Optimization of Robust Asynchronous Threshold Networks Using Local Relaxation Techniques

Cheoljoo Jeong*      Steven M. Nowick

Computer Science Department
Columbia University

*[now at Cadence Design Systems]
Outline

1. Introduction
2. Background: Asynchronous Threshold Networks
3. Gate-Level Relaxation
4. Block-Level Relaxation
5. Experimental Results
6. Conclusions and Future Work
Recent Challenges in Microelectronics Design

- **Reliability challenge**
  - Variability issues in deep submicron technology
    - process, temperature, voltage
    - noise, crosstalk
  - Dynamic voltage scaling

- **Communication challenge**
  - Increasing disparity between gate and wire delay

- **Productivity challenge**
  - Increasing system complexity + heterogeneity
  - Shrinking time to market, timing closure issues
  - Even when IP blocks are used, interface timing verification is difficult
Benefits and Challenges of Asynchronous Circuits

• **Potential benefits:**
  - Mitigates timing closure problem
  - Low power consumption
  - Low electromagnetic interference (EMI)
  - Modularity, “plug-and-play” composition
  - Accommodates timing variability

• **Challenges:**
  - Robust design is required: hazard-freedom
  - Area overhead (sometimes)
  - Lack of CAD tools
  - Lack of systematic optimization techniques
Asynchronous Threshold Networks

- Asynchronous threshold networks
  - One of the most robust asynchronous circuit styles
  - Based on delay-insensitive encoding
    - Communication: robust to arbitrary delays
    - Logic block design: imposes very weak timing constraints (1-sided)

- Simple example: OR2

Boolean OR2 gate

Async dual-rail threshold network for OR2
Asynchronous Threshold Networks in Emerging Technologies

• **Ultra-Low Voltage (ULV):** extreme variability
  - 8051 microcontroller - extreme PVT variability at subthreshold voltages

• **Space Applications:** extreme environments
  - 8-bit data transfer system for space flights
  - Fully operational over 400°C temperature range (-175°C to +225°C)

• **Nano-Magnetic Logic Circuits:**

• **Quantum-Dot Cellular Automata (QCA):**
Challenges and Overall Research Goals

• Challenges in asynchronous threshold network synthesis
  - Large area and latency overheads
  - Few existing optimization techniques
  - Even less support for CAD tools

• Overall Research Agenda:
  - Develop systematic optimization techniques and CAD tools
    for highly-robust asynchronous threshold networks
  - Support design-space exploration:
    automated scripts, target different cost functions
  - Current optimization targets: area + delay + delay-area tradeoffs
  - Future extensions: power (straightforward)
Overall Research Goals

Two automated optimization techniques proposed

1. Relaxation algorithms: multi-level optimization
   - Existing synthesis approaches are conservative = over-designed
   - Approach: selective use of eager-evaluation logic
     • without affecting overall circuit’s timing robustness
   - Can apply at two granularities:
     • gate-level  [Jeong/Nowick ASPDAC-07, Zhou/Sokolov/Yakovlev ICCAD-06]
     • block-level  [Jeong/Nowick Async-08]


2. Technology mapping algorithms
   - First general and systematic technology mapping for robust asynchronous threshold networks
   - Evaluated on substantial benchmarks:
     • > 10,000 gates, > 1000 inputs/outputs
   - Use fully-characterized industrial cell library (Theseus Logic):
     • slew rate, loading, distinct i-to-o paths/rise vs. fall transitions
   - Significant average improvements:
     • Delay: 31.6%, Area: 9.5% (runtime: 6.2 sec)


“ATN_OPT” CAD Package: implements both steps downloadable (for Linux) + tutorials/benchmarks
URL: http://www.cs.columbia.edu/~nowick/asynctools
Basic Synthesis Flow
(Theseus Logic/Camgian Networks)

Single-rail Boolean network

simple dual-rail expansion
(delay-insensitive encoding)

Dual-rail async threshold network

Considered as abstract multi-valued circuit

Instantiated Boolean circuit (robust, unoptimized)
New Optimized Synthesis Flow

Single-rail Boolean network

Relaxation
(i.e. relaxed dual-rail expansion)

“Relaxed” dual-rail async threshold network

Optimally-mapped dual-rail async threshold network

←········· optimized
Focus of this talk

**New Optimized Synthesis Flow**

**Single-rail** Boolean network

**Relaxation** (i.e. relaxed **dual-rail expansion**)

"**Relaxed**" **dual-rail** async threshold network  $\leftarrow$ ***optimized***

**Technology mapping**

**Optimally-mapped** **dual-rail** async threshold network  $\leftarrow$ ***optimized***
Outline

1. Introduction
2. Background: Asynchronous Threshold Networks
3. Gate-Level Relaxation
4. Block-Level Relaxation
5. Experimental Results
6. Conclusions and Future Work
Single-Rail Boolean Networks

- **Boolean Logic Network**: *Starting point for dual-rail circuit synthesis*
  - Modeled using **three-valued logic** with \{0, 1, NULL (N)\}
    - 0/1 = data values, NULL = no data (invalid data)
  - Computation alternates between DATA and NULL phases

- **DATA (Evaluate) phase**:
  - outputs have DATA values only after all inputs have DATA values

- **NULL (Reset) phase**:
  - outputs have NULL values only after all inputs have NULL values
Delay-Insensitive Encoding

- **Approach:**
  - **Single Boolean signal** is represented by **two wires**
  - **Goal**: map abstract Boolean netlist to robust dual-rail asynchronous circuit

- **Motivation:** robust data communication

\[
\begin{array}{c|c|c}
| a_1 | a_0 | a \\hline
| 0   | 0   | \text{NULL} \\
| 0   | 1   | 0 \\
| 1   | 0   | 1 \\
| 1   | 1   | \text{Not allowed} \\
\end{array}
\]

**Encoding table**
Dual-Rail Asynchronous Circuits

- DIMS-Style Dual-Rail Expansion:
  - "delay-insensitive minterm synthesis" style
  - **Single Boolean gate**: expanded into 2-level network

![Diagram of a Boolean OR gate and a DIMS-style dual-rail OR circuit](image-url)
Dual-Rail Asynchronous Circuits (cont.)

- **NCL-Style Dual-Rail Expansion (**Theseus Logic**):**
  - Single Boolean gate: expanded into two NCL gates
  - Allows more optimized mapping (to custom library)
Summary: Existing Synthesis Approach

- **Starting point:** single-rail abstract Boolean network (3-valued)
- **Approach:** performs dual-rail expansion of each gate
  - Use 'template-based' mapping
- **End point:** unoptimized dual-rail asynchronous threshold network
- **Result:** timing-robust asynchronous netlist
Hazard Issues

- Ideal Goal = Delay-Insensitivity (delay model)
  - Allows arbitrary gate and wire delay
    - circuit operates correctly under all conditions
  - Most robust design style
    - when circuit produces new output, all gates stable
      = “timing robustness”

- “Orphans” = hazards to delay-insensitivity
  - “unobservable” signal transition sequences
  - Wire orphans: unobservable wires at fanout
  - Gate orphans: unobservable paths at fanout
Hazard Issues

• Wire orphan example:

Wire orphan example

wire orphan! = unobservable wire transition (at fanout point)

If unobservable wire too slow, will interfere with next data item (glitch)
Hazard Issues

• Gate orphan example:

Gate orphan! = unobservable path through 1+ gates (at fanout point)

If unobservable path too slow, will interfere with next data item (glitch)
Hazard Issues: Summary

- **Wire orphans:** typically not a problem in practice
  - unobserved signal transition on wire (at fanout point)
  - **Solution:** handle during physical synthesis (e.g. Theseus Logic)
    - enforce simple 1-sided timing constraint:
      - similar to “quasi-delay-insensitivity” (QDI)

- **Gate orphans:** difficult to handle
  - unobserved signal transition on path (at fanout point)
  - can result in unexpected glitches: if delays too long
  - harder to overcome with physical design tools

**Invariant of the proposed optimization algorithms:**
ensure no gate orphans introduced
Outline

1. Introduction
2. Background: Asynchronous Threshold Networks
3. Gate-Level Relaxation
4. Block-Level Relaxation
5. Experimental Results
6. Conclusions and Future Work
Overview of Relaxation

• **Relaxation: Multi-level optimization**
  - Allows more efficient dual-rail expansion using *eager-evaluating* logic
  - Idea: *selectively replace* some gates by eager blocks
    • either at *gate-level* or *block-level*
  - Advantage: if carefully performed, *no* loss of overall circuit robustness

• **Proposed flow**

  Single-rail Boolean network

  \[\text{Relaxation}\]

  Relaxed dual-rail async threshold network \[\text{optimized}\]
A dual-rail implementation of a Boolean gate is input-complete w.r.t. its input signals if an output changes only after all the inputs arrive.

Enforcing input completeness for every gate is the traditional synthesis approach to avoid hazards (i.e. gate orphans).
A dual-rail implementation of a Boolean gate is input-incomplete w.r.t. its input signals ("eager-evaluating"), if the output can change before all inputs arrive.

Boolean OR gate

Input-incomplete dual-rail OR network
Gate-Level Relaxation Example #1

- Existing approach to dual-rail expansion is too restrictive.
  - Every Boolean gate is fully-expanded into an input-complete block.
Gate-Level Relaxation Example #1 (cont.)

- Not every Boolean gate needs to be expanded into input-complete block.

Boolean network

Relaxed expansion

Robust expansion

Optimized dual-rail circuit is still timing-robust (gate-orphan-free)
Gate-Level Relaxation Example #2

- Different choices may exist in relaxation.

Relaxation of Boolean network with two relaxed gates
• Different choices may exist in relaxation.

Relaxation of Boolean network with four relaxed gates
Gate-Level Relaxation: Summary

- **Conservative approach:**
  - Every path from a gate to a primary output must contain only robust (input-complete) gates

- **Optimized approach:** [Nowick/Jeong ASPDAC-07, Zhou/Sokolov/Yakovlev ICCAD-06]
  - At least one path from each gate to some primary output must contain only robust (i.e. input-complete) gates (Theorem)
  - ... all other gates can be safely ‘relaxed’ (i.e. input-incomplete)

Resulting implementation has no loss of timing robustness (remains “gate-orphan-free”)
Which Gates Can Safely Be Relaxed?

- Localized theorem: gate relaxation [Jeong/Nowick ASPDAC-07]
  A dual-rail implementation of a Boolean network is timing-robust (i.e. gate-orphan-free) if and only if, for each signal, at least one of its fanout gates is input-complete (i.e. not relaxed).

- Example:

  ![Boolean network diagram]

  Boolean network
Which Gates Can Safely Be Relaxed?

- Localized theorem: gate relaxation [Jeong/Nowick ASPDAC-07]
  A dual-rail implementation of a Boolean network is timing-robust (i.e. gate-orphan-free) if and only if, for each signal, at least one of its fanout gates is input-complete (i.e. not relaxed).

- Example:

  ![Boolean network diagram](image)

  Two fanout gates for signal a
Which Gates Can Safely Be Relaxed?

• Localized theorem: [Jeong/Nowick ASPDAC-07]
  Dual-rail implementation of a Boolean network is timing-robust (i.e. gate-orphan-free) if and only if, for each signal, at least one of its fanout gates is input complete (i.e. not relaxed).

• Example:

  ![Diagram of a Boolean network](image)

  Boolean network
  Two fanout gates for signal a
  Only one of two fanout gates must be input-complete.
Problem Definition

• The Input Completeness Relaxation Problem
  – Input: single-rail Boolean logic network
  – Output: relaxed dual-rail asynchronous circuit, which is still timing-robust

• Overview of the Proposed Algorithm
  – Relaxes overly-restrictive style of existing approaches
    • Performs selective relaxation of individual nodes
  – Targets three cost functions:
    • Number of relaxed-gates
    • Area after dual-rail expansion
    • Critical path delay
  – Based on unate covering framework:
    • Each gate output must be covered by at least one fanout gate.
Relaxation Algorithm

• Algorithm Sketch
  – Step 1: setup covering table
    • For each pair \(<u, v\rangle\), signal \(u\) fed into gate \(v\):
      – Add \(u\) as a covered element (row)
      – Add \(v\) as a covering element (column)
  – Step 2: solve unate covering problem
  - Step 3: generate dual-rail threshold network

![Boolean network and covering table]

- **gates**
- **signals**
- **covering table**
Targeting Different Cost Functions

• **Maximization of Number of Relaxed Gates**
  - Weight of a gate = 1

• **Minimization of Area of Dual-Rail Circuit**
  - Weight of a gate = area penalty for not relaxing the gate

• **Critical Path Delay Optimization in Dual-Rail Circuit**
  - Find a critical path in non-relaxed dual-rail circuit
  - Assign higher weights to critical gates
  - Assign lower weights to non-critical gates
  - **GOAL:** more relaxation of critical path gates
Outline

1. Introduction
2. Background: Asynchronous Threshold Networks
3. Gate-Level Relaxation
4. Block-Level Relaxation
5. Experimental Results
6. Conclusions and Future Work
Block-Level Relaxation

- Block-level vs. Gate-level circuits

<table>
<thead>
<tr>
<th>Block-level circuit</th>
<th>Gate-level circuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consists of large granularity blocks</td>
<td>Consists of simple gates</td>
</tr>
<tr>
<td>Blocks have <em>multiple</em> outputs</td>
<td>Gates have <em>single</em> output</td>
</tr>
</tbody>
</table>

P/G block in prefix adders

Gate-level implementation of P/G block
Why Relaxation at Block-Level?

- Like gate-level relaxation: blocks are either
  - **input complete**: wait for all inputs to arrive
  - **relaxed**: eager, do not wait for all inputs to arrive

- New idea: 3rd possibility
  - “**partially-eager**”:
    - **input complete**: each input vector acknowledged on some output
    - **partially-eager**: allows some outputs to fire early
• Basic approach = direct extension of gate-level relaxation
  - No output in robust block fires before all inputs arrive

\[ z = a + b + c \]
\[ w = abc \]

Block example
Block-Level Relaxation Example

- Basic approach = direct extension of gate-level relaxation
  - No output in robust block fires before all inputs arrive

Input-complete (non-eager)

\[ z = a + b + c \]
\[ w = abc \]

Input-incomplete (eager)
Block-Level Relaxation Example

- **New Option #1: “Biased Approach”**
  - In biased implementation of blocks, **only one output** is implemented in a robust way; other outputs are eager-evaluating.

\[ z = a + b + c \]
\[ w = abc \]

**Input-complete block (and partially eager!)**

**Block example**

Output **z**: waits for all inputs ("non-eager")
Output **w**: early evaluating ("eager")
Block-Level Relaxation Example

• New Option #2: “Distributive Approach”
  • outputs jointly share responsibility to detect arrival of all input vectors
  • each block output: also partially “eager”!

Input-complete block (and partially eager!)

Block example

Output \( z \): waits for inputs \( a/b \) (otherwise eager)
Output \( w \): waits for inputs \( b/c \) (otherwise eager)
Summary: Why Relaxation at Block-Level?

<table>
<thead>
<tr>
<th>Gate-level relaxation</th>
<th>Single Boolean gate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input-complete dual-rail impl. (non-eager)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block-level relaxation (NEW)</th>
<th>Single Boolean block</th>
</tr>
</thead>
</table>

More optimization opportunities + larger design space
Block- vs Gate-Level Relaxation Example

- Gate-level relaxation example

Gate-level 8-bit Brent-Kung adder circuit (Initial Boolean network)
Block- vs Gate-Level Relaxation Example

- Gate-level relaxation example

Gate-level 8-bit Brent-Kung adder circuit w/ relaxed gates marked
Block- vs Gate-Level Relaxation Example

• Block-level relaxation example

Block-level 8-bit Brent-Kung adder circuit (Initial Boolean network)
Block- vs Gate-Level Relaxation Example

- Block-level relaxation example

Block-level 8-bit Brent-Kung adder circuit w/ relaxed blocks marked
Outline

1. Introduction
2. Background: Asynchronous Threshold Networks
3. Gate-Level Relaxation
4. Block-Level Relaxation
5. Experimental Results
6. Conclusions and Future Work
### Experimental Results: Gate-Level Relaxation

- Results for DIMS-style asynchronous circuits

<table>
<thead>
<tr>
<th>Original Boolean network</th>
<th>Unoptimized DIMS circuit</th>
<th>Optimization Run</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#i/#o/#g</td>
<td># relaxed nodes min.</td>
</tr>
<tr>
<td>C1908</td>
<td>33/25/462</td>
<td>343</td>
</tr>
<tr>
<td>C3540</td>
<td>50/22/1147</td>
<td>911</td>
</tr>
<tr>
<td>C5315</td>
<td>178/123/1659</td>
<td>1259</td>
</tr>
<tr>
<td>C6288</td>
<td>32/32/3201</td>
<td>2385</td>
</tr>
<tr>
<td>C7552</td>
<td>207/108/2155</td>
<td>1677</td>
</tr>
<tr>
<td>dalu</td>
<td>75/16/756</td>
<td>633</td>
</tr>
<tr>
<td>des</td>
<td>256/245/2762</td>
<td>2329</td>
</tr>
<tr>
<td>K2</td>
<td>45/43/684</td>
<td>597</td>
</tr>
<tr>
<td>t481</td>
<td>16/1/510</td>
<td>476</td>
</tr>
<tr>
<td>vda</td>
<td>17/39/383</td>
<td>309</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Average percentage</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>51.8%</td>
<td>65.1%</td>
<td>83.9%</td>
</tr>
</tbody>
</table>
Experimental Results: Gate-Level Relaxation

- Results for NCL asynchronous circuits
  – *(style used at Theseus Logic)*

<table>
<thead>
<tr>
<th>Original Boolean network</th>
<th>NCL circuit</th>
<th>Optimization Run</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># full blocks</td>
<td>area</td>
</tr>
<tr>
<td>name</td>
<td># i/#o/#g</td>
<td></td>
</tr>
<tr>
<td>C1908</td>
<td>33/25/462</td>
<td>343</td>
</tr>
<tr>
<td>C3540</td>
<td>50/22/1147</td>
<td>911</td>
</tr>
<tr>
<td>C5315</td>
<td>178/123/1659</td>
<td>1259</td>
</tr>
<tr>
<td>C6288</td>
<td>32/32/3201</td>
<td>2385</td>
</tr>
<tr>
<td>C7552</td>
<td>207/108/2155</td>
<td>1677</td>
</tr>
<tr>
<td>dalu</td>
<td>75/16/756</td>
<td>633</td>
</tr>
<tr>
<td>des</td>
<td>256/245/2762</td>
<td>2329</td>
</tr>
<tr>
<td>K2</td>
<td>45/43/684</td>
<td>597</td>
</tr>
<tr>
<td>t481</td>
<td>16/1/510</td>
<td>476</td>
</tr>
<tr>
<td>vda</td>
<td>17/39/383</td>
<td>309</td>
</tr>
</tbody>
</table>

Average percentage

51.8%  74.1%  82.3%

*(selected MCNC combinational benchmarks)*
Experimental Results: Gate-Level Relaxation

• Minimizing Number of Relaxed Nodes:
  – DIMS circuits: 48.2% relaxed
  – NCL circuits: 48.2% relaxed

• Area minimization:
  – DIMS circuits: 34.9% improvement
  – NCL circuits: 25.9% improvement

• Critical Path Delay optimization:
  – DIMS circuits: 16.1% improvement
  – NCL circuits: 17.7% improvement

No change to overall timing-robustness of circuits
## Experimental Results: Block-Level Relaxation

### Experiment #2: Gate-level vs. Block-level relaxation

- Evaluation on several arithmetic circuits:
  - Brent-Kung/Kogge-Stone adders, combinational multipliers
- Block-relaxation had 8.8% better delay with 10.8% worse area (avg.), compared to gate-level relaxation

<table>
<thead>
<tr>
<th>Original Boolean network</th>
<th>Relaxed gate-level dual-rail circuit</th>
<th>Relaxed block-level dual-rail circuit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name</td>
<td>#i/#o/#g</td>
</tr>
<tr>
<td>8-b Brent-Kung</td>
<td>32/18/49</td>
<td>4688.6</td>
</tr>
<tr>
<td>16-b Brent-Kung</td>
<td>4/34/110</td>
<td>10396.8</td>
</tr>
<tr>
<td>8-b Kogge-Stone</td>
<td>32/18/67</td>
<td>6341.8</td>
</tr>
<tr>
<td>16-b Kogge-Stone</td>
<td>64/34/179</td>
<td>16571.5</td>
</tr>
<tr>
<td>8-b unopt. mult</td>
<td>32/16/323</td>
<td>28828.4</td>
</tr>
<tr>
<td>16-b unopt. mult</td>
<td>64/32/1411</td>
<td>125915.0</td>
</tr>
<tr>
<td>8-b opt. mult</td>
<td>32/16/320</td>
<td>28523.1</td>
</tr>
<tr>
<td>16-b opt. mult</td>
<td>64/32/1408</td>
<td>125610.0</td>
</tr>
<tr>
<td>Average percentage</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experimental Results (cont.)

Experiment #2: Gate-level vs. Block-level relaxation

- Block-relaxation had 8.8% better delay with 10.8% worse area (avg.), compared to gate-level relaxation
- For 16-bit multiplier, 29.5% delay improvement

<table>
<thead>
<tr>
<th>name</th>
<th>#i/#o/#g</th>
<th>area</th>
<th>critical delay</th>
<th>area</th>
<th>critical delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-b Brent-Kung</td>
<td>32/18/49</td>
<td>4688.6</td>
<td>7.48</td>
<td>6094.1</td>
<td>6.64</td>
</tr>
<tr>
<td>16-b Brent-Kung</td>
<td>4/34/110</td>
<td>10396.8</td>
<td>10.69</td>
<td>13587.8</td>
<td>9.65</td>
</tr>
<tr>
<td>8-b Kogge-Stone</td>
<td>32/18/67</td>
<td>6341.8</td>
<td>5.57</td>
<td>9624.9</td>
<td>5.84</td>
</tr>
<tr>
<td>16-b Kogge-Stone</td>
<td>64/34/179</td>
<td>16571.5</td>
<td>6.99</td>
<td>22596.4</td>
<td>7.57</td>
</tr>
<tr>
<td>8-b unopt. mult</td>
<td>32/16/323</td>
<td>28828.4</td>
<td>25.69</td>
<td>24998.4</td>
<td>23.52</td>
</tr>
<tr>
<td>16-b unopt. mult</td>
<td>64/32/1411</td>
<td>125915.0</td>
<td>55.87</td>
<td>108728.0</td>
<td>52.29</td>
</tr>
<tr>
<td>8-b opt. mult</td>
<td>32/16/320</td>
<td>28523.1</td>
<td>20.98</td>
<td>24745.0</td>
<td>15.44</td>
</tr>
<tr>
<td>16-b opt. mult</td>
<td>64/32/1408</td>
<td>125610.0</td>
<td>46.70</td>
<td>108474.0</td>
<td>32.97</td>
</tr>
</tbody>
</table>

Average percentage: 110.8% for gate-level, 91.2% for block-level.
### Experimental Results (cont.)

**Experiment #2: Gate-level vs. block-level relaxation**

- Block-relaxation had **8.8% better delay** with **10.8% worse area** (avg.), compared to gate-level relaxation
- For 16-bit multiplier, **29.5% delay improvement**
- For multipliers, **14.5% smaller area**, on average

<table>
<thead>
<tr>
<th>Original Boolean network</th>
<th>Relaxed gate-level dual-rail circuit</th>
<th>Relaxed block-level dual-rail circuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>#i/#o/#g</td>
<td>area</td>
</tr>
<tr>
<td>8-b Brent-Kung</td>
<td>32/18/49</td>
<td>4688.6</td>
</tr>
<tr>
<td>16-b Brent-Kung</td>
<td>4/34/110</td>
<td>10396.8</td>
</tr>
<tr>
<td>8-b Kogge-Stone</td>
<td>32/18/67</td>
<td>6341.8</td>
</tr>
<tr>
<td>16-b Kogge-Stone</td>
<td>64/34/179</td>
<td>16571.5</td>
</tr>
<tr>
<td>8-b unopt. mult</td>
<td>32/16/323</td>
<td>28828.4</td>
</tr>
<tr>
<td>16-b unopt. mult</td>
<td>64/32/1411</td>
<td>125915.0</td>
</tr>
<tr>
<td>8-b opt. mult</td>
<td>32/16/320</td>
<td>28523.1</td>
</tr>
<tr>
<td>16-b opt. mult</td>
<td>64/32/1408</td>
<td>125610.0</td>
</tr>
<tr>
<td><strong>Average percentage</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

• Local Relaxation Technique:
  – Optimization technique for robust asynchronous threshold circuits
  – Relaxes overly-restrictive style: selective use of “eager evaluation”
  – Can target three different cost functions:
    • # relaxed nodes, area, critical path delay
  – CAD tool developed/released: “ATN-OPT”
  – Gate-level relaxation: exhibits significant improvements
    • 48.2% of gates relaxed (avg.)
    • 25.9% area improvement (vs. NCL custom mapping)
    • 17.7% delay improvement (vs. NCL custom mapping)
  – Block-level relaxation:
    • 8.8% additional delay improvement (best: 29.5%)
    • 10.8% additional area overhead (best: 14.5% reduction)

No change to overall timing-robustness of circuits