Cutless FPGA Mapping

Alan Mishchenko **Sungmin Cho** **Satrajit Chatterjee** **Robert Brayton**

Department of EECS, University of California, Berkeley

{alanmi, smcho, satrajit, brayton}@eecs.berkeley.edu

Abstract

The paper presents a new algorithm for FPGA technology mapping into K-input LUTs. The algorithm avoids cut enumeration by incrementally computing and updating one good K-feasible cut at each node of the subject graph. The main advantage of the algorithm is that it works for very large LUT size while offering dramatic improvements in memory and runtime. For 10-input LUTs, the memory is reduced 20x and runtime 30x, compared to fast state-of-the-art mapping based on resource-aware cut enumeration. Experiments show that optimum-depth FPGA mapping is often found using the proposed algorithm but at the cost of some area penalty. The ongoing work is focusing on improving the quality of area recovery.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*Optimization*; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays*; J.6 [**Computer-Aided Engineering**]: *Computer-aided design (CAD)*

General Terms

Algorithms

Keywords

FPGA, Technology Mapping, And-Inverter Graphs, Cut Enumeration, Area Recovery

1 Introduction

Technology mapping for Field-Programmable Gate Arrays (FPGAs) is the process of transforming a technology-independent logic network, called the *subject graph*, into a network of logic nodes, each of which can be realized as one K-input *look-up table* (LUT). A traditional LUT can implement any Boolean function up to K inputs. The subject graph is often represented as an *AND-Inverter Graph* (AIG) composed of two-input ANDs and inverters.

Most structural methods of FPGA mapping [6][11] start by computing all cuts for each AIG node. Next, the AIG nodes are traversed in a topological order and a dynamic programming approach is used to find an optimum-depth LUT mapping of the AIG. This mapping can often be substantially improved by applying area-recovery heuristics [3][10][11] to reduce the number of LUTs while preserving the depth of the LUT network.

It should be noted that such FPGA mapping algorithms as FlowMap [2] and CutMap [4] do not compute all cuts. However, good cuts in these mappers are found using the maximum-flow algorithm that has high computational complexity. As a result recent state-of-the-art mappers [6][11] use cut enumeration rather than maximum flow.

In a large class of programmable architectures, the LUT size K varies between 3 and 6. For these relatively small LUT sizes, the traditional methods for LUT mapping based on cut enumeration work quite well. For K equal to 4 or 5, exhaustive cut enumeration [12][5] can be applied, resulting in an average of 10-40 cuts stored at each node. When the LUT size is 6, exhaustive cut enumeration may lead to 100+ cuts per node. Cut representation takes substantial memory when mapping large Boolean networks. To remedy the situation, a partial cut enumeration can be used, which heuristically prune the cuts, resulting in reduced memory requirements [5]. However, cut pruning may result in losing good cuts. In this case, the depth-optimality of mapping is not guaranteed.

Another class of modern programmable architectures realizes logic networks using macro-cells, which typically contains LUTs and other logic gates. A straight-forward way of mapping logic into programmable macro-cells starts by computing all K-input cuts for each node where K is the number of macro-cell inputs. Unlike a K-input LUT, a macro-cell cannot implement all logic functions of K inputs. Therefore, the local function of each cut is computed in terms of the cut inputs, and only those cuts whose logic function can be expressed by the macro-cell will be kept for potential mapping. However, this approach is not practical because a macro-cell often has 10 or more inputs while the number of 10-input cuts is extremely large for all but the smallest benchmarks.

This paper presents a new algorithm for FPGA mapping whose runtime and memory requirements are linear in the number of nodes in the subject graph. The proposed algorithm is "cutless" in the sense that it avoids complete or even partial cut enumeration and it does not use maximum flow computation. Instead, it computes and incrementally updates only one cut at each node using linear-time algorithms and thus departs from traditional methods.

The consequences of not having to compute all cuts are remarkable. In terms of memory, the new algorithm requires storage space for only one cut at each node. The runtime is also linear in the subject graph size because the algorithm performs a fixed amount of work at each node.

The actual runtime may depend on area recovery performed but it is typically at least an order of magnitude less than that of state-of-the-art traditional methods.

The only current drawback of the proposed algorithm is an area penalty. We believe that this difficulty can be addressed by developing efficient heuristic area recovery, comparable to those used in cut-enumeration-based methods. Our preliminary work on area recovery is encouraging but not sufficient. Below we discuss the difficulties and propose some solutions hoping to improve them for the final version of the paper.

The rest of the paper is organized as follows. Section 2 describes some background. Section 3 reviews the traditional FPGA mapping algorithm. Section 4 describes the new algorithm. Section 5 discusses the relation between the new algorithm and the known algorithms. Section 6 reports experimental results. Section 7 concludes the paper and outlines future work.

2 Background

A *Boolean network* is a directed acyclic graph (DAG) with nodes corresponding to logic gates and directed edges corresponding to wires connecting the gates. The terms network, Boolean network, and circuit are used interchangeably in this paper.

A node has zero or more *fanins,* i.e. nodes that are driving this node, and zero or more *fanouts*, i.e. nodes driven by this node. The *primary inputs* (PIs) of the network are nodes without fanins in the current network. The *primary outputs* (POs) are a subset of nodes of the network. If the network is sequential, the flip-flop outputs/inputs are treated as additional PIs/POs. In the following, it is assumed that each node has a unique integer number called its *node ID.*

A network is *K-bounded* if the number of fanins of each node does not exceed *K*. A *subject graph* is a *K*-bounded network used for technology mapping. Any combinational network can be represented as an AND-INV graph (AIG), composed of two-input ANDs and inverters. Without limiting the generality, in this paper we assume subject graphs to be AIGs.

A *cut C* of node *n* is a set of nodes of the network, called *leaves*, such that each path from a PI to *n* passes through at least one leaf. A *trivial cut* of the node is the cut composed of the node itself. A cut is *K-feasible* if the number of nodes in it does not exceed *K.*

A *host* node *h* is an imaginary node without logic function, whose fanins are the PO nodes of the network. A cut *C* of the network is a cut of the host node. A cut of a node (or a network) is said to be *dominated* if there is another cut of the same node (or the host node), which is contained, set-theoretically, in the given cut.

A *fanin* (*fanout*) *cone* of node *n* is a subset of all nodes of the network reachable through the fanin (fanout) edges from the given node. A *maximum fanout free cone* (MFFC) of node *n* is a subset of the fanin cone, such that every path from a node in the subset to the POs passes through *n.* Informally, the MFFC of a node contains all the logic used exclusively by the node. When a node is removed or substituted, the logic in its MFFC can also be removed.

The *level* of a node is the length of the longest path from any PI to the node. The node itself is counted towards the path lengths but the PIs are not. The network *depth* is the largest level of an internal node in the network. The delay and area of FPGA mapping is measured by the depth of the resulting LUT network and the number of LUTs in it.

3 Traditional FPGA mapping

Figure 3.0 gives an outline of the classical FPGA technology mapping [2][10], as implemented in [11].

```
traditionalMap( aig, lut_size )
{
    // compute all lut_size-feasible cuts at each node and save them
    traditionalMapCutEnumeration( aig, lut_size );

    // find minimum-depth cut and save itt as the best cut at each node
    traditionalMapDepthOriented( aig, lut_size );

    // update the best cut at each node to save area while preserving depth
    traditionalMapAreaRecovery( aig, lut_size );

    // return the set of nodes, one for each LUT used in the final mapping
    traditionalMapDeriveFinalMapping( aig, lut_size );
}
```

Figure 3.0. The traditional FPGA mapping.

3.1 Depth-oriented mapping

Figure 3.1 shows the pseudo-code of depth-oriented mapping performed in the traditional FPGA mapping.

The AIG nodes are considered in a topological order. At each node, all cuts are enumeration and an optimum-depth cut is found. This cut along with its level is stored at the node. The level of a cut is computed by adding 1 to the maximum level of the cut fanins.

```
traditionalMapDepthOriented( aig, lut_size )
{
    for each aig node n in topological order {
        cut = findCutMinimizingDepth( n );
        setLevel( n, getLevel(cut) );    setCut( n, cut );
    }
}

findCutMinimizingDepth( node )
{
    cut_best = NULL;
    for each cut c of node
        if ( cut_best == NULL or getLevel(cut_best) > getLevel(c) )
            cut_best = c;
    return cut_best;
}

getLevel( cut )
{
    level_max = 0;
    for each node m in cut
        level_max = max( level_max, getLevel(m) );
    return 1 + level_max;
}
```

Figure 3.1. Depth-oriented traditional FPGA mapping.

3.2 Area recovery

Figure 3.2 contains the pseudo-code of the area recovery performed as part of the traditional FPGA mapping. The detailed account of the heuristics based on the area flow and exact local area can be found in [11].

```
traditionalMapAreaRecovery( aig, lut_size )
{
    computeRequiredTimes( aig );
    for each aig node n in topological order {
        cut = findCutMinimizingAreaFlow( n );
        setLevel( n, getLevel(cut) );    setCut( n, cut );
    }
    computeRequiredTimes( aig );
    for each aig node n in topological order {
        cut = findCutMinimizingExactLocalArea( n );
        setLevel( n, getLevel(cut) );    setCut( n, cut );
    }
}
```

Figure 3.2. Area recovery in traditional FPGA mapping.

3.3 Producing a mapped network

The pseudo-code of the procedure used in the traditional FPGA mapping to derive the final LUT network is shown in Figure 3.3. A similar procedure was presented in [12].

The procedure assumes that one K-feasible cut is assigned to each node. In the case of the traditional mapping, this is the best cut chosen to map the node; in the proposed algorithm, this is the K-feasible cut assigned at the node.

The procedure supports two sets of AIG nodes: the nodes used in the mapping (*M*) and the nodes currently in the frontier (*F*). Both sets are initialized to the set of POs. While the frontier is not empty, one node (*n*) is extracted from it, the cut of this node is computed, and the leaves of this cut are explored. If a leaf (*m*) already belongs to the mapping or is a PI, this leaf is skipped; otherwise, the leaf is added to both the mapping and the frontier. When the frontier is empty, the procedure terminates and returns the set *M* of nodes used in the mapping.

```
fastMapDeriveFinalNetwork( aig, lut_size )
{
    // set the mapped nodes and the frontier to be the set of PO nodes
    M = POs;   F = POs;

    // explore each node in the frontier
    while ( $F \neq \varnothing$ ) {
        n = getExtractNode( F );
        cut = getCut( n );
        for each node m in cut {
            if ( $m \in M$ or $m \in$ PIs )
                continue;
            $M = M \cup m$;   $F = F \cup m$;
        }
    }

    // return the set of nodes, one for each LUT used in the final mapping
    return M;
}
```

Figure 3.3. Producing the mapped LUT network.

4 Proposed algorithm

The proposed algorithm is similar to the traditional FPGA mapping algorithm in that it performs depth-oriented mapping followed by area recovery. During depth-oriented mapping all nodes of the network are visited in a topological order and two parameters are computed for each node: level and one K-feasible cut. The details of this step are discussed in Section 4.1.

Area recovery is performed using a sequence of passes over each node in the network. In each pass, a new K-feasible cut can be assigned to each node and the node's level can be updated while ensuring that the largest PO level does not increase. To achieve this, required times for all nodes are computed and used during area recovery. The details of this step are given in Section 4.2.

Finally, the mapped LUT network is produced as in the traditional mapping (see above Section 3.3).

4.1 Depth-oriented mapping

Depth-oriented mapping of the subject graph (an AIG) in the proposed algorithm is similar to that in the traditional mapping. The difference is that, instead of computing all cuts, only one cut is computed and stored at each AIG node. The level of a node is determined as in traditional mapping, by counting the number of logic levels of LUTs needed to implement the node while counting from the PIs. When depth-oriented mapping is finished, the largest level of a PO gives the depth of the resulting LUT mapping. Unlike traditional mapping, there is no guarantee that the resulting mapping is depth-optimum but in practice a depth-optimum mapping is often found.

The depth-oriented phase of the algorithm is illustrated using Figure 4.1. The PI nodes are assigned level 0. Next, all AIG nodes are found that can be implemented in terms of the PI nodes using one LUT. These are the nodes under the first wavy line. Next, all the AIG nodes above the first wavy line are found that can be implemented in terms of the PIs and the nodes below the first wavy line. These are the nodes between the first and second wavy lines. The algorithm proceeds until all nodes have been examined and mapped.

Note that the cut for each node is obtained by simply merging the cuts of its children if their levels are equal and otherwise simply adding the child of least level to the cut associated with the other child. If the merged cut is not K-feasible, then the cut is simply the two children, and the level increases by 1 for that node.

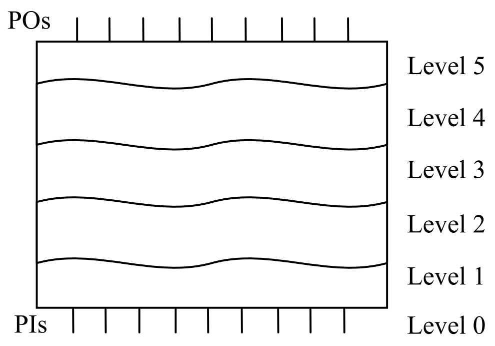

Figure 4.1. Illustration of depth-oriented mapping.

The depth-oriented phase of the algorithm can be implemented in a straight-forward manner, as described above. This implementation requires as many passes over the AIG, as there are mapping levels. Implementation in Figure 4.2 is less straight-forward but computes the node's level and K-feasible cut by visiting each node only once.

```
fastMapDepthOriented( aig, lut_size )
{
    // assign level 0 and elementary cut for each primary input
    for each primary input n {
        setLevel( n, 0 );   setCut( n, {n} );
    }
    // perform mapping of internal nodes by visiting each of them once
    for each AIG node n in topological order {
        // get the fanins of n that are already mapped
        n0 = nodeFanin0( n );   n1 = nodeFanin1( n );
        // determine starting level of the new node
        level = max( getLevel( n0 ), getLevel( n1 ) );
        if ( level == 0 )
            level = 1;
        // determine cuts to be merged at the node
        if ( getLevel( n0 ) == getLevel( n1 ) )  {
            cut0 = getCut( n0 );  cut1 = getCut( n1 );
        }
        else if ( getLevel( n0 ) > getLevel( n1 ) )  {
            cut0 = getCut( n0 );  cut1 = { n1 };
        }
        else if ( getLevel( n0 ) < getLevel( n1 ) )  {
            cut0 = { n0 };  cut1 = getCut( n1 );
        }
        // merge the cuts
        cut = cut0 ∪ cut1;
        // increase the level if the merged cut exceeds the size limit
        if ( getSize( cut ) > lut_size ) {
            cut = { n0 } ∪ { n1 };    level = level + 1;
        }
        // assign level and cut to node n
        setLevel( n, level );    setCut( n, cut );
    }
}
```

Figure 4.2. Efficient depth-oriented mapping.

4.2 Area recovery

Exact area minimization during technology mapping for DAGs is NP-hard [7] and hence not tractable for large circuits. In practice, various heuristics for approximate area minimization are used in the traditional mapping [3][6][10][11]. However, these heuristics cannot be combined with the proposed algorithm because they rely on having a large set of cuts stored at each node or require cut computation using the maximum-flow algorithm.

Currently we use two simple area recovery techniques. The first technique tries to expand each cut used in the current LUT mapping towards the PIs while attempting to increase the number of shared nodes. The second technique tries to replace the current cut by a new cut derived by merging the current cut of one fanin with the unit cut of the other fanins if the other fanin is already used in the mapping. The second technique typically reduces the cut size, which tends to balance the increase of the cut size achieved by the first technique. Both techniques result in cuts with less local area, which leads to an overall reduction of the area of the current mapping. Both methods use the required time information, similar to the traditional method. The depth of a node is allowed to increase as long as the increase does not exceed the required time.

Alternating these two methods works reasonably well but results in more area than in the traditional methods. We plan to experiment with other area recovery methods and report improved results in the final version of the paper.

5 Discussion

5.1 Relation to standard cut enumeration

The proposed approach based on computing one good cut at each node can be seen as an extreme case of resource-bounded standard cut enumeration when only two cuts are stored at each node: the trivial cut and one good cut.

For PIs we store only the trivial cut, which is also the good cut. For internal AND nodes, we store the trivial cut and try to compute the best cut by looking at the four cuts obtained by combining the two cuts of the two fanins. From these four cuts we delete the cuts that are not K-feasible. From the remaining cuts, we prefer the cut that minimizes the level of the node and has the fewest leaves.

It can be shown that the proposed algorithm is depth-optimal for trees. The proof of this statement will be included in the final version of the paper.

5.2 Complexity analysis

The traditional mapper that comes closest to the proposed algorithm, is CutMap [4]. The differences are the following. CutMap uses FlowMap [2] to find an optimum-depth mapping whose complexity is $O(Knm)$, where K is the LUT size while n and m are the number of nodes and edges in the subject graph. Meanwhile, the proposed algorithm uses a simple heuristic to compute a nearoptimum-depth mapping with complexity $O(Kn)$. Area recovery performed by CutMap uses the maximum-flow algorithm with complexity $O(2Kmn^{\lfloor K/2 \rfloor+1})$ [4]. Meanwhile the proposed algorithm performs heuristic area recovery using two method with complexity $O(KCn)$ and $O(Kn)$, where C is a small integer constant limiting the number of nodes that can be added to the cut during the cut expansion.

6 Experimental results

The cutless FPGA technology mapper was implemented in ABC [1], a public-domain synthesis and verification system. The experiments were run on a Windows laptop with a 1.6GHz Pentium 4 CPU and 2GB of RAM. The new mapper was tested and the results are reported in the following two subsections. All results were verified using a combinational equivalence checker in ABC, except for the largest examples shown in Tables 5 and 6.

6.1 Comparison with the traditional mapping

For comparison with state-of-the-art technology mappers we used a selection of public domain benchmarks from previous work [6][11]. To derive the AIG to be used as the subject graph for mapping, the benchmarks were structurally hashed and balanced.

The mapping results for LUT sizes 4, 6, 8, and 10 in terms of depth, the number of LUTs, memory (in megabytes), and runtime (in seconds) are shown in Tables 1-4. Columns "old" shows the results obtained by a state-of-the-art mapper based on exhaustive cut enumeration with improvements proposed in [11]. Columns "new" shows the results of mapping by the proposed mapper.

The pruning parameters used by cut enumeration were set as follows. At each node, at most 2,000 cuts are computed. The resulting cuts were sorted in the increasing number of inputs. 1,000 of the smallest cuts were stored at the node and allowed to propagate during cut enumeration. For small values of K ($K \leq 6$), the limit of 1000 cuts a node is never exceeded. In this case, the traditional mapping finds an optimum-depth solution.

Table 1 shows that for small LUT sizes the new mapper almost always produces depth-optimal mappings. The fact that it gets better depths for $K = 8$ and $K = 10$ is an artifact due to the traditional method computing only a subset of K-input cuts for large K. For the same reason, the runtime of cut computation for $K = 10$ is less than for $K = 8$.

Table 2 shows that the new mapper loses about 5-15% in area, compared to the traditional FPGA mapping. This is because the currently implemented area recovery heuristics are weaker than those used in the traditional mapping. We intend to improve area recovery for the final version of this paper (due December 18). However, it should be noted that even without weak area recovery, such applications as fast prototyping and hardware emulation, can tolerate the increase in area provided that the mapper is fast and memory-efficient.

Table 3 and 4 show that the new mapper dramatically reduces memory and runtime, compared to a traditional mapper based on improvements proposed in [11]. For 10-input cuts, the gains are 30x in memory and 20x in runtime. These gains would be even larger if cut pruning was not used during cut enumeration in the traditional method.

6.2 Performance on large benchmarks

The second set of experiments was designed to show the performance of the new mapper on large benchmarks with large LUT sizes. This experiment was performed by considering multiple timeframes of sequential benchmark *wb_conmax.v* with 1130 PIs, 1416 POs, and 770 latches taken from the IWLS 2005 benchmarks set [8]. This is relevant to hardware emulation where large designs are unrolled and the resulting logic is emulated in hardware to achieve faster simulation.

Table 5 shows the results of FPGA mapping for LUT size 10. The first column shows the number of timeframes used to unroll *wb_conmax.v.* The next two columns show the number of levels and nodes in the AIG used for mapping after structural hashing and balancing. The last four columns show depth, the number of LUTs, memory, and runtime of the proposed algorithm. Note that the runtime is linear in the size of the AIG.

Table 6 contains the results of FPGA mapping applied to 100 frames of *wb_conmax.v* using different LUT sizes. The first column of the table shows the LUT size. The last four columns show depth, the number of LUTs, memory, and runtime of the proposed algorithm. Note that runtime is fairly insensitive to cut size – a factor of 4 in cut size increases runtime only about 50%.

Tables 5-6 report runtimes of the depth-oriented phase of FPGA mapping. Area recovery was not performed in this experiment. Tables 5 and 6 demonstrate that the proposed FPGA mapping algorithm has reasonable memory and runtime requirements when applied to AIGs with millions of nodes. (The reason why area of FPGA mapping increased when going from LUT size 10 to LUT size 12 in Table 6 will be investigated.)

7 Conclusions and future work

This paper presents a new algorithm for FPGA technology mapping, which departs from the traditional approaches based on exhaustive or partial cut enumeration. The algorithm has runtime and memory complexity linear in the number of nodes in the subject graph and works well for LUTs with 12 inputs and more.

Upon closer investigation, it is clear that the traditional mapping dramatically over-computes during cut enumeration, resulting in wasted memory and runtime. This is especially true for large LUTs because less than 1% of all computed cuts ever represent a node during technology mapping. Therefore, the development of a linear-time and linear-space algorithm is well motivated. This paper proposes an efficient algorithm of this type and may lead to a new class of FPGA mapping solutions.

The experimental results show that the proposed algorithm, although heuristic in nature, almost always find an optimum-depth mapping. The runtime and memory requirements are at least an order of magnitude less than the state-of-the-art algorithms based on cut enumeration. The drawback of the current implementation of the proposed algorithm is an area penalty. Our ongoing work focuses on several approaches to address this problem and we hope to report improved results in the final version of this paper. It should be noted that some applications, such as hardware emulation, may not be sensitive to the area increase while benefiting greatly from the reduced runtime and memory.

In addition to improving area recovery, future work will extend the mapping algorithm to work for macro-cells, which can implement a subset of Boolean functions with the given number of inputs. For this purpose a modified cut computation procedure will be used, which guarantees that Boolean function of the cut stored at each node can be implemented using the macro-cell. Still another direction of future work is to extend the mapper to work for sequential circuits with choice nodes.

Acknowledgment

This research was supported in part by SRC contracts 1361.001 and 1444.001, and by the California Micro program with industrial sponsors, Altera, Intel, Magma, and Synplicity.

References

[1] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 60912. http://www.eecs.berkeley.edu/~alanmi/abc/

[2] J. Cong and Y. Ding, “FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs”, *IEEE Trans. CAD*, vol. 13(1), Jan. 1994, pp. 1-12.

[3] J. Cong and Y. Ding, “On area/depth trade-off in LUT-based FPGA technology mapping,” *IEEE Trans. VLSI*, vol. 2(2), Jun. 1994, pp 137-148.

[4] J. Cong and Y. Hwang, "Simultaneous depth and area minimization in LUT-based FPGA mapping," *Proc. FPGA ‘95*, pp. 68-74.

[5] J. Cong, C. Wu and Y. Ding, “Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution,” *Proc. FPGA `99*, pp. 29-36.

[6] D. Chen, J. Cong. “DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs,” *Proc. ICCAD ’04*, pp. 752-757.

[7] A. Farrahi and M. Sarrafzadeh, “Complexity of lookup-table minimization problem for FPGA technology mapping,” *IEEE Trans. CAD*, vol. 13(11), Nov. 1994, pp. 1319-1332.

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

[9] C.-C. Kao, Y.-T. Lai, “An efficient algorithm for finding minimum-area FPGA technology mapping". *ACM TODAES*, vol. 10(1), Jan. 2005, pp. 168-186.

[10] V. Manohara-rajah, S. D. Brown, Z. G. Vranesic, “Heuristics for area minimization in LUT-based FPGA technology mapping,” *Proc. IWLS ’04,* pp. 14-21.

[11] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for LUT-based FPGAs", *Proc. FPGA '06*, pp. 41-49. http://www.eecs.berkeley. edu/~alanmi /publications/2006/ fpga06_map.pdf

[12] P. Pan and C.-C. Lin, “A new retiming-based technology mapping algorithm for LUT-based FPGAs,” *Proc. FPGA ’98*, pp. 35-42.

Table 1. Depth comparison of the traditional and proposed FPGA mapping algorithms (see Section 6.1).

Example	K = 4		K = 6		K = 8		K = 10	
	old	new	old	new	old	new	old	new
b14	21	21	14	14	10	11	11	9
b15	22	22	15	15	13	12	13	10
C1355	4	4	4	4	4	4	4	3
C17	1	1	1	1	1	1	1	1
C1908	9	9	6	6	5	5	5	4
C2670	7	7	5	5	4	4	4	4
C3540	12	12	8	9	7	6	7	6
C432	10	10	7	7	6	6	6	5
C499	4	4	4	4	4	4	4	3
C5315	9	10	6	7	6	5	6	5
C6288	25	25	16	16	20	12	20	10
C7552	8	8	6	6	5	5	5	4
C880	8	8	5	5	4	4	4	4
clma	24	25	14	14	10	11	9	8
pj1	15	15	10	10	8	8	8	7
pj2	8	8	5	5	4	4	4	4
pj3	13	13	9	9	7	7	7	6
s15850	14	14	9	9	8	7	8	6
s35932	4	4	3	3	2	2	2	2
s38417	10	10	6	6	5	5	5	4
Ratio	1.00	1.01	1.00	1.01	1.00	0.96	1.00	0.84

Table 2. Area comparison of the traditional and proposed FPGA mapping algorithms (see Section 6.1).

Example	K = 4		K = 6		K = 8		K = 10	
	old	new	old	new	old	new	old	new
b14	2323	2811	1503	2016	1289	1877	1278	1571
b15	3398	3776	2216	2824	1916	2172	1818	2152
C1355	74	74	64	66	64	55	72	66
C17	2	2	2	2	2	2	2	2
C1908	125	130	96	116	75	92	89	75
C2670	216	233	137	152	132	131	133	125
C3540	362	423	257	315	187	207	221	178
C432	100	102	77	81	64	70	56	70
C499	74	74	64	66	58	55	64	66
C5315	508	573	299	392	251	357	276	330
C6288	508	543	518	520	525	316	548	309
C7552	634	752	467	527	431	458	478	442
C880	129	135	92	107	61	88	64	81
clma	6934	7984	4138	5027	2876	3802	2735	3112
pj1	6095	6872	4369	5206	3528	4329	3481	3679
pj2	1337	1552	891	1063	740	836	647	817
pj3	3837	4316	2555	3319	2019	2706	2094	2287
s15850	1294	1391	1046	1119	952	1035	923	960
s35932	3200	3520	2624	2896	2336	2624	2336	2560
s38417	3569	3851	2733	2890	2463	2602	2464	2547
Ratio	1.00	1.09	1.00	1.15	1.00	1.13	1.00	1.04

Table 3. Memory comparison of the traditional and proposed FPGA mapping algorithms (see Section 6.1).

Example	K = 4		K = 6		K = 8		K = 10	
	old	new	old	new	old	new	old	new
b14	2.97	0.18	32.86	0.23	305.72	0.28	418.33	0.33
b15	3.55	0.25	38.25	0.32	282.29	0.39	480.82	0.47
C1355	0.44	0.02	8.31	0.02	21.63	0.02	24.62	0.03
C17	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
C1908	0.21	0.01	2.22	0.02	15.32	0.02	20.58	0.02
C2670	0.29	0.03	3.11	0.03	14.64	0.04	22.12	0.05
C3540	0.42	0.03	4.16	0.04	27.78	0.05	40.20	0.05
C432	0.09	0.01	0.78	0.01	6.18	0.01	11.66	0.01
C499	0.24	0.01	3.62	0.02	16.57	0.02	18.86	0.02
C5315	0.74	0.05	8.65	0.06	39.99	0.08	58.92	0.09
C6288	2.67	0.06	51.24	0.08	137.24	0.10	152.76	0.12
C7552	1.39	0.06	19.42	0.08	79.87	0.10	92.82	0.12
C880	0.11	0.01	0.70	0.01	4.18	0.02	10.71	0.02
clma	8.26	0.65	48.75	0.84	368.81	1.03	1268.90	1.21
pj1	6.70	0.50	66.83	0.64	462.49	0.79	732.56	0.93
pj2	1.06	0.12	7.23	0.15	57.73	0.19	127.33	0.22
pj3	4.42	0.31	40.51	0.39	337.10	0.48	539.56	0.57
s15850	1.34	0.13	10.21	0.17	57.79	0.20	105.28	0.24
s35932	6.80	0.42	75.68	0.54	323.03	0.66	388.16	0.78
s38417	3.33	0.33	27.44	0.43	160.68	0.53	276.61	0.62
Ratio	1.00	0.12	1.00	0.06	1.00	0.05	1.00	0.05

Table 4. Runtime comparison of the traditional and proposed FPGA mapping algorithms (see Section 6.1).

Example	K = 4		K = 6		K = 8		K = 10	
	old	new	old	new	old	new	old	new
b14	0.15	0.01	2.12	0.02	73.36	0.02	24.36	0.02
b15	0.18	0.02	2.96	0.02	40.60	0.03	35.18	0.02
C1355	0.36	0.01	2.05	0.01	1.69	0.01	1.62	0.01
C17	0.02	0.01	0.02	0.01	0.02	0.01	0.02	0.01
C1908	0.05	0.01	0.17	0.01	2.40	0.01	1.07	0.01
C2670	0.08	0.01	0.30	0.02	1.48	0.01	1.35	0.01
C3540	0.05	0.01	0.30	0.01	4.49	0.01	2.27	0.01
C432	0.03	0.01	0.06	0.01	0.77	0.02	0.89	0.01
C499	0.08	0.02	0.46	0.01	1.28	0.01	0.97	0.01
C5315	0.22	0.02	0.65	0.01	3.53	0.01	2.92	0.01
C6288	0.16	0.01	5.99	0.02	8.67	0.01	7.13	0.01
C7552	0.08	0.01	1.70	0.01	9.24	0.01	4.51	0.01
C880	0.45	0.01	0.47	0.01	0.81	0.01	1.37	0.01
clma	0.39	0.04	1.71	0.09	18.31	0.07	57.21	0.07
pj1	0.32	0.05	4.31	0.05	71.08	0.05	39.87	0.05
pj2	0.07	0.01	0.33	0.01	6.38	0.01	8.51	0.01
pj3	0.22	0.03	2.20	0.03	54.71	0.03	32.11	0.03
s15850	0.07	0.02	0.74	0.01	7.18	0.01	5.74	0.01
s35932	0.29	0.02	5.04	0.03	22.29	0.02	16.69	0.02
s38417	0.16	0.02	2.30	0.02	15.51	0.02	14.38	0.02
Ratio	1.00	0.16	1.00	0.05	1.00	0.03	1.00	0.03

Table 5. Performance of the proposed algorithm on multiple timeframes of *wb_conmax.v* (see Section 6.2).

Number of frames	AIG statistics		FPGA mapping statistics		Computer resources	
	Levels	Nodes	Depth	Number of LUTs	Memory, Mb	Runtime, sec
1	18	40381	4	11069	2.21	0.02
20	284	808135	61	205143	42.68	0.42
40	564	1616285	121	409149	85.28	0.84
60	844	2424435	181	613155	127.88	1.35
80	1124	3232585	241	817161	170.48	1.77
100	1404	4040735	301	1021167	213.09	2.25

Table 6. Performance of the proposed algorithm on 100 timeframes of *wb_conmax.v* for different LUT sizes (see Section 6.2).

LUT size	FPGA mapping statistics		Computer resources	
	Depth	Number of LUTs	Memory, Mb	Runtime, sec
4	602	2279062	114.74	1.89
6	451	1704400	147.52	2.00
8	352	1205319	180.30	2.19
10	301	1021167	213.09	2.24
12	276	1044370	245.87	2.50
14	227	799618	278.65	2.55
16	202	694954	311.43	2.62