Emerging Requirements Of Reconfigurable Computing Systems For Performance Enhancement

[1]M. Aqeel Iqbal, [2]Asia Khannum, [2]Saleem Iqbal and [2]M. Asif
[1, 2]Department of CE, College of E & ME, NUST, Pakistan
[1][DSE, Faculty of E & IT, FUIEMS, Pakistan]

***Abstract* – The reconfigurable computing is intended to fill the gap between the non-flexible but high speed application specific integrated circuits based technology and the most-flexible but slow speed general purpose processor based technology. The main idea behind the reconfigurable computing technology is to tightly glue the performance and flexibility aspects of the both existing technologies into one grain. This kind of tight coupling of both computing technologies on a single silicon chip requires new technological advancement in area of software engineering, computer science and computer engineering. This research paper is intended to present a comprehensive survey about the existing reconfigurable computing systems and their software and hardware level requirements for further enhancement of the existing technology barrier. Many new aspects of existing systems are required to be boosted up at the software and hardware level of such kind of advanced computing platforms to support the next generation computing demands**.

Keywords - Configuration Streams, FPGA Technology Reconfigurable Computing, Reconfigurable Devices, Reconfigurable Functional Units

I. Brief Introduction To Reconfigurable Computing Technology

The General Purpose Processors (GPPs) based computers have served us well over the past few decades. Their broad band applicability for commercial as well as scientific purpose has led to a wide spread use and volume commoditization. Flexibility of dynamic behavior through instruction set architecture of the underlying processor allows a single computing machine to perform a multitude of functions and can be deployed into applications un-conceived at the time the device was programmed and manufactured. This kind of architectural flexibility inherent in general-purpose processor based machines was one of the key components of the computer revolution being accelerated since last few decades. Till now, the programmable micro-processors have been basic driving engine behind the general-purpose computing machines. In such a kind of programmable processor based machines the micro-processors mostly focus on the heavy reuse of a single or a small set of underlying functional units being fabricated on the same chip. With the birth of the Very Large Scale Integration (VLSI) technology, a complete powerful micro-processor can be integrated onto a single integrated circuit or on a single silicon chip in the form of a silicon wafer and the technology is still continuing to provide a growing amount of transistor on the single wafer. The John Moore had predicted few decades ago that the chip density is doubling after every 18 to 24 months. The Intel Corporation has already followed this prediction quite religiously up to now as shown in Fig. 1.

Since the last few decades, the dramatic growth in the technology has led us with requirements of our appetite and need for computing powers to grow faster. Despite of all this fact that processors performance steadily increases, we often find it necessary to prop up these general-purpose processor based computing devices with specialized processing units, like dedicated multipliers to perform the convolution operation, generally in the form of specialized co-processors or ASICs. Consequently, the computers today exhibit an increasing disparity between the general-purpose processing core and its specialized ASICs based units. High performance systems are built from specialized ASICs. Even today's high-end workstations dedicate more active silicon to specialized processing units than to more general-purpose computing units. The general-purpose micro-processor will only be a fractional part of next generation multi-media based personal computers. As this trend continues, the well known term of "General Purpose Computers" will become a misnomer for modern computer technology. Relatively little of the computing machine power in tomorrow's computers can be efficiently deployed to solve any problem.

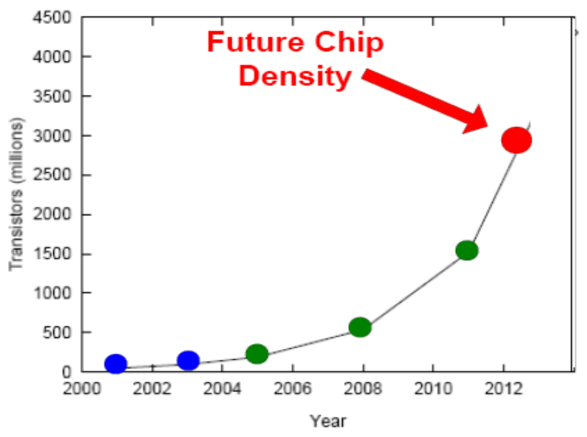

Fig. 1 Increasing Chip densities

There are two main features associated with general-purpose processor based computers which clearly distinguish them from those systems which are based on the ASICs. The way these aspects are handled plays a vital role in distinguishing the various general-purpose computing architectures.

A. Bus Inter-connection System:

In general-purpose computing machines, the internal data-paths between computational units or functional units cannot be hardwired. Different tasks will require different patterns of interconnection system between the functional units. Within a task individual computational routines and operations may also require different interconnectivity of these functional units. General-purpose computing machines must be able to provide the ability to direct the data flows between the functional units. In the memory mapped systems, the memory locations are used to perform this routing function by loading the routing path information from it. If in the case of coarse grain computing more functional units may operate together on a single task or even on a set of tasks, the spatial switching is required to move data-streams among these functional units and the reserved memory space. The flexibility and granularity of this interconnection is one of the major factors which determine the yielded capacity on an active application.

B. Software Level Instructions:

Since the general-purpose computing devices must provide different operations over time space, either within a computational task or between computational tasks, they require additional input streams and new instructions which guide the system how to behave at any point in time space during the computation. Each general-purpose processing element of the system needs one instruction to guide it about what operation is to perform and where is to find its data input-streams. The handling of this kind of additional input is one of the main distinguishing features between different kinds of the general-purpose computing architectures. When the functional diversity is huge and the throughput of the required task is much low then it would not be efficient to build up the entire application data-flow spatially in the computing device. Instead of it the active applications are realized by sharing and reusing the limited hardware resources in time space and hence only replicating the less expensive memory space for instruction and intermediate data storage system.

General-purpose computing devices are specifically intended to be used for those cases, where economically we cannot or we need not to dedicate the sufficient spatial resources to support an entire computational task within an application or where we do not exactly know much about the required task or tasks prior to the process of circuit fabrication to hardwire the functionality of the expected algorithm. The key ideas behind the general-purpose processor based computing are the defer binding of functionality until the computing device is finally employed and to exploit the temporal reuse of the limited functional capacity. Delayed binding and temporal reusability work closely together and occur at many scales to provide the characteristics which we now expect from general-purpose processor based computing devices. We are quite accustomed to exploiting these properties so that unique hardware is not required for every task or a set of tasks or even to an application.

This basic theme recurs at the many different levels in our conventional computing systems as described under.

A. Recurrence at market level – Rather than dedicating a computing machine design to a single task or to an application or even to a set of applications, the design effort may be utilized for many different applications.

B. Recurrence at system level – Rather than dedicating an expensive computing machine to a single application, the computing machine may execute different applications at different times by running different sets of instructions periodically or non-periodically in time space.

C. Recurrence at application level – Rather than spending much of the preciously available resources to build a separate computational unit for each different kind of the function required, the central resources may be employed to perform these functions in sequence with an additional input in the form of an instruction which is telling it how to behave at each point in time space.

D. Recurrence at algorithm level – Rather than fixing the algorithms which an application uses, an existing general-purpose computing machine can be reprogrammed for as many times as required with new techniques and algorithms as they are developed.

E. Recurrence at user level – Rather than fixing the functionality of the machine at the supplier side, the instruction stream specifies the functionality, allowing the end user to use the machine as best as it suits his requirements. Computing machines may be used for functions which the original designers even did not conceive. Furthermore the computing machine behavior may be enhanced or upgraded in the working field without incurring any hardware.

The emerging technological needs have indicated many problems with the existing computing platforms like general purpose processor based machines. The problems indicated so far are not concerned with the notion of general-purpose computing, but are most likely concerned with the implementation techniques being adopted for revolutionarized technology. Since the last few decades, both the industry and academia have focused largely on the task of building the highest performance processors, instead of trying to build the highest performance general-purpose computing machines or computing engine. As the silicon wafer technology continues to increase far beyond the space required to implement a competent micro-processor, it is time to reconsider or re-evaluate the general-purpose computing architectures in the light of shifting resource availability and cost. In particular, an interesting space has opened between the extreme edges of general-purpose processors based computing and specialized ASICs based computing. This design space is the birth of the new domain of computing now known as the reconfigurable computing [1]. Consider the Fig. 2 showing the characteristics of ASICs, GPPs and RC based system characteristics.

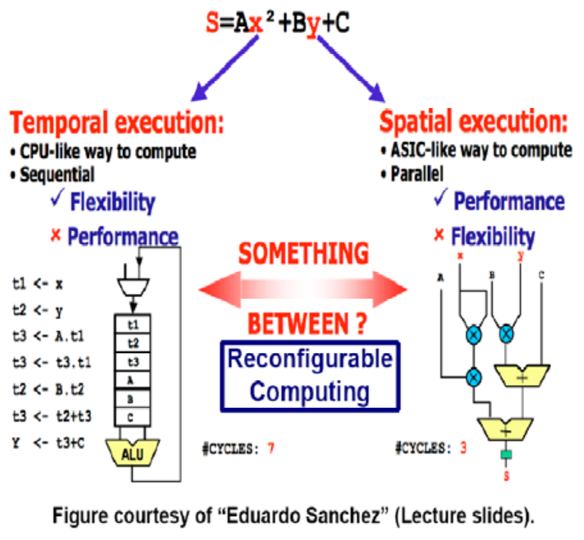

Fig. 2 Reconfigurable Computing Domain

II. Problem Description And Scope

The reconfigurable computing offers all the benefits of general-purpose processor based computing with much greater performance gain than that of provided by the traditional micro-processors. This space is most easily seen by looking at the binding time for device function. Application Specific Integrated Circuits (ASICs) bind the functionality of the required computation to active silicon at fabrication time making the silicon useful only for the designated function. On the other hand the micro-processors bind the functionality to active silicon only for the duration of a single cycle, a restrictive model which limits the amount the micro-processor can accomplish in a single machine cycle while requiring considerable on-chip resources to hold and distribute the instructions. Reconfigurable computing systems/machines allow the functionality to be bounded at a range of intervals within the final system depending on the needs of the running application program. Consider the Fig. 3 which shows the different software and hardware layers in reconfigurable computing systems and Fig. 4 which shows the internal design elements of a typical FPGA device.

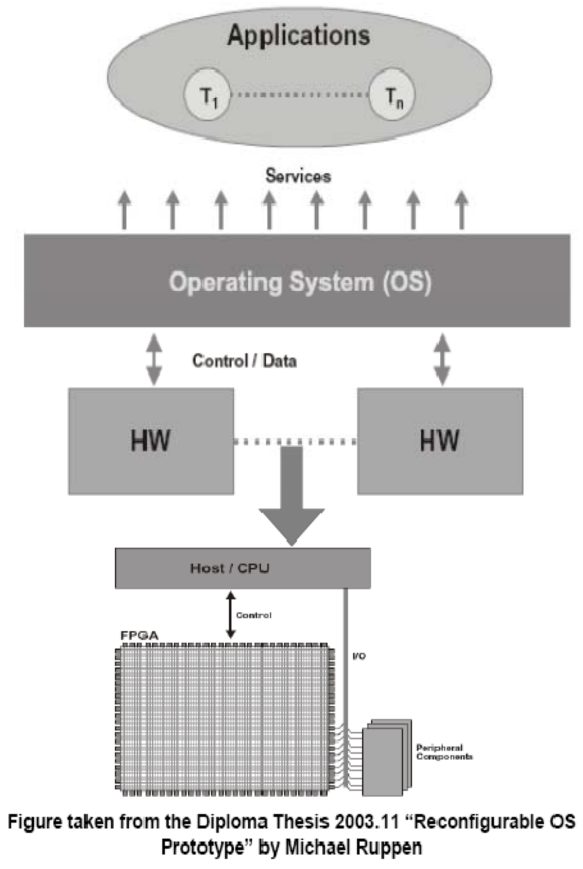

Fig. 3 Reconfigurable Computing System Layers

This flexibility found in the binding time allows the reconfigurable computing devices to make better use of the limited device resources including silicon area as well as the instruction distribution within the active instruction set window [2]. More precisely, the reconfigurable computing architectures offer the following main computing characteristics [3], [4].

1. More application-specific architectural adaptation than conventional micro-processors.
2. Greater computational density than the conventional micro-processors.
3. More efficient and broader reuse of silicon area than in the ASICs.
4. Better opportunities to ride hardware and algorithmic technology curves than ASICs.
5. Better match to current technology costs than ASICs or micro-processors.

In the domain of computing, the reconfigurable computing systems architecture/design is a very much new paradigm with a lot of benefits [5]. Hence due to this reason the most of researchers have been focusing on this area and research work has been involved actively since last two decades. In the beginning due to lack of the most advanced technology of dynamic and partial configurations, the investigators have been facing a lot of problems [6]. But quiet recently the Xilinx's Virtex series of FPGAs including the Virtex-4 platforms and Alrtera's Stratix-II series of FPGAs gives us a lot of computational scope in this area to get most of benefits of reconfigurable computing systems [5], [6].

III. Requirements Of Modern Reconfigurable Computing Systems: A General View

In the following paragraphs the major requirements and issues of the modern reconfigurable computing systems have been presented with short descriptions which have not been yet solved completely.

A. Requirements for New Computing Architectures

1. New reconfigurable computing hybrid architectural models at system level are required.
2. Pipeline based reconfigurable computing architectures are required.
3. Parallel processing reconfigurable computing architectures are required
4. Multithreaded reconfigurable computing architectures are required.
5. Emerging bit-slice technology based reconfigurable computing architectures are required.
6. Data and control dependencies analysis required for run-time reconfiguration.
7. Techniques for low power and high performance reconfigurable computing architectures are required.

B. Requirements for New Memory Architectures and Interface Designs

1. High performance embedded memory architectures for reconfigurable systems are required.
2. High bandwidths memory communication architectures
3. Customizable local memory management units and methodologies are required.
4. Configuration streams reusability transformations at memory level are required for reconfigurable computing systems.
5. Memory storage context transformations are required.
6. Memory communication interfaces for low power applications are required.
7. Memory communication interfaces for high performance applications are required.

C. Requirements for New Design Methodologies

1. Computing system-level modeling of dynamically reconfigurable computing architectures is required.
2. New efficiently reconfigurable computing design methodologies are still immature and inconsistent and hence demanding for more elaboration and research work.
3. New kind of advanced methods to manage the dynamically changing hardware/software resources in process of dynamic/run-time reconfiguration environment are required
4. Efficient methods for seamless interface design for bus macros are required for different operating environment
5. It is required to find metrics for generalized reconfigurable computing architecture characterizations.
6. Configuration overheads are becoming a new kind of system bottleneck and hence reducing of these configuration overheads in reconfiguration process are required.
7. At design level, the new kind of very powerful CAD tools are required to provide more accurate and specific automatic routing and partitioning

D. Requirements for New Software Tools

1. New software development tools and techniques for reconfigurable computing systems are required
2. Emerging automated synthesizer and constraint management software tools for a given application are required.
3. Logic design mapping onto the high density field programmable gate arrays and reconfigurable functional units (RFUs) for partial reconfiguration and dynamic/run-time reconfiguration.
4. Automated run-time reconfiguration temporal partitioning for reconfigurable embedded systems for real-time processing are required
5. New methodologies and architectures are required for advanced hardware/software co-design partitioning.

E. Requirements for Modern Techniques and Algorithms for CAD Tools

1. Fast and optimized routing and placement algorithms and techniques for new FPGAs are required.
2. Hardware and software level partitioning and co-design approaches are required.
3. Mapping algorithms are required for mapping the RTL design library functions on reconfigurable logic unit in run-time reconfigurable architectures.
4. New kind of algorithms and methods for the effective utilization of existing FPGA resources are required.
5. Effective methods are required for the reconfiguration of multiprocessor environments.
6. Design of an advanced operating system is required for a heterogeneous reconfigurable computing system-on-chip technology.

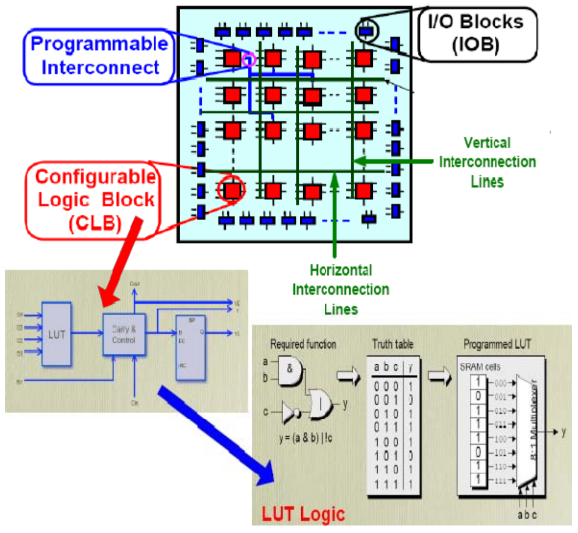

Fig. 4 Basic FPGA Internal Architecture

IV. Requirments Of Reconfigurable Systems At Application Layer Level

Reconfigurable computing systems are now fully capable of providing the execution parallelism at many different levels from intra-operation level parallelism to task level parallelism [7]. In this regard it has been observed that it would be ideal case if high level programming languages and automatic compilation tools could generally compile, profile, partition and parallelize existing code onto reconfigurable computing systems. It would be desirable to expression level computational algorithms and applications in a high level language and use the compiler to automatically detect computational-intensive segments of code and accordingly translate them into customized hardware level instructions and hence leaving inherently sequential or infrequently executed piece of code in software [8].

However, the reality is much different than the expectations arising from the inspiration of the reconfigurable computing technology. The research projects have been attempting to realize this gain but the software tools being available now a days and in the near future are requiring a very comprehensive and in depth understanding of reconfigurable computing systems strengths and weaknesses [8]. A considerable manual effort is required to partition the underlying code and familiarity with hardware level design to map the code segments to reconfigurable computing logic.

A. Programming Language Level Requirements

Programming languages for programmable general purpose processors are mostly algorithmic in their nature, building on the basic Turing machine formalism of instruction fetch, decode and execute cycle of a sequential stream of instructions and their relevant data that read and write memory. In contrast the software mapping algorithms/computation at application layer level to reconfigurable the hardware logic entails the low level fabrications of arbitrary logic circuits, exposing the maximal available amount of parallelism being consistent with hardware resource constraints.

For traditional hardware design, especially for the design of those circuits that must interface to external I/O devices and meet stringent timing constraints of the underlying hardware resource, it is desirable to use tools that mirror an abstraction of the underlying hardware resources. Hardware modules such as logical shift registers, logic comparators, multiplexers, binary adders and other functional units are convenient building blocks that the designer can combine, either graphically or through textual type of commands, so that to create a hierarchical, spatially-parallel hardware level circuit. Software tools are required to simulate the module interactions at the clock cycle level, as are software tools to control and analyze the automatic mapping, placement and logic routing of the modules onto the underlying FPGA device/fabric.

In contrast, a reconfigurable computing problem which is required to map it onto FPGA hardware is initially expressed as a set of sequential algorithmic instructions. There are a large number of different ways that the algorithm can be partitioned between hardware and software and even further, there are many functionally equivalent hardware logic circuits that can be generated from these algorithms [9]. Thus the search space in which to optimize partitioning options, the logic area, the frequency of computation and throughput is very large. Since the vast majority of FPGA based applications fall in the domain of traditional hardware logic design, so the greatest option and basic capability in programming languages and the underlying compilers is skewed toward hardware description languages, schematic layout editors, hardware circuit simulation tools and the synthesis process.

Most of these Computer Aided Design (CAD) tools are very much expensive and are requiring use of high performance workstation platforms. At the lowest level of design, the designer may direct the functionality and interconnection of hardware logic blocks onto the FPGA device and hence creating dense chip-specific designs that optimize the features of the particular chip being targeted. Intellectual Property (IP) blocks of computing systems are designed at this level and often are provided by the different chip vendors as the optimized building blocks for higher level designs. The next level of abstraction is to combine intellectual property blocks with application specific logic like in ASICs. This kind of methodology is called the structural design. At the next level of abstraction, Register-Transfer Level (RTL), registers, function modules, control structure, and timing are all specified by the designer. Finally the algorithmic and behavioral languages available are providing high level functional descriptions of computation. Consider Fig. 5 which shows the CAD tools development process for reconfigurable computing devices like FPGAs.

B. Compilation Tools Level Requirements

Application level program compilation for reconfigurable computing systems is considerably much more difficult than the compilation for conventional programmable general purpose processors. With the conventional programmable processors, the instruction set architectures (ISA) is already specified or given. The problem is to map an abstraction of the instruction set level as represented by a high level programming language onto a concrete instruction set level. With superscalar computing architectures particularly, much of the optimization occurs at run time of the application program in the micro-architecture that implements an underlying ISA and hence in this way simplifying the job of the compiler tool.

In contrast, the field programmable gate array (FPGA) has no instruction set architecture. The task of the algorithmic compiler of the application layer is to devise a micro-architecture being customized to the specific application, including data-paths, system memory hierarchy, Input/output and the sequencer in order to control the sequence of data-path operations, system memory access and Input/output operations. From the data types used and the operations within a program, the compiler is supposed to generate the functional units to execute the primitive operations. More often there is a huge sized module library from which it is required to select function units.

Given the array structures and other kind of required variables in a high level language program, the underlying compiler must be able to decide where each variable resides. The reconfigurable computing systems include a complex memory hierarchy ranging from several orders of magnitude in size and operational latency, making the trades-off space very large for designers. Once variables have been assigned memory locations, the compiler must generate hardware to read and write memory to/from on-chip microprocessor registers and logical and arithmetic function units.

High speed I/O such as a data stream from an analog-to-digital converter (A/D) imposes the hard real time constraints on the design. The combination of these system level constraints and options makes the compiling algorithmic languages of reconfigurable computing systems, a very difficult multi-objective combinatorial optimization problem. Equally difficult tasks are required for register transfer level synthesis of logic gates, mapping gates onto configurable logic blocks, placing virtual logic blocks to physical resource, and routing among the logic blocks. Consider the Fig. 5 which shows the major steps involved in the design and programming of a reconfigurable system.

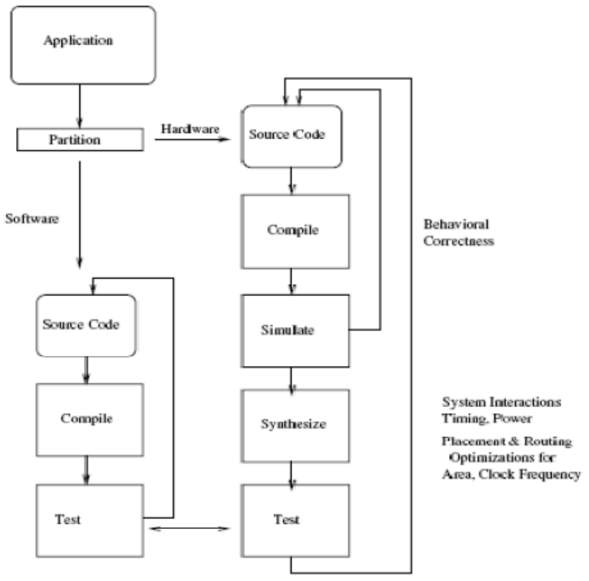

Figure taken from book "Reconfigurable Computing" by Maya Gokhale

Fig. 5 Reconfigurable system Design Flow

C. Application Debugging Level Requirements

The software tools and computing systems which have been available in past have demonstrated about what is possible for debugging the reconfigurable computing applications and what can be improved further in this regard, but despite of all these advancements, the most advanced tools available today for reconfigurable computing systems debugging tend to be trial and error, conventional logic analyzers, and tools like Identify, SignalTap and ChipScope. Although each of these software tools can be very useful in its own perspective, they illustrate the lack of real debugging support in most of the reconfigurable computing systems. This parameter has a significant effect on the productivity for developing validated and reliable complex reconfigurable computing applications.

One of the reasons for this is that many of such type of tools treat the development of reconfigurable computing applications for FPGAs and similar devices to be essentially the same way as the development of application specific integrated circuits (ASICs) hardware. However, the two development cycles are quite different. In the case of ASICs, enormous amounts of time spaces are required to correctly simulate the designs because it is extremely costly to fix the problems with the hardware modules once it has been fabricated. This kind of negative economic factor does not exist with reconfigurable computing systems since the hardware is often available for use and the cost of dynamically changing the design at run time is comparatively small. Of course, with design recompilations taking hours for very large designs, simulation is still important for reconfigurable computing application validation. On the other hand, the simulation of thousands and even millions of cycles can take as much time as a day or more for large complicated designs. So in this regard the recompilation cycle for hardware is not always unreasonable when justifying debugging using the actual hardware.

Another most difficult aspect is the economical factor for reconfigurable computing debugging systems. Unlike the developers of conventional software debuggers that can be retargeted to many systems somewhat easily due to the fact that the target microprocessor and the related system is fixed, the developers of high level as well as low level debuggers for reconfigurable computing applications are faced with a difficulty of supporting considerably smaller system volumes and the costs of developing the debugging support for each different reconfigurable computing system can be quite huge. In order to help this critical situation, some standardization of debugging support across reconfigurable computing systems might be able to encourage the third-party debugging tools [9], [10]. Furthermore in order to generate the information needed to support the hardware level debugging activity using commercial CAD tools would also take the willingness and concerted effort by RTL synthesis, FPGA and reconfigurable computing systems development companies. As reconfigurable computing becomes more and more common, hopefully these technical and non-technical challenges can be addressed so that the better debugging support can be achieved for reconfigurable computing applications.

V. Operating System Level Requirments Of Reconfigurable Systems

Operating Systems for reconfigurable computing systems will require a large number of different abstractions as in a traditional operating system for traditional computing machines. These abstractions may include the items such as a relocatable core library which will be responsible for the interface between each of the cores and application architecture. Such kinds of abstractions are also needed for detailing the procedures of how to communicate between the hardware level applications and the system level operating system. The hardware-to-hardware level abstractions defining how different application cores are required with each other and system input/output. A detailed description of each of them is given below.

A. Requirements for Relocatable Core Library

It has been observed that the pre-placed and pre-routed software cores will need to have a standard interface for their use in an underlying operating system. At the most basic level this requirement means a standard format for communication ports between different modules of the hardware are required. The most prominent example of such a type of static standard is the JBits core library standard. Furthermore a more complex requirement is a standard protocol for each of such available core to interchange with other cores of the system.

B. Requirements for Application Architecture Abstractions

Operating system is basically a general purpose system level tool. Definitely it would be unwise and unnecessary to commit its structure too heavily on application layer architecture. However application-layer architecture options impact on its performance. In the software area of course the operating system runs on the same architecture as the applications do and that architecture is a narrowly defined von Neumann processor architecture. On field programmable gate array, it is expected that the operating system and the applications might want to use the different internal structures. Hence the question for the operating system designer is just what a minimum shared structure is for these applications. For the focus on the application architectures, it is believed that it should be identifying items that are shared between applications either because they must communicate with each other using the shared resource at hardware and software level or because the shared resource has limited capacity such as internal RAM memory interfaces. The whole issue of the relationship between application architectures and the operating system performance (in terms of average response time and throughput) and design is still an open ended question since there have been almost no implementations of an operating system at all. Even it is becoming more and more worst as what has been implemented has taken the route of being very much closely aligned to a well defined specific architecture.

C. Requirements for Hardware/Software Communication Abstraction

It has been observed that there will always be a need for a standardized communication interface between hardware cores and software processes/threads unless and until all the applications being under execution under the supervision of an operating system and the operating system itself are implemented in hardware. A large number of comparative studies being done on the nature and needs of application layer programs and operating systems itself and it has been expected from experiences that it would not be very much useful or beneficial from a performance and design complexity point of view to have all the application functions being implemented in hardware. The rarely used computational functions and very complex control structures for hardware data-paths may not be the best candidates for implementation on hardware level.

The so far historical aspects of computing have shown that the only published abstractions have device driver type software with a message based socket interface to the software applications at intermediate level. It has been observed critically that there are significant performance overheads (in terms of computational latency and communicational latency) in this as compared to a socket-less interface which may be lead to a loss of performance gain for the hardware level module. There may exist a lot of other options like to have a method/function call interface but it is still an open question of whether this can be engineered with any better performance gain or not that the socket based one. As the performance of the software hardware interface is very much critical to the any operating system which involves software and hardware components, hence further investigations on this issue is also very important for the operating system research community.

D. Requirements for Hardware/Hardware Communication Abstraction

Typically the hardware cores are supposed to communicate with each other using some hardware level channels, for performance reasons. In addition the access of these cores to associated memory is supposed to be done at hardware level layer. There is no dought about that most of the application layer architectures suggested have by necessity a notion of inter-core communication and are very specific to the underlying architectures which themselves are very prescriptive in nature. The only RAW projects have been observed that have realized a compiler that generates interconnection structures at this level. This compilation of interfaces is viewed as a part of the application layer as distinct from the operating system. It has been observed that the operating system should also have its own internal data structures for inter-application layer communications. This observation allows us to have a large number of different compilers for the same operating system. It is a situation which is common place in the software layer arena. If a standardized core abstraction includes a fixed communication interface between different modules of the underlying layer then the inter core communication could be implemented by abutment. Another interesting possible option for the operating system is to have a local bus structure similar to common computing systems/platforms and some system-on-chip (SOC) proposals. However the locally associated bus structures are probably a very poor option for reconfigurable computing systems since they serialize all communication between cores.

The internal local bus structure also unnecessarily constrains the layout of cores on the field programmable gate array to bus interface locations. It might seem that the most promising area of investigation/research for interfaces is a parameterizable communication cores that allow the underlying operating system to be able to generate interfaces for heterogeneous application layer cores and thus taking advantage of the reconfigurable nature of the underlying platform and not holding the unduly constrain for the layout or serialize all communications. Interfaces to fast memories being attached directly to the field programmable gate array are a special case of the hardware to hardware interface and are likely to play a vital role in determining the performance gain of the many applications that need extensive off the chip storage. There might be the cases of having a fixed portion of this type of interface to ensure that this performance gain is achieved.

E. Requirements for Global Routing Abstraction

The logic routing between different hardware cores is known as the global routing. Whilst many field programmable gate array based platforms have hierarchical inter routing resources and hence it would seem to be unnecessarily very complex to have a routing abstraction that exposed these many levels [10]. Thus a single level of logic routing could be assumed by the operating system and it might be left to the software tools to optimize the routing logic using the available resources.

VI. CONCLUSION

Reconfigurable computing is becoming an important part of research since last few decades. Reconfigurable computing is intended to fill the gap between extremely inflexible but very high speed application specific integrated circuits based technology and flexible but very slow speed programmable processors based technology. It has been observed that currently available reconfigurable systems have many flaws which at hardware and software level. These flaws have stuck the increasing performance gain of reconfigurable computing systems. There are many emerging requirements at software and hardware level which are best candidates for further explorations of the existing technology. Each one of these requirements can be investigated at multi-dimensional levels and the currently available computing domains can be enhanced to a state-of-art technology which will promise us a very high performance gain for commercial as well as non-commercial computing applications.

REFERENCES

[1]. Compton K. and Hauck S. An introduction to reconfigurable computing. IEEE Computer, April 2000.
[2]. Grant B. Wigley and David A. Kearney "Research Issues in Operating Systems for Reconfigurable Computing", 2002.
[3]. K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," ACM Computing Surveys, vol. 34, no. 2, pp. 171–210, 2002.
[4]. Grant Wigley and David Kearney, "The Management of Applications for Reconfigurable Computing using an Operating System" 2000.
[5]. Francisco Barat, R. Lauwereins, G. Deconinck, "Reconfigurable Instruction Set Processors from a Hardware/Software Perspective"; IEEE Transactions on Software Engineering, Vol. 28, No.9, pp. 847–862, Sept., 2002
[6]. Kiran Bondalapati* and Viktor K. Prasanna, "Reconfigurable computing: Architectures, models and algorithms," CURRENT SCIENCE, VOL. 78, NO. 7, 10 APRIL 2000.
[7]. R. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In DATE '01: Proceedings of the conference on Design, automation and test in Europe, pages 642–649, Piscataway, NJ, USA, 2001. IEEE Press.
[8]. Khaled Benkrid, "High Performance Reconfigurable Computing: From Applications to Hardware", IAENG International Journal of Computer Science, 35:1, IJCS_35_1_04, 2008.
[9]. Azween Bin Abdullah, "Survivability Using Adaptive Reconfigurable Systems", IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.1, January 2009.
[10]. Prof. Sunil Kr. Singh, Dr. M. P. S. Bhatia, Dr. Rajni Jindal, "Architectural Modeling for Hardware and Software in Reconfigurable Embedded System", International Journal of Recent Trends in Engineering, Vol. 1, No. 1, May 2009.

AUTHORS PROFILES

M. Aqeel Iqbal

M. Aqeel Iqbal Is An Assistant Professor In The Department Of Software Engineering, Faculty Of Engineering And Information Technology, Foundation University, Institute Of Engineering And Management Sciences, Rawalpindi, Pakistan. As A Researcher He Has A Deep Affiliation With The College of E & ME, National University Of Sciences And Technology (NUST), Rawalpindi, Pakistan.

Dr. Asia Khannum

Dr. Asia Khannum Is An Assistant Professor In Collge Of E & ME, National University Of Sciences And Technology (NUST), Pakistan. She Did Her Phd In Computer Software Engineering From The Same Institution.

Saleem Iqbal and M. Asif

Mr. Saleem Iqbal and M. Asif are doing PhD and M.S respectivily in Computer Software Engineering from College of E & ME, National University of Sciences and Technology (NUST), Pakistan.