IOSR Journal of Engineering (IOSRJEN)
e-ISSN: 2250-3021, p-ISSN: 2278-8719, www.iosrjen.org
Volume 2, Issue 9 (September 2012), PP 32-37

Intelligent Reconfigurable Computing Systems

Navreet Kaur Rana

Abstract: *The FPGAs have come up with promising rewards in the field of reconfigurable Computing. But if the process of reconfiguration is automated, by following a rule based decision making approach, it can provide more optimal solutions for the issues faced during reconfiguration. A similar idea has given rise to INTELLIGENT RECONFIGURABLE COMPUTINGSYSTEMS.*

Keywords: *Field Programmable Logic Array, Rule based decision making, Reconfigurable computing*

I. INTRODUCTION

Every new technology comes with a scope of research on making it more economical in terms of money and time. Intelligent Reconfigurable computing systems follow a similar trend by implementing the applications over the available reconfigurable hardware. Reconfigurability basically aims at reuse of the hardware for similar problems or sub problems. Along with that, these systems aim at making them respond intelligently by interrogating the availability (or non availability) of different instances of resources. The use of Field Programmable Gate Arrays , their existence with or without processor, the unit of hardware that can be reconfigured, and the possible communication between the processor, the memory unit and the reconfigurable hardware makes it possible for such systems to exist.

The challenges that have been tried to overcome are the structure of the fabric, provision of possible interface between the different logic blocks, and the mapping up of application on the device. These three goals have been discussed one by one along with other responsible factors such as the granularity of the reconfiguration, the size of application and the time of configuration.

This paper majorly discusses about the use of FPGAs (Field Programmable Gate Arrays) for reconfiguration and scrutinizing the availability of resources using intelligent decision making process.

Before FPGAs, came PLAs, PLDs and CPLDs for reprogramming for specific functions. But these devices are smaller in size and are not suitable for reconfiguration of larger units. FPGAs, on the other hand are larger and more complex.

II. ARCHITECTURE OF RECONFIGURABLE HARDWARE

According to the arrangement of logic blocks and the interconnections among different logic blocks FPAGs are classified as follows—

2.1 ROW BASED-

The logic blocks are arranged in rows. The spacing between rows is used for communication among the logic blocks. The diagram below shows the Row Based FPGA.

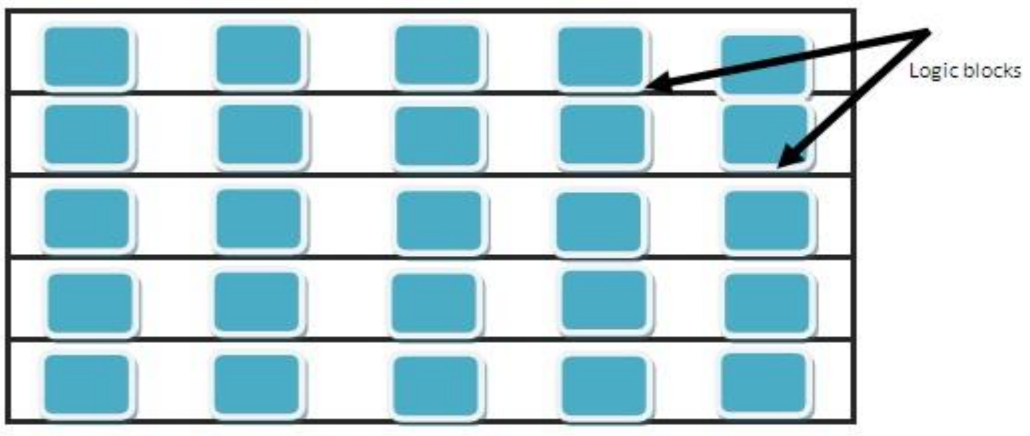

Fig 1 Row Based FPGA

2.2 SYMMETRICAL ARRAY

It has a two dimension array of logic blocks placed as vertical and horizontal lines. Communication among logic components happens with the help of elements at the intersection of the vertical and horizontal lines.

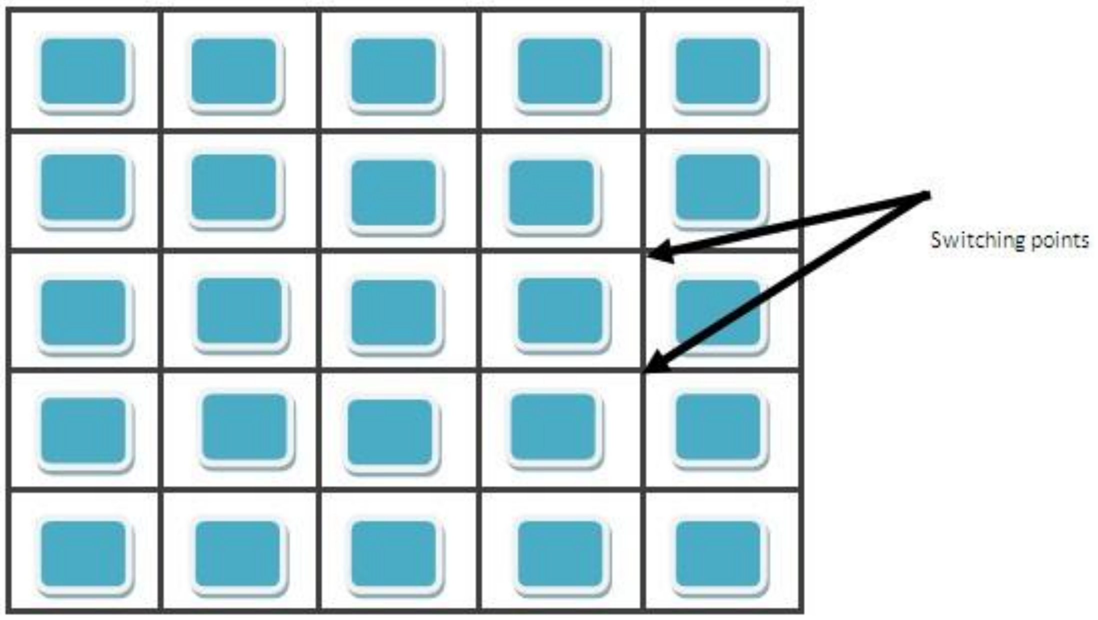

Fig 2 Symmetrical Array

2.3 SEA OF GATES

It is same as that of symmetrical array. It again is a two-dimension matrix like structure. The difference lies in the fact that there is no space left alongside the logic blocks for routing to take place.
The interconnections are made on top of the blocks using wires.

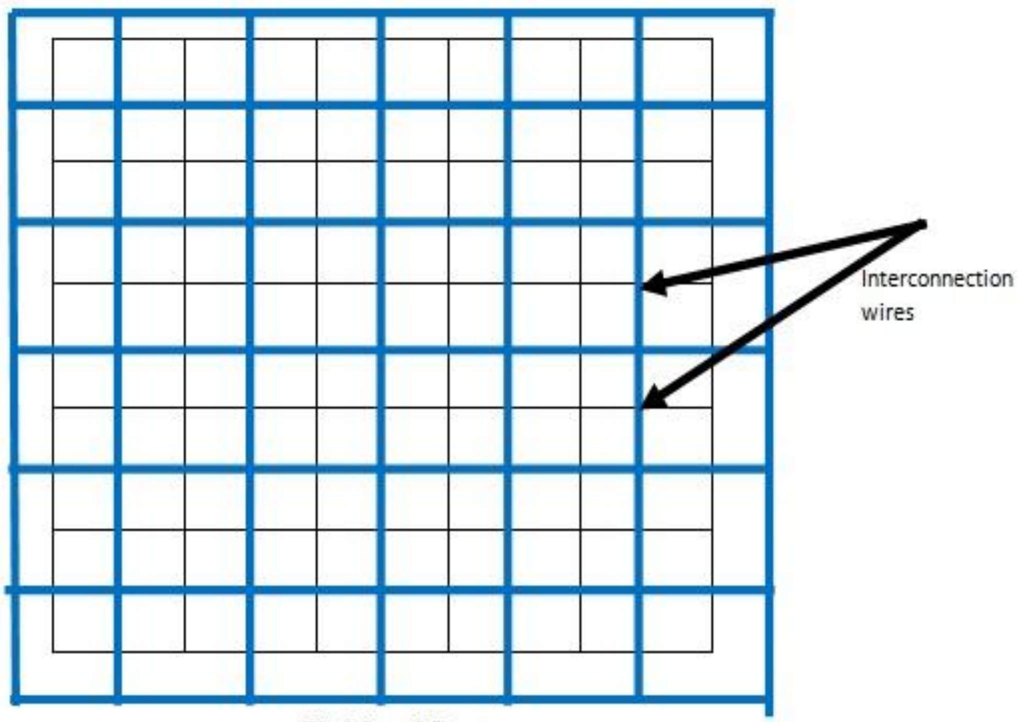

Fig 3 Sea of Gates

2.4 HIERARCHICAL FPGA

Logic blocks are placed hierarchically on the device. Elements with the lowest granularity are at the lowest level in hierarchy. They are grouped together to form the element of the next level. An element at level L comprises of elements from level L1.

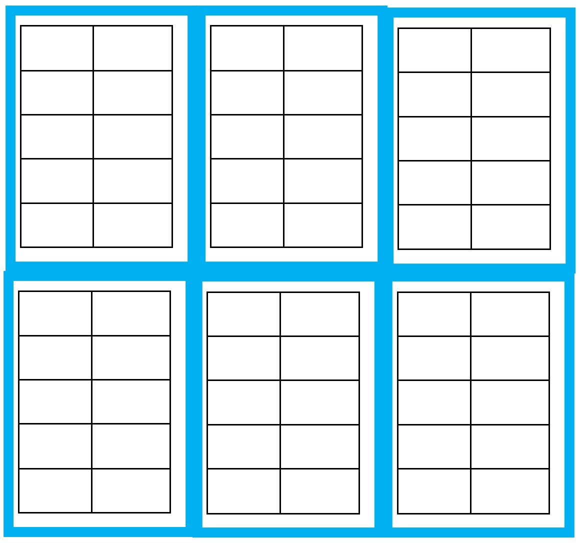

Fig 4 Hierarchical FPGA

III. ISSUES RELATED TO RECONFIGURABILITY

3.1 GRANULARITY

It is important to discuss the concept of granularity now. It is significant to analyze the units of data to be made reconfigurable. The amount in this case refers to the number of cells that are required to be reconfigured. Granularity ranges from **fine grained architecture to coarse grained architecture**.
Fine grained architecture limits the data manipulation of data at bit level.
In case, the manipulation data are larger than the bit, it comes under the coarse grained architecture.
The example of coarse grained architecture –if an ALU is reconfigured it falls in the category of coarse grained architecture.

3.2 SIZE OF APPLICATION

Size of application too plays an important role in deciding the implementation and solution to a problem. The underlying requirement is the need to know whether an application is to be implemented by a microprocessor component or a coprocessor component.
Thus all applications to be implemented can be as follows-

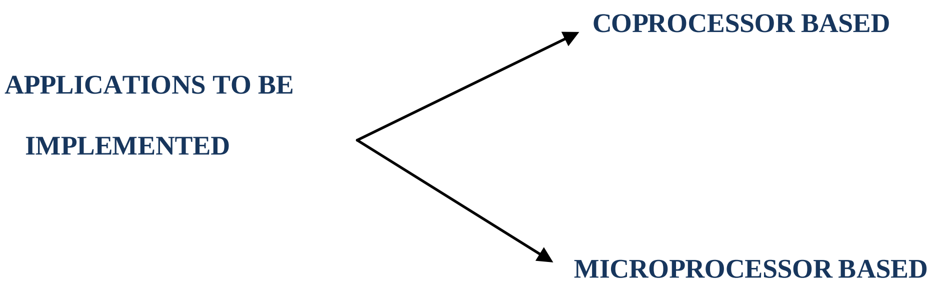

Fig 5 Categorization of Problems

Any problem (belonging to either of the categories above) must be partitioned first, since the resources available to devices are limited in nature and thus give rise to the concept of reconfigurability to make systems more efficient.

IV. FULL AND PARTIAL RECONFIGURATION

Before reconfiguration, it is important to recognize which parts of the problem can be or should be reconfigured.

The entire problem need not be reconfigured. Only these problems which have similar instances of sub problems are efficiently reconfigured. Keeping this in mind, the problems are classified as FULL RECONFIGURATION and PARTIAL CONFIGURATION.

Once the application is partitioned, the partitioned components are placed on the reconfigurable hardware and the available resources are to be managed.

The device has both **possible and impossible placement regions** on it for the modules to be placed on them.

A region is said to be an impossible region because it is not possible to place a new component on a reconfigurable device since it is already occupied.

The possible placement region is the area after removing the impossible region from the total area.

V. WHERE CAN RECONFIGURATION BE APPLIED?

It is important to analyse that to which kind of problems we can apply reconfiguration.

Not all problems are suitable for such kind of implementation. Problems which are comprised of similar sub problems are best suited for such implementations. This means the problems which run same functioning for different instances can give the best optimized results if implemented using reconfigurable systems.

For example, if same processing is to be applied to all the elements of a data structure (such as an array), reconfigurable implementation is worth it.

VI. RECONFIGURABLE COMPUTING SYSTEMS AND INTELLIGENCE

Tasks have to be scheduled and the placement has to be time dependent. Since the resources are limited, the tasks that are partitioned are scheduled thereafter so as to decide which task follows the other. The reason for doing this is that the output of one component could possibly be an input to the other component function.

This is the time when reconfigurable systems can be made .Such systems can be made to behave intelligently at the time when the partitions of the computing problem are mapped onto the reconfigurable devices. The intelligent behavior can be applied using a **rule based approach**.

Rules can be very well developed in a language such as Prolog which itself is rule based.

Before formulating the rules, one must have a complete knowledge of all the logic blocks that can be reconfigured. If a set of rules is formed, followed by their inference and their conclusion it is possible to keep track of the instances of resources that are free or allocated. If the logic block is already allocated to a function, it is marked as not free. But as soon as it becomes free, its status in the table is updated and it is marked as free.

For example rules can be of the form:

- if the block is NOT ALLOCATED, it is FREE,
- If the block is NOT FREE, it is ALLOCATED,
- If the block is ALLOCATED, specify the name of the module it is allocated to,

These rules when implemented on the given resources, brings PROCESSED information about several instances of resources into picture. This information can be brought to use over and over again by updating it each time before configuring any of the logic blocks again.

Following is a small practical example implementing the rule based approach. The example has been devised in Prolog.

Let us assume that that B1, B2 and B3 are three instances of logic blocks on reconfigurable fabric and P1, P2 and P3 are three instances of partitioned problems. All the instances of blocks and problems are taken to be in the form of a *list.*

We give the following information in the database-

```
IsAllocatedTo (B1, P1).
IsAllocatedTo (B2, P2).
```

which signifies that logic block B1 is assigned to problem P1 and so on?

```
1) Block_Free(X) :- not(IsAllocatedTo(X,[H| _ ]),
                    Not(IsAllocatedTo(X,[ _ |T]).
```

Means a logic block is free is it is not allocated to any of the problems.

2) Implemented(X) :- IsAllocatedTo ([H| _],X), !, IsAllocatedTo ([_ |T],X).
   Tells whether a particular problem is allocated to a block or not.

3) Get_free_blocks :- not(IsAllocatedTo (Block, _)) , write(Block), n1,fail.
   Writes all the free blocks available.

The instances of queries that can be made are as follows:-

1) ? Block_Free(B3)
   Yes

2) ? Block_Free(B1)
   No
3) ?IsAllocatedTo (B1,X)
   X is P1

4) ?Implemented(P2)
   Yes

Thus we see by formulating such rules we can automate the process of finding a free logic block and placing the problem on it.
Definitely it will be required to analyze if a particular problem is configurable on a specific device but the criteria for placement of problems on reconfigurable device will be needed to be validated only for free blocks which can be found from above table created.

VII. PROCESSING ENVIRONMENT

A full reconfigurable system may have the essential architecture such as memory and processor within itself but there are other systems which do not form a self-sufficient system and interact with the processer whenever the need be.
One form which systems can take is the independent systems with co-processor architecture. They are well suited for the problems which are fully reconfigurable. All the scheduling and pipelining of the process instructions can be done within the system only. In a way, these systems form an ensemble with all the necessary hardware so that they can work independently.
The other possibility is to have a separate processor. The processor may be tightly or loosely coupled with the reconfigurable architecture.
If tightly coupled, it facilitates the communication and the data transfer but may put a restriction on the freedom of reconfigurable hardware.
Loosely coupled processor on the other hand, is well suited both for the independence of reconfigurable hardware and the processor.

VIII. SCHEDULING

Since the problem is partitioned, it is a pre requisite to schedule the parts of the problem. Scheduling is necessary to know which part will be configured first and followed by which part. This is required to allocate the necessary resources to logic blocks during reconfiguration.
Scheduling is be done under following scenarios-

8.1 On demand scheduling

It is done when the need to schedule tasks arises. The request is made and the parts of the problem are scheduled. The decision making process discussed above can be brought into use to schedule problems.

8.2 Static scheduling

This is done when prior to execution the problem and the sub parts of the problem are analyzed thoroughly and then the problem is scheduled. Analyzing the problem a priori reduces the overhead during reconfiguration.

8.3 Dynamic scheduling

Under this scheme, run time information is used to schedule the tasks. This process is dependent on the characteristics of a problem and the scheduling is done after the problem comes into visibility.

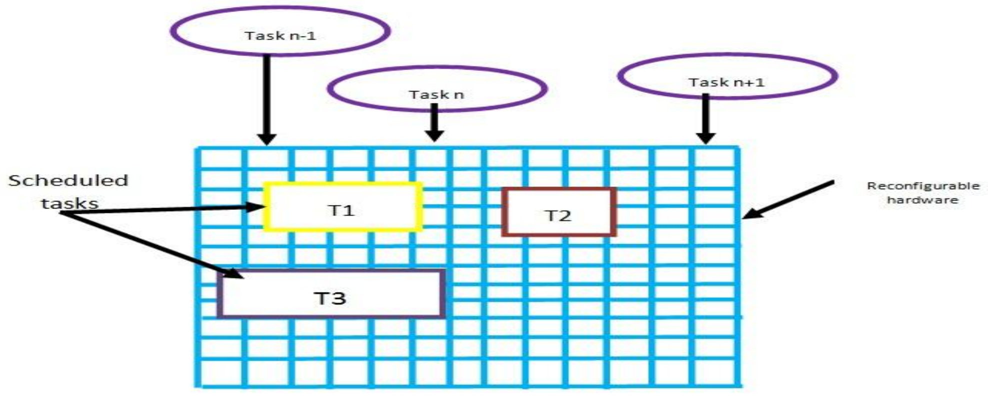

Fig 6 Scheduling of tasks

IX. APPLICATIONS

8.4 CRYPTOGRAPHY- If same encryption technique is used, for all the inputs,
It means same piece of code runs repeatedly for different inputs. Such problems can be encrypted using reconfiguration. Same applies to decryption also.

8.5 IMAGE PROCESSING- At times, image processing is done with the help of fractals and fractals are nothing but structures of self repeating units.

Apart from these two applications, Intelligent Reconfigurable Computing Systems find their use in many other fields of technology.

X. LIMITATIONS

Despite such vivid applications, these intelligent systems are not free from limitations. There is no well defined boundary to determine whether a particular application should be implemented using reconfigurable architecture or not. Secondly, the partitioning of the problem into several reconfigurable and non reconfigurable parts is yet another issue. Thirdly, rules if not very well defined, may not be able to fulfill the purpose of their existence.

XI. Conclusion

Reconfiguration itself gives efficiency to programs but the use of same hardware for similar problems gives rise to complex issues. The problem of scheduling the tasks, the availability of memory and the processor for the hardware etc are the key facts that have been discussed briefly in the paper.

A novel idea of making systems behave intelligently by following a rule based approach is implemented which works in the direction of automating them.

Though further improvements are required, reconfigurable systems totally go hand in hand with the principle **of enhancing performance and efficiency** which forms the base of existence of technology**.**

REFERENCES

Journal Papers:

[1] Ulle Endriss, Lecture Notes An Introduction to Prolog Programming, Institute for Logic, Language and Computation

Books:

[1] Scott Hauck and Andr´e DeHon, Reconfigurable Computing-The Theory and Practice Of FPGA Based Computation.

[2] Christophe Bobda *University of Kaiserslautern, Germany*, Introduction to Reconfigurable Computing Architectures, Algorithms, and Applications