METHODOLOGY FOR DESIGNING HIGH SPEED RECONFIGURABLE CUSTOM COMPUTER

A Thesis

Submitted in partial fulfillment of the
requirement for the award of degree

Of

Master of Engineering
In
Software Engineering
Batch 2002-2004

By

Mr. V.S.Shankar Sriram
(8023117)

Computer Science & Engineering Department
Thapar Institute Of Engineering & Technology
(Deemed University), Patiala-147004 (India)

May 2004

ABSTRACT

Reconfigurable Custom Computers are being proposed as faster means of devices for Application specific computing. A key difference between the classical computer and a Reconfigurable custom computer is that a classical computer solves a problem with the available hardware and/ or logic whereas a custom computer solves the problem with specialized hardware and/ or logic. As an emerging field Reconfigurable Custom Computing is emerging as an important new organizational structure for implementing computations on the best-fit hardware and/ or best-fit logic. It combines the post-fabrication programmability of processors with the spatial computational style most commonly employed in hardware and logic designs. The result changes traditional "hardware" and/ or "software" boundaries, providing an opportunity for greater computational capacity and density within a programmable media. Reconfigurable Computing must leverage traditional CAD technology for building spatial designs.

Specifically we are interested in questions: Can a Reconfigurable custom computer be Applicable to every problem where a classical computer can be used? Is it a costly affair to solve a problem using Reconfigurable custom computer? And also does the Logic selection play a vital role in solving a particular problem? If so, which, why, and how? For answering the above questions we analyzed the fields in which Reconfigurable Custom Computing can be applicable, then the ways in which the Reconfiguration can be done, the technologies/ resources needed and also the parameters that need to be taken into consideration while selecting the logic in an RCC (Reconfigurable Custom Computing) for solving a particular problem. Here we restrict our thesis work in suggesting an algorithm in selecting an appropriate logic for the specified problem. Logic in this sense means algorithm.

In this thesis, we assume an architecture, which has a VLISVM (Very Large Instruction Set Virtual Machine), which can support all possible instructions. We have a Root machine, which has the compiler for compiling the instruction set in the VLISVM. The root machine identifies the Operation, which can be solved using a reconfigurable

custom made computing mechanism. Once it identifies the Operation it selects the needed Instruction set from the VLISVM. Now hardware software partitioning will be done in which it has to decide which part will be implemented in the hardware and which will run above it. Now our work suggests which hardware topology or architecture will be well suited for implementing a particular operation amongst the various options available. Because the Hardware topology and computation topology is modified by reallocating the resources to computation. While suggesting the Architecture various parameters have to be taken into consideration like the priority to be given to the selected problem and the amount of resources available in terms of RAM where the Reconfigured architecture will be downloaded. Obviously the selection of logic, to be implemented as the new hardware architecture plays a vital role because of the reason that a problem can be solved in various ways using various architectural logics. We have taken a small example and tried to solve using various logics, and also attention has been given in the selection of the type of logic.

We focus on the Reconfiguration part and the one used will be of the category, Architectural reconfiguration. This Method of reconfiguration will be Coarse grained because the reconfiguration is merely the programming the interconnections between fixed blocks. No reconfiguration in terms of transistors gates and registers.

CANDIDATE DECLARARTION

This is to certify that thesis entitled, "**Methodology For Designing High Speed Reconfigurable Custom Computer**" submitted by V.S.Shankar Sriram (8023117), in the partial fulfillment of the requirement for the award of degree of Master of Engineering in Software Engineering at Thapar Institute of Engineering & Technology (Deemed University), Patiala, is a bonafide work carried by him under my supervision and guidance.

Mr. Lalit Garg
Lecturer
Department of Computer Science & Engineering
Thapar Institute of Engineering & Technology
Patiala

Ms. Seema Bawa
H.O.D
Department of Computer Science & Engineering
Thapar Institute of Engineering & Technology
Patiala

Dr. D. S. Bawa
Dean Of Academic Affairs
Thapar Institute of Engineering & Technology
Patiala

ACKNOWLEDGMENT

I express my gratitude to Mr. Lalit Garg under whose inspiration, encouragement and guidance I have done my thesis. He let me work on my thesis in complete freedom while strongly supporting my academic endeavors, no matter where they took me. I would like to thank him for introducing me to the problem and providing invaluable advice throughout the course of thesis.

It was a pleasure working at T.I.E.T., and this is mostly due to the wonderful people who have sojourned there over the past years.

This work could never have been accomplished without the inspiration, guidance and support of innumerable friends and colleagues.

LIST OF FIGURES

Number *Page*

Figure 2.1 A more practical hybrid machine, combining a traditional processor with a Reconfigurable device. 8

Figure 2.2 Spatial versus Temporal Computation 10

Figure 2.3 Spatially Configurable Implementation 10

Figure 2.4 Computation definition point 11

Figure3.1 The Y Chart 14

Figure3.2 The Synthesis flow 15

Figure3.3 The growing of transistor density in processors and FPGAs devices 26

Figure 4.1 Basic Architecture of today's Commercial Reconfigurable Processor 39

Figure 4.2 A regular mesh 40

Figure 4.3 Reconfigurable Mesh 41

Figure 5.1 Parallel in Parallel out Architecture 46

Figure 5.2 Serial in Serial out Architecture 47

Figure 5.3 The Proposed Architecture 50

LIST OF TABLES

Number		*Page*
Table 3.1	A sampling of famous FPGA chips available from Xilinx.	25
Table 4.1	comparison of various Reconfigurable systems implemented successfully.	37
Table 4.2	Technical details of the different coarse-grained Reconfigurable architectures	37
Table 4.2	Areas of Application of Reconfigurable computing	37

TABLE OF CONTENTS

Abstract....ii

Acknowledgement....v

1. Introduction....1
   1.1. Introduction....1
   1.2. Problem Specification....2
   1.3. Organization of Thesis....3

2. Reconfigurable custom computing....5
   2.1. Brief History....5
   2.2. Basics Terminologies....8
   2.3. Reconfigurable computing as a new model....8
   2.4. Hardware Vs Software→ Spatial Vs Temporal....10

3. VLSI and FPGA....13
   3.1. Introduction to VLSI....13
   3.2. Design Domains....14
   3.3. The Steps Involved....16
   3.4. Design Actions....17
   3.5. Design Issues And Tools....17
   3.6. High-Level Synthesis....18
   3.7. Scheduling....20
   3.8. Exploring FPGA (Field Programmable Gate Array)....20
      3.8.1 The Revolutionary Technology....23
      3.8.2 FPGA Based Architectures....24

4. FPGA in Reconfigurable Custom Computing....28
   4.1. Introduction....28
   4.2. FPGAs As Enabling Technology....29
   4.3. Reconfiguration Models....31
      4.3.1 Runtime Reconfiguration Categories....33
      4.3.2 Reconfigurability By Design....35
   4.4. Methods Of Reconfiguration....35
   4.5 Couplings Possible....36
      4.5.1 List Of Reconfigurable Computing Architectures....36
   4.6 Reconfigurable Functional Unit (RFU)....38
   4.7 Reconfigurable Computing Models....40
   4.8 Areas Of Application Of Reconfigurable Computing....42

5. Designing A High Speed RCC 43
5.1. Introduction to the problem 43
5.2. Problem Taken 44
5.3. The Methodology 47
5.3.1 Architecture 50
5.3.2 Reconfiguration 50
5.4. Suggested Algorithm 51
5.5. Step by Step Explanation 52

6. Conclusion 55
6.1. Conclusion 55
6.2. Future Scope 56

References 58

CHAPTER 1 INTRODUCTION

"There is remarkably little shaping of computer structure to fit the function to be performed. At the root of this lies the general purpose nature of computers, in which all the functional specialization occurs at the time of programming and not at the time of design"

- Siewiorek, D.; Bell,C. and Newell, A.

1.1 INTRODUCTION

When multiple algorithms are required at different times, the designer has two choices: Choose a compromise set of algorithms and a single compromise architecture that is adequate to achieve performance; or choose an optimal set of algorithms and implement an adaptive architecture that can reconfigure to achieve performance on each algorithm. Reconfigurable computing offers the potential to achieve the second, more optimal approach. Reconfigurable custom computing machines (RCCMs) make use of field programmable gate arrays (FPGAs), chips that can be configured to implement an arbitrary hardware design. FPGAs can be reconfigured, even in the middle of a computation, by simply changing the bit file that provides the configuration. With such a machine, a user designs a special purpose hardware solution for a computational problem that "best fits" it, as if designing a software solution. The design is then synthesized into logic and the synthesized logic used to configure the FPGAs. In other words to these architectures, standard and customer-derived logic engines can be easily added, modified and extended as needed. By moving discrete logic functionality to internal FPGA the designer gets a highly flexible logic solver, based around a standard

processor core. With FPGA logic instead of foundry logic, the logic can be easily revised at any point in the design cycle. “Specials” or custom logic configurations can be quickly created for potential new customers driving down the batch size to single unit quantities.

Identifying the "best fit" of a given application starts with exploring how best to map the application architecture to the set of available resources so as to provide the most efficient use of these resources while delivering the most application performance. Not all applications are suitable for an RCCM.

The Cost involved in developing FPGA based RCCMs should also be taken into consideration. Achieving speed, performance and accuracy in an ASIC(Application Specific Integrated Circuit) specially designed for a particular application is not a big issue but to achieve the same factors using a conventional computing machine for the same application adds more luxury of reuse.

1.2 PROBLEM SPECIFICATION

In this thesis we suggest an algorithm for Logic or Algorithmic selection to solve a specific problem that can be implemented in RCCMs to solve it which can help in increased speed and performance thru puts [Lgarg]. Because the best fit solution for a specified problem will be the one in terms of an appropriate design of hardware and the logic/algorithm needed to solve that particular problem. My thesis work will be limited in suggesting the algorithm to choose among the various logics available needed to solve a particular problem .The reconfiguration of the hardware topology can be implemented using any of the Hardware description languages (HDLs) available like

VHDL(Very High Speed Integrated Circuit Hardware Description Language) or Verilog which is out of the scope of this thesis work. The parameters which need to be taken into consideration before opting for a particular logic are also discussed and based on the parameter the logic to be implemented will change for the specified problem[Lgarg]. In any conventional computation machine the specified problem can only be implemented using the predefined logic. It cannot be changed according to the necessity and need. The concept assumed is that we have a Virtual machine, a root machine and an FPGA to achieve this. It is assumed that the Virtual machine can handle all types of instruction set available. The compiler needed will be available in the root machine [Lgarg].

1.3 ORGANISATION OF THESIS

The **First chapter**, briefly introduces Reconfigurable custom computing machines (RCCM) and provides an insight into how it differs from classical computation models and the promises it hold.

The **Second chapter** of this thesis is devoted to brief history of Reconfigurable custom computing machines, some background material and various types of Reconfigurations that are available. With this in mind, the

Third chapter provides a overview of VLSI. This chapter gives a full exposure of the FPGA(Field Programmable Gate array). Technology

The **Fourth chapter** concerns on the technologies used to reconfigure and justifies why FPGA based technology is the best. The cost involved in doing this and various other factors. The fields in which they have been implemented so for.

The **Fifth chapter** suggests the algorithm what we would like to suggest.

Finally, concluding the thesis in **Sixth chapter** with future scope and directions. And at the end the references from where the materials were collected and used.

CHAPTER 2 RECONFIGURABLE CUSTOM COMPUTING

"We anticipate that a processor combined with reconfigurable resources can achieve a significant performance improvement over either a separate processor or a separate reconfigurable device on an interesting range of problems drawn from embedded computing applications. Reconfigurable devices have proven extremely efficient for certain types of processing tasks. The key to their cost/performance advantage is that conventional processors are often limited by instruction bandwidth and execution restrictions or by an insufficient number or type of functional units. Reconfigurable logic exploits more program parallelism. By dedicating significantly less instruction memory per active computing element, reconfigurable devices achieve a 10x improvement in functional density over microprocessors. At the same time this lower memory ratio allows reconfigurable devices to deploy active capacity at a finer grained level, allowing them to realize a higher yield of their raw capacity, sometimes as much as 10x, than conventional processors.."

BRASS Research Group,
B erkeleyReconfigurableArchitectures,
Systems, and Software.

2.1 BRIEF HISTORY

Any computation can be represented as a combination of abstract data-flow and Control flow graphs, with the nodes in the graphs being primitive operations such as integer addition or comparison. The primary function of a computer is to evaluate such graphs mechanically so as to accomplish some goal. Of course, real computer processors do not operate on such abstract graphs directly; instead, programs are encoded as a collection of machine instructions which can be executed one after another in a specific sequence. But this is just an artifact of the design of the machine, intended originally to simplify the processor's task (and perhaps the programmer's, too). Modern processors, in fact, re-expose instruction level parallelism by dynamically decoding

short sequences of machine instructions into their corresponding data and control-flow forms before executing them. Regardless of how a program is physically encoded, data-flow and control-flow graphs represent the true computation being performed.

A simple processor may have a single all-purpose functional unit known as an ALU (arithmetic and logic unit) that can only execute one operation at a time. More sophisticated processors (superscalar, VLIW) attempt to utilize multiple functional units of different kinds simultaneously to execute programs faster. In concept, the functional units are all a computer needs to evaluate the operations in a dataflow graph. In practice, a computer must also support the physical movement of data among functional units, as well as to and from memory. Computers with multiple functional units clearly have to move data between them, and this not always as trivial as it might sound. But even for a simple processor with a single ALU, there is never more than a small amount of data that can be stored very close to the ALU at one time. A practical general-purpose computer must have a memory hierarchy, with the fastest and smallest memory (usually the registers) closest to the functional units, and increasingly slower but more capacious memory correspondingly farther away. Moving data between different levels of memory, either explicitly or implicitly, is thus another indispensable computer operation.

Although the ideal is to use every functional unit every clock cycle, no processor can achieve this for all programs because of the data and control dependencies inherent in practical algorithms. The flexibility of the functional units, forwarding crossbar, and register file is what makes a standard processor programmable, capable of executing an

arbitrary application with decent performance, even applications not thought of when the processor was built.

The antithesis of a programmable processor is an ASIC (application-specific integrated circuit). In an ASIC, functional unit can be dedicated to individual program operations and wired together to match precisely the calculation being performed. The advantages an ASIC has over a programmable processor are threefold:

- Considerably less overhead is needed to control the mapping of functional units to operations and the routing of data values between them. On a programmable processor this overhead is manifest both in time and in die area.
- With smaller specialized functional units and less overhead circuitry around each one, more functional units can be fit into the same die area.
- Because the operation of each functional unit is known and planned out in advance, functional unit idleness can be minimized.

On a programmable processor, certain kinds of functional units might never be used by a specific application. Obviously, by their very definitions, a programmable device cannot be an ASIC and vice versa. However, as we shall see, reconfigurable devices such as FPGAs share characteristics of both processors and ASICs. On the one hand, FPGAs can implement ASIC-style circuits, while on the other; they are infinitely reprogrammable and thus immanently general-purpose. This leads to the question of whether reconfigurable hardware can capture some of the advantages of ASICs within a general-purpose computing environment.

2.2 BASIC TERMINOLOGIES

Configurable Computing Systems are those computing platforms whose architecture can be modified by the software to suit the application at hand.

The process of altering configurable logic on the fly from within an executable program is called **Run-Time Reconfiguration**.

The **Run-Time Programming of Configurable Computer Systems** gives executable programs the power to alter the 'logic' level of hardware to suit its own needs.

2.3 RECONFIGURABLE COMPUTING AS A NEW MODEL

Reconfigurable computing is one alternative to the superscalar and VLIW paradigms. The main distinction between a reconfigurable device and a standard processor is in the instruction stream: in its purest form, a reconfigurable device has no cycle-by-cycle instruction stream. Rather, the device is configured by loading a complete specification of the function of each part of the device at once. Once configured, the intention is for the device to run in that configuration for a decent interval before being reconfigured. Each configuration mimics an ASIC-like circuit, specialized for the particular task at hand. Changing configurations might take anywhere from a few clock cycles to a few thousand cycles. In accordance with the simpler programming mechanism, the dynamic forwarding crossbar is replaced by a less flexible configurable network for making static connections among the functional units; and short queues of retiming registers associated with each functional unit take the place of the traditional processor's shared, multi-ported register file. The familiar 90-10 rule asserts that 90% of execution time is consumed by about 10% of a program's

code, that 10% generally being inner loops. Reconfigurable devices excel in those cases where the computation represented by a configuration is repeated many times, so that the time required to load a configuration can be amortized over a long execution time and/or overlapped with other execution. When all of an application's important loop bodies can be configured to fit within the reconfigurable machine (one at a time), there would seem to be no need for the overhead of a fully dynamic instruction fetch and issue mechanism, allowing the machine to be leaner and more efficient.

By reducing the hardware to just the essentials needed to support computation, the reconfigurable design scales better to larger sizes than the more complex superscalar and VLIW styles. Although a naive expansion of the configurable network would cause it to grow quadratic ally with the number of functional units, it actually only needs to grow enough to support the connectivity required by real applications. Furthermore, unlike a superscalar or VLIW machine, reconfigurable hardware can easily exploit not only simple ILP but also inter-iteration and thread parallelism, making reconfigurable computing well poised to work with very large numbers of functional units. Fig. 1 shows here a possible hybrid device which is a combination of traditional processor with a reconfigurable device.

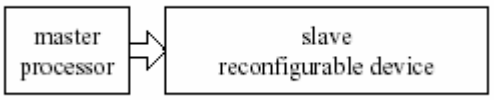

Fig2.1 A more practical hybrid machine, combining a traditional processor with a Reconfigurable device.

2.4 HARDWARE Vs SOFTWARE→ SPATIAL Vs TEMPORAL

When implementing a computation, we have traditionally decided between custom hardware implementations and software implementations. In some systems, we make this decision on a subtask by subtask basis, placing some subtasks in custom hardware and some in software on more general-purpose processing engines. Hardware designs offer high performance because they are:

- **customized to the problem**—no extra overhead for interpretation or extra circuitry capable of solving a more general problem
- **Relatively fast**—due to highly parallel, spatial execution

Software implementations exploit a "general-purpose" execution engine which interprets a designated data stream as instructions telling the engine what operations to perform. As a result, software is:

- **Flexible**—task can be changed simply by changing the instruction stream in rewriteable memory
- **Relatively slow**—due to mostly temporal execution
- **Relatively inefficient**–since operators can be poorly matched to computational task

Figure2.2 depicts the distinction between spatial and temporal computing. In spatial implementations, each operator exists at a different point in space, allowing the computation to exploit parallelism to achieve high throughput and low computational latencies. In temporal implementations, a small number of more general compute resources are reused in time, allowing the computation to be implemented compactly shows that when we have only these two options, we implicitly connect **spatial**

processing with hardware computation and temporal processing with software. [Andr´e]

Spatial Computation **Temporal Computation**

t1←x

t2←A * t1

t2← t2+b

t2← t2*t1

y ← t2+c

Fig. 2.2 Spatial versus Temporal Computation for the expression $y=Ax^2+Bx+C$[Andr´e]

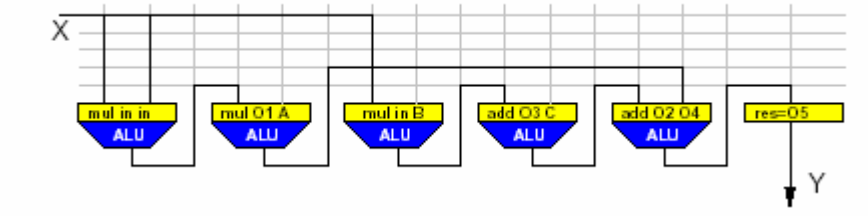

Fig. 2.3 Spatially Configurable Implementation of expression $y=Ax^2+Bx+C$[Andr´e]

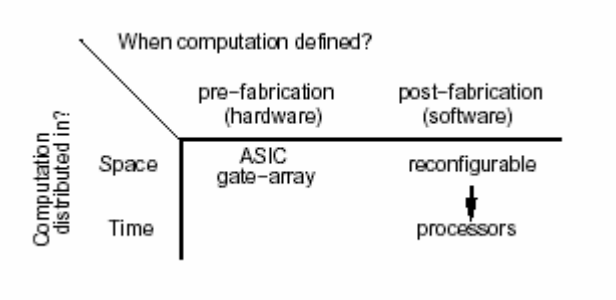

Fig. 2.4 Computation definition point[Andr´e]

The key benefit of FPGAs, and more broadly reconfigurable devices, is that they introduce a class of post-fabrication configurable devices which support spatial computations, thus giving us a new organizational point in this space (Figure 2.4). Figure 2.3 shows a spatially configurable computation for comparison with Figure 2.2. Reconfigurable devices have the obvious benefit of spatial parallelism, allowing them to perform more operations per cycle. As we will see the organization has inherent density advantages over traditional processor designs. As a result, reconfigurable can often pack this greater parallelism into the same die area as a modern processor. That is why a reconfigurable one can be a better that a conventional one in terms of speed and accuracy as it is custom made to suit the specific application.

CHAPTER 3 **VLSI and FPGA**

"New configuration approaches can lead to easier system designs, benefiting a range of applications. Reducing the overall pin count, interface complexity, and resource usage also enables FPGAs to be a flexible digital-signal-processing alternative to DSP architectures".

Jo Pletinckx, Rik Vlaminck, and Jan Vandewege,
Ghent University, Belgium -- EDN, 8/7/2003

3.1 INTRODUCTION TO VLSI

VLSI (Very Large Scale Integration) technology is the arena that has started booming up very recently world wide. VLSI technology has emerged as a very important technology in modern electronics featuring high speed, low voltage operations with minimized area in integrated circuits. The application of this technology ranges from consumer electronics to aerospace and defense electronics.

VLSI technology has advanced to such a state that it would be extremely complex to design digital systems starting at the transistor level or at the logic level. There has been an ever increasing need for design automation on more abstract levels where the functionality and tradeoffs can be clearly stated. This led to the development of CAD algorithms that could search the design space more thoroughly and find nearly optimal designs. Therefore, automation of the design process from conceptualization to silicon became more important and necessary. Designers provided a top-down methodology, where they describe the intent of the design and let CAD tools add detailed physical structure to it. This method of synthesizing systems from a design description became better suited for the design of complex systems. At this point of evolution, VLSI technology has reached a point where high-level synthesis (HLS) of

VLSI chips and electronic systems is becoming more cost effective and less time consuming than being fully hand designed by a group of designers.

The EDA tools replace the conventional design approaches and help the designers which are too expensive, time consuming and simply not applicable for large and complex circuit designs. HDLs are programming languages that have been designed and optimized for digital circuit design and modeling. The two most widely used HDLs are VHDL (Very High Speed Integrated Circuit Hardware Description Language) and Verilog. Although the widespread use of abstraction levels, there is no consensus about a design and synthesis representation model.

3.2 DESIGN DOMAINS:

- Behavioral: black box view
- Structural: interconnection of subblocks
- Physical: layout properties

Each design domain has its own hierarchy. The domains are further subdivided according to a set of general abstraction levels. Following the naming convention of [Gaj88], they are:

- system level
- algorithm level
- micro architectural level
- logic level
- circuit level

The domains and their subdivisions can be graphically visualized in a tri-partite diagram called Y-chart.

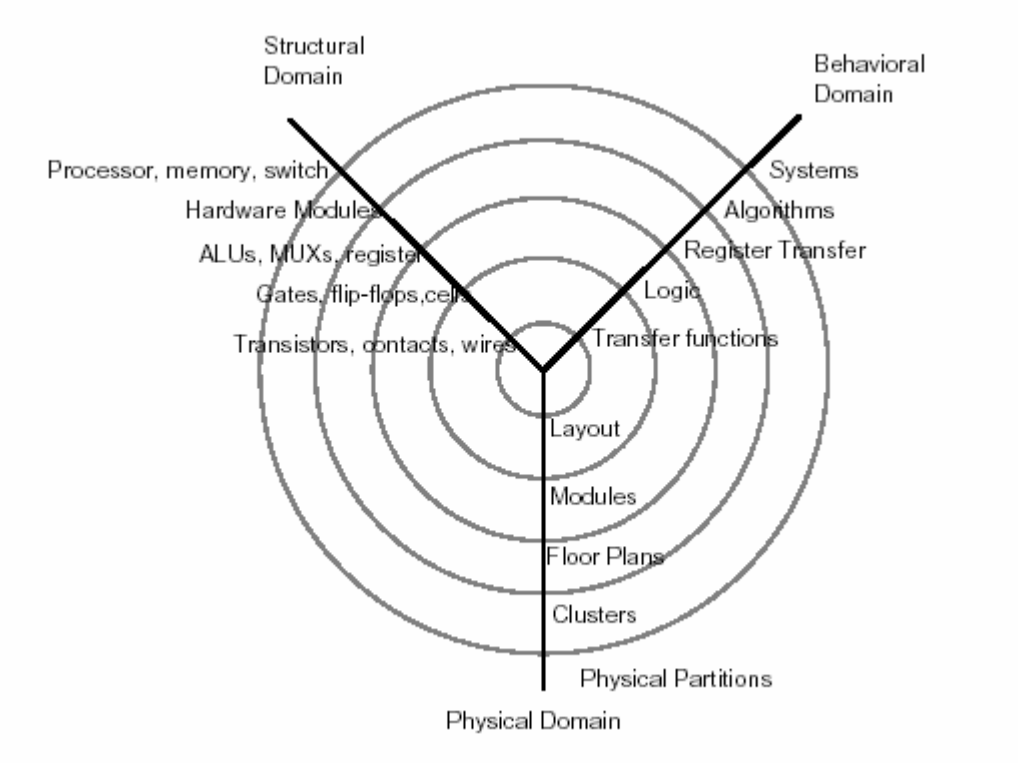

Fig 3.1 The Y Chart

Each domain correspond to an axle in the diagram. Each abstraction level is drawn as a circle that intersects every axle. The intersection points define the respective domain's abstraction levels. One design process can be described in this diagram by indicating its transitions from the circuit specification up to the final layout over the diagram. A single transition is denoted by an arrow that connects two abstraction levels.

- system level - inter-chip synchronization
- algorithm level - scheduling of the operations
- microarchitectural level - process synchronization
- logic level - gate delays
- circuit level - detailed devices timing

3.3 THE STEPS INVOLVED:

The **input description** is an algorithm written in a high level hardware description language. We have currently several ones, as Verilog, Silage, Hardware C and VHDL. The later deserves special attention as it was conceived to be the standard in this area. The **behavioral synthesis** consists in extracting from this initial description control and data flow graphs that are manipulated by scheduling and allocation algorithms to optimize the circuit architecture in terms of timing and size at a high abstraction level. The **architectural synthesis** takes these intermediate data and transform them into a more detailed register transfer level circuit representation. Operators are assigned to specific library modules and variables are associated to memory elements. Next step is the **logic synthesis** which provides detailed netlists for the combinational and sequential blocks. The final step is the **layout generation**, that yields the mask description to be sent to the foundry.

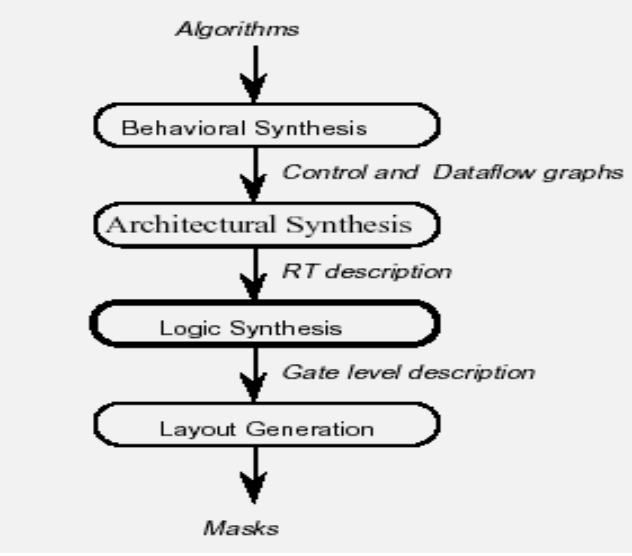

Fig.3.2 The Synthesis flow.

3.4 DESIGN ACTIONS

- **Synthesis:** increasing information about the design by providing more detail (within the same or another design domain).
- **Verification:** checking whether a synthesis step has left the specification intact.
- **Analysis:** collecting information on the quality of the design.
- **Optimization:** increasing the quality of the design by rearrangements in a given description.
- **Design Management:** storage of design data, cooperation between tools, design flow, etc.

3.5 DESIGN ISSUES AND TOOLS

System-level design:

- Partitioning into hardware and software, co-design, co-simulation, etc.
- Cost estimation, design-space exploration

Algorithmic-level design:

- Behavioral descriptions (e.g. in VHDL)
- High-level simulation

From algorithms to hardware modules:

- High-level (or architectural) synthesis

Logic design:

- Schematic entry
- Register-transfer level and logic synthesis
- Gate-level simulation (functionality, power)
- Timing analysis

- Formal verification

Transistor-level design:

- Switch-level simulation
- Circuit simulation

Layout design:

- Floor planning
- Module and cell generation
- Placement and routing
- Layout editing: symbolic and at mask level
- Layout compaction
- Design-rule checking
- Layout extraction

Design management:

- Data bases, frame works, etc.
- The FPGA helps in achieving better, faster, economical and efficient design.

3.6 HIGH-LEVEL SYNTHESIS

High-level synthesis can be described as the process of translation of a behavioral description into a structural description that consists of a set of connected components called the data-path and a controller that sequences and controls the functioning of these components. High-level synthesis starts at the systems level and proceeds downwards to register transfer (RT) level, logic level and finally circuit level,

each time adding some additional information needed at the next level of synthesis. The five major tasks involved in high-level synthesis are described below. The first three steps lead to the data-path formation and the last step leads to the formation of the controller.

1) Compilation : Compilation involves translation of the design description into an intermediate representation that is most suitable for high-level synthesis.
2) Partitioning : Partitioning deals with division of the intermediate representation (i.e, the behavioral description or the design) into sub-representations in order to reduce the problem size.
3) Scheduling : Scheduling partitions the intermediate representation into time steps, thereby generating a finite state machine model.
4) Allocation : Allocation though closely intertwined with scheduling, involves partitioning of intermediate representation with respect to space (hardware resources) which is also known as spatial mapping.
5) Control generation : Finally, this step involves the derivation of the controller that sequences the design and controls the functional and storage units in the datapath.

Scheduling is one of the most important and primary tasks in high-level synthesis. The following section gives a brief description of what scheduling is and why it is necessary.

3.7 SCHEDULING

A Finite State Machine with Datapath (FSMD) model is the most popular one that is used to describe digital systems at the register transfer level. It consists of an FSM called the control unit and a datapath. The datapath consists of the storage and functional units necessary for the system. The FSM consists of a set of states, a set of transitions between states, and a set of actions (involving the datapath) associated with each transition.

Scheduling can be described as the process of dividing the intermediate representation into states and control steps, in such a way that it can directly synthesized into an FSMD model. In other words scheduling does a temporal mapping of the given representation. A behavioral description and hence the intermediate representation consists of a sequence of operations to be performed by the synthesized hardware. The task of scheduling, partitions these operations into time steps such that each operation is executed in one time step. Each time/control step corresponds to one state of the controlling finite state machine in the FSMD model.

3.8 EXPLORING FPGA (Field Programmable Gate Array) :

3.8.1 Introducing FPGA:

To start let us put this way. What are programmable and configurable devices? In which category does an FPGA come? What is the difference between them? **Programmable** -- we will use the term ‘‘programmable” to refer to architectures which heavily and rapidly reuse a single piece of active circuitry for many different functions.

The canonical example of a programmable device is a processor which may perform a different instruction on its ALU on every cycle. All processors, be they microcoded, SIMD, Vector, or VLIW are included in this category.

Configurable -- we use the term ‘‘configurable” to refer to architectures where the active circuitry can perform any of a number of different operations, but the function cannot be changed from cycle to cycle. FPGAs are our canonical example of a configurable device. Once the instruction has been ‘‘configured” into the device, it is not changed during an operational epoch.

One of the key distinction, then, is the balance between a piece of (1) active logic and its associated interconnect, and (2) the local memory to configure the operation of the logic and interconnect. We define one **configuration context** as the collection of bits which describe the behavior of a general-purpose machine on one operation cycle. One programming stream for a conventional FPGA containing instructions for every array element along with interconnect composes a ‘‘configuration context.” One might also think of each of the following as a configuration context.

- one instruction for scalar processor
- one VLIW word composed of instructions for all the parallel functional units
- one line of horizontal microcode

What is an FPGA?

Before the advent of programmable logic, custom logic circuits were built at the board level using standard components, or at the gate level in expensive application-

specific (custom) integrated circuits. The FPGA is an integrated circuit that contains many (64 to over 10,000) identical logic cells that can be viewed as standard components. Each logic cell can independently take on any one of a limited set of personalities. The individual cells are interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. The array of logic cells and interconnect form a fabric of basic building blocks for logic circuits. Complex designs are created by combining these basic blocks to create the desired circuit. [Andr]

What does a logic cell do?

The logic cell architecture varies between different device families. Generally speaking, each logic cell combines a few binary inputs (typically between 3 and 10) to one or two outputs according to a Boolean logic function specified in the user program . In most families, the user also has the option of registering the combinatorial output of the cell, so that clocked logic can be easily implemented. The cell's combinatorial logic may be physically implemented as a small look-up table memory (LUT) or as a set of multiplexers and gates. LUT devices tend to be a bit more flexible and provide more inputs per cell than multiplexer cells at the expense of propagation delay. [Andr]

So what does 'Field Programmable' mean?

Field Programmable means that the FPGA's function is defined by a user's program rather than by the manufacturer of the device. A typical integrated circuit performs a particular function defined at the time of manufacture. In contrast, the

FPGA's function is defined by a program written by someone other than the device manufacturer. Depending on the particular device, the program is either 'burned' in permanently or semi-permanently as part of a board assembly process, or is loaded from an external memory each time the device is powered up. This user programmability gives the user access to complex integrated designs without the high engineering costs associated with application specific integrated circuits. [Andr]

How are FPGA programs created?

Individually defining the many switch connections and cell logic functions would be a daunting task. Fortunately, this task is handled by special software. The software translates a user's schematic diagrams or textual hardware description language (HDLs) code then places and routes the translated design. Most of the software packages have hooks to allow the user to influence implementation, placement and routing to obtain better performance and utilization of the device. Libraries of more complex function macros (eg. adders) further simplify the design process by providing common circuits that are already optimized for speed or area. [Andr]

3.8.2 The Revolutionary technology:

Field programmable gate arrays (FPGAs) are considered to be the technology of future. It should be obvious that every application would be best served by custom circuitry targeted specifically for it; and, in fact, application-specific integrated circuits (ASICs) are often made in response to special needs. But no one can afford to turn out a custom chip for every application he wants to run; and even when they are feasible, state-of-the-art ASICs become more expensive every day. As technology has improved, a market has grown up instead for versatile off-the-shelf parts that can be programmed

to emulate arbitrary digital circuits in place of ASICs. FPGAs are one class of such devices, distinguished by their ability to be reprogrammed (reconfigured) any number of times.

The versatility and reprogrammability of FPGAs comes at a price. Only a few years ago, the algorithms that could be implemented in a single FPGA chip were fairly small. In 1995, for example, the largest FPGAs could be programmed for circuits of about 15,000 logic gates at most. Since a fast 32-bit adder requires a couple hundred gates, the capabilities of such devices were somewhat bounded.

More recently, though, FPGAs have reached a size where it is possible to implement reasonable subpieces of an application in a single FPGA part. This has led to a new concept for computing: if a processor were to include one or more FPGA-like devices, it could in theory support a specialized application-specific circuit for each program, or even for each stage of a program's execution. The unlimited reconfigurability of an FPGA permits a continuous sequence of custom circuits to be employed, each optimized for the task of the moment. Because FPGAs scale better than superscalar techniques, such designs have the potential to make better use of continuing advances in device electronics in the long term.

3.8.3 FPGA based architectures

There are three different degrees of FPGA hardware adaptivity for providing configurable computing:

_ Static: The configuration within the FPGA is the same throughout the lifetime of the system. This means no adaptivity at runtime.

_ Dynamic: The configuration (or parts of it) is changed from time to time to introduce changes to the device. This represents rare adaptation.

_ Context switching: A set of configurations are available in which the FPGA switch between using. This could provide computation speedup and represents frequent adaptation to the best configuration at a given time.

FPGA chips available from Xilinx in 2000 :

		number of 4-input lookup tables	bytes of on-chip data RAM
3.3 V	XC4036XLA	6,156	0
	XC4052XLA	9,196	0
	XC4085XLA	14,896	0
2.5 V	XC40150XV	24,624	0
	XC40250XV	40,204	0
2.5 V	XCV300	6,144	8,192
	XCV600	13,824	12,288
	XCV1000	24,576	16,384
1.8 V	XCV812E	18,816	143,360
	XCV1000E	24,576	49,152
	XCV2000E	38,400	81,920
	XCV3200E	64,896	106,496

Table 3.1 A sampling of FPGA chips available from Xilinx in 2000. All the devices have additional capabilities beyond just the lookup tables and data memory listed. [Hau]

With the introduction of FPGAs with faster reconfiguration times and partial reconfiguration support, it is possible to use FPGAs in a dynamically reconfigurable environment. This technology makes possible the concept of unlimited hardware or "virtual hardware".

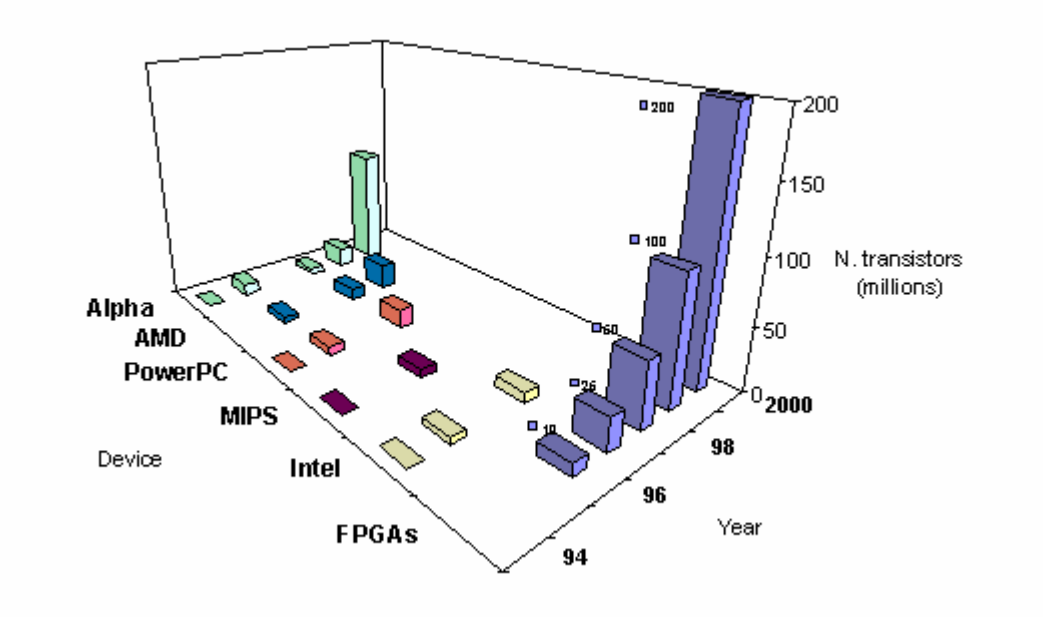

Fig 3.3 The growing of transistor density in processors and FPGAs devices [card]

- Despite these successes, any computer built wholly out of FPGAs must overcome some obstacles:
- FPGA machines are rarely large enough to encode entire interesting programs all at once. Smaller configurations handling different pieces of a program must be swapped in over time. However, configuration time is too expensive for any configuration to be used only briefly and discarded. In real programs, much code is not repeated often enough to be worth loading into an FPGA.
- No circuit constructed with an FPGA can be as efficient as the same circuit in dedicated hardware. Standard functions like multiplications and floating-point operations are big and slow in an FPGA when compared to their counterparts in ordinary processors.

- Problems that are worth solving with FPGAs usually involve more data than can be kept in the FPGAs themselves. No standard model exists for attaching external memory to FPGAs. FPGA-based machines typically include ad hoc memory systems, designed specifically for the first application envisaged for the machine.

CHAPTER 4

FPGA IN RECONFIGURABLE CUSTOM COMPUTING

"Reconfigurable Computing is emerging as an important new organizational structure for implementing computations. It combines the post-fabrication programmability of processors with the spatial computational style most commonly employed in hardware designs. The result changes traditional "hardware" and "software" boundaries, providing an opportunity for greater computational capacity and density within a programmable media."

Andr´e DeHon and John Wawrzynek
Berkeley Reconfigurable, Architectures,
Software, and Systems
Computer Science Division
University of California at Berkeley

4.1 INTRODUCTION

The most common operations that can change the architecture of a system are :

1) Addition of new components: it can be necessary to include new components to an architecture. The ADL must allow the inclusion of a component that was not being used before;

2) Upgrading existing components: a component can be replaced by another with the same signature (interface), but with better performance. The ideal situation is to keep the original component running, if needed, while it is being upgraded;

3) Removal of unnecessary components: if a component is no longer being used by the architecture, it can be removed;

4) Reconfiguration of application architecture: after adding or removing components, it can be necessary to reconnect components and connectors; and

5) Reconfiguration of system architecture: it can be necessary to move a component from one machine to another. The architecture must support the modification of the mapping of components to processors.

4.2 FPGAs AS ENABLING TECHNOLOGY

FPGAs consist of an array of configurable logic blocks and a grid of wires that can be programmed to interconnect the logic blocks. In SRAM-based FPGAs, the function of the logic blocks and the interconnections are controlled by SRAM cells. By writing these

cells, the FPGA is configured. The most important FPGA parameters for applications in reconfigurable computing are: block granularity, density, and reconfiguration time.

Block granularity A logic block consists of ip-ops, a combinatorial function block, and/or RAM cells. The combinatorial function is implemented either by a look-up table (LUT) or by multiplexors. The very popular Xilinx series XC4000 is an example for more complex blocks, where a block contains two 4-bit input LUTs followed by a 3-bit input LUT and two ip-ops. The rather new series Xilinx XC6200 is an example for a finer granularity; here each block implements any 2-bit input function and contains one ip-op.

Density The density is measured by the number of logic blocks in an FPGA or alternatively by the gate count in equivalent 2-input NAND gates. As different vendors use different counting methods, the gate count is a rather dubious parameter. The FPGA XC4085XL from the XC4000 series contains a 56_56 array of logic blocks; the XC6264 from the finer-grained XC6200 series contains a 128 _ 128 block array. The

reported number of equivalent gates for these FPGA types is in the range of 100K - 200K.

Reconfiguration time The reconfiguration time is the time required to write all the SRAM configuration cells of an FPGA. For the XC4000 series this time is 1 - 50 ms, depending on the device density. The XC6200 series was especially designed for fast and partial reconfiguration. Hence, the XC6264 can be reconfigured in about 400 ns. In a partial reconfiguration, only a portion of the FPGA's logic blocks and routing resources are reconfigured. The XC6200 series has a reconfiguration time of about 100 ns per logic block.

The density will further increase along with smaller feature sizes in semiconductor technology. It is expected that the 1M gate FPGA will be built in 3-5 years. For the reconfiguration time, only moderate improvements can be expected. This is due to the fact that writing an SRAM cell in the FPGA is an external memory access. However, current FPGA trends are to increase the bandwidth to the FPGA by using wider data buses or to store multiple configurations on an FPGA and to switch between these configurations.

The optimal block granularity in reconfigurable computing is a matter of ongoing discussion. More complex blocks are more efficient in area and delay for many arithmetic operations and are also more amenable to computer-aided design tools, e.g., logic synthesis. On the other hand, blocks with finer granularity offer greater flexibility, are very effcient for bit-sliced designs, and usually offer more interconnection resources per logic block gates.

4.3 RECONFIGURATION MODELS

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

Compile-Time Reconfiguration (CTR)

The reconfigurable hardware in the system is configured prior to the application's run-time, i.e., at compile-time, and remains static during the run-time. The main goal of CTR architectures is to achieve high performance as hardware accelerators. The critical, i.e., most time-consuming, portions of an application are extracted and synthesized to run on the reconfigurable hardware. The extraction of critical program portions is either done automatically or by user assistance. The granularity of accelerated program portions ranges from instructions to procedures.

Two successful and well-documented examples of hardware accelerators are the DECPerLe-1 architecture developed at the DEC Paris Lab and the Splash-2 architecture developed at the U.S. Supercomputing Research Center (SRC).1 Both architectures attach arrays of FPGAs to workstations. For applications that contain long integer operations, e.g., cryptography, or linear systolic applications, these machines have achieved a higher performance than conventional supercomputers.

Run-Time Reconfiguration (RTR)

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

functional density of FPGAs. The functional density is defined as D = 1=AT. This metric includes the area A in some unit hardware resources and the execution time T for a task. Generally, successfully applied RTR increases T slightly by the additional reconfiguration time, but decreases A to a much greater extend. RTR systems are used in two cases. In the first case, RTR architectures substitute larger CTR systems or systems with several ASICs in embedded systems. Here, the goal of RTR is to reduce the cost. The second case is applications where CTR systems are not feasible due to excessive hardware requirements.

Compared to CTR, RTR involves three additional problems: temporal partitioning, re-configuration overhead, and interconfiguration communication. In temporal partitioning, an application is split into time-exclusive segments. For each of these segments dedicated hardware is designed. Many applications break down into segments or operational phases quite naturally [14]. As the reconfiguration takes places during the application's run-time, the reconfiguration time becomes an overhead. This reconfiguration overhead, i.e., the ratio of the reconfiguration time to the execution time, must be kept as small as possible. Therefore, FPGAs with short reconfiguration times are of utmost importance. The problem of inter- configuration communication arises, when one FPGA configuration produces results that are used by other configurations. Communication methods are to store the results temporarily in a memory or in FPGA registers/RAM cells that are not destroyed during reconfiguration. These three issues are interrelated, as efficient temporal partitioning keeps the overhead from reconfiguration and communication low. RTR can be further divided into global and local RTR. In global RTR, the complete FPGA is reconfigured. To achieve an

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

4.3.1 Runtime Reconfiguration Categories:

We have identified the objective of reconfiguration into three categories primarily. These categories are: a) Algorithmic reconfiguration; b) Architectural reconfiguration; c) Functional reconfiguration.[Neem]

a) Algorithmic reconfiguration:

The goal in algorithmic reconfiguration is to reconfigure the system with a different computational algorithm that implements the same functionality, but with different

performance, accuracy, power, or resource requirements. The need for such reconfiguration arises when either the dynamics of the environment changes or the operational requirements change.

b) Architectural reconfiguration:

The goal in architectural reconfiguration is to modify the hardware topology and computation topology by reallocating resources to computations. The need for this type of reconfiguration arises in situations where some resources become unavailable either due to a fault, or due to reallocation to a higher priority job, or due to a shutdown in order to minimize the power usage. For the system to keep functioning in spite of the fault the hardware topology need to be modified and the computational tasks need to be reassigned.

c) Functional reconfiguration:

The goal in functional reconfiguration is to execute different function on the same resources. The need for this type of reconfiguration arises in situations where a large number of different functions are to be performed on a very limited resource envelope. In such situations the resources must be time-shared across different computational tasks to maximize resource utilization and minimize redundancy. The reconfiguration in this scenario helps improve the functional density of the FPGA device.

4.3.2 Reconfigurability by design:

For circuits which are inherently reconfigurable, a large proportion of the physical design is functionally static. A few well defined regions of the design, however, are identified as being reconfigurable. By effecting changes solely in these areas, the functionality of the overall circuit is modified within some restricted overall area of operation. For example, a multiplier does not turn into a VRAM circuit, but a *3 multiplier may become a *7 multiplier.

Dynamic reconfiguration of FPGAs has recently become viable with the introduction of devices that allow high speed partial reconfiguration, e.g., the Xilinx XC6200 series [14]. Dynamic reconfiguration is usually performed by a software system that decides when to reprogram part of the FPGA and with what. The simplest kind of run-time software simply selects a precompiled circuit and transmits the programming data directly to the FPGA. [Jim]

4.4 METHODS OF RECONFIGURATION:

The research efforts in reconfigurable computing are divided into two groups according to the level of the used hardware abstraction or granularity [Mar]

A) Fine-grained reconfiguration:

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

An important requirement in the context of fine grain reconfiguration is to have a high level representation of the circuit that in itself identifies areas of reconfigurability

symbolically. This representation is submitted to the system and the corresponding circuit is made resident on the board. Later, we refer to the reconfigurable areas of the circuit by their symbolic name. By passing this symbolic reference to the system with the circuit with which it is to be substituted, we can effect the desired style of reconfiguration.[Mar]

B) Coarse-grained Reconfiguration :

The hardware abstraction in this approach is based on a set of fixed blocks, like functional units, processor cores, and memory tiles. The reconfiguration is merely the reprogramming of the interconnections between the fixed blocks. [Mar]

4.5 COUPLINGS POSSIBLE:

A) Coprocessor:

The Reconfigurable unit is a part of the processor and placed next to the core. Examples: Garp, Morphosys and ONE chip98,NAPA

B) Attached processing unit:

The Reconfigurable unit is located outside the processor and connected to a memory or I/O bus contrary to RFUs and coprocessors. There are no extensions to the core processor's instruction set.

4.5.1 List of Reconfigurable Computing Architectures [Rami]

The Following are the list of Reconfigurable Architectures available.

CALISTO,CHESS Array, Colt, CS2000 Family, D-Fabrix , DP-FPGA , DReAM Array, FIPSOC , GARP, Kress Array , MATRIX , MECA Family, MorphoSys , PADDI-1, PADDI-2 , Pleiades , PipeRench , RaPiD , RAW , REMARC.

	PRISC [2] Harvard University	OneChip [3] Toronto University	Chimaera [4] Northwestern University	Garp [5] UC Berkeley	NAPA [6] National Semiconductor	REMARC [7] Stanford University	MorphoSys [8] UC Irvine	OneChip98 [9] Toronto University
Year	1994	1996	1997	1997	1998	1998	1998	1999
Application Domain	GP	GP	GP	GP	GP	MM	MM	GP
Processor Core	R2000	DLX	MIPS	MIPS-II	CompactRISC	MIPS-II	TinyRISC	S-DLX
Status	concept	emulation	concept	simulation	simulation	simulation	VLSI	emulation
Integration								
Coupling	RFU	RFU	RFU	coproc.	coproc.	coproc.	coproc.	coproc.
Concurrent Operation	inherent	inherent	inherent	yes	yes	yes	no	yes
Data Transfer	R	R	R	R/M	R/M/D	R	R/D	R/M
Reconfigurable Unit								
Granularity	fine	fine	fine	fine	fine	coarse	coarse	fine
Multiple Contexts	no	yes	yes	yes	no	yes	yes	yes
Context Fetching	L	P	C/P	P	L	L	P	P

Table 4.1 Comparison of various Reconfigurable systems implemented successfully. [Ralf]

Project	first publ.	Source	Architecture	Granularity	Fabrics	Mapping	intended target application
PADDI	1990	[29]	crossbar	16 bit	central crossbar	routing	DSP
PADDI-2	1993	[31]	crossbar	16 bit	multiple crossbar	routing	DSP and others
DP-FPGA	1994	[3]	2-D array	1 & 4 bit, multi-granular	inhomogenous routing channels	switchbox routing	regular datapaths
KressArray	1995	[4] [10]	2-D mesh	family: select pathwidth	multiple NN & bus segments	(co-)compilation	(adaptable)
Colt	1996	[11]	2-D array	1 & 16 bit inhomogenous	(sophisticated)	run time reconfiguration	highly dynamic reconfig.
RaPID	1996	[26]	1-D array	16 bit	segmented buses	channel routing	pipelining
Matrix	1996	[13]	2-D mesh	8 bit, multi-granular	8NN, length 4 & global lines	multi-length	general purpose
RAW	1997	[15]	2-D mesh	8 bit, multi-granular	8NN switched connections	switchbox rout	experimental
Garp	1997	[14]	2-D mesh	2 bit	global & semi-global lines	heuristic routing	loop acceleration
Pleiades	1997	[32]	mesh / crossbar	multi-granular	multiple segmented crossbar	switchbox routing	multimedia
PipeRench	1998	[28]	1-D array	128 bit	(sophisticated)	scheduling	pipelining
REMARC	1998	[16]	2-D mesh	16 bit	NN & full length buses	(information not available)	multimedia
MorphoSys	1999	[17]	2-D mesh	16 bit	NN, length 2 & 3 global lines	manual P&R	(not disclosed)
CHESS	1999	[18]	hexagon mesh	4 bit, multi-granular	8NN and buses	JHDL compilation	multimedia
DReAM	2000	[19]	2-D array	8 &16 bit	NN, segmented buses	co-compilation	next generation wireless
CS2000 family	2000	[21]	2-D array	16 & 32 bit	inhomogenous array	co-compilation	communication
MECA family	2000	[22]	2-D array		(not disclosed)		tele- & datacommunication
CALISTO	2000	[23]	2-D array		(not disclosed)		tele- & datacommunication
FIPSOC	2000	[24]	8x12, 8x16 array		(not disclosed)		tele- & datacommunication
flexible array	2000	[25]	2-D array	(not disclosed)	(not disclosed)	(not disclosed)	software radio

Table 4.2 Technical details of the different coarse-grained reconfigurable architectures Note: NN stands for “nearest neighbour”. [Rei]

4.6 RECONFIGURABLE FUNCTIONAL UNIT (RFU)

The Reconfigurable unit is integrated into the processor core as any other functional unit. Example: PRISC,ONECHIP,CHIMAERA. This is a new Programmable Logic Device from Xilinx Inc. for Run-Time Reconfigurable Computing. Programmable logic devices, particularly FPGAs, continue to gain momentum over traditional ASICs as the logic solution of choice for today's systems design.

In the last five years, research has shown that reconfigurable logic devices have a great potential in algorithm acceleration within a computing system environment. The multiple run-time reconfiguration use of PLDs is called Configurable or Reconfigurable Computing. The RPU is a natural evolution of the very successful Programmable Logic Device Class and as such should be familiar to users of FPGAs. The primary difference between the RPU and FPGA come from the added features needed for reconfigurable computing. The RFU needs to be integrated with the computer's microprocessing unit, operating system and the standard software language applications being used in the marketplace.

There are three unique features in an RFU:

- Open Architecture -- Resulting in 3rd Party development of tools and compilers
- Dynamically & Partially Reconfigurable Logic -- Enabling Hardware On Demand.
- A Microprocessor Interface -- Configuration times in micro seconds

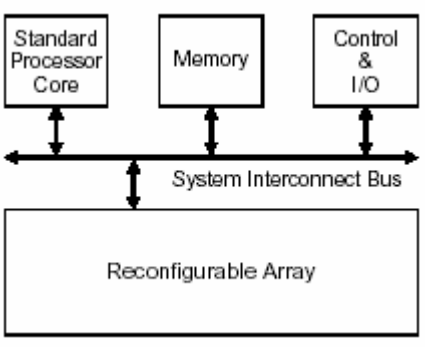

Fig. 4.1 Basic Architecture of today's Commercial Reconfigurable Processor

Recently established FPGA manufacturers and some start-ups have began to show strong interest in combining their FPGA know-how with standard processor cores. Example of such devices are Triscend's E5 and A7, Alter's two Excalibur Families, Atmel's FPSLIC and Chameleon System's CS2000.All these families basically share the same architecture. they integrate a standard processor core with an array of reconfigurable elements and memory on a single chip.

Reconfigurable Computing systems are those computing platforms whose architecture is modified by the software to suit the application at hand. This means that within the application program a software routine has been written to download a digital design (chip design) directly into the Reconfigurable Processing Unit (RPU) of the Reconfigurable Computer. It is like having custom processor chip manufactured for your application program. Most Reconfigurable Computing Systems are plug-in boards made for standard computers such as PCs and workstations. The Reconfigurable board acts as a Co-processor to the main MPU. Calculations normally done within the MPU are now carried out in the RPU instead. The main reasons for using this approach are:

- Very High Speed Performance Gains; the fastest version of any program calculation is one in which a computer chip has been designed just to perform that specific calculation only,
- Flexible Hardware; as new ways of solving problems arise, the RPU optimizes the compute power by implementing the calculations directly into a custom computing chip

4.7 RECONFIGURABLE COMPUTING MODELS:

A) Reconfigurable Mesh

A regular mesh (Fig 4.2)is a N X N square grid with one processor per grid point. Except for the processors at the boundary, every other processor is connected to its neighbors to the left, right, above and below through bi-directional links. The instruction stream is MIMD (that is each node can send and receive a packet from all its (four or less) neighbors in one unit time).

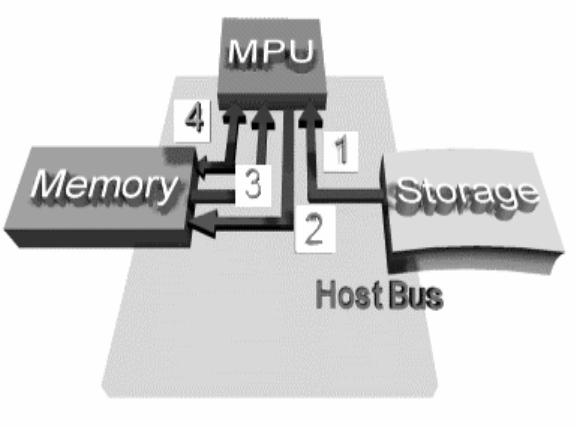

Fig. 4.2 A regular mesh [Kaar]

Reconfigurable Mesh (Fig 4.3) is a theoretical model of a VLSI array of processors

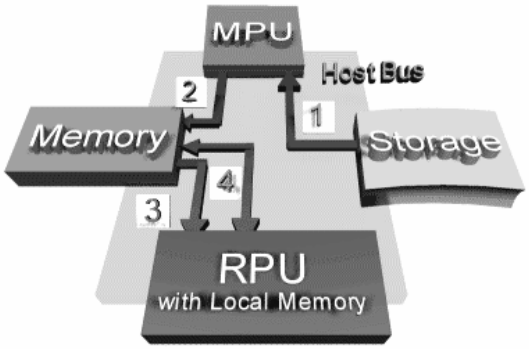

Fig 4.3 Reconfigurable Mesh [Kaar]

overlaid with a reconfigurable bus architecture. A reconfigurable bus architecture consists of a multi-dimensional array of processing elements connected to a bus through a fixed number of I/O ports. Bus reconfiguration is achieved by locally configuring the switches within each PE. Different shapes of buses such as rows, columns, diagonals, zig-zag and staircase can be formed by configuring the switches/ports. [Kaar]

B) Hybrid System Architecture model

Hybrid System Architecture model (HySAM) is a parameterized model of a configurable computing system, which consists of configurable logic attached to a traditional microprocessor. The architecture consists of a traditional processor, standard memory, configurable logic, configuration memory and data buffers communication through an interconnection network. [Kaar]

4.8 AREAS OF APPLICATION OF RECONFIGURABLE COMPUTING:

Reconfigurable architectures can be post fabrication customized for a wide class of applications, including multi-media, communications, networking, graphics and cryptography, to achieve significantly higher performance over general or even special-purpose processor alternatives (such as DSPs). For convenience, we will use the term FPGA to refer to any type of reconfigurable data path, whether implemented using FPGAs or other forms of reconfigurable logic.

The Following table shows some of the areas where Reconfigurable custom computing has been implemented successfully. It also shows the reconfigurable machine being used, the machine compared with, the speedup per chip and the year of implementation.

application	reconfigurable machine	compared against	speedup per chip	year
military target recognition	Splash-2 board, 16 Xilinx 4010's	110 MHz HP 770	7.5	1997
finding Golomb rulers	board with 20 Xilinx 5215's	167 MHz UltraSPARC	1.35	1998
10,000-bit multiply, divide, square root	1 Xilinx 4044XL	UltraSPARC	2–14	1998
frequency-domain sonar beamforming	1 Xilinx 4062XL	40 MHz ADI SHARC	8	1998
computing Goldbach partitions	1 Xilinx 40125	195 MHz MIPS R10000	>13 (1)	1998
rendering Bézier curves	PCI card with 1 Xilinx 6216	desktop PC	48 (2)	1998
genome matching, generalized profiles	VME card with 8 Xilinx 4013's	SPARC 20	7.9	1999
infrared military target recognition	PCI card with 16 Xilinx 4020's	180 MHz Pentium	1.24	1999

Table 4.3 Areas of Application of Reconfigurable computing [Hau]

CHAPTER 5 DESIGNING A HIGH SPEED RCC

"High Performance Computing architectures and Reconfigurable Computing systems have independently demonstrated performance advantages for applications such as digital signal processing and pattern recognition. By exploiting the near hardware specific speed of Reconfigurable Computing systems incorporated into a computer cluster, there is potential for significant performance advantages over software-only or uniprocessor solutions".

Peterson, Gregory D.;
CACI TECHNOLOGIES INC
CHANTILLY VA

5.1 INTRODUCTION TO THE PROBLEM

FPGAs can be reconfigured, even in the middle of a computation, by simply changing the bit file that provides the configuration. With such a machine, a user designs a special purpose hardware solution for a computational problem that "best fits" it, as if designing a software solution. The design is then synthesized into logic and the synthesized logic used to configure the FPGAs. [Recon]

Identifying the "best fit" of a given application starts with exploring how best to map the application architecture to the set of available resources so as to provide the most efficient use of these resources while delivering the most application performance. Not all applications are suitable for an RCM. Applications that involve extensive recursion, for example, are a poor match because the synthesized "hardware" must be of fixed size. But with many applications, including image and signal processing and biological sequence comparison, orders of magnitude speedup can be obtained. This is because hardware execution is inherently parallel, because the overhead of the fetch-

decode-execute cycle is eliminated, and because the "CPU" is custom-designed to accommodate computations not normally supported by instruction set architectures.

Let us assume a Scenario that a general purpose machine exists which can solve some specific problems using Reconfigurable Custom Computing. The machine along with the its routine operations can also render Reconfigurable services. Once the machine identifies a problem for which it has a reconfigurable solution, it switches from the normal mode of solving the problem to the reconfigured mode of solving the same making some alterations in the hardware. With this scenario in mind we have designed an algorithm for logic selection to be implemented for reconfiguration. These types of reconfigurations will be useful only when large amount of computation should be done on a single problem or in other words single operation, multiple/huge data

5.2 THE PROBLEM TAKEN:

We have taken a multiplication here for example. we can implement multiplication in three forms say,

A. Booth's implementation.

B. Parallel in parallel out and,

C. Serial in serial out

A. Booth's Multiplication Algorithm:

1. Depending on the current and previous bits, do one of the following:

00: a. Middle of a string of 0s, so no arithmetic operations.

01: b. End of a string of 1s, so add the multiplicand to the left half of the product.

10: c. Beginning of a string of 1s, so subtract the multiplicand from the left half of the product.

11: d. Middle of a string of 1s, so no arithmetic operation.

2. Shift the Product register right (arithmetic) 1 bit.

Example: 2 x 6

m=2, p=6;

m = 0010

p = 0000 0110

p

0000 0110 0 no-op

0000 0011 0 >> p

1110 0011 0 p = p - m

1111 0001 1 >> p

1111 0001 1 no-op

1111 1000 1 >> p

0001 1000 1 p = p + m

0000 1100 0 >> p

Ans =12.

For this we need to have an adder, a comparator and a shifter.

B. Parallel in parallel out:

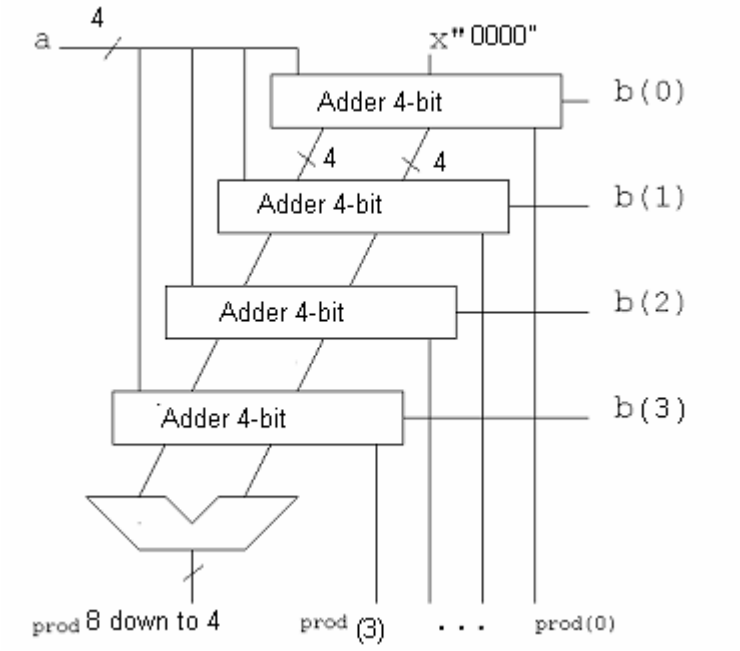

Fig 5.1 Architecture of a Parallel in Parallel out Multiplier

The Parallel in Parallel out Implementation of a Multiplier uses more amount of adders. Say if we have to multiply two 4 bit numbers we should have Four 4- bit adders. Here the Figure shows the architecture of how a Parallel in Parallel out architecture looks like.

C. Serial in Serial out:

The Serial in serial out mechanism is this way that it follows the logic of same parallel in parallel out but the difference is that there will be only one adder instead of 4 adders as in Parallel in Parallel out logic. Using Recursion or Repeatedly doing the Functionality of Parallel in Parallel out to a single adder instead of 4 Adders is the Base of this Serial in Serial out.

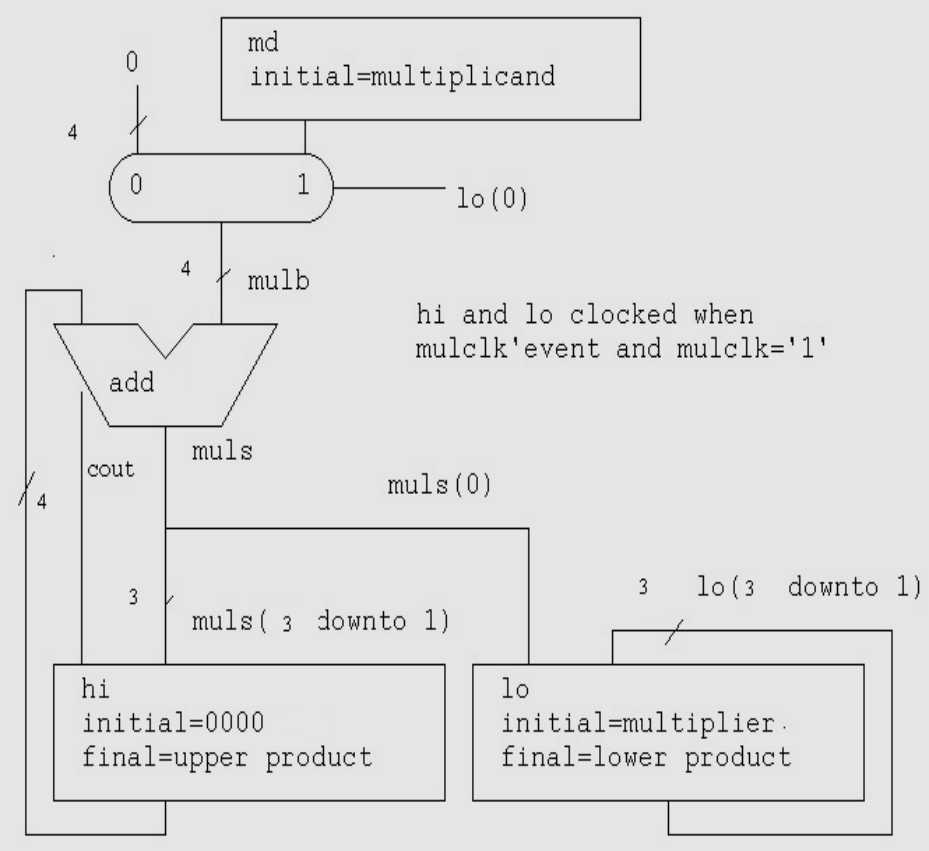

Fig 5.2 Architecture of a Serial in Serial out Multiplier

5.3 THE METHODOLOGY:

The methodology we are going to suggest for designing a high speed reconfigurable custom computer is as follows. Let us take a simple problem of multiplication for example to implement.

Step1:Very Large Instruction Set Virtual Machine (VLISVM):

Let us assume that we have got a virtual machine, which can support all possible instructions. This is a virtual machine, which will be helpful in achieving the high-speed reconfigurable custom computer. It is assumed that this virtual machine can handle all types of instruction set available. This includes the instruction set needed for implementing the multiplication operation which we have taken here as an example. we

can implement multiplication in three forms as stated Early. The instruction set needed for implementing all these three methods will be available here.

Step2:The Compiler for the stated VLISVM:

The compiler for the VLISM stated above will be in the Root machine. The purpose of this compiler is to identify the operations. The Root machine identifies that this is a problem which can be solved using Reconfigured methodology, it fetches the needed Instruction set to solve the problem from the VLISVM. Say if it identifies that it is a multiplication operation. It will select the appropriate instructions from the VLISVM from a complete set of all instruction.

Step3: Deciding the Operations to be performed:

Here once the instruction set has been decided then the operation to be performed should be decided. This will be some thing like the loading of necessary drivers and checking the condition of attached devices like the FPGA whether it is in proper condition and so on.

Step 4:Hardware/Software Partition:

‘‘what functionality to put in hardware and what functionality to put in software?” is all about hardware/software partitioning. In this step we will decide which operation will be coded in hardware and which will run/execute on top of the hardware instruction. Let us elaborate this in a better manner. Say we need to solve an addition and a multiplication problem, it is for granted that we need to have an adder to implement both addition and multiplication. Here the adder functionality can be

implemented in the hardware part and the rest computational logic part will be put in the software part.

In our Example as already stated we need to do multiplication. The Logics which can be used are one amongst the following .Booth's Implementation, Parallel in Parallel out and Serial in Serial out. In all the three cases we need to have Adders, Comparators and Shifter. So these functionalities will be implemented in the Hardware part and the logic needed to solve the problem using these hardware will be on the software part.

Step5: Decision making in selecting the Architecture:

Once the functionalities to be coded or implemented in hardware/Software form, has been decided in the previous step, logic selection should be made how each will have its architecture, among various logics available. This will be based on the Resources Available and urgency of the problem. Normally the RAM attached with the FPGA device where the logic is to be downloaded has two parts one for the Hardware and the Other for the software. We have to check it whether the RAM available is sufficient enough to implement a particular logic.

Say for the same example of multiplication should we have to go with the parallel in parallel out or serial in serial out or a booth implementation. Depending on the resources available the logic should be decided. (i.e.) if the Resources are free, the logic which consumes more resources and which is faster should be decided and if it is busy the logic which consumes less resource and comparatively slower should be chosen.

5.3.1 Architecture:

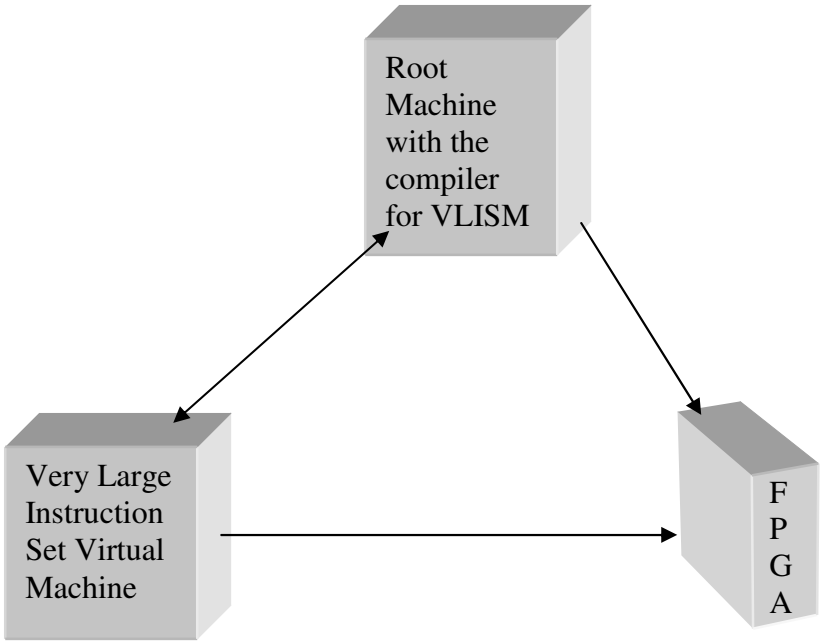

Fig 5.3 The Proposed Architecture

So here once the root machine identifies the problem which can be solved using RCC, it gets the needed instruction set from the VLISVM and compiles it. Now it is ready for the task. Then it identifies the

5.3.2 Reconfiguration:

The reconfigurable hardware in the system is configured prior to the application's run-time, i.e., at compile-time, and remains static during the run-time. This type of reconfiguration is called as compile time Reconfiguration as stated early. Our problem falls under this category. Before running the application the needed hardware will be ready and during runtime no change in the hardware is to be done.

Once the Hardware/ Software Partition is over an appropriate algorithm which can best fit the identified problem according to the present status of the machine in terms of resources and the urgency of the job to be completed is implemented. Then the reconfiguration takes place here. This will be downloading a precompiled code of a Hardware Description Language(like VHDL) for the logic selected or chosen, to the RAM based FPGA . The Reconfiguration used will be of the category Architectural reconfiguration . Because the Hardware topology and computation topology is modified by reallocating the resources to computation. This Method of reconfiguration will be Coarse grained because the reconfiguration is merely the programming the interconnections between fixed blocks. No reconfiguration in terms of transistors, gates and registers .

5.4 SUGGESTED ALGORITHM:

Begin_Designselect

Id_Problem _identity;

Isi_Instructionset_identifier;

Ps_problemset;

while(1)

if (Id==0) ***then***

continue operations in the root machine as normal one;

else

while (stop condition doesn't occur) ***do***

begin

compare Id, ps[n]

if Id==ps[n]

then

assign isi(ps[n]);

```
get Instructionset(isi);
end_if
/***decide the type of logic to be downloaded on to the FPGA***/
get status (Priority(Id)  &&  Resources)
 {
select Logic;
  }
```

download Logic;

5.5 STEP BY STEP EXPLANATION:

Step 1:

Identify the problem which can be implemented using Reconfigurable Hardware. Check for a match the current problem identity(Id_Problem _identity) with the Ps_problemset [n] which is a glossary of all the problems that the machine can solve using reconfigurable computing. If a match is found then the problem falls in the category which the machine can solve using RCC in a better manner. If so go to step 2. Else it is not a problem identified by the machine and use the default method of solving the problem in the root machine itself.

Step 2:

As a match in the Ps_problemset [n] is identified. Assign the corresponding Instruction set Identifier (ISI) for the problem and load the appropriate instruction set from the VLISVM(Very Large Instruction Set Virtual Machine) for solving that problem. Do the hardware / software Partition. which part to put on Hardware and which functionality to put of software?

Step 3:

Make a decision and Select the Appropriate logic, the best fit Architecture to be implemented after checking the following parameters. The resources, in terms of RAM and the Priority Assigned by the user. Here we are checking the RAM because of the reason that if already Reconfiguration has been done for some other problem previously then that may occupy the FPGA's RAM and if it is Vital we cannot Erase that at that point of time. So with the availability of the RAM in FPGA this decision has to be taken.

Step 4:

Once this has been done download the best fit logic to the FPGA and use it as many number of times as needed.

Step 6:

Continue with the normal operations in the machine. Goto Step 1

We have simulated this environment using c.

1. Our Program will identify the multiplication operation amongst a set of operations and it will inform the user that a reconfigurable problem identified on Line number this.
2. Then it will check the amount of Free space in the RAM and it will get input from the user whether the job is urgent or not.
3. Depending on the type of input made by the user and the amount of Resources available it will suggest an algorithm for implementing the multiplication operation.

For this we had to calculate the amount of time taken by each algorithm to solve a problem and the amount of memory space occupied by each algorithm.

The results proved that the Parallel in Parallel out Algorithm is a faster one and consumes more resources, next comes the Booth's implementation and the slowest one is the serial in serial out Implementation.

CHAPTER 6 CONCLUSION

"Nothing is more difficult than being successful. This Indeed, is the true test of life!."

-The sunlit Path, Sri Aurobindo Ashram.

6.1 CONCLUSION

The need for speed in computing machines is increasing day by day and technology has always given fruitful solutions whenever needed. Taking this into account what we feel is that this technology of Reconfigurable custom computing can provide high speed computing engines which are custom tailored and can prove better than an ordinary computing engine in terms of speed, accuracy and better resource utilization. As the FPGAs available nowadays are RAM based they can be reprogrammed any number of times which saves plenty of cost. Think of your ordinary computer performing equivalent to an Application Specific Computer for which it is meant of, that too at a cheaper cost. While reconfigurable computing is a promising approach to achieve ASIC performance with a programmable system, there are still many unsolved and not yet addressed problems. The most difficult of all turns out to be the development of programming models, that are both simple enough to allow the construction of efficient compilers and powerful enough to cover multiple levels of granularity. The availability of such programming models will decide if reconfigurable computing actually becomes the often cited new computing paradigm in general-purpose computing. The mismatch between hardware and software has always been a problem. With FPGA based Reconfigurable Custom computing engines coming into

scenario this mismatch problem can be easily over ruled because of the reason that the hardware and / or the software will be custom tailored to solve the particular Problem. Let us create a new world of reconfigurable custom computing engines which can solve any computational problem in a cheaper, effective and a faster manner with better levels of accuracy. The restriction here is that for simple minor operations going for a Reconfigurable Architecture is a time consuming process and if there are a large amount of computations are to be done on a bulk amount of data this can be effective. Because the Synthesis of the circuit also takes time taking into consideration this delay we suggest that it is not advisable to go for Reconfigurable custom computing for minor operations like multiplication of 10 numbers and so on.

6.2 FUTURE SCOPE

The field of Reconfigurable custom Computing is growing day by day significantly. With the rapidly changing demands for quicker turnaround times and higher performance at a lower cost, a development platform that allows for control and debugging of designs in Real-Time with real-data can enhance the engineer's success. The problems of integrating the hardware and software components are greater as the logic density and application sizes increase. As the boundaries between hardware and software blur so do the boundaries between the hardware engineer and software engineer. Taking Hard out of Hardware will be possible if and only if the Hardware architecture can be developed with respect to specific Applications. Better Speed and Better Accuracy at a cheaper cost or at an affordable cost is what the world is looking for.

Chapter 6 CONCLUSION

The suggested Future work goes this way. A Software can be designed to identify all the solvable problems using reconfigurable custom computing. This software should also suggest all the possible methods in which an identified problem can be solved. Infact this is a very huge work but it is not impossible. A single chip with the software loaded can be embedded with a classical computer which will help that machine in identifying the operation that can be implemented using custom based reconfiguration and the best fit logic that suits the machine with the present status of the machine in terms of resources and the urgency of the job. This logic should be downloaded to the machine's RAM based FPGA, if it is present in the machines database if not present it has to search the web for the precompiled design of the logic needed. If it is available it has to download it and impart the logic. Already research work is going on how to download a Precompiled logic which is needed, from the web.

Giants in FPGA technology may publish the available precompiled logic on to their web sites. So that those who need the precompiled logic for specific problem can get it downloaded whenever necessary.

REFERENCES

[Hau]Augmenting a Microprocessor with Reconfigurable Hardware
By John Reid Hauser, UNIVERSITY OF CALIFORNIA, BERKELEY, Fall 2000

[Neem] Sandeep Neema, Ted Bapty And Jason Scott, Isis, Vanderbilt University, Peabody, Box 36, 230 Appleton Place, Nashville, Tn 37203

[Kaar] SPECIAL PURPOSE HARDWARE FOR IMAGE PROCESSING
Kaarthik Sivashanmugam Dept. of Computer Science University of Georgia

[John]Garp: A MIPS Processor with a Reconfigurable Coprocessor, John R. Hauser and JohnWawrzynek, University of California, Berkeley

[Sch] A Hardware / Software Co-Design System using Configurable Computing Technology John Schewel Virtual Computer Corporation 6925 Canby Ave #103 Reseda, California, USA 91335

[Andr´e]Reconfigurable Computing: What, Why, and Implications for Design Automation Andr´e DeHon and John Wawrzynek, Berkeley Reconfigurable, Architectures, Software, and Systems, Computer Science Division University of California at Berkeley Berkeley, CA 94720-1776 contact: <andre@acm.org>

[Mar] Rekon_gurierbare Rechnerarchitekturen
Marco Platzner _ Computer Systems Laboratory, Stanford University

[Tim] Adapting Software Pipelining for Reconfigurable Computing Timothy J. Callahan and John Wawrzynek University of California at Berkeley

[Tham] A Re-configurable Processor for Petri Net Simulation, John Morris, Gary A Bundell and Sonny Tham Centre for Intelligent Information Processing Systems, Department of Electrical and Electronic Engineering, The University of Western Australia, Nedlands WA 6907, Australia [morris,bundell,tham-s]@ee.uwa.edu.au

[Bro] Configurable Microprocessor Implementation of Low Bit Rate Audio Decoding
Gary S. Brown Tensilica, Inc., Santa Clara, California, USA gsb@tensilica.com

[Sari] Code Transformations to Improve Memory Parallelism _ Vijay S. Pai VIJAYPAI@ECE.RICE.EDU Electrical and Computer Engineering – MS 366, Rice University Houston, TX 77005 USA Sarita Adve SADVE@CS.UIUC.EDU Computer Science, University of Illinois Urbana-Champaign, IL 61801 USA

[Davi]Fall 2002 - Lecture 1 The VLSI Design Process © 2002 Dr. James P. Davis
CSCE 611 Advanced VLSI Design Introduction Research Background Design Methodology Extrapolation

[Sau] Hardware/Software Codesign for FPGA-Based Systems
J.M. Saul, Programming Research Group, Oxford University Computing Laboratory,
Parks Road, Oxford, UK. Email: Jon.Saul@comlab.ox.ac.uk

[Bax] ICARUS: A Dynamically ReconØgurable Computer Architecture
Michael Baxter Ricoh Silicon Valley, 2882 Sand Hill Road #115, Menlo Park CA 94025-7022 mbaxter@rsv.ricoh.com

[recon] http://www.cse.sc.edu/research/reconfigurable
[Andr] http://www.andraka.com/whatisan.htm

[Olu]R. Helaihel and K. Olukotun. Emulation and Prototyping of Digital Systems. In Giovanni De Micheli and Mariagiovanna Sami, editors, Hardware/Software Co-Design, pages 339{366. Kluwer Academic Publishers, 1996.

[Jim]A Dynamic Reconfiguration Run-Time System1
Jim Burns, Adam Donlin, Jonathan Hogg, Satnam Singh, Mark de Wit
The Department of Computing Science
The University of Glasgow
Glasgow G12 8RZ, United Kingdom

[Ralf] Dynamically Reconfigurable Processors ,von Ralf Enzler and Marco Platzner

[card] Architectures and Compilers to Support Reconfigurable Computing

by João M. P. Cardoso and Mário P. Véstias

[rami] Alphabetical List of Reconfigurable Computing Architectures Rami Abielmona - rabielmo@site.uottawa.ca.ns - 2002

[Rei] A Decade of Reconfigurable Computing: a Visionary Retrospective
Reiner Hartenstein (embedded tutorial)
CS Dept. (Informatik), University of Kaiserslautern, Germany
http://www.fpl.uni-kl.de hartenst@rhrk.uni-kl.de