Wavefront Technology Mapping

Leon Stok
IBM TJ Watson Research Center
Yorktown Heights, NY
stokl@watson.ibm.com

Mahesh A. Iyer
Synopsys Inc.
Mountain View, CA
miyer@synopsys.com

Andrew J. Sullivan
IBM Server Development
Fishkill, NY
sullia@us.ibm.com

Abstract

The wavefront technology mapping algorithm leads to a very simple and efficient implementation that elegantly decouples pattern matching and covering but circumvents that patterns have to be stored for the entire network simultaneously. This coupled with dynamic decomposition enables trade-off of many more alternatives than in conventional mapping algorithms.

The wavefront algorithm maps optimally for minimal delay on DAGs when a gain based delay model is used. It is optimal with respect to the arrival times on each path in the network. A special timing mode for multi-source nets allows minimization of other (non-delay) metrics as a secondary objective while maintaining delay optimality.

1 Introduction

In high-performance circuit design, meeting the delay targets in the control logic imposes a major challenge. Control logic usually undergoes changes till very late in the design cycle. Control logic is quite often not regular enough to lead to an intuitive data-flow type implementation. Logic synthesis is therefore necessary to meet the project schedules for these high-performance designs, and ensure a fast and correct implementation of the irregular subcircuits.

Most state-of-the-art logic synthesis systems [2, 11, 13] consist of three phases. The technology mapping phase usually follows the technology independent optimization and precedes the timing correction phase. This decoupling of phases has a major impact on the structure of technology mapped logic, and its delay and area characteristics. Minimizing delay in the technology mapping phase is an important goal in the aforementioned designs.

Conventional technology mapping can be described as a 3-step procedure. First, the technology-independent circuit is decomposed in terms of some primitives to have some well-defined logic structure to aid the technology mapping process. This phase is typically referred to as the circuit decomposition phase. Second, a pattern matcher performs analysis on the circuit and the library, either structurally or functionally, and determines a set of matches for all nodes in the circuit. The third and final phase consists of identifying the best set of matches (based on some cost functions) for the circuit such that every node in the circuit is covered at least once and functionality is maintained. The final set of matches that cover all the nodes in the circuit are used to describe the circuit in terms of the target technology library cells.

Optimal technology mapping for area on tree based subject graphs was described in [6]. Drawback of this algorithm is that it is optimal only for trees. Most practical circuits consist of directed acyclic graphs (DAGs) and require a non-trivial decomposition of the DAGs into trees. In [11] this was extended to include a delay objective using a binning approach for the delays. In this paper, we will present a very simple and practical algorithm that produces delay-optimal results for DAGs.

Technology mapping works mostly on a fixed subject graph. Therefore, the result is (very) dependent on the preceding decomposition of the network. In [9, 10] logic decomposition is combined with the mapping phase itself, to get around the problem of the disconnected phases. This algorithm runs on tree leaf DAGs. Despite its efficient implementation, exhaustive embedding of decompositions leads to a large increase in the subject graph, not practical for the sizes of circuits we are interested in. The wavefront mapper enables the embedding of multiple decompositions on-the-fly. This so-called *dynamic decomposition* significantly reduces the number of decompositions needed. Since the arrival times of all signals up till the inputs of the dynamic decomposed logic are known, we can limit the number of decompositions significantly (when compared to the exhaustive decomposition of [9]), and still obtain near delay-optimal results.

A similar observation on delay optimality as in theorem 1 is made in [8] for simplistic load-independent delay models. In that paper only a graph matching algorithm is described to generate the matches. If only a graph matcher would be used in the wavefront algorithm, one can set the wavefront width equal to the depth of the deepest pattern graph and guarantee delay optimality.

Section 2 describes the timing model we are using in this paper. In section 3 the core wavefront algorithm is outlined, along with some proofs of optimality. Section 4 describes how other objectives can be taken into account and explains

a few other extensions to the basic algorithm. Finally, in section 5 results are shown on various designs examples along with the impact of various parameters on the performance of the technology mapping algorithm.

2 Static timing analysis and Delay Model

Since the wavefront algorithm does a timing driven technology mapping the timing information it uses is an important part of the algorithm. Therefore, this section summarizes our timing analysis and points out the differences with conventional static timing analysis tools.

Static timing analysis [5, 4] is performed on the directed graph of the network. The vertices of the graph are the points at which events can occur (e.g., signals can arrive) and are referred to as timing points. The timing points include boundary pins and pins on logic gates in the network.

Each timing point in the network has an associated *arrival time* $t_a(p)$ and an associated *required time* $t_r(p)$. Arrival times at the primary inputs are given. Timing analysis propagates these arrival times forward through the network and calculates arrival times at all other timing points. In *late mode* timing analysis, at each timing point the latest arriving value is being propagated. Similarly, required times are derived from the required times at the primary outputs. They are propagated backwards through the network. The *slack* $s(p)$ of each timing point is now defined by $s(p) = t_r(p) - t_a(p)$. The worst slack $s_w(p)$ is defined as the most negative slack on any timing point in the network. Note that a critical path can be defined as a path from primary input to primary output on which all timing points will have the same worst slack $s_w(p)$.

Our wavefront algorithm deviates in its uses of standard timing analysis in two ways. Traditionally, delay rules for standard cells have been provided based on the actual size of the gate and are dependent on its output load.

However, if we assume rules that are continuously parameterizable, a quantity directly related to delay such as normalized $gain = C_l/C_{in}$ (where C_l is the capacitive load and C_{in} is the input capacitance of the gate) can be used [7] [1], [14][3]. The gain is not directly dependent on the size of the gate, which increases linearly with C_{in} or the load, but merely depends on the ratio of both. This leads to a simple delay equation $d = k_1 \times gain + k_2$, where k_2 represents the intrinsic delay of the gate. In these *gain-based* delay equations the delay is independent of the load of the cells, and solely dependent on the gain.

More realistic delay equations also include the slew and $beta$ (rise-to-fall ratio) and lead to a set of delay (d) equations of the following form [12]: $d = f(input\_slew, gain, beta)$ and $output\_slew = g(input\_slew, gain, beta)$. Input slew will be propagated by the timer. Beta is given for a particular cell. Thus again, if the value for the gain is known, the delay of each cell is determined. Experiments [7] [1] have shown that these gain based delay models can be as accurate as load-based delay models. During the technology mapping we can either create matches for 'ideal' gain values, or we can explore alternatives by creating matches with different values.

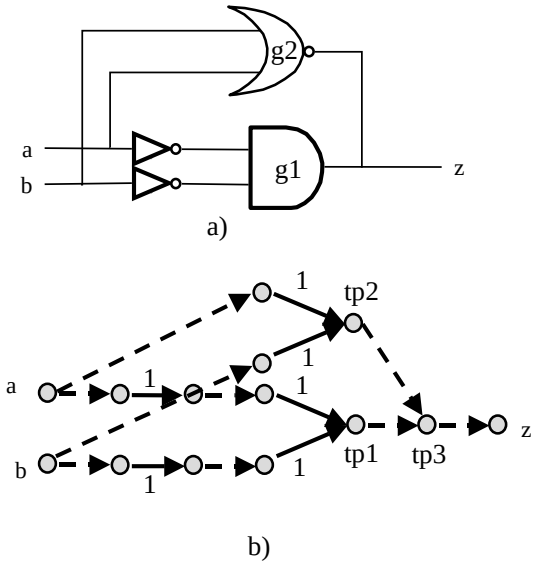

Figure 1: a) Circuit DAG b) Timing Graph

Second, a new late mode timing propagation mode is defined for a special kind of logically equivalent multi-source nets. Note, these are different from wired-or or wired-and logic. In our case the nets embed multiple implementations of equivalent logic in the network. In the final implementation only one of the drivers is needed. From now on we will call these *multi-source* nets for short.

In regular timing analysis, at timing points with multiple incoming arcs the latest arrival time is propagated. In the timing points related to the multi-source nets the earliest arrival time will be propagated.

In figure 1.a) an example is shown to illustrate the timing analysis. The NOR-gate is an alternate implementation for the AND and INVERTERs. Figure 1.b) shows the timing graph where the nodes correspond to the pins of the circuit in 1.a). The dashed edges correspond to the net delay arcs, the solid edges to the gate delay arcs. All net delays are assumed zero and gate delays one. The timing point $tp1$ at the output of gate $g1$ is a regular point and the latest arrival 2 is propagated. However, at the multi-source timing point $tp3$, the value coming from $tp2$ (delay through gate $g2$ is one) is propagated as the earliest, late-mode arrival time.

During the wavefront algorithm matches are implemented directly in the net-list. This will result in a large increase of fanouts at the inputs of the patterns and will result in multi-source nets at the outputs of the patterns.

Algorithm I. Wavefront map

```
Procedure wavefront_map(design, wave_width, max_levels) {
    /* wave_width = width of the wavefront; */
    Levelize the design;
    max_levels = maximum number of levels in the design;
    head_level = 1;
    while (head_level ≤ max_levels) {
      head_nets = list of all nets on head_level;
      tail_level = head_level - wave_width;
      if (tail_level ≤ 0) {
        tail_level = 0;
      }
      /* Generate matches on the head of the wavefront */
      foreach net, n in head_nets {
        generate_and_implement(design, n, tail_level);
      }
      /* Perform covering on the tail of the wavefront */
      if (tail_level ≥ 0) {
        cover_nets(design, tail_level);
      }
      increment head_level;
    }
    /* Move the tail, if it has not yet moved */
    if (tail_level ≤ 0) {
      tail_level = 1;
    }
    /* Perform covering on the remaining uncovered levels */
    while (tail_level ≤ max_levels) {
      cover_nets(design, tail_level);
      increment tail_level;
    }
}
```

However, since our delay model is gain-based the increase in fanout will not invalidate our timing results. Our special multi-source timing propagation algorithm enables the wavefront algorithm to trade-off the various matches as will be discussed in the following section.

3 Wavefront Algorithm

The *wavefront* is a subgraph of the DAG, such that every path from input to output goes through the subgraph. The subcircuit isolated by the wavefront is bounded by the *head* and the *tail* of the wavefront. The head of the wavefront is the boundary closer to the primary outputs (POs) and the tail of the wavefront is the boundary closer to the primary inputs (PIs) of the circuit. If, in figure 2.a), the vertical line with label 1 is the head and the line with label 0 is the tail, the subgraph containing the inverters form the wavefront. In addition to decoupling the match generation and covering problems, the wavefront allows the match generation to work only on a subcircuit, thereby minimizing the number of matches stored at any time. Also, matches are allowed to be generated and maintained dynamically, as opposed to generating all the matches for the entire circuit a-priori.

3.1 Basic Algorithm

Algorithm I outlines the conceptual ideas of the wavefront algorithm. The wavefront algorithm assumes that the circuit is levelized from input to output. The head and tail

Algorithm II.Net Covering

```
Procedure cover_nets(design, tail_level) {
    foreach net, m, in tail_level {
      - Perform static timing analysis and compute the driver
      pin that has the fastest arrival time;
      - fastest_pin = driver pin that has the fastest arrival time;
      - fastest_cell = driver cell for fastest_pin;
      - Disconnect all driver cells on m, except fastest_cell, and
      perform a cleanup operation on their exclusive input cones
      of logic. The technology-independent cell and its exclusive
      input cone of logic are also implicitly cleaned up;
    }
}
```

Algorithm III. Match Implementation

```
Procedure generate_and_implement(design,net,tail_level) {
    /* Match generation */
    match_list = get_all_pattern_matches(design, net, tail_level);
    /* Match implementation as multiple-source net drivers */
    implement_matches(design, match_list, net);
}

Procedure get_all_pattern_matches(design, net, tail_level) {
    - Using the structural, boolean and PLA matchers, generate
    all the pattern matches in the design for net, such that
    the search for pattern matches starts at net and ends
    whenever a net whose level ≤ tail_level is encountered;
    - For each match, store the technology library cell and the
    corresponding nets in design that need to be connected to
    the inputs of the technology cell. Let this data structure
    be match_struct;
    - match_list = list of match_structs for all matches found;
    - return match_list;
}

Procedure implement_matches(design, match_list, net) {
    foreach match_struct in match_list {
      - technology_cell = match cell from match_struct;
      - input_list = list of input nets from match_struct;
      - Make technology_cell, a driver of net in design;
      - Connect all inputs of technology_cell to the
      corresponding nets of design in input_list;
    }
}
```

of the wavefront define a window, such that for a particular step, match generation happens at the head and covering happens at the tail.

Initially, both the head and the tail of the wavefront start at level 0. (The PIs of the circuit are considered as level 0 nets.) The head of the wavefront advances one step and match generation and implementation is done for all nets on this level. Matches are generated using PLA, Boolean and Structural matchers. All pattern matchers have one thing in common; given (a) net(s) in the circuit, they generate a set of technology cell matches for the subcircuit in the wavefront driving the(se) net(s).

Unlike conventional technology mapping, matches are not limited to fanout-free regions; i.e. the match generation search process performs its search across fanout. How-

ever, the subcircuit for the match generation search process is isolated by the wavefront; i.e. the match generation search process starts at a net(s) on the head of the wavefront and stops as soon as it encounters nets on the tail of the wavefront or a net on a level below the tail of the wavefront. As matches are generated for a node, they are implemented in the underlying net-list as drivers of a multi-source net. The multiple drivers on a net include the technology-independent cell and all the technology cell matches found for that node. The match implementation process also connects the input pins of the technology cells to the appropriate nets in the circuit.

Implementing certain pattern matches also requires some amount of logic duplication. A key advantage of implementing matches in the net list as multi-source nets is that this logic duplication is achieved implicitly. (See the example in figure 2). The head of the wavefront keeps advancing one step at a time until matches have been generated for all nets on the highest level of the circuit.

The tail does not start moving until the head has moved for a number of steps equal to the width of the wavefront. Covering for selecting the best match occurs for all nets on the tail of the wavefront. The covering implicitly performs static timing analysis to determine the match with the best arrival time. Implementing the matches in the net list (as multi-source net drivers) allows the algorithm to evaluate all the matches in the context of the design in which they are instantiated. Under the gain-based delay model, the arrival time computed for a match cell output is also guaranteed to be accurate because the inputs nodes of the cell have already been mapped and the cell's delay does not depend on the load driven by the cell. Only this match is retained as a driver to the net and all other matches on the net are deemed sub-optimal. The sub-optimal match cells and their exclusive input cones are removed from the net list. The original technology-independent driver cell on the net is also destroyed. The tail of the wavefront keeps advancing one step at a time until all nets on the highest level of the circuit have been covered.

In figure 2.a) the wavefront algorithm is illustrated by an example. Assume the width of the wavefront equals 4. Assume the technology library contains only INV, NOR, AOIs and XNOR gates. All technology cells and their embedding in the network are shown in dashed lines. Head and tail are shown by dashed vertical lines. When the head (dashed vertical line) reaches level 1, the technology inverters are added, at level 2 the NOR. At level 3 no matches are found in our inverting-only example library. At level 4, the inverter, NOR, AOI and XNOR are inserted. Now the tail starts moving. At level 1 the inverters are selected, at level 2 the NOR. The NOR now implements the function for output y. Finally at level 4 we find the XNOR and disconnect the rest of the network. The final mapped circuit is shown in 2.b). Note the implicit cloning that happened in selecting the XNOR for output z and NOR for output y.

If $w = 3$ the XNOR match would not be found and the result would be as shown in 2.c).

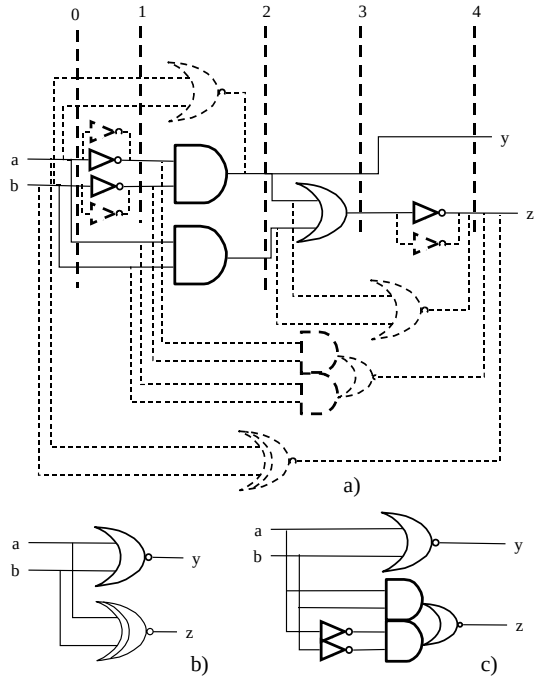

Figure 2: a) Subject circuit b) Mapped circuit $w = 4$ c) Mapped Circuit $w = 3$

3.2 Load-independent delay optimality for DAGs.

In this section, we show that under some conditions the wavefront algorithm will produce optimal-delay mapping under the gain-based delay model. Assume that *w* is the width of the wavefront and *D* is the depth of the circuit. Also assume that for every node, all pattern matches can be found in the subcircuit that the pattern matchers work on.

Theorem 1 *If w = D, then the mapped net list obtained using the wavefront algorithm is optimal in delay, under the gain-based delay model.*

Proof: Since *w = D*, the pattern matchers can search for matches for a node in the entire fanin cone of the node. Thus, all pattern matches can be found for all nodes in the circuit. Under the gain-based delay model, the arrival time on the output of a cell depends only on the arrival times of its inputs and the delay of the cell. The covering step of the wavefront algorithm proceeds in a levelized fashion from inputs to outputs. Hence, when covering a node, the entire fanin cone of the node is guaranteed to be mapped. When covering a node on level 1, the match that produces

the fastest arrival time will be selected as the best match for that node. For nodes on higher levels, by induction, it follows that the arrival times on the inputs of all matches on a node are guaranteed to be the fastest arrival times for those inputs. The technology cell producing the best arrival time at a node will be selected as the best match for that node. Again, by induction, all technology cells on all paths to a PO will be such that the fastest arrival time is produced on that PO. Under the gain-based delay model, this ensures that the mapped circuit is optimal in delay. □

In the general case, a width smaller than the depth of the circuit can be used. Practically, it needs to be set to the depth of the largest decomposition of the library cells that the matchers can find. This is a function of the cells available in the library and the power of the matchers. However, as will be shown in section 5 the width can usually be kept to a relatively small number resulting in significant memory savings to store the patterns, and significant saving in the size of the network to be timed.

4 Extensions to the Algorithm

4.1 Mapping for different fall/rise times

In most timing models for real libraries, different values are maintained for the rising and falling delays. This can be easily accommodated in the wavefront algorithm. Instead of a single match, the fastest match for both the falling and rising delay are maintained. A simple modification to the **cover_nets** procedure ensures that the two patterns remain connected to the net. The timing propagation is modified to allow for a separate earliest late-mode falling and rising time to be propagated through different paths in the graph. In a second pass over the circuit from output to inputs, we can select the driving cell that minimizes the rising and falling delay, the worst of the two or a combination of both.

4.2 Optimizing for other cost functions

In most designs, one is not interested in minimizing the delay through the circuit but in maximizing the *slack*. Since the wavefront algorithm optimizes arrival times at all points in the network it therefore maximizes the slack at all points. However, often designers are satisfied with zero slack and maximizing slack in regions where it is positive (non-critical regions) is not necessary. In this case the arrival of a non-critical signal can be slowed down (by the amount of slack). Critical regions can only be reliably determined if the entire circuit has technology cells. The required time prediction in the unmapped circuit to the right of the head of the wavefront is too unreliable and a two pass approach is chosen here as well.

Instead of keeping only the fastest match, one or more other matches are kept in the first pass. For example, the smallest match. In a second pass, from outputs to inputs, the slower (smaller) patterns can be chosen in the off-critical regions to minimize area. This freedom can be used to optimize other cost functions in a second pass without influencing the worst case slack.

4.3 Dynamic Decomposition

The model as described in section 2 allows for multiple sources on each net. The additional multi-sources can either be technology matches or alternate decompositions of the subject graph. Many different decompositions similarly to [9] can be embedded in the subject network.

However, it is possible to delay some amount of decomposition by preserving large input primitives until such gates are encountered by the head of the wavefront. At this point the arrival times at the inputs to these gates are now defined by the combination of covered logic and the propagation model defined in section 2. These arrival times will reflect the optimal delay values up to this level of logic. Given these arrival times any large primitives at the head of the wavefront are first decomposed into a network of 2-input NAND gates such that the number of logic levels between late arriving signals and the output net of the original primitive is minimized. A simple yet powerful decomposition scheme is the repeated extraction of 2-Input NAND gates corresponding to the pairs of input signals with fastest arrival times. This continues until the original primitive has been completely decomposed into a tree of 2-Input NAND gates. A series of sub-levelization lists can be maintained in order to add newly created nets to the wavefront levelization tables. After the levelization tables are updated, the match generation and tail-based covering proceed in their typical fashion.

5 Experiments and Results

To do a set of experiments a prototype of the wavefront algorithm was implemented. In the first set of experiments the effect of the wavefront size is studied. In table 1 the effect of varying wavefront sizes from 1 to 15 are shown for an industrial library. More patterns (column 3 and 7) are found at larger wavefront sizes. However, for widths of more than 4 the effect on delay is neglectable. Runtimes grow approximately linearly with the number of patterns.

In the second set we compare the wavefront mapping algorithm against a state-of-the-art industrial technology mapping tool. Unfortunately, we have no industrial technology libraries with timing models as described in section 2 available for university tools and could not compare against these.

Only the technology mapping parts are compared, i.e. the same net list is give to both programs right before technology mapping and results are measured directly after a fully mapped network is obtained. Table 2 shows that our wavefront algorithm consistently produces superior delay results. Sometimes there is a large gain in delay for little area (C6288), sometimes there is little gain in delay for

Table 1: Impact of wavefront width

Des	w	pat	delay	CPU	Des	w	pat	delay	CPU
C432	1	172	7.57	4.88	C2670	1	735	7.10	16.62
C432	2	217	6.20	5.74	C2670	2	952	6.66	21.98
C432	4	247	6.09	6.11	C2670	4	1085	6.47	22.37
C432	10	289	6.09	6.26	C2670	10	1155	6.47	26.63
C432	15	299	6.09	7.14	C2670	15	1164	6.42	26.13
C499	1	566	6.91	6.22	C3540	1	1192	10.69	29.63
C499	2	629	6.61	15.36	C3540	2	1586	10.29	37.98
C499	4	750	6.30	15.76	C3540	4	2032	9.95	47.88
C499	10	794	6.28	17.19	C3540	10	2149	9.91	52.75
C499	15	853	6.22	20.33	C3540	15	2164	9.91	54.14
C880	1	429	7.37	10.59	C5315	1	1807	9.45	54.23
C880	2	522	7.01	10.43	C5315	2	2109	9.13	48.31
C880	4	603	6.80	11.49	C5315	4	2562	8.48	53.73
C880	10	640	6.73	12.35	C5315	10	2744	8.48	61.74
C880	15	666	6.73	13.88	C5315	15	2792	8.48	58.46
C1355	1	548	6.08	13.44	C6288	1	3227	31.23	122.61
C1355	2	598	5.78	14.26	C6288	2	3985	30.00	132.84
C1355	4	725	5.59	14.97	C6288	4	4256	29.14	133.1
C1355	10	821	5.52	17.00	C6288	10	4355	29.09	139.27
C1355	15	853	5.52	19.53	C6288	15	4398	29.09	139.33
C1908	1	541	9.04	13.41	C7552	1	2310	14.72	83.6
C1908	2	636	8.55	15.72	C7552	2	2676	14.48	79.72
C1908	4	760	8.17	16.91	C7552	4	3260	14.19	84.56
C1908	10	793	8.17	18.17	C7552	10	3457	14.19	85.59
C1908	15	800	8.17	17.20	C7552	15	3502	14.14	93.73

Table 2: WaveFront mapper and Industrial mapper.

Circuit	*delay*	*WFdelay*	*area*	*WFarea*
C432	8.62	6.09	184	247
C499	7.81	6.30	665	750
C880	8.30	6.80	460	603
C1355	6.61	5.59	648	725
C1908	9.22	8.17	614	760
C2670	6.83	6.47	669	1085
C3540	10.45	9.95	1268	2032
C5315	8.51	8.48	1754	2562
C6288	44.2	29.14	4104	4256
C7552	19.67	14.19	2751	3260

larger area (C5315). On the average the wavefront algorithm produces 22% faster circuits with about 24% area overhead.

6 Conclusions

This paper described the wavefront technology mapping. This mapping method combines delay optimality for DAGs under a gain-based delay model, with implicit cloning and efficient pattern storage and timing calculation. For practical wavefront sizes (i.e. 4) near-delay optimal results are obtained. Delays are 22% better than conventional technology mappers. The multi-source propagation of the timing allows for secondary cost functions to be handled easily.

Acknowledgments

We thank Nate Hieter and Alex Suess for implementing the incremental timing analysis package used by the wavefront algorithm. Many thanks to Daniel Brand, Benjamin Chen, Robert Damiano, Ramsey Haddad, Kevin Harer, Prabhakar Kudva, David Kung, Tony Ma, Lakshmi Reddy, Richard Rudell, Narendra Shenoy, and Lukas van Ginneken for helping us create the environment and system in which we could easily implement the ideas presented in this paper and conduct the experiments.

References

[1] F. Beeftink, P.N.Kudva, D.S.Kung, and L. Stok. Gate size selection for standard cell libraries. In *Proc of the Int. Conf. on Computer Aided Design*, page ??, Nov 1998.

[2] E. Detjens, R. Rudell, G. Gannot, A. Wang, and A. Sangiovanni-Vincentelli. Technology mapping in mis. In *Proc of the Int. Conf on Computer Aided Design*, pages 116–119, Nov 1987.

[3] J. Grodstein, E. Lehman, H. Harkness, B. Grundmann, and Y. Watanabe. A delay model for logic synthesis of continuously sized networks. In *Proc of the Int. Conf. on Computer Aided Design*, pages 458–462, Nov 1995.

[4] R. HitchcockSr. Timing verification and the timing analysis program. In *ACM IEEE Nineteenth Design Automation Conference*, pages 594–604, Las Vegas, June 1982.

[5] R. HitchcockSr., G. Smith, and D. Cheng. Timing analysis of computer hardware. *IBM J. Res. Develop.*, 26(1), January 1982.

[6] K. Keutzer. Dagon: Technology binding and local optimization by dag matching. In *Proc of the 24th Design Automation Conference*, pages 341–347, June 1987.

[7] P. Kudva. Continuous optimizations in synthesis: The discretization problem. In *Proc of Int. Workshop on Logic Synthesis*, pages 408–419, June 1998.

[8] Y. Kukimoto, R. K. Brayton, and P. Sawkar. Delay-optimal technology mapping by dag covering. In *Proceedings of the DAC 1998*, pages 348–351, 1998.

[9] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness. Logic decomposition during technology mapping. *IEEE Trans on CAD*, 16(8):813–834, August 1997.

[10] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness. Logic decomposition during technology mapping. In *Proc of the Int. Conf. on Computer Aided Design*, pages 264–271, Nov 1995.

[11] R. Rudell. Logic synthesis for vlsi design. Technical report, University of California, Berkeley, 1989.

[12] K. Shepard and et al. Design methodology for the s/390 parallel enterprise server g4 microprocessors. *IBM J. Res. Develop.*, 41(4/5):515–547, July/September 1997.

[13] L. Stok and et al. Booledozer logic synthesis for asics. *IBM J. Res. and Develop.*, Vol. 40(4):407–430, July 1996.

[14] I. Sutherland and R. Sproull. The theory of logical effort: Desiging for speed on the back of an envelope. In *Advanced Research in VLSI*, University of California at Santa Cruz, 1991.