Synthesis and optimization of domino logic

Min Zhao and Sachin Sapatnekar
Department of Electrical Engineering
University of Minnesota
Minneapolis, MN 55455

Outline

- Introduction to domino logic
- Domino logic synthesis flow
- Technology mapping of domino logic
- Timing-driven static-domino partitioning
Basics of domino logic

Advantages of domino logic

- Speed advantages
  - Reduced fighting during transitions
  - Fewer transistors per gate, lower capacitive load

- Area advantages
  - Mainly consists of NMOS
  - $N+4$ transistors instead of $2N$ transistor per gate

Therefore, domino logic is widely used in high-performance circuit design.
Disadvantages of domino logic

- Disadvantages
  - Non-inverting nature may require logic duplication
  - Strict timing constraints
  - Charge sharing, noise susceptibility
  - High clock routing overhead
- Need automated techniques considering these issues for domino circuit design

Domino logic synthesis flow

1. Logic description (BLIF, Verilog)
2. Technology independent optimization
3. Partitioning: static-domino, between clock phases
4. Parameterized library technology mapping
5. Timing verification and optimization
6. Noise verification and optimization
7. Physical design
8. Clocking strategy
9. Library layout synthesizer

Timing constraints
Technology mapping of domino logic

What is technology mapping?

- Implement input network with gates in a library.
Parameterized library

- Large NMOS pull-down network of domino gate.
  - Small short circuit current and small driven load.
  - No complementary part.
  - The delay overhead of inverter may offset the advantage of fast switch speeds in small gates.
- Dramatical increase of library number with the increase of length(s) and width(p) of gate.
  \[(s,p): (3,6): 6877; (4,4): 3503; (4,6): 222943\]
- A parameterized library is applied for technology mapping of domino logic.

Problem definition

- A parameterized library
  - A collection of gates that satisfy the constraints on the width and height of the pull-down(pull-up) implementation of a gate.
  - Cell layout produced on the fly
- Technology mapping of domino logic
  - Given
    - An optimized Boolean network
    - A constraint on the width and height of domino gates
  - Find
    - Minimum cost solution to the problem that nodes in the network are implemented in domino logic
**General technology mapping algorithm**

- Dynamic programming algorithm is applied.
- At each network node
  - pattern matching
  - cost calculation for each possible matching
- The cost will be large if the library is large.

**Parameterized library mapping algorithm**

- Starting point
  - Given an arbitrarily optimized network
  - It is first unated
  - Then mapped into a two input AND-OR DAG
  - Then the DAG is decomposed into trees.

- Complexity
  - space complexity: $O(WHN)$
  - time complexity: $O(W^2H^2N)$

  - $W$: maximum number of parallel chains
  - $H$: maximum number of series transistors
  - $N$: number of nodes in the tree
Subsolutions

- Subsolution space at each node.

- Each stored subsolution is optimal for its subtree under specified constraints

Physically,

\[
\{S, P\} (S \geq 1 & P \geq 1) \text{ represents a segment of a domino pull-down whose height and width are } S \text{ and } P
\]

\[
\{1, 1\} \text{ represents a complete domino gate or a PI.}
\]

Basic Operations

- OR operation: \( S = \max(S_l, S_r), P = P_l + P_r \)

- AND operation: \( S = S_l + S_r, P = \max(P_l, P_r) \)

- PI / Gate formation operation: \( S = 1, P = 1 \)

A gate formation operation corresponds to a situation where the structure collected so far is converted to a domino gate with an output at that network node.
Node data structure

- Store the optimal subsolutions for all possible [height, width] combinations from [1,1] to [H,W].
- Each optimal subsolution can be represented as \( \{S, P, C, \{S_l, P_l\}, \{S_r, P_r\}\} \)
  - \( S (1 \leq S \leq H) \) is the maximum height of the current solution.
  - \( P (1 \leq P \leq W) \) is the maximum width of the current solution.
  - \( C \) is the cost.
  - \( \{S_l, P_l\}, \{S_r, P_r\} \) is the subsolutions of left and right child whose combination provides the minimal cost of subsolution \( \{S, P\} \)

Node data calculations

- \( \{S, P\} (S \geq 1 & P \geq 1) \) subsolution at a parent node is obtained by combining optimal subsolutions at child nodes.
- \( \{1, 1\} \) subsolution at a node is obtained from the subsolution of the same node whose cost is minimal.
- The procedure consists of
  - Node constraint functions
  - Node cost functions
**Node cost functions**

- Here, cost is area -- the number of transistors.
- Literal operation: \( C = C + 1 \)
  
  *Literal operation corresponds to a primary input or a situation where a new domino structure is started after gate formation operation.*
- OR/AND operation: \( C = \text{Literal}(C_l) + \text{Literal}(C_r) \)
- Gate formation operation: \( C = C_{\text{min}} + 4 \)
  
  *The minimal cost solution, \( C_{\text{min}} \) is the minimal value out of all \( H \times W \) optimal subsolutions*
  
  *‘4’ includes two clock control transistors + an inverter*

**Node mapping algorithm**

For each valid \([\text{height width}]\) subsolution of the left child {
  
  for each valid \([\text{height width}]\) subsolution of the right child{

  \( \{S, P\} = \text{Node constraint functions} (\{S_l, P_l\}, \{S_r, P_r\}) \);
  
  if \( \{S, P\} \) was within the constraints \((H, W)\) {

    \[
    C = \text{Node cost functions} (C_l, C_r)
    \]

    if \( C < C_{[S,P]_{\text{min}}} \) then \( C_{[S,P]_{\text{min}}} = C \).

    if \( C < C_{\text{min}} \) then \( C_{\text{min}} = C \).

  }

  
}

\( C[1,1] = \text{Gate formation} (C_{\text{min}}) \)
An example

- Of all (S,P) mapping subsolutions for the children only those with minimal cost are stored.

\[
\begin{align*}
C &= C_l + C_r \\
P &= \max(P_l, P_r) \\
S &= S_l + S_r
\end{align*}
\]

**AND node:**
- \(C = C_l + C_r\)
- \(P = \max(P_l, P_r)\)
- \(S = S_l + S_r\)

**Or node:**
- \(C = C_l + C_r\)
- \(P = P_l + P_r\)
- \(S = \max(S_l, S_r)\)

Gate formation:
- \(C = C_{\text{min}} + 4\)
- \(S = 1\)
- \(P = 1\)

Wide domino gate

- NAND, NOR gate can be used to replace inverter.
  - Break up large stacks of series transistors into parallel chains
Wide AND/OR domino gate mapping

- Enlarged subsolution space is used.

- Region a: standard domino gate mapping
- Region b: wide AND domino gate mapping
- Region c: wide OR domino gate mapping

Dual-monotonic gate

- A common dual-monotonic XOR gate.

- The presence of an XOR/XNOR function decomposes the input network into small mapping trees, which causes a larger area and delay cost.
Dual-monotonic gate mapping

- Recognize the XOR/XNOR logic of the network by pattern matching.
- Perform the technology mapping on the AND/OR/XOR/XNOR subject network, mapping AND/OR nodes to the standard domino gate and XOR/XNOR nodes to dual-monotonic gate.
- Permitted mapping scheme.

Implementation and results(1)

- Execution time: < 10 seconds
- Comparison with another domino mapper

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Our approach #trans/#level</th>
<th>Prasad et al. #trans/#level</th>
<th>Reduction %</th>
</tr>
</thead>
<tbody>
<tr>
<td>c8</td>
<td>289/6</td>
<td>328/7</td>
<td>13.5%</td>
</tr>
<tr>
<td>l6</td>
<td>890/2</td>
<td>890/3</td>
<td>0%</td>
</tr>
<tr>
<td>C880</td>
<td>1056/9</td>
<td>1499/7</td>
<td>42.0%</td>
</tr>
</tbody>
</table>

Comparison of various mapping methods

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Basic mapping #trans/#level</th>
<th>Wide AND/OR gate #trans/#level</th>
<th>Dual-mono gate #trans/#level</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1355</td>
<td>1824/9</td>
<td>1824/9</td>
<td>1360/7</td>
</tr>
<tr>
<td>C1908</td>
<td>1978/18</td>
<td>1965/18</td>
<td>1588/14</td>
</tr>
<tr>
<td>k2</td>
<td>2884/16</td>
<td>2738/15</td>
<td>2884/16</td>
</tr>
</tbody>
</table>
**Experimental results**

### Domino mapping vs. static mapping

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Domino #trans/#levels</th>
<th>SIS: 44-3.genlib #trans/#levels</th>
<th>Reduction %</th>
<th>Dup-ratio %</th>
</tr>
</thead>
<tbody>
<tr>
<td>i6</td>
<td>761/3</td>
<td>1194/5</td>
<td>36.3%</td>
<td>13%</td>
</tr>
<tr>
<td>C1355</td>
<td>1360/7</td>
<td>1378/20</td>
<td>1.3%</td>
<td>77%</td>
</tr>
<tr>
<td>C3540</td>
<td>4002/20</td>
<td>3140/34</td>
<td>-27.5%</td>
<td>92%</td>
</tr>
</tbody>
</table>

**Partitioning: Motivation**

- Use domino gates to speed up parts of the circuit; remainder is implemented in static CMOS
- Domino logic is typically multiphase
- General clocking strategy
Another consideration

- Observation: duplication cost can be reduced by proper partitioning
- An example

In addition to the partitioning cost, implementation cost varies with partitions.

Problem definition

- Static-domino partitioning problem
  
  **Given**
  
  - An optimized combinational circuit
  - The delay specification on the output of the network

  **Implement the nodes with domino+static logic**
  
  - Minimize the cost while meeting delay specs
  - Satisfy the precedence constraints that no static logic gate is permitted to fan out a domino gate

- Two-way domino partitioning
  
  - Partition the domino implementation into two phases, with inverters permitted between the phases.
The timing-driven static-domino partitioning algorithm

- Cost: area or power.
- Outline of the algorithm
  - Perform fast static and domino mapping on the entire logic network.
  - Apply a PERT based timing analysis method to find the candidate cut nodes in the network.
  - Build the flow network from the candidate cut nodes. The edge capacities are determined from the cost difference of static and domino implementations.

Static-domino mapping algorithm
Determining candidate cut nodes

- From the static mapping, get
  - \( D_{i,d}(v) \) (\( D_{i,s}(v) \)): the delay from the inputs to node \( v \) using a domino(static) implementation
  - Find the maximum delay from output to node \( v \)
- If maximum delay from input to output through node \( v \)
  - \( D_{i,d}(v) + D_{s,out}(v) < T_{\text{spec}} \)
  \( \Rightarrow v \) is a candidate cut node
**Static-domino partitioning algorithm**

**Finding the minimum cut**

- **Notation:**
  - \( S_1 (D_1) \): the static(domino) implementation cost of region A
  - \( S_2 (D_2) \): the static(domino) implementation cost of region B

- If regions A and B are implemented in static logic,
  \[
  \text{Cost}(s) = S_1 + S_2;
  \]

- If A is domino and B is static:
  \[
  \text{Cost}(d-s) = D_1 + S_2 = D_1 - S_1 + \text{Cost}(s)
  \]

- **Finding the minimum cut (Contd.)**

- Cost(s) is constant.
- Therefore, minimizing the partitioning cost is to find the region A whose \((D_1-S_1)\) is minimized.

- \((D_1-S_1)\) value of a partitioning
  \[
  \sum [d(i)-s(i)] \forall i \in \text{cutset between Region A and B}
  \]

- Build the flow network
  - Edge capacities are \([d(i)-s(i)]\) for each node \(i\)
  - Standard technique used to maintain precedence constraints
Building the maximum flow graph

- Build the vertex-cut maximum flow graph from candidate cut nodes.

Maximum flow graph (contd.)

- Constraints to max-flow min-cut algorithm
  - Maintain the predecessor constraints
  - Handle edges with negative capacities.

To solve the problem,
- Heuristically transform the vertex-cut maximum flow network into an edge-cut maximum flow network
- A positive initial flow is injected into the source node and distributed into the whole network.
- Edges with capacities of $\infty$ are introduced into the graph to force the predecessor constraint.
**Maximum flow graph: Example**

- Build edge-cut maximal flow graph
- Initial flow=32
- Add the edges with capacities of $\infty$

---

**A partitioning flow for a general two-phase clocking strategy**

- Perform static-domino partitioning on the entire network into domino region(1) and static region(2)
- Perform two-way domino partitioning on region 1 to obtain phase I region(3) and phase II region(4)
- Perform static-domino partitioning on region 3 into domino region(5) and static region(6)
Experimental results

Results of static-domino partitioning (one phase)

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Domino #trans</th>
<th>Static #trans #trans/delay</th>
<th>No spec #trans</th>
<th>Spec=(*1.25) #trans</th>
<th>Spec=(*1.05) #trans</th>
<th>CPU (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C3540</td>
<td>4527</td>
<td>2850/1.43</td>
<td>2748</td>
<td>3312</td>
<td>3987</td>
<td>10.9</td>
</tr>
<tr>
<td>des</td>
<td>9945</td>
<td>8134/4.25</td>
<td>7527</td>
<td>7536</td>
<td>7536</td>
<td>60.2</td>
</tr>
<tr>
<td>C7552</td>
<td>7919</td>
<td>5464/2.35</td>
<td>5370</td>
<td>5987</td>
<td>6198</td>
<td>30.9</td>
</tr>
</tbody>
</table>

Experimental results (Contd.)

Partitioning flow for two-phase clocking scheme

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Domino #trans</th>
<th>Static #trans #trans/delay</th>
<th>Spec=(*1.25) #trans/#latches</th>
<th>Spec=(*1.05) #trans/#latches</th>
</tr>
</thead>
<tbody>
<tr>
<td>c2670</td>
<td>1992</td>
<td>1754/1.75</td>
<td>1538/52</td>
<td>1538/52</td>
</tr>
<tr>
<td>K2</td>
<td>2884</td>
<td>2896/1.54</td>
<td>2691/157</td>
<td>2795/115</td>
</tr>
<tr>
<td>C3540</td>
<td>4527</td>
<td>2850/1.43</td>
<td>3063/60</td>
<td>3235/68</td>
</tr>
<tr>
<td>des</td>
<td>9945</td>
<td>8134/4.25</td>
<td>7510/118</td>
<td>7513/119</td>
</tr>
<tr>
<td>C7552</td>
<td>7919</td>
<td>5464/2.35</td>
<td>5754/164</td>
<td>5772/164</td>
</tr>
</tbody>
</table>
Conclusion

- Synthesis procedure for domino logic discussed
- Technology mapper: fast, good solutions
- Partitioning between static and domino to gain advantages of both
- Placed into a flow including transistor sizing and noise fixes for charge sharing