

April 2015

An Efficient Algorithm for On-line Placement of Partially Reconfigurable Devices

U.Lenin Marksia

Department Of ECE, Dr.Sivanthi Aditanar college of Engineering, Tiruchendur-628215. Tamil Nadu, India,
marksia@gmail.com

S. Darwin

Department Of ECE, Dr.Sivanthi Aditanar college of Engineering, Tiruchendur-628215. Tamil Nadu, India,
darwin@gmail.com

Follow this and additional works at: <https://www.interscience.in/ijeee>



Part of the [Power and Energy Commons](#)

Recommended Citation

Marksia, U.Lenin and Darwin, S. (2015) "An Efficient Algorithm for On-line Placement of Partially Reconfigurable Devices," *International Journal of Electronics and Electrical Engineering*: Vol. 3 : Iss. 4 , Article 16.

DOI: 10.47893/IJEEE.2015.1175

Available at: <https://www.interscience.in/ijeee/vol3/iss4/16>

This Article is brought to you for free and open access by the Interscience Journals at Interscience Research Network. It has been accepted for inclusion in International Journal of Electronics and Electrical Engineering by an authorized editor of Interscience Research Network. For more information, please contact sritampatnaik@gmail.com.

An Efficient Algorithm for On-line Placement of Partially Reconfigurable Devices

U.Lenin Marksia^{#1}, S.Darwin^{#2}

^{#1} Assistant Professor, ^{#2} Assistant Professor, Department Of ECE,

Dr.Sivanthi Aditanar college of Engineering, Tiruchendur-628215. Tamil Nadu, India

Abstract- Partial Runtime Reconfigurable (PRTR) FPGAs allow HW tasks to be placed and removed dynamically at runtime. We make two contributions in this paper. First, we present an efficient algorithm for finding the complete set of Maximal Empty Rectangles on a 2D PRTR FPGA, which is useful for online placement and scheduling of HW tasks. The algorithm is incremental and only updates the local region affected by each task addition or removal event. Second, we present an efficient online deadline-constrained task placement algorithm for minimizing area fragmentation on the FPGA by using an area fragmentation metric that takes into account probability distribution of sizes of future task arrivals as well as the time axis. The techniques presented in this paper are useful in an operating system for runtime reconfigurable FPGAs to manage the HW resources on the FPGA when HW tasks that arrive and finish dynamically at runtime. Online placement methods are required that achieve a high placement quality and lead to efficient implementation.

Keywords: MER, MKE, Valley point, FPGA

I. Introduction

An important component of an operating system for PRTR FPGAs is the HW task scheduler and placer, which must find empty space to place a new task, and recycle the occupied area when a task is finished. When HW tasks arrive at runtime, the operating system needs to find an empty space on the FPGA to accommodate the newly-arrived task. There are mainly three approaches for maintaining the empty space on a FPGA device: as a list of Non-Overlapping Rectangles, a list of Maximal Empty Rectangles (MER), or a list of vertices, each with its pros and cons. A MER is defined as an empty rectangle that cannot be fully covered by any other rectangle. Managing the empty space with MERs allows us to fit more tasks on a given area than with non-overlapping rectangles, but it is often time consuming to maintain the complete set of MERs [1], and it is important to reduce the runtime overhead in order to make the technique suitable for online use. In this paper, we present an efficient algorithm for finding the complete set of MERs for a FPGA area.

Reconfigurable HW devices such as Field Programmable Gate Arrays (FPGAs) are very popular in today's embedded systems design due to their low-cost, high-performance and flexibility. A FPGA consists of a rectangular grid of Configurable Logic Blocks (CLBs), also referred to as cells, and the interconnects between them. FPGAs are inherently parallel, that is, two or more HW tasks can execute on a FPGA concurrently as long as they can both fit on it. Partial Runtime Reconfigurable (PRTR) FPGAs, such as the Virtex family of FPGAs from Xilinx, allow part of the FPGA area to be reconfigured while the remainder continues to operate without interruption. In other words, HW tasks can be placed and removed dynamically at runtime. A FPGA can be 1D reconfigurable, where each task occupies a contiguous set of columns, or 2D reconfigurable, where each task occupies a rectangular area. Early versions of Xilinx FPGA devices, such as Virtex-II and Spartan, only support 1D reconfiguration. In 2006, Xilinx introduced the Early-Access Partial Reconfiguration Flow (EAPR) to permit 2D reconfiguration. Virtex-4 and Virtex-V devices support independent reconfiguration of a minimum of 16 CLBs in the same column, making it possible to have 2D dynamic reconfiguration [2]. In this paper, we address the problem of online scheduling of soft real-time HW tasks with unknown arrival times and execution times on a 2D PRTR FPGA. Certain tasks may be rejected when the system is overloaded. The goal is to minimize task rejection

ratio while guaranteeing all accepted tasks to meet their deadlines.

An Operating System (OS) for PRTR FPGAs consists of a HW task scheduler and allocator (task placer), which needs to find an empty area to place a new task, and recycle the occupied area when a task is finished. When a new task arrives at runtime, the OS acts as a dispatcher and decides whether it is accepted or rejected based on its deadline and the occupation condition of the FPGA area. If it is accepted, one MER among all MERs is chosen, and one of the four corners of the chosen MER is in turn chosen to place the task if the MER area is greater than the task.

We first introduce some notations. The FPGA reconfigurable area contains $W \times H$ cells forming a rectangle of width W and height H . Each task T_i is characterized by a tuple of five parameters, $(W_i, H_i, E_i, D_i, A_i)$, where W_i and H_i refer to its width and the height; E_i and D_i and A_i refer to its execution time, deadline and arrival time. There are mainly two approaches to managing the empty space on a FPGA device: maintaining a list of non-overlapping rectangles [3], a list of Maximal Empty Rectangles (MER) [4], or a list of vertices, each with its pros and cons. A MER is defined as an empty rectangle that cannot be fully covered by any other rectangle. Each MER (or more generally, each rectangle) is denoted by a tuple (x, y, w, h) , where (x, y) is the coordinates of its lower left corner, and (w, h) is its width and height in terms of number of cells.

This paper is structured as follows: we first provide some basic definitions in Section 2. We introduce an efficient algorithm for finding MER in Section 3, and related work in section 4, We present performance evaluation results in Section 5, and finally conclude in Section 6.

For simplicity, we assume that the entire FPGA area is uniformly reconfigurable without any pre-configured cells, and tasks can be flexibly placed anywhere on the 2D FPGA area as long as there is enough empty space. In practice it is common to pre-configure some cells of the FPGA area for dedicated purposes such as memory, and application tasks cannot be placed on these cells. This situation can be easily handled in our algorithms by denoting these cells as always in use.

II. Basic Definitions

The FPGA reconfigurable area contains $W \times H$ Configurable Logic Blocks (CLB), forming a rectangle of width W and height H . We use the word cell to refer to CLB in this paper. Each hardware task or MER occupies a rectangular area denoted by the tuple (x, y, w, h) , where (x, y) is the coordinates of its lower left corner, and (w, h) is its width and height in terms of number of cells. Figure 1(a) illustrates these concepts with a 6×10 FPGA area. Two MERs $(2,3,4,2)$ and $(4,3,2,6)$ are highlighted with bold borders. There are 8 MERs in total, with the other 6 being $(1,10,6,1)$, $(5,1,1,10)$, $(2,8,5,1)$, $(3,7,3,2)$, $(1,4,6,1)$ and $(5,1,2,2)$, which are not highlighted.

We use a 2D matrix $M[W + 1][H]$ to represent the FPGA area, defined as:

$$M[i][j] = \begin{cases} M[i - 1][j] + 1 & \text{if } M[i][j] \text{ is not occupied;} \\ 0 & \text{if } M[i][j] \text{ is occupied} \\ & \text{or } i = 0 \text{ or } i = W + 1. \end{cases}$$

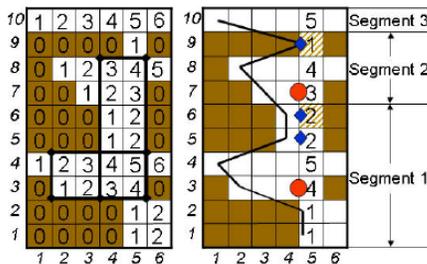
where

$$1 \leq i \leq W + 1, 1 \leq j \leq H$$

Figure 1 shows the values assigned to each cell on the FPGA. For convenience, we add one extra matrix column at horizontal position $W + 1$ to the right edge of the FPGA area, and assign value 0 to cells on that column. This column is not shown in Figure 1. Intuitively, value of $M[i][j]$ is the number of empty cells to its left if cell (i, j) is empty, and is 0 otherwise.

Definition 1 (Key Element) $M[i][j]$ where $1 \leq i \leq W$ and $M[i][j] \neq 0 \wedge M[i + 1][j] = 0$.

Definition 2 (Scan Line) the matrix column that contains one or more Key Elements, e.g., if $M[i][j]$ is a Key Element, then the corresponding Scan Line is Column i .



(a) FPGA Area configuration and its matrix values.

(b) MKEs (circles), Key Elements that are not maximal (lozenges) and Valley Points (shaded cells) on Column 5.

Figure 1. Example FPGA configuration. Dark area denotes occupied cells and white area denotes empty cells.

Intuitively, a Key Element is an empty cell with an occupied cell as its righthand neighbour, or an empty cell on the right edge of the FPGA area. A Scan Line contains one or more Key Elements, hence it has one or more occupied cells as its righthand neighbours. We only need to look for MERs on the left-hand side of each Scan Line, that is, the MERs whose right-hand edge falls on a Scan Line. We will prove that we can find all MERs this way later in this paper.

Definition 3 (Valley Point) a point $s_{i,j}$ on a Scan Line where $s_{i,j} < s_{i,j+1}$, and $\exists k, k < j, s_{i,k} \geq \dots \geq s_{i,j}$.

Definition 4 (Segment and Maximum Key Element (MKE)) If we use an array $\{a_1, a_2, \dots, a_n\}$ to record the vertical coordinates of the valley points of a Scan Line, then the Scan Line is divided into $n + 1$ segments $[1, a_1), [a_1, a_2), \dots, [a_n, H]$ from bottom to top. The MKE is defined as the largest Key Element in a given segment.

A MKE is not necessarily the “peak point” of a segment that contains it. As shown in Figure 1(b), the point $(5,7)$ is the MKE of segment 2 instead of $(5,8)$, because $(5,8)$ does not have an occupied cell as its right-hand neighbour and hence is not a Key Element. Note that there is another Scan Line at Column 6 with MKEs at $(6,2)$, $(6,4)$, $(6,8)$ and $(6,10)$, which is not shown in Figure 1(b) in order to avoid clutter.

III. An Efficient algorithm for finding MER

In this section, we present SLA and prove its correctness and completeness. We start with a basic version of the algorithm in Section 3.1, and then present an enhanced version in Section 3.2 that improves upon its efficiency. We finally present the online version in Section 3.3 for incremental updating MERs upon task addition or removal. The main purpose of presenting the first two versions of SLA in Sections 3.1 and 3.2 is for illustrating the key concepts, but the online version in Section 3.3 is the algorithm that should be used in the actual system.

3.1 Basic SLA

Algorithm 1 Basic SLA for finding MERs based on MKE at (i, j) .

Inputs: FPGA area matrix M and MKE at (i, j) .

Outputs: Set of MERs generated by the MKE at (i, j) .

```

Begin Basic SLA( $i, j$ )
top ( $j$ , bottom ( $j$ );
for  $w = M[i][j]$  to 1 do
t (top, b (bottom);
/* Move top upwards until it gets “stuck”. */
while  $M[i][top + 1]_w$  and  $top + 1 < H$  do
top (top + 1;
end while
/* Move bottom downwards until it gets “stuck”. */
while  $M[i][bottom - 1]_w$  and  $bottom - 1 > 1$  do
bottom (bottom - 1;
end while
/* If top or bottom changed, or within the first for loop iteration,
then a MER is found and recorded. */
if  $w = M[i][j]$  or  $top = 6 = t$  or  $bottom = 6 = b$  then
Record MER( $i - w + 1$ , bottom,  $w$ ,  $top - bottom + 1$ );
end if
end for
End Basic SLA;
    
```

For a MKE at position (i, j) , we use a variable w to iterate from $M(i, j)$ to 1. Intuitively, for each w , we move a horizontal line top with length w and initial position between vertices $(i - w + 1, j)$ and (i, j) upwards until it gets “stuck”, i.e., it hits an occupied cell and cannot move any further. We then move another horizontal line bottom with the same length and initial position downwards until it gets “stuck”. The rectangle between the final positions of top and bottom with width w is recorded as a MER($i - w + 1$, bottom, w , $top - bottom + 1$). However, if top and bottom are moved to the same positions when w is reduced by 1, then the generated rectangle is not a MER and should not be recorded. We provide a formal proof of the correctness of this algorithm in the next section.

In this paper, we use the term scanning the MKE at (i, j) to refer to running Basic SLA with MKE at (i, j) as its input, and the term scanning the Scan Line at Column i to refer to scanning all MKEs on the Scan Line at Column i . Figure 2 shows the

process of scanning the MKE at (5, 7), and Figure 3 shows the process of scanning the MKE at (5, 3). For brevity, we only show the configurations when both top and bottom are “stuck” and a new MER is generated and recorded, and omit any intermediate configurations.

The steps of scanning the MKE at (5, 7) are:

- Step 1: $w = 3$, top = 8, bottom = 7, record MER(3, 7, 3, 2).
- Step 2: $w = 2$, top = 8, bottom = 3, record MER(4, 3, 2, 6).
- Step 3: $w = 1$, top = 10, bottom = 1, record MER(5, 1, 1, 10).

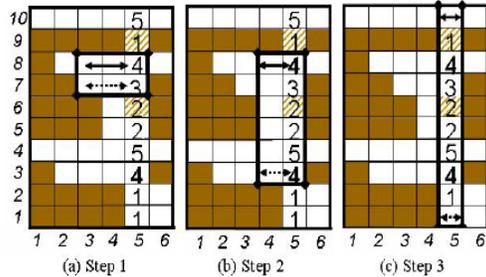


Figure 2. Scanning the MKE at (5,7).

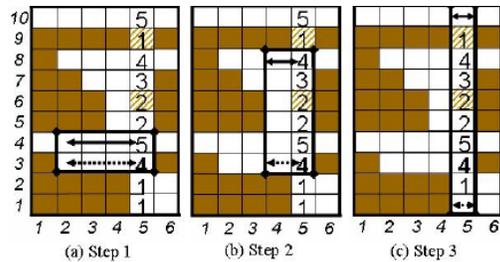


Figure 3. Scanning the MKE at (5,3).

The steps of scanning the MKE at (5, 3) is similar. As we can see, Steps 2 and 3 are identical in Figures 2 and 2. We will propose Enhanced SLA in Section 3.2 to remove this redundancy in the search process. Next, we prove two theorems that guarantee the correctness and completeness of Basic SLA. The proof can be found in [5]. These theorems allow us to only search for all MERs whose right edges fall on Scan Lines in order to find all the MERs. In other words, running Basic SLA for all MKEs on all Scan Lines will generate the complete set of MERs on the FPGA.

Theorem 1. The right edge of any MER must fall on a Scan Line, and any Scan Line must have at least one MER whose right edge falls on it.

Theorem 2. The set of MERs found by scanning all the MKEs on the Scan Line at Column i is the complete set of MERs whose right edges fall on Column i .

IV. Enhanced SLA

Scanning multiple MKEs on one Scan Line using Basic SLA can result in duplicate MERs being generated during the scanning process. For example, scanning the MKE at (5, 3) in Figure 1(b) will generate three MERs (2, 3, 3, 2), (4, 3, 2, 6) and (5, 1, 1, 10), and scanning the MKE at (5, 7) will generate another three MERs (3, 7, 3, 2), (4, 3, 2, 6) and (5, 1, 1, 10). The MERs (4, 3, 2, 6) and (5, 1, 1, 10) are duplicate MERs that are generated twice. In order to improve algorithm efficiency, we

enhance Basic SLA to avoid generating duplicate MERs to get Enhanced SLA shown in Algorithm 2. Instead of scanning each MKE in sequence, i.e., moving top and bottom of each MKE independently, we move them for all MKEs on a Scan Line simultaneously. If the tops and bottoms of multiple MKEs overlap with each other, then the subsequent steps of scanning these MKEs will be identical, and we only need to continue the scanning process for one of the MKEs. By avoiding any redundant scanning process, we improve efficiency of the algorithm.

Algorithm 2 Enhanced SLA for finding MERs based on all MKEs on Scan Line at Column i .
 Inputs: FPGA area matrix M and a Scan Line at Column i .
 Outputs: MERs generated by all MKEs on the Scan Line at Column i .

```

Begin Enhanced SLA
Store all MKEs on the Scan Line into an array Keys[]
Kmax ( MAX(Keys[]))
for w = Kmax to 1 do
Take MKE at (i, j) in Keys[] with value M[i][j] _ w;
Move its top and bottom with length w until “stuck”;
if top or bottom changed or w = M[i][j] then
Record MER;
end if
if tops and bottoms of two or more MKEs overlap then
Keep one of the MKEs in Keys[] and delete others;
end if
end for
End Enhanced SLA
    
```

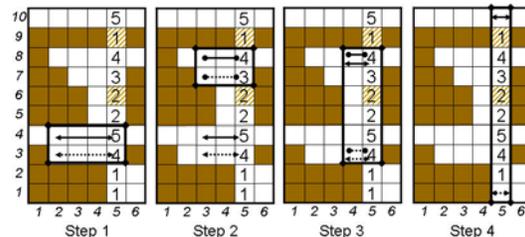


Figure 4. Steps of Enhanced SLA with MKEs at (5,3) and (5,7), and Scan Line at Column 5.

An example is shown in Figure 4, where MKEs at (5,3) and (5,7), and $M(5, 3) = 4, M(5, 7) = 3$. So $Kmax = \max(4, 3) = 4$. Figure 5 can be viewed as combination of Figures 2 and 3 after removing redundant steps that generate duplicate MERs.

The algorithm runs in the following steps:

- Step 1: $w = 4$, top = 4, bottom = 3 for MKE at (5,3), record MER(2, 3, 4, 2).
- Step 2: $w = 3$, top = 8, bottom = 7 for MKE at (5,7), record MER(3, 7, 3, 2).
- Step 3: $w = 2$, top = 8, bottom = 3 for both MKEs at (5,3) and (5,7), record MER(4, 3, 2, 6). Since the tops of MKE(5, 3) and MKE(5, 7) are both moved to Row 8, and their bottoms are both moved to Row 3, one of the MKEs, say (5,7), is deleted from Keys.
- Step 4: $w = 1$, top = 10, bottom = 1 for MKE at (5,3), record MER(5, 1, 1, 10).

V. Online SLA

When a task is added to or removed from the FPGA at runtime, it is likely, although not always true, that only parts of the FPGA area are affected and need to be scanned again to update the set of MERs. We take advantage of this observation to design the Online Scan line Algorithm (Online SLA) that

selectively updates the local region affected by a task's addition or removal instead of the entire FPGA area. This optimization improves algorithm efficiency significantly, as we will show in the performance evaluation section.

The Update Interval is defined as the horizontal interval $[L,R]$ containing all Scan Lines that need to be re-scanned using Enhanced SLA in order to update the set of MERs upon task addition or removal. Taking task addition as an example. When a new task with position (x, y,w, h) is placed on the FPGA, two new Scan Lines are added at Columns $x-1$ and $x+w-1$. Column $x-1$ is the left edge of the Update Interval L , that is, we do not need to check any Scan Lines lying on its left-hand side in order to update the set of MERs. However, Column $x+w-1$ is not the right edge of the Update Interval, and we do need to check Scan Lines lying to its right-hand side. Intuitively, R is the rightmost Scan Line that can be "seen" by the newly added task without getting blocked by other tasks in-between.

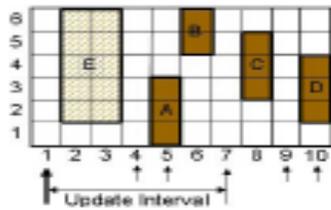


Figure 5. Example illustrating the concept of Update Interval.

As an example, Figure 5 shows a 10×6 FPGA with 4 tasks running initially, i.e., A, B, C and D. There are five Scan Lines at Columns 4, 5, 7, 9 and 10, indicated by the arrows. When a new task E is added, a new Scan Line at Column 1 is added, as indicated by the bold arrow. But we do not need to scan all six Scan Lines at Columns 1, 4, 5, 7, 9 and 10. Instead, we only need to scan those in the Update Interval, that is, those at Columns 1, 4, 5 and 7. Intuitively, propagation of the impact of the addition of task E is blocked by the three tasks A, B and C, which completely eclipse task E on the vertical dimension, so task E cannot "see" task D. Therefore, we do not need to re-run Enhanced SLA to calculate MERs whose right edges fall on Columns 9 and 10. Note that even though the task set (A, B, D) also completely eclipse task E, the task set (A, B, C) results in a smaller Update Interval, and we take the minimum Update Interval as our result.

Algorithm 3 Online SLA for incremental updating the set of MERs upon task addition or removal.

Inputs: FPGA area matrix M and location of the newly added or removed task.

Outputs: MERs deleted or generated due to task addition or removal.

```

Begin Online SLA
Update area matrix  $M$ ;
Get Update Interval  $[L,R]$ ;
for each Scan Line at Column  $i$  in  $[L,R]$  do
Delete MERs whose right edges fall on Column  $i$ ;
Generate MERs using Enhanced SLA;
end for
End Online SLA
    
```

As shown in Algorithm 3, upon addition or removal of a task, MERs whose right edges fall within the Update Interval are removed from the MERs set, and Enhanced SLA is run for each Scan Line in the Update Interval to update the set of MERs. We use a hash table as the data structure to store the set of MERs, using the horizontal positions of their right edges as the hash keys, so that the MERs with the same right edge can be added or deleted efficiently. Next, we analyze the complexity of Online

SLA. It takes time $O(H)$ to find all the MKEs of a Scan Line. When generating MERs from each Scan Line, the total number of movements of tops and bottoms does not exceed H , so the time complexity of Enhanced SLA is $O(H)$. If there are n scan lines, then the time complexity of Online SLA is $O(nH)$. Since the maximum number of Scan Lines is W , the worst case time complexity of Online SLA is $O(WH)$. Simulation Experiments indicate that the average performance is actually much better than the worst case, mainly due to incremental update of local regions in Online SLA.

VI. Related Work

Handa et al [2] presented an efficient algorithm for finding all MERs called the Staircase Algorithm, which works by first finding all the maximal staircases i.e., those that contain at least one MER, and then extracting the MERs from them. Even though both algorithms have the same worst-case complexity, simulation results indicate that Online SLA has better average performance in terms of algorithm running time than the Staircase Algorithm.

Bazargan et al. [1] presented an algorithm for managing the free space on a FPGA area by keeping track of non-overlapping rectangles, and using heuristics to reduce the number of rectangles considered when updating the rectangle list. As discussed in Section 1, this approach is inferior to the MER approach since it sometimes rejects a task even though there is enough empty space on the FPGA. Walder et al. [3] improved upon Bazargan's algorithm by delaying the decision about whether to split a rectangle on the vertical or horizontal direction. They also presented a data structure based on hash matrices for placing a task in constant time. Ahmadinia et al. [4] presented a variant of Bazargan's algorithm for managing the occupied area instead of the free area, in order to reduce the number of rectangles that need to be stored.

VII. Performance Evaluation

We conducted some simulation experiments to evaluate the performance of Online SLA, Enhanced SLA and the Staircase Algorithm [2]. It is not within the scope of this paper to evaluate an end-to-end online scheduling algorithm, which may include admission control, task queuing, priority assignment and task placement. Instead, we are only concerned with a small component of the overall online scheduling problem, i.e., finding the complete set of MERs on a partially occupied FPGA. Therefore, we aim for simplicity when designing the simulation experiments. During each simulation run, tasks are queued and processed in FIFO order, i.e., if a task T at the head of the queue cannot be placed on the FPGA, all other tasks in the queue must wait until T is placed when some running tasks finish execution and are removed from the FPGA. We use First-Fit as the strategy to choose a large-enough MER to place a task, and place the task on the lower-left corner of the chosen MER. Each task's arrival time is a random value between 0 and an upper bound U . We control the workload, and in turn the FPGA area utilization, by using different upper bound U . A smaller U means that the tasks arrivals are more frequent, and the FPGA area utilization is higher. If all tasks arrive in the time interval $[0,100]$, then the last task finishes execution at time 176, and 79.1% of the FPGA area is occupied on average during the simulation run. Online SLA has the best performance due to its incremental update approach, but the difference among the all algorithms are not dramatic. We can also observe that Online SLA performs better under heavy load or light load than medium load. This can be explained as follows: under light load, the number of Scan Lines is smaller; under heavy load, the distance traversed by the top and bottom lines in Algorithm 2 and the Update Interval in Algorithm 3 are both smaller. All of these factors contribute to the reduced running time of Online SLA.

VII. Conclusion

In this paper, we have presented an efficient algorithm for finding the complete set of MERs in a given FPGA area, which is useful for online placement and scheduling of HW tasks. This is only one step in the overall process of online task scheduling or FPGAs. As part of our future work, we plan to investigate other placement algorithms and their interaction with allocation and scheduling, fragmentation problems and implement these algorithms in an actual OS for a combined CPU/FPGA device.

References

- [1] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast Template Placement for Reconfigurable Computing Systems." IEEE Design & Test of Computers, vol. 17, no. 1, pp. 68–83, 2000. Xilinx Virtex-4 Family Overview. Preliminary Specification ds112, Tech. Rep., Xilinx Inc., January 2005.
- [2] M. Handa and R. Vemuri, "An efficient algorithm for finding empty space for online FPGA placement." in DAC, 2004, pp. 960–965.
- [3] K. Bazargan, R. Kastner, M. Sarrafzadeh, Fast template placement for reconfigurable computing systems, IEEE Design & Test of Computers 17 (1)(2000) 68–83.
- [4] A. Ahmadiania, C. Bobda, M. Bednara, and J. Teich, "A New Approach for On-line Placement on Reconfigurable Devices." in IPDPS, 2004, pp. 134–140.
- [5] J. Cui, Q. Deng, X. He, Z. Gu, An efficient algorithm for online management of 2D area of partially reconfigurable FPGAs, in: R. Lauwereins, J. Madsen (Eds.), Design Automation and Test in Europe Conference and Exposition (DATE), ACM, 2007, pp. 129–134.
- [6] M. Handa, R. Vemuri, An efficient algorithm for finding empty space for online FPGA placement, in: DAC, 2004, pp. 960–965.