This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2024.3515876

FPGA Technology Mapping with Adaptive Gate Decomposition

Chang Wu, *Member, IEEE*

***Abstract*—FPGA technology mapping is an extensively studied problem. There is functional decomposition as well as graph covering based approaches. For efficiency consideration, most existing algorithms are graph covering based. However, logic synthesis can affect the graph covering results significantly. In this paper, we propose an FPGA mapping algorithm with gate decomposition. Bin-packing is used to generate gate decompositions during mapping to avoid the decomposition choice pre-generation problem in existing approaches. Our results show that our algorithm can get an average of 13% area reduction over the state-of-the-art lossless synthesis based mapping algorithm in ABC. When compared with industrial tool Vivado, we can get a significant area reduction of 35% on average.**

***Index Terms*—FPGAs, Circuit Synthesis.**

I. Introduction

FPGA technology mapping is to map a generic gate level netlist into FPGA logic cells, which is a crucial step in FPGA synthesis. There are many FPGA technology mapping algorithms being proposed in decades for either area or delay optimizations [1][2][3][5][7][8][11][14]. It is worth mentioning that Flowmap is the first delay optimal algorithm in polynomial time [5]. For area minimization, however, the problem is shown to be NP-hard [4]. Cong, *et al.*, proposed a cut-enumeration based mapping algorithm for area optimization. The major contribution of their work is efficient cut enumeration, ranking and pruning based on an effective area computation. Later on, Chen and Cong proposed DAOmap for depth-optimal area minimization with cut enumeration [11]. Mishchenko, *et al.*, proposed priority cuts for mapping [14].

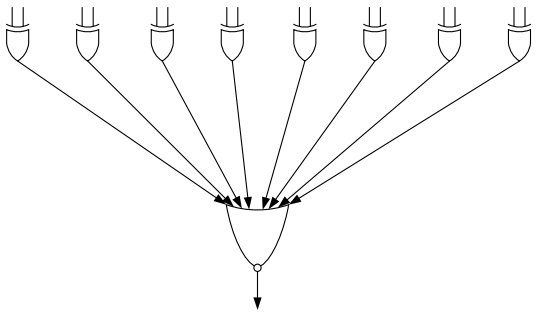


Figure 1 An 8-bit comparator design.

Chang Wu is with the School of Microelectronics, Fudan University, Shanghai, China (e-mail: wuchang@fudan.edu.cn).

However, all the graph covering based mapping algorithms suffer one problem that mapping quality is affected greatly by logic synthesis, in particular, gate decomposition. Let us look at an 8-bit comparator shown in Figure 1. To perform mapping for 6-LUT FPGAs, balanced tree decomposition for wide gates is normally performed, which will lead to a mapping solution with 4 6-LUTs as shown in Figure 2(a). However, if we choose a different gate decomposition as shown in Figure 2(b), we can get a mapping solution with 3 6-LUTs.

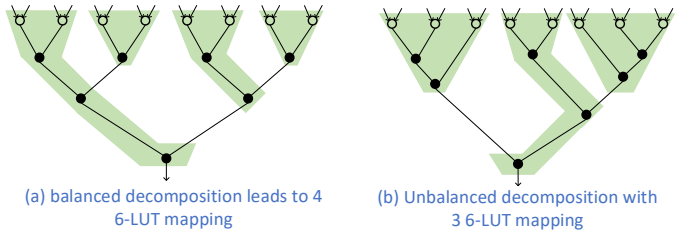


Figure 2 Different decompositions with different mapping.

Lehman, *et al.*, proposed to consider various gate decomposition choices in mapping [6]. Chen and Cong extended this approach for FPGA mapping [10]. Mishchenko, *et al.*, proposed to save intermediate synthesis results as gate decomposition choices [13]. All those algorithms pre-generate a set of gate decomposition choices to improve mapping results. In general, the larger the decomposition choice set, the better mapping quality, however, at the cost of longer runtime. For a w-input gate, the number of gate decomposition choices is to the exponential order of w. It is impractical to enumerate all possible gate decompositions for gates with 8 to 10 inputs, not to mention that there are designs with wide gates with over 100 inputs.

In this paper, we propose a simultaneous gate decomposition with mapping algorithm for K-LUT based FPGAs. We use the cut enumeration approach [8] as our mapping engine. For wide cuts (input size larger than K), we use bin packing algorithm [1] to generate gate decomposition and mapping solutions. Cut ranking and pruning are performed during the (wide) cut enumeration process. The main contribution of our algorithm is that gate decomposition is performed simultaneously during cut enumeration. Thus, we do not need to generate gate decomposition choices a priori. Secondly, we propose a wide cut enumeration and pruning heuristic to reduce the wide cut enumeration complexity from $O(C^W)$ to $O(C^2W)$, where C is the bound on the number of cuts for each gate during cut enumeration and W is the bound of gate input size. For area optimization, our experimental results show an average of 35% area reduction over a leading industrial tool

Authorized licensed use limited to: Dalian University of Technology. Downloaded on January 18,2025 at 07:20:51 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2024.3515876

[17] and 13% area reduction over the state-of-the-art lossless synthesis based mapping algorithm [13]. A preliminary version of our work is published in ACM FPGA'23 [18].

The remainder of this paper is organized as follows. Section 2 gives a review of related works. Our algorithm with wide gate decomposition is presented in Section 3, while details of wide cut enumeration and pruning is presented in Section 4. An extension of our algorithm for delay minimization is presented in Section 5. Our experimental results are presented in Section 6 with discussions and conclusion in Section 7.

II. Review Of Related Works

Cut enumeration based mapping has shown to be very efficient to solve the graph covering FPGA technology mapping problem [8]. Let $G(V,E)$ be a graph of a combinational logic netlist. V is the node set representing gates and primary inputs (PIs) and primary outputs (POs). E is the edge set representing connections between nodes. A fanin cone C_v of node v is a sub-netlist of G with v and all its predecessors. A cut is a partition $(X,\bar{X})$ of C_v, such that $\bar{X}$ is a cone of v. The cutset $C(X,\bar{X})$ is a set of nodes in X with fanout in $\bar{X}$. A cut is K-feasible if $|C(X,\bar{X})| \leq K$.

To solve the mapping with simultaneous gate decomposition problem, we choose to use the cut enumeration engine proposed in [8]. Bin-packing is used to do simple gate decomposition as the Chortle-crf algorithm [1]. Here, we will briefly review the cut enumeration algorithm [8] and bin-packing based gate decomposition algorithm [1].

In [8], Cong, *et al.*, proposed to do cut enumeration for all nodes in a netlist and then select a cut for each node to form a mapping solution. Since a K-cut of a node is also a K-cut of its fanin node, enumeration of all K-cuts of a node can be done by enumerating and combining all K-cuts of its fanin nodes. The main contribution of Praetor algorithm is a novel cut ranking and pruning algorithm based on effective area computation [8]. For K larger than 4, the number of K-cuts for a node can be over several hundreds. Cong, *et al.*, proposed a fanout distribution based effective area for mapping area estimation of the logic cone rooted at each cut as

$$A(cut) = \sum_{v \in cut} \frac{A(v)}{fanout(v)} + 1, \qquad (1)$$

where v are input nodes to the cut. It was shown that the effective area was a lower-bound to the mapping area and the average estimation error is only 11% [8]. With effective area, Cong, *et al.*, proposed to rank cuts in increasing order of effective area number and cuts and prune low ranked cuts for efficient cut enumeration. Mapping solutions can be generated by selecting min-area cuts for nodes from POs to PIs. This algorithm is very efficient and versatile to adapted to various optimization objectives.

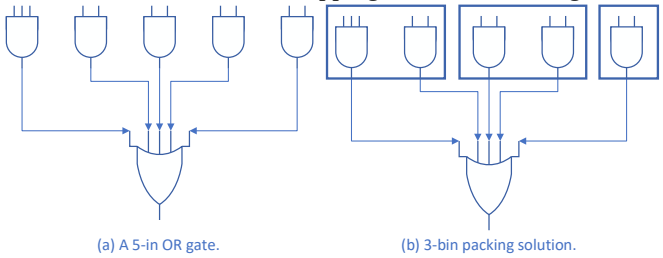


Figure 3 Bin-packing for gate decomposition.

For K-LUT based FPGAs, a wide gate needs to be decomposed before mapping. R. Francis, *et al.*, proposed a bin-packing based FPGA mapping algorithm [1]. Their algorithm works on netlists with all simple gates. A simple gate is a logic gate, such as AND/OR/XOR, that can be decomposed in any input combinations. Bin packing is a well-known problem of packing a number of items with different sizes into given size bins. The objective is to minimize the number of bins needed for packing. Although bin-packing problem is NP-hard, there are a number of heuristics for reasonable quality. For a wide gate with w inputs, one can view each fanin gate as a K-feasible bin and perform bin-packing for the w bins. The process can be illustrated with an example shown in Figure 3(a) with a 5-in OR gate. For K=5, the bin packing result is shown in Figure 3(b). After that, we can generate the gate decomposition as shown in Figure 4(a) and form a mapping solution as shown in Figure 4(b). A pseudo code of first-fit-decreasing bin-packing algorithm is shown in Algorithm 1. We refer readers to [1] for more details about chortle-crf algorithm.

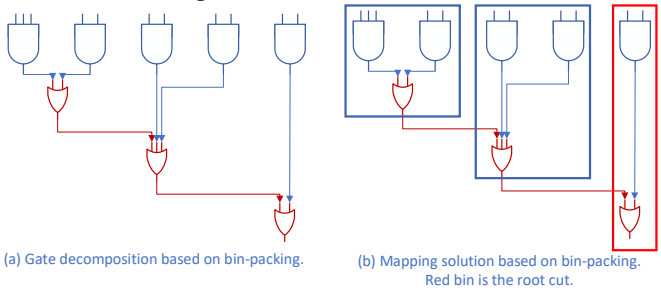


Figure 4 Mapping with bin-packing, K=5.

Algorithm 1 First-Fit-Decreasing Binpacking Algorithm

```
start with an empty bin list
while there are unpacked boxes {
   if the largest unpacked box will not fit within any bin is the bin list {
      create an empty bin and add it to the end of the bin list
   }
   pack the largest unpacked box into the first bin it will fit within
}
```

III. Simultaneous Cut Enumeration And Gate Decomposition

In this section, we will present our simultaneous mapping with gate decomposition algorithm, named AGDMap. Cut enumeration with ranking and pruning is the main engine of our algorithm. Our objective is area minimization. The difference of AGDMap to Praetor [8] is that we consider wide simple gate decomposition during cut enumeration. AGDMap will enumerate K-feasible cuts for 2-input gates. But for 3 or more input simple gates, AGDMap will enumerate all cuts and

Authorized licensed use limited to: Dalian University of Technology. Downloaded on January 18,2025 at 07:20:51 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2024.3515876

perform simultaneous mapping and decomposition for cuts with size larger than K through bin-packing.

Below are some terminologies we use in this paper.

- Terminologies:

$sup(u)$	Support set of node u
$fanout(u)$	Fanout number of node u
$root(cut)$	LUT at the root node of the bin-packing mapping of cut
$bp(cut)$	Bin-packing area of cut

For a given bin-packing and mapping solution of a wide cut as shown in Figure 4(b), $root(cut)$ shown in red color is the LUT of the fanout node. Obviously, $root(cut)$ of a K-feasible cut is the cut itself.

Similar to Praetor algorithm [8], AGDMap also works in two steps: cut enumeration with pruning, cut selection for mapping generation. Generic logic synthesis is performed before mapping, followed by simple gate collapse to form wide simple gates. Basically, the input netlists for mapping include 2-input gates and multi-input simple gates. Cut enumeration is performed for all gates from PIs to POs in a topological order. For wide gates, we no longer require the enumerated cuts to be K-feasible.[1] For a cut with input size larger than K, we will perform bin-packing algorithm to form a K-LUT mapping solution as discussed in the previous section. Our cut enumeration algorithm is shown in Algorithm 2.

Algorithm 2 Cut Enumeration

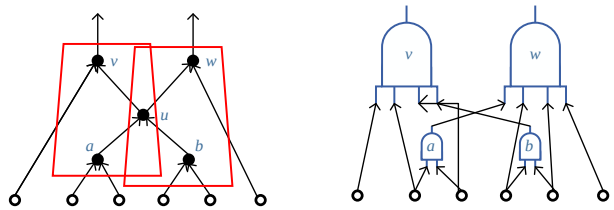
CutEnumeration(G(V, E))
for every *u* in V from PIs to POs
  let list(u) = ∅
  if (u is a PI)
    let $cut = \{u\}$, $A(cut)$=1.
    insert cut into list(u)
  if (u is a PO)
    let v be the driving node to u.
    for each *cut* in list(v)
      $let\ cut_1 = cut, A(cut_1) = \frac{A(cut)}{fanout(v)}$
      insert cut_1 into list(u)
  if (u is an internal node)
    let $sup(u) = \{v_0, v_1, .., v_{w-1}\}$
    for each combination of cuts in list(v_i)
      let $cut = \bigcup root(cut_i)$, and
      $A(cut) = \sum \frac{A(cut_i)-1}{fanout(v_i)} + bp(cut)$.
      Insert cut into list(u) with ranking and pruning.
    create a trivial $cut = \{u\}$ and
      $A(cut) = \min_{\forall cut_i \in list(u)} A(cut_i) + 1$.
    insert cut into list(u)

The pruning and ranking process can be done similarly to that in Praetor [8], except that we use the root cut size in cut ranking. We define a cut ranking order as follows.

[1] Actually, we could consider wide cuts even for 2-input nodes. But this is unnecessary since there is no further gate decomposition can be done for 2-input nodes.

Definition: The ranking of cut_1 is higher than cut_2 if
$\begin{cases} A(cut_1) < A(cut_2), or \\ A(cut_1) = A(cut_2)\ and\ |root(cut_1)| < |root(cut_2)|. \end{cases}$

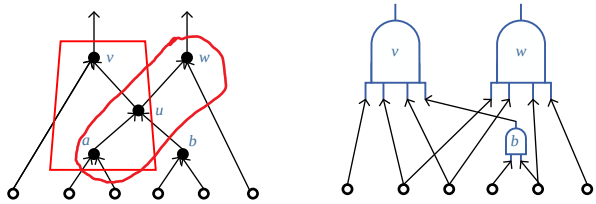
Based on this ranking, one can prune low ranked cuts based on a given capacity C of cut list of every node. After cut enumeration, cut selection can be performed for mapping generation. Initially, all gates are marked internal, and all POs and PIs are mark LUT-roots. Then, we traverse all nodes from POs to PIs. If a node is marked as LUT-root, we will select a cut with the smallest effective area and mark all inputs to the cut as LUT-roots. Otherwise, we skip the node. The process is repeated until we finish traversing all nodes in a design. After cut selection, a mapping solution can be formed by implementing each selected cut with an LUT for K-feasible cuts or bin-packing solution for wide cuts for all LUT-root nodes.



(a) different cuts of u are used by v and w (b) solution with 4 4-LUTs

Figure 5 Cut selection incur logic duplication.

One problem is that if a multi-fanout node is covered by multiple selected cuts, logic duplication is needed for mapping generation. For example, if we select cuts for node v and w using different cuts of u, we will get a mapping solution of 4 4-LUTs as shown in Figure 5. However, if we select cuts for nodes v and w using the same cut of u, we will get a mapping solution of 3 4-LUTs as shown in Figure 6.



(a) same cut of u is used by v and w (b) solution with 3 4-LUTs

Figure 6 Sharing subcut leads to better mapping.

To alleviate this problem, we propose an iterative cut selection algorithm with sharing based cost as follows:

$$A'(cut) = \sum_{v_i \in cut} \big(1 - mark(v_i)\big) \frac{A(v_i)}{fanout(v_i)} + bp(cut), \qquad (2)$$

where $mark(v_i) = 1$ if v_i is marked as LUT-root during the current iteration of cut selection and 0, otherwise. The fanout count of a node is based on the previous mapping solution, or the original netlist for the first iteration of cut selection. After one iteration of cut selection, the fanout count of an LUT-root node can be calculated as the number of selected cuts use this node as inputs or, 1 for an internal node. The intention of this approach is to select cuts with shared inputs of other cuts (nodes marked as LUT roots) to reduce logic duplications. A pseudo code of our cut selection algorithm for the *i*-th iteration

Authorized licensed use limited to: Dalian University of Technology. Downloaded on January 18,2025 at 07:20:51 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2024.3515876

is shown in Algorithm 3. We will perform multiple cut selection process until the area does not improve.

Algorithm 3 Iterative Cut Selection

CutSelection(G(V, E), *i*)
$mark_i(u) = \begin{cases} 1, & if\ u\ is\ PI/PO \\ 0, & if\ u\ is\ internal\ node \end{cases}$
for every u in from POs to PIs
if (u is a PO)
let v by the driving node to u.
$mark_i(v) = 1$
if (u is an internal node and $mark_i(u) = 1$)
compute $A'(cut)$ for all cuts of u based on Eq.2.
select a cut with the smallest cost and set
$mark_i(v) = 1$ for all $v \in cut$
return $area_i$ = node count with $mark_i(u) = 1$

IV. Cut Enumeration And Pruning For Wide Gates

It is presented in [8] that to enumerate all possible K-cuts of a node, one can enumerate all K-cuts of its fanin nodes and combine them to form all K-cuts of that node. For a two-input node, the cut enumeration can be done in $O(C^2)$, where C is the number of K-cuts of the input nodes, which is normally 20~50 for K of 6 with ranking and pruning strategy. However, for a node with w inputs, where w can be over 100 for computation-oriented designs, this cut enumeration will have a complexity of $O(C^w)$, which is too high to be practical.

In this section, we propose a fast cut enumeration heuristic at a runtime complexity of $O(wC^2)$.

Let sup(u) = {v_0, v_1, .., v_{w-1}} for a w-in node u. We first combine cuts of v_0 and v_1 to form a partial cut list of u. We call those cuts as partial cuts of u since they are formed by cuts from a partial set of inputs. Then, we combine the partial cuts with cuts of other inputs one-by-one until we finish the cut combination for v_{w-1} to get the full cut list of u. A pseudo code of our algorithm is shown in Algorithm 4.

Algorithm 4 Cut Enumeration for Wide Gates

CutEnumeration(u)
let $\sup(u) = \{v_o, v_1, .., v_{w-1}\}$.
let $cut = \emptyset$, $A(cut) = 0, list_1 = \{cut\}$.
for *i* from 0 to *w-1* {
let $list = \emptyset$
for each cut_1 in list(v_i) {
for each cut_2 in $list_1$ {
let $cut = root(cut_1) \cup cut_2$.
$A(cut) = A(cut_2) - bp(cut_2) + \frac{A(cut_1)-1}{fanout(v_i)} + bp(cut)$
insert cut into $list$ with ranking and pruning
}
}
let $list_1 = list$
}
let $cut = \{u\},\ A(cut) = \min_{cut' \in list_1} A(cut') + 1$
insert cut into $list_1$
return $list_1$

To calculate the effective area $A(cut)$, we subtract 1 from $A(cut_1)$, while bin-packing area $bp(cut_2)$ from $A(cut_2)$. The reason is that $bp(cut)$ includes $bp(cut_2)$ already, but only the root cut for cut_1.

Cuts in a (partial) cut list are ranked in increasing order of effective area and root cut size. Besides, we limit the capacity of each cut list to be a given C.

In our preliminary work [18], we chose to not perform bin-packing for partial cuts to save runtime. However, more careful study shows that ignoring bin-packing area for partial cuts will lead to less accurate effective area estimation, thus affects ranking and pruning. Because the partial cut size can be $O(Kw)$ for a w -input node and bp(cut) may be not negligible for large w. Our test results show that with consideration of bin-packing area for partial cuts, we can reduce the mapping area from 5695 to 5261 for 2mm, which is an 8.25% reduction. As a result, we choose to perform bin-packing for all the partial cuts.

The entire flow of our algorithm is shown in Algorithm 5.

Algorithm 5 AGDMap(G(V, E))

generic logic synthesis
gate decomposition into 2-bounded netlist
simple gate collapse to form wide gates
//cut enumeration with bin-packing
CutEnumeration(G(V, E))
//iterative cut selection
let $A^* = \infty$, $mark^*(u) = 0$ and $mark^*(PI/PO) = 1$
i = 0
do {
flag = false
A = CutSelection(G(V, E), i++)
if ($A < A^*$) {
flag = true;
$A^* = A$;
$mark^*(u) = mark_i(u)\ for\ all\ u \in V$
} while (flag);
form mapping solution based on $\{mark^*(u)\}$, i.e., form LUTs for every $mark^*(u) = 1$
return the mapping solution

Let W be the bound of wide gate input size. The cut size of a wide gate is O(KW). First-fit bin-packing can be done in $O(K^2W^2)$. Cut enumeration complexity is $O(WC^2)$. Thus, the total runtime of cut enumeration with bin-packing is $O(K^2W^3C^2|V|)$. The runtime of cut selection is O(C|V|) since each node has O(C) cuts. Thus, the total runtime of our mapping algorithm is $O(K^2W^3C^2|V|)$, which is polynomial to W and |V|. C is a pre-given constant for cutlist size, which is 30 in our implementation.

Our test results show that most designs can be done in few seconds for W of no more than 100 on an AMD Ryzen 9 CPU at 2.2GHz. For larger W, the runtime can reach several minutes, which is still acceptable for modern computing power.

Authorized licensed use limited to: Dalian University of Technology. Downloaded on January 18,2025 at 07:20:51 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2024.3515876

V. Cut Enumeration For Delay Optimization

Due to the versatility of cut enumeration approach, the extension of our algorithm for delay minimization is quite straightforward. Basically, we compute a delay value for each cut, besides its effective area, for ranking and pruning.

Let $\sup(cut) = \{v_0, v_1, .., v_{k-1}\}$ for a K-feasible cut. The delay value we compute is

$$D(cut) = \max_{i\in[0,k-1]} \big(D(v_i)\big) + 1. \quad (3)$$

We have $D(PI) = 0$. For an internal node or PO node u with cut list $list(u)$, the delay value of u is

$$D(u) = \min_{cut\in list(u)} D(cut). \quad (4)$$

For a wide cut, we will performance delay-driven bin-packing [2]. Let $d(cut)$ be the LUT level of the bin-packing solution of cut. The delay value of a wide cut is

$$D(cut) = \max_{i\in[0,k]} \big(D(v_i)\big) + d(cut). \quad (5)$$

Actually, a more accurate delay calculation is to consider different delays from different inputs to the root node. For the ease of presentation, we omit the detail of this delay formulation in this paper.

For delay optimization, the ranking of cuts can be done in increasing order of (delay, area, cutsize). For cut pruning, we choose to keep both min-delay and min-area cuts and a number of area/delay tradeoff cuts for each cutsize based on a given cutlist capacity. In this way, we guarantee to find the min-delay mapping solutions with consideration of area minimization as well. Due to the heuristic nature of cut pruning, we cannot guarantee the optimality of delay optimization of our algorithm. But our test results show that our algorithm is on par with state-of-the-art lossless synthesis based mapping algorithm [13] for all the designs we tested.

VI. Experimental Results

To demonstrate the effectiveness of our algorithm, named AGDMap, we implement our algorithm on top of ABC [9] with a new command agdmap [19]. Our implementation is based on the AIG netlist [9], thus, only wide AND gates is considered in our implementation. The wide AND gates are formed for AIG nodes from POs to PIs. For each node, we identify all its fanins without multiple fanouts and no negative edge marks.

15 computation-oriented designs from EPFL [16] and HLS generated PolyBench [15] are used in our test. The design statistics are shown in Table 1.

We perform ABC commands “dsd; resyn; resyn2; dchoice” before mapping.[2] Our algorithm is compared with the state-of-the-art lossless synthesis mapping algorithm “if -K 6 -a” in ABC [13]. “dchoice” command will generate a number of gate decomposition choices during logic synthesis. The command “if” will consider those choices during mapping. Besides, we also compared our algorithm with an industrial tool AMD Vivado 2019.2 [17]. The comparison with Vivado is a combined effect of logic synthesis with mapping, since we have no control to run mapping only in Vivado.

Table 1 Wide gate statistics. "gate" means gate count in the design. "wide gate" means wide simple gates. "max size" is maximum size of simple gates, "average size" is the average simple gate size.

design	gate	wide gate	max size	average size
adder	1019	0	2	2
arbiter	1634	145	11	4
cavlc	476	96	6	3
div	18589	2052	8	3
log2	22020	2075	23	5
mem_ctrl	33201	6691	14	3
multiplier	21883	1387	4	3
sin	3799	380	14	5
2mm	12267	414	125	24
3mm	22003	535	175	35
aes_core	23587	3453	5	3
atax	5422	234	114	18
deriche	23257	289	385	71
trisolv	9686	172	225	45
mips	10851	1784	32	3

Table 2 Comparison for Area Minimization.

design	AGDMap	Vivado		ABC Lossless	
	LUT	LUT	vs AGDMap	LUT	vs AGDMap
adder	192	326	1.70	199	1.04
arbiter	651	1691	2.60	674	1.04
cavlc	124	93	0.75	108	0.87
div	3297	12502	3.79	4112	1.25
log2	6539	8749	1.34	7846	1.20
mem_ctrl	10288	11197	1.09	10646	1.03
multiplier	4798	5929	1.24	5820	1.21
sin	1307	1681	1.29	1435	1.10
2mm	5262	7046	1.34	6271	1.19
3mm	9631	12444	1.29	11431	1.19
aes_core	3574	3571	1.00	4300	1.20
atax	2306	2860	1.24	2801	1.21
deriche	10517	13389	1.27	12852	1.22
trisolv	4133	5290	1.28	4941	1.20
mips	3235	2759	0.85	3288	1.02
geomean			1.35		1.13

Our test results for area minimization are shown in Table 2, where columns “LUT” list the number of LUTs in the mapping solutions. Our results show that, when compared with AGDMap, Vivado’s area is 35% larger, while lossless-synthesis is 13% larger. The runtime of AGDMap on an AMD Ryzen 9 5900X CPU can be done in less than a few seconds, except 4 minutes for deriche with many very wide gates. Consider that modern FPGAs are widely used in computation

[2] Command dsd may not run through for some designs. In this case, we skip dsd in our script.

Authorized licensed use limited to: Dalian University of Technology. Downloaded on January 18,2025 at 07:20:51 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2024.3515876

acceleration, our results show the big advantages of AGDMap over existing FPGA mapping algorithms.

Table 3 Impact of dchoice to AGDMap."geomean'" are the geometric mean results without arbiter and div.

design	AGDMap			ABC Lossless	
	w/o dchoice	w/ dchoice	w/ vs w/o	LUT	vs. w/o dchoice
adder	192	192	1.00	199	1.04
arbiter	6554	651	0.10	674	0.10
cavlc	108	124	1.15	108	1.00
div	6537	3297	0.50	4112	0.63
log2	6763	6539	0.97	7846	1.16
mem_ctrl	10501	10288	0.98	10646	1.01
multiplier	4805	4798	1.00	5820	1.21
sin	1308	1307	1.00	1435	1.10
2mm	5247	5262	1.00	6271	1.20
3mm	9636	9631	1.00	11431	1.19
aes_core	3549	3574	1.01	4300	1.21
atax	2345	2306	0.98	2801	1.19
deriche	10504	10517	1.00	12852	1.22
trisolv	4133	4133	1.00	4941	1.20
mips	3199	3235	1.01	3288	1.03
geomean			0.82		0.93
geomean'			1.01		1.13

Notice that dchoice will invoke a round of resyn and resyn2 and save intermediate choices. Thus, it will affect synthesis as well. For example, the AIG node number of arbiter can be reduced from 32767 to 2023 by dchoice. We list AGDMap results on input netlists with and without dchoice in Table 3. Our results show that dchoice mainly affects AGDMap for arbiter and div. By excluding these two designs, the geometric mean of AGDMap results on input netlists without dchoice is actually 1% better than with dchoice, while as is still 13% better than lossless-synthesis.

To show the impact of wide gate input bound on mapping area, we compared the results of limiting wide gate input bound to be 8 with that of no bound. A wide gate input bound W means that we only collapse and form simple gates with input size of no more than W. Our results are shown in Table 4. It shows that by limiting wide gate input bound to be 8, the area increase is only 2%, while runtime reduction is 60%. However, for designs with wide gates with over 100 inputs, the area penalty by limiting W can be as big as 14%.

For delay optimization, we also compared AGDMap with lossless synthesis algorithm with data shown in Table 5. Our results show that AGDMap can get pretty much the same logic levels for all the designs, with 11% area reduction.

Table 4 Impact of Wide Gate Input Bound W on mapping.

design	AGDMap		AGDMap W=8			
	LUT	CPU(s)	LUT	vs W=∞	CPU(s)	vs W=∞
adder	192	0.004	192	1.00	0.004	1.00
arbiter	651	0.078	652	1.00	0.077	0.99
cavlc	124	0.007	124	1.00	0.007	1.00
div	3297	0.397	3297	1.00	0.395	0.99
log2	6539	2.589	6552	1.00	2.392	0.92
mem_ctrl	10288	1.026	10287	1.00	1.028	1.00
multiplier	4798	0.430	4798	1.00	0.424	0.99
sin	1307	0.499	1306	1.00	0.498	1.00
2mm	5262	22.18	5644	1.07	1.875	0.08
3mm	9631	65.07	10217	1.06	4.048	0.06
aes_core	3574	0.27	3574	1.00	0.274	1.01
atax	2306	0.344	2626	1.14	0.522	1.52
deriche	10517	511.0	11169	1.06	4.276	0.01
trisolv	4133	123.0	4206	1.02	2.052	0.02
mips	3235	1.194	3261	1.01	1.197	1.00
geomean				1.02		0.40

Table 5 Comparison for delay Minimization.

design	AGDMap		ABC Lossless			
	LUT	Level	LUT	vs AGDMap	Level	vs AGDMap
adder	192	64	193	1.01	64	1.00
arbiter	717	14	695	0.97	14	1.00
cavlc	125	4	112	0.90	4	1.00
div	5414	857	5044	0.93	854	1.00
log2	7313	65	8236	1.13	66	1.02
mem_ctrl	10823	20	11064	1.02	20	1.00
multiplier	5364	53	6063	1.13	53	1.00
sin	1385	35	1470	1.06	35	1.00
2mm	6002	7	6893	1.15	7	1.00
3mm	9970	8	12248	1.23	8	1.00
aes_core	3583	5	4368	1.22	5	1.00
atax	3098	6	3690	1.19	6	1.00
deriche	10597	8	16961	1.60	8	1.00
trisolv	4219	9	5477	1.30	9	1.00
mips	3325	7	3436	1.03	7	1.00
geomean				1.11		1.00

To show the impact of AGDMap on glue logic designs, we test AGDMap on a set of 20 MCNC designs[3]. Our test results are shown in Table 6. The column "max W" lists the maximum gate input sizes in the designs. We can see that they are much smaller than that of the computation-oriented designs. Nevertheless, AGDMap can still reduce the area of the lossless synthesis algorithm by 6%, while loses to Vivado by 5%.

[3] Downloaded from https://github.com/verilog-to-routing/vtr-verilog-to-routing/tree/master/vtr_flow/benchmarks/blif/2.

Authorized licensed use limited to: Dalian University of Technology. Downloaded on January 18,2025 at 07:20:51 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2024.3515876

Table 6 Comparison of AGDMap with Vivado and ABC Lossless on MCNC Designs.

design	max W	AGDMap	Vivado		ABC Lossless	
		LUT	LUT	vs AGDMap	LUT	vs AGDMap
alu4	7	213	241	1.13	210	0.99
apex2	7	47	449	9.55	52	1.11
apex4	9	514	155	0.30	463	0.90
bigkey	6	694	681	0.98	771	1.11
clma	6	198	272	1.37	207	1.05
des	6	594	475	0.80	694	1.17
diffeq	13	606	664	1.10	621	1.02
dsip	4	677	457	0.68	1353	2.00
elliptic	19	1699	1770	1.04	1822	1.07
ex1010	9	560	170	0.30	538	0.96
ex5p	5	101	100	0.99	95	0.94
frisc	18	1594	1809	1.13	1718	1.08
misex3	13	239	232	0.97	246	1.03
pdc	10	772	559	0.72	766	0.99
s298	11	520	232	0.45	478	0.92
s38417	26	2093	2349	1.12	2201	1.05
s38584.1	10	2375	2176	0.92	2485	1.05
seq	10	518	546	1.05	526	1.02
spla	7	169	168	0.99	171	1.01
tseng	8	634	657	1.04	651	1.03
geomean	9			0.95		1.06

VII. Conclusion

In this paper, we present an FPGA mapping algorithm with adaptive simple gate decomposition. Our source code and HLS generated benchmarks are available in [19] for research purpose. The advantage over existing choice-generation with gate decomposition algorithms is that our algorithm can automatically generate and identify needed gate decompositions guided by cut enumeration and ranking. As a result, our algorithm can avoid the dilemma that pre-generating a large set of gate decomposition choices will lead to long runtime, or a small choice set will result to qor degradation.

Our test results shows that our algorithm can improve the mapping area over the state-of-the-art lossless synthesis mapping algorithm [13] by 13% with as much as 25% on a set of computation-oriented designs with many wide simple gates. Besides, our algorithm can improve the results of a leading industrial tool Vivado by 35%, which is very significant for such a mature industrial tool.

Considering the case that modern FPGAs are widely used for computation acceleration, we think AGDMap should have big impact for modern applications. In the future, we will try to improve the pruning strategy for bigger wide gates and consider complex gate decompositions.

Acknowledgment

The author wants to thank Mr. Zhiyong Zhang and Feng Chang for their help in preparing many of the test data. Besides, Mr. Zhiyong Zhang also helped to open source our code in github for research and evaluation purpose.

References

[1] R. Francis, J. Rose, Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," 28th Design Automation Conference, pp. 227-233, 1991.
[2] R. Francis, J. Rose, Z. Vranesic, "Technology Mapping of Lookup Table-Based FPGAs for Performance," International Conference on Computer-Aided Design, pp. 568-571, 1991.
[3] R. Murgai, N. Shenoy, R. Brayton, A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Lookup Architectures," IEEE International Conference on CAD, pp. 564-567, 1991.
[4] A. Farrahi, M. Sarrafzadeh, "Complexity of the Lookup-table Minimization Problem for FPGA Technology Mapping," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, No. 11, pp. 1319-1332, 1994.
[5] J. Cong, Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, No. 1, pp. 1-12, 1994.
[6] E. Lehman, Y. Watanabe, J. Grodstein, H. Harkness, "Logic Decomposition during Technology Mapping," International Conference on CAD, pp. 264-271, 1995.
[7] J. Cong, Y. Hwang, "Simultaneous Depth and Area Minimization in LUT-Based FPGA Mapping," International Symposium on FPGAs, pp. 68-74, 1995.
[8] J. Cong, C. Wu, Y. Ding, "Cut Ranking and Pruning: Enabling A General and Efficient FPGA Mapping Solution," International Symposium on FPGAs, pp. 29-35, 1999.
[9] R. Brayton, A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," International Conference on CAD, pp. 24-40, 2010.
[10] G. Chen, J. Cong, "Simultaneous Logic Decomposition with Technology Mapping in FPGA Designs," International Symposium on FPGAs, pp. 48-55, 2001.
[11] D. Chen, J. Cong, "DAOmap: A Depth-Optimal Area Optimization Mapping Algorithm for FPGA Designs," International Conference on Computer-Aided Design, pp. 752-759, 2004.
[12] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, T. Kam, "Reducing Structural Bias in Technology Mapping," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 25, No. 12, pp. 2894-2903, 2006
[13] A. Mishchenko, S. Chatterjee, R. Brayton, "Improvements to Technology Mapping For LUT-Based FPGAs," International Symposium on FPGAs, pp. 41-49, 2006.
[14] A. Mishchenko, S. Cho, S. Chatterjee, R. Brayton, "Combinational and Sequential Mapping with Priority Cuts," International Conference on CAD, pp. 354-361, 2007.
[15] L. Pouchet, *et al.*, "Polybench: The Polyhedral Benchmark Suite," 2012.
[16] L. Amaru, P. Gaillardon, G. Micheli, "The EPFL Combinational Benchmark Suite," 2015.
[17] Xilinx, Inc., "Vivado Design Suite User Guide," 2019.
[18] L. Fan, C. Wu, "FPGA Technology Mapping with Adaptive Gate Decomposition," International Symposium on FPGAs, pp. 135-140, 2023.
[19] AGDMap, https://github.com/yongshiwo/abc.git

Chang Wu (Member, IEEE) received the B.S. degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 1986, the M.S. degree from the Institute of Pattern Recognition and Artificial Intelligence at the same university in 1989, and the Ph.D. degree in computer science from University of California at Los Angeles in 1999.

Authorized licensed use limited to: Dalian University of Technology. Downloaded on January 18,2025 at 07:20:51 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2024.3515876

From 1989 to 1995, he worked at Beijing Integrated Circuit Design Center as a Senior Software Engineer. From 1999 to 2003, he cofounded and worked as a senior software engineer at Aplus Design Technologies, Inc., Los Angeles, CA. He also worked at Magma Design Automation, Xilinx and Cadence from 2003 to 2014. Since 2014, he works as a professor at Fudan University, Shanghai, China. His research area includes logic synthesis and optimization, high level synthesis, architecture design and software and hardware co-optimization for high performance computing and accelerator designs.

Authorized licensed use limited to: Dalian University of Technology. Downloaded on January 18,2025 at 07:20:51 UTC from IEEE Xplore. Restrictions apply.
© 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.