morgan Kaufmann Series in Systems on Silicon, (2008).