Parallelizing FPGA Technology Mapping through Partitioning

Chuyu Shen†, Zili Lin†, Ping Fan‡, Xianglong Meng‡, and Weikang Qian†*
†University of Michigan-SJTU Joint Institute, Shanghai Jiao Tong University, Shanghai, China
‡Capital Microelectronics, Ltd., Beijing, China
*Email: qianwk@sjtu.edu.cn

***Abstract*—The traditional FPGA technology mapping flow is very time-consuming as modern FPGA designs become larger. To speed up this procedure, this paper proposes a new approach based on circuit partition to parallelize it. The idea is to split the original circuit into several sub-circuits and assign each one to a core of a multi-core processor for simultaneous technology mapping. Compared to other existing parallelization methods, our method has the benefit of being independent of the detailed mapping algorithm. Our proposed partition method is able to minimize the quality loss caused by the partitioning. We have successfully integrated the proposed approach into an industrial FPGA mapping platform. The proposed flow gains a speed-up of 1.6X on average on a quad-core processor with negligible influence on the LUT count and the critical path length.**

***Keywords*—FPGA, technology mapping, parallelization, circuit partitioning.**

I. Introduction

During the past 20 years, field-programmable gate array (FPAG) has become a widely used design platform due to its reconfigurability and low development cost. The basic logic units, i.e., look-up tables (LUTs), in the state-of-the-art FPGA chips have already reached million scale. For example, Xilinx Virtex UltraScale+ provides over 3,000,000 LUTs and Altera Stratix 10 offers 5,000,000 LUTs. The increasing number of units on a chip brings many benefits such as reducing the cost and enhancing the integration level so that more complicated applications can be implemented within one chip. However, it also raises challenges to computer-aided design (CAD) tools, which are used to synthesize the FPGA designs. Because the scale of the target design increases, the runtime of the design flow increases.

As multi-core processors become more prevalent today, parallel computing is applied in many different fields to improve runtime. The most time-consuming steps in the FPGA CAD flow are placement and routing. A number of methods have been proposed to parallelize these two steps [1], [2]. In this paper, we focus on another step in the flow – technology mapping. It also takes a large amount of time, because it has high complexity and is often called for many iterations to improve the quality of the final circuit [3]. Although several approaches have been proposed to parallelize the technology mapping [4], [5], they all focus on parallelizing some steps in the mapping algorithm, such as cut enumeration. Therefore, they depend on the mapping algorithm. Once the algorithm is changed, the parallel implementation should also be modified. In this paper, we propose a novel method to parallelize technology mapping based on circuit partition. As a consequence, it is independent of the mapping algorithm. The mapping engine is just called as a *black box* on different cores to map all the partitions simultaneously. After that, all mapped sub-circuits are merged together. However, since circuit partition reduces the global optimization opportunities, the quality of the final mapped netlist usually becomes worse. In this work, we develop a partition algorithm to minimize such quality loss. We have successfully integrated the proposed approach into an industrial FPGA mapping platform. The proposed flow gains a speed-up of 1.6X on average on a quad-core processor. It only leads to an average of 3.06% increase in LUT count and its influence on the critical path length is negligible.

The rest of the paper is organized as follows. In Section II, we describe the details of the proposed method. In Section III, we show the experimental results of our approach. Finally, in Section IV, we conclude the paper.

II. Methodology

In this section, we will show the details of our proposed method.

A. Overview and Problem Formulation

Since the bulk of the work in technology mapping is on combinational circuits, we focus on speeding up the technology mapping of combinational circuits. In this case, the inputs and outputs of the sequential units are treated as primary outputs (POs) and primary inputs (PIs) of the combinational circuit.

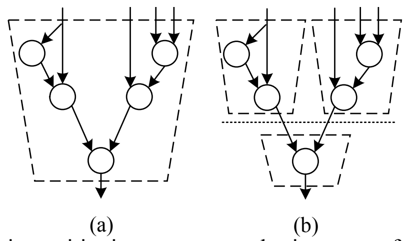

Fig. 1. Circuit partitioning can cause the increase of the area and level of the mapped netlist. (a) The result of mapping the entire netlist. (b) The result of mapping two sub-netlists cut by the dashed line.

As mentioned in Section I, the proposed parallelization flow is based on circuit partitioning. The basic idea of the flow is as follows. When given a gate netlist, instead of applying technology mapping directly, we first partition it into several sub-netlists. Then, we assign each sub-netlist to one core. In this way, we can do the mapping of multiple sub-netlists simultaneously. Finally, the mapped sub-circuits are merged into one piece. Since the runtime of technology mapping is proportional to the size of the circuit, the parallel mapping on the sub-circuits can save a lot of time.

However, if we partition the original gate netlist randomly, it will cause some quality loss on the final LUT netlist. For example, consider a gate netlist shown in Fig. 1, in which each node represents a gate. If we do the technology mapping on the whole circuit, 1 4-LUT is needed and the critical path length after mapping is 1. However, if we bi-partition the gate netlist as the dashed line shows, do the parallel mapping, and merge the result, 3 4-LUTs are needed. Furthermore, the critical path length of the LUT netlist increases by 1. In summary, both the LUT count and the critical path length of the circuit are worse than the one obtained by mapping the original circuit directly. For simplicity, in the following, we will call our method *the partition-based parallel mapping* and the direct mapping of the original circuit *the original mapping*. We will call LUT count *area* and critical path length *level*. In order to assure a good quality on the final circuit while improving the runtime, the partition-based parallel mapping algorithm should meet the following requirements:

1. The circuit obtained by applying the algorithm should have exactly the same function as the original gate netlist.
2. The algorithm is faster than the original mapping method.
3. The algorithm is independent of the underlying mapping algorithm.
4. The algorithm has small influence on the circuit quality, which is measured by area and level.

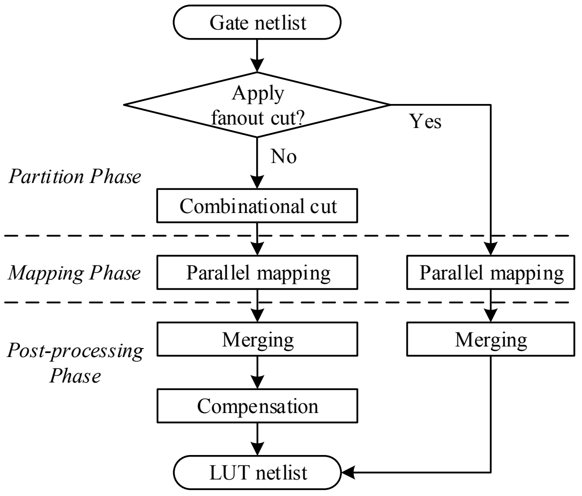

Fig. 2. Proposed partition-based parallel mapping flow.

To meet the above requirements, we have developed two partition methods which are integrated in our flow. The first one is called *fanout cut*. It causes no quality loss for the final circuit, but there are some restrictions in applying it. The second one is called *combinational cut*. It is applicable to all the circuits, but it will cause some quality loss. The whole flow of the proposed parallel mapping algorithm is shown in Fig. 2. After reading a gate netlist, we first apply fanout cut. If the circuit can be properly divided into several partitions, we then do the parallel mapping and merge the results back to the final LUT netlist. Otherwise, we apply the combinational cut. Since the combinational cut will lead to quality loss, a compensation step will be applied after merging the sub-circuits to reduce the loss. The details of fanout cut, combinational cut, and compensation are discussed in the following subsections.

B. Fanout Cut

The fanout cut partitions the circuit by only splitting those PIs with multiple fanouts, as illustrated in Fig. 3. However, it is only applicable when there is no connection between blocks A and B except at PIs, as shown in Fig. 3. In this case, the fanout cut is applied successfully and we obtain two partitions. On the other hand, if blocks A and B are still connected even after we split the PI with multiple fanouts, we cannot apply fanout cut and we have to further apply the combinational cut method.

For blocks A and B shown in Fig. 3, since they are disconnected except at PIs, the technology mapping of block A and that of block B are independent in the original mapping flow. Therefore, performing parallel mapping on them will not affect the mapping result. Thus, if fanout cut can be applied, it will not cause any quality loss.

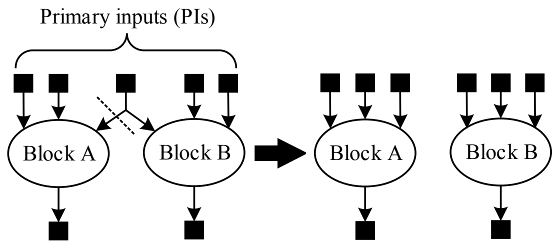

Fig. 3. Illustration of fanout cut.

It should be noted that even if splitting PIs with multiple fanouts can generate the required number of partitions, the sizes of the sub-circuits may be unbalanced. Since the runtime of parallel mapping is proportional to the size of the largest sub-circuit, the acceleration ratio of the flow will be significantly reduced if the size of one sub-circuit is extremely large. In this case, we also need to apply combinational cut.

C. Combinational Cut

As mentioned in Section II-A, if we cut the circuit randomly, both the area and the level of the final circuit are worse than the one obtained by the original mapping. The combinational cut is developed to minimize such quality loss. The cut method includes two strategies which aim at minimizing the area increase and the level increase, respectively. The first strategy is based on grouping the nodes on the critical path and the second one is based on minimizing the cutsize.

One observation from our experimental study is that the nodes on the critical paths in the gate netlist are more likely to form the LUTs on the critical path in the LUT netlist. Therefore, in order to minimize the level increase, the basic strategy we use is to prevent those critical paths in the gate netlist from being cut. Specifically, before the partition, we will select all the nodes with top 10% smallest slack values, which are considered as nodes on the critical paths. Here, the slack values are obtained by assuming the unit delay model for each LUT and performing timing analysis. With those non-critical nodes removed from the original netlist, the selected critical nodes form a number of connected components. Note that by this method, the nodes on the same critical path belong to the same connected component. In the following partition step, each connected component is treated as a single node which cannot be cut. As a consequence, this prevents the critical paths in the gate netlist from being cut.

In our experiment, we find that after applying the partitioning and the parallel mapping, the critical path length of each sub-netlist of LUTs is no more than that of the LUT netlist produced by the original mapping. However, the critical path length increases after we merge the sub-netlists of LUTs together. Our analysis shows that the length increase is due to those LUTs formed by the nodes on the non-critical paths in the gate netlist, which are connected to the critical paths. One example is shown in Fig. 4. In the left of the figure, we show part of a critical path in a LUT netlist, which corresponds to a critical path in the gate netlist. The left part also shows the logic gates covered by LUT2. The number next to each node in LUT2 is the slack of that node. From the slack values, we can see that nodes A, B, C, D, and E are on the critical paths while node F is not. Then by our method, nodes A, B, C, D, and E are grouped into the same connected component. Now consider a bi-partition in which the edge connecting nodes D and F is cut. As shown in the middle of Fig. 4, after the partitioning and parallel mapping, the critical path lengths of both sub-netlists of LUTs do not increase compared to the original mapping result. However, when we merge the two sub-netlists of LUTs, the critical path length increases by one due to the LUT formed by the non-critical node F, as shown in the right of Fig. 4. To address this problem, when selecting the nodes, we not only choose the nodes on the critical paths, but also include a number of nodes on the non-critical paths which are connected to the critical paths. The details of this selection process will be discussed when we elaborate the combinational cut algorithm later. In this way, the level increase caused by the non-critical nodes can be mitigated.

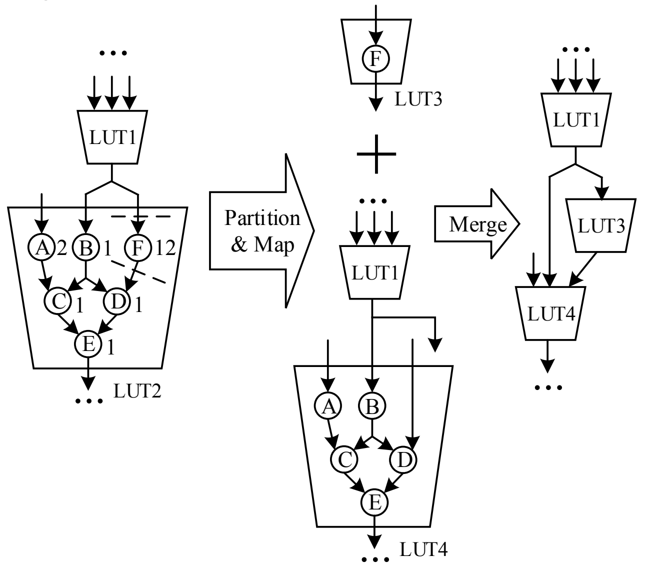

Fig. 4. Illustration that LUTs formed by the nodes on the non-critical paths in the gate netlist may cause the increase of the critical path length after the sub-netlists of LUTs are merged.

For minimizing the area increase, we find that the cutsize plays an important role. According to the structure of LUTs, one PO must be covered by one LUT. Since one cut in the original netlist generates a pair of PI and PO in two sub-circuits, respectively, minimizing the cutsize will reduce the number of extra LUTs needed to cover those additional POs caused by the partitioning. For this purpose, we apply the partition algorithm, hMETIS [6], to cut the circuit, which can partition a graph into k parts with roughly equal sizes while minimizing the cutsize.

The pseudocode of the combinational cut which combines the two strategies is shown in Algorithm 1. Assume that we want to partition the netlist into k sub-netlists. We first select top critical nodes based on the slack values and form connected components from them (Line 4). Then we augment each connected component by selecting a number of adjacent non-critical nodes (Lines 5-10). This is done in an iterative way (Line 5). In each iteration of the outer while loop, we go through each connected component one by one (Line 6). For each connected component, we augment it by including all non-critical nodes that are adjacent to a node in the component (Lines 7-8). The outer while loop is stopped when the number of remaining nodes is less than half of the size of the input netlist (Lines 9-10). At that time, each connected component is expanded by including a number of non-critical nodes. After that, each connected component is treated as a single node in a graph with node weight as its size (Lines 11-12). The remaining non-critical nodes in the netlist are given a weight of 1. Finally, hMETIS is called to do a k-way partition on this weighted graph (Line 13). This produces k sub-netlists which will be fed into k different cores for parallel technology mapping.

Algorithm 1 Combinational Cut
```
1.  **inputs:** gate netlist $G$ and number of partitions $k$
2.  **outputs:** sub-netlists of gates $c_1, c_2, ..., c_k$
3.  calculate the slacks of all nodes in the netlist;
4.  select nodes with top 10% smallest slack values and assume
    they form $M$ connect components $b_1, b_2, ..., b_M$;
5.  **while** 1 **do**
6.      **for** each connected component $b_i$ **do**
7.          $b_{i,old} \Leftarrow b_i$;
8.          add all remaining nodes which are adjacent to a node in
            $b_{i,old}$ to $bi$;
9.      **if** number of remaining nodes < half of the size of $G$ **then**
10.         **break**;
11. **for** each connected component $b_i$ **do**
12.     treat it as a single node in a graph with weight as its size;
13. apply hMETIS in node weighted mode to get $k$ partitions
    $c_1, c_2, ..., c_k$;
14. **return** $c_1, c_2, ..., c_k$;
```

D. Compensation

As mentioned earlier, although the combinational cut can reduce the increase on area and level, it still causes some quality loss. In order to further minimize the loss, after merging, we will do a compensation.

For technology mapping, it usually applies a basic mapping step for a number of iterations. For example, as the authors of the logic synthesis tool ABC mentioned in the user guide, in order for the area to converge, the user needs to call its basic technology mapping step for more than 10 iterations [7]. The basic idea of the compensation is to execute a small number of basic mapping iterations on the merged LUT netlist. This is done on a single core. The majority of the mapping iterations are done in the parallel mapping phase before the compensation. The acceleration ratio of the proposed flow with compensation can be estimated as

$$S = \frac{nt_1}{(n-m)t_2 + mt_1 + t_3}, \qquad (1)$$

where the numerator and denominator represent the runtime of the original mapping flow and that of the parallel mapping flow, respectively. In Eq. (1), n and m $(m < n)$ are the iteration numbers in the original mapping flow and in the compensation phase, respectively. The iteration number in the parallel mapping phase is $(n - m)$. t_1 is the runtime of one iteration of technology mapping on the entire netlist, t_2 is the runtime of one iteration in the parallel mapping phase, and t_3 is the runtime for the partitioning and merging steps in the parallel mapping flow.

Assume that the runtime t of one basic mapping iteration is a polynomial function on the size N of the netlist, i.e., $t = c \cdot$

N^a, where $a \geq 1$ and $c > 0$ are constants. Assume that in the parallel mapping, we partition the original netlist into $k \geq 2$ sub-netlists of equal sizes. Then, we have $t_1 = k^a \cdot t_2$. As our experiment shows, when the design is very large, t_3 is negligible compared to t_1 and t_2. Thus, Eq. (1) can be approximated as

$$S \approx \frac{nk^a}{(n-m)+mk^a} = \frac{nk^a}{n+m(k^a-1)}. \qquad (2)$$

From Eq. (2), we can see that to increase the acceleration ratio, we should reduce the number of mapping iterations applied in the compensation step. However, reducing the iteration number will reduce the compensation for the quality loss. Thus, there is a trade-off between the acceleration ratio and circuit quality.

III. Experimental Results

In this section, we show the experimental results of the proposed partition-based parallel mapping method. We implemented the flow in CME Synthesis Technology which is an industrial FPGA mapping platform developed by Capital Microelectronics (CME), Ltd. A number of large designs from IWLS 2005 benchmarks [8] were chosen as input netlists. All the experiments were conducted on a desktop with Intel Core i7 3.50Ghz CPU and 16GB RAM. For the proposed flow, 2/3 of the original mapping iterations are assigned for the parallel mapping phase and the remaining 1/3 for the compensation phase.

Table 1. Comparison between the proposed flow and the original mapping flow on area, level, and runtime.

benchmark	#LUT			level			runtime (s)		
	orig. flow	ours	increase (%)	orig. flow	ours	increase (%)	orig. flow	ours	Speed-up ratio
b17	8220	8292	0.87	14.4	13.4	−7.46	371	199	1.86
b18	19236	19654	2.17	37.1	37.1	0	785	477	1.65
b19	37342	38551	3.24	47	46.1	−1.02	1816	1058	1.72
b22	3948	3703	−6.21	14.2	14.3	0.70	168	97	1.73
des_perf	6740	6488	−3.74	5	5	0	313	164	1.91
tate_pairing	47212	47312	0.21	6	6	0	1793	1385	1.29
wb_conmax	12974	13466	3.79	8	8	0	387	252	1.54
aes_core	5705	5985	4.91	7	7	0	575	351	1.64
vga_lcd	13976	16028	14.68	8	8	0	715	511	1.40
ethernet	10696	11377	6.73	11.5	11.5	0	409	273	1.50

Table 1 shows the experimental results comparing the proposed flow with the original mapping flow on area (#LUT), level, and runtime. In this experiment, the parallel mapping uses 4 cores. Since hMETIS involves some randomness, the results of our algorithm are the average of 3 runs. (We found the variance of these 3 runs is very small due to the good convergence of hMETIS.) From the table, we can see that both the area and the level of the circuits produced by the proposed flow are almost the same as those of the circuits produced by the original flow. For the benchmark `des_perf`, the result produced by the proposed flow is even better. An average acceleration ratio of 1.6X is achieved. Since technology mapping usually takes around 30% of the total FPGA compiling time, the acceleration will lead to around 12% decrease in the total compiling time.

We also studied how the number of cores used, which is equal to the number of partitions, affects the proposed parallel mapping flow. Table 2 shows the average LUT count increase ratio, level increase ratio, and speed-up ratio of the proposed flow compared to the original flow for 2, 4, and 6 cores used. We can see that with more cores used, the LUT count increases slightly. This is because using more cores requires more partitions and hence, the larger cutsize, which causes the increase of LUT count. With more cores used, the level increase is negligible. This demonstrates the effectiveness of our critical-path-oriented strategy in preserving the level of the netlist. As expected, the speed-up ratio increases with more cores used. However, the growth rate of the speed-up ratio decreases with more cores. This actually is consistent with Eq. (2), which is a non-linear function on the number of cores k.

Table 2. The average LUT count increase ratio, level increase ratio, and speed-up ratio of the proposed flow compared to the original flow for different numbers of cores.

#Core	#LUT increase ratio (%)	level increase ratio (%)	speed-up ratio
2	2.73	1.31	1.32
4	2.69	−0.64	1.62
6	3.98	0.90	1.69

IV. Conclusion

In this paper, we proposed a new flow to parallelize the technology mapping. Different from the traditional methods which parallelize the steps in the mapping algorithm, our method is based on circuit partitioning. This makes it independent of the mapping algorithm. We developed two partition methods, fanout cut and combinational cut, to minimize the quality loss caused by the partitioning. Experimental results showed that our method accelerates the mapping flow while producing circuits with almost the same quality compared to the original mapping flow.

Acknowledgment

The authors acknowledge a generous support and help from the Capital Microelectronics (CME), Ltd.

References

[1] Ludwin, A., Betz, V., & Padalia, K. (2008). High-quality, deterministic parallel placement for FPGAs on commodity hardware. *International Symposium on Field Programmable Gate Arrays* (pp.14-23).

[2] Chan, P. K. & Schlag, M. D. F. (2003). Parallel placement for field-programmable gate arrays. *International Symposium on Field Programmable Gate Arrays* (pp.43-50).

[3] Mishchenko, A., Chatterjee, S., & Brayton, R. (2007). Improvements to technology mapping for LUT-based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 26(2), 240-253.

[4] Kennings, A. & Ravishankar, C. (2011). Parallel FPGA technology mapping using multi-core architectures. *Canadian Conference on Electrical & Computer Engineering* (pp.274-279).

[5] Boppana, V., Saxena, P., Banerjee, P., Fuchs, W. K., & Liu, C. L. (1996). A parallel algorithm for the technology mapping of LUT-based FPGAs. *Euro-Par'96 Parallel Processing* (pp.828-831).

[6] Karypis, G., Aggarwal, R., Kumar, V., & Shekhar, S. (1997). Multilevel hypergraph partitioning: Application in VLSI domain. *Design Automation Conference* (pp.5526-529).

[7] Mishchenko, A. *et al.* (2007). ABC: A system for sequential synthesis and verification. http://www.eecs.berkeley.edu/~alanmi/abc/

[8] IWLS 2005 benchmarks. http://www.iwls.org/iwls2005/benchmarks.html