Architecture-Aware Reconfiguration-Centric Floorplanning for Partial Reconfiguration

Kizheppatt Vipin and Suhaib A. Fahmy

School of Computer Engineering, Nanyang Technological University,
Nanyang Avenue, Singapore
{vipin2,sfahmy}@ntu.edu.sg

Abstract. Partial reconfiguration (PR) has enabled the adoption of FPGAs in state of the art adaptive applications. Current PR tools require the designer to perform manual floorplanning, which requires knowledge of the physical architecture of FPGAs and an understanding of how to floorplan for optimal performance and area. This has lead to PR remaining a specialist skill and made it less attractive to high level system designers. In this paper we introduce a technique which can be incorporated into the existing tool flow that overcomes the need for manual floorplanning for PR designs. It takes into account overheads generated due to PR as well as the architecture of the latest FPGAs. This results in a floorplan that is efficient for PR systems, where reconfiguration time and area should be minimised.

1 Introduction

Partial reconfiguration (PR) makes use of the fact that the functionality implemented in an FPGA can be altered by selectively modifying the contents of part of the configuration memory, while the remaining portions continue to operate. Although PR has several advantages, it also entails a number of design challenges. One of these is the floorplanning of reconfigurable regions.

For standard static FPGA design, floorplanning is generally only of interest to expert designers trying to highly optimise a design. The tools available from vendors are sufficiently versatile to perform area-constrained placement and routing, while achieving timing closure. Current PR tools do not perform floorplanning automatically, and require considerable input from the designer. The designer is expected to have knowledge about the physical architecture of the target FPGA, as well as the details of the PR process and the run-time costs associated with it, if they are to come up with an efficient floorplan. Manual floorplanning based on these factors is cumbersome and often leads to sub-optimal results and consumes a large amount of design time. This floorplanning requirement has made PR less attractive to adaptive system designers.

In this paper we propose methods, which can help system engineers adopt PR without the need for manual floorplanning. The tool we propose can be integrated into the existing FPGA tool chain. We are interested primarily in adaptive applications where reconfiguration occurs at the module level, and the sequence of configurations is unknown up front. We consider the overheads associated with PR as well as the characteristics of target devices. We are interested in recent families of FPGAs such as the Xilinx Virtex-5 and Virtex-6, which are highly heterogeneous in nature, include embedded processors and transceivers, and comprise an irregular arrangement of DSP and Block RAM columns. For PR applications, we are typically concerned with reducing reconfiguration time and area. An intelligent arrangement and allocation of PR regions can result in reduced area and hence allow designs to fit on smaller devices. Cost functions are used that take into account several factors such as resource wastage and reconfiguration time, rather than ASIC floorplanning metrics.

The remainder of the paper is organised as follows: Section 2 discusses related work, Section 3 presents background on the PR process, Section 4 introduces the proposed floorplanning approach, Section 5 presents experimental results and Section 6 concludes the paper.

2 Related Work

Several approaches to FPGA floorplanning have been published, although work related to floorplanning for PR is less abundant. Traditionally, FPGA floorplanning is considered as a fixed-outline floorplanning problem, as introduced in [1] and further extended in [3]. The authors present a resource-aware fixed-outline simulated-annealing and constrained floorplanning technique. Their formulation can be applied to heterogeneous FPGAs but the resulting floorplan may contain irregular shapes, which are not allowed in current PR designs. Another interesting study is presented in [9], which presents an algorithm called "Less Flexible First (LFF)". In order to perform placement, the authors define the flexibility of the placement space as well as the modules to be placed. A cost function is derived in terms of flexibility and a greedy algorithm is used to place modules. The generated floorplan will have only rectangular shapes, but the approach only addresses older-generation FPGAs and is unsuitable for recent families due to their heterogeneous resources.

Findings in [2] are based on slicing trees. Using this method it can be ensured that the floorplan contains only rectangular shapes. Here, the authors assume that the entire FPGA fabric is composed of a repeating basic tile, which contains all types of FPGA resources including Configurable Logic Blocks (CLBs), Block RAMs and DSP slices. Although this assumption is valid for older-generation FPGAs, such as the Xilinx Spartan-3, more recent FPGAs such as the Xilinx Virtex-5 family, do not have such a repeated tile architecture.

Yuh et al. have published two methods for performing floorplanning for PR. One method is based on using a T-tree formulation [10] and the other is based on a 3D-sub-Transitive Closure Graph (3D-subTCG) [11]. Using T-trees, each reconfigurable operation is represented as a 3D-box, with its width and depth representing the physical dimensions and its height being the execution time required for the operation. Here the reconfiguration operations are at task level rather than functional level and the authors consider older-generation Virtex FPGAs, which require columnar reconfiguration.

Montone et al. present a reconfiguration-aware "floorplacer" in [4]. They consider the latest architecture of Virtex-5 FPGAs. Initially the design is divided into reconfiguration areas based on the minimisation of temporal variance of the resource requirement. Then a floorplacer tries to minimise the area slack using simulated-annealing. In [5] a floorplanning method based on sequence pairs is presented. In this work, authors have shown how sequence pairs can be used to represent multiple designs together. Here, designs are the circuitry present in the FPGA at different instances. An objective function tries to maximise the common areas between designs and simulated-annealing is used for optimisation. Although simulated-annealing-based floorplanners have been developed, for soft modules, which are common in PR designs, the results are not satisfactory [12].

All existing work we have found focuses on the static properties of a particular placement. Hence, the placement is not optimised for the dynamic behaviour of a partially reconfigurable system. Other work relies on fixed task-graphs and hence only optimises for a fixed sequence of configurations. This paper presents an approach that optimises the runtime properties by finding a placement that results in the lowest possible reconfiguration time, considering the lowest level granularity of heterogeneous resources on modern FPGAs, for designs where the adaptation is at a functional level and hence unpredictable.

3 PR Floorplanning Considerations

In this section we develop a device model and explore the factors to be considered while designing an efficient floorplanner for PR. The limitations of several existing methods will be also explained.

3.1 Architecture Considerations

For efficient floorplanning, the tool should be aware of the FPGA architecture and special requirements arising due to PR. Xilinx Virtex-5 FPGAs are divided into rows and columns. The number of rows in a device depends upon the size of the device. The smallest configurable unit is a *frame*. A *frame* is one bit wide and spans an entire device row [8]. Resources such as CLBs, Block RAMs etc. are arranged in a columnar fashion extending the full height of the device and are referred to as *blocks*. A tile is one row high and one block wide, and contains a single type of resource, as shown in Fig. 1. One CLB tile contains 20 CLBs, one DSP tile contains 8 DSP Slices, and one BRAM tile contains 4 Block RAMs arranged vertically. A CLB tile contains 36 *frames*, a DSP tile 28 and a Block RAM tile 30 *frames* arranged side by side. The data size of a $frame$ is 164 bytes. Embedded processors and transceivers are arranged along device row boundaries, but they obstruct the continuity of DSP and BRAM columns. Most existing floorplanners have not taken this device architecture into account, claiming only to use regular grid structures.

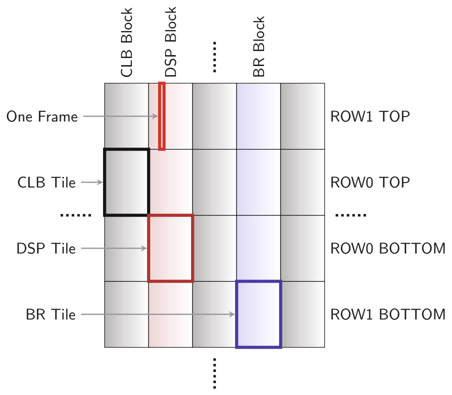


Fig. 1. Virtex 5 FPGA architecture

3.2 PR Operation

Partial reconfiguration is performed by modifying the circuitry implemented in the PR regions. Any modification in a region requires the full reconfiguration of the corresponding region. For accurate calculation of efficient PR schemes, the reconfigurable regions should be considered in terms of tiles since configuration must occur on a per tile basis. To use regions with incomplete tile boundaries, extra circuitry is required to read, modify, and write configuration information, resulting in increased area and latency. Reconfigurable regions must always be rectangular in shape. Since each tile is one device row high, the height of reconfigurable regions is an integer multiple of device rows. Partial bitstreams are loaded into the FPGA with the help of an Internal Configuration Access Port (ICAP), which is usually controlled by an embedded processor. The size of the bitstream, and hence the reconfiguration time of a region, is directly proportional to the total area of the region, irrespective of how many resources present in the region are actually utilised.

3.3 Required Reconfigurable Area

An important factor in floorplanning a reconfigurable region is its area. At runtime, reconfigurable regions implement different functional instances at various points in time. These functional instances are called *modes*. The required area (A_{ra}), in frames, for a PR region is the net area required for implementing all the modes assigned to it. This area is calculated by taking the maximal resource requirement for each resource type, in tiles, and multiplying by the number of frames needed for each tile of that type: Note that there is some overhead in this resource requirement due to it being based on whole tiles.

$$A_{ra} = \sum_j W_j * N_j, \qquad j \,\epsilon\, CLB, DSP, BlockRAM. \tag{1}$$

where W_j is the number of *frames* per type of tile j and N_j is the number of tiles of type j needed.

3.4 Actual Reconfigurable Area

When a design is placed, the actual area may differ from the initial requirement due to the rectangular shape requirement for PR regions or the disparate arrangement of resources on the FPGA fabric. Mathematically, the actual area (A_{aa}) of a region is calculated as

$$A_{aa} = \sum_i W_i * N_i, \qquad i \epsilon\, CLB, DSP, BlockRAM. \tag{2}$$

where W_i is the number of *frames* per tile of type i and N_i is the number of tiles of type i allocated in the region. The result is the number of frames to configure the placed region.

3.5 Resource Wastage

The resource wastage for a particular placement of a reconfigurable region (A_{rw}) is the difference between the actual area and the required area of that region, in frames. The total resource wastage of a full floorplan (A_{tw}) is the sum of resource wastage among all the regions.

$$A_{rw} = A_{aa} - A_{ra}. \tag{3}$$

$$A_{tw} = \sum_r A_{rw}. \tag{4}$$

While floorplanning partial regions, the floorplanner should try to minimise the total resource wastage in order to minimise reconfiguration time and maximise the resources available for implementing static logic.

3.6 Wirelength

Total wirelength is an important parameter in determining the effectiveness of floorplanning. Here we consider the Manhattan distance between regions and the total wirelength between two regions is calculated as the product of the Manhattan distance between them and the number of wires connecting them. Several previous researches consider total Half Perimeter Wire Length (HPWL) as the minimisation objective function for their floorplanner. Practically, HPWL has very little impact in FPGA floorplanning. In ASIC floorplanning, HPWL gives a figure of compactness of cells and hence the best timing achievable, but in FPGAs, where all resources as well as routing between them are fixed, HPWL does not give an accurate measure of timing performance.

3.7 Static Logic

Static logic is the area of the FPGA with fixed functionality. I/O pins are always assigned to the static region, since assigning I/O pins to reconfigurable regions may cause undesirable switching during reconfiguration. There is no restriction on the shape of static logic. To make optimal use of FPGA resources, and achieve timing closure, it is better not to restrict the shape of static logic or allocate a special location for it. The reconfigurable regions should be floorplanned in such a way that, the area available for the implementation of static logic is maximised, and it should be floorplanned after the PR regions.

4 Proposed Floorplanner

The input to our proposed floorplanner is the partition information of reconfigurable regions and their connectivity information. A connectivity matrix is used, whose element (i, j) represents the number of nets between region i and region j. The output of the floorplanner is a set of area constraints, which specify the coordinates of the bottom left and top right corners of each region. These constraints can be used for generating the *user constraints* file, which will be used by the vendor specific place and route tool for generating the final configuration bitstreams. The floorplanning problem can be formulated as follows:

Given:

- M regions with resource requirement 3-tuple, (n_{CLB}, n_{BR} and n_{DSP}) for each region,
- an FPGA of width W and height H,
- with N_{CLB}, N_{BR} and N_{DSP} resources available,
- and R device rows,

partition the FPGA into M rectangles, so that:

- each region can be mapped into a rectangle, which contains sufficient resources,
- rectangle height being an integer multiple of device row,
- no rectangles overlapping,
- minimising the cost function.

The outputs are the (x_{min},y_{min}) and (x_{max},y_{max}) coordinates of each rectangle so that $0 \leq x_{min} \leq x_{max} \leq W$ and $0 \leq y_{min} \leq y_{max} \leq H$.

4.1 Columnar Kernel Tessellation

Mapping an area directly using FPGA primitives is not practical, due to number of factors such as the large search space, limited number of available primitives in the FPGA, fixed primitive locations, rectangular shape region constraint, etc. Hence we adopt a new method called Columnar Kernel Tessellation. A kernel is a structure one device row high, containing FPGA primitives, which can be repeated in the vertical direction to satisfy a region's resource requirements. The availability of kernels for floorplanning a region changes based on the floorplanning of previous regions. The smallest kernel is a single tile. Each tile can be clustered with nearby tiles and can form new kernels.

The first step of floorplanning is to calculate the resource usage of each region in terms of reconfigurable tiles. For this purpose, the input resource utilisation

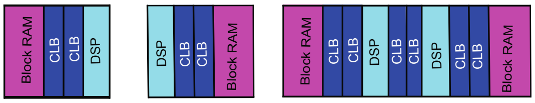


Fig. 2. Two Block RAM-DSP kernels and a merged kernel

values are divided by the corresponding number of resources available in a tile. This may result in some overhead if the resources needed do not use a whole tile. For example in Virtex-5 FPGAs, the required number of CLBs will be divided by 20, DSPs by 8, and Block RAMs by 4. The floorplanner maintains a database of FPGA architectures that contains information about the resource type of each device column. The different types of columns are mapped to a single co-ordinate system for better management. Each tile in the FPGA is encoded using a data-structure with information including location, resource type, used or not, and availability. Once a tile is used to floorplan a region, its *use field* is set to true. The tiles belonging to the locations of hard processors and transceivers are set to be unavailable.

In order to perform floorplanning, the regions are initially sorted according to descending resource requirements, into a floorplanning schedule. Regions are selected based on the following ordered criteria.

1. Require both DSP as well as Block RAM tiles,
2. Require DSP and CLB tiles,
3. Require Block RAM and CLB tiles,
4. Require CLBs tiles alone.

This classification is based on the fact that DSP tiles are the least available and hence the most precious FPGA resource. Block RAM tiles are weighted next and CLB tiles are the most abundant resource available, and so given the least weighting. Regions belonging to each group are sorted in descending order of DSP, Block RAM and CLB tiles required. Regions are selected from the scheduling list in sequential order and packed.

The floorplanner starts with regions which use both DSP tiles and Block RAM tiles. The floorplanner selects a kernel, which contains both DSP and Block RAM tiles. To generate kernels, the resource column information from the database is utilised. For each DSP column, the nearest Block RAM column location is calculated. The nearest tiles of DSP and Block RAM along with the tiles between them are merged to create kernels. These kernels are merged again and larger kernels are created. This operation is illustrated in Fig. 2 in the case of a Virtex-5 FX70T FPGA. It is to note that when kernels are merged, the CLB tiles in between them are also included in the resulting kernel. All kernels are one device row high. From the set of available kernels, the kernel with smallest size is chosen and used for packing. Example calculation of kernel size is shown in Fig. 3. Kernels are repeated in a columnar direction to meet the region's resource requirements. The minimum number of kernels required for packing is equal to the number of DSP tiles required divided by the number of DSP tiles in the

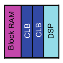
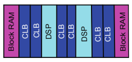
Kernel	#DSP tiles	#BR tiles	#CLB tiles	#Frames	#Bytes
	1	1	2	28 + 30 + 36 = 94	94 * 164 = 15416
	2	2	6	2*28 + 2*30 + 6*36 = 332	332 * 164 = 54448

Fig. 3. Example calculation of the size of kernel

kernel. The maximum number of kernels that can be used to satisfy the DSP resource requirement is equal to the number of device rows, i.e. the full device height, since the height of a kernel is one device row. If the arrangement of a kernel cannot meet the required number of DSP tiles, that kernel is discarded and the kernel with next lowest resource requirement is selected and used for packing.

Once the DSP-BR kernels are packed, the remaining BR and CLB resources required for that region are calculated. If more Block RAMs are needed, the nearest Block RAM column is selected from the database and used. Preference is given to columns towards the right and left edges of the FPGA in order to maximise free space available towards the centre of the FPGA. If more CLB tiles need to be allocated, CLB columns towards the device edge are selected and allocated. Once the allocation is performed, the tiles which are used are marked in the database as *used*.

Now the regions which use only DSP and CLB tiles are packed. For this purpose, the kernels considered contain only DSP tiles and CLB tiles. The minimum number of kernels required for packing will be equal to the number of DSP tiles required divided by the number of DSP tiles in the kernel. Tiles which are not marked as *used* or *not available* are used to generate the new set of kernels. This same process is followed once more for regions containing Block RAMs. Finally, regions containing only CLB tiles are packed.

The inherent rectangular shape of kernels and the columnar repetition guarantees that the allocated area for each region will be of rectangular shape and region height will be an integer multiple of device rows. The floorplanner follows a divide and conquer method. The packing of each region reduces the search space for implementing subsequent regions as well as the number of kernels available. The algorithm runs a number of times, each time starting with a different kernel for packing. The number of iterations can be specified or can be stopped when a required cost objective is met. At the end of each complete packing, a cost function is evaluated for the floorplan . The cost function is defined as

$$CF = \alpha * A_{tw} + \beta * WL. \quad (5)$$

where A_{tw} is the total resource wastage and WL is the total wirelength between regions. α and β are weight factors with $\alpha > \beta$. For designs where reconfiguration time is highly critical compared to speed of operation, the value of β can be set to zero and for applications where timing is highly critical rather than

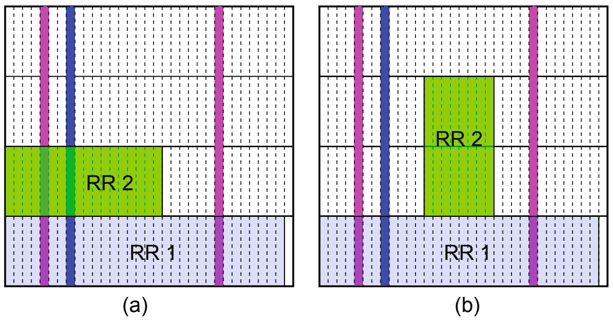


Fig. 4. (a) Resulting floorplan from [4], (b) Resulting floorplan from our method

reconfiguration time α can be set to zero. For other applications, the value of α and β are weighted accordingly.

At the end of each complete floorplan generation, a post processing step is performed, in which the regions are moved along the columnar direction towards the middle of the columns. If this movement improves wirelength, the movement is accepted otherwise it is rejected. Also, regions that occupy the same columns are swapped and wirelength is recalculated. This move is also accepted only if it improves wirelength. These moves are a type of pseudo-simulated-annealing. This is possible due to the fact that the resources are arranged in columnar fashion in the FPGA, and moving a region along its columns does not affect resource availability, provided there are no unavailable tiles in the direction of movement.

5 Case Study

A direct comparison of our method with existing methods is not possible due to the unavailability of other floorplanning tools and uniform benchmark circuits. Hence, we use a reported case study, taken from [4], and compared the results using our method. The system implemented consists of a CAN controller, Floating Point Unit (FPU), FIR filter, CRC controller and an Ethernet controller. Based on the partitioning algorithm, modules are partitioned into two reconfigurable regions. The CAN controller, FPU and CRC are implemented in reconfigurable region 1 (RR 1) and FIR filter and Ethernet controller are implemented in region 2 (RR2), as per [4]. The design is implemented in a Virtex-5 LX30T device. Region 1 requires 24% of the available CLB and 5 Block RAMs and region 2 requires 13% of CLBs. The static region requires 61% of CLBs and 40 Block RAMs. The resulting floorplan reported in the paper is given in Fig. 4.a. It is clear that although region 2 does not require Block RAMs or DSP slices, the resulting floorplan includes these resources. This leads to increased region size, higher configuration time and additional storage requirement. Furthermore, these resources cannot be used elsewhere in the design. This floorplan uses a total of 1766 frames.

A floorplan determined by our method is shown in Fig. 4.b. Region 2 is floorplanned in such a way that no DSP slices and Block RAMs are used. Hence our method uses 58 fewer frames and reserves more resources for static logic implementation. The smaller size of the region also contributes to 18.4 KB (9.2×2 since there are two partial bitstreams for that region) less storage requirement and a corresponding improvement in reconfiguration time.

For a more complex investigation, we could find no existing work to compare to, nor standard tools to use, so we floorplanned an in-house design using our proposed method and compared it to an ad-hoc floorplan based on previous experience with some optimisation effort. The ad-hoc floorplanning is done as per Xilinx PR floorplanning guidelines, with the help of the PlanAhead software. The selected design is a software defined radio (SDR) targeted for Xilinx Virtex-5 FX70T FPGA. The SDR chain consists of a matched filter, carrier recovery circuit, demodulator, signal decoder and video decoder. Each module has a number of *modes* with different resource requirements. *Modes* are mutually exclusive implementations of the module with the same set of inputs and outputs. Partitioning of *modes* into regions is beyond the scope of this paper, but is explained in our previous work [6]. Here we assume each module is assigned to a single region, and hence the resource requirement of each region is the requirement of the largest *mode* of the module assigned to it. Here, all modules are connected in sequential order with a 64 bit wide bus. The static logic contains a PowerPC-440 embedded processor, external memory interface and an ICAP controller. The different regions and associated resource requirements are given in Table 1. The *rq′d* field indicates the exact number of resources required, the *tiles* field indicates the required number of tiles needed to satisfy the resource requirement and the *waste* field indicates the resources wasted due to rounding the resources to tiles. The total number of frames wasted due to the tiling operation is roughly 115 frames.

Table 1. Resource utilisation for reconfigurable regions

Region	CLBs			BRs			DSPs			No. of Frames
	Rq'd	Tiles	Waste	Rq'd	Tiles	Waste	Rq'd	Tiles	Waste	
Matched Filter	500	25	0	0	0	0	34	5	6	1040
Carrier Recovery	123	7	17	0	0	0	8	1	0	280
Demodulator	97	5	3	8	2	0	0	0	0	240
Decoder	234	12	6	2	1	2	0	0	0	462
Video Decoder	1100	55	0	6	2	2	34	5	6	2180
Total	2054	104	26	16	5	4	76	11	12	4202

As per the description in Section 4.1, the order for floorplanning regions is the video decoder first, followed by the matched filter, carrier recovery, demodulator and finally the decoder. The result of the ad-hoc floorplanning and 5 of the best floorplans using our method are given in Table 2.

Table 2. Resource wastage and total wirelength for different floorplans

Plan No.	Wastage, A_{tw} (frames)	Wirelength, WL (Normalised)
Adhoc	956	7420
Plan1	466	8640
Plan2	486	9056
Plan3	592	11776
Plan4	516	7392
Plan5	556	9120

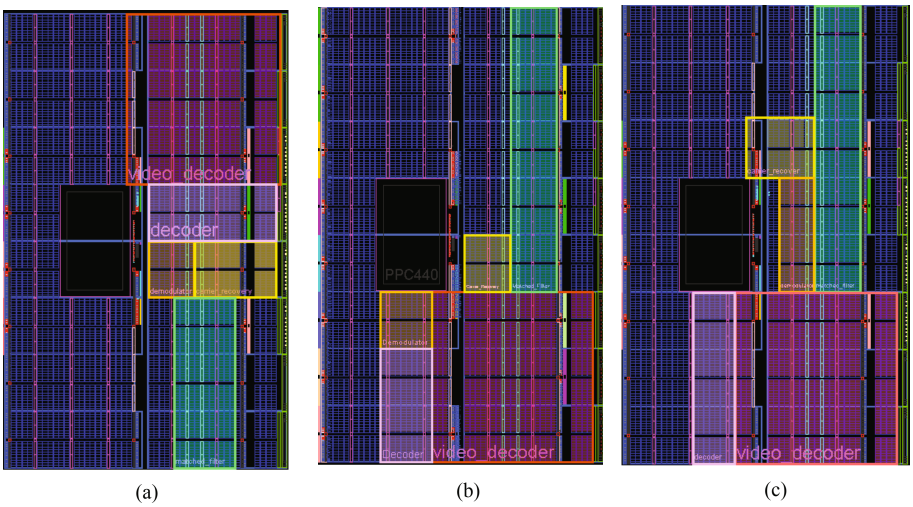


Fig. 5. Floorplans using (a) An ad-hoc approach, (b) proposed method with minimum resource wastage, (c) with minimum wirelength

There is no fixed relationship between the resource wastage and wirelength, owing to the rectangular shape requirement of the reconfigurable regions as well as the disparate arrangement of resources. The ad-hoc plan, the plan which produces minimum resource wastage (Plan1) and the plan which gives minimum wirelength (Plan4) are shown in Fig. 5. When the value of α is set to zero in the cost function, Plan 4 is the preferred floorplan, and when β is set to zero, Plan 1 gives the best results. We can see that the proposed floorplanner performs well on area: all the floorplans have lower resource wastage from 38% to 51%, which corresponds to a decrease in reconfiguration from 7% to 9.5% compared to the ad-hoc plan. Since the modules of the regions are in a continuous chain, the ad-hoc method is able to achieve good total wirelength. In usual FPGA floorplanning without PR, the ad-hoc floorplan is fairly acceptable, since all the resource requirements are satisfied and wirelength is minimised. But for PR designs, the resource wastage creates a considerable overhead in terms of reconfiguration time. Moreover, storing 956 frames of configuration frames requires 153 KB extra storage memory for each system configuration.

These results demonstrate the advantage of considering the required implementation metrics in floorplanning. While static designs only require the floorplan to fit and achieve timing, a PR design's reconfiguration time is also affected by the floorplan. Our approach ensures this is factored into the floorplanning process, and results in savings as shown.

6 Conclusion

In this paper we introduced a novel method for PR design floorplaning. The method described is fully compatible with the latest vendor-supported PR toolflows. The heterogeneous and irregular architecture of modern FPGA families are considered and floorplanning cost functions tailored for PR are introduced. Our floorplanning method is portable to older FPGAs by considering them as having a single device row. Our study proves that it is possible to optimise the area requirement considering the tile constraint. We have also found that a significant area overhead is generated due to the tiling requirement of reconfigurable regions. Hence, considering tiling at the partitioning stage, prior to floorplanning may yield more efficient designs. We intend to integrate this floorplanner with our previous work on efficiently partitioning modules into reconfigurable regions. The partitioning tool and floorplanner along with the vendor supplied synthesis and place and route tools can essentially automate most of the PR design flow, which is our long term goal. [7] This will lead to wider adoption of partial reconfiguration and development of more efficient and effective adaptive systems.

References

1. Adya, S., Markov, I.: Fixed-outline floorplanning through better local search. In: Proceedings of ACM/IEEE International Conference on Computer Design (2001)
2. Banerjee, P., Sangtani, M., Sur-Kolay, S.: Floorplanning for partially reconfigurable FPGAs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30(1), 8–17 (2011)
3. Feng, Y., Mehta, D.: Heterogeneous floorplanning for FPGAs. In: Proceedings of International Conference on VLSI Design (2006)
4. Montone, A., Santambrogio, M., Sciuto, D., Memik, S.: Placement and floorplanning in dynamically reconfigurable FPGAs. ACM Transactions on Reconfigurable Technology and Systems 3(4), 24:11–24:34 (2010)
5. Singhal, L., Bozorgzadeh, E.: Multi-layer floorplanning for reconfigurable designs. IET Computers & Digital Techniques 1(4), 276–294 (2007)
6. Vipin, K., Fahmy, S.: Efficient region allocation for adaptive partial reconfiguration. In: Proceedings of the International Conference on Field Programmable Technology, FPT (2011)
7. Vipin, K., Fahmy, S.: Enabling high level design of adaptive systems with partial reconfiguration. In: Proceedings of the International Conference on Field Programmable Technology, FPT (2011)

8. Xilinx Inc.: UG191: Virtex-5 FPGA Configuration User Guide (2010)
9. Yuan, J., Dong, S., Hong, X., Wu, Y.: LFF algorithm for heterogeneous FPGA floorplanning. In: Proceedings of Asia and South Pacific Design Automation Conference, ASP-DAC (2005)
10. Yuh, P., Yang, C., Chang, Y.: Temporal floorplanning using the T-tree formulation. In: Proceedings of IEEE/ACM International Conference on Computer Aided Design, ICCAD (2004)
11. Yuh, P., Yang, C., Chang, Y., Chen, H.: Temporal floorplanning using 3D-subTCG. In: Proceedings of Asia and South Pacific Design Automation Conference, ASP-DAC (2004)
12. Zhan, Y., Feng, Y., Sapatnekar, S.: A fixed-die floorplanning algorithm using an analytical approach. In: Proceedings of Asia and South Pacific Design Automation Conference, ASP-DAC (2006)