ISSN (Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

Volume 3, Special Issue 3, March 2014

2014 International Conference on Innovations in Engineering and Technology (ICIET'14)
On 21st & 22nd March Organized by

K.L.N. College of Engineering, Madurai, Tamil Nadu, India

Finite State Machine Based Reconfigurable Architecture For Image Processor

-#1J.Kanimozhi, *2Konda Abinaya Chandrasekaran, #3A B Abhinayapriya
Department Of ECE Department , KLN College of Information Technology, Tamilnadu, India.
Department Of ECE Department , KLN College of Information Technology, Tamilnadu, India.
Department Of ECE Department , KLN College of Information Technology, Tamilnadu, India.

ABSTRACT—Adaptively evolvable systems require some reconfigurable capabilities, habituated within the FPGA. FPGA's can be integrated with either Dynamic Partial Reconfiguration (DPR) or Virtual Reconfiguration Circuits (VRC). Since DPR allows lesser resource utilization on the FPGA available logic, which has a positive repercussion in power consumption compared to VRC's, we are using FPGA's with native DPR as our reconfigurable hardware. Evolvable hardware system has been employed to ease automation, in which the reconfiguration is driven by an evolutionary algorithm. Choosing evolutionary algorithm for DPR over the optimization algorithms is because these algorithms are incremental, and hence the processing circuit will be affected only by small changes reducing the reconfiguration times. One of the problems of the evolvable systems is that, they have to be trained to the conditions in which the system will operate. The generation of this training data is done offline, reducing the system's autonomy. We have proposed the enhancement of this evolvable system by injecting the concept of Finite State Machine which reduces its dependence. Considering that our system concentrates on adaptive image processing, we have utilized the Self Tuning Architecture (STA) based on FSM and combined them with the evolutionary algorithm, to implement the image processor.

KEYWORDS—Evolvable Hardware, FSM, Evolutionary Algorithm, Dynamic Partial Reconfiguration**.**

I. INTRODUCTION

Evolvable Hardware is a very vast domain, which requires complex circuit designs for specific applications. To make a versatile device, these complicated circuits have to change according to the need and the environment in which they have to work. For this purpose, we require an algorithm that is capable of finding an optimal solution to the given problem. These systems, on being evolvable, must be initially made adaptable by training them with a set of data, giving the conditions in which our device is going to work. But, this set of data is usually generated offline, which reduces the systems autonomy. To make the system automated, the algorithm used here is the Finite State Machine based Evolutionary Algorithm (EA).

An example of this Evolvable Hardware is a Digital Image Processor, which removes the noise from the image, and performs its processing. This task is quite complicated requiring various modules, which may not be known during the hardware design. Hence we have to design a system that is capable of reconfiguring during the run-time, so as to develop a self-evolvable architecture. An optimization algorithm used to design this reconfigurable hardware is the FSM based Evolutionary algorithm. This work is a continuation of [4] where a self-reconfigurable architecture was designed using Evolutionary algorithm. Here we have enhanced the same using the concept of Finite State Machine to improve the Speed of processing.

Now since our processor has to be designed for various purposes, we are using general purpose graphics processing unit (GPGPU) which has parallel computational capabilities. We can also design a customized hardware that can create a more efficient system for a particular purpose. This can be done by employing application-specific integrated circuits (ASICs). But, these ASICs can only be designed once, for a static purpose which cannot be modified after manufacturing. Hence we go on for Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays (FPGAs), which contain logic

gates that can be arbitrarily configured and interconnected to implement the digital circuits. FPGA provides higher configuration capability, which allows embedding of a complete system on it, providing a System on Programmable Chip (SoPC). FPGA allows massively parallel hardware to be implemented, along with a small processor for controlling the overall process. This accelerates the computational tasks and develops a completely autonomous system.

Here we have used the Dynamic Partial Reconfiguration technique to reconfigure our image processor. The processors parameters are capable of being modified in on-line mode (during runtime) only by utilizing DPR.

We can use DPR for:

- reusability of resources for modifying the processing functions
- a trade-off between resource consumption and accuracy
- enhance the performance by increasing cores [1].

Moreover, DPR has been found to be better than the Virtual Reconfiguration Circuits (VRC) by Sekanina [2].

II. RELATED WORKS

In contrast to conventional hardware where the structure is irreversibly fixed in the design process, evolvable hardware (EHW) is designed to adapt to changes in task requirements or changes in the environment, through its ability to reconfigure its own hardware structure dynamically and autonomously. This paper introduces EHW chips and six applications developed as a part of Ministry of International Trade and Industries (MITI's) Real – World computing project; an analog EHW chip for cellular phones, a clock-timing architecture for Gigahertz systems, a neural network EHW chip capable of autonomous reconfiguration, a data compression EHW chip for electro-photographic printers, and gate level EHW chip used in robotic navigation and prosthetic hands. This capacity for adaptation, achieved by employing efficient search algorithms based on the metaphor of evolution, has great potential for the development of innovative industrial applications [3].

Dynamic Partial Reconfiguration (DPR) is used for the purpose of automation without any human interruption. This feature makes it possible to use FPGA with smaller configuration memory, so it will reduce the power consumption. In DPR, while run-time, different configurations were sent via the configuration access port to the configuration memory DPR allows us to control number of cores (NC), number of input (NI) bits and number of output (NO) bits content [1]. Using DPR in developing a test system such as this is definitely a viable option; however, the following drawbacks need to be understood prior to determining whether DPR is a good fit for the target application [11].

Reconfigurable hardware is gaining a steadily growing interest in the domain of space applications. This hardware allows for changing or adapting payload processing during the flight mission. Reconfiguration is done while a new image arrives inside the image processor [10]. The genetic operators of a genetic algorithm in a VLSI layout design should be adapted to the specific layout problem rather than selecting an unnatural representation that would allow the use of traditional genetic operators. Hence using of genetic algorithm as the evolutionary algorithm gives the optimal solution for selection of the processing core. Enhancing this evolutionary algorithm with the introduction of Finite State Machines would further reduce the logic required, thereby reducing the area and time required.

An FSM is called self-reconfigurable if reconfiguration of either output function or transition function is initiated by the FSM itself and not based on external reconfiguration events. Here our transition function for the purpose of routing in the Mesh topology uses the concept of FSM thereby supporting the reconfiguration.

III. PROPOSED SYSTEM

A. Evolutionary Algorithm

Evolutionary algorithms are optimization algorithms based on trial and error and random incremental search. These algorithms are inspired by the natural process of evolution that allows different species of living beings to appear and adapt to the different conditions of the environment. Natural evolution achieves this with a mechanism known as natural selection: each individual in a species generates a certain number of copies of itself during its lifespan. These are not exact copies; they have slight alterations that make them behave differently, but have similar characteristics. The longer an individual lives, the more copies it can make. Although this is affected by a huge number of random factors, the ability of an individual to live longer and make more copies of itself, which will also be able to live longer and make more copies of themselves, creates an unbalance in this random propagation that will result in more desirable characteristics to be perpetuated while less desirable ones will eventually disappear. This is often referred to as survival of the fittest.

An interesting aspect of natural evolution is that it does not require any control or explicit specification of techniques required to survive: individuals in a population will simply adapt to the conditions of their environment, and new individuals with new characteristics will appear naturally, without needing an intentional creation of these characteristics. With the same philosophy, evolutionary algorithms aim to find solutions to a certain problem without the developer to explicitly indicate how these solutions should be, only supplying a method to compare solutions. The evolutionary algorithm will generate random solutions to the problem, evaluate them, compare them, and generate new solutions by picking the best solutions and applying small random changes on them.

In order to compare different solutions, a method known as fitness function is supplied. This is a function that maps an individual to a numeric value known as fitness based on its performance: individuals that perform better will have a higher (or lower) fitness than those with worse performance. For example, in the case of the implementation of a discrete filter, the evaluation of the fitness function could consist in filtering a known training pattern and comparing the result sample by sample with a golden reference, returning a numeric value such as the sum of absolute errors (SAE) or the peak signal-to-noise ratio (PSNR). In this case, the fitter individuals would be those with lower SAE or higher PSNR.

The general form of an evolutionary algorithm is:

- Generate a certain number of random solutions (the initial population)
- Evaluate those solutions in order to calculate their fitness
- Until a certain condition is met (e.g. a number of iterations has been reached, or a good enough result has been achieved)
- Choose some of the solutions according to their fitness (and possibly some other factors such as a random selection)
- Generate new solutions by applying certain operations to the selected ones
- Evaluate the new solutions
- Remove some of the solutions according to their fitness (and possibly some other factors such as age or a random selection)
- Once the condition is met, return the best solution that has been obtained during the evolution.

A common type of evolutionary algorithm is the genetic algorithm. In a genetic algorithm, each solution (phenotype) is unambiguously represented by a genotype, which is a representation of the variable characteristics of a solution in the form of a sequence of values known as genes (usually bits, but sometimes other data types are used, such as integers or reals). The set of all possible genotypes is known as the genotype space.

In a genetic algorithm, the evolution is performed on the genotypes, which are then implemented in order to evaluate the fitness of the corresponding phenotypes. The process to generate new genotypes involves two operations known as genetic operators: (1) Crossover: Recombination of the genes of two (or more) genotypes, usually by cutting both genotypes at the same random point (the crossover point), and generating a genotype formed by the first part of the first genotype and the second part of the second genotype. This is known as one-point crossover. (2) Mutation: Modification of a genotype by randomly choosing a certain number of genes and changing them, either by flipping them (in the case of bits), adding or subtracting a random value (in the case of reals), or replacing them with a random value. The number of genes that is changed is known as mutation rate.

Genetic algorithms usually perform a crossover over two genotypes and then apply a mutation to the result, although the crossover step can be omitted for simplified genetic algorithms.

B. Dynamic Partial Reconfiguration

The Reconfigurable hardware is a design approach based on a digital circuit whose functionality can be changed. One of the ways to achieve this is using FPGAs which are often used for digital circuit prototyping and testing, but due to their versatility, they are starting to be used for some applications as part of a final product. This way, the product will not only be able to update or change its internal software or firmware, but also improve or modify its hardware capabilities.

Some FPGAs go one step further, and are able to set the configuration of part of the FPGA from the FPGA itself while the rest of it continues working, which allows the development of autonomous reconfigurable systems that do not rely on an external device to change their functionality. This is known as autonomous dynamic partial reconfiguration of hardware (DPR). Circuits with such capability are said to be self-reconfigurable, which is a desirable condition or prerequisite for evolvable hardware.

The main advantage of being able to reconfigure hardware and not only software is that, whereas a software program can be made very big and complex, it is usually limited by the fact that its execution is sequential, and thus a big program will take a long time to execute. In addition, executing a program on a processor has an overhead due to the time spent decoding the instructions, fetching the variables, and storing the results. Hardware, on the other hand, can be parallelized, which means that several operations can be performed at the same time, with digital circuits tailored for a specific application rather than a general purpose one, potentially resulting on very shorter execution times, especially when a huge amount of operations has to be performed.

The problem with hardware is that it is often a static design with a fixed functionality. Reconfigurable hardware addresses this problem by letting the system change its functionality by implementing several circuits and allowing it to choose the output of one of these circuits. This can be achieved by implementing all the circuits and selecting one of the outputs through a multiplexor. This is done in the arithmetic logic units (ALUs) that can be found on microprocessors, and some implementations of evolvable hardware such as the virtual reconfigurable circuits (VRCs). Rather than this, using DPR to implement reconfigurable hardware has several advantages. First, implementing a circuit for a single functionality and replace it with another circuit using DPR rather than having all possible circuits synthesized at once implies less resource usage on the FPGA available logic, which may also have a positive repercussion in power consumption. Second, including a multiplexor in the circuit involves increasing the latency, which means

that the circuit will have to be more segmented or work at a lower frequency, so it is an advantage to get rid of it.

It has to be noted, however, that DPR has a time overhead when the reconfiguration is performed, but this is not a problem if said reconfiguration is not often performed, and allows the final circuit to be faster once the reconfiguration process has finished.

C. FPGA for Reconfigurable Hardware

The problem Modern Xilinx FPGAs provide an internal port that can be used to access and modify the FPGA configuration. This port is called the Internal Configuration Access Port (ICAP). This port is used to send and read configuration commands and settings, and to read and write configuration data. The reconfiguration process of a Xilinx FPGA is carried out by sending a sequence of commands and data through the ICAP port. Without going into detail, the steps that take place when performing a reconfiguration, either total or partial, are: (1) Configuration setup, in which several configuration options are set and checked by sending commands through the ICAP port. One of these options is the address from where data will start being written; (2) Configuration data loading, in which data for the FPGA configuration memory is written to the ICAP. (3) Configuration end, in which a checksum of the written data is optionally checked, and the FPGA is set to start working.

All these instructions and data are contained in the bitstream files used to configure the FPGA. In particular, the configuration data segment found in these files can be used for extracting configuration data for a partial circuit (a partial bitstream).

In order to perform autonomous dynamic partial reconfiguration, Xilinx provides a peripheral, the Hardware ICAP (HWICAP), which allows communication with the ICAP port directly through the processor (usually a MicroBlaze soft-core processor). However, this is only a low level interface, requiring the developer to implement all the configuration options manually. Moreover, it requires the configuration words to be sent one by one by the processor, which can take a long time for sending large amounts of configuration data.

D. Finite State Machine

A finite-state machine, or FSM for short, is a model of computation based on a hypothetical machine made of one or more states. Only a single state can be active at the same time, so the machine must transition from one state to another in order to perform different actions. FSMs are commonly used to organize and represent an execution flow.

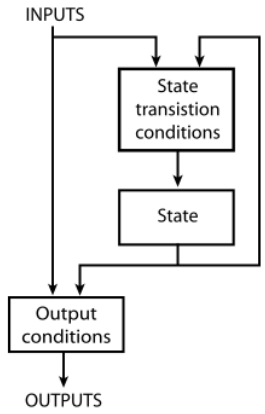

Fig 1: Finite State Machine Flow

In our Finite State Machine, we will define all the states our hardware can occupy and at any one time, our data will be in any one state. Our state machine is designed by these steps: (1) initially our system hardware implementing the state machine is designed; (2) the digital logic to implement the finite state machine; (3) assigning a unique binary value to each of the state that the machine can be in; (4) Designing our hardware so that its logic converts from current state and current input values to the correct next state values and stores that value; (5) Generating the outputs of the state machine by using combinational logic.

IV. ARCHITECTURE USED

The Architecture used by us is shown below:

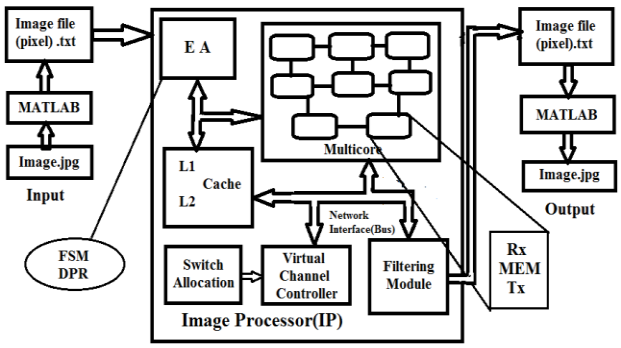

Fig 2: Block diagram of Image Processor

Initially the image is sent in as a 256*256 file into the matlab code. This converts our given image into a text file which has all the pixels as a hexadecimals. This is shown as

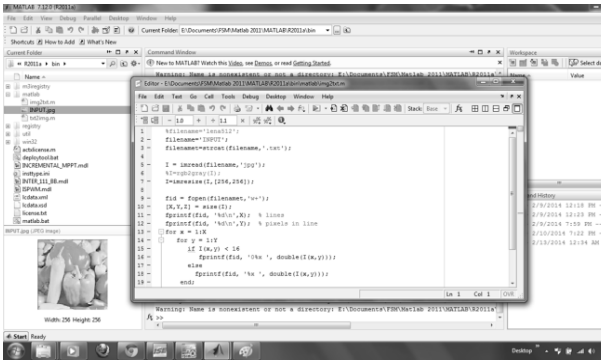

Fig 3: Image to Pixel conversion – coding

The output that is obtained can be seen as a text file, which is sent as the input to the image processor.

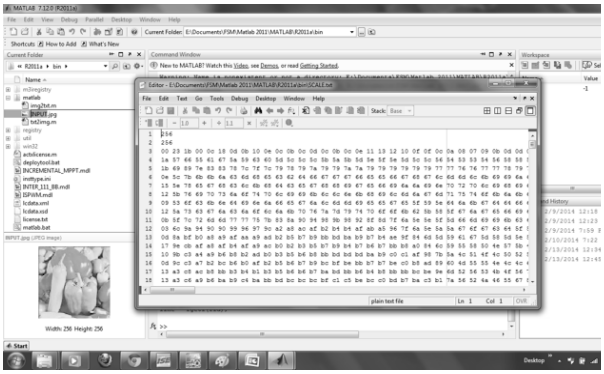

Fig 4: Generated text file which is the input to the image processor

This processor has the FSM based Evolutionary Algorithm which first checks in the pixel density of the input image and adapts the processor to the required environment. Depending upon the pixel density of the image, image processor(IP) will select the number of core. This allows Dynamic Partial Reconfiguration of the processor whenever necessary. Each unit of the multi-core processor is made of 3 units: Receiver which fetches the data in, a memory that stores each bit of data, and a transmitter which allows the transmission of data. The memory consists of 16 bits as it has to store in a hexadecimal value. L1 L2 Cache memory is used for initializing the image processor, it has the programs for every other module used. The Virtual Channel Controller is used for creating signals and paths within the core while the switch allocation unit is used to give direction to the paths and allocate pins for the processor dynamically during reconfiguration.

The Network Interface Bus is used to connect all the devices together. A SRAM memory is used externally in the processor which stores all the input data before processing.

The flow of the data from input to output is shown as

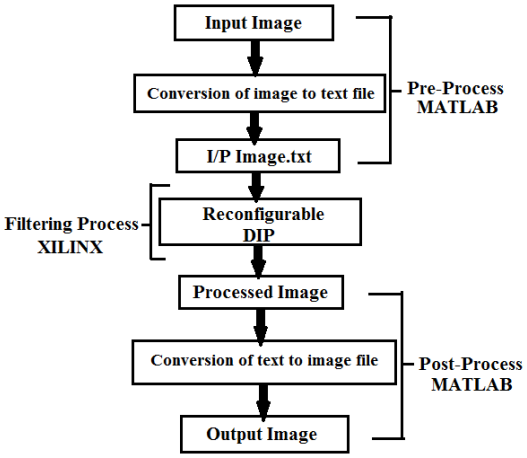

Fig 5: Flow of the Process

V. OBSERVED RESULTS

The output of our completed module has been observed. It is seen that there is an increase in speed of processing as the simplified logic for reconfiguration reduces the time consumption. This further reduces the area and power consumed by the hardware. This can be seen in core module as the time consumed is 4.881ns and the total memory used is 247.33Megabytes. The power consumed is 0.039W.

The simulated output of the existing and proposed system is shown in Fig.6 and Fig.7.The functionalities of both the systems are the same, but its processing parameters vary.

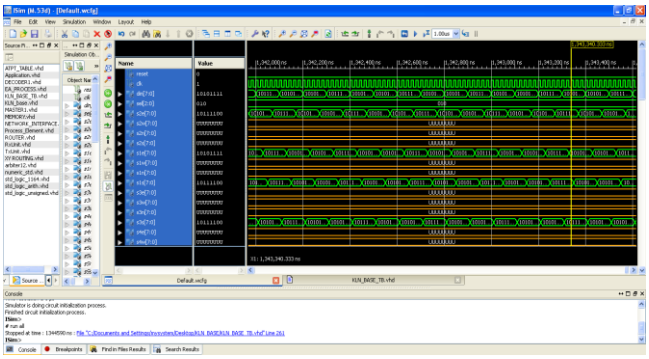

Fig 6: Simulated output of the existing system

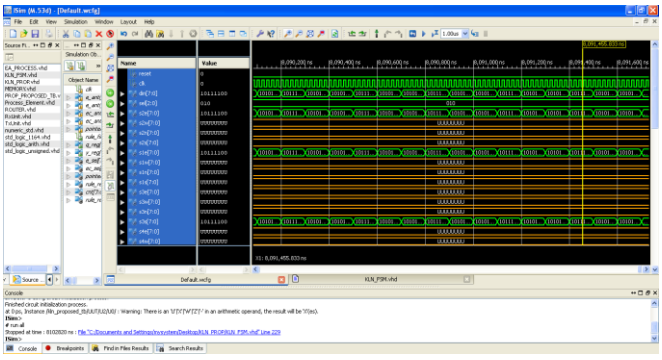

Fig 7: Simulated output of the proposed system

The observed memory and power of our proposed system is shown in Fig.8 and Fig.9, the obtained values are 147600kb and 0.039kW respectively.

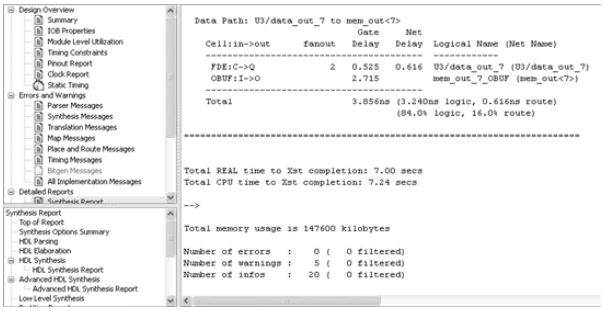

Fig 8: Memory used by the FSM based image processor

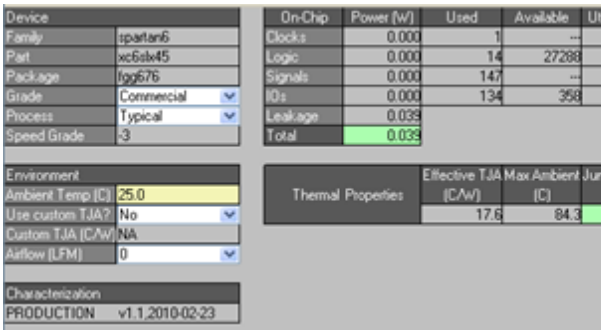

Fig 9: Power used by the FSM based image processor

We compared the various parameters between the existing system and our proposed system. It has been tabulated as follows.

Version	Slice Reg	Slice LUT's	LUT FF pairs	Total Memory used	Time used
EA-IP	193	56	215	249.86MB	5.603ns
FSM+EA -IP	20	14	21	247.55MB	4.881ns

Table 1: Comparison between existing and proposed system

It is seen the area for memory usage and time used for the processing of FSM based image processor has been reduced by 2.31MB and 0.722ns.

VI. APPLICATION AND ADVANTAGES

The advantages of using this system can be listed as follows:

- Area and Power consumption has been reduced, while speed of processing has been increased.
- Can be used for remote sensing, military, medical imaging, weather forecasting applications and other multimedia applications.
- Fully automated system for multimedia applications.
- Since it uses the concept of FSM, it computational complexity has been reduces, and resource utilization reduces.

REFERENCES

[1] "*A Dynamically Reconfigurable Pixel Processor System Based on Power/Energy-Performance-Accuracy Optimization*" Daniel Llamocca, Member, IEEE, and Marios Pattichis, Senior Member, IEEE, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 23, No. 3, March 2013.

[2] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, and L.Sekanina, "*Implementation Techniques for Evolvable Hw Systems: Virtual vs. Dynamic Reconfiguration*," Proc. Int'l Conf. Field Programmable Logic and Applications (FPL), pp. 547-550, 2012.

[3] "*Real-World Applications of Analog and Digital Evolvable Hardware*", Tetsuya Higuchi, Masaya Iwata, Didier Keymeulen, Hidenori Sakanashi, Masahiro Murakawa, Isamu Kajitani, Eiichi Takahashi, Member, IEEE, Kenji Toda, Mehrad Salami, Nobuki Kajihara, and Nobuyuki Otsu, IEEE Transactions on Evolutionary Computation,Vol. 3, No. 3, Sept 1999.

[4] B Rube´n Salvador, Andre´s Otero, Javier Mora, Eduardo de la Torre, Teresa Riesgo, and Lukas Sekanina, "*Self-Reconfigurable Evolvable Hardware System for Adaptive Image Processing*" IEEE Transactions on Computers,Vol. 62, No. 8, Aug 2013.

[5] Yibin Li, Zhiping Jia, Shuai Xie, and Fucai Liu, *"Dynamically Reconfigurable Hardware With a Novel Scheduling Strategy in Energy-Harvesting Sensor Networks"* IEEE Sensors Journal, Vol. 13, No. 5, May 2013.

[6] Karel Jezernik, Robert Horvat, and Jože Harnik, "*Finite State Machine Motor Controller"* IEEE Industrail electronics magazine, pp. 13-23, Sept 2012.

[7] S. Bayar, A. Yurdakul, M. Tukel "*A Self-reconfigurable platform for general purpose image processing systems on low-cost spartan-6 FPGA's"*,IEEE Transactions on Evolutionary Computation, Vol. 5, No. 1, June 2011.

[8] Rahul Kalra and Roman Lysecky, "*Configuration Locking and Schedulability Estimation for Reduced Reconfiguration Overheads of Reconfigurable Systems"* IEEE Transactions on VLSI Systems, Vol. 23, No. 7, April 2010.

[9] Luca Sterpone, Member, IEEE, Mario Porrmann, Member, IEEE and Jens Hagmeyer, Student Member, IEEE "*A Novel Fault Tolerent and Runtime Reconfigurable platform for satellite payload processing"* IEEE Transactions on Computers,Vol. 62, No. 8, Aug 2013.

[10] "*Supporting a Wide Variety of Communication Protocols using Dynamic Partial Reconfiguration"* Richard Dunkley, IEEE Instrumentation and Measurement Magazine, pp.26-32, Aug 2013.

[11] "*A genetic algorithm for Channel Routing in VLSI Circuits*" Jens Lienig and K. Thulasiraman,IEEE Transactions in Evolutionary Computation, 1994.