J. Basic. Appl. Sci. Res., 2(10)10185-10202, 2012
© 2012, TextRoad Publication
ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

Exploring High Performance Processing Through Reconfigurable Computing Systems

M. Aqeel Iqbal, Shoab Ahmed Khan

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

ABSTRACT

The reconfigurable computing is anticipated to block-up the breach among extremely inflexible but high performance Application Specific Integrated Circuits (ASICs) based technology and most flexible but slow speed General Purpose Processors (GPPs) based technology. The main inspiration behind the reconfigurable computing technology is to tightly couple the performance and flexibility aspects of both of these existing technologies into one platform. This kind of tight coupling of both technologies requires high advancements in the area of computing science and engineering. This research paper presents a comprehensive work about the existing reconfigurable computing systems. It highlights the hardware and software level issues being investigated into the domain of reconfigurable computing. It also highlights the emerging requirements being pointed out for further enhancement of the existing computing technology. A large number of emerging software and hardware level aspects have been investigated to cope up with the existing technology barrier. Special considerations are required in this domain for high performance computing systems to boost up the software and hardware development of such kind of advanced computing platforms to support the next generation high speed data-intensive computing demands.

KEYWORDS: Configuration Streams, FPGAs, High Performance Processing, Programmable Logic Devices, Reconfigurable Computing, Reconfigurable Interconnections

1. INTRODUCTION

The computing systems being in use now-a-days are the most evolved form of the previously used computing devices and systems. They are providing a most flexible form of computing platforms and are competent of executing a huge quantity of computing applications. The basic philosophy of GPPs based computing is that the software is developed by using high level programming languages. These programming languages at near to the ground level are utilizing the basic instruction set of the underneath machine architecture. Due to this fact the similar hardware resource can be utilized for unlimited number of computing applications. Regrettably, this kind of hardware generalization is gained at the price of despoiled speed [1]. In practice the running software application being available in form of a set of machine specific instructions is stored in main memory and hence is required to be fetch-decode-execute each time. Application-specific-integrated-circuits (ASICs) are supplementary source of computation which provides us an alternating solution which point-outs the despoiled performance of GPPs based computing system. But the ASICs based systems are exceptionally inflexible to accept the hardware changes. In ASICs once the design is fabricated, there is no way to do modifications or to promote for new requirements [1, 2].

In this regard by looking at these two barriers of computational styles in terms of system design and philosophy, the computing systems have been divided into three major categories including *Computing Systems Based on Dedicated Resources*, *Computing Systems Based on Specialized ISAs* and *Computing Systems Based on ASIPs*. The details of each one of these categories have been listed down here.

1.1 Computing Systems Based On Dedicated Resources

In order to determine the computational power and performance of computing system, it is obligatory to identify its dedicated and non-dedicated hardware resources. In this regard a key matter being implicated in the growth of computing architectures is their specialization. From the specialization of computing architectures we mean their amendment to a single application in execution. There are so many reasons which can guide the systems designers towards pursuing a dedicated computing architecture. Among these many reasons, the obligation for obtaining elevated performance gains in a particular application domain is one. Along with them the timing issues being implicated in hard as well as soft real-time applications pushed the designers towards developing the architectures sloping towards application-specific solutions which could more uncompromisingly meet the underpins requirements. Also along with these reasons there are other very significant parameters like development cost and power consumptions, which are key design factors being

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

occupied in the domain of embedded systems. All these described reasons and technological constraints enforce the need for a new design pattern which should take all of these features into account and should guide to the definition of new, specialized computing architectures. In this regard, on one extreme side stand the general purpose computing systems, in which generally core concern is to achieve the high performance in a very big range of applications. On the other extreme side stand the application-specific computing systems like ASICs, where embedded applications direct the in-use architectural blueprint [1,2]. Conservative benchmarks are not used to calculate the performances of dedicated architectures, but they are tested only on the applications for which the machine has been designed for.

1.2 Computing Systems Based On Specialized ISAs

The central elements of the Instruction Set of a general purpose processor recognized as the op-codes are the key bricks into which a high level language code is busted down for execution on a processor. The exclusive operation specified by every op-code of the machine instruction is executed on a devoted and extremely optimized computational unit like ALU. Hence the functions specified by each of the op-codes can be seen as the hardware implementation bricks of the software execution. If these tiny sized bricks are enlarged in size by performing more complex operations than those which are classically available in the Instruction Set of a Reduced Instruction Set Computer (RISC) based processors, the instructions in the code will correspond to more multi-faceted computational units. Hence possibly they might be characterized by much longer computational latencies and might be contributing the execution of much bigger computations when compared to RISC based computational units. Since hardware based execution is much quicker than that of software based execution, hence an application being busted down into larger bricks will be much faster to execute than one being busted down into smaller bricks. However, while fundamental and uncomplicated operations are common to many high level applications, the more complex operations turn out to be the less likely common to a large group of applications. Hence it is observed that while a complex computing component will speed up the execution when comprehensively used by a particular application under process, it is also factual that the area committed to such computational component would result in the exhausted resources, when running those applications that do not demonstrate such complex operations in their code [2, 3]. Moreover, the accessibility of such type of multi-faceted instructions further complicate the several code generation responsibilities such as code selection and register allotment between others applications.

1.3 Computing Systems Based On ASIPs

Application-Specific-Instruction-Set-Processors (ASIPs) endeavor to adjust the instruction set of the under laying processor to the application being in implementation. In general the ASIPs-based computing architectures are typically intended to provide single-chip integration and consist of an ASIP used as a central computing core, encircled by a great number of Application Specific Integrated Circuits (ASICs). Inside the computing configuration a part of the instruction set is executed on the central processing core which characteristically comprises of the standard arithmetic operations, memory read/write operations and control-flow instructions. The rest of the instruction set of the computing system is committed to specialized instructions which are tuned to the application. In pre-mature days these systems were anticipated to execute through the committed data-paths being implemented by the ASICs which were acting as hardware accelerators. But now no more integrated circuit based solutions are rising. Frequently used examples of such kind of specialized instructions may be a full phase of a well known Viterbi-decoder algorithm or of a Discrete Cosine Transform (DCT) being used in digital signal processing applications. Such a class of architectural associations is usually known as heterogeneous integrated circuits [1, 3]. The main shortcoming of the ASIPs based computing model is the fact that it involves a stage of design of a new compilation instrument (compiler) for each of such type of new processor. In this reference a cheaper solution being obtainable is to use an off-the-shelf DSP.

However the trade-offs concerned in the design on a committed machine is usually that of achieving a much elevated degree of circuit customization and computing performance at the cost of designing a quite new computing machine. Among the many issues implicated, the main dilemma of ASIPs is that the re-targetable compilation tools and techniques are not yet grown-up enough to permit the portability of a compiler to a complete set of this class of computing processors [2]. Hence each time a new type of compiler is desirable to be designed for each one of them. In order to prevail over these nominal drawbacks, a very fascinating computing approach was introduced in the last few decades. This computing approach recognized as Reconfigurable Computing (RC) involves the high level blending of an off-the-shelf general purpose processor with a reconfigurable logic, positioned at diverse levels of integration. Although Field Programmable Gate Arrays (FPGAs) technology has been in the marketplace for last few decades, but it has become probable only in recent times that their performance has become adequate for large scale computing applications [2, 3]. The now automated software tools are designing more established structures, so that the high level integration with conservative processors has become feasible. A large number of commercial as well as research projects are now investigating this computing approach called Reconfigurable Computing Technology (RCT).

Consider the figure-1 in which a two dimensional graph between the two vital aspects e.g. “Design Flexibility” and “Execution Performance” of the diverse computing platforms, now in hand, have been drawn. In this figure five types of computing systems have been outlined. These including General Purpose Processor (GPP), Digital Signal Processor (DSP), Reconfigurable Computing Systems (RCS), Application Specific Instruction-set Processor (ASIP) and Application Specific Integrated Circuit (ASIC). All of these five systems have been placed on the graph according to their design flexibility and computing speed. It can be seen that the GPPs are the most flexible but extremely slow speed

computing systems whereas the ASICs are extremely inflexible but high speed computing systems. DSPs are closely associated with GPP category as they have also programmable behavior. Similarly ASIPs are closely associated to that of ASICs as they are application specific designs. As far as the RCS are concerned they are truly a nice compromise of both technologies hence they stay in the center of the graph from flexibility as well as performance point of view.

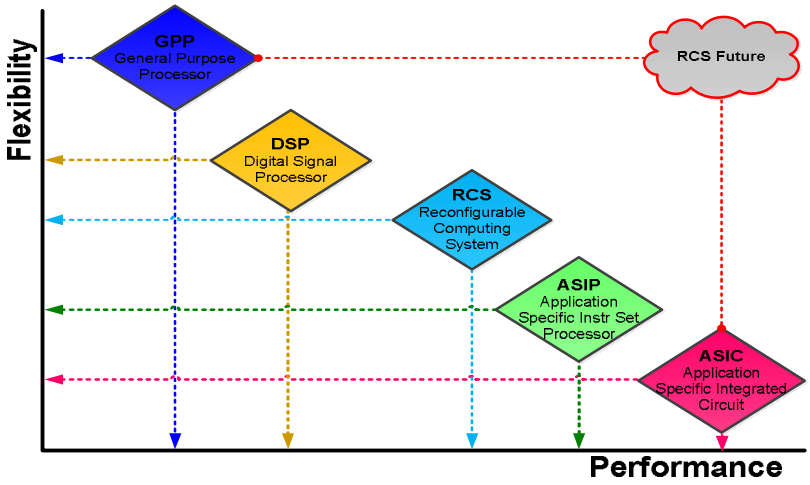

Figure-1: Flexibility vs. Performance Comparison of Computing Systems

2. Reconfigurable computing

The feature of an adaptive system is that it is being talented to grip the algorithmic changes to same extent as is done by ordinary GPPs. This drastically restricts ASICs since the designer must be talented to anticipate the hardware desirable for future designs. FPGAs offer the immense flexibility to be accustomed elegantly over the time period of the application execution. GPPs allow the designer to effortlessly modify the behavior of the system by changing the software, but algorithms can rapidly surpass the processing power obtainable by the chip. Adding an additional processor in the existing system characteristically requires wide-spread modifications to software in order to take benefit of the supplementary computational power of the computing platform. Since their birth, it has been observed that Field Programmable Gate Arrays (FPGAs) are less competent as compared to conventional *ASICs*, if they implement the identical design. As far as the power/energy expenditure is concerned the electrical power is required to program the FPGAs and also to maintain their configurations during the operation. Hence it also adds up a feature of disadvantage as compared to the ASICs technology. On the other hand; conversely once implemented, the ASICs are enormously inflexible and nearly cannot be personalized as easily as FPGAs. This feature theoretically imposes a considerable constraint on the longevity of the concerned system. Both FPGAs and ASICs have time-consuming design and testing cycles allied with them as compared to the design and testing time of software on GPPs. This cost is counterbalance by the performance gains realized in the programmed applications.

One application where these performance gains can effortlessly be demonstrated is the emergent field of the multi-media streaming applications. New algorithms become part of growing standards on a regular basis. In order to prevent an underlying device to become outdated, the software and/or hardware must go forward to support the new algorithms and their necessities. For GPPs, the software can be customized easily but the complicatedness arises when the processor speed becomes the expansion edge. The set of computations that can be speeded up by using reconfigurable hardware are recognized either during compile time or run-time and later on are mapped against the reconfigurable logic [4, 5]. In this situation the computations which have composite control of execution sequence and internal data structures are executed run on the underneath GPPs. This kind of rational partitioning of the computations between general purpose processor and the associated reconfigurable computing logic hardware resource is performed manually in old versions of the reconfigurable computing systems. But now it is habitually done by using contemporary automated or semi-automated software tools. These rationally partitioned computations have to be compiled into executable code on the beneath laying processor and in case of reconfigurable computing hardware it is compiled in shape of configuration bit streams on the reconfigurable computing hardware.

Consider the figure-2 in which the basic philosophy of the reconfigurable computing and reconfigurable system design has been presented. In this figure it has been described that there exist two broad categories of computing approaches which are strictly based on the concept of Locality of Reference. The locality of reference points out that the computational behavior can be either Temporal or Spatial. Temporal computations can be well executed using general purpose processor based computing platforms. Spatial computations can be well computed using some sort of application specific circuits like that of ASICs. Reconfigurable computing has been introduced to cope up some intermediate form of

computing so that to encompass both of conventional computing approaches. Reconfigurable computing is in fact a nice blend of both forms of existing technologies.

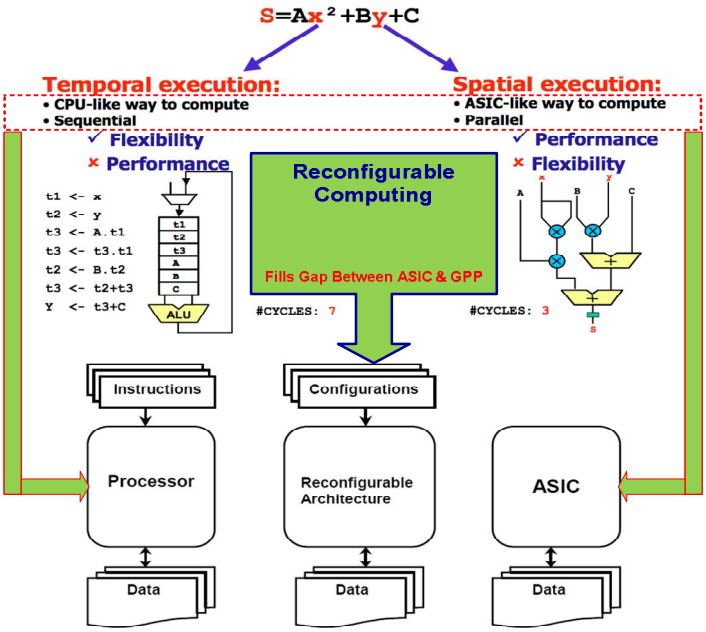

Figure-2: Concept of Reconfigurable Computing

The initiative of processing using reconfigurable natured computing systems remained an ultimate focal point of investigation since a last couple of decades, however the mainstream endeavors/projects have looked for the potential of linking multiple commercial available off the shelf field programmable gate arrays to an accessible micro-processor (central processing unit) by the way of a typical secondary-bus such as the well known *Peripheral Component Interconnect (PCI)* interface [5, 6]. If it is indeed object to make reconfigurable computing an intensive computing standard of the upcoming era, the essential components should be caught closer to each others. Quite recently only a small group of investigations have well thought-out of critically integrating a GPP and field programmable gate arrays into an introverted device. Where the both devices have been tailored to cooperate with each other narrowly and hence there continue a majority of very obvious momentous questions concerning that how such type of devices can be manufactured and later on according to requirements can be programmed using high level languages. Also the questions rise up that how it would be best fitted within an offered GPP based very flexible computing skeleton [6]. In recent times it has been experimented that the modern age reconfigurable computing is truthfully becoming a momentous component of highly intensive processing based research in the fields of computing system architectures. This is first and foremost due to the reason that modern reconfigurable computing concepts combine the payback of both flexible software and non-flexible application specific integrated circuits based implementations. Equivalent to an application specific integrated circuit, the reconfigurable computing systems are artistic with a technique to plot logic circuits into hardware resources and hence achieving far superior computing performance/speed than that of software as a result of getting rid from the conventional fetch-decode-execute cycle of conservative GPPs.

Consider the table-1 in which the synergism between General Purpose Processor (GPP) based computing platforms and Reconfigurable Computing Systems based computing platforms have been compared in different aspects. The different aspects which have been highlighted include Adopted Processing Style, Parallelism Level Exploited, Supporting Clock Speed, Application Level Partitioning and Commercial Level Availability. This kind of synergism between the two approaches may help us in choose the right platform for application execution and computing machine construction.

Table-1: GPP vs. RCS Synergism

	General Purpose Processor (GPP)	Reconfigurable Computing System (RCS)
Adopted Processing Style	– Software Based Control Flow – Temporal (Reuse of Fixed Hardware)	– Hardware Based Control Flow – Spatial (Unfolding Parallel Operations with Changeable Hardware)
Parallelism Level Exploited	Coarse-grain Level	Fine-grain Level
Supporting Clock Speed	– Very Fast – Saturating Rate	– Relatively Slow – Increasing Speed
Application Level Partitioning	Comparatively Easy	Relatively Difficult
Commercial Level Availability	Commercial-Off-The-Shelf (COTS), Multi-purpose	Commercial-Off-The-Shelf (COTS), Multi-purpose

Reconfigurable computing is still at the profundity of micro-code stage and consequently requiring programmers to have a substantial knowledge about very low level circuit design in order to construct an application. A large number of C-based languages exist that map to HDLs; conversely those languages still require the programmer to think on the scale of near to the ground level [6]. As seen in software, principal output gains will not take place unless the level of reasoning about problems changes. A program implem 10188 ·cuit that takes benefit of the parallelism offered by hardware is trickier to create than a series of instructi a processor. In order to make gains in output, the abstraction level at which engineers make reconfigurable computing programs needs to be elevated. Along the manner, some optimizations may be given up but that will permit for the development of bigger and more complex systems that could not be imaginably created otherwise. Several attempts are being made at doing this through developing advance level languages for programming hardware in a C-like style. Many of these approaches present a small step ahead in raising the altitude of abstraction but still it requires a momentous knowledge of reconfigurable computing programming to construct systems.

The reconfigurable computing hardware needs to be configured by means of configuration bit streams before the true execution can be performed. There are a number of application areas where reconfigurable computing domain has been confirmed to achieve a momentous performance gain. The most frequent of these application areas include long multiplications like required in the convolution of signals, cryptography, genetic algorithms, image processing, genomic database exploration and digital signal processing. The nature and variety of the reconfigurable computing systems results in a wide collection of implementation issues with respect to running applications. Computational power for promising high speed systems is principally dependant on the underlying hardware resources and secondarily on the system software being installed. In fact the performance is also dependant on the type and the level of the integration of both the hardware and software resources [6]. The integration of system software and core hardware exists in the form of the layered technology. In many aspects, the reconfigurable computing has been changing the responsibilities of each of these layers. In the case of the *General Purpose Processor* based systems; the domain anticipation is a single programmable processor running with a multi-threaded/multi-tasking operating system that executes numerous software tasks. In a similar way, in the case of the *Reconfigurable Computing (RC),* the anticipation is generally a single high speed *Field Programmable Gate Array* acting as a multi-threaded/multi-tasking operating system that loads numerous hardware tasks.

Consider the figure-3 in which a typical reconfigurable computing system design has been presented. The figure shows that a reconfigurable computing system is comprised of a set of configuration controlling instructions, certain blocks of configuration streams, reconfigurable logic hardware resources and programmable I/O interface. Configuration controlling instructions can be of two types including one those which contain configuration bit streams and second those which contain the address of configuration streams. First type of instructions are basically using immediate type of addressing mode while the second type of instructions are using memory direct type of addressing mode or register direct addressing mode. First type of instructions is definitely fast while second type of instructions is comparatively slow. Along with these aspects both types of instructions have many other pros and cons as far as their implementation on the hardware design is concerned. Configuration memory is usually a kind of simple buffer memory or sometimes a fast cache. The reconfigurable computing hardware resource is a typical FPGA or any other reconfigurable device. Similarly the I/O interface is a standard interface being already deployed in conventional programmable processors.

In the case of software model, the operating system is accountable for scheduling different tasks, swapping them in and out of the processor according to their load, the garbage collection of core hardware and software resources, synchronization of collaborating tasks etc [7]. These similar expectations should be truthful for the FPGA configurations. Software systems swap tasks at a rate that makes logic to avoid the thrashing of the system. System software is generally expected to be platform dependant and hence it only runs on the platform for which it has been compiled. To run on various platforms, developers must make diverse builds. Generic builds such as Java based systems exist but do not have the performance analogous to that of a platform-specific embedded application. These identical properties are true for hardware. Java supports the *Just in Time (JIT)* compiler which is accountable for converting standard Java based byte code to platform specific code prior to executing. The same can be done in hardware by allowing the system to optimize the design over time while it is in exercise. When the optimization is complete, the specific best fitted configuration can be swapped in to improve the system performance. Caching is normally done by GPPs to improve performance of tasks that are regularly executed. The same can be done with hardware configurations provided that the chip support is available. The hardware configurations can be saved in the locally coupled caches for each of the execution unit or the same copy of the configuration can be shared between the different units [7]. These configurations are generally saved in the associated memories in the form of contexts. The system deals with these contexts in an analogous way as an operating system does with its contexts of the processes. Each context is associated with a Configuration Control Block (CCB).

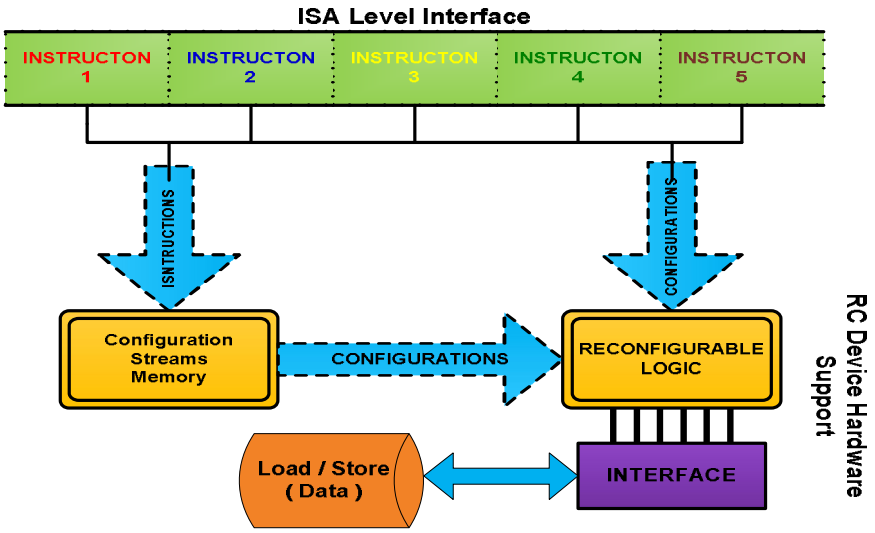

Figure-3: Design of a Typical Reconfigurable Computing System

Consider the figure-4 in which the design of a typical High Performance Reconfigurable Computing Systems (HPRCS) has been shown. The diagram shows that in order to build a high performance reconfigurable computing system, we need to integrate multiples computing nodes so that to achieve high performance parallel processing where each computing node itself is composed of a standard General Purpose Processor (GPP) and a Reconfigurable Processing Unit (RPU). Both of these units have been connected together using some sort of high performance interconnection structures. Such kind of multiple computing nodes have been then connected together using some sort of network topologies.

3. Characteristics of reconfigurable computing systems

Hardware platforms for reconfigurable computing systems have been accessible in different forms. There have been diverse criterions being considered for the classifications of the existing reconfigurable computing systems. But in general there are four distinguishing properties which can be used as a means for the classification of on hand reconfigurable systems.

3.1 Reconfiguration Style

The basic description of reconfigurable computing systems allows for platforms which merge conventional pre-determined components of ASICs with one or more reconfigurable computing fabrics. The equilibrium between these two main types of digital circuits typifies an underlying physical hardware platform. This property of the reconfigurable computing systems is referred as the reconfiguration style of a system. The device reconfiguration style is a macroscopic feature of the computing system. Reconfiguration style can be viewed as an un-interrupted scale of recycling process. On the one excessive side of the scale are the systems which are fully reconfigurable or algorithm adjustable. Such systems are fully comprised of reconfigurable fabrics at coarse-grain level or at fine-grain level [8]. On the other excessive side are non-reconfigurable architectures, such as traditional von Neumann architecture based computers. Somewhere in between these two extreme boundaries are the semi-fixed and variable systems. These types of systems merge the fixed components, such as sequential processors or even fine-grained components capable of reconfigurable routing process.

3.2 Device Computational Granularity

The notion of computational granularity deals with the size of the reconfigurable logic elements that can explicitly be reconfigured by the programmer either statically or dynamically. On the one extreme side the very little processing units can be reconfigured and are recognized as *Fine-grained Reconfigurable Logic Units*. In these fine-grained reconfigurable logic units the reconfiguration logic is potentially very small even it can be at the intensity of a single logic gate being independently reconfigurable. Moving towards the coarser end of the scale; such category of systems exist which propose control over bigger operational components like adders, multipliers, shifters and logical units etc. Reconfigurable computing fabrics can have harmonized granularity, but hybrid fabrics are also widespread. Granularity has a great number of the significant implications. Primarily, it has an apparent bounding consequence on the admittance of the system as an application development platform. Fine-grained reconfigurable fabrics tolerate for the implementation of comprehensive, problem specific designs which implies a superior potential for proficient hardware exploitation in the shape of virtual hardware resources. However on the other side of image, the coarse-grained components can be optimized for performance on scale deeper than that of logic gates. Coarse grain/specific components can also be

prepared which demonstrate shorter transmission delays than rationally counterpart implementations with reconfigurable logic gates. Due to this potentially muscular feature of computing, the up-and-coming computing inclination is towards the heterogeneous reconfigurable computing fabrics, which recommend a jumble of reconfigurable logic gates and frequently used arithmetic and routing components.

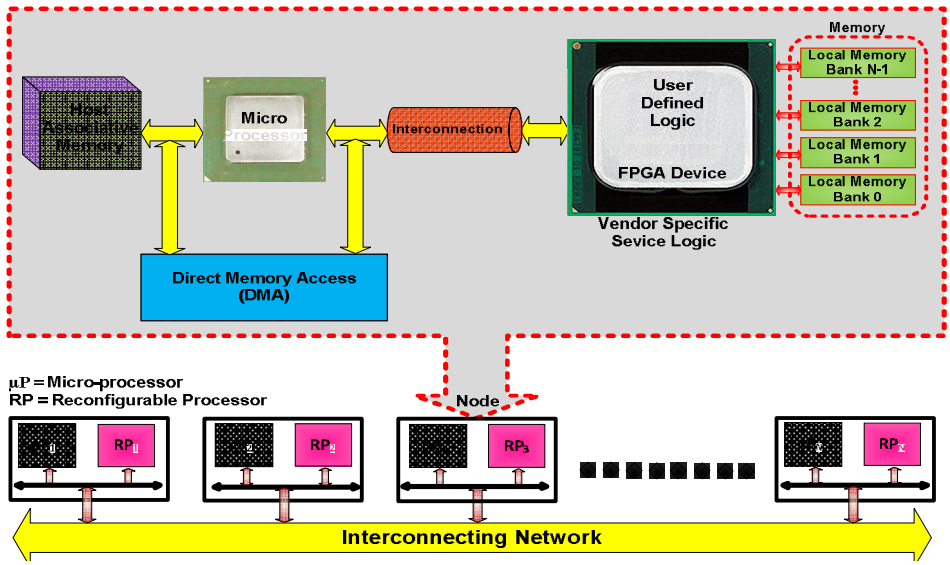

Figure-4: Design of a typical HPRCS

3.3 System Reconfiguration Overheads

The final feature of the system is also associated to the reconfigurable processing fabric on atomic level. The reconfiguration overhead is the quantity of delay that is obligatory to load a fresh circuit description into the fabric in the shape of the configuration streams. In the definition of the reconfiguration overhead the compilation or preprocessing delay/time is excluded. Hence we are only concerned in the time overhead it takes to physically change the hardware to a new configuration. In relation to reconfiguration overhead, we formulate an extensive division between two classes of systems. On the one hand are those systems which are statically reconfigurable. A statically reconfigurable computing fabric is a fabric that cannot amend its configuration at runtime. On the other hand, the runtime reconfigurable fabrics are also accessible which can be loaded with new configurations throughout the execution of an application. Run-time reconfigurability is a very valuable and influential property of any reconfigurable system. It can facilitate to alleviate the problem of limited hardware resources. Of course with the gained power for the high performance computing such a category of reconfigurable systems initiate the increased management complexity.

4. Coupling approaches for reconfigurable logic units

There have been a large number of diverse computing architectures premeditated for use in reconfigurable computing field. One of the most significant variations between these architectures is the level of coupling of Reconfigurable Logic Unit (RLU) with the host micro-processor [4, 9]. Programmable computing logic intends to be ineffectual at implementing some classes of operations including long running loops and execution level branching control. In order to execute a computing application most competently in a high performance reconfigurable computing platform, the parts of the vigorous program that cannot be smoothly mapped to the reconfigurable computing logic are run on a host GPP core. In the intervening time the parts with an elevated deliberation of computation that can attain from realization in hardware resource are fully or partially mapped to the reconfigurable computing unit [10]. For those computing platforms that make use of a general purpose processor in concurrence with reconfigurable computing unit, there are plentiful numbers of ways in which these two computational cores can be tied up together.

Given the distinction between fully and partially reconfigurable computing architectures, we can further make the categorization among available systems by basing on the nature and form of the communication logic interfaces obtainable among the reconfigurable computing logic and the extremely inflexible fixed logic. In a broader vision the overall reconfigurable computing systems can be grouped between two types of computing systems. On one extreme side are those systems which are known as *Loosely Coupled Reconfigurable Computing Systems* in which the reconfigurable fabric executes algorithmic computations with a certain degree of autonomy like an autonomous parallel task. One the other extreme side are those systems which are known as *Tightly Coupled Reconfigurable Computing Systems in which* the reconfigurable computing fabric works as fundamental component of the CPU like a special expansion to the ALU on which custom instructions can be implemented. More accurately reconfigurable computing systems can be locally grouped in four diverse categories. Different coupling approaches for the such systems include; the reconfigurable fabric

as a *Functional Unit*, the reconfigurable fabric as a *Co-processing Unit*, the reconfigurable fabric as an *Attached Processing Unit* and the reconfigurable fabric as a *Standalone Processing Unit [4]*.

Consider the figure-5 in which the different coupling approaches have been highlighted which are in practice for reconfigurable circuitry interfacing with standard processing units. The different coupling approaches being demonstrated include Tight Coupling of reconfigurable computing logic as integrated unit within a standard processor, Loosely Coupling of reconfigurable computing logic as being interfaced to an expansion bus of system like PCI bus, COM port, USB port or Parallel port. Other two schemes being under use include interfacing of reconfigurable computing logic with processor as a supplementary Co-processor or interfacing it as an Attached Processing Unit like a Direct Memory Access (DMA) controller. As the distance between the reconfigurable computing circuit unit and the standard processor unit increases, the configuration overhead increases and the system characteristic of being autonomous increases or improves.

4.1 Tightly coupled functional unit type approach

In first coupling approach the reconfigurable logic hardware can be utilized exclusively to make available the reconfigurable functional units inside a host processor. This permits for a conventional programming atmosphere with the accumulation of conventional instructions that may possibly amend over the time. Here the reconfigurable computing logic units accomplish their tasks as functional computing units on the central GPP core data-path where the internal register memories are used to snatch the input and output data operands. This form of coupling technique is extensively used in *Reconfigurable-Instruction-Set-Processors (RISPs)* also known as reconfigurable processors.

4.2 Dedicated Co-Processor Type Approach

In second coupling approach a reconfigurable component may be used as a coprocessor. Basically a co-processor is superior to a functional computing unit and is artistic to carry out computations without the steady command of the available host micro-processor. As an alternative the micro-processor initializes the reconfigurable computing unit and either transmits the necessary data to the core computing logic or alternatively it provides information about to where this data may be bring into memory [11].

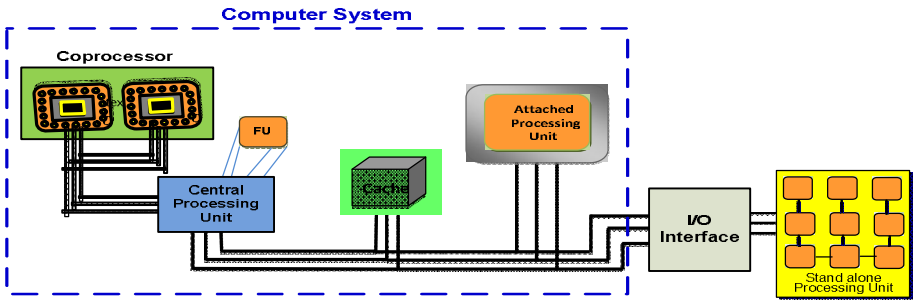

Figure-5: Different Coupling Approaches for Reconfigurable Logic

4.3 Attached Processing Unit Type Approach

In third coupling approach the attached reconfigurable processing unit behaves as if it is an additional micro-processor in a multiprocessor computing platform. The host processor's data cache is not manifest to the directly attached reconfigurable computing resource. Consequently, there is an elevated stoppage in the communication among the central host micro-processor and the reconfigurable computing logic in that when communicating configuration data/information the input data and output results. Nonetheless, this style of reconfigurable hardware does permit for an immense agreement of computation sovereignty by shifting huge pieces of a computation in excess of the reconfigurable computing hardware [11].

4.4 Loosely Coupled Standalone Unit Type Approach

In last coupling approach the most freely attached category of reconfigurable computing hardware is that of an expansionary detached data processing unit. This class of reconfigurable computing hardware communicates infrequently with available host processing unit. This replica is equivalent to that of networked computer workstations, where data processing may take place to a great extent restricted of a gigantic harmony of communication. Every one of these types has differing payback and limitations/drawbacks. The more closely coupled the incorporation of the reconfigurable computing hardware, the more frequently it can be utilized inside an application or set of the applications due to a slighter communication time overhead. Nevertheless, the hardware is incompetent of working for substantial portions of time without intervention from a host processor, and the amount of reconfigurable computing logic available is usually quite inadequate. The more liberally coupled forms stand for greater parallelism in program implementation, but experience from elevated communications time overhead. In applications that demand a massive deal of communication, this can shrink or get rid of any speeding up payback gained through this type of the reconfigurable computing hardware.

5. Programmable logic devices

There is a variety of Programmable Logic Devices (PLDs) being used in the industry for the development of the digital circuits. Following is a categorization of these devices according to their domestic design structures.

5.1 Application Specific Integrated Circuits (ASICs)

Application Specific Integrated Circuit (ASIC) is an integrated circuit tailored for a fastidious use, rather than anticipated for general-purpose use. For example, a chip premeditated to run in a digital voice recorder is an application specific integrated circuit. Application specific standard products are intermediary between application specific integrated circuits and industry standard integrated circuits. As feature sizes have shrunk and drawing tools enhanced over the years, the utmost complexity probable in an application specific integrated circuit has full-fledged from 5,000 logic gates to over 100 million logic gates. Contemporary application specific integrated circuits often include complete 32-bits-to-64-bits processors, memory blocks together with Read Only Memory (ROM), Random Access Memory (RAM), Electrically Erasable Programmable Read Only Memory (EEPROM), Flash Memory and other outsized building blocks. Such an application specific integrated circuit is often termed a System-on-Chip (SOC). Designers of ASICs use a HDL, such as Verilog or VHDL, to depict the functionality of application specific integrated circuits. The application specific integrated circuits belong to the family of micro-electronic circuits. The application specific integrated circuits vendors have also been classically supported by the contemporary software tools like Computer Aided Design (CAD) tools that have automated the processes of synthesis and circuit layout. The vendors have been now proficient of to layout the chip, generate the masks, and fabricate the application specific integrated circuits.

5.2 Masked Gate Arrays (MGAs)

The more specifically a masked gate array is an ASIC with a distinctive domestic architecture that is consisting of rows and columns of standard transistor structures being incorporated at micro level. Each one of these fundamental cells comprises of the analogous form of equivalent number of transistors which are not connected by default. None of these transistors being incorporated at micro level on the logic gate arrays are originally connected at all. The requisite connections are being determined entirely by the circuit design that will be afterward implemented. Once the requisite blueprint is ready, the automated layout software figures out about which transistors to hook up. Throughout the preliminary footstep the low level logic functions are allied together and later on in the second step when the low altitude functions have been routed out suitably, these would in turn be coupled together. The automated software tool would carry on this process until the whole chip design is absolutely finalized.

5.3 Simple Programmable Logic Devices (SPLDs)

There exists an incomprehensible array of vocabulary among manufacturers for this collection of devices commonly known as SPLDs. The Programmable Array Logic (PAL) devices descend into this class. More multifarious devices are occasionally referred to as Erasable Programmable Logic Devices (EPLDs) even though a number of manufacturers use this as a standard term to cover all programmable logic devices. Programmable array logic devices outline the doorway level into the collection, characteristically replacing five to six Transistor-Transistor-Logic (TTL) chips in a 20-to-22 pin package. They grant a good preface to the features, architecture and applications of Simple Programmable Logic Devices (SPLDs). The programmable array logic was made-up to grant an alternative to Small-Scale-Integration chips in applications where personalized combinational or sequential logic is obligatory. In its original shape it engaged Bipolar Junction Transistor (BJT) technology and fusible associations for programming. Among the primary devices to be accessible were a combinational logic device made up of an array of AND/OR logic gates structure and others which supplemented flip-flops to facilitate sequential logic circuits to be implemented. The contemporary counterparts are the devices manufactured using Complementary Metal Oxide Semiconductor (CMOS) technology and combining characteristics from both these devices. The programming fuses have been replaced by electrically programmable cells, in this manner permitting the device to be again and again reconfigurable. Hence in summary Simple Programmable Logic Devices (SPLDs) are the devices like PROMs, PLAs, PALs and GALs. The following table abruptly describes their characteristics with respect to their programming capabilities.

Consider the table-2 in which the characteristics of different SPLDs have been analyzed. In this figure the only two design parameters have been chosen including the structure of AND-array and OR-array being implemented in SPLDs. Structure of any AND-array/OR-array can be either “Fixed Structure” or it can be a “Programmed Structure”. Based on the nature of this structure, the PROMs, PLAs, PALs and GALs have been compared. If a structure is fixed, then it would be quite fast but inflexible to adopt modifications. Conversely if a structure is programmable then it would be quite flexible to adapt the incoming design variations but it would be slow speed.

Tabl-2: Characteristics of SPLDs Members

Programmable Device	AND-Array Style	OR-Array Style
Programmable Read Only Memory (PROM)	Fixed	Programmable
Programmable Logics Array (PLA)	Programmable	Programmable
Programmable Array Logic (PAL)	Programmable	Fixed
Generic Array Logic (GAL)	Programmable	Fixed

- Programmable Read Only Memories (PROMs)

There are several types of Programmable Read Only Memories (PROMs) being available for utilization. Some of the PROMs available now can be programmed once only and consequently they are known as One-time-Programmable (OTP). But there are some supplementary types of PROMs which have dissimilar characteristics such as Flash EPROM, Ultra-violet Erasable Programmable Read Only Memories (UV-EPROMs) and Electrically Erasable Programmable Read Only Memories (EEPROMs) that can be erased and programmed multiple times.

Consider the figure-6 in which the design of a typical PROM has been shown. Generally PROMs have a build in Fixed AND-array and a programmable OR-array. The programmable OR-arrays are used to realize the actual Boolean function being implemented inside the PROM. The set of fixed AND-arrays are used to generate all possible Min-terms which are later on used by the OR-array to generate the SOP and finally required Boolean expression.

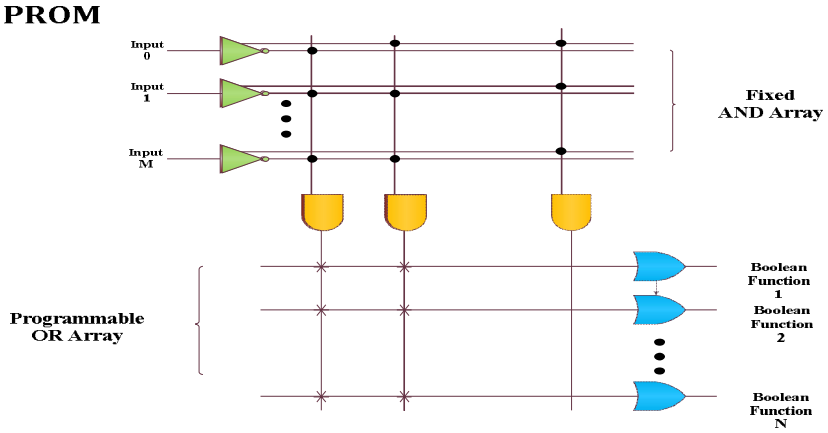

Figure-6: Internal Design of a Typical PROM

- Programmable Logic Arrays (PLAs)

A Programmable Logic Array (PLA) is a type of programmable logic device used to realize combinational logic circuits of simple to high level complexity. The programmable logic array has a set of programmable AND logic gate surfaces, which connect to a set of programmable OR logic gate surfaces, which can then be provisionally complemented to turn out an output. This arrangement permits for a great number of logic functions to be synthesized in the Sum-Of-Products (SOPs) canonical forms. A combinational logic circuit may occasionally have don't-care conditions. When implemented with a read only memory, a don't care state becomes an address input that will in no way occur. The expressions at the don't care address need not be programmed and may be left in their novel position. The consequence is not all the bit patterns accessible in read only memory are used which may be well thought-out a misuse of obtainable module. One application of a programmable logic array is to realize the control over a data-path. It defines a variety of states in an instruction set, and produces the next state. Programmable logic arrays should keep up a correspondence to a state diagram for the system. Other frequently used programmable logic devices are programmable logic array, complex programmable logic device and field programmable gate array. Make a note that the use of the expression Programmable does not point toward that all programmable logic arrays are field-programmable. In truth a lot of are mask programmed through fabrication in the similar manner as a mask read only memory. This is principally factual of programmable logic arrays that are implanted in more multifaceted and plentiful integrated circuits such as micro-processors. Programmable logic arrays that can be programmed after fabrication is known as field programmable gate arrays. The programmable logic arrays were introduced as a substitute solution to cover up the speed and input restrictions of the conservative programmable read only memories.

Consider the figure-7 in which the design of a typical PLA has been shown. Generally PLAs have a build in programmable AND-array and a programmable OR-array. The programmable OR-arrays are used to realize the actual Boolean function being implemented inside the PLA. The set of programmable AND-arrays are used to generate the required set of Min-terms which are later on used by the OR-array to generate the SOP and finally required Boolean expression.

- Programmable Array Logics (PALs)

The phrase Programmable Array Logic (PAL) is used to depict a family of programmable logic device semiconductors used to realize digital logic functions in digital circuits. Programmable array logic devices comprise of a miniature programmable read-only memory and supplementary output logic circuitry used to realize fastidious most wanted Boolean logic functions with a small number of elements. Using dedicated equipment the programmable array logic devices are made to be field programmable devices. Every programmable array logic device is One-Time-Programmable (OTP). It refers to the fact it is not possible to revise and use it again after its preliminary programming. Early programmable array logic is commonly packaged as 20-pin Dual-Inline-Package (DIP) components made-up in

silicon using Bipolar Junction Transistor (BJT) technology using OTP Titanium-Tungsten-Programming (TTP) fuses. Afterwards released devices have been made up of Complementary Metal Oxide Semi-conductor (CMOS) technology. The novel 20pins-to-24pins programmable array logics have been described as medium-scale integration devices. Most significant substance that has been added includes the clocked elements like usual flip-flops. Programmable Array Logics are now talented of implementing an outsized number of logic functions together with clocked sequential logics required for state machine implementations. This has been one of the most significant developments that permitted the programmable array logics to substitute much of the customary logic in numerous designs. Programmable array logics are now tremendously fast solutions.

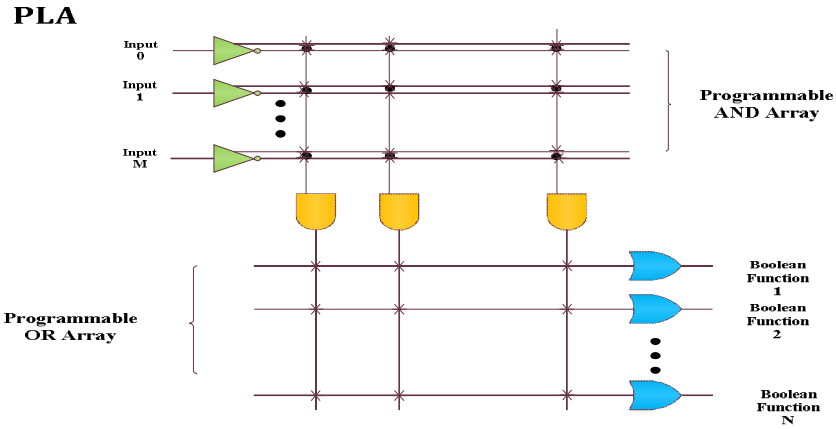

Figure-7: Internal Design of a Typical PLA

Consider the figure-8 in which the design of a typical PAL has been shown. Generally PALs have a build in programmable AND-array and a fixed OR-array. The fixed OR-arrays are used to realize the actual Boolean function being implemented inside the PAL. The set of programmable AND-arrays are used to generate required set of Min-terms which are later on used by the OR-array to generate the SOP and finally required Boolean expression.

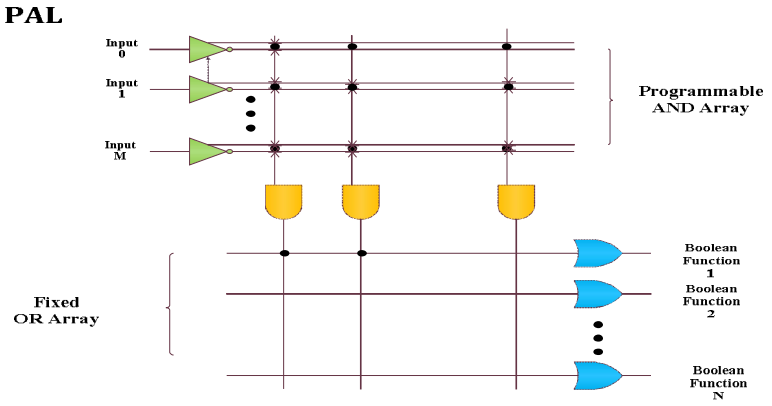

Figure-8: Internal Design of a Typical PAL

- Generic Array Logics (GALs)

Generic Array Logics (GALs) were introduced by Lattice Semiconductor Corporation. Generic array logic has presented us most well known Complementary Metal Oxide Semiconductor based EPROM and EEPROM type variations by just using the programmable array logic perception. Generic array logic structural design has a set of reprogrammable AND-array, a set of fixed OR-array and a set of reprogrammable output logic. Generic array logic is analogous to programmable array logic with Output Logic Macro Cells (OLMCs). These macro cell based structures provide us better flexibility in terms of digital logic circuit mapping. OLMCs can be configured either for a combinational logic output or can be configured for a kind of registered output. Generic array logic can be erased quite easily and as well as can be reprogrammed. It is a usual industry practice to replace an entire set of diverse programmable array logics. The control fuses for the generic array logic macro-cells permit every macro-cell to be configured in one of three fundamental configurations. These configurations match up to a variety of types of Input / Output configurations found in the programmable array logic devices that the generic array logic is planned to substitute. Generic Array Logics can emulate a number of PALs consequently now there is no need for PAL any longer. PALs are using the fuse technology while the GALs are using the electrically erasable cells based technology. PALs are one-time-programmable

logic devices but GALs are erasable and reprogrammable devices. PALs have predetermined IO functions whereas the GALs have a selectable IO structure.

Consider the figure-9 in which the design of a typical GAL has been shown. Generally GALs have a build in fixed AND-array and a fixed OR-array. The fixed OR-arrays are used to realize the actual Boolean function being implemented inside the PROM. The set of fixed AND-arrays are used to generate all possible Min-terms which are later on used by the OR-array to generate the SOP and finally required Boolean expression.

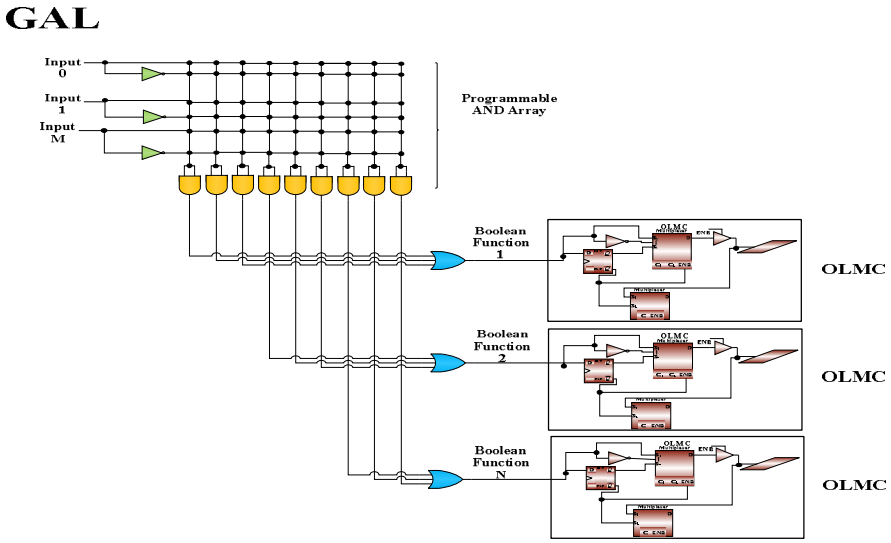

Figure-9: Internal Design of a Typical GAL

Consider the table-3 in which PALs and GALs have been compared in terms of their implementation technology, ability of reconfiguration and structure of Input /Output interface. Technology parameter includes the implementation technologies which include either fuse/anti-fuse technology or electrical erasable technology. Reconfigurability includes either one-time programmable or erasable / programmable.

Tabl-3: PAL vs. GAL Characteristics

Parameters	PAL	GAL
Device Technology	Fuse/Anti-fuse	Electrically Erasable Cells
Reconfigurability	One-time Programmable (OTP)	Erasable and Reprogrammable
Input / Output	Fixed Function	Selectable

5.4 Complex Programmable Logic Devices (CPLDs)

Complex Programmable Logic Devices (CPLDs) are fundamentally designed to become visible just like a bulky set of PALs being incorporated in a sole silicon chip and have been allied together with each other through a set of programmable cross bar switches. As far as the domestic design viewpoint and their development procedure is apprehensive, the CPLDs are using the identical automated development tools and are based on the equivalent technologies, but they have a much more density and hence can grip much more multifarious logic. As far as the in-house design of CPLDs is concerned, each manufacturer has a diverse design disparity but in general they are all comparable in a sense that they consist of Function Blocks (FBs), Input / Output Blocks (IOBs) and a Programmable Interconnection Matrixes (PIMs). These devices are programmed by means of programmable elements where the programmable elements can be of different types depending upon the technology of the manufacturer including UV-EPROM cells, EEPROM cells or Flash EPROM cells.

Consider the figure-10 in which the internal design of a typical CPLD has been shown. The basic components being integrated inside the CPLD include a set of PAL like blocks which have been connected together by using some kind of internal bus interconnecting network. Also it has a set of Programmable I/O blocks. PAL like blocks are used to realize the actual combinational logic circuits of low to moderate level density. The interconnection network is used to make more and more complex circuits by interconnecting the multiple PALs like blocks. Through the I/O interface the logic circuit that has been embedded inside the CPLD is connected to external system.

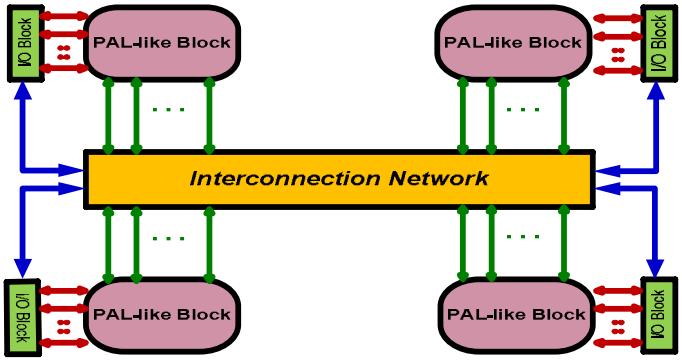

Figure-10: Internal Design of a Typical CPLD

5.5 Field Programmable Gate Arrays (FPGAs)

The basic FPGA structural design consists of a set of Configurable Logic Blocks (CLBs), a set of Configurable I/O blocks (IOBs) and a network of Programmable Interconnection Switches [12], [13]. There is also a routed set of clock circuitry for driving the clock signals to each logic block of the device and supplementary logic resources such as ALUs, Block Memories and Decoders. The fundamental programmable elements being used for a FPGA include the Static RAM (SRAM) cells, Fuses and Anti-fuses [12, 13].

Consider the figure-11 in which the internal design of a typical field programmable gate array has been shown. The fundamental building block of any FPGA is a Configurable Logic Block in which the digital circuit design is implemented. In this block the logic circuit is realized in term of SOP form Boolean expressions and multiplexing logic among these SOP terms. Generally this logic design concept is known as look-up-tables. Along with CLBs, there is a set of programmable routing paths which run horizontally as well as vertically throughout the devices. The main function of these routing paths is to provide connectivity between different CLBs. There is a set of programmable I/O interfaces which are used to connect the device internal logic with its external environment.

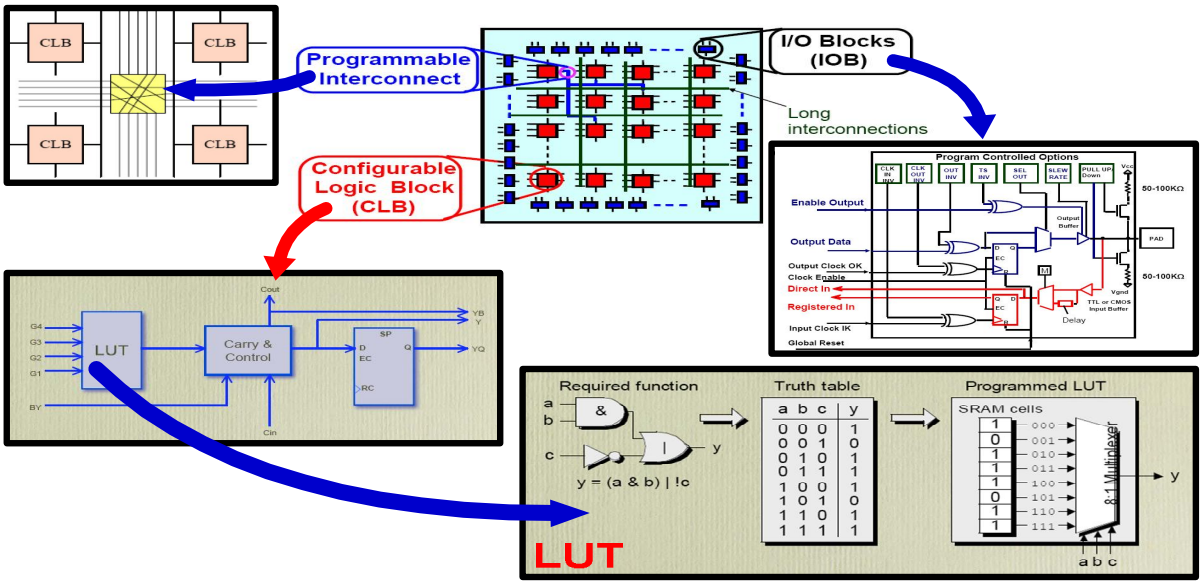

Figure-11: Internal Design of a Typical FPGA

In the following paragraphs the details of inside components of the FPGAs have been discussed. For reference, the architectures like those provided by Xilinx have been discussed.

– **Configurable Logic Blocks(CLBs)**

Configurable Logic Blocks (CLBs) surround the fundamental logic used for programming the circuits in FPGAs. From the granularity point of view there are two foremost types of designs including the Coarse-grain and Fine-grain designs. In coarse-grain (large-grain) architectures, these CLBs will enclose adequate programmable logic to construct a little state machine [14]. In fine-grain (small-grain) architectures, more similar to a true gate array ASICs, the configurable logic blocks will enclose only very small and indispensable logic [14]. CLBs contain SRAM cells, fuses or

anti fuses for creating arbitrary combinatorial logic functions. CLBs also contain D-type flip-flops for clocked storage elements and a group of multiplexers in order to route the programmed logic inside the block and outside the block. These multiplexers also permit polarity assortment, reset operation and clear input selection.

– **Configurable Input /Output Blocks (IOBs)**

Input / output blocks (IOBs) are used to transport signals onto the FPGA chip and propel them back to exterior devices being attached with FPGA [13]. I/O blocks are consisting of an input buffer gate and an output buffer gate with tri-state logic and open collector output controls logic. There is a set of pull up resistors on the outputs and even sometimes a set of pull down resistors being provided along the I/O pins of the device.

– **Programmable Interconnection Network (PIN)**

The Programmable Interconnection Network (PIN) of an FPGA is very dissimilar as compared to that of a CPLD internal interconnection network. It is more strictly associated and analogous to that of a gate array ASICs internal interconnections. There is a group of horizontal as well as vertical extended lines which are used to hook up critical CLBs that are physically far from each other on the similar chip without introducing much more delay. These horizontal and vertical lines are used as the internal buses stirring the signals inside the chip. There is also a group of short lines stretched out throughout the device which are used to connect individual CLBs which are positioned physically nearer to each other. Also there is a set of programmable switch matrix, like that of originated in a distinctive CPLDs, to connect these long and short lines mutually in a particular way defined by the routing program. These programmable switch matrix stretch inside the chip and permit the connection of CLBs to interconnection lines and interconnection lines to each other and to the programmable switch matrix [14]. Tri-state buffers have been provided which are used to hook up many CLBs to a long line and hence creating a bus. Exceptional type of long lines extends out horizontally as well as vertically, called the global clock lines. These lines have been particularly designed for low impedance and thus rapid transmission times. These lines are connected to the clock buffers and to each clocked element in each CLB of the device. In this way the clocks are dispersed all over the FPGA device [15].

– **Internal Clock Circuitry**

In order to supply the high rate clock signals inside the entire chip, a set of special I/O blocks with extraordinary high drive clock buffers, known as clock drivers, have been disseminated throughout the chip. These buffers have been linked to clock input pads and drive the clock signals onto the global clock lines of the device. These clock lines have been premeditated for providing the low skew times and fast transmission times.

6. FPGA CONFIGURATION METHODOLOGIES

Field programmable gate arrays have been in commercial use since last few decades. One of the most significant features of FPGAs is their aptitude to act like a vacant hardware resource for the ending buyer. Providing supplementary speed-up than uncontaminated software based designs geared up on GPPs and added suppleness than ASICs based fixed-functioned solutions characteristically depends upon the FPGAs being a reconfigurable device. In reality each configurable building block found in an FPGA requires a 1-bit of storage in order to preserve a user defined configuration. For a typical Look-Up Table styled/based FPGA, these reconfigurable locations usually comprise of the stuffing of the Configurable Logic Block (CLB) and the connectivity of the in-house programmable routing resources. These configuration bits usually known as configuration bit streams have diverse meanings for LUTs and course-plotting fabrics.

The configuration being used for a representative FPGA can be thought of as a simple binary file whose contents in actuality map the bit-by-bit to the reconfiguration streams/bits in the FPGA? The size of these bit streams are reliant on the hardware being deliberated as well as on the FPGA being used. The contemporary FPGAs necessitate the superior and more multifarious bit streams. There have been many configuration methods used for storing a single bit of binary information within a distinctive FPGA [15]. Following is a short portrayal of the most frequently used configuration methods for FPGAs.

6.1 SRAM Based Configurations

In current era the most extensively adopted technique being used for storing the configuration data in widely available FPGAs is based on a volatile natured static random access memory. This configuration scheme has turned out to be the most admired and extensively used scheme because it provides fastest and endless number of times reconfigurations. Key drawbacks associated to SRAM based configurations appear in the shape of additional power utilization and data volatility. Along with these parameters the magnitude of the SRAM based cells is unadventurously larger and more multifaceted design as compared to the other technologies being in use for the configuration of an archetypal FPGA [15]. Due to this massive power utilization the device dissipates considerably larger standing power because of leakage current of the transistors.

6.2 Flash Memory Based Configurations

Although less fashionable than SRAM but quite a lot of families of FPGA devices are using flash memory to grasp the configurations of the device. Flash memory is a non-volatile device and can merely be programmed for a restricted

amount of times because of its limited data erasing aptitude. From the non-volatility of flash memory we strictly meant that the configuration bit streams (data) that have been written to it leftovers at hand yet when electric power of the FPGA apparatus is switched off. Hence due to this attribute, flash memory based FPGAs in dissimilarity with static random access memory based FPGAs; remain fully configured by means of user-defined hardware logic throughout the power-up cycles. Hence they do not necessitate any additional storage resource to program at boot-up time. Flash-based FPGAs can be standing by quite instantaneously after the booting of the device is completed. Along with these factors there are some other aspects like a flash memory cell is being prepared by using lesser number of transistors as contrasted to an SRAM based cell. Consequently this type of system design can consume much lesser standing power utilization as there are only smaller amount of transistors to contribute to leakage current of the device.

6.3 Anti-Fuse Based Configurations

A third approach being used to accomplish the programmability of an FPGA device is based on anti-fuse technology. In actuality an anti-fuse is a metal-based link which behaves in a contradictory manner to that of a fuse. Hence in this way it creates a wire or a kind of short circuit among the two end-points of an anti-fuse. Anti-fuse technology has a number of advantages and a small number of disadvantages. It is notable that a typical anti-fuse has no tendency to be made reprogrammed. Once a connection has been fused, it has undergone a physical alteration or transformation that cannot be upturned. FPGAs based on anti-fuse technology are usually well thought-out to be One-Time-Programmable (OTP).

7. Fpga based system development

Since their commercial prologue in the last decade, the *FPGAs* have revolutionized the way of designing the digital systems. Since then, these commodity parts have turn out to be priceless system components due to their potential to realize a random number of diverse logic functions proficiently. They are competent of being effortlessly reconfigured on fly as the system hardware requirements transform according to the demands of the executing application. The progressive improvements in chip integration technology have augmented the logic aptitude of these devices from the equivalent of a handful of simple TTL logic gates to a massive aptitude of a mid-sized *ASICs* chip nowadays. A number of commercial vendors have announced campaigns to initiate devices with the capacities of up to few millions of logic gates on a sole chip in just few years in the near future. With this obtainable plenty of computational logic and routing resources, several new-fangled application areas in the field of computing have become practicable for FPGAs technology. Although the hardware system aptitude has full-grown to an elevated extent in recent-times accessible systems but still to a great degree the core software is desirable to involuntarily drawing and rout the user defined digital designs. The utilization of FPGAs in an accessible prototyping system is usually a sort of tradeoffs when compared to an ASIC realization of the identical design. A fully personalized realization of any digital logic circuit will always give superior performance than an FPGA but probably at a higher cost due to condensed fabrication volume as contrasted to the commodity FPGAs. Furthermore it is promising to make tradeoffs in the realization of prototyped digital designs within the FPGA device. Usually the contributed investigation in this vicinity has been paying attention on achieving the optimized circuit arrangement solution in terms of minimized resources or optimized performance gain, at the cost of a considerably amplified assessment time overhead.

The FPGA based system development begins from the fundamental *Hardware Description Language (HDL)* like Verilog-HDL and VHDL coding and stops at the production of configuration streams for an archetypal FPGA device. In such type of HDL coding the hardware particulars are presented in the textual shape. Later on the procedure of development passes in the course of many steps and eventually reaches up to a spot where the HDL code is altered in a shape that is adequate for FPGA device and consequently emulation is performed.

7.1 Logic Synthesis Process

The initial footstep involved in the growth of digital system by means of FPGAs, is the logic synthesis process. This process translates the register transfer level portrayal of an arbitrary hardware blueprint into an optimal digital logic gate level manifestation. Since from the last few decades the digital logic circuit synthesis has appeared as an indispensable fraction of the computer aided design tools the proposal has been anticipated already in subject of various texts. Logic synthesis of digital circuits essentially involves the synthesis of two main types of digital circuits known as the *Combinational Logic Circuits* (CLC) and *Sequential Logic Circuits (SLC)*. The computed outputs of an arbitrary combinational logic digital circuit are only dependant on current data inputs given to the circuit. Digital logic Adders, Digital logic Multiplexers, Digital logic Decoders, Digital logic Encoders and other such category of elementary types of Boolean logic equations are well known examples of combinational logic digital circuits. On the other hand a sequential logic circuit needs a kind of feedback mechanism since it has to preserve circuit state subsequent to the data inputs have been disconnected and hence the circuit state is rationalized depending upon a digital logic clock signal. Digital logic design constructs like *Level-Triggered-Latches* and *Edge-Triggered-Flip-flops* are the fundamental building blocks of sequential logic circuits. From these fundamental building entities the more composite sequential logic circuits like memory registers, digital logic counting circuits (counters) and Mealy-State-Machines as well as Moor-State-Machine all can be constructed. Logic synthesis process which is used for the design of digital logic combinational circuits has the following phases of operations.

- The process will be used to produce a group of digital logic circuit expressions from the provided *Register-Transfer-Level (RTL)* design specification.
- Then the process is required to translate these logic circuit expressions into a standard design construct of two levels *Sum-Of-Products (SOP)* or two levels *Products-Of-Sum (POS)* logic or even into some other multilevel Boolean logic.
- Then the process enters into a phase where it is required to optimize this two levels or multi-levels logic circuit expression according to a bit cost model of digital design.

For digital design the synthesis process for sequential logic circuits is apprehensive with finite state machines. In this regard the under laying Boolean logic synthesis techniques and corresponding algorithms are required to do the following set of high level operations.

- Firstly they are required to provide the specifications of any arbitrary state machines into register transfer level circuit description.
- The second step being required is to optimize the total number of working states of the underlying synchronous sequential logic design.
- The third step being required in this regard is to provide high level design encoding of system working states in a condensed binary coded representation form.
- The last step being involved is to provide a high level complete optimization of the entire digital system design.

The elementary confrontation and ultimate intention of digital logic synthesis process is to provide the optimization for high level design trade-offs in circuit area, computing speed and power consumption in order to accumulate the blueprint constraints of the principal digital logic system. This optimization can take place at the technology free altitude of digital system design or can be plagued to a fastidious comprehension technology. Technology autonomous optimizations demonstrate a vicinity-cost mock up based on the number of literals in the assembly of Boolean logic circuit expressions and a stoppage cost model based on the length of the greatest longing sequence in the group of Boolean logic expressions. They endeavor to diminish the surplus logic being caught up and the familiar sub-expressions.

7.2 Technology Mapping Of Logic

After the synthesis of the logic performed mechanically or manually using CAD tools into a group of prefabricated design primitives which are comprehensible by the FPGA tools, the first major fragment is of translating a logic design articulated into these prefabricated templates. So far accessible circuit level technology mapping techniques diverge depending on the specific system architecture however there are a large number of Boolean functions being occupied which are usually used in any approach used for FPGAs.

– Conventional logic optimization techniques for FPGA based designs have been worn in order to get rid of the unnecessary logic from the nominated design along with simplifying the defined circuit logic. Yet despite the fact that a certain level of optimization has already been made available throughout the synthesis process of identified design, more regularly the supplementary optimization levels may be probable to attain. Particularly, if numerous design units or modules that have been separately synthesized for diverse optimization levels and purposes are being incorporated collectively.

– The logic mapping software tools have been using an algorithm to envelop the Netlist of being accessible library primitives with a set of logic resources that are performing an equal function. So far a large number of successful algorithms have been urbanized for performing the Netlist covering operation using four-input LUT logic elements. These algorithms primarily consider the characteristic that how to bundle logic functions in LUTs based on constraints such as chip area, digital circuit velocity and algorithmic velocity.

7.3 Device Logic Placement

Throughout the process of the logic placement the existing CAD tools find out a proficient assignment for every mapped logic block between all of the probable physical locations for that block inside the reconfigurable logic device. The logic placement may be resultant or reliant on several constraints such as minimizing wire length, ensuring routability inside the array cells and as well as maximizing the circuit performance.

7.4 Programmable Logic Routing

The last step made in the low-level FPGA design realization is the routing of the signals among inputs and outputs of the located logic blocks by suitably configuring the pass transistors, buffers, and multiplexers. Routing algorithms in reality can be motivated by many different restraint factors. A few of the more victorious algorithms be inclined to consider both the timing constraints and routing overcrowding constraints. A lot of these routing algorithms internally have a little form of network routing method which outfits Dijkstra's shortest path algorithm to probably route among logic resources where the costs of every route are associated to some balance of timing and overcrowding constraints.

7.5 Configuration Stream Generation

The last step being implicated in the low-level design flow for FPGA based systems is the production of the concrete data streams used for programming the FPGA apparatus. These configuration bit streams utter how a variety of

physical resources of the FPGA apparatus are configured. The configuration bit streams itself may also include some type of data checksums or some sort of CRC values for inspecting data uprightness. In case of Xilinx FPGA devices and as well as in supplementary families of configurable devices, the instructions for controlling the configuration method and parameters are intermingled with the configuration bit streams. Excellent examples of this form of configuration tools are including the JBits tools and Application Programming Interface (API) formed by Xilinx Corporation for their XC4000 and Virtex series of FPGA families. The JBits API (Application-Programming-Interface) typically allows the Java tool programmers/developers to generate design level circuit configuration data streams by straightforwardly identifying about how the individual actually easy to get to resources of the device can be configured. For example in this regard the data values for a specific LUT could be set or reset, a specific input to a digital logic multiplexer could be selected, memory buffers and passing bipolar junction transistors could also be autonomously turned on or turned off etc. For well known partially reconfigurable Virtex field programmable gate arrays provided by Xilinx Corporation, the vendor has also provided an accompanying application programming interface on apex of JBits which is known as the JRoute. It permits to the Java developers to systematically produce and take away the associations among logic inputs and outputs of these hardware resources. In conclusion the JRoute based circuit design routing algorithms significantly optimized the configuration of digital circuit designs when working at the low-level of notion provided by JBits.

8. Conclusion

Reconfigurable computing technology has been a part of active research from previous few decades. The basic concept of this computing domain is to tightly integrate the both high flexibility aspect of general programmable processors and elevated processing performance feature of the application specific integrated circuits onto a sole silicon technology chip. By placing the computationally intensive portions of algorithms onto the reconfigurable logic, the dramatic performance gains have been demonstrated so far. In order to give a big leap to under laying reconfigurable computing technology, the high performance reconfigurable computing systems have been introduced in the current era. These systems are based on the tight coupling of the standard micro-processor core and the reconfigurable logic on the same chip, hence bringing up the both technologies quite closer to each other. Now the programmable logic devices like FPGAs are closely integrated in the form of a computing core with standard processor core. This highly level integration of two different computing cores has introduced a great potential for computing powers. The potential is in use but still a lot of achievable computing potential is under wastage due to the lack of the required automatically controlling software tools. Software tools ranging from application layer to system layer should be further investigated for enhancing the computational power of the existing computing systems and hence supporting the future computing applications.

REFERENCES

[1]. Kuon, J. Rose, “Measuring the gap between FPGAs and ASICs”, Proceedings of the ACM/SIGDA 14th International Symposium on Field-Programmable Gate Arrays (FPGA’ 06), pp. 21–30, Monterey, Calif, USA, 2006.

[2]. R. Hartenstein, “A decade of reconfigurable computing: a visionary retrospective”. Proceedings of Design, Automation and Test in Europe Conference (DATEC’ 01), pp. 642-649, Munich, Germany, 2001.

[3]. Benkrid, Khaled, “High performance reconfigurable computing: from applications to hardware”, IAENG International Journal of Computer Science (IJCS), Vol. 35, Issue. 01, 2008.

[4]. M. Aqeel Iqbal, Shoab A. Khan, Uzma Saeed Awan, “Reconfigurable computing systems related hardware and software perspectives”, International Journal of Intelligent Information Technology Application (IJIITA), Vol. 02, No. 05, pp. 209-217, 2009.

[5]. K. Compton, S. Hauck, “Reconfigurable computing: a survey of systems and software”, ACM Computing Surveys, Vol. 34, No. 02, pp. 171–210, 2002.

[6]. M. Aqeel Iqbal, Asia Khannum, Saleem Iqbal, M. Asif, “Emerging requirements of reconfigurable computing systems for performance enhancement”, International Journal on Computer Science and Engineering (IJCSE), Vol. 02, No.05, pp. 1572-1579, 2010.

[7]. M. Aqeel Iqbal, Uzma Saeed Awan, Shoab A. Khan, “Reconfigurable computing technology used for modern scientific applications”, Proceedings of 2nd IEEE International Conference on Education Technology and Computer (ICETC’ 10), pp. 36-41, Shanghai, China, 2010.

[8]. R. Hartenstein, “Coarse grain reconfigurable architecture”, Proceedings of Conference on Asia Pacific Design Automation (APDA’ 01), pp. 564-570, 2001.

[9]. A. DeHon, J. Adams, M. DeLorimier, et al., “Design patterns for reconfigurable computing”, Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM’ 04), pp. 13-23, Napa Valley, Calif, USA, 2004.

[10]. M. Aqeel Iqbal, Farooque Azam, Uzma Saeed Awan, Saif ullah Hammad, "Performance enhancement techniques for modern reconfigurable computing systems", International Journal of Computer Applications (IJCA), Vol. 27, No. 09, pp. 33-38, 2011

[11]. T. J. Todman, G. A. Constantinides, S. J. E.Wilton, O.Mencer, W. Luk, P. Y. K. Cheung, "Reconfigurable computing: architectures and design methods", Proceedings of IEE Computers and Digital Techniques, Vol. 152, No. 02, pp. 193-207, 2005.

[12]. A. G. Ye, J. Rose, "Using multi-bit logic blocks and automated packing to improve field-programmable gate array density for implementing data-path circuits" Proceedings of IEEE International Conference on Field-Programmable Technology (ICFPT' 04), pp. 129-136, Brisbane, Australia, 2004.

[13]. S. J. E. Wilton, "Implementing logic in FPGA memory arrays: heterogeneous memory architectures", Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM' 02), pp. 142-147, Napa Valley, Calif, USA, 2002.

[14]. C. Plessl, M. Platzner, Zippy, "A coarse-grained reconfigurable array with support for hardware virtualization", Proceedings of the 16th International Conference on Application-Specific Systems, Architecture and Processors (ICASAP' 05), pp. 213-218, 2005.

[15]. Y. Qu, K. Tiensyrj a, K. Masselos, "System-level modeling of dynamically reconfigurable co- processors", Proceedings of the 14th International Conference on FPL, Vol. 3203 of Lecture Notes in Computer Science, pp. 881-885, Tampere, Finland, 2004.