

Optimal Simultaneous Mapping and Clustering for FPGA Delay Optimization

Joey Y. Lin

Magma Design Automation
lin@magma-da.com

Deming Chen

Department of ECE
University of Illinois, Urbana-Champaign
dchen@uiuc.edu

Jason Cong

Department of Computer Science
University of California, Los Angeles
cong@cs.ucla.edu

ABSTRACT

Both technology mapping and circuit clustering have a large impact on FPGA designs in terms of circuit performance, area, and power dissipation. Existing FPGA design flows carry out these two synthesis steps sequentially. Such a two-step approach cannot guarantee that the final delay of the circuit is optimal, because the quality of clustering depends significantly on the initial mapping result. To address this problem, we develop an algorithm that performs mapping and clustering simultaneously and optimally under a widely used clustering delay model. To our knowledge, our algorithm, named *SMAC* (simultaneous mapping and clustering) is the first delay-optimal algorithm to generate a synthesis solution that considers a combination of both steps. Compared to a synthesis flow using state-of-the-art mapping and clustering algorithms *DAOmap* [7] + *T-VPACK* [17] — *SMAC* achieves a 25% performance gain with a 22% area overhead under the clustering delay model. After placement and routing, *SMAC* is 12% better in performance.

Categories and Subject Descriptors

B.6.3 [Hardware]: Design Aids — *automatic synthesis*

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

FPGA, technology mapping, clustering, dynamic programming

1. INTRODUCTION

An FPGA chip generally consists of programmable logic blocks, programmable interconnections, and programmable I/O pads. The LUT-based FPGA architecture dominates the existing programmable chip industry, in which the basic programmable logic element is a K -input lookup table. Most FPGAs are hierarchical in nature. For example, Altera Stratix II device families [2] and Xilinx Virtex-4 device families [24] provides logic array blocks (LABs) [2] or configurable logic blocks (CLBs) [24] that can accommodate a cluster of basic logic elements (BLEs) with fast local interconnects. In this paper we address performance optimization for FPGA designs across two important synthesis

steps in the FPGA design flow — technology mapping into LUTs and LUT netlist clustering.

For LUT-based FPGAs, technology mapping converts a given Boolean circuit into a functionally equivalent network comprised only of LUTs. It is an important synthesis step for FPGA design flow because it directly defines the number of LUTs used for the design and the critical path length passing through these LUTs. There are many technology mapping algorithms published in the literature to improve mapping depth/delay [7][9][16][18][25], area [7][11][12][18][19], or power consumption [3][8][13][14]. FlowMap [9] is the first algorithm to guarantee a depth-optimal mapping solution in polynomial time under the *unit delay model*. Other depth-optimal algorithms try to minimize area or power under delay constraints [3][7][13][14]. Clustering has traditionally been used in the VLSI industry to extract underlying circuit structures and construct a natural hierarchy in the circuits. In [21], the first delay-optimal clustering algorithm under the *general delay model* [20] was derived. In [23], authors presented a low-power clustering algorithm with the optimal delay. There are a few prior research efforts on FPGA clustering [5][6][10][13][17] [22]. The optimization goals were on routing track reduction [5], power reduction [6][13][22], performance improvement [6][10][17], and area reduction [22].

Existing FPGA synthesis flow carries out technology mapping and mapped netlist clustering in two separate stages. The mapping algorithm does not consider how the LUTs will be clustered next. During the clustering stage, the solution quality is already limited by the initial mapping result, and there is no guarantee that the final delay of the circuit is optimal. Moreover, FPGA architectures are often equipped with short or dedicated intra-cluster wires in the clusters and long inter-cluster wires in the routing channels. These techniques make the *unit delay model* used in the mapping stage significantly different from the *general delay model* in the clustering stage. Under the inaccurate unit delay model, the mapping stage cannot evaluate the inter-cluster delay introduced by clustering constraints. Even the common area-delay tradeoff techniques can be wrongfully applied to a timing-critical portion of the network during mapping.

Figure 1 is an example that illustrates the sub-optimality of uncooperative mapping and clustering. We use 3-input LUTs and a cluster capacity of 3 (i.e. at most 3 LUTs in each cluster) in this example. For the small netlist in (a), the optimal delay and area mapping creates a netlist with 5 LUTs, and the optimal clustering afterwards uses two clusters in (b). The critical path includes 3 inter-cluster delays (counting the edges between I/Os and clusters) and 1 intra-cluster delay. On the other hand, with some nodes being duplicated, another solution uses 6 LUTs for mapping in (c) and two clusters in (d). Its critical path includes 2 inter-cluster delays and 2 intra-cluster delay. This is better than the solution offered by separate optimal mapping and clustering shown in (b).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2006, July 24-8, 2006, San Francisco, California, USA.
Copyright 2006 ACM 1-59593-381-6/06/0007...\$5.00.

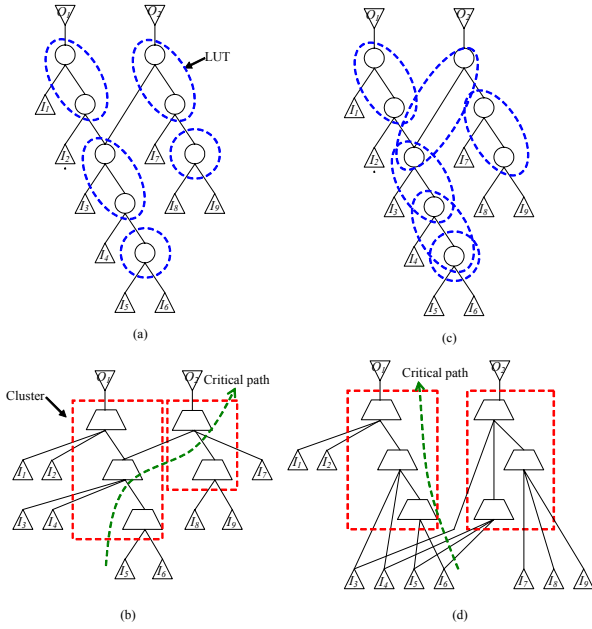


Figure 1. Mapping and clustering comparison

To address the suboptimal problem of separate mapping and clustering, we develop an algorithm that considers both mapping and clustering simultaneously in its delay calculation and final solution generation. We use a dynamic programming approach for the optimal delay propagation. During final solution generation, mapping and clustering solutions are produced simultaneously, guided by time and cluster-capacity requirements. To reduce area overhead, we also investigate some useful heuristics for the delay/area tradeoff. To our knowledge, our algorithm is the first delay-optimal algorithm that combines both mapping and clustering to achieve better synthesis solutions for FPGAs.

In Section 2 we provide some definitions and formulate the simultaneous mapping and clustering problem. Section 3 provides a detailed description of our algorithm. Section 4 describes some area reduction heuristics. Section 5 presents experimental results, and Section 6 concludes this paper.

2. PRELIMINARIES, DEFINITIONS AND PROBLEM FORMULATION

Figure 2 shows the basic structure of a logic block with size M , i.e., containing M BLEs. The logic block inputs and outputs are fully connected to the inputs of each BLE. It also shows some details of the peripheral circuits around the logic block for signal switching and driving.

We would like to introduce some definitions used in our work. A Boolean network can be represented by a DAG where each node represents a logic gate, and a directed edge (i, j) exists if the output of gate i is an input of gate j . A PI node has no incoming edges and a PO node has no outgoing edges. We treat the flip-flop outputs as special PIs and the flip-flop inputs as special POs. We use $input(v)$ to denote the set of nodes which are *fanins* of gate v . Given a Boolean network N , we use O_v to denote a *cone* rooted on node v in N . O_v is a sub-network of N consisting of v and some of its predecessors, such that for any node $w \in O_v$, there is a path from w to v that lies entirely in O_v .

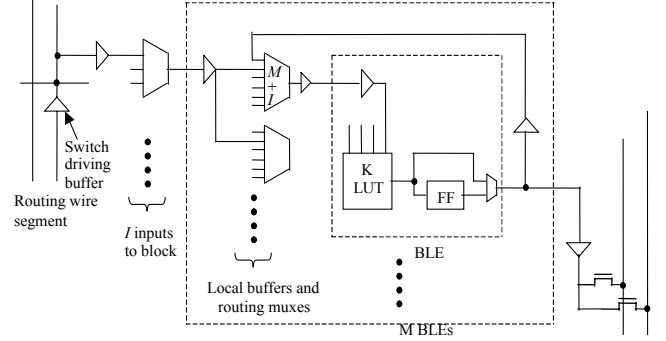


Figure 2: A logic block and its peripheries

The maximum cone of v , consisting of all the PI predecessors of v , is called a *fanin cone* of v , denoted as F_v . We use $input(O_v)$ to denote the set of distinct nodes outside O_v , that supply inputs to the gates in O_v . A *cut* is a partitioning (X, X') of a cone O_v such that X' is a cone of v . A *K-feasible* cut represents that X' can be implemented by a K -input LUT (K -LUT). The *level* of a node v is the length of the longest path from any PI node to v . The level of a PI node is zero. The *depth* of a network is the largest node level in the network. A Boolean network is *l-bounded* if $|input(v)| \leq l$ for each node v .

In order to represent the FPGA logic block architecture in the DAG network, we define a cluster rooted on a node set R , denoted as C_R , as a subgraph such that any path connecting two arbitrary nodes in C_R lies entirely in C_R . The roots in R are also the outputs of C_R . $node(C_R)$ represents the set of nodes contained in C_R . Here, each node will be a LUT cell. A cluster is *M-feasible* if $|node(C_R)| \leq M$. The notation $|node(C_R)|$ is also called the *size* of the cluster. In this case, we have $1 \leq |R| \leq M$. A size- M cluster can be implemented by a size- M logic block.

Because the exact layout information is not available during the mapping and the clustering stage, we use a *general delay model* introduced in [20]. Following this model, we set each interconnection edge between two clusters with a constant delay $D_{ext-edge}$, each interconnection edge between two nodes in the same cluster with a constant delay $D_{int-edge}$, and each node in the network with a delay D_{node} . The optimal critical path delay of the mapped and clustered FPGA design under this model is called the *optimal circuit delay*.

The simultaneous mapping and clustering problem for FPGA delay optimization is to cover a given l -bounded ($l \leq K$) Boolean network with K -feasible cones (or equivalently, K -LUTs), and cluster these LUTs in such a way that we can achieve the optimal circuit delay under the general delay model. In this work, the input networks are all 2-bounded, i.e., $|input(n)| \leq 2$. We use 4-LUT, and our cluster size is 10. Notice all of these parameters are changeable within reasonable ranges. We would like to mention that, in our current formulation, we do not consider cluster input constraint (number of inputs allowed for a cluster). It is understood that the cluster input constraint is chosen so that one can utilize at least 98% of the LUTs in the clusters in most of cases [1]. Since the input constraint is designed to be met in most cases, we focus on satisfying the cluster size constraint. The same dynamic programming paradigm used in SMAC can be extended to handle the input constraints as well, but with increased time complexity.

3. ALGORITHM DESCRIPTION

We build our optimization process with two phases — arrival time calculation and cluster realization. The arrival time calculation phase examines all the possible clustering solutions based on different mapping, and calculates the optimal signal arrival times each node is able to achieve, with the consideration of the area (number of LUTs) in the cluster. The second step is the cluster realization phase that forms the mapping and clustering solutions. In this phase, we first find a required delay target. This target may come from the best signal arrival time we can achieve, or can be set by users' timing constraints. Then, the clustering and mapping solutions are formed simultaneously, based on the guidance of arrival times calculated in the first phase. Area reduction techniques are also applied on the delay non-critical portions of the design.

3.1 Optimal Arrival Time Computation

The signal arrival time of a node v is defined as the longest path delay that starts from any PI to v . Our goal is to find the smallest signal arrival time for v among all the mapping and clustering solutions. This value is an important factor for the optimization, as the largest signal arrival time among all the POs is the circuit delay. Moreover, this value also helps to indicate the criticality of each node. Accurate calculation of the best possible arrival time becomes a key problem in the algorithm.

A dynamic-programming-based algorithm is designed to calculate the best arrival time of each node. Given a 2-bounded network N , a topological order of all the nodes can be generated. Starting from PIs to POs, every node v in N is traversed, and its arrival time is calculated. A clustering solution S_v of v represents a mapping of the input cone O_v into a sub-network N_v consisting of K -LUTs, and $|node(N_v)| \leq M$; i.e., the sub-network N_v can be mapped into one cluster. For each S_v , $Arr(S_v)$ is denoted as the optimal arrival time of S_v . Let $Clusters(v)$ denote the entire set of feasible clustering solutions rooted at v . Given a predecessor node u of node v , the optimal arrival times of the clustering solutions in $Clusters(u)$ are already calculated when the program reaches v . Our dynamic programming propagates optimal arrival times of the complete clustering solutions.

The first step is to find all the K -feasible cuts of node v . Previous mapping work (e.g., [11]) provided efficient ways to enumerate a complete K -feasible set of cuts (cut set) for a node. We use a similar method to compute cut sets. Let $Cuts(v)$ denote the cut set for v . For any cut Cut_v in $Cuts(v)$, let $u_1, u_2, \dots, u_{|Cut_v|}$ be the input nodes of the cut. $Clusters(v)$ can be constructed from the clustering solutions on the input nodes of Cut_v .

$Clusters(v) =$

$$\left\{ S_v \mid S_v = L_v \cup \left\{ \bigcup_{i=1}^{|Cut_v|} (S_{u_i} \in Clusters(u_i) \text{ or } \phi) \right\}, Cut_v \in Cuts(v), |S_v| \leq M \right\}$$

This formula means that we either pick an empty set or a clustering solution from the input nodes of every Cut_v in $Cuts(v)$. Let a clustering solution on an input node to be referred as a *sub-cluster* (it is used to construct a larger clustering solution on v). After joining all the sub-clusters and the LUT L_v (generated by Cut_v) together, if the new cluster is still M -feasible, i.e., meeting the capacity constraint of the cluster, then the union is a good clustering solution for v . Since $Cuts(v)$ includes all the input nodes based on the cuts on v , and we enumerate all the sub-clusters on these inputs, we are able to cover all the clustering solutions rooted at v . Figure 3 is an example to illustrate this idea. Here we choose $S_{u_1}, S_{u_2}, S_{u_3}$ and ϕ for u_4 and form a cluster S_v .

Theorem 1. $Clusters(v)$ represents all the M -feasible clustering solutions rooted at v .

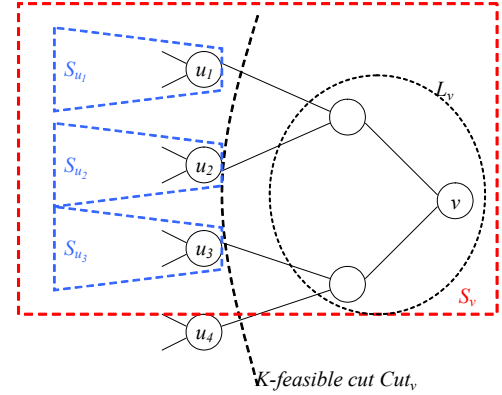


Figure 3. Illustration of clustering solution construction

Due to the length restriction, the proofs of this theorem and subsequent theorems are left out, and can be found in [15]. The optimal arrival time of a clustering solution S_v can be derived from its sub-clusters. If S_v is a super set of a sub-cluster S_{u_i} of input node u_i , node v and u_i are in the same cluster and the signal propagation goes through an intra-cluster delay $D_{int-edge}$. Otherwise, if the empty set is used when creating S_v from u_i , node v and u_i are in different clusters, and the signal arrival time has to be added with an inter-cluster delay $D_{ext-edge}$.

$$Edge_delay(i) = \begin{cases} D_{int-edge}, & S_{u_i} \subseteq S_v \\ D_{ext-edge}, & otherwise \end{cases} \quad (*)$$

$$Arr(S_v) = \max_{i=1 to |Cut_v|} \{ Arr(S_{u_i}) + Edge_delay(i) + D_{L_v} \}$$

In Figure 3, $D_{int-edge}$ is applied for the edges u_1 to v , u_2 to v and u_3 to v , and $D_{ext-edge}$ is added on the path u_4 to v . Theorem 1 shows that all the clustering solutions are listed, and the above delay calculation accurately propagates the arrival time.

Theorem 2. Arrival time propagation in (*) calculates the optimal signal arrival times of all the clustering solutions.

The cluster capacity M is predetermined by the FPGA architectures. For example, M equals 4 and 8 for the Xilinx Virtex and Virtex-4 families; M is 10 for Altera Stratix family. We are able to enumerate the clustering solutions when M is small. However, the runtime complexity becomes an increasing concern when M is as large as 10. Designing a runtime-efficient algorithm is the key to make it practical.

For each node there are a large number of clustering solutions. In order to implement the signal arrival time computation in a runtime-memory efficient manner, we choose not to keep any of the clustering solutions. Instead, an array of size M is introduced, where M is the capacity of the cluster. $Arr(v, m)$ represents the best signal arrival time among all the S_v solutions where $|S_v| \leq m$, $1 \leq m \leq M$. Therefore, the clustering solution S_v in the original algorithm is represented by its size $|S_v|$, and the $Arr(S_v)$ is represented by $Arr(v, |S_v|)$.

For any K -feasible cut Cut_v in $Cuts(v)$, with input nodes $u_1, u_2, \dots, u_{|Cut_v|}$, let $area_1, area_2, \dots, area_{|Cut_v|}$ be any integer combinations that satisfy $0 \leq area_1, area_2, \dots, area_{|Cut_v|} \leq M$, and $area_1 + area_2 + \dots + area_{|Cut_v|} + 1 \leq M$ (Δ). The idea is to pick a cluster of size $area_1$ from u_1 , a cluster of size $area_2$ from u_2 , ..., and form a large cluster whose area is still less than or equal to M . The "1" in the inequality represents the LUT area for node v itself (L_v). If $area_1$ is zero, it

represents that the empty set is chosen for u_i , and an inter-cluster delay should be applied on the edge from u_i to v . Therefore, let the m represents the area of the cluster, then the value $Arr(v, m)$ can be computed recursively as follows.

$$Edge_delay(i) = \begin{cases} D_{int-edge}, & area_i > 0 \\ D_{ext-edge}, & otherwise \end{cases}$$

$$Arr(v, m) = \min_{Cut, e \in Cuts(v)} \left\{ \min_{\substack{[0, m] \\ \sum_{i=1}^K area_i = m}} \left\{ \max_{u_i \in Cut_v} \{ Arr(u_i, area_i) + Edge_delay(i) + D_{Lv} \} \right\} \right\}$$

$$Arr(v, 0) = \min_{area=1toM} \{ Arr(v, area) \}$$

Furthermore, we define $Arr(v, 0)$ as the best arrival time for node v , and the above formula describes how different edge delays are added to the time computation accordingly. As an example, Figure 4 shows how to compute the arrival times of node v , from a 3-cut $\{u_1, u_2, u_3\}$. Let D_{Lv} , $D_{int-edge}$ and $D_{ext-edge}$ be 1, 2 and 7 respectively. $Arr(v, 1)$ is calculated using $Arr(u_1, 0)$, $Arr(u_2, 0)$ and $Arr(u_3, 0)$, each with an extra delay of 1+7 ($D_{Lv} + D_{ext-edge}$); $Arr(v, 2)$ is calculated using $Arr(u_1, 0)$, $Arr(u_2, 1)$ and $Arr(u_3, 0)$, with $Arr(u_1, 0)$ and $Arr(u_3, 0)$ plus 1+7, and $Arr(u_2, 1)$ plus 1+2 ($D_{Lv} + D_{int-edge}$); $Arr(v, 4)$ is calculated with $Arr(u_1, 1)$, $Arr(u_2, 1)$ and $Arr(u_3, 1)$, and each value needs to plus 1+2.

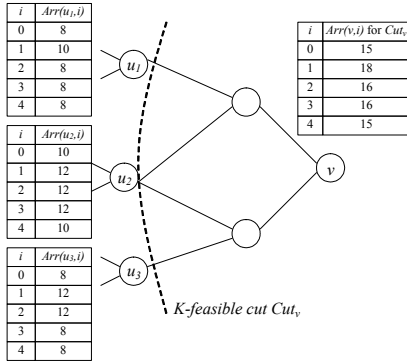


Figure 4. Arrival time computation

With the help of the dynamic programming procedure, we are able to calculate the possible signal arrival times of each node. With this accurate timing information, we can still guarantee timing optimality and be able to guide our clustering realization procedure to form the mapping and clustering solution with good timing/area trade-off.

3.2 Cluster Realization Phase

The clustering realization step performs mapping and constructs clustering solutions simultaneously. The major task is to achieve the timing requirement of the design or the best signal arrival time calculated in the arrival time computation phase. The circuit area is kept as small as possible at the same time.

Given a timing constraint $Target$, we first assign $Target$ to the POs of the circuit. We also assign an area constraint M to the POs, to indicate the current area requirement for the cluster. It is obvious that the constraint $(Target, M)$ satisfies the following Property 1. After assigning the constraint $(Target, M)$, we traverse from POs to PIs along a reverse topological order to form the mapping and clustering solutions.

Property 1. For each pair of constraint (T, A) on a node v , there exists a solution m at node v , which satisfies $1 \leq m \leq A$, and $Arr(v, m) \leq T$.

When we map each node v , assume we have a set of constraints (T_i, A_i) on v . The signal arrival time of node v must come before the target T_i , and the cone rooted at v cannot exceed A_i LUTs in the same cluster. The requirements indicate the timing and area restrictions for selecting the mapping solution.

Theorem 3. If the requirement satisfies Property 1 at node v , we can find a mapping solution that propagates new requirements to the input nodes u_i , and satisfies Property 1 at u_i .

The proof of Theorem 3 is an inverted process of the arrival time calculation. By properly distributing the area and timing requirements to the input nodes u_i , the existing of at least one mapping solution is guaranteed.

Since Property 1 is satisfied on the PO nodes, by the introduction of Theorem 3, we can claim that Property 1 is satisfiable at any nodes in the network.

If node u receives multiple requirements from different fanout nodes, duplications of node u may be necessary, since the fanout clusters may be different. The duplication causes a large area increase and makes it difficult for the placement and routing stages to meet the timing constraint. Therefore, we apply a few area reduction techniques in Section 4 to control the area.

3.3 Algorithm Analysis

3.3.1 Complexity Analysis

The most time-consuming part of the algorithm is the updating of the arrival times and finding a feasible solution during the cluster realization stage. In each operation, we need to find out all the integer permutation that satisfies the condition (Δ) . The number of combinations is $C(M+K-1, K)$, where K is the cut size ($|Cut_v|$). The complexity appears to be in a non-polynomial order. But with K and M as constants given by the architecture, the combination number is actually a constant. For $M=10$ and $K=4$, we have $C(M+K-1, K)$ as 715. So the complexity of the algorithm is $O(w)$ where w is the number of cuts in the network. Theoretically, w may be in the order of $O(N^K)$ in the worst case. However, for almost all practical circuit, it can be regarded as $O(CN)$ for small K ($K \leq 6$). Therefore, the overall time complexity of the SMAC algorithm can be considered linear with a large constant factor.

Our experiments show that the runtime of SMAC is within 100X of the standard DAOmap plus T-VPACK flow. Nevertheless, on a 750MHz Solaris machine, it takes only about 3 minutes for a design with 1000 LUTs. In general, SMAC has comparable runtime compared to the existing industrial tools for FPGA synthesis when many re-synthesis optimization features are turned on to achieve good performance. So, it is practical to apply the SMAC algorithm to real-life designs. In fact, as we shall discuss in Section 0, we are making very good progress of further runtime reduction of SMAC.

3.3.2 Optimality Analysis

When the clustering capacity M is large, during the arrival time calculation stage, we do not keep all the clustering solutions and their corresponding signal arrival times. Instead, we propose a method to keep the best signal arrival times for the clusters with each area cost. Under this situation, we can only claim that the signal arrival time is optimal when the mapping solutions of each cut inputs are duplication free. Otherwise the area of the cluster at node v could be smaller than the sum of all the sub-clusters. The tradeoff of the optimality is that we have a linear algorithm with fast runtime. In our implementation, we chose the one with fast runtime, since runtime is an important factor for industrial tools.

Another factor that could affect optimality comes from the sequential elements. Our algorithm works on the combinational part of the logic, which separates the input and output of registers into two units, PO and PI. In our algorithm, we cannot guarantee that these two pins are put in the same cluster. Therefore, we conservatively assign an inter-cluster delay to the edge that comes out of the sequential element. By doing this, we can guarantee that the signal arrival time we compute is achievable. The best arrival time is proved to be at most $D_{ext-edge} - D_{int-edge}$ larger than the ideal solution.

4. AREA REDUCTION TECHNIQUES

With the timing constraint target set to infinity, our algorithm acts similar to a cut enumeration based mapping algorithm. However, with the timing constraint set and the logic duplicated in the clustering, the LUT number can increase several times over the result of DAOMap [7]. Therefore, different area reduction techniques are applied in our algorithm. More details are explained in the technical report [15].

4.1 Area-aware Mapping Selection

Given the timing and area requirements at a node, there may be many cuts and integer combinations to satisfy these requirements. We evaluate these cuts and combinations with an area cost function, which represents how much area increase this mapping solution may bring in. We select the cut and combination with the smallest area increase. The area cost function combines the estimation method in [11] and the iterative approach in [16].

4.2 Online Packing

During the clustering realization phase, the clusters we form may not be fully occupied. This is because the area requirement is only an upper bound, and the requirements distributed to the cut inputs may not realize this much area. As a result, this may create many small clusters and many unnecessary inter-cluster delays. In order to reduce the number of small clusters, we apply an online packing scheme during the clustering realization phase. The algorithm contains two types of packing heuristics. One is the predecessor packing that helps to reduce the number of inter-cluster delays. The predecessor packing step packs the consecutive elements into the same cluster, and converts many inter-cluster edges into the intra-cluster edges. It results in the increase of the available timing slacks and the number of mapping choices on nodes. Another packing algorithm packs the fanouts of the same node into one cluster. Therefore, the duplication introduced by the fanout clustering requirement is minimized. At the end of the flow, we also perform a bin-packing heuristic to pack the rest of the clusters to full capacity. The goal is to reduce area and help placement and routing.

4.3 Controlled Duplication

There exist nodes with large fanout numbers in the decomposed network. Assume that a node v has k fanouts, the area/timing constraints derived from these fanouts may require v to appear in many different clusters. When k is large, there is no easy way to avoid such duplications through the packing methods mentioned in Section 4.2. Therefore, a heuristic is adopted to preserve the large-fanout nodes in advance. We assign all the edge delay coming out of v with the inter-cluster delay $D_{ext-edges}$ and guarantee that v does not need to be duplicated during the clustering realization stage.

Table 1. Comparison on area (CLB number), delay with General Delay Model (GDM), and delay after P&R between DAOMap+T-VPACK and SMAC

Circuits	DAOMap+T-VPACK			SMAC			
	CLB No.	GDM Delay	P&R Delay (ns)	CLB No.	GDM Delay	P&R Delay (ns)	Run-time (sec)
alu4	118	48	32.7	141	40	28.8	38
apex2	141	58	36.6	191	44	32.9	53
apex4	113	45	34.6	190	38	31.1	43
bigkey	149	25	21.4	226	20	19.1	64
clma	528	109	68.4	682	89	60.0	843
des	123	50	36.5	186	35	30.8	63
diffeq	98	75	37.9	124	58	31.5	40
dsip	138	25	20.1	226	21	19.3	43
elliptic	210	97	52.2	237	75	43.9	302
ex1010	396	58	50.8	622	46	46.1	381
Ex5p	92	50	35.3	143	40	31.7	35
frisc	219	127	66.1	241	116	60.6	144
misex3	108	58	33.3	137	38	27.4	35
pdc	291	66	50.3	368	51	43.6	205
s298	100	132	66.9	101	112	58.0	33
s38417	405	71	39.3	479	58	35.4	3378
s38584	397	68	35.1	408	57	34.3	1498
seq	130	45	31.0	170	35	27.4	46
spla	222	61	41.4	275	48	38.9	120
tseng	78	80	39.7	97	71	37.1	39
				Area Over-head	Perf. Gain (GDM)	Perf. Gain (P&R)	
Ave.				22.5%	25.4%	12.3%	

5. EXPERIMENTAL RESULTS

The experiments are based on the 20 largest MCNC benchmarks. Standard synthesis and physical design flows are processed to guarantee a fair comparison. We use *script.algebraic* in *SIS* to do the technology independent optimization, followed by *tech_decomp* and *dmig* in *RASP* for decomposition. The FPGA architecture is extended from the VPR [4] standard architecture file. We use default VPR settings with the routing channel width set to 100 for all the benchmarks since modern FPGA devices usually provide enough routing segmentations.

Based on these standard pre-mapping and post-clustering settings, we compare our algorithm SMAC with a synthesis flow that consists of a state-of-the-art delay-optimal area-minimization mapping algorithm DAOMap [7]¹ and a popular timing-driven clustering algorithm T-VPACK [17]. T-VPACK also tries to reduce area by reducing the total number of inputs on the clusters and by not allowing node duplications. We apply the same general delay model in T-VPACK as used in SMAC.

Table 1 shows the experimental results. We can observe that on average, SMAC offers a 25.4% performance gain through delay reduction under the general delay model (GDM) compared to the standard synthesis flow, with a 22.5% area overhead in terms of CLB counts. This is a reasonable increase in the delay-area trade-off experience. After placement and routing, SMAC is 12.3% better in performance.

¹ The very recent mapping algorithm published in [18] reported better mapping area than DAOMap with similar mapping depth.

Table 2. Experiments on area reduction techniques

Circuits	Complete	Area cost	Online	Bin pack	Preserve
alu4	141	160	155	227	198
apex2	191	215	198	275	334
apex4	190	201	188	204	324
bigkey	226	226	226	757	264
...
Average	1	1.19	1.04	1.64	1.70

Experiments are also carried out to study the area reduction effects of different techniques. We compare our complete flow with different flows generated by removing individual area reduction techniques presented in Section 4. In Table 2, Column 2 is the complete flow. Columns 3 to 6 represent the flows without area cost estimation, online packing, bin-packing or the duplication control respectively. The area differences are 19%, 4%, 64% and 70% respectively. Reader may refer to [15] for details.

6. CONCLUSIONS AND ONGOING WORK

Existing FPGA design flows carry out technology mapping and clustering separately. In this work we presented the first delay-optimal simultaneous mapping and clustering algorithm. We showed that our algorithm achieved a considerable amount of performance gain compared to a design flow using separate mapping and clustering. We believe this work presented a promising direction, and it can be extended in different aspects. The cluster input constraint can be supported by current framework, although the runtime will increase due to consideration of more constraints. Our current research on this topic focuses on two directions: (i) further area reduction by adopting other mapping heuristics and flexible restrictions on node duplications, and (ii) further runtime reduction. Our preliminary result shows that a 10X speed-up of the SMAC algorithm is possible. We shall report more details as we make further progress.

7. ACKNOWLEDGEMENT

This work is partially supported by the National Science Foundation under the grant CCR-0306682 and a grant from Magma Design Automation under the California MICRO program.

REFERENCES

- [1] E. Ahmed, J. Rose, "The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density," *IEEE Transactions on VLSI System*, 2004.
- [2] Altera, Stratix II Device Family, 2004.
- [3] J. Anderson and F. N. Najm, "Power-Aware Technology Mapping for LUT-Based FPGAs," *Intl. Conf. on FPT*, 2002.
- [4] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.
- [5] E. Bozozgah, et al., "R-Pack: Routability-driven Packing for Cluster-Based FPGAs," *ASPDAC*, Jan. 2001.
- [6] D. Chen and J. Cong, "Delay Optimal Low-Power Circuit Clustering for FPGAs with Dual Supply Voltages," *Int. Sym. on Low Power Electronics and Design*, Aug. 2004.
- [7] D. Chen and J. Cong, "DAOmap: A Depth-Optimal Area Optimization Mapping Algorithm for FPGA Designs," *ICCAD*, Nov. 2004.
- [8] D. Chen, J. Cong, F. Li, and L. He, "Low-Power Technology Mapping for FPGA Architectures with Dual Supply Voltages," *Int. Sym. FPGA*, 2004.
- [9] J. Cong and Y. Ding, "An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *ICCAD*, Nov. 1992.
- [10] J. Cong and M. Romesis, "Performance-Driven Multi-Level Clustering with Application to Hierarchical FPGA Mapping," *Design Automation Conference*, June 2001.
- [11] J. Cong, C. Wu, and E. Ding, "Cut Ranking and Pruning: Enabling A General and Efficient FPGA Mapping Solution," *Int. Sym. on FPGA*, Feb. 1999.
- [12] R. J. Francis, et al., "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Design Automation Conf.*, 1991.
- [13] J. Lamoureux and S.J.E. Wilton, "On the Interaction between Power-Aware CAD Algorithms for FPGAs," *ICCAD*, Nov. 2003.
- [14] H. Li, W. Mak, and S. Katkooi, "Power Minimization Algorithms for LUT-Based FPGA Technology Mapping," *ACM Trans. on Design Automation of Electronic Systems*, Vol. 9, No. 1, Jan. 2004.
- [15] J. Lin, D. Chen, J. Cong, "Optimal Simultaneous Mapping and Clustering for FPGA Delay Optimization," *Technical report 060018*, Computer Science Department, UCLA.
- [16] J. Y. Lin, A. Jagannathan and J. Cong, "Placement-Driven Technology Mapping For LUT-Based FPGAs," *Int. Sym. on FPGA*, Feb. 2003.
- [17] A. Marquardt, V. Betz, and J. Rose, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density," *Int. Sym. on FPGA*, Feb. 1999.
- [18] A. Mishchenko, S. Chatterjee, R. Brayton, and M. Ciesielski, "An Integrated Technology Mapping Environment," *Int. Workshop on Logic and Synthesis*, 2005.
- [19] R. Murgai, et al., "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *ICCAD*, Nov. 1991.
- [20] R. Murgai, R.K. Brayton, and A. Sangiovanni-Vincentelli, "On Clustering for Minimum Delay/Area," *ICCAD*, Nov. 1991.
- [21] R. Rajaraman and D.F. Wong, "Optimal Clustering for Delay Minimization," *Design Automation Conference*, Jun. 1993.
- [22] A. Singh and M. Marek-Sadowska, "Efficient Circuit Clustering for Area and Power Reduction in FPGAs," *Int. Sym. FPGA*, 2002.
- [23] H. Vaishnav and M. Pedram, "Delay Optimal Clustering Targeting Low-Power VLSI Circuits," *Transactions on CAD*, Vol.18. No.6, Jun. 1999.
- [24] Xilinx, Virtex-4 Device Family, 2004.
- [25] H. Yang and D.F. Wong, "Edge-map: Optimal Performance Driven Technology Mapping for Iterative LUT based FPGA Designs," *ICCAD*, Nov. 1994.