Adaptive Free Space Management of Online Placement for Reconfigurable Systems

Trong-Yen Lee, Che-Cheng Hu, and Chia-Chun Tsai

Abstract—**The FPGA can be reconfigured both dynamically and partially. Such reconfigurable FPGA allows several tasks to be executed, placed and removed at the runtime. Therefore, the hardware resources management in FPGA on the online placement becomes very important. Most techniques for finding empty space are based on rectangle. In this paper, we propose an adaptive free space management for finding candidate space with rectangular or nonrectangular to place newly arriving tasks. The adaptive free space management uses two procedures to find all feasible candidate space for arriving tasks, namely C-Look and CSAF. Experiment results show that the proposed method reduces 76.49% in rejection rate, 68.12% in total task execution time, and 76.32% in total task waiting time.**

***Index Terms*—FPGA, free space management, candidate space.**

I. Introduction

Dynamically partial reconfigurable (DPR) *filed programmable gate arrays* (FPGAs) allow hardware tasks to be placed and removed at the run time while other tasks in DPR FPGA system are still working. In such a DPR FPGA system, an important component of host CPU is the hardware task placer. The hardware task placer must find feasible candidate space to place a newly arriving task.

The placement management of DPR FPGA can be classified into two types which are offline and online placement. Current researches solving the free space management of DPR FPGAs have focused on online placement [1][2]. Moreover, most researches finding empty space for new task are based on rectangular space [1]-[4].

Bazargan *et al*. [1] proposed a *maximal empty rectangle* (MER) management method which is broadly referenced. The MER will maintain the set of all maximal empty rectangles, only one of them will is decided when a new task can be placed.

Walder *et al*. [3] presented three partitioners based on Bazargan *et al*. [1]: enhanced Bazargan, On-The-Fly (OTF) and enhanced OTF. These enhanced methods improve the placement quality. They also propose a hash matrix to maintain all free rectangles.

Ahmadinia *et al*. [4] presented a free space management based on contour of union of rectangles (CUR) algorithm. The CUR algorithm is applied to find out the contour of axis-parallel rectangles. A new task can be placed in any location within the contour of axis-parallel rectangular. After finding all feasible free spaces for an arriving task, they use routing-conscious placement method to decide an optimal placement such as minimum communication cost.

In this paper, we propose an adaptive free space management of online placement to find all candidate space with rectangular or nonrectangular for placing newly arriving tasks. In all feasible candidate space, we use *multi-strategy fit* (MSF) strategy [5] to determine the best candidate space from all candidate space to place the newly arriving task.

The rest of this paper is organized as follows. Section II shows how the free space is managed. The performance comparison of the proposed method with related works will be shown in Section III. Finally, Section IV draws conclusions.

II. Free Space Management

First of all, we briefly describe the system model, FPGA model and task model in online placement system. The online placement system model has following several characteristics. First, the system model knows the properties of tasks only at runtime. Second, all tasks are no priority. Third, each task is independent work.

In FPGA model, the FPGA surface is composed of homogeneous *configurable logic blocks* (CLBs) and is arranged in a two-dimensional array. The CLB cannot be configured alone, because currently FPGA technology configures reconfigurable logic resources by column-based. We also assume that when configuration area is enough, then sufficient routing resources are able to meet the requirements of the arriving task.

In task model, the parameters of a single task comprise task index, task height, task width and task area. Each task area is equal to the number of CLBs. The shape of tasks can be rotated or segmented. The communication cost and I/O routes are not considered in the task while the task is placed. A more detailed description of system model, FPGA model and task model are described in [5]. In the following, proposed efficient free space management for DPR FPGAs is described.

Manuscript received December 30, 2009. This work was supported in part by National Science Council, ROC under Grant NSC-98-2221-E-027-071.

Trong-Yen Lee is with the Department of Electronic Engineering, National Taipei University of Technology, Taipei, Taiwan ROC. (e-mail: tylee@ntut.edu.tw).

Che-Cheng Hu registered Ph. D. at the Institute of Computer and Communication, National Taipei University of Technology, Taipei, Taiwan ROC. (e-mail: s4419010@ntut.edu.tw).

Chia-Chun Tsai is with the Department of Computer Science and Information Engineering, Nanhua University, Chia-Yi, Taiwan, ROC. (e-mail: chun@mail.nhu.edu.tw).

A. Free Space Matrix

In [6], the free space matrix uses two forms of one-dimensional array to maintain a two-dimensional array with columns and rows. So, it must scans *x* axis (column) and *y* axis (row) two times that is spending time. Moreover, the method only records the free space length of each row and column. Consequently, this method is not apt to find out the continuous free space of rectangle or nonrectangle.

For improving the drawback, our proposed method can solve the limit by recording row information only. Assume that a placement situation of FPGA surface is shown in Fig. 1. The proposed method only maintains free space information of rows (called *R*) in the FPGA surface of two-dimensional arrays, which is given by

$$R = \{r_i \mid i \in (1, 2, \ldots, H)\}, \text{ where } i \text{ is } i\text{th row}$$

with

$$r_i = \begin{cases} \{(cs_j, sp_j) \mid j \in (1, 2, \ldots, \frac{W}{2})\}, \text{ if } FS_i > 0 \\ (0, 0), \text{ if } FS_i = 0 \end{cases}$$

, where

j = segmentation quantity of free space in ith row (FS_i)
cs_j = continuous space quantity of jth segmentation
sp_j = starting point of jth segmentation

For example, Fig. 1 shows an FPGA dimension with 10 rows and 10 columns. In this figure, r_3 = (6, 1) represents 6 units continuous space (*cs*) and the starting point (*sp*) form column 1 respectively in third row. The other example, r_8 = {(2, 1), (1, 6), (1, 10)}, represents three segmentations of free space (*j*) in the eighth row.

B. Resolve Candidate Space

The proposed method includes two procedures to search candidate space for online placement. The first procedure (C-Look) searches rectangular candidate space that has to meet the form of incoming task. The second procedure (CSAF) searches nonrectangular candidate that has to equal or exceed the area of incoming task. These two procedures are detailed described in Subsection II.B.1 and II.B.2, respectively.

B.1 Circular Look

When a new task requires to load into an FPGA, there might be a lot of candidate spaces for devoting the newly incoming task. Therefore, we propose a procedure called *circular look* (C-Look) [7] to select all feasible rectangular candidate space for incoming task. In this procedure, all of *x* axis (columns) and *y* axis (rows) in the FPGA surface must be scanned. But, if there are spaces already used in the scanning range, these spaces will be left out. For example, when a configuration of new task is required, then the scanning range of FPGA surface is illustrated as Fig. 2. Because the scanning range of FPGA surface is according to the column (*tCol*) and row (*tRow*) of incoming task (see Fig. 2(a)), the scanning range (*sCol* and *sRow*) of FPGA surface can be reduced, as shown in Fig. 2(b). The scanning range of FPGA surface are given by

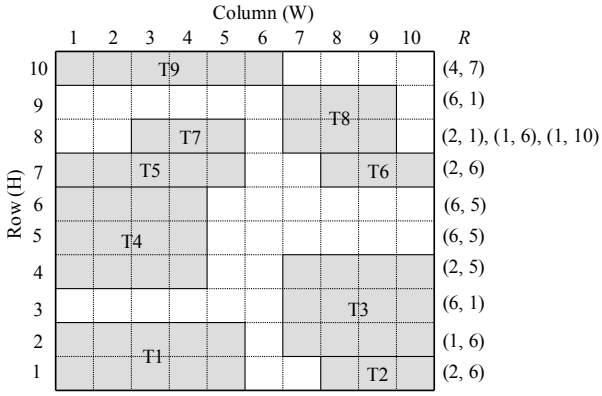

Fig. 1: Free space matrices

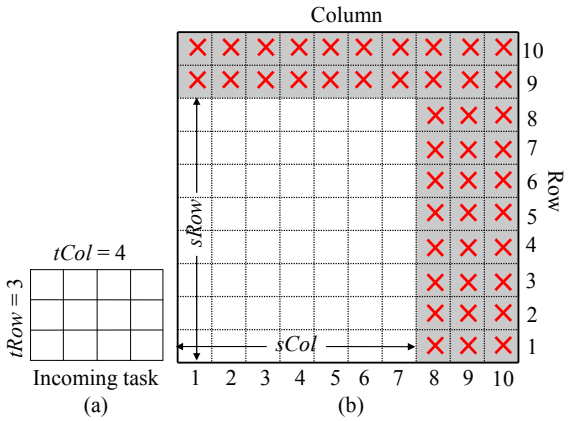

Fig. 2: (a) Incoming task (b) Scanning range of FPGA surface

$$sRow_{si} \mid si \in \{1, 2, \ldots, (Row - tRow + 1)\}$$
$$sCol_{sj} \mid sj \in \{1, 2, \ldots, (Column - tCol + 1)\}$$

and

$$\forall (sRow_{si}, sCol_{sj}) \in \text{Free space}$$

, where

$si \subset i$ of r_i, (sith row in the FPGA surface)
$sj \subseteq (sj \cap r_{si})$, ($sj$th column in the FPGA surface)

In addition, if the C-Look has found candidate space (*cas*) in *sj*th column (represented cas_{sj} = True), then C-Look could leave *sj*th column out in the future scanning. Therefore, the new scanning range of *sCol* is given by

$$sCol_{nsj} \mid nsj \in \{ \exists sj : (cas_{sj} \neq \text{True})\}$$

, where

nsj = the new scanning rang for column in FPGA

Consequently, the C-Look procedure can reduce a lot of scanning ranges.

While searching for candidate spaces, each free space will check resources which are delimited to rectangular have been used or not. If there is a totally free space, this space has candidacy. The candidate space is given by C_k = ($sRow_{si}$, $sCol_{nsj}$, *tRow*, *tCol*), where *k* is the number of candidate space. Moreover, the upper bound of *k* is (*Column* – *tCol* + 1).

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

An example for searching the candidate space by C-Look is shown in Fig. 3. In this example, the scanning ranges of C-Look are $sRow_{si}$ = {1, 2, … , 8} and $sCol_{sj}$ = {1, 2, … , 9}. Moreover, the C-Look only scans free space in the FPGA surface such as ($sRow_1$, $sCol_6$) and ($sRow_1$, $sCol_7$) and so on. When the candidate C_1 = (3, 5, 3, 2) is found, the scanning of $sRow_4$ to $sRow_8$ will leave out the $sCol_5$ (cas_5 = True). Similarly, when the candidate C_2 = (5, 6, 3, 2) is found, the scanning of $sRow_6$ to $sRow_8$ will leave out the $sCol_6$ (cas_6 = True).

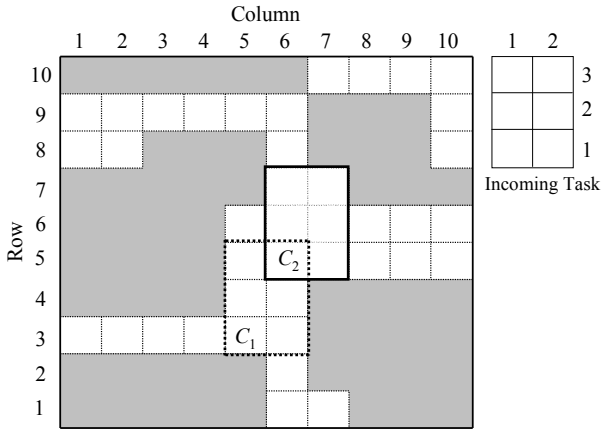

Fig. 3: An example for of searching the candidate space by C-Look

B.2 Continuous Space of Arbitrary Form

If the C-Look failed to get the candidate space, then the second proposed procedure called *continuous space of arbitrary form* (CSAF) is used to search candidate space. The CSAF improved from a widely known algorithm *contour of union of rectangles* (CUR) [4][8]-[11] for finding the feasible continuous space of arbitrary form. The CSAF is based on the observation that the free space consists of serial continuous rectangle spaces, as shown in the Fig. 4. The Fig. 4(a) has four rectangles which can be integrated into a single nonrectangle shown as Fig. 4(b).

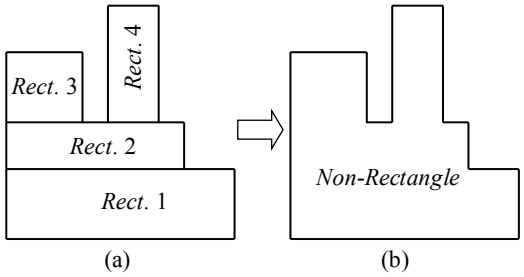

Fig. 4: Free space of arbitrary form consists of simple geometry

Given a continuous space structure which is shown in the Fig. 5(a). The data structure of arbitrary continuous space (*acs*) is associated with free space matrix r_i. The symbols of *acs* data structure are defined as following: sp_{cr} and ep_{cr} are represented as start point and terminal point, where *cr* represents scan line in the *i*th row at present (scan by row). *seg* and *ta* are represented as the number of segmentation in *acs* and total area of continuous space.

The sp_{cr} and ep_{cr} are used for checking whether it has connection with free space of the next row (represented *cr*+1). If FS_i has connection with *acs*, the sp_{cr}, ep_{cr} and *ta* are given by

$$sp_{cr} = sp_j$$
$$ep_{cr} = cs_j + sp_j - 1$$
$$ta = \sum_{i=1}^{cr} (cs_j \text{ of } r_i)$$

, where sp_j and $cs_j \in r_{cr}$, moreover, $cr \in i$ of r_i ($r_{cr} \in r_i$)

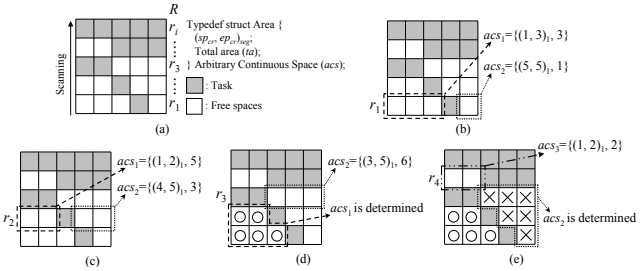

Fig. 5: (a) Continuous space structure (b) Scan line in r_1 (c) Scan line in r_2 (d) Scan line in r_3 (e) Scan line in r_4

Firstly, when the scan line is in r_1 (*cr* = 1), the CSAF can get two independent continuous spaces (acs_1 and acs_2), as shown in Fig. 5(b). Next, when the scan line is in r_2 (*cr* = 2), then the acs_1 and acs_2 are changed, as shown Fig. 5(c). Thirdly, when the scan line is in r_3 (*cr* = 3), the continuous space and form of acs_1 are determined because free space in r_3 has no connection with acs_1, as shown in Fig. 5(d). Moreover, the acs_2 is redefined again. Finally, when the scan line is in r_4 (*cr* = 4), the continuous space and form of acs_2 are determined because free space in r_4 has not connection with acs_2, as shown in Fig. 5(e). Moreover, the CSAF can get a new independent continuous space (acs_3).

The CSAF may encounter three kinds of situations, while CSAF searches for continuous space of the arbitrary form. The three kinds of situation are branch to merge (BM), merge to branch (MB) and BM mix with MB which are shown in Fig. 6. In Fig. 6(a), first, there are two independent *acs* (acs_1 and acs_2) at the beginning. Second, a rectangle of *x* to *x′* joins acs_1 with acs_2. Finally, two independent *acs* merge into one *acs*. Moreover, *ta* is sum of the area of three rectangles. In Fig. 6(b), one *acs* is divided into two rectangles (*l*1 and *l*2). However, these branches are still the same *acs*. Therefore, the *acs* has two groups of sp_{cr} and ep_{cr} (represented *seg* = 2). In Fig. 6(c), firstly, the preceding part is the same as Fig. 6(b). Next, a rectangle of *x* to *x′* combined this branch. Therefore, the *seg* reduces form two groups to one group. The *ta* is sum of the area of four rectangles.

For example, the Fig. 7 shows a result of searching candidate space of arbitrary form. In this example, we use CSAF procedure to search the continuous space (see Fig. 7(a)) that can get acs_1 and acs_2, as shown in Fig. 7(b). However,

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

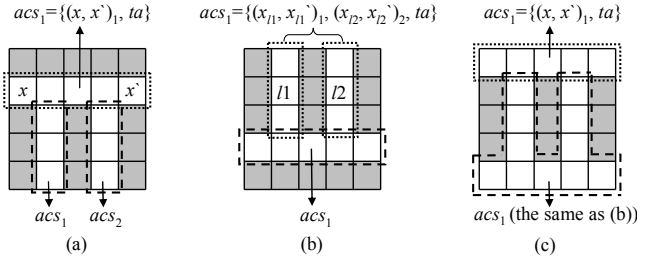

Fig. 6: The three kinds of situations in CSAF that may meet: (a) BM, two independent *acs* merge into one *acs* (b) MB, a *acs* has branches (*seg* = 2) (c) BM mix with MB, a *acs* has the branch and mergence

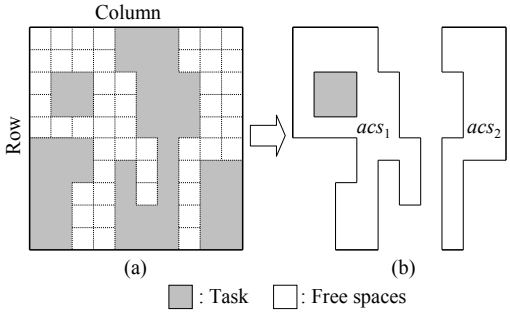

Fig. 7: A set of existing arbitrary continuous spaces (white) on a partially reconfigurable FPGA

these *acs*(*ta*) have to equal or exceed the area of incoming task. If the *acs*(*ta*) is smaller than the area of incoming task, this *acs* will not has candidacy. Therefore, the *acs* must accord with the following condition that is given by

$$acs(ta)\begin{cases} \geq tRow \times tCol, \text{has candidacy} \\ < tRow \times tCol, \text{no candidacy} \end{cases}$$

In the CSAF procedure, the candidate space C_k is redefined as following. The C_k area can be a continuous space of arbitrary (*acs*) form. The C_k area has to equal or be greater than the area of incoming task. Note that the *acs* area may be greater than the area of incoming task. Therefore, each *acs* may has a lot of candidate spaces.

The candidate spaces for newly arriving task are selected by C-Look procedure or CSAF procedure in free space management. A fitter is used to find the best candidate space from all feasible candidate space to allocate for newly arriving task. The fitter has been developed in [5]. The fitter considers time factor, fragmentation factor and modifiability factor, in order to improve time overhead and placement quality, such as rejection rate and fragmentation. A more detailed description of fitting strategy is described in [5].

C. Adaptive Free Space Management Algorithm

The detailed flow of proposed adaptive free space management (AFSM) will be described in this subsection. The algorithm proposed adaptive free space management is shown in Fig. 8. The AFSM algorithm consists of two procedures which describe in the following.

Adaptive Free Space Management (AFSM)
Input: *FPGA resources status, incoming task information* (*tRow* and *tCol*)
Output: *All feasible candidate space for newly arriving task*
begin
1 **If** (ASFM has newly arriving task requirement)
2 { **If** (the number of free space greater than incoming task area)
3 { *cs* ← ***C-Look***(*tRow, tCol*); /*Find all candidate spaces (*cs*).*/
4 **If** (*cs* > 0)
5 **Return** all candidate spaces;
6 **Else**
7 { *cs* ← ***CSAF***(*tRow, tCol*); /*Find all candidate spaces (*cs*).*/
8 **If** (*cs* > 0)
9 **Return** all candidate spaces;
10 **Else**
11 **Return** the incoming task is rejected;
12 }
13 }
14 **Else**
15 **Return** the incoming task is rejected;
16 }
end

Fig. 8: Adaptive free space management algorithm

In step 1, the AFSM receives a newly arriving task requirement from DPR FPGA system. Next, if the number of free space in DPR FPGA is greater than newly incoming task area, then steps 3 to 13 are executed. Otherwise, AFSM will return a message that newly arriving task is rejected (step 15). In step 3, C-Look procedure will find all feasible candidate space for arriving task according to task information, *tRow* and *tCol*. If candidate space for arriving task is found by C-Look (*cs* > 0, step 4), the AFSM will return all feasible candidate space to fitter (step 5). Moreover, the flow of AFSM will be end. If the C-Look procedure failed to get the candidate space, then the second procedure CSAF is used to search candidate space (Step 7). If candidate space for arriving task is found by CSAF (*cs* > 0, step 8), the AFSM will return all feasible candidate space to fitter (step 9). Moreover, the flow of AFSM will be end. If the CSAF procedure failed to get the candidate space, then AFSM will return a message that newly arriving task is rejected (step 11).

III. Experiment Results

The proposed adaptive free space management of online placement has been evaluated and compared in this section. The experiment environment uses Dev-C++ compiler to evaluate the performance of proposed method. Moreover, measurement environment have been implemented on a Pentium D 3GHz PC running under the Windows XP operating system. We assume an FPGA with size of 96x40 CLBs, corresponding to the Xilinx Virtex device.

In the experimentation framework, we have randomly generated three different kinds of task sets. All of task sets are composed of 1000 tasks which have to be placed on the FPGA. In other words, all of tasks must be executed. Moreover, we assumption those tasks are independence and there are no priority. The size of task dimensions is between 5 to 100 CLBs. In addition, the execution time and the arrival time of tasks are distributed form 20 to 200 time units and 1 to 3 time units, respectively.

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE I. Performance comparison with related works

	Rejection Rate				TT Execution Time				TT Waiting Time			
	BL[1]	LIF[16]	BFIT[17]	Our	BL[1]	LIF[16]	BFIT[17]	Our	BL[1]	LIF[16]	BFIT[17]	Our
Task Set 1	37.3%	16.7%	35.1%	5.8%	16391	10590	15988	4652	421798	181036	442579	73717
Task Set 2	36.0%	16.6%	29.9%	7.0%	16105	9947	18230	4777	398985	155512	375425	67032
Task Set 3	32.9%	13.5%	32.9%	4.4%	19557	9645	18345	3971	396168	143729	428418	50893
Average	35.4%	15.6%	32.6%	5.7%	17351	10060.7	17521	4466.7	405650.3	160092.3	415474	63880.7
Reduced	–	–	–	76.49%	–	–	–	68.12%	–	–	–	76.32%

TT: Total task. *Reduced*: the new method compared to [1], [16] and [17], for example (Rejection Rate), {[(35.4-5.7)/35.4] + [(15.6-5.7)/15.6] + [(32.6-5.7)/32.6]}/3 = 76.49%

We also consider an important factor which is to interfere in other tasks of the same columns while the configuration of new task is required. Then, the influenced tasks will be suspended during the configuration of new task [2][12]-[14] that causes extra time overhead to save and restore them [15]. Therefore, we assume the interfered time is 3 time units.

Table I compare the rejection rate, execution time and waiting time of the method of Bazargan *et al*. [1], S. P. Fekete *et al*. [16] and M. Esmaeildoust *et al*. [17], with that of the proposed method. The column "Rejection Rate" in Table I, clearly, the proposed method on the average has the less rejection rate in all task sets. Therefore, the proposed method reduces 76.49% compared with the [1], [16] and [17]. The columns "TT Execution Time" and "TT Waiting Time" in Table I, we have considered the three factors [5] in this work. Clearly, the proposed method requires less execution time and waiting time in all task sets. Therefore, the proposed method reduces 68.12% and 76.32% compared with the [1], [16] and [17].

IV. CONCLUSION

In this paper, an adaptive free space management of online placement to find all candidate space for placing newly arriving tasks in DPR FPGA systems is presented. Moreover, the proposed method uses C-Look and CSAF procedures to find the candidate spaces with rectangular and nonrectangular. The experimental results show that our proposed method reduces 76.49% in rejection rate, reduces 68.12% in total task execution time and reduces 76.32% in total task waiting time.

REFERENCES

[1] K. Bazargan, R. Kastner, and M. Sarrafzadeh, **"**Fast template placement for reconfigurable computing systems," *IEEE Design and Test of Computers*, Vol. 17, pp. 68-83, 2000.

[2] A. Ahmadinia and J. Teich, "Speeding up online placement for XILINX FPGAs by reducing configuration overhead," in *Proceedings of the IFIP International Conference on VLSISOC*, Dec. 2003, pp. 118-122.

[3] H. Walder, C. Steiger, and M. Platzner, "Fast online task placement on FPGAs: Free space partitioning and 2D-hashing," in *Proceedings of the International Parallel and Distributed Processing Symposium*, April 22-26, 2003, p. 178.

[4] A. Ahmadinia, C. Bobda, S. P. Fekete, J. Teich, and J. v.d. Veen, "Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices," *IEEE Transactions on Computers*, Vol. 56, pp. 673-680, May 2007.

[5] Trong-Yen Lee, Che-Cheng Hu, and Chia-Chun Tsai, "Multi-Strategy Online Placement for Dynamically Partial Reconfigurable Device," in *Proceedings of the International Conference on High-Speed Circuits Design*, October 26-27, 2009, pp. H-20-H-26.

[6] A. A. ElFarag, H. M. El-Boghdadi, and S. I. Shaheen, "Fragmentation aware placement in reconfigurable devices," in *Proceedings of the 6th International Workshop on System on Chip for Real Time Applications*, Dec. 2006, pp. 37-44.

[7] M. Javad and I. Khan, "Simulation and performance comparison of four disk scheduling algorithms," in *Proceedings of the IEEE TENCON 2000*, September 24-27, 2000, pp. 10-15.

[8] A. Ahmadinia, C. Bobda, S. P. Fekete, J. Teich, and J. v.d. Veen, "Optimal routing-conscious dynamic placement for reconfigurable devices," in *Proceeding of the Field Programmable Logic and Applications*, August 2004, pp. 847-851.

[9] R.H. Gu¨ ting, "An Optimal Contour Algorithm for ISO-Oriented Rectangles," *J. Algorithms*, vol. 5, pp. 303-326, 1984.

[10] W. Lipski Jr. and F.P. Preparata, "Finding the Contour of a Union of ISO-Oriented Rectangles," *J. Algorithms*, Vol. 1, pp. 235-246, 1980, errata in 2 (1981), 105; corrigendum in 3 (1982), 301-302.

[11] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction. Springer*, 1985.

[12] Xilinx, Inc. "Virtex series configuration architecture user guide," Xilinx XAPP151 v1.7, October 20, 2004. http://www.xilinx.com/bvdocs/appnotes/xapp151.pdf

[13] D. Mesquita, F. Moraes, J. P. L. Möller, and N. Calazans, "Remote and Partial Reconfiguration of FPGA: tools and trends," in *Proceedings of the International Parallel and Distributed Processing Symposium / Reconifgurable Architectures Workshop*, April 2003, p. 8.

[14] H. Walder and M. Platzner, "Online Scheduling for Block-partitioned Reconfigurable Devices," in *Proceedings of the Design Automation and Test in Europe*, 2003, pp. 290-295.

[15] L. Levinson, R. Männer, M. Sessler, and H. Simmler, "Preemptive Multitasking on FPGAs," in *Proceedings of the Field-Programmable Custom Computing Machines Symposium*, 2000, pp. 301-302.

[16] S. P. Fekete, Jan C. van der Veen, A. Ahmadinia, D. Göhringer, M. Majer, and J. Teich, "Offline and online aspects of defragmenting the module layout of a partially reconfigurable device," *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 16, pp. 1210-1219, September 2008.

[17] M. Esmaeildoust, M. Fazlali, A. Zakerolhosseini, and M. Karimi, "Fragmentation aware placement algorithm for a reconfigurable system," in *Proceedings of the Second International Conference on Electrical Engineering*, March 25-26, 2008, pp. 1-5.

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)