World Applied Sciences Journal 28 (8): 1051-1060, 2013
ISSN 1818-4952

DOI: 10.5829/idosi.wasj.2013.28.08.11930

Performance Oriented Design Perspectives of Run-Time Reconfigurable Computing Systems

M. Aqeel Iqbal and Shoab A. Khan

Department of Computer Engineering College of Electrical and Mechanical Engineering
National University of Sciences and Technology (NUST), Pakistan

Abstract: Reconfigurable computing is becoming an integral part of emerging scientific research since last few decades. The main theme beyond this new technology is to integrate the performance benefits of application specific integrated circuits with the hardware flexibility of programmable processors in a single chip. The reconfigurable computing devices like field programmable gate arrays have already been playing a vital role in the enhancement of the existing technology but still reconfigurable computing is suffering from many technological drawbacks including the huge configuration overheads of existing reconfigurable devices, the lack of supporting high level software tools, the support of immature compilation tools and even the lack of supporting operating systems for the existing optimally reconfigurable devices. This research paper introduces the reconfigurable computing from its hardware and software perspectives being existent and further demanded for future work and emerging research dimensions in this area. Many factors have been pointed out for further improvements so that the under laying technology can be rapidly boosted up so that to support the requirements of the emerging scientific applications.

Key words: Reconfigurable Computing · Configuration Overheads · FPGAs · ASICs · RFUs

INTORDUCTION

In the domain of computing there have been two primary methods that have been used traditionally for the execution of computations or algorithms. On the one extreme side the first known method is to use the *Application Specific Integrated Circuits (ASICs)* to perform the operations in hardware. Due to the fact that these ASICs have been designed to perform a specific given computation, they are very fast and efficient when executing the exact computations for which they were designed. But along with this advantage they are suffering with a problem that, after fabrication process the circuits cannot be altered. On the other extreme side the second known method is to use the programmable processors which are a far more flexible solution. Processors in fact execute a set of instructions to perform a required computation. By changing the software instructions, the functionality of the system can be altered without physically changing the hardware. However, the disadvantage of this flexibility is that the performance of the computing system suffers and is far below as is compared to that of ASICs as shown in Figure 1 and Figure 2. The processor must read each instruction from memory, determine its meaning after an op-code decoding process and only then can execute it. All this activity introduces a huge amount of performance overhead for each of individual operation. Reconfigurable computing is intended to fill this gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware [1, 2] as is shown in Table 1.

Table 1: Computing Systems Characteristics

Programmable	Fixed	Reconfigurable
Computes any computable function	Computes one function	Computes a number of function
Function is defined after fabrication	Function is defined before fabrication	Function is defined after fabrication
Connections on fabric are fixed	Connections on fabric are fixed	Connections on fabric are programmable

Corresponding Author: M. Aqeel Iqbal, Department of Computer Engineering College of Electrical and Mechanical Engineering National University of Sciences and Technology (NUST), Pakistan.

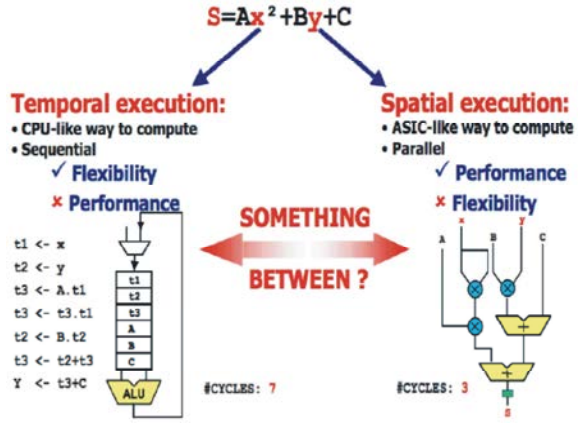

Fig. 1: ASIC vs. GPP Computing

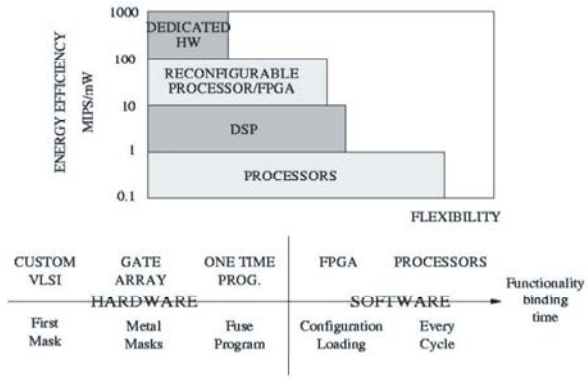

Fig. 2: Comparison of Computing Systems

Computing Using ASICs: The antithesis of a programmable general purpose processor is an ASIC based solution. In ASICs, functional units are dedicated to individual operations to be computed and wired together to match precisely the computations being performed. The advantages an ASIC has over programmable processor can be summarized as:

- Considerably very less executional overhead is needed to control the mapping of functional units to operations and the routing of data-path values between them. On the other side a programmable processor has an overhead which is manifest in time space and chip area.
- Due to very optimal specialized functional units and less execution overhead circuitry around each one, more functional units can be fit into the same chip die area and hence chip density can be enhanced dramatically.
- Because the operation of each functional unit is already known and planned out in advance, hence due to this fact the functional unit idleness overhead can be minimized. On a programmable processor, some of the available functional units might never be used by a specific application [1].

Computing Using GPPs: General-purpose computing devices like CPUs being introduced are specifically intended for those cases where, economically, we cannot dedicate sufficient spatial resources to support an entire algorithm or computational task or where engineers do not know enough about the required computational tasks prior to the fabrication to hardwire the functionality. The key ideas behind general-purpose computing are the delayed binding of functionality until device is fabricated and hence introducing the hardware flexibility and exploit the temporal reuse of limited functional units hence causing partial resource utilizations. Delayed binding of functionality and temporal reuse of available resources work closely together and occur at many scales to provide the characteristics that are well known now from general-purpose computing devices. Users are quite accustomed to exploiting these properties so that unique hardware is not required for every task or application. This basic theme recurs at many different levels in our conventional systems.

On Market Level: Instead of the dedicating a machine design to a typical single application or a set of applications, the design effort may be utilized for many different applications. Hence system may demonstrate dynamic nature of its computing characteristics.

On System Level: Instead of the dedicating an expensive machine to a single application, the machine may perform many different applications at different times by running different sets of instructions. Hence machine demonstrates diversity in its structure.

On Application Level: Instead of the spending precious real estate to build a separate computational unit for each of newly demanded different function, the central resources may be employed to perform these functions in sequence with an additional input, an instruction, telling it how to react or behave at each point of computation in time.

On Algorithm Level: Instead of the fixing the algorithms which an application uses, an existing general-purpose machine can be reprogrammed and reused with new techniques and algorithms as they are developed.

On Application User Level: Instead of the fixing the function of the machine at the supplier side, the instruction stream specifies the function, allowing the end user to use the machine as best suits his application needs. Machines may be used for functions which the original designers did not conceive from it. Further, machine reaction or behavior may be upgraded in the application field without incurring any hardware or hardware handling costs. In the past, processors were logically or virtually the only devices which had these characteristics and served as general-purpose building blocks. Today, many other emerging devices, including reconfigurable devices, also exhibit the key properties and benefits associated with general-purpose computing.

In reference to the computational theory, it is artifact that algorithmically any computation can be generally represented as sequential or concurrent combinations of abstract data-flow and control-flow graphs, with the nodes of the graphs being representing the primitive operations such as integer addition, subtraction, multiplication or division etc. The primary function of a computer has always been to evaluate such kinds of computational algorithmic graphs mechanically or electronically so as to accomplish some real life goal. Of course, the real computer processors cannot operate on such abstract graphs directly. Instead, the algorithmic programs are encoded as a set/collection of machine instructions which can be executed one after another in a specific sequence. But this is just an artifact of the design of the machine, intended originally to simplify the processor task and perhaps the programmer task too. In fact the modern processors re-expose instruction level parallelism by dynamically decoding the short sequences of machine instructions into their corresponding data-flow and control-flow forms before their execution. Regardless of the fact that how a program is physically encoded or decoded, the data-flow and control-flow graphs technically represent the true computations being performed.

In concept, the functional units are all a computer needs to evaluate the operations in a data-flow graph. In reality, a computer must also support the physical movement of data among the available different functional units, as well as to and from memory. Digital system designers face the fundamental trade-off between flexibility and performance when making a choice between different computing systems. Customized hardwire based technology like ASICs provides a high performance grade and low power consumption due to the fact of being specialized, but lack the flexibility since any kind of new

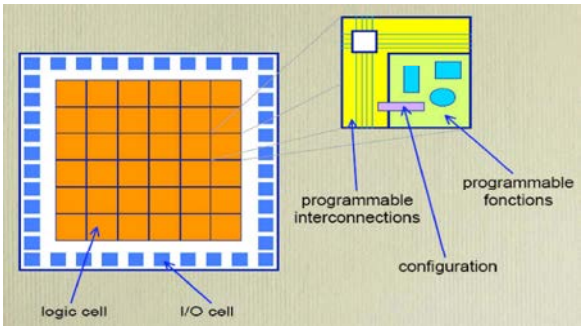

Fig. 3: FPGA Internal Architecture

changes require whole redesign and rewiring. On the other hand the software based solutions operate with software instructions. Conceptually a great flexibility comes from easy development and maintenance of the software code, but execution of instructions as a machine program introduces high overheads in performance and area.

Reconfigurable Computing technology is introduced to fill the gap between hardware and software based solutions of design. The main goal of reconfigurable computing is to achieve the performance better than that of software solutions, while maintaining a bit greater flexibility than hardware solutions [2]. Reconfigurable computing devices are composed of many computational elements known as *Configurable Logic Blocks (CLBs)* whose functionality is determined through programmable configurations. These configurable elements are connected by a set of programmable routing resources as shown in Figure 3. The idea of configuration is to map the desired logic functions of a design to the processing units within a reconfigurable device and use the programmable interconnects to connect processing units together to form the necessary circuit [3]. Great flexibility comes from the programmable nature of processing elements and routings. Performance can be made better than software based approaches due to reduced execution overhead. Under this definition, *Field Programmable Gate Array (FPGA)* is a form of reconfigurable computing device [3]. FPGAs and other reconfigurable computing devices have been shown to accelerate a variety of new applications, such as encryption algorithms and streaming applications. Obviously, by the stated definitions, a programmable device cannot be an ASIC and vice versa. However, reconfigurable devices such as FPGAs share common characteristics of both the programmable processors and hardwired ASICs. On the one hand, FPGAs can implement ASICs-style circuits, while on the other hand; they are infinitely reprogrammable and thus

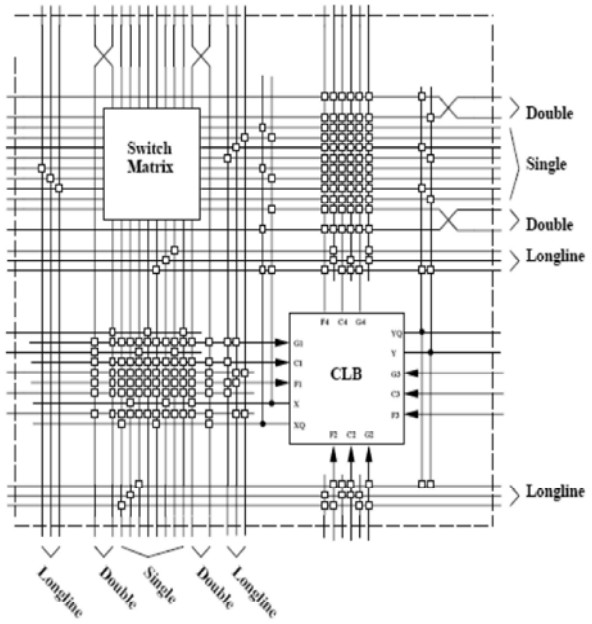

Fig. 4: FPGA Routing Structure

immanently demonstrate the behavior of general-purpose processor. This leads to the question of whether reconfigurable hardware can capture some of the advantages of ASICs within a general-purpose computing environment.

Computing Using Reconfigurable Devices: Reconfigurable computing (RC) devices generally fill their silicon area on a chip with a large number of computing blocks also known as configurable built-in primitives, interconnected via a programmable switches network [3] as shown in Figure 4. The operation of each built-in primitive can be programmed as well as interconnection patterns can be adjusted or modified. Computational tasks can be implemented spatially on the device with intermediate flowing directly from the producing function to the receiving function. Since thousands of reconfigurable units can be placed on a single silicon chip, the significant data-flow may occur without crossing chip boundaries. To first order, one can think about turning an entire task into hardware data-flow and mapping it on the reconfigurable substrate. Reconfigurable computing generally provides spatially-oriented processing rather than the temporally-oriented processing typical of programmable architectures such as microprocessors. The key differences between the reconfigurable systems and the conventional processors are:

Program Instruction Distribution: Instead of the broadcasting a new instruction to the existing functional units on every new cycle, the instructions are locally configured at hand, allowing the reconfigurable device to compress instruction stream distribution and effectively deliver more instructions into active silicon on each cycle.

Spatial Routing of Intermediate Streams: As the space permits, the intermediately generated data values are routed in parallel from producing function to consuming function rather than forcing all communication to take place in time through a central resource bottleneck.

Finer-GrainedProgrammablePrimitives: Reconfigurable computing devices provide a large number of independently configurable or programmable building blocks known as programmable primitives, allowing a greater range of computations to occur per time step. This effect is largely enabled by the compressed instruction distribution.

Distributed Deployable Resources: Resources such as memory interconnect and functional units are distributed and deployable based on need rather than being centralized in large pools. Independent local access allows the reconfigurable logic designs to take the advantage of high, local, parallel on-chip bandwidth, rather than creating a central resource bottleneck.

Reconfigurable computing is supposed to be one of the best alternatives to the conventional superscalar and VLIW paradigms [1]. The main difference between a reconfigurable device and a standard micro-processor is in the instruction stream. In its original or purest form, a reconfigurable computing device has no cycle-by-cycle instruction stream. Instead of it, the device is commonly configured by loading a complete specification of the functions of each part of the device at once. Once the device is being configured, the intention is for the device to run in that configuration for a decent interval of time before being again reconfigured. Each configuration may demonstrate an ASIC-like circuit, specialized for the particular task at hand. Changing configurations might take anywhere from a few clock cycles to a few thousand clock cycles [1, 3]. In accordance with the simpler programming mechanism, the dynamic forwarding crossbar switch is replaced by a less flexible configurable switch network for making static connections among the functional units and the short queues of retiming registers associated with each of functional units take place of traditional processor shared, multi-ported register file.

The very famous and familiar 90-10 rule asserts that 90% of execution time is consumed by about 10% of a program code, that 10% generally being inner loops. Reconfigurable computing devices excel in those special cases where the computation represented by a configuration is repeated many times again and again, so that the time required to load a configuration can be amortized over a long execution time and/or overlapped with other execution. When all of the important loop bodies of an application can be configured to fit within the reconfigurable computing machine, there would seem to be no need for the execution overhead of a fully dynamic instruction fetch and issue mechanism, allowing the machine to be leaner and more efficient.

By reducing the hardware to just the essentials needed to support computation, the reconfigurable computing design scales better to larger sizes than the more complex superscalar and VLIW architectures. Although an expansion of the configurable routing network would cause it to grow quadratically with the increase in the number of the functional units, it only needs to grow enough to support the connectivity required by the real applications. Furthermore, unlike a superscalar or VLIW machine architecture, the reconfigurable computing hardware can easily exploit not only the simple Instruction Level Parallelism (ILP) but also the inter-iteration and thread parallelism, making reconfigurable computing well poised to work with very large numbers of functional units.

In particular the reconfigurability can be obtained by using the devices known as *Field Programmable Gate Arrays* [4]. In FPGAs, the involved technology offers the possibility to vary the functionality of the chip at design time as well as at run time, depending on the value assumed by particular inputs, called *Configuration Streams* [4]. FPGA design consists of a layout of identical cells, implementing from simple Boolean operations to more complex 4-bits arithmetic, connected through a vast routing network of programmable wiring switches. Both the functionality as well as the connections of the cells among wires can be programmed through the setting of particular values held in SRAM based cells. This kind of devices presents therefore high flexibility, at the expense of a lower speed and integration than that exhibited by ad-hoc ASICs. Reprogrammability is in general exploited by a particular kind of programmable processors, called *Reconfigurable Processors.* Figure 5 is showing the core of a typical reconfigurable processor.

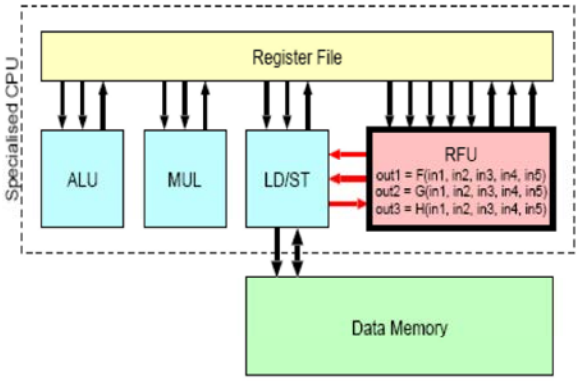

Fig. 5: Reconfigurable Processors Core

These in general consist of a micro-processor core coupled with a reconfigurable device element, implemented using an FPGA technology, that can adopt one or more functions on-the-fly. Different reconfigurable processor classes have been defined, based on the manor in which the micro-processor and reconfigurable device elements have been connected with each others [5, 6].

FPGA Used As Computing Device: Field-programmable gate arrays (FPGAs) have really dramatically revolutionized the way the digital hardware has been designed and built since their commercial introduction in the mid-1980. Over this period of time, these commodity digital parts have instantly become invaluable system components due to their ability to implement huge number of different logic functions efficiently and their ability of being easily reconfigured as system hardware requirements change under the effect of the running applications. Gradually growing progressive improvements in technology has really increased the logic density/capacity of these computing devices from the equivalent of a few handful simple TTL logic gates just a decade ago to the density of a recent huge sized application specific integrated circuits today commercially available.

Several commercial vendors have already been providing devices with density/capacities of more than one million logic gates. With this available abundance of logic density and programmable routing resources, many new application areas for FPGAs have become feasible and are under further research. Recent advances in logic emulation technology have made scalable hardware based systems with hundreds of FPGA devices available for verification of prototype digital logic designs systems and for use as custom computing platforms for

applications with large amounts of fine-grained as well as coarse-grain parallelism [4, 5]. While hardware system density as well as capacity has matured to an extent considerably in recent systems, the underlying software systems still needed to automatically map user designs and applications to hardware are still in their infancy. Technically the greatest limitation to the use of contemporary multi-FPGA based systems technology for computation and emulation is the amount of time overhead required to place and route the individual circuits inside the individual FPGA devices. In certain existing systems, this compilation time is on the magnitude of hundreds of CPU hours as opposed to tens of minutes for typical processor based computing systems/platforms [6].

Such a long turn-around time overhead from conceptual development to physical implementation (emulation) significantly limits the on-the-fly modification or upgrading capability of the most of the existing FPGA based computing applications. Almost all existing FPGAs place and route systems are very optimized to use as much of the logic and routing resources in a target device as virtually possible. For most of the FPGA designers being developing a single logic design over several weeks or months, the compilation time overhead measured in hours is quite tolerable and preferable to the greater expense of purchasing a larger device that will be only partially used and filled. For designers using FPGA devices for reconfigurable computing, systems compile time overheads of several hours are unreasonable as compared to compile time overheads for processors that typically require only few seconds or maximum of few minutes [7]. Furthermore, much of this compile time is currently spent performing placement and routing on functional logic components, used across a set of FPGA computing applications, when a library of pre-placed and pre-routed macros in conjunction with a macro-based floor planner could be used instead [1, 7].

The use of FPGAs in a system is generally a trade-off when compared to its possible ASIC solution [8]. A fully customized implementation of a hardware circuit will always give us the higher performance than an FPGA based system implementation but likely at a some what higher cost due to reduced production volume compared to the commodity FPGA [6]. It is quite possible to make similar trade-offs in the implementation of designs inside the FPGA based systems. Figure 6 is showing the different possible ways of coupling a reconfigurable core with a standard processing unit like CPU. For a fixed sized

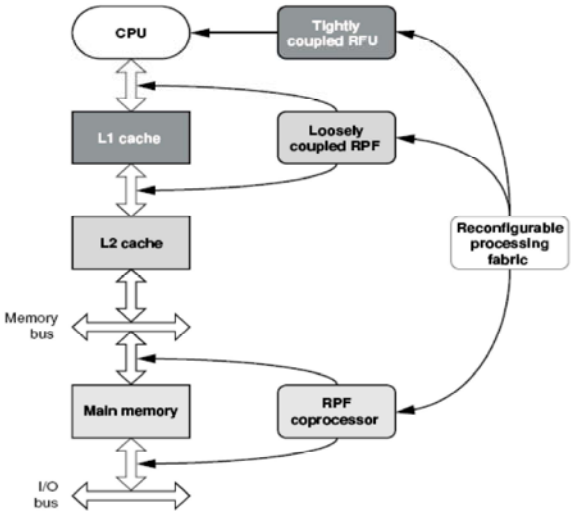

Fig. 6: Coupling Approach for Reconfigurable Fabric

device it is possible to vary the amount of time overhead needed to place and route a circuit design based on the layout algorithm chosen and the amount of design logic and interconnect targeted to the device. Typically it has been observed that fine-grained place and route approaches currently employed by FPGA vendors do not scale up well in terms of compilation time overhead versus quality with increased device logic capacity and chip density and that there is a need for new kind of under laying layout approaches.

No doubt the logic placement, the floor planning and the routing algorithms for VLSI layout have been widely studied for many decades; the little work has been done in the optimization of these algorithms to find a feasible solution quickly at the cost of an intermediate increase in required resources [8]. Generally, work in this kind of research area has been focused on achieving the optimal layout solution, in terms of minimized required resources or optimal performance, at the cost of a significant increase in search evaluation time overheads [7]. Recent trends in field of reconfigurable computing indicate that in many special cases the incremental changes in FPGA circuits may be required over the execution period of an application. By isolating placement and routing resources into some kind of specific regions of the device these changes can be made without replacing and rerouting the large amounts of logic circuitry left unmodified by the change. This additional requirement of isolation requires special consideration in the layout process and additional cost since; in general, existing architectures are not designed to directly support this design style [8].

FPGA Based System Development: Digital circuit designing has evolved rapidly over the last few decades. The digital logic circuits were initially designed with vacuum tubes and transistors. Integrated Circuits (ICs) were invented over the time where logic gates were placed on a single chip. The first integrated circuit (IC) chips were SSI (Small Scale Integration) chips where the gate count was very small. Then the chips known as MSI (Medium Scale Integration) came. With the advent of LSI (Large Scale Integration) technology, the digital designers could put thousands of logic gates on a single chip. The chip designers now began to use the circuits and logic simulation techniques to verify the functionality of the complex building blocks of the magnitude of few thousands transistors. With the advent of VLSI (Very Large Scale Integration) technology, the digital designers could design single chips with more than 100,000 transistors. Due to the enormous complexity of these circuits, it was not possible to verify these circuits on a bread board. Computer-aided techniques became critical for verification and design of VLSI digital circuits. Computer based simulation type programs to do an automatic placement and routing of circuit also became very much popular.

Hardware Description Languages (HDLs): Since so many years the traditional programming languages such as "Fortran", "Pascal" and "C" were being used to describe computer programs that were sequential in their execution. Similarly, in the digital design field *Hardware Description Languages (HDLs)* came into existence. HDLs allowed the digital designers to model the concurrency of processes found in active hardware modules or elements [9]. Hardware description languages such as Verilog-HDL and VHDL became popular out of all of the invented HDLs. Verilog-HDL was invented in 1983 at Gateway Design Automation and later on the VHDL was developed under contract from DARPA. Both Verilog-HDL and VHDL simulators to simulate large digital circuits quickly gained acceptance from designers [9].

Even though the hardware description languages were popular for logic verifications, the digital designers had to manually translate the HDL-based design into a schematic circuit with interconnections between logic gates. The advent of logic synthesis changed the digital design methodology radically. Digital circuits could be described at a register transfer level (RTL) by use of an HDL. Thus now the digital designer had to specify how the data flows between registers and how the design processes the data. The details of logic gates and their

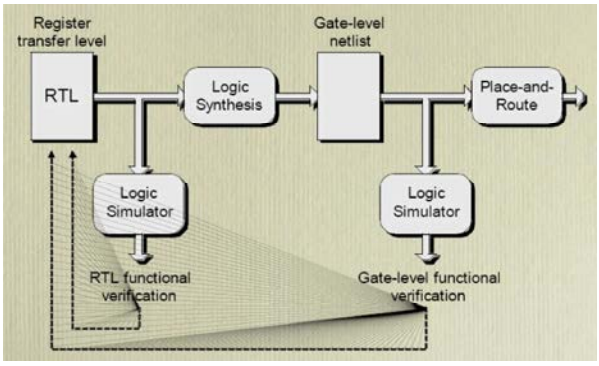

Fig. 7: RTL Based Design Simulations

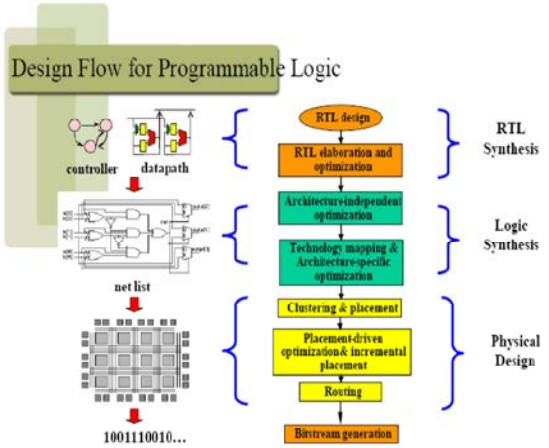

Fig. 8: Design Flow for Programmable Logic

interconnections to implement the logic circuits were automatically extracted by logic synthesis tools from the provided RTL description. Digital designers now no longer had to manually place gates to build digital circuits. They could describe complex circuits at an abstract level or even at algorithmic level in terms of functionality and data flow by designing those circuits in HDLs [9]. Logic synthesis tools would implement the specified computational functionality in terms of logic gates and logic gate interconnections. HDLs also began to be used for system-level design. HDLs were used for the simulation of system boards, interconnection buses, FPGAs (Field Programmable Gate Arrays) and PALs (Programmable Array Logics). A common approach is to design each IC chip, using an HDL and then verify system functionality via simulation.

HDLs Based Development Flow: A typical digital logic design flow and logic simulations steps for designing VLSI based integrated circuits are shown in Figure 7 and Figure 8. The digital design flow shown is typically used by digital designers who use HDLs. In any design,

specifications are written first of all at initial step. Specifications in fact describe abstractly the in depth functionality, interface and overall internal architecture of the digital circuit to be designed. A behavioral description is then created to analyze the design in terms of functionality, performance and other high-level issues.

The behavioral level description is manually converted into an RTL description in an HDL. The digital designer has to describe the data flow that will implement the desired digital logic circuit. From this point onward, the design process is done with the assistance of available EDA tools. Logic synthesis tools convert the RTL description to a gate-level netlist. A gate-level netlist is in fact a description of the logic circuit in terms of logic gates and connections between them. Logic synthesis tools ensure that the gate-level netlist meets timing, area and power specifications. The gate-level netlist is input to an automatic placing and routing tool, which creates a circuit layout. The circuit layout is verified and then fabricated on a chip. Thus, most design activity is concentrated on manually optimizing the RTL description of the circuit. After the RTL description is done ok, EDA tools are available to assist the designer in further processes. Designing at the RTL level has shrunk the digital design cycle times from few years to few months and even few days [10]. It is also possible to do many design iterations in a short period of time. Behavioral synthesis tools have begun to emerge recently. These tools can create RTL descriptions from a behavioral or algorithmic description of the circuit. As these tools mature, digital circuit design will become similar to high-level computer programming. Designers will simply implement the algorithm in an HDL at a very abstract level. EDA tools will help the designer convert the behavioral description to a final IC chip.

Existing Potential Research Areas: An important aspect of reconfigurable computing devices is their ability to reconfigure the functionality of computational units or elements in response to the changing operating conditions and data sets of the running application. While SRAM-based FPGAs have supported few hundred milli-seconds reconfiguration overheads rates for some time, only recently devices have been created that allow for rapid device reconfigurations at run-time and are known as Run-time Reconfigurable Devices [10, 11]. Dynamically reconfigurable FPGAs known as *DPGAs* contain multiple interconnect and the logic configurations for each programmable location in a reconfigurable computing device. Often these computing architectures have been designed to allow the configuration switching in a small number of system clock cycles measuring nano-seconds rather than milli-seconds [11]. While several DPGA devices have been developed in research environments, none are currently commercially available due to the large configuration overhead costs associated with the required large configuration memory. In order to promote the hardware reconfigurations at lower cost, several commercial FPGA families have been introduced recently that allow for fast partial reconfiguration of FPGA functionality from off-chip memory resources available [11]. A significant challenge to the use of these reconfigurable computing devices is the development of compilation softwares and techniques which will partition and schedule the order in which the computations will take place and will determine about which circuitry must be changed. While some preliminary work in this thrilling area has been completed, more advanced tools are still needed to fully leverage the new hardware technology. Other software approaches that have been applied to dynamic reconfiguration include the definition of hardware sub-routines and the dynamic reconfigurations of the instruction sets.

While high-level compilation for micro-processors has been an active research area for few decades, the development of compilation technology for reconfigurable computing is still in its infancy and needs more research dimensions to be explored [11, 12]. The compilation process for FPGA-based systems is often very complicated by the lack of identifiable coarse-grained structures in fine-grained FPGAs and the dispersal of the logic resources across many pin-limited reconfigurable computing devices on a single computing system or platform. In particular, since most reconfigurable computing systems contain multiple programmable devices, design partitioning forms an important aspect of most compilation systems. Several compilation systems for reconfigurable computing hardware have followed a traditional multi-device ASIC design flow involving pin-constrained device partitioning like concepts [13].

In order to overcome the pin limitations and to achieve the full logic utilization on a per-device basis using this approach, either excessive internal device interconnection or I/O counts have been needed. A hardware virtualization approach has also been outlined that promotes the high per device logic utilization. Following the design partitioning and placement techniques, the inter-FPGA wires is scheduled on inter-device wires at compilation time, allowing the pipelining of communication. Inter-device pipelining also

forms the basis of several FPGA system compilation approaches that start at the behavioral level. A high-level synthesis technique has also been described that outlines inter-FPGA scheduling at the RTL level. Further more the functional allocation has also been performed that takes into account the amount of logic available in the target system and available inter-device interconnect. Combined resource communications and functional resource scheduling has also been performed to fully utilize the available logic and communication resources. The Inter-FPGA communication and FPGA-memory communication have also been virtualized since it has been recognized that memory rather than inter-FPGA bandwidth is frequently the critical resource in the reconfigurable computing systems and individual device synthesis using RTL compilation.

Reconfigurable Computing Applications: Reconfigurable computing systems that have been based on FPGAs have shown impressive speedups for a large number of computing applications by customizing the underlying hardware logic of the computing platform or system to create exactly the hardware functionality required [14]. Typically, due to the large size of the circuits so created to perform the computation, multiple FPGA devices are needed for design implementation. A large number of recently completed projects have used hundreds of FPGA devices in concert as a reconfigurable computing platform to solve computational challenges such as shortest-path search calculations, array sorting, FFT calculations and special purpose processor implementations [15]. Latest advancements in the high-level compilation technology for these computing domains will likely lead to a rapid increase in the number of potential applications for reconfigurable computing [16-18]. As the complexity of reconfigurable computing applications and targeted platforms grows, the ability of application designers to map the desired designs by hand to reconfigurable hardware becomes limited by the amount of time needed to analyze the complex variety of new hardware implementation tradeoffs available [15]. These limitations have given rise to many new automated high-level design flows for emerging multi-FPGA based reconfigurable computing systems.

CONCLUSION

Although the reconfigurable computing has been an active research area since last few decades, the reconfigurable computing technology is still suffering from many emerging engineering challenges. In the next few years it is expected that there would be many radical changes in all aspects of this design space. FPGA architectures are supposed to cope with the ever more powerful fabrication technologies, altering conventional interconnections and resource mixes to best support multi-million gate designs. Reconfigurable computing also puts its own demands on the emerging chip architectures like system on chip technology. The scientific community must find a way to make their desired features cost-effective for the commercial chip vendors, either by making them valuable to be inserted into general-purpose architectures or by growing a new commercial market for reconfigurable computing chips.

REFERENCES

1. Compton, K. and S. Hauck, 2002. Reconfigurable computing: a survey of systems and software, ACM Computing Surveys, 34(2): 171-210.
2. Philip Garcia, Katherine Compton, Michael Schulte, Emily Blem and Wenyin Fu, 2006. An overview of reconfigurable hardware in embedded systems, EURASIP Journal on Embedded Systems, pp: 1-19.
3. Benkrid, Khaled, 2008. High Performance Reconfigurable Computing: From Applications to Hardware IAENG International Journal of Computer Science, vol. 35, issue 1. February.
4. Compton, K. and S. Hauck, 2000. An introduction to reconfigurable computing, IEEE Computer Society.
5. Katherine Compton, 2008. Reconfiguration Management in Reconfiguration Computing, (ed.) S. Hauck & A. Dehon, Morgan Kaufman, pp: 65-86.
6. Francisco Barat, R. Lauwereins and G. Deconinck, 2002. Reconfigurable Instruction Set Processors from a Hardware/Software Perspective, IEEE Transactions on Software Engineering, 28(9): 847-862.
7. Hartenstein, R., 2001. A Decade of Reconfigurable Computing: A Visionary Retrospective". In Design, Automation and Test in Europe Conference (DATE), 642(64): 13-16.
8. Kuon, I. and J. Rose, 2006. Measuring the gap between FPGAs and ASICs, in Proceedings of the ACM/SIGDA 14th International Symposium on Field-Programmable Gate Arrays (FPGA '06), pp: 21-30.
9. DeHon, A., J. Adams, M. DeLorimier, *et al*., 2004. Design patterns for reconfigurable computing, in Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04), pp: 13-23.

10. Hartenstein, R., 2002. Trends in reconfigurable logic and reconfigurable computing, in Proceedings of the 9th IEEE International Conference on Electronics, Circuits and Systems (ICECS '02), pp: 801-808.
11. Hauck, S., 1998. The Roles of FPGAs in Reprogrammable Systems Proceedings of the IEEE, 86(4): 615-638.
12. Bauer, L., M. Shafique and J. Henkel, 2008. A Computation and Communication-Infrastructure for Modular Special Instructions in a Dynamically Reconfigurable Processor; Proc. of IEEE International Conference on Field Programmable Logic and Applications (FPL 2008), Heidelberg, Germany, pp: 203–208, pp: 8-10.
13. Peck, W., E. Anderson, J. Agron, J. Stevens, F. Baijot and D. Andrews, 2006. Hthreads: A computational model for reconfigurable devices. In 16th International Conference on Field Programmable Logic and Applications, Madrid, Spain.
14. Todman, T.J., G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk and P.Y.K. Cheung, 2005. Reconfigurable computing: architectures and design methods, IEE Proceedings: Computers and Digital Techniques, 152(2): 193-207.
15. Hauck, S., T. Fry, M. Hosler and J. Kao, 2004. CHIMAERA: Integrating a Reconfigurable Unit into a High-Performance, Dynamically-Scheduled Superscalar Processor, in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 12: 2.
16. Mueen Uddin, Asadullah Shah, Raed Alsaqour and Jamshed Memon, 2013. Measuring Efficiency of Tier Level Data Centers to Implement Green Energy Efficient Data Centers, Middle-East Journal of Scientific Research, 15(2): 200-207.
17. Hossein Berenjeian Tabrizi, Ali Abbasi and Hajar Jahadian Sarvestani, 2013. Comparing the Static and Dynamic Balances and Their Relationship with the Anthropometrical Characteristics in the Athletes of Selected Sports, Middle-East Journal of Scientific Research, 15(2): 216-221.
18. Anatoliy Viktorovich Molodchik, 2013. Leadership Development: A Case of a Russian Business School, Middle-East Journal of Scientific Research, 15(2): 222-228.