| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

|| Volume 9, Special Issue 1, October 2020 ||

Placement Algorithm for Dynamically Partially Reconfigurable FPGA

Asst. Prof. Shiji K. Asst. Prof. Fasna K. K., Asst. Prof. Sreedevi K., Asst. Prof. Yasir Moidutty

Assistant Professor, Department of Electronics and Communication Engineering, Royal College of Engineering and Technology, Thrissur, Kerala, India

Assistant Professor, Department of Electronics and Communication Engineering, Royal College of Engineering and Technology, Thrissur, Kerala, India

Assistant Professor, Department of Electronics and Communication Engineering, Royal College of Engineering and Technology, Thrissur, Kerala, India

Assistant Professor, Department of Electronics and Communication Engineering, Royal College of Engineering and Technology, Thrissur, Kerala, India

ABSTRACT: Run time Partially Reconfigurable FPGAs have various applications in the field of Cryptography, Image processing, Network Security, Video streaming etc., because of low power consumption, high density, flexibility and high performance. Furthermore, the dynamism and true multitasking makes it popular in the area of today's advanced computing platforms. A powerful operating system is required to manage and sharing the resources among the various applications system is required. Reconfigurable computing system can be reconfigured at run time by a reconfigurable operating system. The main part of this operating system is resource management. Proposed a technique, called reusing based scheduling (RBS) for online scheduling and placement. This RBS algorithm will minimize the task rejection ratio and overall execution time of the task in re-configurable device.

General Terms

Algorithm, placement, Field Programmable Gate Arrays, Processing units.

KEYWORDS: Reconfigurable computing, on-line scheduling, configuration reusing, RPU partitioning, replacement management, probability of recurrence.

I. INTRODUCTION

Reprogrammable nature of FPGAs introduces new challenges and opportunities in design automation and still there is lack of developing CAD tools for synthesis and Compilation. Runtime and support partial reconfigurability which makes able to change the configuration of part of FPGA dynamically without disturbing other parts. This feature leads to high flexibility along with high performance in computing systems. Reconfigurable computing (RC) systems comprise one or several reconfigurable processing unit (RPU) along with a CPU. RC systems allow us to execute tasks in a true multitasking manner. The trend of utilizing reconfigurable FPGA's is drastically increasing due to high density, parallel computing nature, flexibility and low cost as compared to ASIC and general purpose processors. The main function of resource management unit is online scheduling and placement of hardware tasks, developing algorithms for the same is a challenging issue.

While using RC systems can increase the computing performance, it also leads to complex allocation situations for dynamic task sets. A set of system services forms a reconfigurable operating system. As a common operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs. The operating system is a vital component of the system software in a computer system.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

|| Volume 9, Special Issue 1, October 2020 ||

Application programs usually require an operating system to function. This work deals with the resource management part of reconfigurable operating system. Same time this work focus on the problem of on-line scheduling the tasks to the RPU in an RC system. Here propose an online scheduling and placement algorithm which exploits resource reusing and reduces reconfiguration overhead of the tasks to improve the total execution time of the application.

II. PROBLEM DEFINITION

RPU and Task Models

A task is a function synthesized to digital circuit and can be programmed onto the reconfigurable device (RPM. A task has a size and a shape. The size gives the area requirement of the task in reconfigurable units. We assume all task shapes to be rectangle and the execution time of tasks is known in advance. Generally, each task is defined as a 6-tuple *Ti* — (wi, *hi, ei, ai, di, ri)* where *wi, hi, ei, ai, di* and *ri* denote width, height, execution time, arrival time, deadline and reconfiguration time of the task, respectively['

Tasks can be real-time or non real-time. Deadline *(di)* is defined only for real-time tasks and determines the latest finish time of the task. Also, the tasks can be dependent or independent. Dependent tasks may be represented as a directed task graph. In this work we focus on soft real-time and independent tasks.

Also, the task execution can be preemptive or non preemptive. Although preemption is a very useful technique that yields efficient and fast on-line scheduling algorithm, we believe that it is too expensive for current reconfiguration technology, because preemption and requires heavy (large) context switches and need additional external memory [1 Hence, in this work we focus on a non-preemptive task model, i.e., once a task is loaded onto the device, it runs to completion.

The mapping of tasks to an RPU strongly depends on the area model of RPU. Therefore it is necessary to describe our RPU area model. In general, there are four area models. The flexible 2D area model (see Fig.1(a)) is allowed to allocate rectangular tasks anywhere on the 2D reconfigurable surface. This model has been used by many researchers. The advantage of this model is high device utilization but on the other hand high flexibility leads to difficult scheduling and placement algorithms. Another model is partitioned 2D model (see Fig.1(b)) where the reconfigurable surface is split into a statically-fixed number of allocation sites, so called blocks[' Each block can accommodate at most one task at a time. Such a model facilitates the scheduling and placement but causes internal fragmentation. In the flexible 1 D area model shown in Fig.1(c), tasks can be allocated anywhere along the horizontal device dimension and the vertical dimension is fixed. This model matches well current FPGA technology that is partially reconfigurable in vertical chip-spanning columns " This model leads to low complexity placement problem. On the other hand, it suffers from both internal and external fragmentation. Finally, there is a partitioned lD area model as shown in Fig.1(d). This model combines the characteristics of a partitioned model and 1D model. Again the disadvantage lies in the potentially high internal fragmentation.

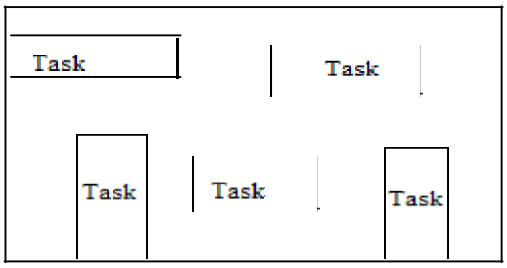


Fig: 1(a) Flexible 2D

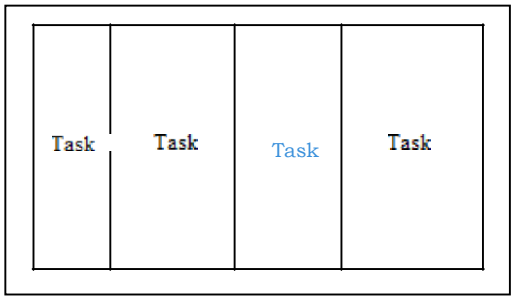


Fig: 1 Partitioned ID

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

|| Volume 9, Special Issue 1, October 2020 ||

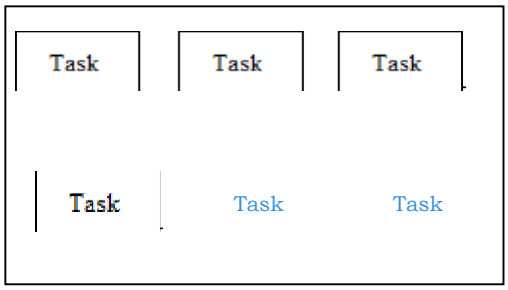


Fig: 1(b) Partitioned 2D

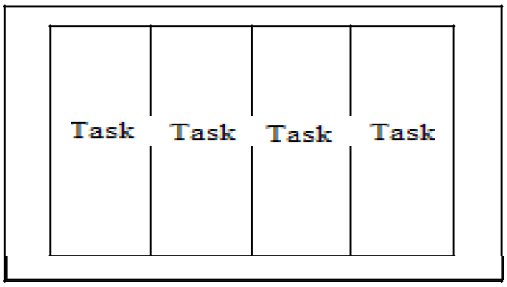


Fig: 1(d) Partitioned lD
Fig: 1 Reconfigurable area models.

In this paper, address the problem of on -line scheduling of real-time tasks to the lD area model. The tasks have known execution time and there is no dependency among them. Such assumptions are the same as used in related research work. There is a large variety of applications which can satisfy these assumptions and exploit our proposed algorithm including: reconfigurable co-processor implementation, image and video processing, cryptography, telecommunication and neural network implementation. This algorithm is more suitable for these applications, because in these cases online tasks will come repeatedly. That's why probability of incoming tasks can be considered as Poisson distribution.

Scheduling and placement problem definition:

In this subsection, define the problem of scheduling and placement. As stated earlier, in this work we use lD area model for device area. Therefore the location of each task *Ti* on the device surface is indexed with xi which is the horizontal position of bottom-left of rectangular task (*Ti).*

In an on-line scenario, new tasks arrive during the system's runtime. For each arriving task *Ti,* scheduler has to find feasible start time. If scheduler and placer cannot find a feasible placement and start time for the arrived task that satisfies the aforementioned constraints, the arrived task will be rejected. In such real-time systems, it is assumed that when a task is rejected, it is referred to the resource out of the system for execution. Two main goals are considered for on- line scheduling and placement in RC systems. One of them is to minimize the task rejection ratio (TRR) in the system. Task rejection ratio is the ratio of the number of rejected tasks to the total number of arrived tasks.

Another main goal in on-line scheduling and placement is to decrease the overall execution time of the tasks.

Regarding above descriptions, state the problem in this paper as follows:

On-line scheduling and placement of independent tasks, in the RPU with ID area model. The main goals are minimizing the task rejection ratio and overall execution time of the tasks.

II. RELATED WORKS

Thomas Marconi, Yi Lu, K. Bertels and G. Gaydadijiev proposed Online Hardware Task Scheduling and Placement Algorithm on Partially reconfigurable devices [6] ' In this paper, propose an online hardware task scheduling and placement algorithm and evaluate it performance. Experimental results on large random task set show that our algorithm outperforms the existing algorithms in terms of reduced total wasted area up to 89.7%, has 1.5 % shorter schedule time and 31.3% faster response time.

In this paper new free space always will be allocated at the leftmost position. At this point, the free space list SL

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

|| Volume 9, Special Issue 1, October 2020 ||

contains only a single free space (PSI) defined by its leftmost column *(CL I),* its rightmost column (CA 1) and free time UTI . When a new task *T* 1 arrives, the algorithm searches the free space list SL and places it on the leftmost edge of EU (according to its alignment status). This action reduces the size of Al, toggles the alignment status of *FSC Tom* leftmost to rightmost, and creates a new free space NS2. NS2 has (CL2, C'fi2) dimension and its free time is *FTP* and leftmost alignment status.

Assume there is another task *T* 2 simultaneously arriving with TI the free space list SL will be processed again. Because the alignment status of *EU* was changed to rightmost, *T* 2 will be placed on rightmost edge of FN1. This action reduces the Al size and again toggles the alignment status of *FSC* to leftmost. The size of U2 is also adjusted and a new free space *FSC* (CL3, Cfi3) is created with free time fK and leftmost alignment status.

This algorithm maintains two linked lists: a free space list (SL) and a task list (TL). The SL contains all free spaces *ii* with their previous pointers *PPi,* dimensions (C'Li and *CRi),* free times *FTi,* alignment statuses *ASi* and next pointers *NPi.* The free time is the time when the corresponding free space can be used. The alignment status is a boolean determining the placement location of the task (leftmost or rightmost) within this free space segment.

- Keeping all MERs: (KAMERs) algorithm increases the space requirement of the algorithm by a linear factor and also slows down the insertion and deletion operations.
- Keeping non overlapping empty rectangles: in order to avoid the quadratic space requirement and increased run time, we can keep only linear number of rectangles in terms of the number of modules. When a new module arrives the algorithm searches in the list of empty rectangles for all empty rectangle that can accommodate the task. Then algorithm uses bin-packaging rule to choose one.

J. Tabero, I.Septien , H.Mecha and D. Mozos presented a approach to manage run time of reconfigurable resources by an operating system with extended hardware multitasking functionality [' . Rectangular hardware tasks are placed at free locations in a two dimensional reconfigurable resource. Area management is done with techniques derived from bin- packing heuristics. A structure consisting of a set of vertex list, each one describing the contour of each unoccupied area fragment in the reconfigurable device is presented. Some vertices of such structures may be used as reasonable complexity and gives better result, in terms of device fragmentation, than similar approaches.

Ali ahmadinia and Jurgen Teich [' developed an new fitting algorithm to reduce the reconfiguration overhead in turn speed up the online placement for FPGAs. Elena perez Ramo et a1. [' proposed the configuration memory hierarchy that provides the first Reconfigurable Computing system with energy savings of 22.5%,without performance degradation by developing the configuration mapping algorithm and integrated into reconfiguration computing manager. Farhad Mehdi Pour et a1. [' concentrating on reducing the Reconfiguration latency that affects the system performance and develop temporal partitioning algorithm. For set of data flow graphs, the placement time improvement range from 0% to 37.5% and reconfiguration speed improvement range from 0.0 to 1.24.

III. METHODOLOGY

Proposed an online scheduling and placement algorithm called reusing-based scheduling (RBS) algorithm [' This algorithm exploits resource reusing (task reusing) and reduces reconfiguration overhead of the tasks to improve the total execution time of the application. By using this algorithm can solve the problem of on-line scheduling for RC systems with real-time, non-preemptive task model and lD area model.

Here already discussed about some online scheduling and placement algorithm which is utilized, free space list. Same time, they can manage one updating list also. In that updating list they can managed where the tasks are placed, how much width has for each placed tasks and which time each tasks will be over. In those papers they didn't bother about the reconfiguration time. For every task, some configuration time is there. Reconfiguration time increases means reconfiguration overhead is also increases.

So in this proposed work just consider the reconfiguration time and use configuration reusing. This configuration reusing can decrease the reconfiguration overhead. If the repeated tasks are coming means this configuration reusing is mostly useful. These type of configuration reusing is mostly useful for the applications like audio and video processing, cryptography etc. because in these applications have known execution time and there is no dependency among them. By using this configuration reusing can reduce reconfiguration overhead and improve the total execution time of the tasks.

In this RBS algorithm, the total area of FPGA can divide in to two parts. One is reusing area and other is no reusing area. Same time tasks are divided in to two types. Mostly frequently coming tasks are called significant tasks and others are called non-significant tasks. These significant tasks are placed on the reusing area and non-significant tasks are placing in non-reusing area.

Fig: 2(a) shows an example in which task T2' is an instance of T2 (a copy of T2) and arrives at time a2'. In this situation this task should be reconfigured again on the free area of RPU. In this figure, dotted parts of the tasks (Ri) indicates the reconfiguration time of the tasks.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

|| Volume 9, Special Issue 1, October 2020 ||

Idea of RBS method is that some of arrived tasks are not removed from RPU surface after their completion of their execution and they stay in RPU surface as far as possible. That's why, in case of repeated tasks, the existing configuration is used and reconfiguration overhead will be eliminated.

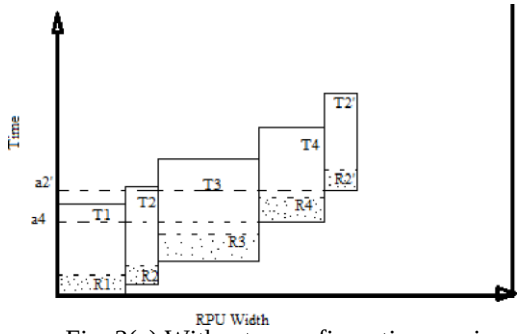


Fig: 2(a) Without reconfiguration reusing

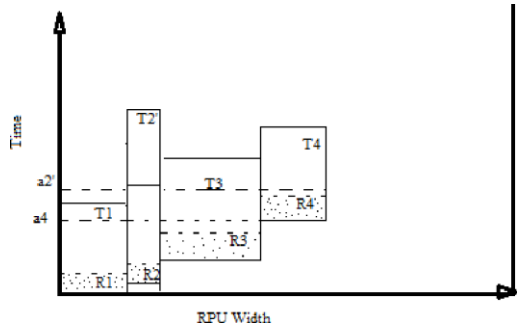


Fig: 2(b) With reconfiguration reusing Fig: 2 An example of task scheduling and placement.

The basic idea of RBS algorithm can explain by using the figure: 2.

Significant and Non-Significant tasks:

The significant tasks can define by two different parameters,

(1) Reconfiguration overhead (Oi) and (2) Probability of task recurrence in the future (Pi).

Reconfiguration overhead can be calculated by the ratio of reconfiguration time to execution time ie,(rJei). The probability of task recurrence can be explained by Poisson distribution. In case of poisson distributions want to find out the average number of arriving tasks (k) in past time interval It.

$$k = (fx\,x)/n. \qquad (1)$$

x is number of fixed tasks, fx is the frequency of each fixed task and n is the total number of tasks.

Here use Poisson distribution for finding the probability of arriving \ tasks in next time interval At. By using equation (1) calculate the values for k to get the probability of arriving task. By using equation (2) can find out the probability of arriving tasks.

$$PAt(m) = (\text{\textquotedbl}\ x\ e\ \text{`})/\ m! \qquad (2)$$

Before going the placement section want to satisfy one condition. That condition is explained by using equation (3).

$$SG = (Oi > ml)\quad (Pi > m2) \qquad (3)$$

ml and m2 are the threshold between 0 & 1.

Partition of RPU Surface

In this algorithm partition the RPU surface into two partitions P1 and P2. Fig.4 shows a total scheme of this partitioning.

$$WP1 = \quad W, WP2 = b\ \ W \qquad (4)$$

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

|| Volume 9, Special Issue 1, October 2020 ||

$0 < \alpha, \beta < 1, \alpha + \beta = 1$

In initial partitioning, the width of each partition is defined by using equation (4).

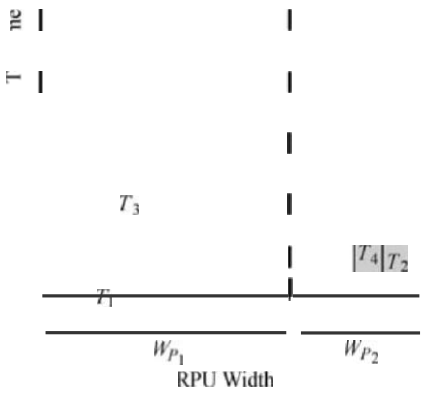


Fig 3. Total Scheme of RPU partitioning

where *WP I* and denote the width of partitions *PI* and f2, respectively. Also, « and Q are positive coefficients between 0 and 1.

Partition II is dedicated to placing the non-significant tasks and partition P2 is dedicated to placing the significant tasks.

Task Placement in RBS Method

In this method use scanning- based placement method for task placement on the RPU surface. The tasks are placed from left to right in partition *PI*. In contrast, the tasks are placed from right to left in partition H.

When new task arrives at the system, two different situations may occur.

1. Newly arrived task is non-significant
2. Newly arrived task is significant.

If the newly arrived task is non-significant, the free area can scanned using stuffing method. If there is enough area for new task then place the task in the PI area. While the free area is not sufficient means consider about partition resizing. After partition resizing the above procedure will repeat, then also enough area is not there means reject that non-significant task. If the newly arrived task is significant, the free area can scanned using stuffing method and there is enough area means place that task in partition P2. In case of next task, it will check the coming task is already placed or not. If it is already differentiation two variables are used ml and m2. The value of these variables is in threshold between 0 & 1. For better result we fine tune these values. By using different set of inputs, can fine tune these values and can find out one equation for finding the values of ml & m2.

IV. EXPERIMENTAL RESULT

Differentiated number of tasks as significant and non- significant by using poisson distribution. Separated the whole area of FPGA for two parts as P1 and P2. Placed the significant tasks on partition P2 and non-significant tasks on partition PI.

With respect to partition area and incoming online tasks the partition resizing is possible. If the number of non significant task is high, then the partition line will move to the portion of significant tasks and vice versa.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

|| Volume 9, Special Issue 1, October 2020 ||

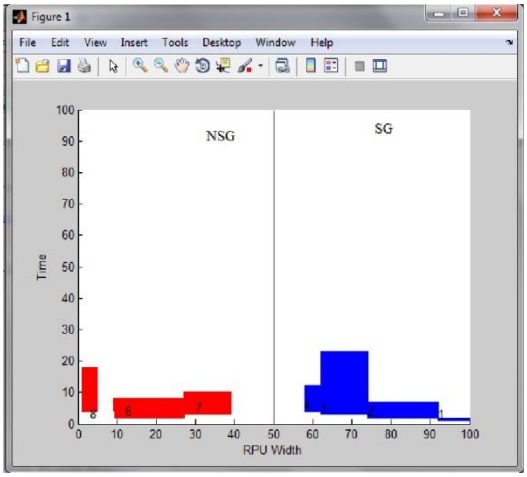
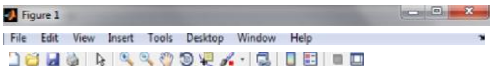
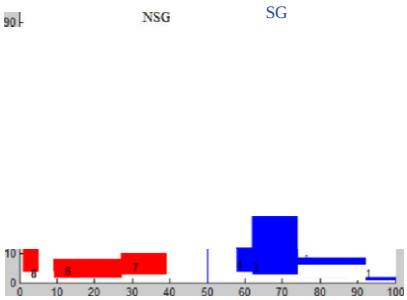


Fig 4. Task placement snap shot before RBS algorithm



placed task, then check the execution time of running task and arrival time of coming task. The arrival time of coming task is greater than or equal to execution time of running task, the coming task will place the same location of running task. That's why no need of reconfiguration of the same location. So for some applications reconfiguration overhead and execution time can reduce efficiently. But this is useful only for repeated tasks are coming.



Partition Resizing

When there is not enough free area on a partition to place new task, partition resizing may be a good solution to prevent task rejection. Partition resizing is to expand the size (width) of the
partition provided that there is excess free area on adjacent partition [' We define a parameter, called partition utilization
{PR. For finding that parameter equation (5) is used.

$$i = \frac{Woccupied + Wpreserved}{Wpi} \quad (5)$$

where *Woccupied, Wpreserved* and *W/ree* are total width of occupied area, total width of preserved area and total width of free area on partition *Hi,* respectively.
Resizing of partition Pi is allowable only if equation (6) is satisfied.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

|| Volume 9, Special Issue 1, October 2020 ||

Fig 5. SG Task placement snap shot after RBS algorithm

Created one updating chart for getting placement details and to find out where we can place the tasks with low task rejection ratio.

$$PUk \quad 0.6 \qquad (6)$$

Above constraint states that resizing of partition *Pi* is allowed only if the adjacent partition of *Pi has* $PUk < 0.6$. Therefore, if Pi's neighbor has $PUi > 0.6$, the resizing of partition *Pi* will not be allowed.

In case of RBS algorithm, differentiation of task as significant or non-significant is very important. For finding this

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

|| Volume 9, Special Issue 1, October 2020 ||

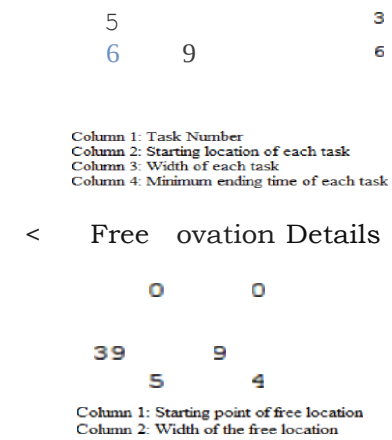


Fig.6 Task placement updating chart snapshot

V.CONCLUSION

Most of the online placement algorithms are used for reducing execution time. Here described the problem of on-line scheduling for RC systems with real-time, non-preemptive task model and lD area model. Here proposed one algorithm (RBS) which performs on-line scheduling and placement based on maximal configuration reusing (task reusing). By using this algorithm reconfiguration overhead can reduce and improve the execution time. That's why improve the application performance.

REFERENCES

[1] Ali Ahmadinia, Jurgen Teich, "Speeding up on line placement for Xilinx FPGA by reducing Configuration Overhead", Proceedings in International Conference VLSI-SOC 2003 on 1-3 Dec 2003, Germany, pp 118- 122.

[2] Bassiri MM, Shahhoseini HS. Configuration Reusing in On-Line Task Scheduling for Reconfigurable Computing Systems. Journal of computer science and technology. 26(3): 463-473 May 2011. DOI 10.1007/s 11390-011-1147-2

[3] Bazargan K, Kastner R, Sarrafzadeh M. Fast template placement for reconfigurable computing systems. In *IEEE Design and Test of Computer.s,* 2000, 17(1): 68- 83.

[4] Elena perez Ramo, Javier Resano, Daniel Mozos, Francky Catthoor, "A Configuartion Memory hierarchy for fast Reconfiguration with reduced energy consumption overhead", IEEE 2006.

[5] Farhad Mehdipour, Morteza Saheb Zamani, Hamid Reza Ahmadifar, Mehdi Sedighi, and Kazuaki Murakami, "Reducing Reconfiguration Time of Reconfigurable Computing Systems in Integrated Temporal Partitioning and Physical Design framework", IEEE 2006.

[6] Marconi T, Lu Y, Bertels K Gaydadjiev G. Online hardware task scheduling and placement algorithm on partially Reconfigurable devices. In *Pr oc. ARCS,* Dresden, Germany,

[7] Roman S, Mecha H, Mozos D, Septien J. Constant complexity scheduling for hardware multitasking in two dimensional reconfigurable field-programmable gate arrays. *Journal of IET Comput. Digit. Tech.*, 2008, 2(6): 401-412.

[8] Getting started with MATLAB: Rudra Pratab, Oxford University Press. Inc, Version 6

[9] Programming in MATLAB: Marc E. Herniter, Bill Stenquist Publication,Second Edition