An Integrated Technology Mapping Environment

Alan Mishchenko Satrajit Chatterjee Robert Brayton

Department of EECS
University of California, Berkeley
{alanmi, satrajit, brayton}@eecs.berkeley.edu

Maciej Ciesielski

Department of ECE
University of Massachusetts, Amherst
ciesiel@ecs.umass.edu

Abstract

This paper describes a flexible and efficient environment for technology mapping featuring a common set of algorithms for both standard cells and LUT-based FPGAs. The algorithms and data structures can be customized for various objectives and constraints, such as delay optimization, area recovery, and power and placement improvement under delay and area constraints. Experimental results show superior results for both standard cells and FPGAs when compared with state-of-the-art mappers.

1 Introduction

Most approaches to technology mapping for both standard cells and LUT-based FPGAs focus on delay and area optimization. A typical mapping scenario is to find the optimal delay possible for the given logic structure of the subject graph mapped using the given library [5][18][17], followed by heuristic approaches to area recovery [15][6][4][13]. The mappings produced are not optimized for power, may lead to congested placements, and often require extensive fanout optimization by buffering, gate duplication, and gate sizing.

Several approaches have been proposed to alleviate these problems by performing incremental changes to the mapped netlist. For example, remapping for improved placement has been considered in [3][26][16][24]. A difficulty with these approaches is that optimization for objectives other than delay and area is deferred to a resynthesis stage when the complete set of choices explored during mapping is no longer available. As a result, resynthesis has to rediscover a subset of these choices, which leads to long runtimes and suboptimal quality.

The contributions of this paper are three-fold.

1. We developed an integrated set of algorithms and data structures, which with minor changes can perform mapping while optimizing for various objectives, minimizing area after delay-optimal mapping, and improving power and placement by minimizing switching activity and wirelength. We use load-independent delay model for the standard cell mapping, while showing that the final results still show advantage when a load-dependent model is used to evaluate them. Furthermore, we propose a way of incorporating load-dependent information into the iterative optimization performed after the initial load-independent mapping.
2. The environment is applicable for FPGAs as well as standard cells. This allows for uniform treatment of both types of mapping and leads to a cross-fertilization of these research areas. For instance, the combination of mapping and retiming developed for FPGAs [20][8] is applicable to standard cell mapping while preserving the optimality claims: the final mapping found is the best one in terms of delay over all possible mappings and retiming of the original circuit.
3. We show several specific improvements over state-of-the-art mappers. These are presented in four case studies:
   3.1. The first shows multi-objective optimization for power under delay and area constraints using estimates of switching activity which is incrementally updated in the process of mapping.
   3.2. The second shows multi-objective optimization for placeability under delay and area constraints using estimates of wirelength incrementally updated during the mapping.
   3.3. The third looks at new method for area recovery under delay constraints combining two heuristics, area flow and exact area at a node. These are shown to be equally applicable to both standard cell and FPGA mapping, leading to area recovery comparable to, or better than, the best published results, e.g. our mapper equals DAOmap in delay while using 7% fewer LUTs.
   3.4. The last study illustrates how the simplified delay assumptions currently used can be extended to more accurate delay models. As an example, we show how to extend to the SIS load-dependent delay model. As one point of comparison, we produce results with the same area and 39% less delay compared to SIS using the same delay model.

In general, the new environment has been shown to be efficient, both in terms of results and run-time (improving on optimization quality over state-of-the-art mappers while using a fraction of the compute time).

A software prototype of the technology mapping environment has been implemented and is being extended and fine-tuned. For example, the currently used models to compute some metrics are simple and will be replaced by more accurate ones. Thus, the wirelength computation, done assuming the placement of cells or LUTs in rows by their topologic level in the netlist, will be replaced by an incremental placer.

The paper is organized as follows. Section 2 describes background on technology mapping. Section 3 describes the algorithms and data structures constituting the proposed technology mapping environment. Section 4 describes the principles and procedures of multi-objective optimization. Section 5 presents the case studies dealing with mapping for power, placeability, area recovery for standard cells and FPGAs, and load-dependent delay models. Section 6 concludes the paper and outlines future work.

2 Background

A *network* is a directed acyclic graph, in which each node is represented by a single-output completely specified Boolean function. The external inputs to the network are called *primary inputs* (PIs) and the external outputs are called *primary outputs* (POs). The nodes providing immediate inputs to a given node are called its *fanins*. The nodes driven by the output of a given node are called its *fanouts*. The *maximum fanout-free cone* (MFFC) of a node is the set of nodes in the transitive fanin cone of a node, which fanout only to other nodes in the MFFC.

The combinational network used as an input for mapping is called the *subject graph*. The subject graph is derived by applying algebraic decomposition, as described in [21], to the technology-independent netlist resulting in a network of two-input ANDs and inverters called an And-Inv Graph (AIG). The notions of the subject graph and the AIG are used interchangeably. In this work, we consider only combinational networks. If the network is sequential it is cut at the latch boundary and the resulting combinational network is considered for mapping.

Technology mapping consists of expressing the functionality of the network using a combination of gates from the library [11]. Each gate in the library implements a completely specified Boolean function of a limited number of inputs. Besides the functionality, a gate in the library is characterized by technology-dependent parameters, such as pin-to-pin delays, gate size, maximum load, etc.

In this work, we study technology mapping from a functional point of view using a simple delay model, which takes into account pin-to-pin delays for rising and falling phases while assuming that the delay is independent of the load. The primary advantage of this model (used commonly in a gain-based flow) is that it allows quick evaluation of a large number of topologies and thereby enables the efficient mapping environment that is described in Section 3. Of course, the final performance of the circuit is not captured by this simple model, but as we show in our experiments in Section 5.4, making decisions based on this simple model can still lead to better final performance than making decisions based on a more accurate model, but looking at fewer topologies.

The above discussion notwithstanding, we plan to extend the mapper to make some of the later decisions (in subsequent mapping passes) using more accurate delay models. This would further improve the quality of results.

When technology mapping is performed for FPGAs, it is assumed that a gate (a *k*-input look-up table or LUT) can implement any function up to *k* inputs. LUTs of different sizes and delays can also be used by the mapper. In this work, LUTs are assumed to have no more than 6 inputs. This is motivated by the fact that exhaustively computing cuts for more than 6 inputs is not practical. However, in on-going research not presented here, we consider an extension of mapping for LUTs of any size (i.e. 10) with constraints on the set of functions that can be implemented.

A *cut* for node *n* in the Boolean network is a set of nodes *C*, such that every path from the PIs to node *n* passes through at least one node in *C*. Node *n* is called the *root* of the cut. Nodes in *C* are called the *leaves* of the cut. The computation of all the cuts of size up to six for all nodes in the network can be performed efficiently by traversing the network in the topological order from the PIs to the POs [7].

Each cut is characterized using a completely specified Boolean function of its root in terms of the leaves [12][21]. The truth table of this function, conveniently represented as a 32-bit or 64-bit bit-string, is used to establish the connection between a cut, and the gates implementing it. The gates are hashed by their truth tables and, for each cut one constant-time look-up is performed, returning the pointer to the set of gates with the same functionality. This process is called *matching*. In the software implementation, the connection between gates and cuts is made, up to the phase-assignment, which increases the mapping quality, but this detail is not important for the material discussed in the rest of this paper.

A *match* of a cut is a gate that can implement the cut. A *match* of a node is a match of some cut rooted at this node. The *best match* of a node is the match, selected to map the node to optimize some optimization criterion, such as delay, area, or wirelength.

3 Technology mapping environment

The overall flow of technology mapping discussed in this paper is shown in Figure 1. The input to mapping is the network, the gate library, and the delay constraints. Typically, the network has been optimized before mapping by technology-independent synthesis [1]. The library is pre-processed to compute the truth tables for each gate and the gates are hashed by their truth tables. The delay constraints include the PI arrival times and the PO required times. If the delay constraints are not given, the arrival times of the PIs are set to 0, and the required times of the POs are set to the minimum delay achieved by technology mapping of the circuit with the given logic structure. (In practice this is done by setting the required times to 0.)

```
TechnologyMapping( network N, library L, delay constraints D)
{
        Step 1: G = TransformNetworkIntoAIG( N );
        Step 2: ComputeCutsAndMatches( G, L );
        Step 3: mapping M1 = MappingForMinDelay( G, L );
                if ( delay constraint D is infeasible for M1 )
                        return FAIL;
        Step 4: ComputeRequiredTimes( G, M1 );
                mapping M2 = OptimizationPass1( G, M1, D );
                ComputeRequiredTimes( G, M2 );
                mapping M3 = OptimizationPass2( G, M2, D );
                …
        Step 5: TransformToMappedNetwork( G, Mk );
}
```

Figure 1. Pseudo-code of the technology mapping flow.

Technology mapping begins by deriving an AIG subject graph *G* in Step 1. In Step 2 the AIG nodes are annotated with their matches. Step 3 assigns the best match for each node while trying to achieve the minimum delay at the node. If there is a tie, the match leading to the smallest area, is taken. If the delay constraints are infeasible, the mapper returns FAIL. In the subsequent steps, the required times of the POs are set according to the delay constraints. These required times at the POs are never violated in the course of the following transformations.

At the beginning of each optimization pass in Step 4, the required times of the internal nodes are recomputed. This is necessary since with every optimization pass, the mapping changes (to better meet the optimization criterion), and consequently, the internal required times change (in order to maintain the fixed PO required times.) The required times for the internal nodes are used to bound the search for possible re-mappings during the optimization pass.

On termination, the best mapping is returned in Step 5. This best mapping starts at the POs and continues recursively to the leaves of the cuts used for the best matches until the PIs are met. Note that although in the algorithm of Figure 1 the best matches are assigned for each AIG node, not all of the AIG nodes are used in the mapping, only those which correspond to the outputs of the gates used by the best matches. If an internal node is not used in the current mapping, its required time is set to $+\infty$ in Step 4.

The common technology mapping algorithms and data structures of the environment described in this paper consist of the subject graph *G* derived in Step 1, a set of matches assigned in Step 2, the arrival times computed in Step 3 and the required times repeatedly computed in Step 4.

The flexibility of the environment comes from the fact that *all* matches are available during *each* optimization pass. Although only one match is selected as best at a node according to the current cost function, all matches are stored for future use. In practice, the number of matches of an internal node ranges from 10 to 1000, which gives substantial freedom to modify the selected mapping as the cost function changes.

The mapping environment gains in optimality if the number of matches stored at each node is increased. This can be achieved by using supergates and choice nodes [20][21]. Both approaches reduce structural bias by adding decompositions to the subject graph that were not present in the original netlist. Supergates enumerate gate combinations of limited logic depth (as a preprocessing step on the gate library) while choice nodes accumulate functionally-equivalent structurally-different logic cones that occur during separate synthesis steps.

4 Multi-objective optimization

This section discusses the adaptation of the optimization performed in Step 4 of the technology mapping flow of Figure 1 for various optimization criteria.

4.1 Optimization pass

Optimization is performed in separate passes over the mapped network. The pseudo-code of the generic optimization pass is shown in Figure 2. The AIG nodes of the subject graph are visited in topological order from PIs to POs. At each node, all matches are considered, and the current cost function is evaluated for each match. The match with the best cost is found and assigned as the best match at the node.

```
OptimizationPass( subject graph N, cost function F )
{
    match mBest;
    for each AIG node n of the subject graph N from PIs to POs
    {
        mBest = NULL;
        for each match m of node n
        {
            if ( CompareMatches^F( m, mBest, n ) > 0 )
                mBest = m;
        }
        set mBest as the best match at node n;
    }
}
```

Figure 2. Pseudocode of the optimization pass.

During optimization, the nodes in the transitive fanin cone of the given node are already mapped, so their arrival times are known. However, the fanouts are known only for the previous iteration. These fanouts are used to estimate some parameters, such as load, with the understanding that they may change in the current iteration (see Section 5.4). One way of partially overcoming the fanout bias, is to compute a linear combination of fanout counts in two consecutive iterations, as suggested in [19]. On termination, the procedure in Figure 2 assigns the best matches at each node using the current cost function while delay constraints are taken into account.

The number of optimization passes performed depends on how quickly convergence is reached. The speed of convergence depends on the cost function and the circuit. Since estimation of some parameters (such as fanouts) is not accurate, the cumulative improvement in the given cost function, which is used to measure the convergence, is not monotonic. For example, in area flow minimization, we often observe that, after several iterations, the total area flow of the mapped network starts oscillating around some value. Typically the magnitude of oscillation is round 1% of the total amount of the area flow. In general, we found that, for most of the parameters, one optimization pass is enough, while some of them may require two or three passes.

A typical cost function uses several metrics with various constraints, such as delay and area, or area and wirelength. Typical constraints are the required times at a node and a limit on the load of each gate type. In a given cost function, several metrics can be grouped according to some priority. A primary metric evaluates the quality of a match while lesser metrics are used as tie breakers.

Figure 3 shows a typical cost function, which uses area flow as the primary metric and arrival times as a secondary metric. (Area flow is formally defined below in Section 4.2.) It assumes that match m_1 is the current best match at the node, while match m_2 is a new candidate. It returns -1 if m_1 is strictly better than m_2, 1 if m_2 is strictly better than m_1, and 0 if both matches are equal when compared using these metrics.

```
int CompareMatches( match m1, match m2, node n )
{
    // using the required time as a constraint
    if (ArrivalTime(m2) > RequiredTime(n) )
        return -1;
    // comparing area flows
    if (AreaFlow(m1) < AreaFlow(m2) )
        return -1;
    if (AreaFlow(m1) > AreaFlow(m2) )
        return 1;
    // a tie; comparing the arrival times
    if (ArrivalTime(m1) < ArrivalTime(m2) )
        return -1;
    if (ArrivalTime(m1) > ArrivalTime(m2) )
        return 1;
    return 0;
}
```

Figure 3. A cost function to compare two matches.

There are two approaches to optimize for multiple objectives:

- combining several metrics into one cost function (as shown in Figure 2 for area flow and arrival times) and performing one or more optimization passes with this single function;
- repeating the optimization passes with different cost functions corresponding to each objective.

Experimentation is needed to determine which approach is best for a specific optimization type. For example, we observed that area recovery is best performed with one pass of area flow minimization (see Figure 3), followed by one pass of exact area

minimization where area flow in Figure 3 is replaced by the area of the MFFC of gates used to implement the match. Area recovery is described in more detail in Section 5.3.

4.2 Metrics

An optimization objective comes with a *metric*, which indicates (perhaps roughly) how well the current netlist meets the objective. For example, dynamic power may be measured by the amount of switching activity at the inputs and outputs of the gates under random simulation. Although random simulation only approximates power dissipation, it is fast. A good metric should be a compromise between speed and accuracy. The presented mapping environment can be used with arbitrary metrics, as long as computation and updating procedures are provided to evaluate and incrementally update the metric when the mapping changes.

Delay and *area* of a mapped circuit are well-known metrics for evaluating the mapping quality.

Area flow [19] (*effective area* [7]) 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 for the PIs is set to 0. Area flow at a node n is: $AF(n) = [Area(n) + \Sigma_i AF(Leaf_i(n))]/ NumFanouts(n)$, where $Area(n)$ is the area of the best match at n, $Leaf_i(n)$ is the i-th leaf of the cut of the best match at n, and $NumFanouts(n)$ is the number of fanouts of n.

If nodes are processed from the PIs to the POs, updating area flow is similar to computing it. The advantage of area flow over area of the cone is that area flow gives a more global view of the mapping. Area flow estimates the amount of sharing between the logic cones without the need to re-traverse the cones.

Switching activity under random simulation is computed using a simulation vector s stored at each node of the subject graph. We assume that the probability of the output of a gate switching is $Prob(s) = 2 * Nzeros(s) * Nones(s) / Nbits(s)^2$, where $Nzeros(s)$, $Nones(s)$, and $Nbits(s)$ are the numbers of zeros, ones, and bits, respectively, in the simulation vector s of the gate.

Wirelength is a commonly used metric to evaluate the quality of placement. Minimizing the sum of lengths of all wires during technology mapping is a heuristic for improving the final placement of the design, even if the placement information from the mapping is not used by the final placer.

5 Case-studies

The methods discussed in the following four case studies were implemented in a stand-alone ASIC/FPGA mapper, tested in the MVSIS environment [22]. The runtimes (except Case 3) are measured on a 1.6GHz computer with 1Gb RAM under Windows XP. The first two cases illustrate the flexibility of the environment in using different metrics. The third case shows its superiority (in area/delay mode) in both standard cell and FPGA mapping compared to state-of-the-art methods, while the fourth case illustrates how the delay model can be adapted to a load-dependent delay model.

5.1 Power-aware mapping

In this section, we show how mapper can be used to reduce the dynamic power as measured using switching activity under random simulation. Switching activity is measured before mapping begins by propagating random simulation patterns through the subject graph and recording the switching probability at each AIG node. Switching activity of the network is computed as the sum of the probabilities of switching of all the signals including the PIs.

In this study, we first map the circuit in the *baseline* mode (mapping for minimum delay followed by heuristic area recovery). Then, we perform one optimization pass, which greedily selects the best match at each node to reduce switching activity.

The cost function computes the total of the switching activities of the gates in the MFFC of each match. The match that satisfies the required times and minimizes the switching activity is selected at each node. The area of the MFFC is used as a tie-breaker in prioritizing matches. As a result of this optimization, the total switching activity is reduced but the area can increase. This is because, in the process of remapping, larger gates can be selected. These gates typically subsume AIG nodes that switch often, resulting in an overall reduction of switching activity.

More accurate metrics to measure power dissipation in a mapped circuit can take gate capacitance and leakage into account. Another interesting possibility is to use the notion of "switching flow", which is similar to that of area flow. Switching flow may have a complementary strength compared to the exact switching activity of the MFFC used in this experiment.

Experimental results

Table 1 gives the statistics for a selection of benchmarks from the MCNC, ITC'99, and PicoJava benchmark sets. The benchmarks are chosen randomly among those having large size. The benchmarks for the experiments were pre-optimized by *mvsis.rugged* script in the MVSIS environment [22]. The standard cell library used was *mcnc.genlib* from the SIS distribution [25]. The gate delays were assumed to be load-independent.

The first column contains the benchmark name. The next three columns contain the number of inputs, outputs, and latches. The column "FF lits" gives the number of literals in the factored forms of the nodes after optimization with *mvsis.rugged.* The last two columns contain the number of nodes and levels in the AIG derived by balancing the factored forms of the nodes. This AIG is used as the subject graph in all the experiments reported below.

Table 1. Statistics for the benchmarks used.

Name	Ins	Outs	Latches	FF lits	AIG nodes	AIG levels
b14	32	54	245	8343	5641	110
b17	37	97	1415	37636	28891	148
c1355	41	32	0	604	423	20
c3540	50	22	0	1345	970	41
c6288	32	32	0	3651	2135	133
des	256	245	0	3849	3252	16
i10	257	224	0	2556	1963	45
pj1	1769	1063	0	17710	14221	70
pj2	690	429	0	3300	2982	21

Table 2 reports the results of power minimization for the benchmarks. The second column contains the delay derived by minimum-delay mapping. This delay is used as a constraint for both area optimization in the baseline mapping mode and power optimization. The next three columns report area, power, and runtime of the baseline mapping, which performed area optimization without trying to reduce power. The last three columns contain the same parameters after running one round of power optimization for the mapping derived in the baseline mode.

Table 2. Reduction in switching activity during mapping.

Name	Delay	Baseline mapping			Power optimization		
		Area	Power	T, s	Area	Power	T, s
b14	74.90	12676	1874	1.7	13209	1715	1.8
b17	103.80	55508	6116	6.1	60163	5428	6.5
c1355	17.30	1398	219	0.2	1386	206	0.3
c3540	33.50	2183	296	0.2	2276	268	0.4
c6288	83.10	7499	1108	1.2	7124	958	1.3
des	13.60	6420	668	0.6	7008	608	0.7
i10	36.20	4095	520	0.6	4256	476	0.6
pj1	41.80	27094	4025	2.8	28497	3738	2.8
pj2	15.90	5118	925	0.3	5626	877	0.4
Ratio		1.00	1.00	1.00	1.04	0.91	1.24

Table 2 shows that power optimization results in a 9% reduction in the switching activity, a 4% increase in area and a 24% increase in runtime. (Delay remains unchanged.)

5.2 Placement-aware mapping

In this study, we demonstrate the use of wirelength as a metric for optimization.

We used the following simplistic way of deriving the placement information. Before mapping, the levelized subject graph is placed by assigning each AIG node a row equal to its logic level (starting from the PIs) and a column equal to its place in the row, derived by a DFS traversal (starting from the POs). In this traversal, each AIG node is added to its row after all the previously visited nodes have been added. The rows are balanced to create a placement symmetric with respect to the vertical axis, as shown in Figure 4. The width and the height of each AIG node are set to 1.

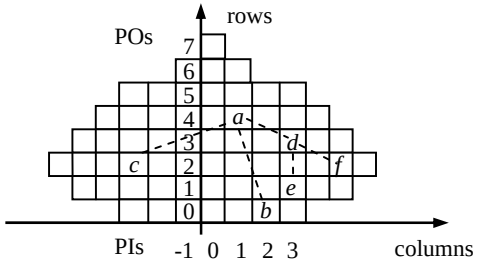

Figure 4. Illustration of the placement model.

The placement of the AIG nodes does not change during mapping. The placement of each gate is determined by the placement of the AIG node representing its root. The wirelength of a match is computed as the sum of the lengths of all two-terminal wires connecting the gates in the MFFC of the match. The length of a two-terminal wire is the Manhattan distance between its terminals (the semi-perimeter of the bounding box of the net originating at the output of a fanin gate and terminating at the input of a fanout gate). The wirelength of a match is used in the cost function similar to how area flow is used in Figure 3.

Figure 4 illustrates the proposed simple placement for the AIG with 8 logic levels (numbered 0 through 7). The PO of the AIG is on top. The PIs are at the bottom. Each cell of the placement corresponds to one node of the AIG. The wirelength of match *a* is the sum of the lengths of all wires in the MFFC of *a*. In this example, the MFFC includes the gate rooted at node *a* and the gate rooted at node *d*. The gates rooted at nodes *b, c, e,* and *f* are the leaves of the MFFC. The wirelength of the gate at *a* is dist(*ab*) + dist(*ac*) + dist(*ad*) = (dist(*ab*)$_x$ + dist(*ab*)$_y$) + (dist(*ac*)$_x$ + dist(*ac*)$_y$) + (dist(*ad*)$_x$ + dist(*ad*)$_y$) = (1+4) + (4+2) + (2+1) = 14. The wirelength of the gate at *d* is 5. Therefore, the wirelength of match *a* is 19.

Experimental results

Table 3 shows the results of wirelength reduction after one optimization pass performed on top of the baseline mapping. The experimental setting is the same as in the previous experiment, except that wirelength, instead of the switching activity, is the primary parameter in the cost function used for optimization. The notations in Table 3 are the same as in Table 2, except that instead of measuring the switching activity, the wirelength is measured.

Table 3. Reduction in wirelength during mapping.

Name	Delay	Baseline mapping			Wirelen optimization		
		Area	Wirelen	T,s	Area	Wirelen	T,s
b14	74.90	12676	655478	1.7	13289	589641	1.9
b17	103.80	55508	6570636	6.1	58260	5843631	6.7
c1355	17.30	1398	13282	0.2	1429	12878	0.2
c3540	33.50	2183	39176	0.2	2288	36297	0.2
c6288	83.10	7499	172404	1.2	7513	141939	1.3
des	13.60	6420	471372	0.6	6634	451189	0.7
i10	36.20	4095	123604	0.6	4157	114192	0.7
pj1	41.80	27094	4034882	2.8	28145	3680862	3.1
pj2	15.90	5118	384638	0.3	5317	359665	0.3
Ratio		1.00	1.00	1.00	1.03	0.92	1.08

Table 4 shows the corresponding numbers for technology mapping with supergates [20]. In this case, it is assumed that the gates inside one supergate are placed close to each other. In the approximate computation, the increase is wirelength due to the wires connecting the gates inside a supergate are ignored because the wires between the supergates are on average 50x longer than the size of the supergate using the given placement model. Note that the delays, areas, and runtimes reported in Table 4 differ from those in Tables 2 and 3. This is because supergates lead to improvements in delay at the price of increased runtime.

Table 4. Reduction in wirelength when using supergates.

Name	Delay	Supergate mapping			Wirelen optimization		
		Area	Wirelen	T,s	Area	Wirelen	T,s
b14	47.80	12666	582524	8.8	15198	478876	13.8
b17	65.40	54108	6097610	31.9	63460	4978561	49.9
c1355	14.70	1366	10798	0.6	1388	10273	0.9
c3540	25.50	2756	44556	1.1	2944	39186	1.7
c6288	63.70	9336	174895	5.1	9517	135210	7.7
des	12.20	7487	540267	2.7	7939	422711	4.3
i10	25.20	4433	124902	1.9	4848	114202	2.8
pj1	28.00	28946	4021123	13.6	32038	3462869	21.7
pj2	12.80	5673	398874	1.6	6230	353871	2.4
Ratio		1.00	1.00	1.00	1.09	0.85	1.54

The conclusion from Tables 3 and 4 is that the wirelength can be reduced by 8% after baseline mapping and by 15% after mapping with supergates, at the cost of 3% and 9% increase in area and some increase in runtime. The use of supergates leads to a more substantial reduction because of the additional structural freedom exploited by the mapper. We expect that the use of choice nodes [20] will lead to even larger reductions.

It should be noted that, due to the simplistic placement model, these measurements are very approximate and may not lead to the same improvements after an accurate placement. Future

experiments will include the use of a standard cell placer for assessing wirelength gains from the simplistic model. Another experiment will include the use of an incremental placer during the mapping phase.

5.3 Area recovery for standard cells and FPGAs

Exact area minimization during technology mapping for DAGs is NP-hard [10] and therefore not tractable for large circuits. However, various heuristics for approximate area minimization during mapping [14][13][19][27][4][6] have shown good results.

In this study, we use a combination of two known heuristics, that perform well in practice and when combined with the other algorithms in our mapping environment provide superior results compared to state-of-the-art mappers. We demonstrate this for both standard cell and FPGA mapping using the same basic algorithm.

The first heuristic optimizes the area flow of a network mapped first for optimum delay. At each node, the match that optimizes area flow is selected, provided that it does not violate the required time. (The exact cost function is shown in Figure 3.) The second heuristic looks at the area to be gained by locally updating the best match at a node. In this case, the area cost of a match is equal to the sum of areas of all gates in the MFFC of the match, i.e. the gates to be removed from (added to) the mapping if the match is not used (is used).

The advantage of using these two heuristics in this particular order is that they are complementary; area flow has a global view (selecting logic cones with more shared logic) while the exact area at a node has a local view.

Experimental results

In this experiment, the mapping flow, outlined in Figure 1, is applied to both standard cell and LUT-based FPGA mappings.

Standard cells

Table 5 contains the mapping results for standard cells. Some of the smaller benchmarks have been excluded from the set while a larger one (s15850) was included. The circuits were pre-optimized by *mvsis.rugged.* The first column lists the benchmark name. The next two columns show the results of a delay-optimal mapping in SIS using a load-independent library *mcnc.genlib*.

Table 5. Comparison of the two area-recovery heuristics for standard cell mapping.

Name	SIS		New mapping with area recovery					
	Delay	Area	Delay	Area	1	2	1+2	T, s
b14	133.7	12701	47.8	12702	.55	.55	.61	9.5
b17	174.5	56273	65.4	54338	.53	.49	.58	39.6
des	21.9	5769	12.2	7545	.25	.21	.28	2.8
i10	58.2	3815	25.2	4448	.46	.47	.53	1.9
pj1	66.3	27073	28.0	28978	.45	.48	.53	14.9
s15850	47.5	6000	27.8	6429	.43	.43	.47	1.8
Average					.45	.44	.50	
Ratio	1.00	1.00	0.46	1.10				

Columns 4 and 5 show the results of the new technology mapping with area recovery using supergates [21], which are used to increase structural flexibility exploited in technology mapping. The area and delay numbers are those achieved by the mapper with both area flow and exact area heuristics.

The next three columns try to isolate the benefits of each heuristic. The column labeled "1" shows the area **reduction** achieved over the first pass of the mapper (i.e. the delay optimal pass) by applying only the area flow heuristic. Likewise, the column labeled "2" shows the area reduction achieved by applying only the exact area heuristic.

The column labeled "1+2" shows the reduction when area flow optimization is followed by exact area optimization. These numbers correspond to the absolute numbers reported in column 5. Thus for the pj1 benchmark, after the first delay-optimal mapping pass, the area is 61655 (not shown in table). If only area flow is used for recovery, the area is 33910 (reduction of 0.45); if only exact area is used the area is 32060 (reduction of 0.48), and if both are used, the area is 28978 (reduction of 0.53).

The area and delay measurements in Table 5 show that the new mapper, on average, reduces delay more than two times, compared to SIS while the area increases by only 10%. The results clearly show that using a combination of the two heuristics works better than applying each one independently.

The area can be further improved by optimizing phase assignment at the gate boundaries and sweeping equivalent gates in the mapped circuit.

FPGAs

A similar dynamic of area recovery using the proposed combination of two heuristics is observed in mapping for FPGAs.

Table 6 compares the mapping results using DAOmap [4] and the proposed flow in baseline mode implemented in MVSIS. Both tools were run on a 4 CPU 3.00GHz computer with 510Mb RAM under Linux. The benchmarks were pre-optimized in SIS using *script.algebraic* followed by decomposition into two-input gates using command *dmig* implemented in the RASP package [9]. To ensure identical starting logic structures, the same pre-optimized benchmark files used in [4][1] were used in this experiment.

Table 6. Comparison with DAOmap [4].

Example	DAOmap			MVSIS		
	Depth	LUTs	Time, s	Depth	LUTs	Time, s
alu4	6	1065	0.5	6	994	0.3
apex2	7	1352	0.6	7	1202	0.4
apex4	6	931	0.7	6	892	0.3
Bigkey	3	1245	0.6	3	797	0.4
Clma	13	5425	5.9	13	4429	1.7
Des	5	965	0.8	5	1020	0.5
Diffeq	10	817	0.6	10	854	0.4
Dsip	3	686	0.5	3	686	0.3
Elliptic	12	1965	2.0	12	2015	0.8
ex1010	7	3564	4.0	7	3265	1.1
ex5p	6	778	1.0	6	744	0.3
Frisc	16	1999	1.9	15	2011	0.9
misex3	6	980	0.8	6	955	0.3
Pdc	7	3222	4.6	8	2919	1.2
s298	13	1258	2.4	13	825	0.3
s38417	9	3815	3.8	9	3852	1.6
s38584	7	2987	27.0	7	2843	1.3
Seq	6	1188	0.8	6	1109	0.3
Spla	7	2734	4.0	7	2529	1.0
Tseng	10	706	0.6	10	758	0.3
Ratio	1.00	1.00	1.00	1.00	0.93	0.42

[1] Three sequential benchmarks from this set (clma, s38417, and s38584) could not be entered into MVSIS due to a limitation of the BLIF parser. These benchmarks were preprocessed by SIS to remove the latches. The resulting logic cones were used for both DAOmap and MVSIS.

Columns 2 and 5 give the number of logic levels of LUTs after technology mapping. These are equal in all but two cases. This supports the fact that both mappers perform delay-optimal mapping for the given logic structure. Differences may be explained by minor variations in manipulating the subject graph, such as AIG rebalancing performed by MVSIS.

Columns 3 and 6 show the number of LUTs after technology mapping. The difference between the results produced by the two mappers reflects the fact that they use different area recovery heuristics and possibly, that MVSIS performs area recovery in a topological order, while DAOmap uses a reverse topological order.

Columns 4 and 7 report the runtimes in seconds. These include the time needed to construct the subject graph and perform technology mapping with area recovery but not the time needed to read the input BLIF file. For smaller benchmarks, the differences in runtimes might be explained by the improvements to the basic data structures implemented in MVSIS. The increased runtime advantages of MVSIS on larger benchmarks may be due to better scalability and filtering heuristics employed by the MVSIS mapper.

In summary, Table 6 demonstrates that the proposed technology mapping environment is applicable to FPGA mapping. In fact, the FPGA mapping engine implemented in our environment is superior to a state-of-the-art FPGA mapper with 7% less LUTs using less than half the runtime on average.

5.4 Mapping using a load-dependent delay model

This case study was designed to demonstrate that the mappings computed in the proposed environment, which currently uses a simple load-independent delay model, largely preserve their quality when the delay is evaluated using a load-dependent delay model.

The load-dependent delay model of SIS was adopted. This assumes that the delay D of each pin p of a gate g is computed as follows: $D(p) = D_{ind}(p) + C_p * NFans(g)$, where $D_{ind}(p)$ is a load-independent delay of the pin, C_p is a constant, and $NFans(g)$ is the number of fanouts of gate g. The *genlib* gate libraries included in the distribution of SIS use this model.

Experimental Results

In this experiment, summarized in **Table 7,** we consider a randomly selected subset of large benchmarks. The benchmarks are pre-optimized using script *mvsis.rugged* and mapped using three different methods: (1) SIS, (2) MVSIS in the baseline mode, and (3) MVSIS with supergates. In all cases, we report the resulting area and delay as measured by the SIS command *print_map_stats* and *print_delay* using the gate library *mcnc.genlib* from the standard SIS distribution.

Table 7. SIS mapping vs. new mapping algorithms for load-dependent delay model.

Example	SIS		Baseline		Supergates	
	Area	Delay	Area	Delay	Area	Delay
b14	13988	259.0	10503	148.7	10791	118.0
b17	64089	321.1	48444	238.3	48836	170.7
c1355	812	41.1	1233	31.5	1119	35.3
c3540	2283	84.2	1885	58.7	2420	55.4
c6288	6104	253.0	6075	141.3	7540	116.2
pj1	30713	146.1	24295	96.3	26377	87.0
pj2	5548	95.0	4847	71.3	5483	71.4
Ratio	1.00	1.00	0.93	0.68	1.01	0.61

The average ratios of improvements of the MVSIS mapper versus the SIS mapper are shown in the bottom row of the table. The (load-independent) MVSIS mapper in baseline mode produces 7% smaller area and 32% shorter delay, compared to SIS, even when the final delay is measured using the load-dependent delay model. For mapping with supergates, the MVSIS mapper produced, on average, 1% worse area and 39% better delay.

It should be emphasized that the load-independent mapping result was only **evaluated** with the load-dependent delay model, while SIS used this model during the mapping process also. However, we propose an iteration process, where the load from the previous iteration is used to approximate the load of the current iteration. The initial iteration would use the load-independent delay model. Iteration should improve the delays considerably.

We believe that using a simple delay model at the beginning of the mapping process quickly finds a starting point, which is beneficial for iterative improvement by the transformations targeting a complex delay model, e.g. use of 2D tables to specify delays and slews for pin-to-pin rising/falling signals. In future work, we will explore the potential of the iterative process when applied to such accurate delay models.

6 Conclusions and future work

We described an integrated environment for technology mapping and demonstrated its flexibility by using different optimization criteria such as power optimization, placement-awareness, and efficient area recovery. Although some of the metrics used in these studies were simplistic, we believe they reflect general tendencies and show the potential of developing more accurate metrics for optimization in the new environment.

Future work will include integrating industrial delay models in standard cell mapping, extending FPGA mapping to apply to large input LUTs, performing placement-aware mapping using a realistic placer, and generalizing the mapping flow to work for sequential circuits as discussed in [23].

Acknowledgement

This research was supported in part by NSF contract, CCR-0312676, by the MARCO Focus Center for Circuit System Solution under contract 2003-CT-888 and by the California Micro program with our industrial sponsors, Fujitsu, Intel, Magma, and Synplicity.

The authors are grateful to Jason Cong and Deming Chen for providing the set of pre-optimized benchmarks used in [4], which allowed for a comparison with DAOmap reported in Table 6.

References

[1] R. Brayton, G. Hachtel, A. Sangiovanni-Vincentelli, "Multilevel logic synthesis", *Proc. IEEE*, Vol. 78, Feb.1990, pp. 264–300.

[2] R. K. Brayton and C. McMullen, "The decomposition and factorization of Boolean expressions," *Proc. ISCAS '82*, pp. 29-54.

[3] C.-W. Chang, C.-K. Cheng, P. Suaris, M. Marek-Sadowska, "Fast post-placement wiring using easily detectable functional symmetries", *Proc. DAC '00*, pp. 286-289.

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

[5] 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.

[6] J. Cong, Y. Ding, “On area/delay trade-off in LUT-based FPGA technology mapping”, *IEEE Trans. VLSI*, Vol. 2(2), June 1994, pp. 137-148.

[7] 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.

[8] J. Cong and C. Wu, “Optimal FPGA mapping and retiming with efficient initial state computation”, *IEEE Trans. CAD*, vol. 18(11), Nov. 1999, pp. 1595-1607.

[9] J. Cong et al, *RASP: FPGA/CPLD Technology Mapping and Synthesis Package*. http://ballade.cs.ucla.edu/software_release/rasp/htdocs/

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

[11] S. Hassoun and T. Sasao, eds., *Logic synthesis and verification*, Kluwer 2002, Chapter 5, “Technology mapping”, pp. 115-140.

[12] U. Hinsberger and R. Kolla, “Boolean matching for large libraries,” *Proc. DAC `98*, pp 206-211.

[13] D.-J. Jongeneel, R. Otten, Y. Watanabe, R. K. Brayton, “Area and search space control for technology mapping,” *Proc. DAC ’00*, pp. 86-91.

[14] 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.

[15] K. Keutzer, “DAGON: Technology binding and local optimizations by DAG matching”, *Proc. DAC ’87*, pp. 617-623.

[16] V. N. Kravets and P. Kudva, “Implicit enumeration of structural changes in circuit optimization”, *Proc. DAC ’04*, pp. 438-441.

[17] Y. Kukimoto, R. K. Brayton, P. Sawkar, “Delay-optimal technology mapping by DAG covering”, *Proc. DAC ’98*, pp. 348-351.

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

[19] 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.

[20] A. Mishchenko, X. Wang, T. Kam, “A new enhanced constructive decomposition and mapping algorithm”, *Proc. DAC ‘03*, pp.143-147..

[21] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, “Reducing structural bias in technology mapping”, *Proc. IWLS ‘05*.

[22] MVSIS Group. *MVSIS: Multi-Valued Logic Synthesis System.* UC Berkeley. http://www-cad.eecs.berkeley.edu/mvsis/

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

[24] M. Pedram and N. Bhat. “Layout driven technology mapping,” *Proc. DAC ‘91*, pp. 99-105.

[25] E. Sentovich, et al. “SIS: A system for sequential circuit synthesis”, Tech. Rep. UCB/ERI, M92/41, ERL, Dept. of EECS, Univ. of California, Berkeley, 1992.

[26] R. S. Shelar and S. S. Sapatnekar, "A predictive distributed congestion and its application to technology mapping," *Proc. ISPD ’04*, pp. 210 - 217.

[27] M. Teslenko, E. Dubrova, “Hermes: LUT FPGA technology mapping algorithm for area minimization with optimum depth”, *Proc. ICCAD ’04*, pp. 748-751.