Optimal Technology Mapping for Single Output Cells*  

Uwe Hinsberger and Reiner Kolla  
Lehrstuhl für technische Informatik  
Universität Würzburg  
97070 Würzburg, Germany  

Abstract  
This paper presents a new approach to technology mapping for arbitrary technologies with single output cells. It overcomes the restrictions of tree-mapping based methods. Optimal algorithms for special cases of DAG-mapping are presented: for minimum delay mapping and for duplication-free mapping under a class of simple cost functions (including area and delay). Heuristics for duplication of logic and for AT-tradeoffs are developed and applied to LUT-FPGAs.  

1 Introduction  
Compiling a set of boolean functions into an optimum hardware implementation is a very ambitious task. That is why it is usually divided into a technology independent and a technology mapping part. The former minimizes abstract cost measures as depth or literals of a factored representation [1]. The latter gets a network with nodes representing primitive functions and compiles it into a circuit using more complex cells of a given target technology. In this paper we consider the second problem.  

Keutzer [8] tackles the problem by means of methods originating from code generation. He decomposes the network into its fanout-free regions and performs tree-mapping for each of them. This idea is extended by [13] to compute AT-tradeoffs over a more exact delay model. Both approaches are restricted to cells that can be modeled by a tree of primitive gates. Thus cells as multiplexers and exclusive-or’s cannot be considered. Detjens et.al. [3] proposed DAG-covering heuristics as a solution for this problem. They also were the first to consider the potential profit resulting from duplication of logic at fanout-points. Efficient algorithms for FPGA-mapping were recently presented by [2].  

Our approach generalizes from our previous results on sizing [5, 6] and tree-mapping [7]. Some observations are similar to those presented in [2]. But our algorithms are simpler even if correctness proofs are more involved. Moreover our methods are applicable for arbitrary technologies with single output cells.  

We start with a formal statement of the mapping problem and with some annotations concerning problem complexity. We assume that the result of technology independent optimization is represented as a graph whose nodes are primitive boolean functions (e.g. inverters and nands).  

Definition 1 A Boolean Network is a directed acyclic graph $N = (V, E)$. The nodes  
$$V_{pi} := \{v \in V \mid \text{indegree}(v) = 0\}$$  
are considered as primary inputs; the other nodes represent primitive gates. $V_{po}$ is the set of gates feeding the primary outputs; we assume that  
$$V_{sink} := \{v \in V \mid \text{outdegree}(v) = 0\} \subseteq V_{po}$$  
The edges $E$ represent the connections.  

Figure 1 shows a boolean network for a full adder. All nodes (except $V_{pi}$) represent nands. Concrete target technologies enable the use of common cells computing more complex boolean functions. Such a cell can be substituted for a whole subgraph $C$ of $N$; subgraphs of that kind are represented as dashed boxes in our figures. We assume that each cell has a single output, i.e. exactly one node that has no successor within $C$. Nodes that have no predecessor within $C$ are interpreted as inputs of the cell. The other nodes of $C$ will be called “hidden within” $C$, because after substitution they are no longer visible. The boolean value at the cell output should depend only upon the values at cell inputs. Thus for each node $v$ of $C$ except the cell inputs every edge with target $v$ should belong to $C$.  

Definition 2 A Cell is a connected subgraph $C = (V_C, E_C)$ of $N$ with the following properties  
1. $C$ has exactly one root $O(C)$, the output of $C$.  
2. There is a set $I(C) \subset V_C$, the inputs of $C$ with  
   (a) $\forall u \in I(C) : \exists (w, v) \in E_C$  
   (b) $\forall u \in V_C - I(C) : (w, v) \in E \implies (w, v) \in E_C$  
3. $C$ can be realized by a single gate of the target.
technology.

\[ H(C) := V_C \cap I(C) \setminus \{O(C)\} \]

are nodes "hidden" within \( C \).

For most nodes \( V \) there is not only one possible realization but a large number of alternative cell candidates \( C \). Let \( C(v) := \{ C \mid O(C) = v \} \) be the set of these candidates and \( C = \bigcup_{v \in V} C(v) \). We do not consider in this paper how to compute \( C(v) \). We assume for each target technology a procedure to be available which fulfills this task. Such a procedure may implement graph pattern matching [3], boolean matching [9, 10] or maxflow algorithms [11, 12]. It depends upon technology what is the appropriate method. Actually this procedure is the only thing which has to be replaced if the target technology changes.

What we will consider within this work is the problem of how to find an optimal subset \( R \subseteq C \) which computes all primary output functions.

**Definition 3** A realization of \( N \) is a subset \( R \subseteq C \) such that

1. For each primary output \( v \in V_{po} \) there is exactly one cell \( C \in R \) with \( O(C) = v \).
2. For each cell \( C \in R \) and each input \( w \in I(C) \setminus V_{pi} \) there is exactly one \( C' \in R \) with \( O(C') = w \).
3. No proper subset \( R' \subset R \) fulfills 1. and 2.

Let \( R \) be the set of all possible realizations.

\( R \) represents the search space for us to minimize various cost functions. Often we will also be interested not in arbitrary realizations but just in realizations with specific properties; e.g. realizations satisfying a given delay constraint or realizations with a given node representing the output of a cell.

**Definition 4** A node \( v \in V \) is external under \( R \in R \) if \( v \in V_{pi} \) or if there is a cell \( C \in R \) with \( O(C) = v \).

Let \( R(v) := \{ R \in R \mid v \text{ external under } R \} \)

and \( R^{\leq T} := \{ R \in R \mid \text{ delay}(R) \leq T \} \).

**Example:** Figure 1 shows a realization \( R = \{ C_1, C_2, C_3, C_4 \} \) with external nodes 1, 2, 7, 8, 9, 12 and 13. The nodes 3, 4, 5, 6, 10 and 11 are hidden within cells. Thus \( R \in R(1) \) and \( R \in R(9) \) but \( R \notin R(3) \). If for the sake of this example we use depth as a measure for delay \( R \in R^{\leq 3} \) and \( R \in R^{\leq 10} \) but \( R \notin R^{\leq 2} \). This example illustrates the special case of a duplication-free realization: each edge \( e \in E \) is contained within exactly one cell of \( R \). Definition 3 is more general; it allows realizations as \( R' = \{ C'_1, C'_2, C'_3 \} \) in figure 2a, where the edges \( (7, 9) \) and \( (8, 9) \) are contained within two different cells. \( R' \) is superior to \( R \) with regard to both cost functions, area and delay. Figure 2b gives a different interpretation of this realization; \( R' \) can be considered as the result of two consecutive passes: first some parts of the network (the edges \( (7, 9) \) and \( (8, 9) \) and the node 9) are duplicated, then duplication-free mapping is performed on the resulting network.

We use this two-pass approach to the general problem for reasons of complexity: We have shown that area minimization under duplication is NP-complete (even if all cells are trees). In section 3 we will show however that optimal duplication-free mapping can be performed in time \( O(|C|) \) for a wide range of cost functions (including area and delay). Unfortunately computing the optimal AT-tradeoff remains NP-complete. But there is a very good heuristic for the duplication-free case; in many cases the results are even optimal. We will consider this topic in section 4. Based on the methods for duplication-free mapping we will develop heuristics for duplication in section 5. We demonstrate the viability of our approach by means of experimental results presented in section 6.

2 Duplication-free mapping

First of all we give a characterization of the cells which are relevant for duplication-free mapping. We call them primary cells because duplication-free mapping represents the kernel of our approach. High quality secondary cells should be recognized by the heuristical duplication pass and transformed to primary cells before solving the primary problem.

**Definition 5** A cell \( C = (V_C, E_C) \) is a secondary cell if there is a node \( v \in H(C) \) such that

1. \( v \in V_{po} \) or
2. \( \exists (v, w) \in E \text{ but } (v, w) \notin E_C \)

Otherwise \( C \) is called a primary cell.

For an illustration see the cell \( C_3 \) in figure 2a; \( C'_3 \) is secondary because the node 9 \( \notin H(C'_3) \) fans out to the node 12 outside \( C'_3 \). Thus any realization which uses \( C'_3 \) has to duplicate at least the edges \( (7, 9) \), \( (8, 9) \) and the node 9. Notice however that neither the edge \( (1, 3) \) nor \( (7, 9) \) prevent \( C'_1 \) from being primary because 1 and 7 are pins of \( C'_1 \).

Let \( C_{po}(v) \) be the set of primary cells with output \( v \) and \( C_{po} = \bigcup_{v \in V} C_{po}(v) \). The notion of primary cells results in a more simple characterization of duplication-free realizations:
Lemma 1 A subset \( R \subseteq \mathcal{C}_{pr} \) represents a realization of \( N = (V, E) \), if and only if for each \( e \in E \) there is exactly one cell \( C = (V_C, E_C) \in R \) with \( e \in E_C \).

Let \( \mathcal{R}_{pr} \) be the set of these realizations; \( \mathcal{R}_{pr}(v) \) and \( \mathcal{R}_{pr}^{\leq T} \) are defined analogously to \( \mathcal{R}(v) \) and \( \mathcal{R}^{\leq T} \).

Tree-mapping can be done efficiently because a bad realization of a subtree \( T \) can be substituted by an optimal realization of \( T \) without affecting the other parts of the network. We will show that duplication-free mapping allows similar substitutions.

Lemma 2 Let \( N = (V, E) \) be a network, \( r \) any node in \( V \) and \( N_r = (V_r, E_r) \) the subnetwork rooted at \( r \).

1. Let \( C = (V_C, E_C) \in \mathcal{C}_{pr} \) be any primary cell with \( r \notin H(C) \); then either \( E_C \subseteq E_r \) or \( E_C \cap E_r = \emptyset \).

2. For any \( R_1, R_2 \in \mathcal{R}_{pr}(r) \) let

\[
R_1' := \{ C \in R_1 \mid E_C \cap E_r = \emptyset \} \\
R_2' := \{ C \in R_2 \mid E_C \cap E_r \neq \emptyset \}
\]

Then \( R := R_1' \cup R_2' \in \mathcal{R}_{pr}(r) \).

Proof:

1. If \( O(C) \in V_e \), then \( E_C \subseteq E_r \) by definition. We will show that \( O(C) \notin V_e \) implies \( E_C \cap E_r = \emptyset \). Assume that \( O(C) \notin V_e \) and \( \exists e \in E_C \cap E_r \). There is a path \( (e_1, \ldots, e_n) \) with \( \text{target}(e_n) = r \) and \( e_j \in E_r \) for all \( j \). Let \( i \) be maximal with \( e_i \in E_C \cap E_r \). Then \( \text{target}(e_i) \in H(C) \). If \( i < n \), we have a contradiction to the assumption \( r \notin H(C) \). If \( i = n \), \( C \) is secondary because \( e_{i+1} \in E \) but \( e_{i+1} \notin E_C \), a contradiction to \( C \in \mathcal{C}_{pr} \).

2. Because of lemma 1 we have to prove that for each \( e \in E \) there is exactly one \( C \in R \) with \( e \in E_C \). It is easy to see that for each \( e \) there is at least one such cell. It remains to show, that for two different cells \( C, C' \in R \) the condition \( E_C \cap E_{C'} = \emptyset \) holds. If both cells are within \( R_1' \subseteq R_1 \) (or if both are within \( R_2' \subseteq R_2 \)) this is true because \( R_1 \subseteq \mathcal{R}_{pr} \) (and \( R_2 \subseteq \mathcal{R}_{pr} \)). The only interesting case is \( C \in R_1' \) and \( C' \in R_2' \) (vice versa); then \( E_C \cap E_{C'} \neq \emptyset \) and by part (1) of this lemma \( E_C \subseteq E_{C'} \). Together with \( E_C \cap E_r = \emptyset \) this implies \( E_C \cap E_r = \emptyset \).

We will use this result to derive a generic algorithm for efficiently computing a cheapest duplication-free realization under a variety of "simple" cost functions.

3 Simple cost functions

Throughout this section we will assume that the cost of a realization \( R \in \mathcal{R}_{pr} \) can be determined by a topological traversal through the network.

Definition 6 Let \((Val, \leq)\) be any totally ordered set. A cost function \( \Lambda : \mathcal{R}_{pr} \rightarrow Val \) is simple iff there are easy to compute functions

\[
f_{po} : V_{po} \rightarrow Val \quad f_{po}, f_C : Val \times \ldots \times Val \rightarrow Val
\]

such that

- \( \Lambda(R) = f_{po}(\lambda_R(o_1), \ldots, \lambda_R(o_n)) \)
- \( \{o_1, \ldots, o_n\} = V_{po} \) and the \( \lambda_R \)'s can be computed as follows:

- For \( v \in V_{po} \):
- For each cell \( C \in R \) with \( I(C) = \{w_1, \ldots, w_m\} \) and \( O(C) = v \):
- \( \lambda_R(v) := f_{po}(v) \)
- \( \lambda_R(w_1), \ldots, \lambda_R(w_m) \)
- \( f_{po} \) and \( f_C \) are monotone,
  - i.e. for any \( x, x' \in Val \) such that \( x \leq x' \):
    - \( f_{po}(x) \leq f_{po}(x') \) and \( f_C(x) \leq f_C(x') \)
  - \( \lambda_R \) is well-defined, because the outputs of the cells in \( R \) are all different and not in \( V_{po} \).

Before demonstrating how to minimize \( \Lambda \) we want to point out that at most usual cost functions, area and delay, are simple. As to delay this is easy to see:

\[
T(R) = \max_{v \in V_{po}} \tau_R(v)
\]

\[
\tau_R(v) = \begin{cases} 
0 & \text{if } v \in V_{po} \\
\max_{w \in I(C)} [\tau_R(w) + \text{delay}(C)] & \text{if } v = O(C), C \in R
\end{cases}
\]

Computing the area of \( R \) according to definition 6 requires some attention. In tree-networks we can compute the area of the subtree rooted at some external node \( r \in V - V_{po} \) by adding the area of the root cell \( C \) to the area of the subtrees rooted at the inputs of \( C \). For general acyclic networks this is not correct, because the subtrees rooted at different cell inputs may overlap. However we can manage this problem by considering a forest of trees which together span \( N = (V, E) \). Let \( F = (V, E_F) \) a subnetwork of \( N \) such that for each \( v \in V - V_{sink} \) exactly one edge \((v, w) \in E_F\); then

\[
A(R) = \sum_{v \in V_{sink}} \alpha_R(v)
\]

\[
\alpha_R(v) = \begin{cases} 
0 & \text{if } v \in V_{po} \\
\text{area}(C) + \sum_{w \in I(C) \cap \text{pred}_F(v)} \alpha_R(w) & \text{if } v = O(C), C \in R
\end{cases}
\]

computes the area of any realization \( R \in \mathcal{R}_{pr} \); herein \( \text{pred}_F(v) \) denotes the predecessors of \( v \) within \( F \) and \( \text{pred}_F \) represents the reflexive and transitive closure of \( \text{pred}_F \). We will show now that the classical dynamic programming approach used for tree mapping can be generalized for duplication-free mapping. We start by proving that there always is a realization \( R \in \mathcal{R}_{pr} \) with all external nodes \( v \) simultaneously having minimal \( \lambda_R(v)\).

Lemma 3 Let \( N = (V, E) \) be an arbitrary network.

For each \( v \in V \) there is a realization \( R_v \in \mathcal{R}_{pr}(v) \) such that for all external nodes \( w \in V_v \) of the subnetwork \( N_v = (V_v, E_v) \) rooted at \( v \):

\[
\lambda_{R_v}(w) = \min_{R \in \mathcal{R}_{pr}(v)} \lambda_R(w)
\]

Proof: Induction on the depth of \( v \).

The induction basis is trivial because for each \( v \in V_{po} \) the value \( \lambda_{R_v}(v) = f_{po}(v) \) is independent of \( R \). For the induction step we consider any realization \( R \in \mathcal{R}_{pr}(v) \) which minimizes \( \lambda_R(v) \). Let \( C \) be the cell in \( R \) with \( O(C) = v \). By induction hypothesis we know that for each \( w \in I(C) \) there exists a realization \( R_w \) which fulfills \( \lambda_R(w) = \min_{R \in \mathcal{R}_{pr}(w)} \lambda_R(x) \) for all external nodes \( x \) in the network \( N_w \). By the second part of lemma 2 we know that \( R' := R_1' \cup R_2' \) with

\[
R_1' := N_v \quad R_2' := N_v \quad R_{po} := N_v \quad R_{sink} := N_v
\]
yields a duplication-free realization. Further for each external node $y$ of $R'$ the following condition holds

$$
R'_1 := \{ C \in R \mid E_C \cap E_y = \emptyset \}
$$

$$
R'_2 := \{ C \in R_w \mid E_C \cap E_y \neq \emptyset \}
$$

if $y \in V_w \implies \lambda_{R'}(y) = \lambda_{R_w}(y) = \min_{R \in R_w} \lambda_R(y)$

if $y \in V - V_w \implies \lambda_{R'}(y) \leq \lambda_R(y)$ because of the monotony of $f_C$. We iterate the construction for all $w \in I(C)$ (each time replacing $R$ by $R'$) and get a realization $R_w$ with the required property.

As a corollary we observe that there is a realization $R \in R_{pr}$ such that $\lambda_R(v)$ is minimal for all external nodes of $R$. This result suggests a dynamic programming approach for finding a realization of minimal cost.

Let

$$
\hat{A} := \min_{R \in R_{pr}} A(R)
$$

$$
\hat{\lambda}(v) := \min_{R \in R_{pr}(v)} \lambda_R(v) \text{ for all } v \in V
$$

Then $\hat{A}$ can be computed as follows

$$
\hat{A} = f_{po}(\hat{\lambda}(o_1), \ldots, \hat{\lambda}(o_n))
$$

with $\{o_1, \ldots, o_n\} = V_{po}$. The values $\hat{\lambda}(v)$ are obtained by a topological traversal through the network: For the primary inputs $v \in V_{pi}$ we have $\hat{\lambda}(v) = f_{pi}(v)$; for $v \in V - V_{pi}$

$$
\hat{\lambda}(v) = \min_{C \in E_{pr}(v)} f_C(\hat{\lambda}(I_1(C)), \ldots, \hat{\lambda}(I_m(C)))
$$

with $I(C) = \{I_1(C), \ldots, I_m(C)\}$. For each node $v$ we also store the cell $\hat{\rho}(v)$ which yields the minimum. Thus by a succeeding backward traversal we can determine which nodes $v_e$ should be external under an optimal solution and collect the corresponding cells $\hat{\rho}(v_e)$ within a set $\hat{R}$. Notice that there can be no ambiguities in determining whether $v$ to realize externally or internally; if there was any ambiguity we would have a situation as in the following sketch:

But this scenario immediately leads to a contradiction because the cell $C$ is secondary and therefore could never be a candidate for duplication-free mapping.

### 4 AT-tradeoff

In this section we consider the problem of how to minimize area under a delay constraint $T_{max}$; i.e. we are looking for

$$
\hat{A}(T_{max}) = \min_{R \in R_{po}} A(R)
$$

We have shown that this problem is NP-complete. However for fanout-free networks there are algorithms that solve the problem efficiently [13]. We will show that similar reasoning yields efficient methods for computing lower bounds and good approximation solutions for acyclic networks. For a first approach remember that we computed the area $A$ by means of a spanning forest $F$ of $N$. Essentially at fanout points we broke all fanout-branches but one (figure 3a). Area resources were added up only along edges of $F$ (unbroken edges). Now we use $F$ as an argument of the algorithm [13] for computing AT-tradeoffs in the fanout-free case. Obviously this yields a lower bound for $\hat{A}(T_{max})$; however the result is not equal to $\hat{A}(T_{max})$ because we have to make an optimistic assumption for the arrival time at the broken edges $e$; i.e. 0 or the unconstrained minimum $t(source(e))$.

For a better bound every fanout-branch of $v$ should "pay" for the arrival time at $v$; the edges starting in $v$ should "share" the cost for the subnetwork $N_v$. Moreover we want to continue to use the algorithm for fanout-free networks. Theoretically it would be possible to make $v$ fanout-free by generating an appropriate number of replicas of $N_v$, one for each fanout-branch (figure 3b). Each replica would be charged with its share in the cost for $N_v$. By iterating this process we could transform $N$ into a leaf-DAG $N'$ with the same cost. Obviously such an approach would be intractable because $N'$ could grow exponentially with the size of $N$. But we can work with $N$ as an implicit representation of $N'$.

We continue by working out this idea formally. We model the sharing of cost by edge labels $\epsilon(e)$.

Let $\epsilon : E \rightarrow [0, 1]$ be any function such that for all $v \in V - V_{sink}$ the following condition holds:

$$
\sum_{e \in E, \text{source}(e) = v} \epsilon(e) = 1
$$

For a realization $R \in R_{pr}$ we define $A_{\epsilon}(R)$ recursively by

$$
A_{\epsilon}(R) := \sum_{v \in V_{sink}} \alpha_{\epsilon, R}(v) \text{ with }
$$
\[
\alpha_{e,R}(v) = \begin{cases}
0 & \text{if } v \in V_{pi} \\
\text{area}(C) + \sum_{(x,y) \in E_C} \varepsilon(x,y) \cdot \alpha_{e,R}(x) & \text{if } v = O(C), C \in R \\
\infty & \text{if } v \in V_{pi}, t < 0 \\
\min_{C \in E_{pr}(v)} f(C,t) & \text{if } v \in V - V_{pi}
\end{cases}
\]

It is not difficult to see that \( A_{e}(R) = A(R) \) for all realizations \( R \). Actually if choosing \( \varepsilon \) in a manner such that for each node \( v \in V - V_{sink} \) there is exactly one edge \((v,w)\) with \( \varepsilon(v,w) = 1 \) we get our previous method for computing \( A(R) \). We now proceed as if we wanted to minimize the area of the implicitly represented leaf-DAG \( N' \) under a delay constraint:

\[
\hat{A}_{e}(T_{max}) = \sum_{v \in V_{sink}} \hat{\alpha}_{e}(v,T_{max}) \text{ with } \hat{\alpha}_{e}(v,t) = \begin{cases}
0 & \text{if } v \in V_{pi}, t \geq 0 \\
\infty & \text{if } v \in V_{pi}, t < 0 \\
\min_{C \in E_{pr}(v)} f(C,t) & \text{if } v \in V - V_{pi}
\end{cases}
\]

and \( f(C,t) = \text{area}(C) + \sum_{(x,y) \in E_C} \hat{\varepsilon}(x,y) \cdot \hat{\alpha}_{e}(x,t_{delay}(C)) \).

These computations are relevant only not for \( N' \) but also for \( N \) itself because of the following result:

**Lemma 4** For each \( R \in R_{pr} \) and each external node \( v \) of \( R \):

\( \hat{\alpha}_{e}(v,t_{delay}(R)) \leq \alpha_{e,R}(v) \)

The proof is by induction on the depth of \( v \). As a corollary we state

\[ \hat{A}_{e}(T_{max}) \leq \min_{R \in R_{pr}^{T_{max}}} A_{e}(R) = \min_{R \in R_{pr}^{T_{max}}} A(R) \]

thus \( \hat{A}_{e}(T_{max}) \) represents a lower bound for \( A(T_{max}) \).

Notice however that this bound is not necessarily exact: When computing \( \hat{\alpha}_{e}(v,t) \) we do as if we had to realize \( N' \), not \( N \); i.e. during minimization different replicas of the same subnetwork \( N_{e} \) are considered independent one of the other. This may result in a different realization for each replica. The problem becomes evident if we try to construct an optimal realization \( \hat{R} \in R_{pr} \) by means of a backward traversal. Assume that we know all \( \hat{\alpha}_{e}(v,t) \) and the cells \( \hat{\rho}_{e}(v,t) \) which yield the minimum in each case. We start at the nodes \( v \in V_{sink} \) and realize them by means of \( C_{v} := \hat{\rho}_{e}(v,T_{max}) \). Further we inform each input \( w \in I(C_{v}) \) of its required arrival time \( T_{max} - \text{delay}(C_{v}) \) and put \( w \) into a queue \( Q \) of nodes to be realized externally. In the same manner we process all nodes of \( Q \) in order of decreasing depth. All seems very simple. But there is a problem with this approach. What should we do if there are different timing requirements for a node \( v \)?

A pragmatic solution for this problem is to comply with the most stringent requirement. This yields a realization \( \hat{R} \) that meets \( T_{max} \). However \( \hat{R} \) may need more area then necessary. The quality of the approximation solution can be estimated by a comparison between \( A(R) \) and the lower bound \( \hat{A}_{e}(T_{max}) \). Experimentally we found that for FPGAs our algorithm produces optimal tradeoffs most time. If it does not, iteratively improving \( \varepsilon \) may help. Moreover we can use \( \hat{A}_{e}(T_{max}) \) as a lower bound in a branch-and-bound algorithm which always results in optimal solutions.

### 5 Duplication of logic

If we confine ourselves to duplication-free mapping we mostly let slip delay-minimal solutions; sometimes we even lose area-minimal realizations (as in figure 1). Duplication for minimal area is NP-complete whereas delay optimization under duplication can be done by similar methods as in the duplication-free case: Let

\[ \hat{\tau}(v) = \begin{cases}
0 & \text{if } v \in V_{pi} \\
\min_{C \in E_{pr}(v)} \max_{w \in I(C)} \left[ \hat{\tau}(w) + \text{delay}(C) \right] & \text{if } v \in V - V_{pi}
\end{cases} \]

then \( \hat{T} = \max_{v \in V_{pi}} \hat{\tau}(v) \) yields the optimal delay. Computing a delay-optimal realization \( \hat{R} \) can be done by means of a backward traversal as above; however \( A(R) \) may be very bad. On the other hand, the AT-tradeoff algorithm for the duplication-free case only explores a subdivision of the general solution space with low area and relatively high delay. How can we find solutions in between? Our approach to this problem is a preprocessing pass performing duplications towards a delay constraint \( T_{max} \); i.e. we try to manage with a minimum number of duplications such that \( T_{max} \) just gets within reach of an algorithm for the primary problem.

For each node \( v \) we compute not only the optimal arrival time \( \hat{\tau}(v) \) but also the optimal arrival time \( \hat{\tau}_{pr}(v) \) under the constraint that we use a primary cell for the node \( v \):

\[ \hat{\tau}_{pr}(v) = \begin{cases}
0 & \text{if } v \in V_{pi} \\
\min_{C \in E_{pr}(v)} \max_{w \in I(C)} \left[ \hat{\tau}(w) + \text{delay}(C) \right] & \text{if } v \notin V_{pi}
\end{cases} \]

Notice that we use \( \hat{\tau}(w) \), not \( \hat{\tau}_{pr}(w) \) on the righthand side of the equation. Thus for all nodes but \( v \) we are free to use arbitrary cells. Then we compute a realization \( \hat{R} \) such that \( T(\hat{R}) \leq T_{max} \); we try to manage with a minimum number of secondary cells in \( \hat{R} \). In order to fulfill this task we perform a backward propagation of timing requirements similar to the approach in section 4: First we put all primary outputs \( v \) into a queue \( Q \) and note down their required arrival time \( \tau_{req}(v) = T_{max} \). For all other nodes \( u \) we initialize \( \tau_{req}(u) = \infty \). Then we process all nodes \( v \) of \( Q \) in order of decreasing depth; if \( \tau_{req}(v) \geq \hat{\tau}_{pr}(v) \), we add the primary cell \( C \) that yields \( \hat{\tau}_{pr}(v) \) to \( \hat{R} \); otherwise we add the secondary cell \( C \) that yields \( \hat{\tau}(v) \) to \( \hat{R} \) and transform it into a primary cell by the appropriate duplications. In either case we put each input \( x \in I(C) \) into \( Q \) and set

\[ \tau_{req}(x) = \min \left\{ \tau_{req}(x), \tau_{req}(v) - \text{delay}(C) \right\} \]
After all nodes of $Q$ have been processed we finally get a realization $\tilde{R}$ with $T(\tilde{R}) \leq T_{\text{max}}$. $A(\tilde{R})$ may still be suboptimal. But we also have constructed a network $\tilde{N}$ such that $T_{\text{max}}$ is within reach of the algorithm of section 4. By starting this algorithm with $\tilde{N}$ as an argument we get a realization $\tilde{R}$ which minimizes area under the delay constraint $T_{\text{max}}$. We may hope that $A(\tilde{R})$ is close to the minimum for the entire problem, because the preprocessing pass carried out duplications only along the paths critical under $T_{\text{max}}$.

As we saw in figure 2 duplications may also be useful for minimizing area. We are currently working with two very simple heuristics, one before and one after the main pass. The preprocessing heuristic tries to minimize the number of cells instead of the area. We are looking for secondary cells all of whose pins $v$ surely (or probably) have to be realized externally; e.g. if $v \in V_{in} \cup V_{op}$ or $\text{fanout}(v)$ is very high. Such cells are transformed into primary ones. This results in a network which surely (or probably) can be realized with at most as many cells as the original network. Actually the number of cells may be expected to decrease because many small fanout-free regions are replaced by fewer, even though larger regions. The postprocessing heuristic greedily looks for small cells with multiple fanout such that merging these cells into all succeeding fanout-free regions results in a decrease of cost.

6 Benchmarking

The prototype of such a technology mapper was implemented. We started with LUT-based FPGAs as a target technology. Cells were determined by means of an iterated maxflow-algorithm as proposed by [11]. However we observed that not all cells need to be enumerated for an optimal realization. Moreover we have developed efficient methods that just compute sufficient subsets of cells. The details of this approach can be found in [12].

Table 1 shows some of our first experimental results obtained for the 1989 MCNC benchmarks. Size and depth of the initial network, of a minimum area and of a minimum depth solution (with 5-input LUTs) are presented for each circuit. We list only the large circuits ($\geq 500$ nodes within the initial network) and some samples for which [2] demonstrated their mapper FlowMap-r to be superior to Misp-gpa [11] and Chortle [4]. A fair comparison is difficult, because [2, 4, 11] perform a MIS-pass before mapping, whilst we are currently working with the benchmarks as they are. However for observing the trend we mark our results which are better (worse) than those reported for FlowMap-r with $+$,$+$,$+$ ($-,-,-$). Insignificant differences (up to 5% in area and 1 level in depth) are represented by a single sign; $+$ and $-$ symbolize more important differences and O marks a draw. As a first conclusion we can say that our program although starting with an unoptimized network stands comparison with the best tools for FPGAs reported up to now. Our minimum area realizations tend to be a little better, whilst minimum depth solutions are somewhat worse. For the sake of a more detailed comparison however we will have to use the same initial networks as [2, 4, 11].

<table>
<thead>
<tr>
<th>Circuit</th>
<th>initial netw. $(A_0, T_0)$</th>
<th>min. area $(A, T_w)$</th>
<th>min. depth $(A_w, T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>9aynnl</td>
<td>(108,14)</td>
<td>$+$ (55,7)</td>
<td>0 (55,7)</td>
</tr>
<tr>
<td>C1355</td>
<td>(619,27)</td>
<td>(66,5)</td>
<td>(74,4)</td>
</tr>
<tr>
<td>C1958</td>
<td>(715,38)</td>
<td>(143,11)</td>
<td>(161,7)</td>
</tr>
<tr>
<td>C2870</td>
<td>(1100,25)</td>
<td>(204,12)</td>
<td>(306,7)</td>
</tr>
<tr>
<td>C3540</td>
<td>(1161,41)</td>
<td>(345,21)</td>
<td>(390,11)</td>
</tr>
<tr>
<td>C409</td>
<td>(597,25)</td>
<td>$+$ (66,5)</td>
<td>(74,4)</td>
</tr>
<tr>
<td>C5515</td>
<td>(1388,35)</td>
<td>(417,9)</td>
<td>(426,8)</td>
</tr>
<tr>
<td>C6288</td>
<td>(2400,123)</td>
<td>(814,22)</td>
<td>(814,22)</td>
</tr>
<tr>
<td>C7552</td>
<td>(3087,38)</td>
<td>(866,9)</td>
<td>(895,7)</td>
</tr>
<tr>
<td>C680</td>
<td>(394,23)</td>
<td>$+$ (100,12)</td>
<td>$+$ (143,7)</td>
</tr>
<tr>
<td>ahu2</td>
<td>(300,39)</td>
<td>$-$ (130,17)</td>
<td>$-$ (153,9)</td>
</tr>
<tr>
<td>ahu4</td>
<td>(720,42)</td>
<td>$+$ (218,16)</td>
<td>$-$ (258,11)</td>
</tr>
<tr>
<td>apexc</td>
<td>(831,10)</td>
<td>$+$ (212,8)</td>
<td>$-$ (213,5)</td>
</tr>
<tr>
<td>apex7</td>
<td>(269,17)</td>
<td>$+$ (77,7)</td>
<td>$-$ (86,4)</td>
</tr>
<tr>
<td>count</td>
<td>(134,34)</td>
<td>$+$ (31,5)</td>
<td>$-$ (31,5)</td>
</tr>
<tr>
<td>dos</td>
<td>(4679,20)</td>
<td>(1540,10)</td>
<td>(2005,6)</td>
</tr>
<tr>
<td>fg2</td>
<td>(1137,15)</td>
<td>(410,6)</td>
<td>(532,4)</td>
</tr>
<tr>
<td>k2</td>
<td>(1971,19)</td>
<td>(787,8)</td>
<td>(876,0)</td>
</tr>
<tr>
<td>pair</td>
<td>(1674,26)</td>
<td>(390,9)</td>
<td>(451,6)</td>
</tr>
<tr>
<td>rot</td>
<td>(689,25)</td>
<td>(210,13)</td>
<td>(237,7)</td>
</tr>
<tr>
<td>too.large</td>
<td>(748,24)</td>
<td>(300,15)</td>
<td>(326,8)</td>
</tr>
<tr>
<td>vda</td>
<td>(926,14)</td>
<td>(396,8)</td>
<td>(487,5)</td>
</tr>
<tr>
<td>x3</td>
<td>(765,19)</td>
<td>(250,7)</td>
<td>(252,5)</td>
</tr>
<tr>
<td>z4ml</td>
<td>(70,10)</td>
<td>$+$ (9,2)</td>
<td>$+$ (9,2)</td>
</tr>
</tbody>
</table>

Table 1: Results on some of the MCNC Benchmarks

References