International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:2, 2009

RFU Based Computational Unit Design For Reconfigurable Processors

M. Aqeel Iqbal
Faculty of Engineering and Information Technology
Foundation University, Institute of Engineering and Management Sciences
New Lalazar, Rawalpindi Cantt. Pakistan
maqeeliqbal@hotmail.com

***Abstract* -** Fully customized hardware based technology provides high performance and low power consumption by specializing the tasks in hardware but lacks design flexibility since any kind of changes require re-design and re-fabrication. Software based solutions operate with software instructions due to which a great flexibility is achieved from the easy development and maintenance of the software code. But this execution of instructions introduces a high overhead in performance and area consumption. In past few decades the reconfigurable computing domain has been introduced which overcomes the traditional trades-off between flexibility and performance and is able to achieve high performance while maintaining a good flexibility. The dramatic gains in terms of chip performance and design flexibility achieved through the reconfigurable computing systems are greatly dependent on the design of their computational units being integrated with reconfigurable logic resources. The computational unit of any reconfigurable system plays vital role in defining its strength. In this research paper an RFU based computational unit design has been presented using the tightly coupled, multi-threaded reconfigurable cores. The proposed design has been simulated for VLIW based architectures and a high gain in performance has been observed as compared to the conventional computing systems.

Keywords - Configuration Stream, Configuration overhead, Configuration Controller, Reconfigurable devices.

I. Reconfigurable Computing

In conventional computing, there are two primary methods being used for the execution of algorithms. The first method is to use the hardwired technology, such as *Application Specific Integrated Circuits (ASICs)* to perform the operations in hardware. A fully customized hardware is specially designed to execute an application or dedicated operation. The high specialization will result in very fast and efficient execution for the designed task as well as the power consumption overhead is minimal since the designers will avoid to use unnecessary logic. However, the huge amount of development work is required to achieve the extreme efficiency. Also, the hardwired circuit cannot be altered after fabrication. Any changes, modifications or updates to circuit require a re-design and re-fabrication of the chip. This is an expensive process in effort, time and cost for maintaining hardwired technology. The second execution method is to use the software based programmable microprocessors commonly known as general purpose processors (GPP). Applications are represented in the form of sequenced codes. The microprocessor executes the codes or instructions to perform a computation. Instruction Set Architecture (ISA) interfaces between the instructions and the execution hardware. The changes in either end will not affect the functionality of the other side as long as the ISA specification was followed. Therefore changes in software instructions can alter the functionality of an operation without the need to change any of the hardware resources. It gives a great flexibility to the designers to freely modify the software code. This great flexibility is in trade-off with the huge performance overhead. During execution the processor fetches each instruction from the memory, decodes its meaning and then performs the operation of the instruction and hence it follows a conventional fetch-decode-execute instruction cycle which introduces a performance overhead. These overheads result in degraded performance and more power consumption. While ISA serves as an interface between the software and the hardware it also limits the potential growth of the system. After chip fabrication any operations to be implemented must be built based on the ISA specifications. Improvements in the hardware must maintain the full ISA specification even obsolete ones to be backward compatible with the existing software programs.

Reconfigurable Computing is introduced to fill the gap between hardware and software based systems. The goal is to achieve the performance better than that of software based solutions while maintaining the greater flexibility than that of the hardware based solutions. Reconfigurable computing, using reconfigurable processors, is based on reconfigurable core being integrated inside the processor as shown in Fig. 1. The reconfigurable core is composed of many computational elements whose functionality is determined through the programmable configurations. These elements some times known as Configurable Logic Blocks (CLBs) or Processing Units (PU), are connected by programmable routing resources. The idea of configuration is to map logic functions of a design to the processing units within a reconfigurable device and use the programmable interconnects to connect processing units together to form the necessary circuit. Huge flexibility comes from the programmable nature of processing elements and the routings. Performance can be much better than software based approaches due to the reduced execution overhead. Under this definition *Field Programmable Gate Array (FPGA)* is a form of reconfigurable computing device or system. FPGAs and other reconfigurable computing devices have been shown to accelerate a variety of the applications such as the encryption algorithms and the streaming applications.

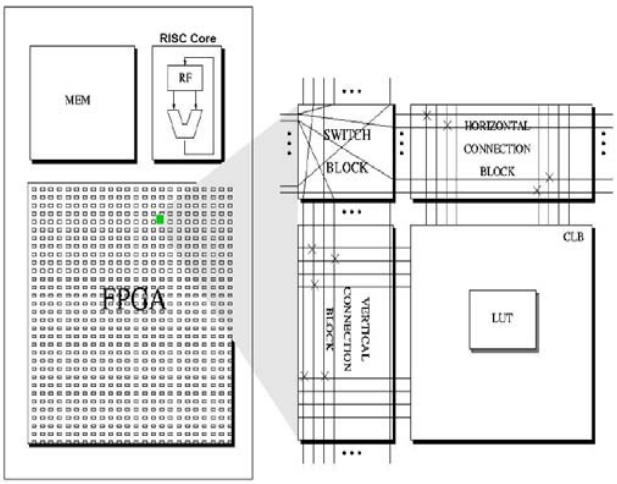


Fig. 1 Typical Reconfigurable Processor Architecture

While field programmable gate arrays demonstrated good performance in various applications, still it has shortcomings like mentioned below:

A. Logic Granularity: The Classic FPGAs have a much low granularity in its design. When processing units are chained together to form a bigger operation, the low granularity enables better utilization. However, there will be large overhead to control and connect the many processing units, resulting in performance penalty.

B. Support for Reconfiguration: Configuration of the FPGA is done at initialization. Reconfiguration for a new application program usually requires the chip to be taken down and reprogrammed. Certain FPGAs may support run time reconfiguration, but it may take up to thousands of micro seconds to complete.

C. Hardware Constraints: FPGAs can only implement the applications within the limited size of its hardware resources. This size restriction will also make compilation more difficult. The disadvantages of FPGAs make it an unsuitable choice in certain applications. Many reconfigurable computing systems are developed and under research to make up the shortcomings of FPGAs.

II. Related Work In Active Domain

A large no of reconfigurable processing architectures have been proposed in previous few decades. Previously proposed reconfigurable architectures generally fit into one of two major categories depending on the size of the computations they map onto the reconfigurable computational units and the nature of the design of reconfigurable computational units being used in them. In broader sense, a reconfigurable system either belongs to fine-grain design approach or to coarse-grain design approach.

A. Fine-grained Reconfigurable Architectures, such as CHIMERAE [4] integrate the small blocks of reconfigurable logic into superscalar processor architectures, treating the reconfigurable logic as programmable ALUs that can be configured to implement application-specific instructions. These systems can achieve the better performance than the conventional superscalar processors on a wide range of applications by mapping the commonly executed sequences of instructions onto their reconfigurable units, but the maximum speedup they can achieve is limited by the small amount of logic in their reconfigurable units.

B. Coarse-grained Reconfigurable Architectures, such as REMARC [5] provide larger blocks of reconfigurable logic that are less tightly-coupled with the programmable portions of the processor. These architectures can achieve extremely good performance on applications that contain long-running active nested loops that can be mapped onto the processor's reconfigurable arrays but perform less well on applications that require frequent communication between programmable and reconfigurable portions of the processor. Systems such as Pilchard that integrates FPGAs into conventional workstations over the processor's memory bus display similar behavior, although the relatively low bandwidth of a microprocessor's memory bus makes them even more sensitive to the amount of the communication that an application requires between the processor and the FPGA.

III. Proposed Architecture

The proposed computational unit is composed of a Configuration Unit (CONFU), which contains a Configuration Controller and a Multi-port Configuration Memory and a Bus Interface Unit (BIU), which contains the internal data-paths of computational unit and a set of fixed point registers and a Reconfigurable Core Unit (RCU), which contains an array of reconfigurable functional units (RFUs) being integrated with reconfigurable FPGA Cores. Consider Fig. 2 for the interfaces of the proposed computational unit with instruction pipeline.

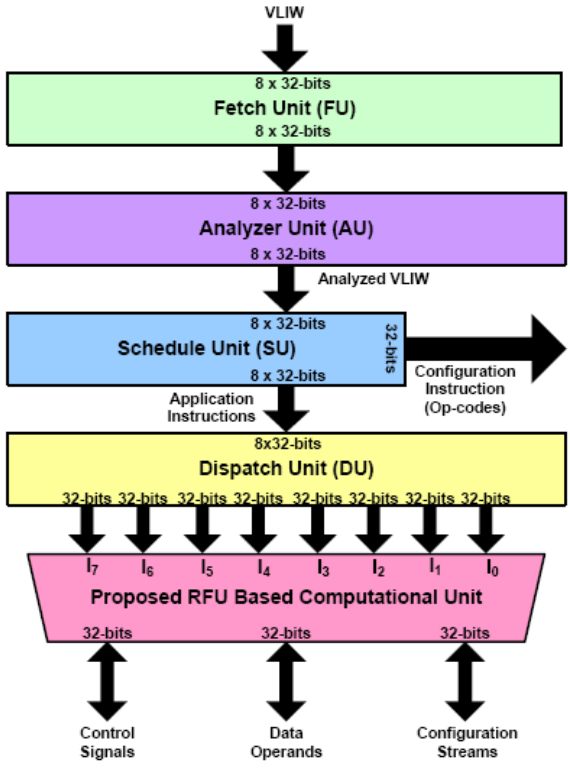


Fig. 2 Interfaces With Instruction Pipeline

A. Configuration Unit (CONFU): CONFU is composed of a *Configuration Controller* and a *Multi-port Configuration Memory* as shown in Fig. 3. Configuration controller receives the op-codes of the eight instructions of the VLIW from the fetch unit and on the basis of these op-codes it decides to load one of the configuration blocks available in the memory for each RFU (if required). Also it checks if the op-code is a No Operation (NOP) or is same as that of any one of the existing op-codes. If so then the configuration controller does not load this new configuration into the RFUs but the hardware that is already loaded in RFUs is reused and hence the configuration time that was required for the reconfiguration of RFUs is saved and only those RFUs are reconfigured that are new ones. Hence the processor always takes the minimum possible time to reconfigure the RFUs during the execution of the application program and always has the most optimal configuration overhead [1]. Hence such a kind of optimistic configuration controller contributes a great factor of speed in the execution of applications by reconfigurable processors.

The proposed computational unit has been dedicatedly interconnected with the configuration controller by using dedicated local buses as shown in Fig. 3. Since the proposed computational unit design is based on a VLIW architecture where it has eight reconfigurable functional units so that to activate a maximum of eight threads of applications, hence some times more than one RFU required configuration at the same time. If we reconfigure the multiple RFUs by using the conventional single-port configuration memories, then it will face a huge configuration overhead. Such a high configuration overhead greatly degrades the performance of computational unit and hence the performance of reconfigurable processor is drastically reduced [2], [3]. In order to avoid such kind of the performance reducing factors, a multi-port configuration memory has been used with eight configuration ports. In this way the computation unit is capable of loading configurations in all of its reconfigurable functional units at the same time.

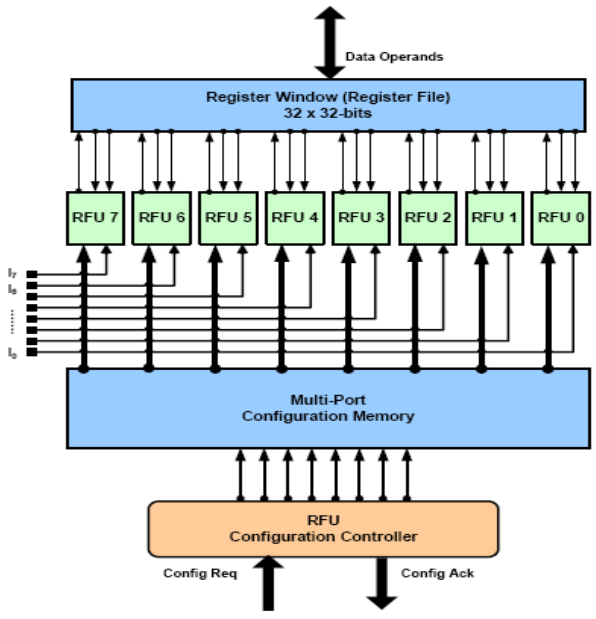


Fig. 3 Interfaces With Configuration Unit

B. Bus Interface Unit (BIU): Bus Interface Unit of the proposed computational unit contains a set of interconnecting local buses and a set of fixed point registers being used for program execution. Consider the Fig. 4. The BIU contains the following major modules.

a) External IO Logic (EIOL)
b) RFUs Data-in/Data-out Logic (RDIOL)
c) General-Purpose and Flag Registers (GFRs)
d) Registers Input/Output Logic (RIOL)

a) External IO Logic (EIOL)

The EIOL of the BIU is used to load instructions in the instruction register, source operands in general-purpose registers and the configuration stream in RFUs. The second job of the EIOL is to store the configuration stream being loaded in the RFUs for the analysis purpose and results being generated after the execution of VLIW. The source operands Sr-1and Sr-2 are loaded into the internal general-purpose registers (GPRs) by the External De-MUX of size 1 x 24. The address given for the Data-in is connected to the select lines of De-MUX as well as to Decoder (5 x 24) input. De-MUX selects one of the general-purpose registers for data loading and the decoder enables its output channel connecting to the registers through the MUX of the size 2 x1. This MUX receives 32-bits data operand from External De-MUX at input "1" and receives 32-bits results from RFUs at the input "0". If the Ext_IO_En=0 then it selects the result coming from the RFUs and loads it in the register. If the Ext_IO_En=1 then it selects the data coming from the External De-MUX and loads it in the registers. Since there are eight RFUs that can load their results in the same register, hence in order to solve this problem an 8 x 1 MUX (32-bits) is interfaced with each register input. Each MUX is controlled by the *RFU Data-path Controller* (see algorithm) which analyzes the Destination Addresses of all the RFUs and selects only that RFU whose output is valid output. In order to store the results and the flags being available in the GPRs and flag registers (FLGs) into the data cache of the processor, the 32 x 1 External MUX (32-bits) is used which can read the contents of the selected register and sends it to the data cache of the reconfigurable processor.

*b) RFUs Data-in / Data-out Logic (*RDIOL*)*

In order to load/store the data across the RFUs there are two 32 x 1 MUXs (32-bits) and one 1 x 24 De-MUX (32-bits) for each RFU as shown in Fig. 4. Using two MUXs the RFU is able to read the source data operands (Sr-1 and Sr-2) from any one of the 32 registers and using the one De-MUX it stores its results back to any one of the GPRs. Flags generated during execution of the VLIW are loaded into relevant FLGs.

c) General-Purpose and Flag Registers (GFRs)

There are eight FLGs (32-bits) and twenty four GPRs (32-bits). GPRs can be read and written by programmer but the FLGs can only be read by the programmer and can not be written. RFUs can read/write any one of these thirty two registers. More than one RFU can read the contents of the same register at the same time but only one RFU can write in a register at the same time because the read operation is shareable but the write operation is not shareable.

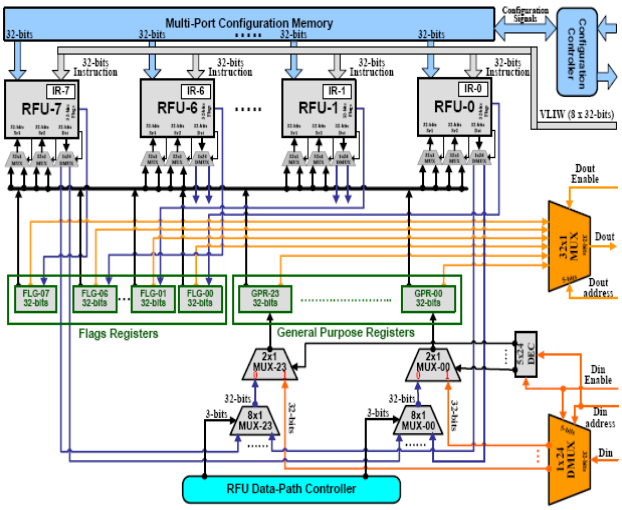


Fig. 4 Proposed Computational Unit Design

d) Registers Input/Output Logic (RIOL)

FLGs are loaded with the flags, being generated by the RFUs and can be read by the programmer through the External MUX. In case of the GPRs, the programmer can read the registers through the External MUX but in order to write contents into registers there is a 2 x 1 MUX (32-bits) which selects the data for the register either from some RFU output or from data cache. The 8 x 1 MUX interfaced at the input of the 2 x 1 MUX selects the valid RFU for the results to be stored in the register. In order to select the valid RFU for results, there is a RFU Data path Controller (whose algorithm is shown) being attached with all MUXs. This controller reads the select lines of all the De-MUXs of RFUs and after analysis it selects that RFU whose output is a valid output.

C. Reconfigurable Core Unit (RCU): Reconfigurable core unit is consisting of a layer of eight RFUs that are the computational units of reconfigurable processor and can be reconfigured at any time according to the demands of the running applications. They have been tightly coupled with integrated field programmable gate array cores as shown in Fig. 5. The outputs generated by the RFUs are also read by the FGL and the flags are calculated for each RFU. Flag register is a 32-bits register but recently only Carry Flag, Sign Flag, Zero Flag, Overflow Flag and Equal Flag have been computed in the system and the remaining twenty-seven bits are available for the future extension.

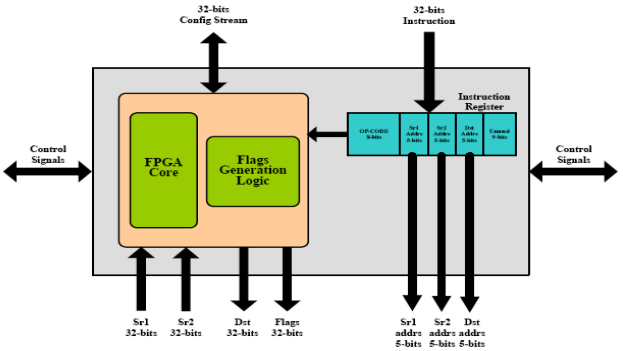


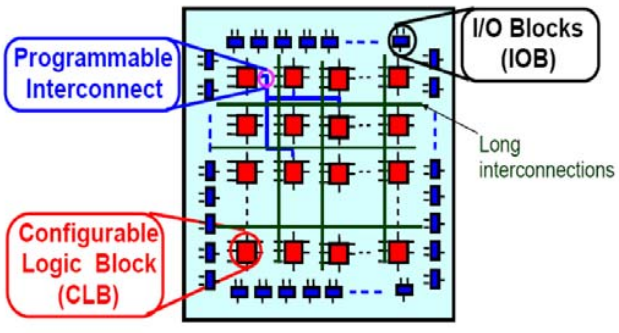
Fig. 5 Reconfigurable Functional Unit (RFU)

RFU Data-path Controller Algorithm

The Algorithm Initially Reads the Register Addresses (GPR-mn) and Destination Operand Addresses (RFUn-Dest-Address) of all RFUs

```
if (RFU0-Dest-Address = GPR-00 Address)
    Then Select-Line = 0;

else if (RFU1-Dest-Address = GPR-00 Address)
    Then Select-Line = 1;

else if (RFU2-Dest-Address = GPR-00 Address)
    Then Select-Line = 2;
.....
.....
.....
else if (RFU7-Dest-Address = GPR-00 Address)
    Then Select-Line = 7;
else Select-Line = Default; // Default = Nil;

.....
.....
.....
.....
.....

    if (RFU0-Dest-Address = GPR-23 Address)
    Then Select-Line = 0;

else if (RFU1-Dest-Address = GPR-23 Address)
    Then Select-Line = 1;

else if (RFU2-Dest-Address = GPR-23 Address)
    Then Select-Line = 2;
.....
.....
.....
else if (RFU7-Dest-Address = GPR-23 Address)
    Then Select-Line = 7;
else Select-Line = Default; // Default = Nil;
```

Field Programmable Gate Array (FPGAs) consists of an array of Configurable Logic Blocks (CLBs) overlaid with an interconnection network of wires known as *Programmable Interconnect* as shown in Fig. 6. Both the logic blocks and the interconnection network are configurable. The configurability is achieved by using either anti-fuse elements or SRAM memory bits to control the configurations of transistors. The Anti-fuse technology utilizes strong electric currents to create a connection between two terminals and is typically less reprogrammable. The SRAM based configuration can be reprogrammed on fly by downloading different configuration bits into the SRAM memory cells.



SRAMS cells throughout the FPGA determine the functionality of the device

Fig. 6 FPGA Architecture

Typical configurable logic block architectures contain a look-up table, a flip-flop, an additional combinational logic and SRAM memory cells to control the configurations of the configurable logic block as shown in Fig. 7. The configurable logic blocks at the periphery of the device also perform the I/O operations. The interconnection network can be reconfigured by changing the connections between the configurable logic blocks and the wires and by configuring the switch boxes, which connect different wires as shown in Fig. 9. The switch boxes known as *Programmable Switch Matrix* (PSM) for the interconnection network are also controlled by SRAM memory cells. The functions computed in the configurable logic block, the interconnection network and the I/O blocks can be configured using external data. A typical I/O Block design is shown in Fig. 8. FPGAs typically permit unlimited no of reconfigurations by using either serial interfaces or parallel interfaces like those provided by Xilinx series FPGAs. These versatile devices have been used to build processors and coprocessors whose internal architecture as well as the interconnections can be configured to match the needs of a running application.

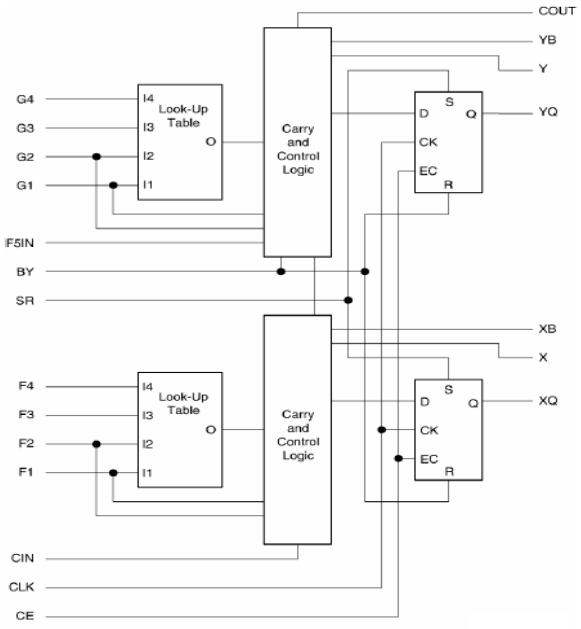


Fig. 7 Configurable Logic Block (CLB)

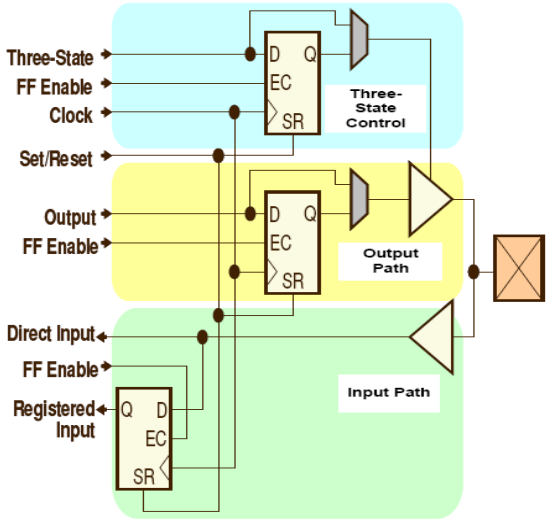


Fig. 8 Input / Output Blocks (IOB)

Current and future generation of reconfigurable devices ameliorate the reconfiguration cost by providing partial and dynamic reconfigurability [6], [7]. In partial reconfiguration, it is possible to modify the configuration of a part of the device while configuration of remaining parts is retained [12], [13]. In the dynamic reconfiguration, the devices permit this partial reconfiguration even while other logic blocks are performing computations [14], [15]. Devices in which multiple contexts of the configuration of logic block can be stored in the logic block and the context switched dynamically have also been proposed [6], [7]. In order to further increase the performance of such devices by reducing the configuration overheads, the concepts of Configuration Cloning [2], Configuration Pre-fetching [2], Configuration Context-Switching, Configuration Compression and Intelligent Configuration [3] techniques have also been proposed.

Typically, the application requirements increase at a rate faster than the increase in the size of logic resources on most FPGA devices. FPGA architectures also have limits on the I/O capability due to the limitation on the number of I/O pins on the device. To map large applications onto configurable logic, various systems have been designed which have several FPGAs on a board. These architectures also provide local memory and dedicated or programmable interconnect between the FPGAs known as *Field Programmable Interconnect* (FPIC). These board level architectures are usually designed to function under an external controller or use one of the on-board FPGAs as a controller.

IV. Performance Analysis Equation

Following is the mathematical equation being formulated for the calculations of the total no of cycles (T_{Total}) consumed by the proposed computational unit for the execution of an application. Consider the Table.1 for the analysis equation parameters.

$$T_{Total} = N_{VLIW}\,(T_{VFT} + T_{OFT}) + E_{VLIW} + \beta_{CNF}$$

Where $\beta_{CNF} = (N_{CNF} \times T_{CNF})$,
$N_{VLIW} = (N_{INST} + N_{NOP}) / 8$
$E_{VLIW} = \sum (E_{VLIW\text{-}0}, E_{VLIW\text{-}1}, E_{VLIW\text{-}2} \ldots E_{VLIW\text{-}N})$

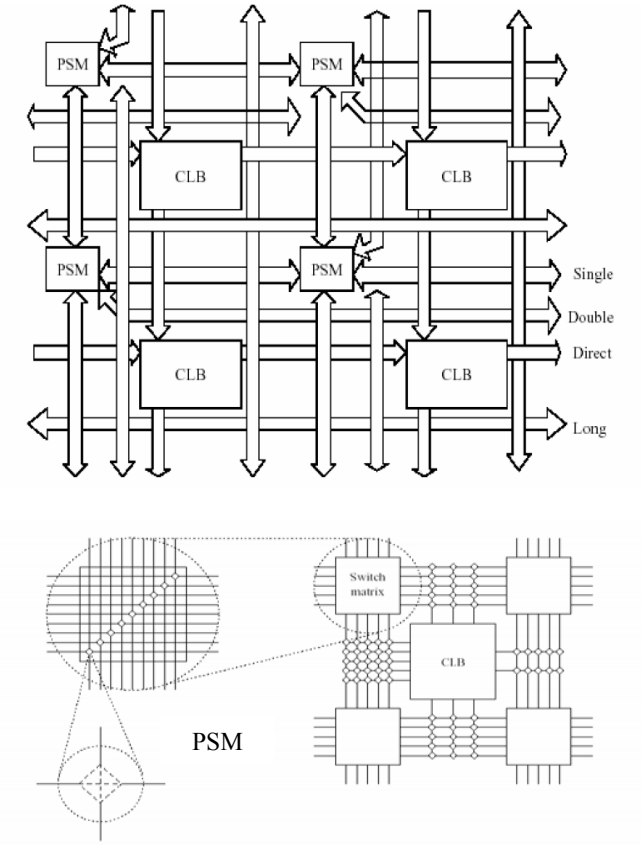


Fig. 9 Programmable Interconnect

V. Calculated Performance Gain

In order to measure and compare the performance of proposed computational unit, a reconfigurable processor (RISP) being integrated with the proposed computational unit has been analyzed and benchmarked with a typical DSP (TMS320C6X) processor by executing a variety of application programs on both of them. Performance statistics have been measured in terms of the no of clock cycles being consumed by each of them for execution. It has been observed that the segments of code of an application being under execution containing loops will be drastically boasted as shown in the graph in Fig. 10.

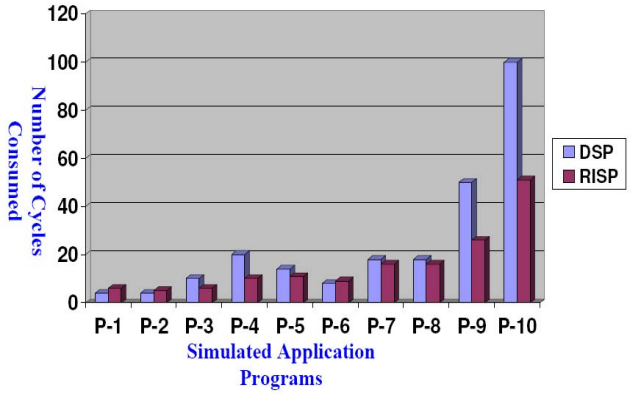


Fig. 10 Performance Using Proposed Computational Unit

VI. Comparison With Existing Systems

In this section the proposed computational unit design of the reconfigurable processor has been compared with the computational units of some of the well known reconfigurable architectures.

A. Configuration Granularity: The proposed computational unit is fine-grain architecture. There exist many systems using this approach like CHIMERAE [4]. Using fine-grain approach the system can be reconfigured at instruction level and even at operator level. But there exist many other systems which use the coarse-grain architecture and can be reconfigured at ALU level. Among them are the REMARC [5], PipeRench [8], Garp [9] and Napa [10].

B. RFU Coupling: The proposed computational unit is a tightly coupled architecture like CHIMAERA [4]. Others may use a coprocessor approach like GARP [9]. Tightly coupled designs have small configuration overheads but are suffered by dependant execution of RFUs with standard CPU core.

C. Operands Coding: The proposed computational unit is based on a *fixed operand coding* scheme. Some designs are based on the *hardwired operand coding* scheme like CHIMAERA [4] or are based on the *flexible operand coding* scheme like Napa [10.

D. Instruction Coding: The instruction of proposed computational unit is decoded such that the op-code of each instruction is being translated into the address of the concerned configuration block in the configuration memory like in CHIMAERA [4]. Other alternative is to use the op-code as an identifier to a configuration table which contains the address of the concerned configuration block in the configuration memory.

E. Program Concurrency: The proposed computational unit can execute more than one instruction (eight) at the same time.

Table.1 Analysis Equation Parameters

Parameters Description	Possible Values
Total Instructions (N_{INST})	1, 2, 3…..J Per Program
Total NOPs Used (N_{NOP})	0 - 7 Per VLIW
Total VLIWs (N_{VLIW})	1, 2, 3…..K Per Program
VLIW Fetch Time (T_{VFT})	1 Cycle Per VLIW
Operand Fetch Time (T_{OFT})	0 - 1 Cycle Per VLIW
VLIWs Execution Time (E_{VLIW})	1, 2, 3….L Cycles Per Program
Total Configuration Time (β_{CNF})	0, 1, 2…M Cycles Per Program
Total Configurations (N_{CNF})	0 - N_{VLIW} Per Program
Configuration Time (T_{CNF})	1, 2, 3…N Cycles Per VLIW

Most reconfigurable systems are only able to execute one instruction at the same time. They are based on both CISC and RISC designs. The only superscalar processor that we know of this type is the OneChip98, which is based on the superscalar version of DLX.

F. Configuration Memory: The proposed computational unit is using a multi-port configuration memory unlike the existing architectures which are so for being designed using the simple single-port configuration memory.

VII. Research Areas In Active Domain

The following topics outline the different aspects of reconfigurable computing that research has been addressing in the past several years and still there is a lot of research work required to explore new ideas that can further enhanced the performance of reconfigurable computing systems.

A. Computing Architectures

As for as the reconfigurable architectures are concerned, a variety of reconfigurable devices and system architectures have been developed which propose the various ways of organizing and interfacing the configurable logic resources. Certain architectures are based on fine-grain functional units and some are based on coarse-grain functional units. Coarse-grain architectures are configured on the fly to execute an operation from a given set of the operations. Also the commercially available reconfigurable architectures are exploring the integration of the reconfigurable logic and the microprocessors on the same chip.

B. Algorithmic Synthesis

Dynamically reconfigurable architectures give rise to new classes of problems in mapping computations onto the architectures. New algorithmic techniques are needed to schedule the computations. Existing algorithmic mapping techniques focus primarily on loops in general purpose programs. Loop structures provide repetitive computations, scope for pipelining and parallelization and are candidates for mapping to reconfigurable hardware.

C. Software Tools

Current software tools still rely on CAD based mapping techniques. But, there are several tools being developed to address run-time reconfiguration, compilation from high-level languages such as C, simulation of dynamically reconfigurable logic in software and the complete operating system for dynamically reconfigurable platforms. There is a significant lack of research in development of models of reconfigurable architectures that can be utilized for developing a formal framework for mapping applications. The Reconfigurable Mesh model was the earliest theoretical model that addressed dynamic reconfiguration in computation and communication structure. However, Reconfigurable Mesh model is more theoretical and hardware implementations have only been able to approximate the delay and speed assumptions in the model. There have been several research efforts that focused on developing architectures and the associated software tools for mapping onto their specific architecture. Some of these projects have addressed generic mapping techniques that can be extended to a class of the reconfigurable architectures. Such projects include Berkeley Garp [9], National Semiconductor NAPA [10], Northwestern MATCH [11] and CMU PipeRench [8]. Several efforts have also focused on developing the parameterized libraries and components, precision being one of the parameters. Most FPGA device vendors provide such highly optimized parameterized libraries for their architectures. Efforts have also been made to generate such modules using high-level descriptions.

D. Configuration Pipelining

Pipelined designs have been studied by several researchers in the configurable computing domain. The concept of virtual pipelines and their mapping onto physical pipelines has also been analyzed. A group has addressed some of the issues in mapping virtual pipelines onto a physical pipeline by using incremental reconfiguration in the context of PipeRench [8]. Yet another group described the pipeline morphing and virtual pipelines as an idea to reduce the reconfiguration costs. A pipeline configuration is morphed into another configuration by incrementally reconfiguring the stage by stage while computations are being performed in the remaining stages. Virtual pipelines are mapped onto physical pipelines by morphing between pipeline stages. But, morphing is limited to architectures, which support fast reconfiguration of the order of a single pipeline stage execution.

E. Simulation Tools

Several simulation tools have been developed for the reprogrammable FPGAs. Most of the tools are device based simulators and are not system level simulators. The most significant effort in this area has been the Dynamic Circuit Switching based simulation tools provided by a group of researchers. These tools study the dynamically reconfigurable behavior of FPGAs and are integrated into CAD framework. Though the simulation tools can analyze the dynamic circuit behavior of FPGAs, the tools are still low level. The simulation is based on CAD tools and requires the input design of the application to be specified in VHDL. The parameters for the design are obtained only after processing by the device specific tools. They described a visualization tool for reconfigurable libraries. They developed tools to simulate behavior and illustrate design structure. Their emphasis is on visualization of library modules and not system level simulation or application performance analysis.

VIII. Conclusion

Reconfigurable computing is becoming an important part of research in the field of digital computing. By placing the computationally intensive portions of applications onto the reconfigurable hardware, the applications can be greatly accelerated. Similar to software-only implementations, the mapped circuit is flexible, and can be changed over the lifetime of the system or even during the course of executions of the applications. Additionally, the computations mapped to the reconfigurable logic are executed in hardware, and therefore have performance similar to an ASIC. This performance stems from bypassing the fetch-decode-execute cycle of traditional microprocessors as well as allowing the parallel execution of multiple operations.

The proposed RFU based computational unit provides us a great performance parameter over the traditional processor computational units. In the proposed computational unit the hardware changes according to requirements of applications being executing. Hence the system follows the strategy of, the demand-driven hardware should be swapped in and the unused hardware should be swapped out and hence providing more hardware than that available in the system during the execution of the applications. This reconfiguration of the hardware does not stop the application being under execution. Due to the partial reconfiguration of device, the time overheads required to reconfigure device are compensated because the application keeps continue its operation during the reconfiguration of the device. RFU based computational units are very suitable for those applications where different kinds of processing units are frequently required to boast up the performance of executing application.

REFERENCES

[1] M. Aqeel Iqbal and Uzma Saeed Awan, 'Reconfigurable Instruction Set Processor Design Using Software Based Configuration', *Proceedings of IEEE computer society, IEEE* International Conference on Advanced Computer Theory and Engineering 2008 (ICACTE-2008), *December 20-22, 2008, Phuket Island, Thailand.*

[2] M. Aqeel Iqbal, Shoab A. Khan and Uzma Saeed Awan, 'RISP Design with Most Optimal Configuration Overhead for VLIW Based Architectures', *Proceedings of IEEE computer society, 2nd IEEE International Conference on Electrical Engineering 2008 (ICEE-2008), March 25-26, 2008, UET Lahore, Pakistan.*

[3] Aziz-Ur-Rehman, Dr. Aqeel A. Syed and M. Aqeel Iqbal, 'Intelligent Reconfigurable Instruction Set Processor (IRISP) Design', *Proceedings of IEEE computer society, 11th IEEE* International Multi-topic Conference 2007 (*INMIC-2007), Dec 28-30, 2007, COMSATS Lahore, Pakistan.*

[4] Ye, Z. A., Moshovos, A., Hauck, S., and Banerjee, P., "CHIMAERA: A High-Performance Architecture With a Tightly-Coupled Reconfigurable Functional Unit," Proceedings of the 27th International Symposium on Computer Architecture, pp. 225-235, 2000.

[5] Miyamori, T. and Olukotun, K., REMARC: Reconfigurable Multimedia Array Coprocessor IEICE Transactions on Information and Systems E82-D, vol. pp. 389-397, Feb, 1999.

[6] Xilinx, Virtex Series FPGAs, http://www.xilinx.com, 2001.

[7] Xilinx, Inc. Virtex II Configuration Architecture Advanced Users' Guide. March, 2000.

[8] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer. "PipeRench: A Coprocessor for Streaming Multimedia Acceleration", in Proc. Intl. Symp. on Computer Architecture, May 1999.

[9] Hauser, J. R. and Wawrzynek, J., "Garp: A MIPS Processor With a Reconfigurable Coprocessor," IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 12-21, 1997.

[10] C. Rupp, M. Landguth, T. Garverick, E. Gomersall, H.Holt, J.Arnold and M. Gokhale, "The NAPA Adaptive Processing Architecture", IEEE Symposium on FPGAs for Custom Computing Machines, Apr. 1998.

[11] Altera Inc.. Altera Mega Core Functions, San Jose, CA, 1999. http://www.altera.com/html/tools/megacore.htm

[12] Philip James-Roxby and Steven A. Guccione. Automated Extraction of Run-Time Parameterisable Cores from Programmable Device Configurations. In Proceedings of IEEE Workshop on Field Programmable Custom Computing Machines, pages 153-161, April 2000.

[13] Edson L. Horta and John W. Lockwood. PARBIT: A Tool to Transform Bitfiles to Implement Partial Reconfiguration of Field Programmable Gate Arrays (FPGAs). Washington University Department of Computer Science Technical Report WUCS-01-13. July 2001. (Available at http://www.arl.wustl.edu/arl/projects/fpx/parbit

[14] S. McMillan and S. Guccione, "Partial run-time reconfiguration using JRTR," in Field-Programmable Logic and Applications / The Roadmap to Reconfigurable Computing (FPL'2000), (Villach, Austria), pp. 352–360, Aug. 2000.

[15] X. Inc., "Configuration and readback of Virtex FPGAs using (JTAG) boundary scan." Xilinx XAPP139, Feb. 2000.