

# Delay Minimization and Technology Mapping of Two-Level Structures and Implementation Using Clock-Delayed Domino Logic

Jovanka Ciric, Gin Yee, and Carl Sechen

Dept. of Electrical Engineering, Univ. of Washington, Box 352500, Seattle, WA 98195, U.S.A.

## Abstract

*This paper presents a new delay minimization and technology mapping algorithm for two-level structures (TLS) implemented using clock-delayed (CD) domino logic. We take advantage of CD domino's high-speed, large fan-in NOR and OR gates to increase the speed of a circuit by partial collapsing. The algorithm is delay-driven and the delays are obtained from a characterized CD domino library. The results on eight combinational MCNC benchmark circuits show an average speed improvement of 89% for CD domino with TLS, compared to static CMOS implementations generated by Synopsys. CD domino with TLS using our tools produced on average 44% faster circuits than CD domino benchmarks minimized and mapped using Synopsys. At last, the delay results for CD domino with TLS were on average 22% better than for standard domino.*

## 1. Introduction

Dynamic circuits have become a standard circuit technique used in today's state-of-the-art microprocessors for meeting timing requirements not achievable with static CMOS. Design automation of random logic blocks using dynamic circuits has been limited due to noise considerations, possible charge sharing, and lack of tools. The need for making faster circuits and time-to-market pressure necessitates building tools for automatic logic synthesis of various dynamic logic families.

Dynamic circuits are faster than static CMOS for three reasons: 1) They have a lower switching threshold, 2) a smaller input capacitance, and 3) typically fewer logic levels are needed to implement a circuit since wider gates (gates with more devices in parallel in the pull-down network) are possible. However, the synthesis of the most popular dynamic logic family, standard domino logic, is complex because only non-inverting gates are allowed in the network. Therefore, the network has to be made positive unate prior to mapping. In the case of clock-delayed (CD) domino logic, both inverting and non-inverting gates are permitted in the design, which means that similar algo-

rithms as for static CMOS synthesis and delay optimization can be applied to CD domino logic.

Previous work on delay minimization was done during the technology-independent phase ([2], [3]), or technology mapping phase [1]. The FlowMap algorithm [12], developed for lookup-table FPGAs, labels each node with its optimal depth using the maximum flow algorithm. Optimization techniques applied individually during each of the phases have limited effect, because decisions made in one phase affect the result of the next phases. The best results in delay minimization are obtained when logic decomposition and technology mapping are performed simultaneously ([4], [5]).

The algorithm in [3] uses clustering and partial collapsing for delay minimization. It is based on Lawler's algorithm [7], which minimizes under a cluster capacity constraint the maximum number of clusters a signal has to traverse from primary inputs to primary outputs. The Touati et al. algorithm [3] applies Lawler's algorithm for clustering the nodes, and then performs collapsing of all nodes in the cluster into their root. The drawback of this approach is that no delay information is known (all nodes have load-independent unit delay).

A generalization of Lawler's algorithm for circuit partitioning is presented in [6]. It describes a delay-optimal clustering algorithm under the general delay model: each gate has a characterized delay, delay between the clusters is  $D$ , and there is no interconnection delay between gates in the same cluster. The clustering constraint is the area capacity, but any monotone constraint is valid.

The goal of this work is to minimize the delay of the circuit implemented using high-speed CD domino gates. A common way to improve the speed of a circuit is to implement it as a PLA structure, by collapsing the network into two gate levels. However, not all circuits can be collapsed into two levels of logic without paying an intolerable cost in area. The alternative approach is to partially collapse the network, in order to reduce the number of gate levels without causing a huge area increase. We are mapping the network onto two-level structures (TLS), so we don't need the matching techniques as in [1], [4], and [5]. A TLS is in essence a PLA with variable AND and OR planes. We take advantage of the fast high fan-in gates in

clock-delayed domino logic, to implement the TLS in a high-speed manner.

We developed a new delay minimization and technology mapping algorithm for TLS implemented using CD domino logic. The delay-driven clustering and collapsing algorithm is tailored to minimize circuit delay. In our approach, we collapse the nodes which belong to the same cluster and the delay of the resulting node changes, so the premises of delay optimality in [6] can not be applied. The reason is that the collapsing may also simplify the node so the delay is not monotonic. Instead of cluster size, the clustering algorithm uses *delay improvement* to select the nodes belonging to the same cluster.

Then a technology mapper was developed to map the minimized circuit onto a set of TLS. Since we characterize the delays of the TLS, the delay minimization algorithm can use the actual delay information to minimize the delay.

The remainder of the paper is organized as follows. Section 2 describes clock-delayed domino logic. The synthesis and design flow of CD domino gates using TLS is presented in Section 3. The delay minimization algorithm and technology mapping onto TLS is explained in Section 4. Experimental results comparing CD domino with TLS decomposition and static CMOS on eight MCNC benchmark circuits are shown in Section 5. The comparison with standard domino logic [10] and CD domino mapped using Synopsys tool are also included. Finally, concluding remarks are in Section 6.

## 2. Clock-Delayed (CD) Domino Logic

Clock-delayed domino is a self-timed dynamic logic family that provides both inverting and non-inverting functions [8]. A CD domino gate consists of a dynamic gate, and an optional delay element for the clock signal. The self-timed clock output of the delay element tells the fan-out gates when the data output is ready. The clock delay ensures that the clock output's evaluation edge for the fan-out gates occurs when the driving gate's output is stable, and prevents false evaluation of the subsequent gates caused by non-monotonic input transitions.

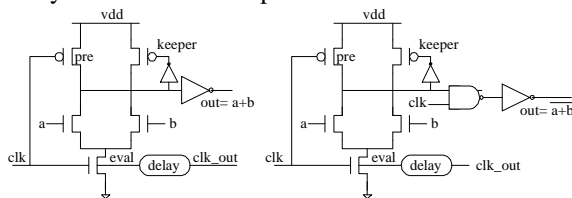


Figure 1. CD domino footed OR2 and NOR2 gate

Fig. 1 depicts the precharge-low footed CD domino OR2 and NOR2 gates. Inverting gates (e.g. the NOR2 in Fig. 1) have a NAND gate attached to the precharge node, which receives one input from the clock. This ensures that the output will be zero during precharge. The size (speed) of the NAND gate must be tuned to the speed of the nMOS

dynamic gate to prevent an unacceptable glitch (e.g. more than 1% of  $V_{dd}$ ) at the output at the start of evaluation. The precharge-low footless CD domino gates look the same, except that the evaluation transistor is omitted. The footless CD domino gates are faster than footed gates. We also use dual-output gates in the design, when both polarities of a signal are needed. In this way, the only inverters in a design appear at the primary inputs and outputs. An example of a footed dual\_OR2 gate is shown in Figure 2. All CD domino gates are sized for speed.

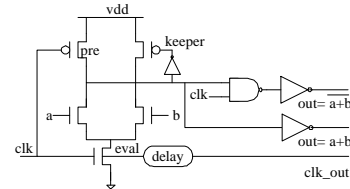


Figure 2. CD domino footed dual\_OR2 gate

Our CD domino circuits are fully leveled as shown in Fig. 3. In this way, only one delay element is needed per level. This delay element is tuned to the slowest gate at its corresponding level, plus a margin (we use 20%, as this is an upper bound on a typical industrial process) to allow for process variations. The delay element is typically a copy of the slowest gate in that level, augmented with additional wire load to yield the desired margin.

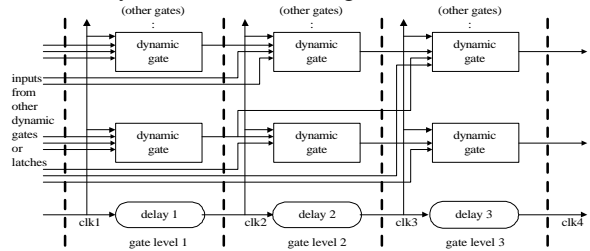


Figure 3. CD domino leveled clocking scheme

All of the gates at the same stage ( $i$ ) receive inputs from the previous stage ( $i-1$ ). In order to fully levelize a circuit, it may be necessary to insert dynamic buffers if a signal is generated more than one stage prior to the stage that uses the signal as an input. For example, a signal that is generated in stage  $i-3$  and is used as an input in stage  $i$  must propagate through stages  $i-2$  and  $i-1$  by way of dynamic buffers.

One of the key advantages of CD domino is that it uses single-rail circuits to implement a logic function. This is in contrast to standard domino logic, where dual rail circuits are needed for mapping a binate logic network. Another important advantage of CD domino is that high-speed, large fan-in NOR and OR gates are provided, with only two nMOS transistors in series for footed gates, and only one nMOS for footless gates. Moreover, very large fan-in NOR and OR gates can be decomposed into two or three levels.

### 3. CD Domino Synthesis using TLS

Motivation for our proposed TLS comes from PLAs, a common implementation of logic blocks that need to be high speed. In TLS, the PLAs AND plane is implemented using CD domino NOR gates and the OR plane is implemented using CD domino OR gates (Fig. 4).

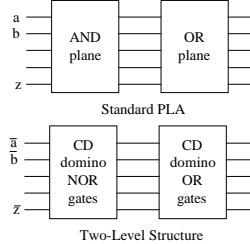


Figure 4. Standard PLA vs. Two-Level Structure

The problem we are addressing is how to decompose a network into one or more TLS (Fig. 5) such that the overall delay is minimized, where the underlying technology is CD domino. In degenerate cases, a TLS can be an OR gate (missing AND plane) or a NOR gate (missing OR plane).

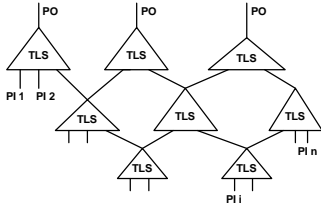


Figure 5. TLS decomposition of a network

The synthesis and design flow for CD domino logic using TLS is presented in Figure 6. The delay minimization algorithm will be described in Section 4.1, and the technology mapping algorithm in Section 4.2. The design is carefully verified after each step in the flow.

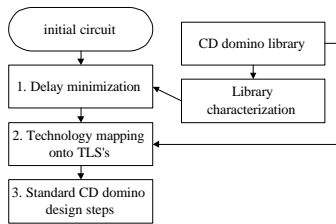


Figure 6. CD domino synthesis and design steps [8]

### 4. Delay Minimization and Technology Mapping for Two-Level Structures

The purpose of the delay minimization algorithm is to partially collapse the network, in order to minimize the delay after it is mapped onto TLS. The main premise in the algorithm is that at any stage, we know the delay of each node in the network, since it will be mapped onto a TLS

during technology mapping. All of the NOR and OR CD domino gates in the library are pre-characterized for delay versus load. Therefore, given a particular number of literals for a product term and a particular number of product terms, it is straightforward to look up the relevant delay for a node to be implemented as a TLS. The delay model takes into account the loading on the node. Currently, the loading due to interconnect is not included, but it would be easy to include these estimates if desired.

#### 4.1. Delay Minimization Algorithm

The delay minimization algorithm is based on Rajaraman's algorithm [6]. The algorithm consists of two phases: *Labeling* (Fig. 8) and *Mapping*. At the beginning, the original network is minimized using SIS [9], and decomposed into inverters, OR and AND gates with 2-11 inputs, using the SIS command `tech_decompose -a 11 -o 11`.

In the labeling part, the nodes are visited in topological order from primary inputs to primary outputs. The label of the primary inputs is assigned to 0. The label of each internal node in the network corresponds to the best found delay realized at the node. When a node  $v$  is visited, all nodes in its fan-in cone have been already processed and labeled accordingly. The node  $v$  is potentially merged using *espresso* (from SIS) with one of the nodes in its fan-in cone. *Espresso* is a command in SIS for two-level minimization. The nodes in the fan-in cone are sorted in non-increasing order of their labels. In that way,  $v$  will be first merged with its predecessors with greater delays. If the delay at the output of node  $v$  is smaller after merging, it becomes the new best delay for the node. The merged predecessor node is added to the cluster of  $v$ . The process of potential merging stops when the delay doesn't improve after checking all input nodes to the cluster of  $v$ . The best delay at  $v$  is saved as its label, together with the nodes which belong to the cluster of  $v$ .

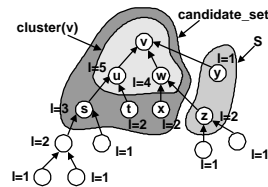


Figure 7. An example of Labeling ( $v$ )

Before outlining the algorithm *Labeling*, we will define the terms used. *Cluster* ( $v$ ) refers to the set of nodes in the fan-in cone of node  $v$  which are merged with  $v$ . *Label* ( $v$ ) corresponds to the maximum delay at the output of  $v$ . *Input* ( $v$ ) are all the nodes which are inputs to node  $v$ . The *best\_found\_cluster\_delay* is the best found delay so far realizable at the output of *cluster* ( $v$ ). *Candidate\_set* is the set of nodes in the fan-in cone which could become part of *cluster* ( $v$ ) if after merging all the nodes in the *candidate\_set* the resulting overall delay is smaller than the *best\_found\_cluster\_delay*. *Candidate\_set* is a superset of *cluster* ( $v$ ).  $S$  is the array of nodes in the fan-in cone of  $v$ ,

sorted in non-increasing order of their labels, which need to be processed.  $Node\_delay(v)$  is the delay from any input of  $v$  to its output (simple node delay). Figure 7 shows an example of labeling node  $v$ .

At the beginning, nodes  $u$ ,  $w$  and  $y$  are in  $S$ , sorted in non-increasing order of their labels. Let's say that after merging  $u$  and  $w$  with  $v$ , the resulting delay at the output of  $v$  improves. Then,  $u$  and  $w$  become part of  $cluster(v)$ , and nodes  $s$ ,  $t$ , and  $x$  are added to  $S$ . First, node  $s$  is tried to merge with  $cluster(v)$ , and if the delay does not improve, it is added to  $candidate\_set$ . Then, nodes  $t$  and  $x$  are tried and added to  $candidate\_set$  because the delay is not better. If after trying nodes  $z$  and  $y$  the delay does not improve, the process ends and  $cluster(v)$  contains nodes  $v$ ,  $u$ , and  $w$ . Otherwise,  $candidate\_set$  becomes part of  $cluster(v)$ , and the nodes which are input to the nodes in  $candidate\_set$  are added to  $S$ .

#### Algorithm Labeling (node $v$ )

```

Begin
cluster( $v$ )  $\leftarrow$  { $v$ };
candidate_set  $\leftarrow$  { $v$ };
best_found_cluster_delay = node_delay( $v$ ) + max{label( $k$ ) |  $k \in$ 
input( $v$ )};
 $S \leftarrow$  input( $v$ ); sorted in non-increasing order of their labels
while ( $S \neq \emptyset$ )
  remove the first node  $u$  from  $S$ ;
  candidate_set  $\leftarrow$  candidate_set  $\cup$  { $u$ }
   $S \leftarrow S \setminus \{u\}$ ;
   $u' \leftarrow$  espresso {candidate_set};
  new_delay = node_delay ( $u'$ ) + max {label ( $i$ ) |
 $i \in$  input(candidate_set) };
  if (new_delay < best_found_cluster_delay)
    cluster( $v$ )  $\leftarrow$  cluster( $v$ )  $\cup$  candidate_set;
    best_found_cluster_delay = new_delay;
     $S \leftarrow S \cup$  input(candidate_set);
  endif
endwhile
label ( $v$ ) = best_found_cluster_delay;
save {label ( $v$ ),  $\hat{v} \leftarrow$  espresso {cluster ( $v$ )}};
end

```

Figure 8. The Labeling algorithm

Note that in the *Labeling* algorithm  $u'$  is a single node, obtained by running *espresso* on the set of nodes in the candidate set. Also, in the implementation of the algorithm, instead of saving the set of nodes in  $cluster(v)$ , we save a single node  $\hat{v}$  obtained by performing *espresso* on all nodes in  $cluster(v)$ . From now on, we will denote  $\hat{v}$  to be the merged node obtained by running *espresso* on  $cluster(v)$ .

After the labeling phase, all nodes in the network are marked with their best delays and how they should be merged with other nodes in the fan-in cone to obtain these delays. The label of each node  $v$  in the network is given by the formula:

$$label(v) = node\_delay(\hat{v}) + \max\{label(i) | i \in input(\hat{v})\}$$

In the next phase, called *Mapping*, the nodes are visited from primary outputs to primary inputs. In this phase, we generate a new network  $N'$ , by selecting the clusters among those generated in the first phase. We maintain a list  $L$  of nodes whose clusters will be added to the new network  $N'$ . Initially  $L$  is set to the set of all primary outputs of  $N$ . Then the following 3 steps are repeated until  $L$  is empty: 1) remove a node  $v$  from  $L$ ; 2) add the merged node  $\hat{v}$  to the new network  $N'$ ; 3) for every node  $x \notin L$ , such that  $x$  is an input to  $\hat{v}$  and  $\hat{x} \notin N'$ , add  $x$  to  $L$ .

The complete algorithm for delay minimization is outlined in Fig. 9.

#### Algorithm Min\_Delay (Network $N$ )

```

Begin
sort all non-PI nodes in topological order to obtain list  $T$ 
/* Labeling phase */
for all PI nodes  $p \in N$ , do label ( $p$ ) = 0;
while ( $T \neq \emptyset$ )
  remove first node  $v$  from  $T$ ;
  call Labeling ( $v$ );
endwhile
/* Mapping phase */
 $L \leftarrow PO$ ;
 $N' \leftarrow \emptyset$ ;
while ( $L \neq \emptyset$ )
  remove node  $v$  from  $L$ ;
   $N' \leftarrow N' \cup \{\hat{v}\}$ ;
   $L \leftarrow L \cup input(\hat{v}) \setminus \{\hat{u} | \hat{u} \in N'\}$ ;
endwhile
 $N'$  is the new partially collapsed network;
end

```

Figure 9. The delay minimization algorithm

The key difference between Rajaraman's [6] and our algorithm is that we perform two-level minimization of the nodes in the cluster and obtain a new merged node whose delay is accurately estimated from the characterized library. Also, we don't have a cluster size as a clustering constraint. The only constraint in clustering is delay improvement. This algorithm is a heuristic, but we achieved satisfactory results as reported in the next section. To make the algorithm optimal, one would have to try all the subtrees of the node's fan-in cone (rooted in the node), and pick the one with the smallest delay. The reason is that the *espresso* command can return a node that is simplified, so the monotonic property for the delay doesn't hold. But the complexity of the optimal algorithm is exponential, so it is not practical for the usual circuit sizes.

## 4.2. Mapping onto Two-Level Structures

The technology mapper maps the minimized and partially collapsed network to a TLS representation. Each node in the network is represented in a sum-of-products form. Each product term is implemented using a CD domino NOR gate, and the sum of product terms is implemented using either a CD domino OR, NOR or

dual\_OR gates, depending on the needed signal polarities. The mapper also recognizes XOR and XNOR functions and implements them as a single gate. If both polarities of the XOR gate output are needed, the mapper inserts a dual\_XOR gate.

If the number of literals in a product term or the number of product terms exceed the maximum fan-in of the gates in the CD domino library, the NOR or OR gates are decomposed into two levels.

The CD domino library contains NOR, OR, and dual\_OR gates from 2 to 11 inputs, XOR and dual\_XOR gates, and a static inverter for inverting the primary inputs. The mapped circuits are fully leveled, and appropriate delay elements including 20% margin are then added, as described in Section 2.

## 5. Experimental Results

To evaluate the effectiveness of the new delay minimization and technology mapping algorithm for TLS, we compared Synopsys-generated static CMOS implementations of eight MCNC benchmark circuits with those we generated using footless CD domino implementations (Table 1). CD domino circuits were placed using the TimberWolf placement tool, and routed using Cadence’s Silicon Ensemble. TimberWolf was constrained to place the cells belonging to a logic level in adjacent rows [11]. The static CMOS results were obtained using Silicon Ensemble for both placement and routing. We used the 0.8 $\mu$ m MO-SIS CMOS process for the simulations. Pathmill was used to measure the longest path delay through each benchmark for static CMOS and Hspice for CD domino. Both versions of each benchmark were logically verified to ensure they each yield the same combinational logic function as implied by the input descriptions.

The static CMOS library contains 30 cells: 26 logically different cells, plus 2x and 4x inverters and buffers. The cells are the same as in the lib2 library from SIS with the addition of buffers. They are: 2-4 input NAND and NOR gates, XOR, XNOR, eight AOIs and eight OAI, inverters and buffers. The nMOS device widths were 8 $\mu$ m, and the pMOS device widths ranged from 15 $\mu$ m to 27 $\mu$ m for the various gates. For CD domino, the logic devices in all gates were 5.5 $\mu$ m (except for the XORs where they were 7.5 $\mu$ m). The precharge devices had the same widths as the logic devices. The static library was characterized using Hspice, and the delay data were used for creating the Synopsys target library. The static benchmarks were minimized for delay and tech-mapped using the Synopsys synthesis tool. We used four scripts for delay optimization: with flattening and structuring, with flattening and no structuring, without flattening and with structuring, and without flattening and structuring. The reported delays are from the script that produced the smallest delay. The CPU times for CD domino include the times for both the delay

minimization and mapping tools. The longer running times are for the benchmarks which start with a larger number of nodes. The mapping phase takes less than a second for all the benchmarks.

Note that our new delay minimization and technology mapping algorithms have sharply reduced the number of gate levels compared to the static CMOS implementations, despite the fact that the latter are minimized for speed via Synopsys. Also, each gate level is quite fast since the gates are typically NOR, OR or XOR gates. So even though our delay elements are tuned to the slowest gate at each level, and even though each delay element incurs an additional delay due to the 20% margin requirement, significant speed gains over static CMOS are readily available. Generally, the gate count increase ratio is less than the delay improvement ratio. Our current delay minimization algorithm does not consider gate count in any way. We are currently extending the algorithm to avoid aggressive collapsing of nodes off the critical path to achieve appreciable area reduction.

**Table 1.** Benchmark results for static CMOS and CD domino using two-level structures (TLS).

benchmark	active area [mm <sup>2</sup> ]	#gate levels	delay [ns]	delay ratio	CPU time [s]
s – static					
t - TLS					
des (s)	11.21	12	6.95	1.84	-
des (t)	20.07	7	3.78	1	349.2
term1 (s)	0.48	11	3.86	2.46	-
term1 (t)	3.76	4	1.57	1	10.1
x3 (s)	2.40	10	4.22	2.42	-
x3 (t)	8.52	4	1.74	1	18.6
k2 (s)	3.68	15	7.07	2.53	-
k2 (t)	12.57	5	2.79	1	293.9
C1355 (s)	1.69	17	9.07	1.95	-
C1355 (t)	4.71	8	4.66	1	33.8
t481 (s)	0.17	8	2.53	1.59	-
t481 (t)	0.26	4	1.59	1	1.2
dalu (s)	2.65	14	6.13	1.32	-
dalu (t)	6.37	9	4.64	1	102.2
rot (s)	2.81	13	5.62	1.04	-
rot (t)	10.45	11	5.41	1	72.6

We also directly compared our delay minimization and technology mapping algorithm with the mapper in Synopsys, both using the same previously mentioned CD domino library. To encourage Synopsys to use higher fan-in gates, a large fanout limit was used. Note that the standard CD domino design steps [8] reduce the fanout limit to 16. We also gave Synopsys a high cost for inverter usage (since our library provides both polarities of the output) so that inverters appear only at the primary inputs. The delay data from the characterized CD domino library were used for creating a Synopsys dynamic target library. Two minimization scripts were applied with high mapping effort: flat-

tening and structuring, and flattening and no structuring. After mapping, the final circuit is obtained with standard CD domino design steps. The pre-layout results are presented in the column CD-Syn in Table 2. The delays were obtained from *dyn*, a CD domino synthesis tool [11].

On average, CD domino with TLS using our new tools produced 44% faster circuits. These results show that our delay minimization and technology mapping algorithm is clearly more effective at minimizing delay than Synopsys when using the same library.

We compared our CD domino results with standard domino results published in [10]. The standard domino results were also obtained using MOSIS 0.8 $\mu$ m CMOS process. The logic devices in the gates were 12 $\mu$ m, evaluate devices were 20 $\mu$ m, and precharge devices were 8 $\mu$ m. The results presented in the DOM column of Table 2 show an average 22% speed improvement for CD domino with TLS. Note that only the TLS results are post-layout, implying that the speed improvements shown are actually lower bounds on what would be obtained after layout for the CD-Syn and standard domino implementations.

**Table 2.** Delay comparison between CD domino with TLS, CD domino using Synopsys (CD-Syn), and standard domino (DOM).

benchmark	CD-TLS	CD-Syn	%	DOM	%
	[ns] (post- layout)	[ns] (pre- layout)	worse than CD-TLS	[ns] (pre- layout)	worse than CD-TLS
des	3.78	7.49	98	N/A	N/A
term1	1.57	2.90	84	2.0	27
x3	1.74	2.52	45	2.8	61
k2	2.79	4.68	68	3.9	40
C1355	4.66	6.04	30	N/A	N/A
t481	1.59	1.67	5	N/A	N/A
dalu	4.64	4.62	0	4.7	1
rot	5.41	6.71	24	4.3	-21

## 6. Conclusion

We developed a new delay minimization and technology mapping algorithm for two-level structures implemented using clock-delayed (CD) domino logic. This is a delay-driven algorithm based on partial collapsing. We use ideas similar to Rajaraman’s algorithm, but with performing two-level minimization on the nodes in the cluster. We also apply the delay improvement as a clustering constraint, not the cluster size. The algorithm labels each node in the network with its best achievable delay, and then selects the clustering solution such that the overall circuit delay is minimized. The delays are obtained from a characterized CD domino library. We take advantage of high-speed, large fan-in NOR and OR gates from CD domino, to reduce the number of gate levels and increase the speed of the circuit. The circuit is mapped onto a network of TLS implemented using CD domino gates from the library.

Eight combinational MCNC benchmarks were implemented using static CMOS and CD domino with TLS decomposition. The results show that CD domino circuits are on average 89% faster than static CMOS. To compare the effectiveness of our algorithm, we minimized and mapped CD domino benchmarks using the Synopsys synthesis tool. CD domino with TLS using our tools produced on average 44% faster circuits. CD domino with TLS is on average 22% faster than standard domino.

## 7. Acknowledgements

We are grateful for the financial support provided by the SRC, CDADIC, National Science Foundation and Sun Microsystems. We would like to acknowledge the help of Tyler Thorp.

## References

- [1] Y. Kukimoto, R. Brayton, and P. Sawkar, “Delay-Optimal Technology Mapping by DAG Covering,” *Proc. of 35<sup>th</sup> ACM/IEEE Design Automation Conference*, pp. 348-351, June 1998.
- [2] K.J. Singh, A. Wang, R. Brayton, A. Sangiovanni-Vincentelli, “Timing Optimization of Combinational Logic,” *Proc. of IEEE/ACM Int. Conference on Computer-Aided Design*, pp. 282-285, 1988.
- [3] H. Touati, H. Savoj, and R. Brayton, “Delay Optimization of Combinational Logic Circuits by Clustering and Partial Collapsing,” *Proc. of IEEE/ACM Int. Conference on Computer-Aided Design*, pp. 188-191, Nov. 1991.
- [4] E. Lehman, et al., “Logic Decomposition during Technology Mapping,” *Proc. of IEEE/ACM Int. Conference on Computer-Aided Design*, pp. 264-271, November 1995.
- [5] M. Iyer, L. Stok, and A. Sullivan, “Wavefront Technology Mapping,” *Proc. of IWLS98*, pp. 419-426, June 1998.
- [6] R. Rajaraman, and D. F. Wong, “Optimum Clustering for Delay Minimization,” *IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, No. 12, pp. 1490-1495, December 1995.
- [7] E.L. Lawler, K.N. Levin, and J. Turner, “Module Clustering to Minimize Delay in Digital Networks,” *IEEE Transactions on Computers*, Vol. C-18, pp. 47-57, Jan. 1966.
- [8] G. Yee, and C. Sechen, “Dynamic Logic Synthesis,” *Proc. of IEEE CICC*, pp. 345-348, May 1997.
- [9] E. M. Sentovich, et al., “SIS: A System for Sequential Circuit Synthesis,” *Technical Report UCB/ERL M92/41*, University of California at Berkeley, May 1992.
- [10] T. Thorp, G. Yee, and C. Sechen, “Domino Logic Synthesis Using Complex Static Gates,” *Proc. of IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 242-247, November 1998.
- [11] G. Yee, “Dynamic Logic Design and Synthesis Using Clock-Delayed Domino,” *Ph.D. thesis*, University of Washington, Seattle, June 1999.
- [12] J. Cong, and Y. Ding, “FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs,” *IEEE Trans. on Computer-Aided Design*, pp.1-12, January 1994.