WireMap: FPGA Technology Mapping for Improved Routability

Stephen Jang, Billy Chan, Kevin Chung
Xilinx Inc.
{sjang, billy, kevinc}@xilinx.com

Alan Mishchenko
University of California, Berkeley
alanmi@eecs.berkeley.edu

ABSTRACT

This paper presents a new technology mapper, WireMap. The mapper uses an *edge flow* heuristic to improve the routability of a mapped design. The heuristic is applied during the iterative mapping optimization to reduce the total number of pin-to-pin connections (or edges). The average edge reduction of 9.3% is achieved while maintaining depth and LUT count of state-of-the-art technology mapping. Placing and routing the resulting netlists leads to an 8.5% reduction in the total wire length, a 6.0% reduction in minimum channel width, and a 2.3% reduction in critical path delay. Applying WireMap has an additional advantage of reducing an average number of inputs of LUTs without increasing the total LUT count and depth. The percentages of 5- and 6-LUTs in a typical design are reduced, while the percentages of 2-, 3-, and 4-LUTs are increased. These smaller LUTs can be merged into pairs and implemented using the dual output LUT structure found in commercial FPGAs. WireMap leads to 9.4% fewer dual-output LUTs after merging.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids – Optimization;
B.7.1 [**Integrated Circuits**]: Types and Design Styles – Gate arrays

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

FPGA, Technology Mapping, Cut Enumeration, Area Flow, Edge Flow

1. INTRODUCTION

Technology mapping for Field-Programmable Gate Arrays (FPGAs) transforms a technology-independent logic network, called the *subject graph*, into a network of logic nodes with no more than K inputs. Each node is represented using a K-input *look-up table* (LUT) implementing any Boolean function up to K inputs. The subject graph is often a network composed of one- and two-input gates, for example, an *AND-Inverter Graph* (AIG).

A standard technique to map into K-input LUTs uses cut-enumeration [8][17][6][12]. In this approach, the subject graph is traversed in a topological order. Each time a node is visited and its cuts are computed. A trivial cut consists of the node itself. A non-trivial cut is a set of nodes in the transitive fanin of a node blocking all the paths from the primary inputs to the node. A K-feasible cut is a cut that can be implemented by a K-LUT. The LUT sizes used in the present-day FPGAs are 4, 5, and 6.

During technology mapping, each cut is ranked using different cost-functions, such as the number of levels (the delay from PI to this node) and area (the number of LUTs in the transitive fanins). Area estimation of a cut is difficult because of the inaccuracy in determining fanouts in the mapped network. Therefore, during area recovery several optimizations are performed to improve the selected mapping. Each pass uses one or more different cost-functions for cut ranking [8][12].

The main contributions of this paper are as follows. First, a new optimization during technology mapping, called *edge recovery*, is motivated and described. This mapping optimization reduces the edge count (the number of pin-to-pin connections) for the entire design while preserving depth and area. Second, during edge recovery a new heuristic called *edge flow* is proposed to globally optimize the number of edges. The use of the edge-flow is a breakthrough because is allows for a seamless integration of a key placement metric into the traditional mapping based on cut-enumeration. As a result, the proposed mapper, WireMap, is able to produce a netlist with greatly reduced edge count, without sacrificing area and delay measured as the number of LUTs and the depth of the LUT network. Intuitively, minimizing edges reduces the wirelength after routing, which, in turn, diminishes congestion and improved design performance.

The quality of FPGA mapping (both delay and area) is often substantially improved after performing several iterations of mapping with structural choices (MSC), as presented in [10][5]. In our experiments, MSC reduced LUTs, logic levels, and edges by 9.1%, 7.9%, and 8.1%, respectively, compared to the same mapping without structural choices.

WireMap, proposed in this paper, improves this highly optimized result even further. In our experiments, the number of optimizating iterations during technology mapping was kept unchanged. Even so, WireMap produced mapped netlists that are better than the original ones in terms of LUTs, logic levels, and edges by 10.2%, 10.1% and 16.7%, respectively. In other words, WireMap reduces the number of edges by an additional 9.3%, compared to the high-effort iterative MSC!

To confirm that the edge count reduction helps routability, the netlists derived using MSC and WireMap were placed and routed using VPR [4]. As a result, the critical path delay was reduced by 2.3% on average, total wire length by 8.5%, and minimum channel width by 6%.

The algorithm also modifies the LUT size distribution by reducing an average number of LUT inputs while maintaining LUT count and depth. The percentages of 5- and 6-LUTs in a design are both reduced, while the percentages of 2-, 3-, and 4-LUTs are increased. The higher frequency of small LUTs leads to 9.4% fewer dual-output LUTs after merging for a commercial FPGA.

The rest of the paper is organized as follows. Section 2 describes background. Section 3 reviews the traditional FPGA mapping with area recovery. Section 4 describes the edge flow heuristic and algorithms used by WireMap. Section 5 reports experimental results. Section 6 concludes the paper and outlines future work.

2. BACKGROUND

The presentation of the background material in Sections 2 and 3 is based on [12] and [15].

The combinational part of a design can be represented by a *Boolean network,* which is a directed acyclic graph (DAG) with vertices (or nodes) corresponding to logic gates and directed edges corresponding to the routing wires connecting the gates in the implemented circuit. The terms Boolean network, circuit, and netlist are used interchangeably in this paper.

Each node in the graph has zero or more *fanins,* with each fanin having a driver. A node also has zero or more *fanouts*, which are the loads driven by this node. The *primary inputs* (PIs) are nodes without fanins while the *primary outputs* (POs) are nodes without fanout. If the network is sequential, it contains registers whose inputs and output are treated as additional POs/PIs for the purposes of combinational logic optimization and mapping.

The cut-based structural mapping for K-input LUTs is applied to *subject graphs* that are K-bounded, that is, the number of fanins of any node is does not exceed K. Note that any given network can be decomposed to create a K-bounded graph suitable for mapping. In this work, ABC [3] is used to perform technology mapping starting from the AND-INV graph (AIG), which is a Boolean network composed of two-input ANDs and inverters. However, the cut-based structural mapping can be applied to any K-bounded subject graph.

An *edge* in the subject graph is a connection between two nodes that are in the fanin/fanout relationship with each other. The *edge count* of the subject graph is the total number of edges in it. The concepts of edge, wire, and pin-to-pin connection are used interchangeably in this paper.

Given a node *n*, a *cut C* is a set of nodes of the network, called *leaves*, such that every path from a PI to *n* passes through at least one leaf. The cut *covers* the root *n* and all the nodes found on the path from the leaves to the root, excluding the leaves. A cut is said to be *K-feasible* if the number of leaves in it does not exceed *K*. A cut is said to be *dominated* if there is another cut of the same 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 transitive 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*.

The *level* of a node is the number of nodes on the longest path from any PI to the node. The node itself is counted but the PIs are not. The network *depth* is the largest level of any internal node in the network.

A *mapping* selects one K-feasible cut, called the *representative cut*, for each internal node of the subject graph. The mapping procedure also computes and incrementally updates a subset of nodes whose representative cuts cover all non-PI nodes in the network. These nodes are said to be *used* in the mapping.

A starting mapping is found by assigning one “good” cut at each node in the network. Next, the mapping is iteratively updated by performing several optimization passes over the network. Each pass selectively modifies the representative cut of one node at a time and propagates changes to other nodes. Each such modification typically changes the set of used nodes. The changes may propagate recursively from the node towards the PIs.

The *area* of the mapping is the number of nodes used in the mapping. Heuristic iterative optimization of a mapping discussed in this paper is greedy in the sense that it modifies the representative cuts of one node at a time, in such a way that the area of the current mapping is reduced or remains the same.

3. CUT-BASED FPGA MAPPING

Figure 3.0 shows a self-explanatory pseudo-code of the traditional FPGA technology mapping [7], as implemented in [12].

```
traditionalMap( aig, K)
{
    // compute all K-feasible cuts at each node and save them
    traditionalMapCutEnumeration( aig, K );
    // assign a min-depth cut as the representative at all nodes
    traditionalMapDepthOriented( aig, K );
    // update the representative cut at each node to save area
    traditionalMapAreaRecovery( aig, K );
    // return the set of nodes used in the final mapping
    traditionalMapDeriveFinalMapping( aig, K );
}
```

Figure 3.0. The traditional FPGA mapping.

3.1 Cut Enumeration

This subsection reviews the cut enumeration method from [17][8]. Let *A* and *B* be two sets of cuts. For convenience operation $A \Diamond B$ is defined as follows:

$$A \Diamond B = \{u \cup v \mid u \in A, v \in B, |u \cup v| \le k\}$$

Let $\Phi(n)$ denote the set of K-feasible cuts of node n. If n is an AND node, let n_1 and n_2 denote its fanins. The set of cuts of node n is computed using sets of cuts of its fanins:

$$\Phi(n) = \begin{Bmatrix} \{\{n\}\} & : n \in \text{PI} \\ \{\{n\}\} \cup \Phi(n_1) \Diamond \Phi(n_2) & : \text{otherwise} \end{Bmatrix}.$$

This formula translates into a simple procedure computing all K-feasible cuts of all nodes in a single traversal from the PIs to the POs. Performing the cut computation in a topological order guarantees that the fanin cuts, $\Phi(n_1)$ and $\Phi(n_2)$, are available when the node cuts, $\Phi(n)$, are computed. In practice, the cut set of an AND node is computed by merging the two cut sets of its children, as shown above, and adding the trivial cut (the cut composed of the node itself) while keeping only K-feasible cuts. The computed cuts are filtered on-the-fly by removing dominated cuts. This reduces runtime and memory without compromising quality of mapping.

3.2 Depth-Oriented Mapping

Assuming that all cuts of all nodes are computed by the cut enumeration procedure described in Section 3.1, a depth-oriented mapping [7][15] can be derived by traversing the nodes in a topological order, and at each node finding the cut that minimizes the level of the mapping. This cut, along with its level, are stored at the node. The level of a cut is computed by adding 1 to the largest level of the cut leaves. The pseudo-code of depth-oriented mapping is shown in Figure 3.2

```
traditionalMapDepthOriented( aig )
{
    for each aig node n in topological order {
        cut = findCutMinimizingDepth( n );
        setLevel( n, getLevel(cut) );
        setRepresentativeCut( n, cut );
    }
}
getLevel( cut )
{
    level_max = -∞;
    for each node m in cut
        level_max = max( level_max, getLevel(m) );
    return 1 + level_max;
}
```

Figure 3.2. Traditional depth-oriented FPGA mapping.

3.3 Area Recovery

The above depth-oriented mapping computes a LUT mapping whose depth is minimum for the LUT size and the given logic structure of the subject graph [7]. The depth minimization on every path from the PIs to the POs leads to the phenomenon known as *area duplication*, when some AIG nodes are covered by more than one cut, leading to an increased LUT count. This disadvantage is addressed by area recovery phase performed after depth-oriented mapping as shown in Figure 3.3.

Previous work [12] has shown that applying two complementary heuristics, as shown in Figure 3.3, in the given order produces good practical results. The first heuristic (area flow) has a global view and selects logic cones with more shared logic. The second heuristic (exact local area) provides a missing local view by minimizing the area exactly at each node. The following subsections give an overview of these heuristics.

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

Figure 3.3. Area recovery in traditional FPGA mapping.

3.3.1 Global View Heuristic

Area flow [11] (*effective area* [8]) is a useful extension of the notion of area. It can be computed in one pass over the network from the PIs to the POs. Area flow of the PIs is set to 0. Area flow of a node n is computed as follows:

$$AF(n) = \frac{[Area(n) + \sum_i AF(Leaf_i(n))]}{NumFanout(n)} \qquad (1)$$

where $Area(n)$ is the area cost of the LUT used to map the current cut of the node n, $Leaf_i$(n) is the i-th leaf of the cut at n, and $NumFanouts(n)$ is the number of fanouts of the node n in the currently selected mapping. If a node is not used in the current mapping, for the purpose of area flow computation, its fanout count is assumed to be 1.

If nodes are processed from the PIs to the POs, computing area flow is fast. Area flow gives a global view of how useful for the current mapping in the logic in the cone. Area flow estimates sharing between cones without the need to re-traverse them.

3.3.2 Local View Heuristic

The exact local area of a node is the area added to the mapping by using the current node in the mapping. The *exact area of a cut* is defined as the sum of areas of the LUTs in the MFFC of the cut, i.e. the LUTs added to the mapping if the cut is set as the representative cut of the node.

The exact area of a cut is computed using a fast local DFS traversal of the subject graph starting from the root node of the cut. The *reference counter of a node* in the subject graph is equal to the number of times it is used in the current mapping, i.e. the number of times it appears as a leaf of the representative cut at some other node, or as a PO. The exact area computation is called for each cut. It adds the cut area to the local area being computed, dereferences the cut leaves, and recursively calls itself for the representative cuts of the leaves whose reference counters are zero. This procedure recurs as many times as there are LUTs in the MFFC of the cut, for which it is called. This number is typically small, which explains why computing the exact area is fast. Once the exact area is computed, a similar recursive referencing is performed to reset the reference counters to their initial values, before computing the exact area for other cuts.

3.4 Producing a Mapped Network

The procedure used in the traditional mapping to derive the final LUT network is shown in Figure 3.4 [17][15].

```
fastMapDeriveFinalNetwork( aig, K )
{
    // set the mapped nodes and the frontier to be the PO nodes
    M = POs;   F = POs;
    // explore each node in the frontier
    while ( $F \neq \emptyset$ ) {
        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 used in the final mapping
    // each of these nodes will be implemented using one K-LUT
    return M;
}
```

Figure 3.4. Producing the mapped LUT network.

The procedure assumes that one K-feasible representative cut is assigned at each node. Two sets of AIG nodes are supported: 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 representative 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 returns the set *M* of nodes used in the mapping.

4. EDGE RECOVERY

The proposed algorithm is similar to the traditional technology mapping presented in Section 3 in that it considers all nodes in a topological order and minimizes depth, followed by several passes of mapping optimization. In the end, the mapped LUT network is produced as in the traditional mapping (see above Section 3.4).

The major modifications to the algorithm in Section 3 are as follows. In addition to the traditional area recovery, an edge recovery is performed during mapping optimization to reduce the total number of pin-to-pin connections (or edges) after technology mapping. As the result, the placer can generate a solution with smaller wire length.

Edge recovery consists of two major parts. First, the edge flow heuristic is used to globally select logic cones for minimizing the edge count. Second, an exact edge algorithm is used to minimize the edge count for the MFFC of the given cut.

The following subsections give an overview of these heuristics.

4.1 Global View Heuristic

The *edge flow* is analogous to the area flow [10] used during area recovery, as described in Section 3.3.1. The main goal of area flow is to “predict” the global LUT counts, so the mapping optimization can pick the cuts with the lowest area during the matching phase. Similarly, the edge flow “predicts” the total number of pin-to-pin connections in the transitive fanin of a node. By minimizing this number, the number of wires during placement and routing is reduced, thereby improving routability.

The definition of the *edge flow* cost function is as follows:

$$EF(n) = \frac{[Edge(n) + \sum_i EF(Leaf_i(n))]}{NumFanout(n)}, \qquad (2)$$

where *Edge*(*n*) is the total number of fanin edges of the LUT used to map the current representative cut of node *n*, $Leaf_i(n)$ is the *i*-th leaf of the representative cut of node *n*, and *NumFanouts(n)* is the number of fanouts of node *n* in the currently selected mapping. The global area and edge recovery is shown in Figure 4.1.

The algorithm in Figure 4.1 has a similar structure as the one in Figure 3.3, except that it focuses on the global edge reduction only, instead of both global and exact local reduction. During global edge reduction, instead of only finding one cut that has the minimum area flow, it finds all the cuts with the same minimum area flow. Among the tie cuts, it selects the cut with the lowest edge flow as the representative cut.

```
globalAreaEdgeRecovery( aig, K )
{
    computeRequiredTimes( aig );
    for each aig node n in topological order {
        cuts = findCutsMinimizingAreaFlow( n );
        repres_cut = findCutMinimizingEdgeFlow( cuts );
        setLevel( n, getLevel( repres_cut ) );
        setRepresentativeCut( n, repres_cut );
    }
}
edgeFlow( cut )
{
    edgeflow = numLeaves( cut );
    for each leaf of cut  { // visit the leaf nodes of the cut
        fanouts = nodeFanouts( leaf );
        if ( fanouts == 0 )
            fanouts = 1;
        edgeflow += getEdgeFlow( nodeReprCut(leaf) ) / fanouts;
    }
    setEdgeFlow( cut, edgeflow );
    return edgeflow;
}
```

Figure 4.1. Global area/edge recovery algorithm.

The edge flow computation for a cut, shown in Figure 4.1, follows the definition of the edge flow (2). The fanout counter is adjusted to be one, if the leaf node has no fanouts, meaning that this leaf is not used in the current mapping. The edge flow computation is not recursive; it uses the saved result of the edge flow previously computed for the representative cuts of the leaves.

4.2 Local View Heuristic

The exact local area (edge count) of a node is the area (edge count) added to the mapping by selecting the current node as the one used in the mapping. The exact area (edge count) of a cut is defined as the sum of areas (edge counts) of the LUTs in the MFFC of the cut, i.e., the LUTs to be added to the mapping, if the cut is selected as the representative one.

```
localAreaEdgeRecovery( aig, K )
{
   for each aig node n in topological order {
      cuts = findCutsMinimizingExactArea( n );
      repres_cut = findCutMinimizingExactEdge(cuts );
   }
}
exactEdgeCount( cut )
{
   if ( cutIsRepresentative( cut ) ) {
       edges1 = exactEdgeCountDeref( cut );
       edges2 = exactEdgeCountRef( cut );
   } else {
       edges2 = exactEdgeCountRef( cut );
       edges1 = exactEdgeCountDeref( cut );
   }
   assert( edges1 == edges2 );
   return edges1;
}
exactEdgeCountRef( cut )
{
   edges = numLeaves( cut );
   for each leaf of cut  { // visit the leaf nodes of the cut
      nodeDecrementRefCounter( leaf );
      if ( nodeRefCounter( leaf ) == 0 && !nodeIsPi( leaf ) )
        edges += exactEdgeCountRef( nodeReprCut(leaf) );
   }
   return edges;
}
exactEdgeCountDeref( cut )
{
   edges = numLeaves( cut );
   for each leaf of cut  { // visit the leaf nodes of the cut
      if ( nodeRefCounter( leaf ) == 0 && !nodeIsPi( leaf ) )
        edges += exactEdgeCountDeref( nodeReprCut(leaf) );
      nodeIncrementRefCounter( leaf );
   }
   return edges;
}
```

Figure 4.2. Exact local edge reduction.

The computation of combined exact area and edge recovery is shown in Figure 4.2. Nodes are considered in the topological order and for each of them the cuts minimizing exact area is computed. If there is a tie, the cut minimizing the exact edge count is selected.

The recursive procedure to compute exact edge count of a cut is also shown. The computation is this procedure varies depending on whether a cut is the representative cut of a node that is used in the mapping. In this case, the node is referenced and so are the leaves of its representative cut. This is why the computation of the exact edge count works on the leaves by first referencing them. If the reference counter of a leaf becomes 0, the recursive computation is called for the leaf; otherwise the leaf is skipped. This is because the edges of the representative cut of the leaf are added towards the total leaf count if the leaf participates in the current mapping only through the current cut.

After calling the procedure for the exact edge count or exact area count, which dereference (reference) the node, the corresponding referencing (dereferencing) procedure should be called, which restores the original reference counters of the nodes.

Although not shown in the pseudo-code, it should be noted that when the representative cut of the node is updated, the old cut is dereferenced first, followed by referencing the new cut. This ensured that, at any time, only the nodes used in the mapping are referenced and the sum total of exact local areas (edge counts) of these nodes is the exact local area (edge count) of the mapping.

4.3 WireMap

The concept of edge flow can be added to the area recovery phase of the cut-based structural mapping, as shown in Figure 4.3.

In this algorithm, edge count is used as an additional cost function when choosing a cut. In WireMap, area (edge) flow is used as the first (second) tie-breaker in the first pass when a delay-optimum mapping is computed. When the global view heuristic is applied (Section 4.1), area flow becomes the primary cost function and edge flow becomes the tie-breaker used to choose among the cuts, whose arrival times do not exceed the required times. When the local view heuristic is applied (Section 4.2), exact area becomes the primary cost function and the exact edge count becomes the tie-breaker used to choose among the cuts.

```
wireMap( aig, K )
{
   // compute all K-feasible cuts at each node and save them
   traditionalMapCutEnumeration( aig, K );
   // assign a min-depth cut as the representative at each node
   traditionalMapDepthOriented( aig, K );
   // update the representative cut at each node to save area
   globalAreaEdgeRecovery( aig, K );
   localAreaEdgeRecovery( aig, K );
   // return the set of nodes used in the final mapping
   traditionalMapDeriveFinalMapping( aig, K );
   performLUTMerging( aig, K );
}
```

Figure 4.3. Area recovery using edge flow.

Not shown in the pseudo-codes of this section is the use of timing constraints. In both global and local recovery phases, the arrival time of each cut is recomputed based on arrival times of the leaves of the cut that have possibly changed earlier in the current phase. If the arrival time of the cut exceeds the required time of the node, the cut is not considered for area/edge recovery. Note that because the required times are recomputed between the mapping passes using the representative cuts of the nodes, the representative cuts are always guaranteed to have a feasible arrival time. This is why the above procedures never run into a situation when the arrival time of all cuts at a node exceeds the required time.

5. DUAL-OUTPUT LUT MERGING

In several commercial FPGAs, the larger LUT structure (6-input or 7-input) can be "fractured" into smaller LUTs. This ability to "tap" a second output from a larger LUT complex is essential for a good balance of area/speed when implementing large designs. For the timing-critical part, large LUTs can be used to reduce the depth of mapping. For the non-timing-critical part, smaller LUTs can be used and these small LUTs can be combined into a larger LUT complex to reduce the logic cell count.

A typical example is Xilinx's Virtex-5 FPGA (Figure 5.1) [19]. In this architecture the 6-input LUT has two usable output pins. It can implement two independent LUTs if the total number of unique pins in them does not exceed five. Similarly, in Altera's Stratix II FPGA [1], an ALM can implement two independent LUTs if the total number of input pins does not exceed 8 and meets specific constraints on input sharing and LUT sizes. Combining of smaller–input LUTs into larger structures is called LUT-merging.

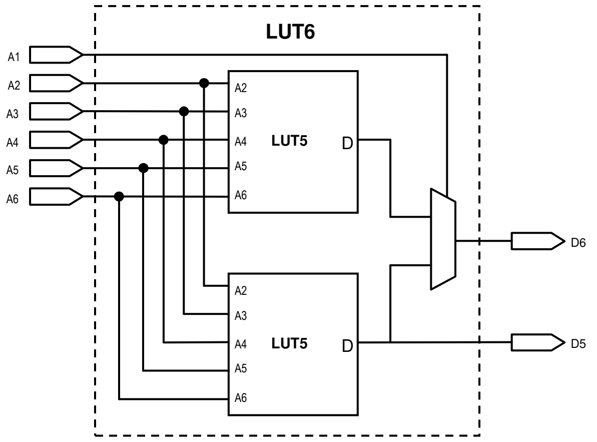

Figure 5.1.Virtex-5 dual-output LUT.

The LUT-merging minimization problem for Virtex-5 can be formulated as finding the maximum number of disjoint pairs of mergeable LUT functions, such that each pair of merged functions has no more than 5 inputs. It can be demonstrated that the solution of this problem is equivalent to that of finding the maximum cardinality matching in a graph in which each node is a LUT and in which an edge is added between each pair of LUTs that can be merged [16].

WireMap produces smaller-size LUTs in the non-timing-critical region, which is repeatedly updated during mapping optimization, because the cost function has the number of fanin edges for the cut in the numerator. The edge recovery described in Section 4 produces a LUT distribution with higher frequency of smaller input LUTs (with 2, 3, or 4 inputs), compared to the mapping optimization without edge recovery.

It is interesting to compare this result with the LUT distribution obtained in [2]. WireMap is able to reduce both the number of 5- and 6-LUTs, while the number of 2-, 3-, and 4-LUTs is increased. Hence the result of WireMap is much more "mergeable" for FPGA architectures like Virtex-5. In Section 6, we discuss the effect of the improved LUT distribution on LUT-merging.

6. EXPERIMENTAL RESULTS

The proposed algorithm was implemented in ABC [3] as a new command *wiremap*. Technology mapping was applied to 20 large circuits from MCNC, ISCAS'89, and ITC'99 benchmark suites targeting a 6–LUT architecture. The experiments ran on an Intel Xeon 2-CPU 4-core computer with 8 Gb RAM. All mapped networks were verified using the combinational equivalence checker in ABC (command "cec").

The following ABC commands are used in the experiments:

- *fpga* [12] performs FPGA mapping into K-LUTs using exhaustive cut enumeration. This algorithm is based on the traditional area flow and exact area recovery. It can be applied to both a Boolean network and a set of structural choices derived for the Boolean network.
- *choice* [5] is a logic synthesis script to derive structural choices. It performs 15 passes of AIG rewriting [13] and collects three AIG snapshots of the network: the original one, the final one, and the intermediate one saved after the first 5 rewriting passes.
- *wiremap* is the new FPGA mapping algorithm based on optimization with edge recovery, as described in this paper. This algorithm also can be applied to a network or a set of structural choices derived for the network.

The experimental data is listed in Tables 6.1 and 6.2. The row "Ratios" in the tables are the geometric averages of the corresponding ratios in the columns. The results of three experiments are described in the following subsections.

6.1 Technology Mapping Only

To evaluate the contribution of WireMap to the quality of technology mapping, three mapping runs were performed for each example, as shown in the three vertical sections of Table 6.1. These include the baseline mapping, MSC, and WireMap.

Columns "luts", "lev", and "edg" show the number of LUTs, LUT levels, and edge count (the total number of pin-to-pin connections), respectively. Column "t" shows runtime in seconds. The last column in each section ("clb") will be discussed later.

The baseline mapping is computed by the traditional FPGA mapping in ABC (command "*fpga*") using default settings.

MSC includes four iterations of the script "*choice; fpga*", which repeatedly runs the traditional FPGA mapping (command "*fpga*") on a set of structural choices derived by command "*choice*". Note that the result is already significantly better than a single pass of traditional FPGA mapping due to the reduced structural bias.

The third section of Table 6.1 shows the results for WireMap. Similar to the second section, four iterations of the script "*choice; wiremap*" were applied. Command "*wiremap*" uses the edge flow heuristics in addition to the traditional area recovery with area flow and exact area.

The experimental results lead to the following observations:

- MSC achieved a 9.1% reduction in LUTs, 8.1% reduction in edges, and 7.1% reduction in logic levels, compared to the one-pass traditional mapping.
- The number of wires (or pin-to-pin connections) has been reduced by 9.3% on average using WireMap versus MSC.
- The design depths are the same in almost all cases. In one case, one level is added and in two cases one level is reduced. On average, the depth is slightly better, by 2.4%.
- LUT count after WireMap decreases by 1.3% on average.
- Runtime of WireMap is reduced by 8.9% on average compared to MSC. This is likely because it takes less time to map LUTs with fewer inputs.

6.2 Technology Mapping with VPR

Next, the mapped designs were processed using VPR [4]. Table 6.2 presents the data of this experiment. Columns "twl" and "mcw" denote the total wire length and minimum channel width determined by VPR respectively. Column "cpd" shows critical path delay when VPR is run with the smallest possible channel width that was routable for all three mapped netlists.

When clustering is performed (using T-VPACK), a single LUT-FF pair in the Logic Element (LE) is targeted. In the place/route runs, the smallest possible CLB array is chosen to avoid spreading. Channel width and segment distribution are preset such that they are the same for every run. This makes the comparison of wire length meaningful. The VPR cost function was set to reduce bounding-box cost exclusively.

The following observations can be made from Table 6.2:

- When WireMap is used, the total wire-length after place-and-route is reduced by 8.5%, compared to MSC.
- The minimum channel width to route the designs is reduced by 6.0%, compared to MSC.

The reduction in wire-length leads to better design performance and routability. We confirm this by running another set of VPR experiments with the channel width selected to the highest of the three channel widths determined by the previous experiment. The results are shown in the "cpd" column of Table 6.2, indicating that the critical path delay is reduced by 2.3% on average. While total wire length and minimum channel width relate directly to the number of point-to-point edges in the design, the measurement of the critical path delay does not. Therefore, it is surprising the critical path delay is improved in many benchmark designs.

We speculate that the reason for the critical path delay improvement is the reduced edge count leading to improved placement quality for the critical path components. In some designs, higher fanout of nets in the critical path tend to increase critical path delay. Because the algorithm and cost function is not fanout sensitive, such a side-effect is not surprising.

6.3 LUT Distribution and LUT merging

The final experimental study focuses on LUT-size distribution and dual-LUT merging. Command "*xl_merge*" in SIS [18] is used to perform the merging operation. It selects pairs of nodes of the network to be merged while minimizing the total cluster count.

The result after LUT merging is shown in the "clb" column in Table 6.1. After LUT merging, the dual-output LUT count in Virtex-5 is reduced by 9.4% with WireMap, compared to MSC.

Figure 6.1 compares the LUT size distribution produced by MSC versus WireMap. For MSC, 47% of all LUTs are LUT6, while for WireMap, the number of 6-LUT is reduced by 9%. There is also a reduction in 5-LUTs by 3%. In contrast, the ratios of 2-, 3-, and 4-LUTs are increased by 5%, 5%, and 2%, respectively.

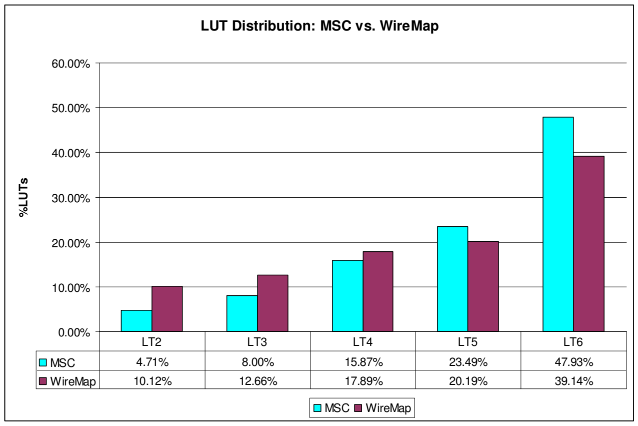

Figure 6.1. LUT distribution comparison

7. CONCLUSIONS

A new practical heuristic, *edge flow*, is proposed and incorporated into FPGA technology mapping. The resulting mapper, WireMap, differs from the known mappers in that it minimizes the total number of edges in the mapped network. As a result, when targeting 6-LUTs, a reduction of 9.3% in the average number of wires (or pin-to-pin connections) in the design is observed, with minimal changes in depth and LUT count.

The reduced pin-to-pin connections leads to an average 8.5% total wire-length reduction after place-and-route. Moreover, the minimum channel width requirement is also reduced by 6% over the entire benchmark set. When a fixed channel width is targeted, the critical path delay of the routed circuit is reduced by 2.3%.

In terms of LUT distribution, edge flow also produces a higher percentage of smaller LUTs without increasing LUT count or levels. Smaller LUTs enhance the merging capability for a commercial dual-output LUT. In our experiments, this advantage translated into 9.4% CLB savings.

Intuitively, the shorter wire-lengths produced by edge flow should lead to shorter routes and lower power consumption. Also, smaller LUTs created by WireMap can also be exploited to reduce power after placement/routing by appropriate programming of the memory values [9]. We plan to evaluate the potential power reduction due to WireMap as part of future work.

Since using edge flow leads to smaller-size LUTs while keeping the total number of LUTs almost unchanged, fewer LUTs lead to fewer AIG nodes after the mapped network is converted back into an AIG during the iterative computation of structural choices. This gives an example of a positive role the edge recovery has on the results of synthesis. It is likely that edge flow can be further tuned and used in other synthesis algorithms to improve the routability of designs after mapping.

Future work will also include

- fine-tuning to achieve better trade-offs between logic level, LUT count, edge count, and LUT size distribution,
- using edge flow in priority-cut-based mapping [15],
- integrating edge flow into post-mapping resynthesis [14].

8. REFERENCES

[1] Altera. Stratix III Device Handbook, http://www.altera.com/literature/hb/stx3/stratix3_handbook.pdf

[2] Altera. "Improving FPGA performance and area using an adaptive logic module", http://www.altera.com/literature/cp/cp-01004.pdf

[3] Berkeley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and Verification*, Release 61225. http://www.eecs.berkeley.edu/~alanmi/abc/

[4] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for deep-submicron FPGAs*, Kluwer Academic Publishers, 1999, ISBN 0-7923-8460-1.

[5] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping", *Proc. ICCAD '05*, pp. 519-526

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

[7] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs", *IEEE TCAD,*, Vol. 13(1), Jan. 1994, pp. 1-12

[8] 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.

[9] S. Gupta, A. Anderson, L. Farragher, and Q. Wang. "CAD techniques for power optimization in Virtex-5 FPGAs", to appear in *Proc. Custom Integrated Circuits Conference*, 2007.

[10] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE Trans. CAD*, vol. 16(8), 1997, pp. 813-833.

[11] V. Manohara-rajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in LUT-based FPGA technology mapping," *Proc. IWLS '04,* pp. 14-21.

[12] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for LUT-based FPGAs", *Proc. FPGA '06*, pp. 41-49.

[13] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis", *Proc. DAC'06*, pp. 532-536.

[14] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang, "SAT-based logic optimization and resynthesis", *Proc. IWLS '07*, pp. 358-364.

[15] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts", *Proc. ICCAD '07*.

[16] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic synthesis for programmable gate arrays", *Proc. DAC'90,* pp. 620-625.

[17] P. Pan and C.-C. Lin, "A new retiming-based technology mapping algorithm for LUT-based FPGAs," *Proc. FPGA '98*, pp. 35-42.

[18] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. "SIS: A System for Sequential Circuit Synthesis". *Memorandum No. UCB/ERL M92/41,* Department of EECS. UC Berkeley, May 1992.

[19] Xilinx White Paper. "Achieving higher system performance with the Virtex-5 family of FPGAs", http://direct.xilinx.com/bvdocs/whitepapers/wp245.pdf

Table 6.1. Comparison of traditional area flow and edge flow mapping results (K = 6).

	Baseline					Mapping with Structural Choices					WireMap				
	luts	lev	edg	t,s	clb	luts	lev	Edg	t,s	clb	luts	lev	edg	t,s	clb
alu4	807	6	3927	0.6	652	742	5	3520	6.79	585	742	5	3298	7.25	550
Apex2	983	6	4664	0.75	778	807	6	3850	10.73	654	805	6	3574	11.11	603
b14	1214	13	5620	1.94	976	1162	13	5578	61.36	935	1163	13	5014	53.7	823
b15	2169	15	11073	2.25	1856	2103	15	10485	61.35	1804	2056	14	9499	51.21	1626
b17	6507	21	33151	6.73	5625	6480	18	32906	191.55	5602	6419	18	29552	169.62	5090
b20	2490	15	11953	3.95	2024	2380	14	11768	138.27	1980	2312	14	10582	118.02	1760
b21	2569	15	12418	4.15	2098	2391	14	11807	135.8	1995	2399	14	10781	116.26	1815
b22	3742	15	18027	5.89	3074	3613	14	17910	187.25	3053	3618	14	16426	180.92	2787
Clma	3310	10	15576	2.72	2585	2392	9	11520	44.55	1952	2478	9	10846	42.61	1833
Des	681	5	3541	0.94	624	502	4	2643	14.39	473	498	3	2192	15.48	370
ex5p	624	5	3019	0.6	497	562	4	2716	10.46	450	578	4	2666	10.49	437
Elliptic	1800	10	8777	1	1662	1859	10	9173	17.95	1682	1807	9	8362	18.86	1569
Frisc	1750	14	8610	1.35	1621	1798	12	8753	28.02	1668	1690	12	7662	24.37	1524
i10	629	9	2863	0.59	470	603	7	2765	9.81	465	574	8	2375	8.75	404
Pdc	2305	7	11307	2.4	1923	2012	6	10061	77.7	1695	1891	6	8795	73.69	1476
s38584	2740	6	11574	1.63	1996	2667	6	11219	17.32	1948	2648	6	10580	17.54	1849
s5378	392	4	1553	0.3	253	357	4	1469	2.66	248	359	4	1346	2.69	226
Seq	933	5	4521	0.67	750	737	5	3577	11.69	599	732	5	3276	11.21	551
Spla	1862	6	9062	2.06	1538	1588	6	8065	52.58	1350	1515	6	7013	49.14	1198
Tseng	657	7	2546	0.46	455	645	7	2488	5.18	452	645	6	2343	5.41	423
Geomean	1480	8.65	6988	2.05	1194	1346	7.97	6420	54.27	1102	1329	7.78	5824	49.42	998
Ratio	**1**	**1**	**1**	**1**	**1**	**0.909**	**0.921**	**0.919**	**26.473**	**0.923**	**0.898**	**0.899**	**0.833**	**24.107**	**0.836**
Ratio						**1**	**1**	**1**	**1**	**1**	**0.987**	**0.976**	**0.907**	**0.911**	**0.906**

Table 6.2. Wirelength, channel width and critical path delay comparison.

	Baseline			MSC			WireMap		
	twl	mcw	cpd*	twl	mcw	cpd	twl	mcw	cpd
alu4	15896	15	83.87	13594	13	78.94	14014	15	78.26
Apex2	20995	16	88.45	17004	14	90.27	16197	14	90.97
b14	18331	11	148.02	18768	13	129.97	16265	10	149.57
b15	38895	15	180.51	36037	14	203.55	33401	14	195.91
b17	117551	16	249.05	120451	15	222.67	113153	15	225.29
b20	38672	11	152.00	39000	12	155.19	33885	12	139.94
b21	38684	11	143.18	39093	12	170.93	33791	11	135.72
b22	61069	13	157.05	63852	14	157.88	56914	13	150.88
Clma	70021	18	167.45	48469	15	131.35	47018	15	136.31
Des	19571	8	91.91	16944	10	101.23	13222	7	129.25
elliptic	28546	13	150.32	29670	14	181.29	24611	12	133.04
ex5p	12314	14	87.90	10346	13	73.26	11039	13	75.15
Frisc	33763	15	159.11	35412	14	154.96	30398	14	134.79
i10	16383	9	211.59	17186	8	155.60	15103	8	162.49
Pdc	64130	21	162.38	52978	21	148.93	47431	19	154.44
s38584	28083	11	72.55	26760	10	68.97	24723	9	71.15
s5378	4685	8	40.31	4358	10	49.26	4261	9	40.91
Seq	20151	16	85.97	15640	16	86.58	15005	15	87.30
Spla	44885	18	151.14	37925	19	130.99	34542	18	137.23
Tseng	5718	7	49.91	5610	7	48.91	5365	7	47.47
Geomean	26524	12.76	119.55	24440	12.79	116.45	22364	12.03	113.81
Ratio	**1.00**	**1.00**	**1.00**	**0.921**	**1.002**	**0.974**	**0.834**	**0.943**	**0.952**
Ratio				**1.00**	**1.00**	**1.00**	**0.915**	**0.94**	**0.977**

twl = total wire length, mcw = minimum channel width required to route in VPR
*cpd = critical path delay using the smallest possible channel width across the three implementations