Block-Level Relaxation for Timing-Robust Asynchronous Circuits Based on Eager Evaluation

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
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 [NEW]
2. Technology mapping algorithms

- First general and systematic technology mapping for robust asynchronous threshold networks
  [Jeong/Nowick Async-06, IEEE Trans. On CAD (April 2008)]

- Evaluated on substantial benchmarks:
  - > 10,000 gates, > 1000 inputs/outputs
  - Industrial (Theseus Logic): DES, GCD
  - Academic: large MCNC circuits

- Use fully-characterized industrial cell library (Theseus Logic):
  - slew rate, loading, distinct i-to-o paths/rise vs. fall transitions

- Advanced technique: area optimization under hard delay constraints

- Significant average improvements:
  - Delay: 31.6%, Area: 9.5% (runtime: 6.2 sec)

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

**Single-rail** Boolean network

| Considered as 
<table>
<thead>
<tr>
<th>abstract multi-valued circuit</th>
</tr>
</thead>
</table>

| simple dual-rail expansion |
| (delay-insensitive encoding) |

**Dual-rail** async threshold network

<table>
<thead>
<tr>
<th>Instantiated Boolean circuit (robust, unoptimized)</th>
</tr>
</thead>
</table>
New Optimized Synthesis Flow

1. Single-rail Boolean network
2. Relaxation (i.e. relaxed dual-rail expansion)
   - “Relaxed” dual-rail async threshold network \( \leftarrow \) \textit{optimized}
3. Technology mapping
   - Optimally-mapped dual-rail async threshold network \( \leftarrow \) \textit{optimized}
Focus of this paper

Single-rail Boolean network

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

“Relaxed” dual-rail async threshold network \(\Rightarrow\) \(\text{optimized}\)

Technology mapping

Optimally-mapped dual-rail async threshold network \(\Rightarrow\) \(\text{optimized}\)
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*
  - Modelled using **three-valued logic** with \{0, 1, NULL\}
    - 0/1 = data values,  NULL = no data (invalid data)
  - Computation alternates between **DATA and NULL phases**

![Boolean OR gate](image)

- **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
Single Boolean gate: expanded into dual-rail network
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
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

- **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
  - Relaxation
  - Relaxed dual-rail async threshold network →*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.
• Not every Boolean gate needs to be expanded into input-complete block.

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
Gate-Level Relaxation Example #2 (cont.)

- 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:

  ![Dual-rail implementation diagram](attachment:diagram.png)  
  **Boolean network**
  **Two fanout gates for signal a**
  **Only one of two fanout gates must be input-complete.**
Gate-Level Relaxation Algorithm

- Gate-level relaxation based on unate covering
  - **Step 1: setup covering table**
    - Captures requirements on which gates cannot be relaxed
    - For each pair \(<u, v>\), 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**
    - Picked gates: expanded into *input-complete* block
    - Other gates: expanded into *input-incomplete* block
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

- $((g_l, p_l), (g_r, p_r))$
- $((g_{out}, p_{out}))$

### Gate-level implementation of P/G block

- $g_l, g_r, p_l, p_r$
- $g_{out}, p_{out}$
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
Block-Level Relaxation Example

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

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

Input-complete (non-eager)
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)

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.

\[
\begin{align*}
z &= a + b + c \\
w &= abc
\end{align*}
\]

**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!)

```
a b c
```

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

**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>Block-level relaxation (NEW)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single Boolean gate</strong></td>
<td><strong>Single Boolean block</strong></td>
</tr>
<tr>
<td>Input-complete dual-rail impl. (non-eager)</td>
<td>Input-complete dual-rail impl. (non-eager)</td>
</tr>
<tr>
<td>Input-complete dual-rail impl. (eager)</td>
<td>Input-complete dual-rail impl. (partially-eager)</td>
</tr>
<tr>
<td>Input-incomplete dual-rail impl. (eager)</td>
<td>Input-incomplete dual-rail impl. (eager)</td>
</tr>
</tbody>
</table>

More optimization opportunities + larger design space
Block-Level Relaxation Algorithm

• Sketch:
  - **Step #1: set up covering table**
    • Captures requirements on which gates cannot be relaxed
  - **Step #2: solve “unate covering problem”**
  - **Step #3: generate dual-rail threshold network**
    • Picked block: expanded into *input-complete* dual-rail logic
      - Pick "most desirable" input-complete impltn. from several choices
      - e.g. for full-adder block in ripple-carry adder,
        pick biased dual-rail logic which is eager w.r.t. cout
    • Other blocks: expanded into *input-incomplete* dual-rail logic
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
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

Experiment #1: Effectiveness of block-level relaxation

Block-level synchronous (Boolean) arithmetic circuit

dual-rail mapping without block-level relaxation

dual-rail mapping with block-level relaxation

Unoptimized dual-rail arithmetic circuit

Relaxed dual-rail arithmetic circuit

compared
### Experimental Results (cont.)

**Experiment #1: Effectiveness of block-level relaxation**

- **13.1% delay reduction** (avg.)
- **27.2% area improvement** (avg.)

<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>Original block-level network</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-b Brent-Kung</td>
<td>32/18/49</td>
<td>9020.2</td>
<td>8.45</td>
<td>6094.1</td>
<td>6.64</td>
</tr>
<tr>
<td>16-b Brent-Kung</td>
<td>4/34/110</td>
<td>21599.9</td>
<td>12.19</td>
<td>13587.8</td>
<td>9.65</td>
</tr>
<tr>
<td>8-b Kogge-Stone</td>
<td>32/18/67</td>
<td>16208.6</td>
<td>7.68</td>
<td>9624.9</td>
<td>5.84</td>
</tr>
<tr>
<td>16-b Kogge-Stone</td>
<td>64/34/179</td>
<td>44916.0</td>
<td>13.36</td>
<td>22596.4</td>
<td>7.57</td>
</tr>
<tr>
<td>8-b unopt. mult</td>
<td>32/16/323</td>
<td>29231.2</td>
<td>25.01</td>
<td>24998.4</td>
<td>23.52</td>
</tr>
<tr>
<td>16-b unopt. mult</td>
<td>64/32/1411</td>
<td>126786.0</td>
<td>53.78</td>
<td>108728.0</td>
<td>52.29</td>
</tr>
<tr>
<td>8-b opt. mult</td>
<td>32/16/320</td>
<td>28984.4</td>
<td>17.66</td>
<td>24745.0</td>
<td>15.44</td>
</tr>
<tr>
<td>16-b opt. mult</td>
<td>64/32/1408</td>
<td>126538.0</td>
<td>37.02</td>
<td>108474.0</td>
<td>32.97</td>
</tr>
<tr>
<td><strong>Average percentage</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>72.8%</strong></td>
<td><strong>86.9%</strong></td>
</tr>
</tbody>
</table>

**dual-rail circuit**
Experimental Results (cont.)

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

**Gate-level** synchronous (Boolean) arithmetic circuit

**Block-level** synchronous (Boolean) arithmetic circuit

- Dual-rail mapping w/ gate-level relaxation
- Dual-rail mapping w/ block-level relaxation

Relaxed dual-rail arithmetic circuit compared

Relaxed dual-rail arithmetic circuit
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

<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 #i/#o/#g</td>
<td>area</td>
</tr>
<tr>
<td></td>
<td></td>
<td></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>
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>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>110.8%</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**
- 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><strong>28828.4</strong></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>
Conclusions and Future Work

• **Block-Level Relaxation**
  - Optimization technique for robust "asynchronous" circuits
  - Relaxes overly-restrictive style of existing approaches
  - More relaxation opportunities than gate-level relaxation
  - Comparison to existing gate-level relaxation:
    • Average delay improvement of up to 8.8% (best: 29.5%)
    • Average area overhead of 10.8% (best: 14.5% reduction)
  
  No change to overall timing-robustness of circuits

• **Future Work**
  - Hybrid scheme that combines gate-level and block-level relaxation techniques