TIMING ISSUES IN
MULTI-LEVEL LOGIC
OPTIMIZATION

©Giovanni De Micheli

Stanford University
Outline

- Timing verification.
  - Delay modeling.
  - Critical paths.
  - The false path problem.

- Algorithms for timing optimization.
Timing verification and optimization

- Verification:
  - Check that a circuit runs at speed:
    * Satisfies I/O delay constraints.
    * Satisfies cycle-time constraints.

- Optimization:
  - Minimum area
    * subject to delay constraints.
  - Minimum delay
    * (subject to area constraints).
Delay modeling

- Gate delay modeling:
  - Straightforward for bound networks.
  - Approximations for unbound networks.

- Network delay modeling:
  - Compute signal propagation:
    * Topological methods.
    * Logic/topological methods.
Gate delay modeling
unbound networks

- Virtual gates:
  - Logic expressions.

- Stage delay model:
  - Unit delay per vertex.

- Refined models:
  - Depending on fanout.
Network delay modeling

• For each vertex $v_i$.

• Propagation delay $d_i$.
  
  – I/O propagation delays are usually zero.

• Data-ready time $t_i$.
  
  – Input data-ready times denote when inputs are available.
  
  – Computed elsewhere by forward traversal:

  
  $$t_i = d_i + \max_{j \mid (v_j, v_i) \in E} t_j$$
Example

- Propagation delays:
  - \( d_g = 3; d_h = 8; d_m = 1; d_k = 10; d_l = 3; \)
  - \( d_n = 5; d_p = 2; d_q = 2; d_x = 2; d_y = 3; \)
Network delay modeling

- For each vertex $v_i$.

- Required data-ready time $\bar{t}_x$.
  
  - Specified at the primary outputs.
  
  - Computed elsewhere by backward traversal:
    
    $$\bar{t}_i = \min_{j \mid (v_i, v_j) \in E} \bar{t}_j - d_j$$

- Slack $s_i$.
  
  - Difference between required and actual data-ready times $s_i = \bar{t}_i - t_i$. 
Required data-ready times:

\[ \bar{t}_x = 25 \text{ and } \bar{t}_y = 25. \]
Example

- \( s_x = 2; s_y = 0 \)
- \( \bar{t}_m = 25 - 2 = 23; s_m = 23 - 21 = 2; \)
- \( \bar{t}_q = 25 - 3 = 22; s_q = 22 - 22 = 0; \)
- \( \bar{t}_l = \min\{23 - 1; 22 - 2\} = 20; s_l = 20 - 20 = 0; \)
- \( \bar{t}_h = 23 - 1 = 22; s_h = 22 - 11 = 11; \)
- \( \bar{t}_k = 20 - 3 = 17; s_k = 17 - 13 = 4; \)
- \( \bar{t}_p = 20 - 3 = 17; s_p = 17 - 17 = 0; \)
- \( \bar{t}_n = 17 - 2 = 15; s_n = 15 - 15 = 0; \)
- \( \bar{t}_b = 15 - 5 = 10; s_b = 10 - 10 = 0; \)
- \( \bar{t}_g = \min\{22 - 11; 17 - 10; 17 - 2\} = 7; s_g = 7 - 3 = 4; \)
- \( \bar{t}_a = 7 - 3 = 4; s_b = 4 - 0 = 4. \)
Topological critical path

- Assume topologic computation of:
  - Data-ready by forward traversal.
  - Required data-ready by backward traversal.

- Topological critical path:
  - Input/output path with zero slacks.
  - Any increase in the vertex propagation delay affects the output data-ready time.

- A topological critical path may be false.
  - No event can propagate along that path.
Example

- All gates have unit delay.

- All inputs ready at time 0.

- Longest topological path: \((v_a, v_c, v_d, v_y, v_z)\).
  - Path delay: 4 units.

- Critical true path: \((v_a, v_c, v_d, v_y)\).
  - Path delay: 3 units.
Sensitizable paths

- A path in a logic network is *sensitizable* if an event can propagate from its tail to its head.

- A *critical path* is a sensitizable path of maximum weight.

- Only sensitizable paths should be considered.

- Non-sensitizable paths are *false* and can be discarded.
Sensitizable paths

• *Path*:
  – Ordered set of vertices.

• *Inputs* to a vertex:
  – Direct predecessors.

• *Side-inputs* of a vertex:
  – Inputs not on the path.
Dynamic sensitization condition

- Path: \( P = (v_{x_0}, v_{x_1}, \ldots, v_{x_m}) \).

- An event propagates along \( P \) if
  \[ \frac{\partial f_{x_i}}{\partial x_{i-1}} = 1 \ \forall i = 1, 2, \ldots, m. \]

- Remark:
  - Boolean differences are function of the side-inputs and values on the side-inputs may change.

  - Boolean differences must be true at the time that the event propagates.
Example

Path: \((v_a, v_c, v_d, v_y, v_z)\)

- \(\partial f_y / \partial d = e = 1\) at time 2.

- \(\partial f_z / \partial y = e' = 1\) at time 3.

- Not dynamically sensitizable because \(e\) settles at time 1.
Static sensitization

• Simpler, weaker model.

• We neglect the requirement on *when* the Boolean differences must be true to propagate an event.

• There is an assignment of primary inputs $c$ such that $\partial f_{x_i}(c)/\partial x_{i-1} = 1 \forall i = 1, 2, \ldots, m$.

• May lead to *underestimate* delays.
Example

- Not statically sensitizable.
Example

- All gates have unit propagation delay.
Example

- Topological critical paths:
  - \( \{(v_a, v_d, v_g, v_o); (v_b, v_d, v_g, v_o)\} \)
  - Path delay: 3.
  - Not statically sensitizable.

- Other path:
  - \( (v_a, v_e, v_o) \)
  - Path delay: 2.

- Assume:
  - \( c = 0 \) and \( a, b \) dropping from 1 to 0.
  - Event propagates to output !!!
Modes for delay computation

- **Transition mode:**
  - Variables assumed to hold previous values.
    * Model circuit node capacitances.
  - Need two input vectors to test.

- **Floating mode:**
  - Circuit is assumed to be memoryless.
  - Need only one test vector.
  - Variables have unknown value until set by input test vector.
Modes for delay computation

- *Floating mode* delay computation is simpler than *transition mode* computation.

- *Floating mode* is a pessimistic approach.

- *Floating mode* is more robust:
  - *Transition mode* may not have the *monotone speed-up* property.
Monotone speed-up property

- Propagation delays are upper bounds.
  - What happens if gates are faster than expected?

- We must insure that speeding-up a gate does not slow-down the circuit.
  - Topological critical paths are robust.
  - What about dynamically sensitizable paths in transition mode?
Example

- Propagation delays: 2 units.
- Shaded gate: 3 units and 1 unit.
Static co-sensitization

- Assumption:
  - Circuit modeled by $AND, OR, INV$ gates.
  - $INV$ are irrelevant to the analysis.
  - Floating mode.

- Controlling values:
  - 0 for $AND$ gate.
  - 1 for $OR$ gate.

- Gate has controlled value.
Static co-sensitization

- Path: $P = (v_{x_0}, v_{x_1}, \ldots, v_{x_m})$.

- A vector statically co-sensitizes a path to 1 (or to 0) if
  - $x_m = 1$ or (0) and
  - $v_{x_{i-1}}$ has a controlling value whenever $v_{x_i}$ has a controlled value.

- Necessary condition for a path to be true.
False path detection test

- For all input vectors, one of the following is true:
  
  - (1) A gate is controlled and
    
    * the path provides a non-controlling value
    
    * a side-input provides a controlling value.

  - (2) A gate is controlled and
    
    * the path and a side-input have controlling values
      
    * the side-input presents the controlling value first.

  - (3) A gate is not controlled and
    
    * a side-input presents the non-controlling value last.
Example

- Path: \((v_a, v_c, v_d, v_y, v_z)\).
- For \(a = 0, b = 0\)
  - condition (1) occurs at the OR gate.
- For \(a = 0, b = 1\)
  - condition (2) occurs at the AND gate.
- For \(a = 1, b = 0\)
  - condition (2) occurs at the OR gate.
- For \(a = 1, b = 1\)
  - condition (1) occurs at the AND gate.
Important problems

- Check if circuit works at speed $\bar{t}$.
  - Verify that all true paths are faster than $\bar{t}$.
  - Show that all paths slower than $\bar{t}$ are false.

- Compute groups of false paths.

- Compute critical true path:
  - Binary search for values of $\bar{t}$.
  - Show that all paths slower than $\bar{t}$ are false.
Algorithms for delay minimization

- Alternate:
  - Critical path computation.
  - Logic transformation on critical vertices.

- Consider quasi critical paths:
  - Paths with near-critical delay.
  - Small slacks.
REduce Delay ( $G_n(V, E), \epsilon$ ){
    repeat {
        Compute critical paths and critical delay $\tau$;
        Set output required data-ready times to $\tau$;
        Compute slacks;
        $U =$ vertex subset with slack lower than $\epsilon$;
        $W =$ select vertices in $U$;
        Apply transformations to vertices $W$;
    } until (no transformation can reduce $\tau$);
}
Transformations for delay reduction

- Reduce propagation delay.

- Reduce dependencies from critical inputs.

- Favorable transformation:
  - Reduces local data-ready time.
  - Any data-ready time increase at other vertices is bounded by the local slack.
Example

- Unit gate delay.

- Transformation:
  - Elimination.

- Always favorable.

- Obtain several area/delay trade-off points.
Example

- Iteration 1: eliminate $v_p, v_q$. (No literal increase.)
- Iteration 2: eliminate $v_u$. (No literal increase.)
- Iteration 3: eliminate $v_r, v_s, v_t$. (Literals increase.)
More refined delay models

- Elimination:
  - Reduces one stage.
  - Yields more complex and slower gates.
  - May slow other paths.

- Substitution:
  - Adds one dependency.
  - Loads and slows a gate.
  - May slow other paths.
Example

- NAND delay = 2. INVERTER delay = 1.

- All input data-ready are 0, except $t_d = 3$. 
Speed-up algorithm

- Determine a subnetwork $W$ of depth $d$.

- Collapse subnetwork by elimination.

- Duplicate vertices with successors outside $W$:
  - Record area penalty.

- Resynthesize $W$ by timing-driven decomposition.

- Heuristics:
  - Choice of $W$.
  - Monitor area penalty and potential speed-up.
Algorithms for minimal-area synthesis under delay constraints

- Make network *timing feasible*.
  - May not be possible.

- *Minimize area* while preserving timing feasibility.
  - Use area optimization algorithms.
  - Monitor delays and slacks.
  - Reject transformations yielding negative slacks.
Making a network timing feasible.

- Naive approach:
  - Mark vertices with negative slacks.
  - Apply transformations to marked vertices.

- Refined approach.
  - Transform multiple I/O delay constraints into single constraint by delay padding.
  - Apply algorithms for CP minimization.
  - Stop when constraints are satisfied.
Example
\[ \bar{t} = [2332]^T \]
Summary

- Timing optimization is crucial for achieving competitive logic design.

- Timing optimization problems are hard:
  - Detection of critical paths.
    * Elimination of false paths.
  - Network transformations.