TimeBoost

Fine-Grained Interleaving of Multithreaded Lagrange Relaxation based Gate Sizing with Buffering Optimizations

Apostolos Stefanidis, Dimitrios Mangiras, Giorgos Dimitrakopoulos

Integrated Circuits Lab
Electrical and Computer Engineering
Democritus University of Thrace
Xanthi, Greece

Outline

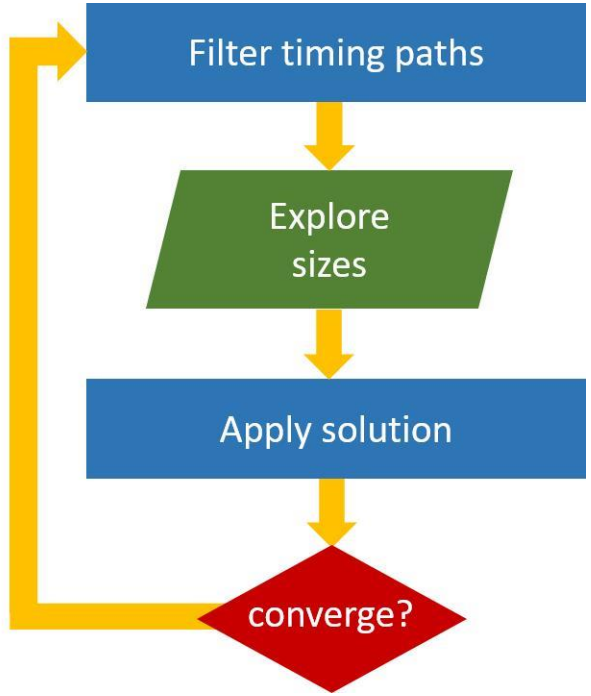
- Design optimization methods
- The order of application dilemma
- Fine-Grained Interleaving of Sizing/Buffering transformations
  - New Lagrange-relaxation-based gate sizing engine
    - Uniform treatment of all types of cells
  - Simplified buffering heuristics
  - Timing/Power Recovery steps
- Implementation
- Conclusions

Timing-driven design optimization

- Satisfy MMMC timing constraints and improve area and power performance without affecting functionality nor violating design rule constraints
  - A multidimensional problem that involves all steps of the flow
  - Placement – synthesis – routing – CTS all interact and affect the final result
  - Inherently complex and computationally challenging
- TAU 2019 workshop contest focused on logic sizing and buffering optimizations
  - Initial design placed with full SPEF wire parasitics and partial clock tree
  - Properly upsize/downsize given gates
  - Add/Remove buffers to datapath and clock nets

Gate sizing

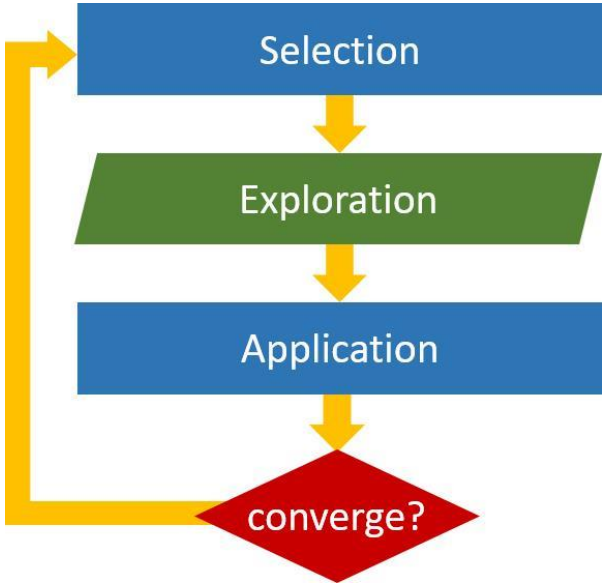
- **Gate sizing**
  - Decrease delay of driving gate for late timing violations
  - Decrease input capacitance to speedup driving gate
  - Increase delay for early timing violations
  - Save power/area
- **FF sizing**
  - Reduce clock-to-Q delays
  - Affect indirectly required arrival times on D pins
  - Changes clock pin capacitance
- **Local Clock Buffer sizing**
  - Increase/Decrease clock arrival time
  - Alter required arrival times on D pins
  - Useful clock skew optimization
- **Threshold voltage selection**
  - Can accompany cell sizing
  - Trades-off speed/leakage power (not required in TAU contest)
  - fully supported by our flow



Main buffering optimizations

- **Critical path isolation**
  - Reduce the capacitance seen by the driver of a net to speedup critical timing arcs
  - Add buffers at the non-critical endpoints
- **Buffering large fanout/capacitance nets**
  - Add buffers at the root of nets to ease driving their large fanout capacitance
  - Helps also in downsizing upstream gates to reduce leakage/area
- **Hold buffering optimization**
  - Add delay to improve early arrival times
  - Can be applied directly at the endpoints or to internal nodes to maximize buffer sharing
- **Local clock buffering on clock nets to introduce useful clock skew**
  - Normally handled during CTS
  - It can be applied incrementally post CTS to match the result of placement/sizing/buffering post CTS timing optimizations
- **Wire repeaters (Not allowed in TAU contest)**
  - Split wires with buffers to speedup wire traversal

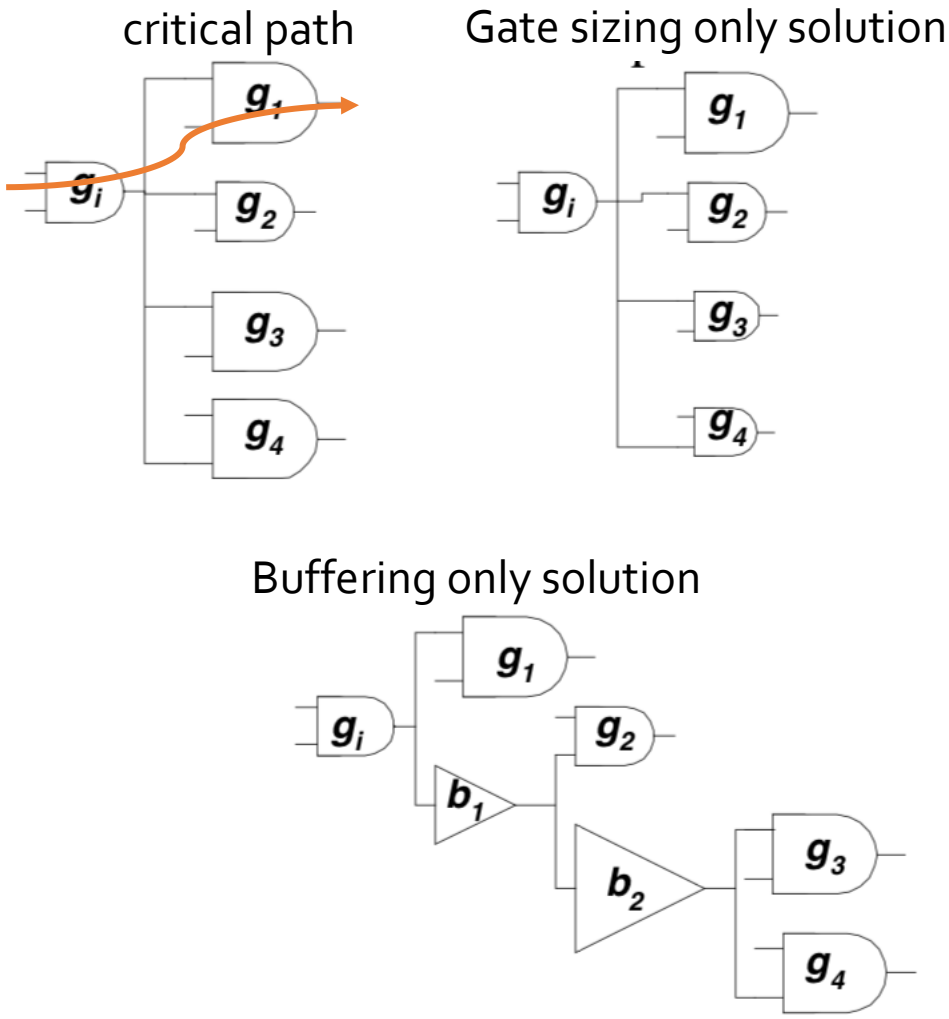
Main algorithmic loop



- Many iterations needed for convergence
  - Explore local solutions
- Runtime affected by number of critical nets and their complexity
- Incremental timing updates affect both QoR and runtime

How to apply optimization methods

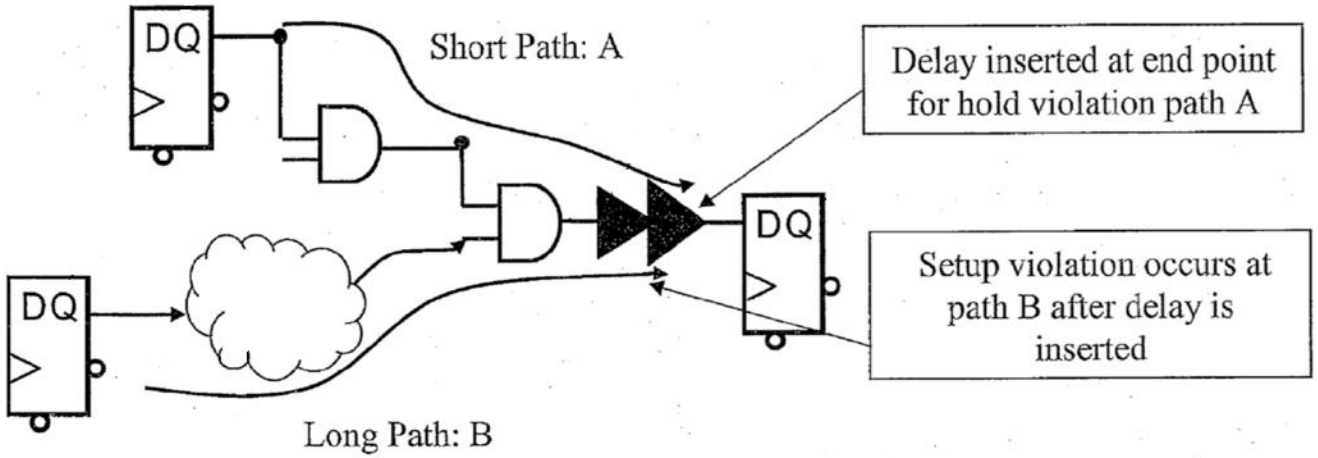
- The order of application of each optimization heuristics is critical to the final result
  - A gate sizing tool will likely size down the non-critical sinks *g2* to *g4* to improve the critical path's timing
  - Buffering tool will likely build a buffer tree to isolate the non-critical gates from driver *gi*.
- Each step tries to make use of all the freedom in the optimization space
  - It does not leave much optimization opportunity for the other.
  - Each step is limited in the kind of optimization that it can perform
- Rerunning heuristics not effective
  - Runtime is lost
  - Each algorithm needs many iterations to re-converge to the new solution after the "disruption" of previous solution



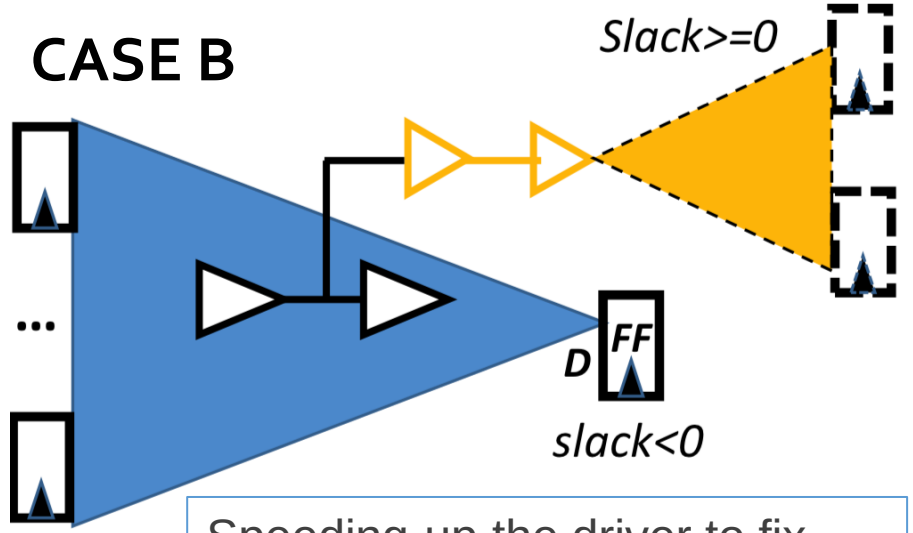
Extra examples

A change in a critical timing path can deteriorate a non-critical path

CASE A



CASE B

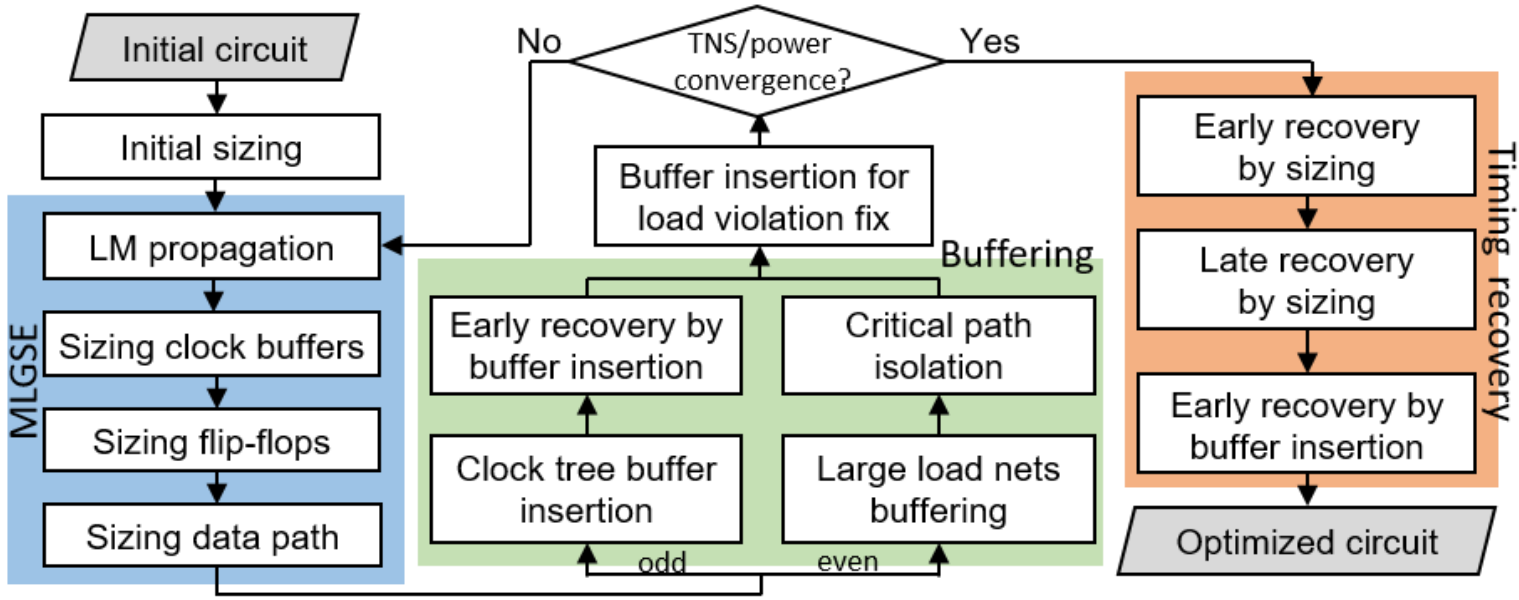


Speeding-up the driver to fix setup violation cause faster slew into positive slack region and cause a hold violation.

If gate sizing is interleaved in a fine-grained manner with hold buffering insertion any setup violation introduced can be easily removed in the following iterations

What we propose

- **Fined grained interleaving** of sizing and buffering optimizations
  - No algorithm runs to completion
  - **Sizing and buffering are interleaved per-iteration**
- Allows for **joint convergence**
  - Each sizing decision drives buffering additions and each buffering addition/removal guides next sizing choices
  - **Buffers are added gradually**
  - **Sizes adopt smoothly to design restructuring**



- Sizing is done with a new and multithreaded Lagrange-relaxation-based gate sizing engine (MLGSE) that handles uniformly gates, ffs and local clock buffers
- Once convergence is reached final recovery steps are applied
- Initial sizing focuses on cap and slew violations
  - All following steps do not to introduce new violations

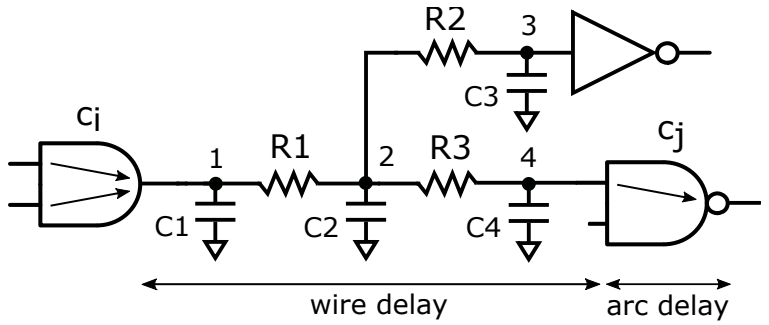
Lagrange gate sizing

$$\textbf{min}: \sum_{c_j \in cells} P(c_j) + A(c_j) - \sum_{j \in POs} slk_j^L - \sum_{j \in POs} slk_j^E$$

$$\textbf{s.t.:}\ slk_j^L \leq 0 \text{ and } slk_j^E \leq 0, \forall j \in POs$$

$$slk_j^L \leq r_j^L - a_j^L \text{ and } slk_j^E \leq a_j^E - r_j^E, \forall j \in POs$$

$$a_i^L + d_{i \to j}^L \leq a_j^L \text{ and } a_i^E + d_{i \to j}^E \geq a_j^E, \forall cells$$



- Introduce Lagrange multipliers

$$\textbf{min}:$$

$$\sum_{c_j \in cells} P(c_j) + A(c_j) - \sum_{j \in POs} slk_j^L - \sum_{j \in POs} slk_j^E +$$

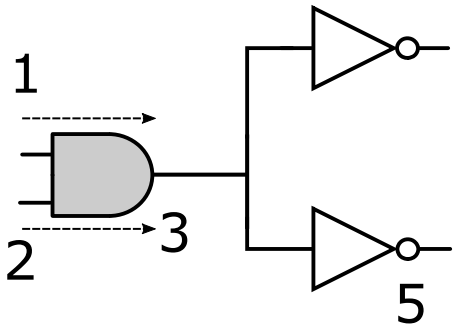
$$\sum_{j \in POs} \left(\lambda_{j0}^L slk_j^L + \lambda_{j0}^E slk_j^E \right) +$$

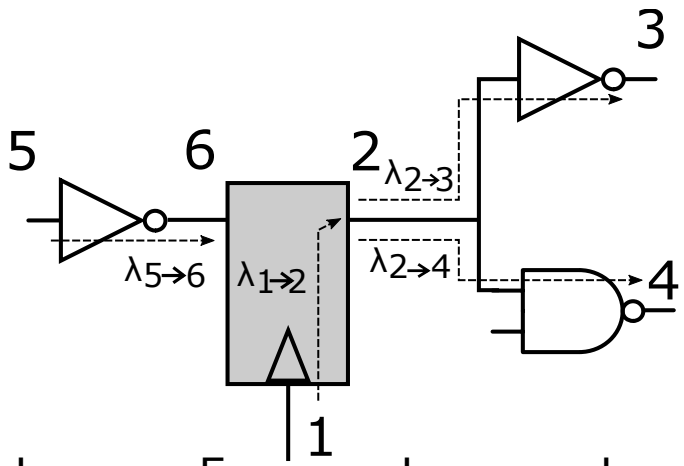
$$\sum_{j \in POs} \left(\lambda_{j1}^L (slk_j^L - r_j^L + a_j^L) + \lambda_{j1}^E (slk_j^E - a_j^E + r_j^E) \right) +$$

$$\sum_{c_j \in cells} \left(\sum_{i \in fanin_j} \lambda_{i \to j}^L (a_i^L + d_{i \to j}^L - a_j^L) + \lambda_{i \to j}^E (a_j^E - a_i^E - d_{i \to j}^E) \right)$$

$$\Rightarrow \quad \textbf{min}: \sum_{c_j \in cells} \left(\sum_{i \in fanin_j} \lambda_{i \to j}^L d_{i \to j}^L + \sum_{i \in fanin_j} \lambda_{i \to j}^E (-d_{i \to j}^E) \right)$$

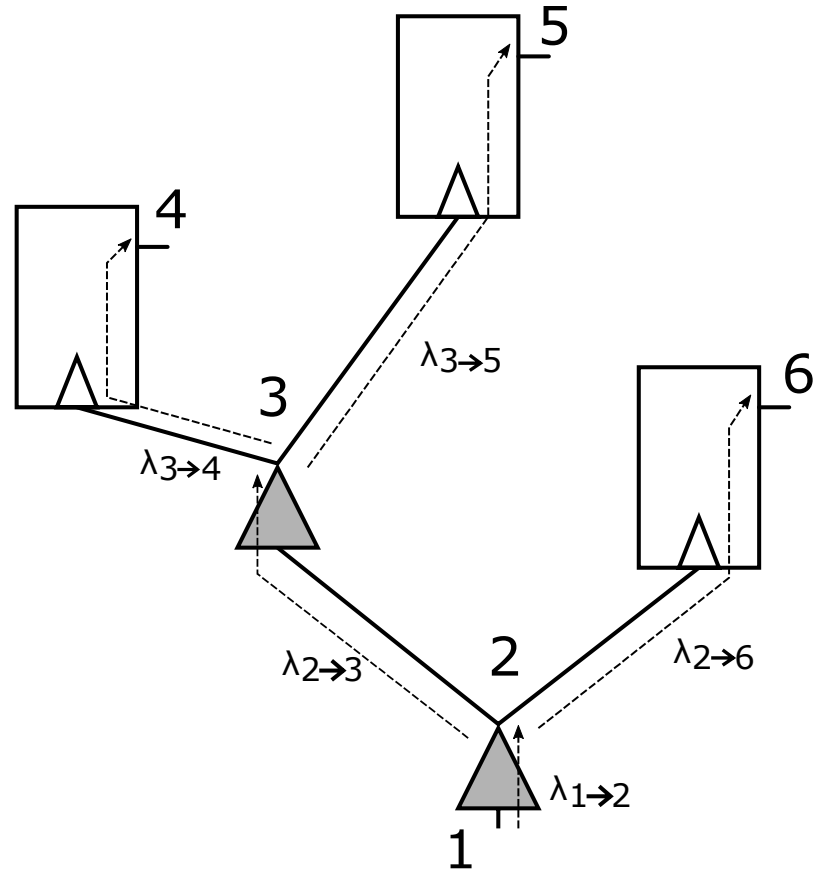
Lagrange multipliers optimality conditions





$$\lambda^{L}_{1\to2} = \lambda^{E}_{5\to6} + \lambda^{L}_{2\to3} + \lambda^{L}_{2\to4}$$
$$\lambda^{E}_{1\to2} = \lambda^{L}_{5\to6} + \lambda^{E}_{2\to3} + \lambda^{E}_{2\to4}$$

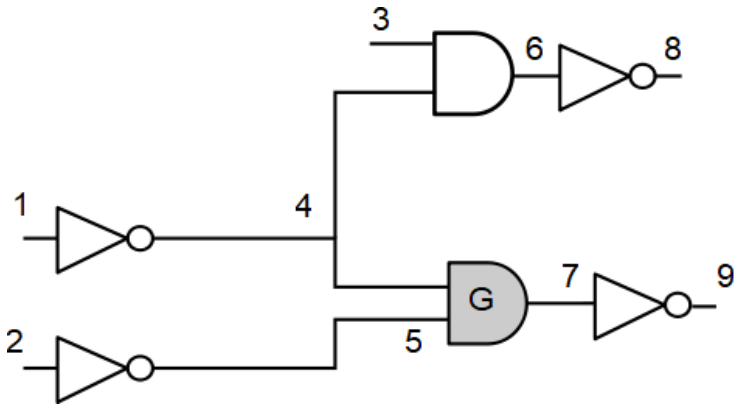
- Lagrange multipliers should be distributed to the design according to the optimality criteria
- Lagrange multipliers for FFs and Local clock buffers used for the first time in gate sizing

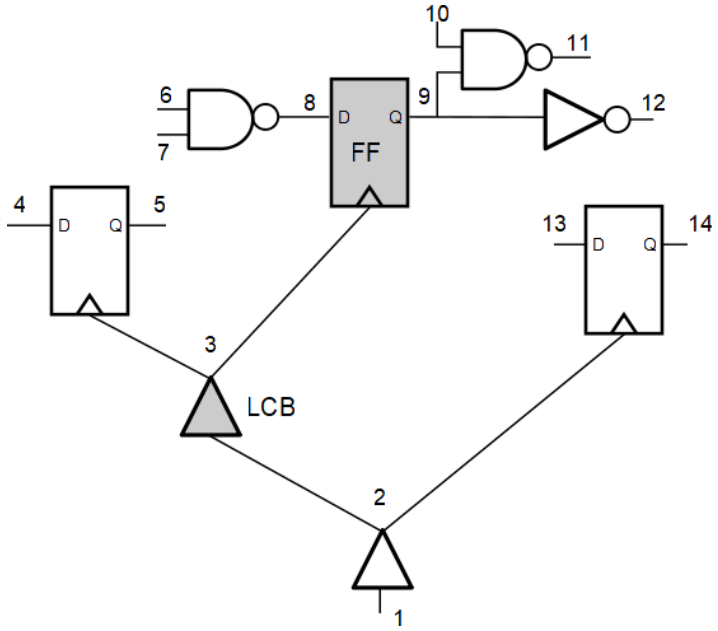


$$\lambda^{L}_{1\to2} = \lambda^{L}_{2\to6} + (\lambda^{L}_{3\to5} + \lambda^{L}_{3\to4})$$
$$\lambda^{E}_{1\to2} = \lambda^{E}_{2\to6} + (\lambda^{E}_{3\to5} + \lambda^{E}_{3\to4})$$

Lagrange gate sizing

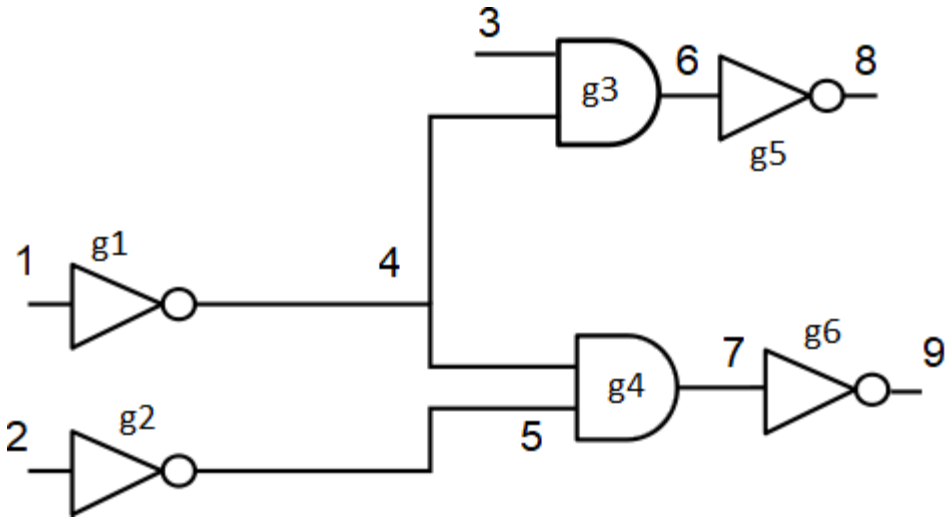
- After each resize, timing information is recalculated locally.
- For each size, the local cost function is calculated.
- Local cost function consists of:
  - Leakage power
  - λ*d value for local arcs
  - Local arcs: cell arcs, fanin arcs, fanout arcs, side arcs





Multithreaded implementation

- The cells should be resized in forward topological order.
- Each cell knows how many cells need to be resized before it.
- For cells belonging to the same logic level and share a fanin, a random decision is made.
- When a cell is resized, it notifies the cells that depend on it that it finished resizing.
- When a cell has zero dependencies, it is pushed into a ready queue, from where threads pick cells to resize.
- Example: first resize g1-g2, make a decision about g3-g4 (who to resize first), then resize g5-g6.



Buffering optimizations

- Applied on a small number of critical paths per iteration
  - Runtime kept under control
  - Buffer insertion is smoothly integrated with gate sizing
- Late buffering optimization
  - Add increasingly larger buffer sizes next to the driver of the large cap net until the ratio of the output load to the input load of each gate added locally is approximately 4 (from theory of logical effort).
- Hold Buffering at the endpoints
  - Add buffer with an input capacitance at least as large as the endpoint capacitance.
  - Ensures that extra delay is always added since the delay of the driving gate remains either the same or it is slightly increased.
- Clock buffering insertion
  - Insert additional local clock buffer on the clock pin of a FF if we need to slow down the clock signal for this register.
    - D pin late slack more critical than the Q pin late slack or
    - Q pin early slack more critical than the D pin early slack
  - We don't insert buffers if both sides are non critical.
- Buffering for critical path isolation
  - Reduce the input capacitance of the non-critical branches of a net

Timing recovery steps

- Sort all violating nets based on the number of violating endpoints present in their fanout cone
- Resize the gate that drives the net driving the most violating endpoints
- Downsize (or upsize) the gate by one size.
- Perform local timing update and calculate the new local negative slack
- If it is improved compared to the initial slack perform an incremental timing update
  - If TNS improves keep this gate version and restart
  - If TNS is not improved revert the change and move on to the next most critical net

- The algorithm stops if all timing violations are solved, if the TNS stops improving or if a certain number of incremental timing updates is reached
- This recovery steps are performed twice: once for the remaining late timing violations and for the early timing violations.

Implementation

- The proposed techniques have been integrated to RSYN physical design framework
  - Already used significantly by our research team
  - User friendly C++ API – Fast response and good memory usage
- RSYN timing engine needed a lot of tuning to support TAU contest
  - Added support for multiple corners
  - Covered not-supported timing arcs (i.e., reset pins)
  - After necessary changes results match with official timing engine of TAU contest [OpenTimer]

Conclusions

- Timing closure is a complex process that involves many iterative optimization steps applied in various phases of the physical design flow.
- The optimal order of application of the available optimization heuristics is far to be reached.
- Interleave in a fine-grained manner a Lagrange-relaxation-based gate sizing engine with multiple forms of buffering optimizations
  - Powerful extension of Lagrange-relaxation-based formulations to handle all cells (gates, FFs, LCBs) in a unified manner
  - Buffers are added gradually
  - Sizes adopt smoothly to design restructuring