Timing Optimization of Combinational Logic *

Kanwar Jit Singh Albert R. Wang Robert K. Brayton Alberto Sangiovanni-Vincentelli

Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, CA 94720.

Abstract

We present an algorithm for speeding up combinational logic with minimal area increase. A static timing analyzer is used to identify the critical paths. Then, a weighted min-cut algorithm is used to determine the subset of nodes to be resynthesized. This subset is selected so that the speed-up is achieved with minimal area increase. Resynthesis is done by selectively collapsing the logic along the critical paths, and then decomposing the collapsed nodes to minimize the critical delay. This process is iterated until either the timing requirements are satisfied or no further improvement can be made. The algorithm has been implemented and tested on many design examples with promising results.

1 Introduction

We view timing optimization as a three-phase process. In the first phase, the circuit is globally restructured to reduce the maximum level or the longest path in the circuit. This is usually accomplished in a technology independent fashion. For example, changing from a ripple-carry-adder to a carry-look-ahead adder or something in between is accomplished in this phase. The second phase is to speed up the circuit during the physical design process. Transistor sizing or timing driven placement of modules are examples of such optimizations. In the last phase, when we have an actual design, a more accurate timing analyzer is used to fine tune the circuit parameters. This paper considers only the first phase, viz. technology independent logic resynthesis.

In earlier works speed-up is achieved by making local changes in the topology. [5,3] reduce the delay by adding buffers and decomposing an *existing gate* into gates containing early and late arriving signals, with the latter being placed closer to the output. SOCRATES [1] uses a rule based system to improve the timing by local transformations. These may not lead to the global restructuring that we are trying to achieve. However the strength of this approach is that it fully exploits the features of the library and technology being used. We feel that our approach followed by a SOCRATES type local improvement step would yield good results.

We approach the problem of restructuring the logic with a global view. Our approach is similar to the "Circuit Re-synthesis" step of the Yorktown Silicon Compiler [6]. In both approaches, a sub-network is defined and a critical section to be transformed is identified. The significant contrasts are that our algorithm focuses only on logic resynthesis and operates on a technology independent representation of the circuit. The algorithms in [6] combine device sizing with logic manipulation and are somewhat specialized to domino CMOS designs. [6] uses transformations only between neighboring gates whereas we provide a parameter controlling the extent of the transformations.

Our algorithm uses a timing driven decomposition of the network into 2-input gates. This is important since the manner in which a complex gate is implemented changes its delay characteristics. We can use various models for computing delays. One of them is a fast technology mapping [4] of the two input gates into a standard cell library. This provides more accuracy to our timing estimates. We present a new algorithm, based on timing constraints, for decomposing a complex function into two input gates. This is done recursively from the bottom up, so that at each stage the input arrival times are fairly accurate.

Until now, the emphasis of MIS [2] has been on area optimization. The algorithms presented here have been integrated in MIS. The designer can now make an area/delay tradeoff.

2 The speed-up algorithm

The algorithm takes as input a network of 2-input NAND gates and inverters. Timing constraints are specified as the arrival times at the primary inputs and required times at the primary outputs. The algorithm manipulates the network to achieve speed up until the timing constraints are satisfied or no further decrease in the delay is possible. The output of the algorithm is also in terms of 2-input NAND gates and inverters.

Basic definitions are given in the next subsection and illustrated in Figure 1. The main loop of the algorithm is shown in Figure 2 and is explained in sections 2.2 through 2.5.

2.1 Basic definitions

Starting with the primary input arrival times, the arrival times for each of the signals is computed. Using the required times at the outputs, we compute the required times for all signals. The *slack* at a node s, is defined to be $R_s - A_s$ where A_s is its arrival time and R_s its required time.

An ϵ-network is defined as a sub-network in which all the signals have a slack within ϵ of the most negative slack.

For each node in the network we define a *d_critical_fanin_section* as the set of nodes that (1) are in the transitive fanin of the node, (2) are at most distance d (d levels of logic) away from the node, and (3) are part of the ϵ-network.

The operation *partial-collapse* on a node collapses all the nodes in the *d_critical_fanin_section* of the node. Internal nodes in this section that fanout elsewhere will be duplicated in this process.

2.2 Weight of the critical nodes

The function chosen to assign weights to the nodes in the ϵ-network is crucial since it determines the nodes that will be selected for speed-up. Associated with a node we define two components for its weight — an area penalty (W_a) and a potential

*This research was supported by DARPA, Micro, Intel, AMD, Phillips and SGS

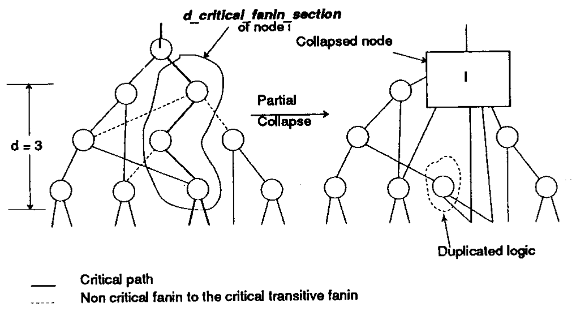

Figure 1: Definition of the terms

```
d = Distance up to which the critical fanins are collapsed.
speed_up( network, dist, $\epsilon$)
    do {
        delay_trace();
        generate( $\epsilon$-network);
        node_list = node_cutset( $\epsilon$-network);
        foreach node $\in$ node_list
            partial_collapse( node, d );
        foreach node $\in$ node_list
            speedup_node( node);
    } while ( delay decreases &&
            timing constraints not satisfied)
```

Figure 2: Outline of the resynthesis algorithm

for speed-up ($\mathcal{W}_t$). The area penalty is due to the duplication of some logic during the *partial-collapse* of a node. The potential for speed-up ($\mathcal{W}_t$) is determined by a heuristic that tries to identify the possibility of reducing the arrival time of the node after partial_collapse and resynthesis. These components are weighed depending on the area/delay tradeoff desired.

α = Coefficient controlling the area-delay tradeoff.

$$\mathcal{W} = \mathcal{W}_t + \alpha \times \mathcal{W}_a$$

where $\mathcal{W}_a$ = Number of literals in the duplicated logic,
$\mathcal{W}_t$ = Potential for speedup.

In order to define the "potential" for speed-up we look at the relation between the delays and arrival times in the existing network. Consider a node n, in the ϵ-network. For each node, i, that fans into the *d_critical_fanin_section* of n, we know its arrival time, A_i, and the delay, D_i, from it to the node n. Consider the following situations :

- The standard deviation (σ) of the vectors (A_i, D_i) is small. This implies a near balanced decomposition already exists in the transitive fanin of the node when the inputs arrive at similar times. Hence there is not much scope for improving the existing decomposition.
- Let $D = \beta \times A + \delta$ be the least square error straight line fitting the data points (A_i, D_i). A negative value of the slope β indicates that early arriving signals (small A_i) pass through a larger delay. This too suggests that the current decomposition is skewed in the right direction, reducing the "potential" for speedup.

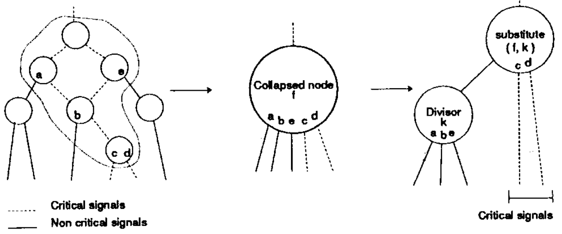

Figure 3: Basic idea of the timing decomposition

```
speedup_node( f )
    k = CHOOSE_BEST_TIMING_DIVISOR( f );
    if ( k != NULL ) {
        substitute ( f, k );
        speedup_node( k );
        /* Update the arrival time at inputs of f */
        delay_trace();
        speedup_node( f );
    } else AND_OR_decomp( f );
```

Figure 4: Routine to speed-up a node

We define $\mathcal{W}_t$ so that the selection of nodes with a large σ and a β close to +1 are favored. Thus

$$\mathcal{W}_t \propto \frac{|\arctan(\beta) - \pi/4|}{\sigma^2}$$

2.3 Minimum weighted cutset

After assigning the node weights, the maxflow-mincut algorithm is applied to generate a node cutset (separator set) of the ϵ-network. Speeding up all the nodes on the node cutset of the ϵ-network by a given amount γ ($\gamma < \epsilon$), would, in most cases, reduce the delay through the ϵ-network by at least γ. However, resynthesis may increase the delay through other parts of the circuit resulting in a speedup less than γ. The minimum weighted cutset provides us with a minimal area increase when the nodes are resynthesized.

2.4 Partial collapse

The network used for the cutset analysis is in the form of 2-input NAND gates. We collapse all the nodes in the *d_critical_fanin_section* of the node to generate a large node to decompose later. The choice of the distance d for the *partial-collapse* influences the algorithm and is explained in Section 3. Decomposition of the collapsed node is explained in the next section.

2.5 Timing decomposition

The general idea of timing-decomposition of a node is to place late arriving signals closer to the output. This is illustrated in Figure 3. Figure 4 describes the **speedup_node** algorithm for the decomposition of a node taking into account the arrival times at the inputs. It consists of two phases — (1) extracting divisors that reduce delay and (2) decomposing a node into a NAND-NAND tree.

Notice that the decomposition of any node is based on the correct updated arrival times of its input. This is a result of the

```
$\mathcal{K}$ = {level_0 kernels} $\cup$ {level_0 kernel_intersections}
**CHOOSE_BEST_TIMING_DIVISOR**( $f$ )
    $\mathcal{D} = \mathcal{K}$;        /* $\mathcal{D}$ is the set of divisors */
    $\rho = 0.1$;      /* Determined experimentally */
    for ( $n \in \mathcal{K}$)
        $f = q \cdot n + r$;   $\mathcal{D} = q \cup \mathcal{D}$;
    for ( $n \in \mathcal{D}$)
        $F_{in}$ = Signals that fan into $n$;
        $C_t = \rho \times \min_{i \in F_{in}} A_i + 1 - \rho \times \max_{i \in F_{in}} A_i$;
        $C_a$ = Literals saved if $n$ is extracted.
        $C(\ n\ ) = C_t - \alpha \times C_a$;
    return $j$ s.t. $C(\ j\ )$ is minimum;
```

Figure 5: Selection of divisors for timing resynthesis

bottom-up approach we adopt for the decomposition. Note that in Figure 4, we call **speedup_node** for node k before calling it for the new f resulting from substituting k into f. Thus during the decomposition of a node, we decompose first the parts that will eventually be closer to the inputs (e.g. the divisors which are extracted) and update their arrival times. This allows the decomposition to adjust dynamically to the updated delay of the extracted divisor which is now an input to the new node, f.

2.5.1 Kernel based decomposition

After a *partial_collapse* of the node, the notion of a critical path no longer exists since the critical path depends on the decomposition of the node. With this in mind, the objective is, given the arrival times at the inputs of a node, to decompose it in a manner to reduce the arrival time at its output. In decomposing a node to reduce its delay, we want to preserve the area savings that results from extracting good divisors. The search for divisors is restricted to a set of kernels $\mathcal{K}$, which for efficiency reasons consists of **level_0 kernels** and **level_0 kernel intersections**. In addition to the nodes in $\mathcal{K}$ we also consider their co-divisors while choosing the best divisor to extract.

The weight of a divisor is a linear sum of an area-component and a timing component. The area component, C_a, reflects the literals saved if the divisor were extracted. The timing component, C_t, is designed to prefer the divisor with the smallest arrival time for its latest arriving input. In the case of a tie the divisor with the largest spread of input arrival times is preferred. The weighting procedure for divisors is given in Figure 5.

2.5.2 AND-OR decomposition

After all divisors containing early arriving signals are exhausted, a timing-driven NAND-NAND decomposition is carried out. The routine AND_OR_decomp (Figure 6) decomposes a function $\mathcal{F}$, given as a sum-of-products, into a NAND-NAND tree. Each cube of $\mathcal{F}$ is decomposed into a tree of 2-input AND gates. A delay trace updates the arrival times at the output, x_i, of the AND-trees representing the cubes $c_i \in \mathcal{F}$. The cube $\prod \overline{x_i}$ represents the function $\overline{\mathcal{F}}$ and can be decomposed by the routine AND_decomp since the arrival times at its inputs are now known.

The AND_decomp routine (Figure 7) decomposes a cube into a 2-input AND tree so as to reduce the arrival time at the cube output. Since the decomposition creates a tree this procedure guarantees that the resulting 2-input AND decomposition has a minimum arrival time. An example of the decomposition of a 4 input cube is illustrated in Figure 8.

```
$\mathcal{F}$ is a multi-cube function.
**AND_OR_decomp**( $\mathcal{F}$ )
    foreach cube $c_i \in \mathcal{F}$
        **AND_decomp**( $c_i$ );
    delay_trace();
    $\overline{\mathcal{F}} = \prod_i \overline{x_i}$; /* $x_i$ represents the cube $c_i$ */
    **AND_decomp**( $\overline{\mathcal{F}}$ );
```

Figure 6: The AND_OR decomposition routine

```
$\mathcal{F}$ is a cube.
**AND_decomp**( $\mathcal{F}$ )
    if ( | $\mathcal{F}$ | > 2 ){
        $l_1$ = Earliest arriving input of $\mathcal{F}$;
        $l_2$ = Next earliest arriving input ;
        $c = l_1 \cdot l_2$;
        substitute( $\mathcal{F}$, c);
        delay_trace();
        **AND_decomp**( $\mathcal{F}$ );
    }
```

Figure 7: The AND_decomp routine to decompose a cube

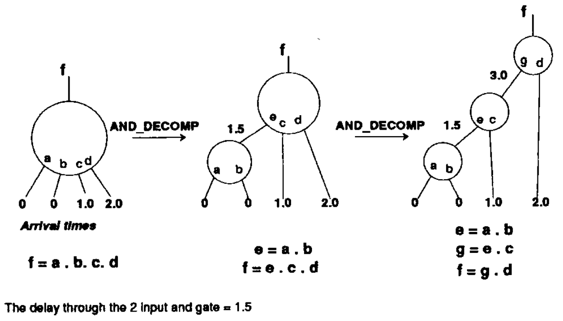

Figure 8: Example of a cube decomposed by AND_decomp

3 Controlling the algorithm

Several parameters govern the run time and the quality of the results. We discuss the influence of each of these in this section.

ϵ specifies the size of the ϵ-network. Using a large ϵ might result in selecting nodes for speed-up from a region where speeding up does not reduce the critical delay. Thus, area is wasted. Selecting an ϵ too small results in a slow algorithm.

d is the depth of the *d_critical_fanin_section*. A large d is useful in making relatively large changes in the delay since the larger nodes provide greater flexibility in restructuring the logic. However, due to the time spent in collapsing and the large number of divisors available for large nodes, the run time increases rapidly as d is increased.

α controls the tradeoff between area and speed. The larger is α (the coefficient multiplying the area component of the weight), the more we want to avoid the duplication of logic

Example	Area			Delay		
	Before	After	%Inc	Before	After	%Dec
5xp1-hdl	121	278	129.75	22.01	17.21	21.81
5xp1	237	252	6.33	13.20	11.76	10.91
9sym-hdl	186	188	1.08	32.50	26.30	19.08
9sym	403	405	0.50	18.47	16.93	8.34
9symml	343	354	3.21	22.45	21.89	2.49
alupla	286	346	20.98	23.80	19.31	18.87
bw	339	339	0.00	15.20	14.10	7.24
con1	36	36	0.00	5.80	5.80	0.00
duke2	706	720	1.98	23.83	19.76	17.08
f2	44	44	0.00	5.29	5.29	0.00
f51m-hdl	118	176	49.15	24.01	19.81	17.49
f51m	245	255	4.08	13.86	12.32	11.11
misex1	117	117	0.00	9.85	9.85	0.00
misex2	210	210	0.00	10.90	10.90	0.00
misex3	806	807	0.12	29.43	23.48	20.22
misex3c	970	1001	3.20	19.90	18.30	8.04
rd53-hdl	59	64	8.47	13.70	12.30	10.22
rd53	112	114	1.79	10.55	9.65	8.53
rd73-hdl	111	162	45.95	21.00	19.10	9.05
rd73	231	233	0.87	14.31	13.28	7.20
rd84-hdl	137	145	5.84	22.60	21.40	5.31
rd84	410	416	1.46	22.10	20.57	6.92
sao2-hdl	280	461	64.64	41.53	36.41	12.33
sao2	312	334	7.05	15.90	14.70	7.55
vg2	169	203	20.12	13.20	10.80	18.18
z4ml-hdl	94	135	43.62	17.90	12.24	31.62
z4ml	63	129	104.76	15.90	10.90	31.45

Table 1: Results of timing optimization

during partial_collapse. In cases when we want a speed-up irrespective of the increase in area, we set $\alpha = 0$.

model The delay trace performed on the circuit can use a variety of delay models. The most primitive is the *unit_delay_model* which assigns a delay of 1 unit to a gate. The *unit_fanout_delay_model* incorporates an additional delay of 0.2 units for each fanout. The *library_delay_model* uses the delay data in the library cells to provide more accurate delay values. By using a crude delay model (e.g. *unit_fanout_delay_model*) initially and a more refined delay model later one can significantly reduce the run time.

4 Results

The above algorithm has been implemented and integrated with MIS [2] . Table 1 shows the results obtained on examples from the MCNC logic synthesis benchmark set. Each was run through MIS using an algebraic script to get a multi-level description. Then this was resynthesized for timing using our algorithm. The *unit_fanout_delay_model* was used during the resynthesis. All delays were measured after mapping the circuit into the MCNC standard cell library. The increase in speed is entirely due to changing the structure of the circuits. The average delay reduced by 13 % while the area increase was 10.9 %.

A typical curve showing the area vs. delay as the algorithm progresses is plotted in Figure 9. This demonstrates the ability of the algorithm to tradeoff area for speed.

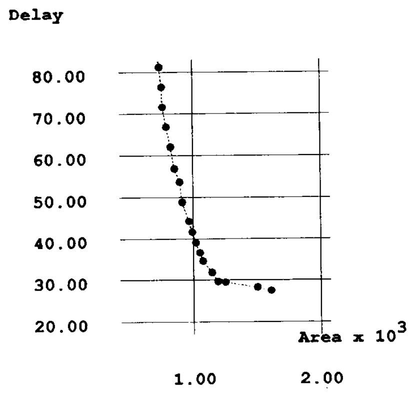

Figure 9: Typical Area-Delay tradeoff curve

5 Conclusions and future work

Algorithms for the speed up of combinational logic have been proposed. These algorithms have been implemented and integrated with the logic synthesis system MIS. Results obtained by running the algorithm on a number of design examples are encouraging since they demonstrate the potential for speed-up by simply restructuring the logic.

There is still room for improving the algorithms described above. Choosing a schedule for varying the parameters of the algorithm to obtain good quality results in a reasonable amount of time needs to be investigated further. In addition, we need to look at ways of combining transistor sizing and logic resynthesis.

References

[1] K. Bartlett, W. Cohen, A. de Geus, and G. Hachtel. Synthesis and optimization of multilevel logic under timing constraints. *IEEE Transactions on Computer-aided design*, CAD-5(4):582–595, October 1986.

[2] R. K. Brayton, Richard Rudell, Alberto Sangiovanni-Vincentelli, and Albert R. Wang. MIS: A multiple-level logic optimization system. *IEEE Transactions on Computer-aided design*, (6):1062–1081, November 1987.

[3] J. Darringer, D. Brand, J. Gerbi, W. Joyner, and L.Trevillyan. LSS: A system for production logic synthesis. *IBM Journal of Research and Development*, 28(5):326–328, September 1984.

[4] Ewald Detjens, Gary Gannot, Richard Rudell, Alberto Sangiovanni-Vincentelli, and Albert Wang. Technology mapping in MIS. *ICCAD-87 Digest*, :116–119, 1987.

[5] Mark Hoffman and Jac. K. Lim. Delay optimization of combinational static CMOS logic. In *Proceedings of the Design Automation Conference*, 1987.

[6] Giovanni De Micheli. Performance-oriented synthesis of large-scale domino CMOS circuits. *IEEE Transactions on Computer-aided design*, CAD-6(5):751–765, September 1987.