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

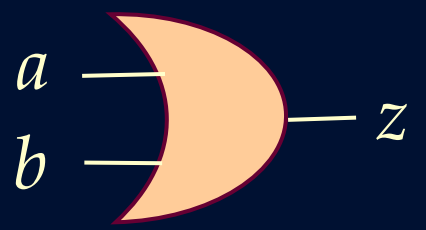
Benefits and Challenges of Asynchronous Circuits

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

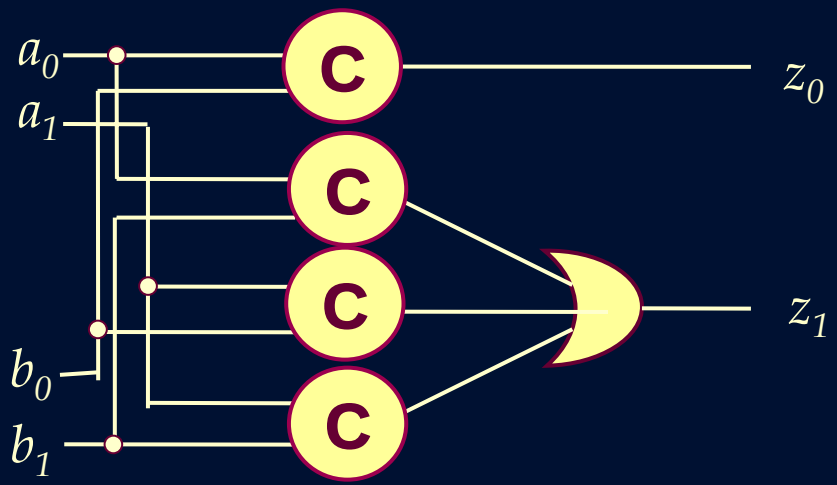
- Challenges:
  - Robust design is required: hazard-freedom
  - Area overhead (sometimes)
  - Lack of CAD tools
  - Lack of systematic optimization techniques

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]

Overall Research Goals (cont.)

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

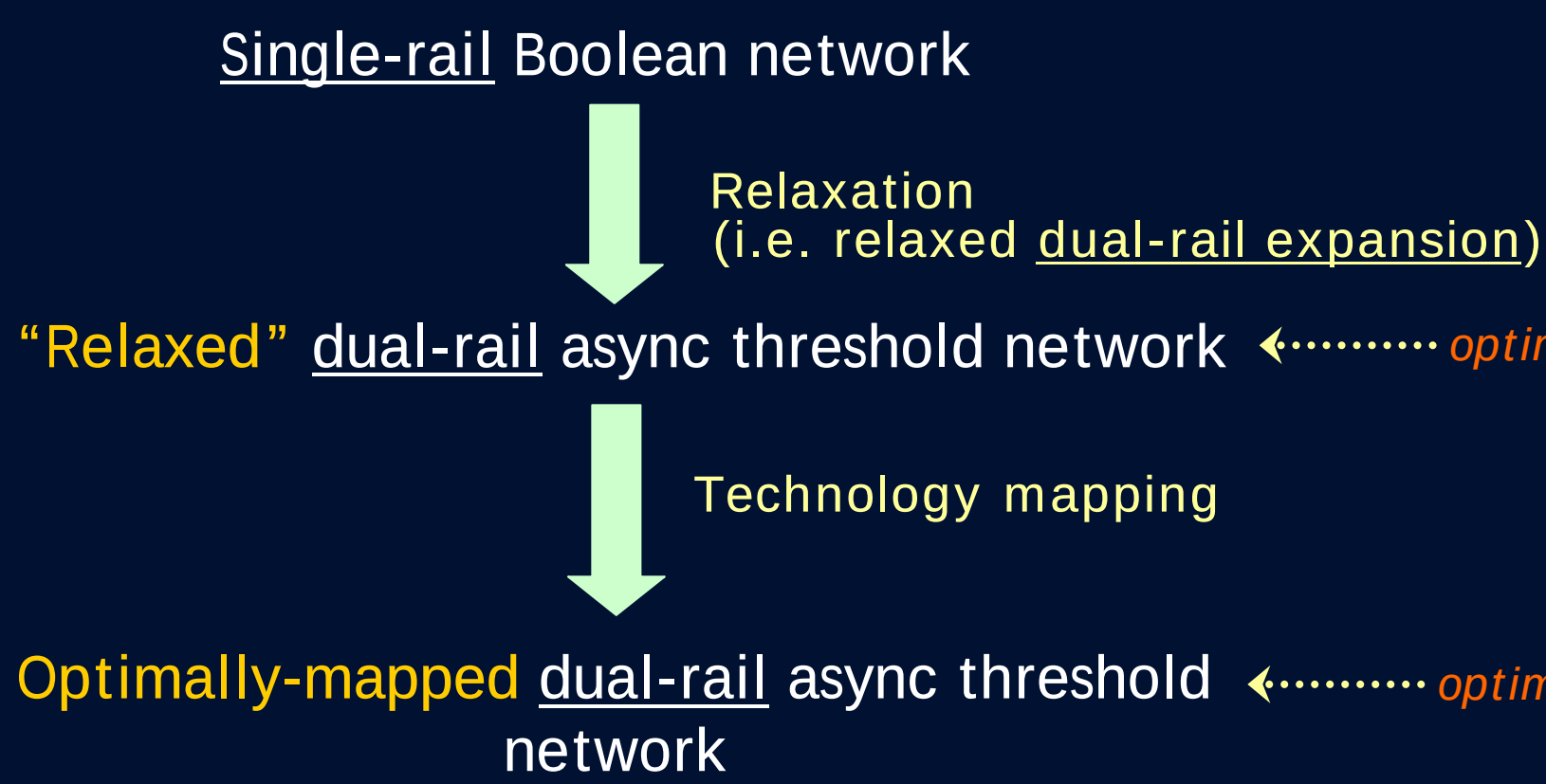
Basic Synthesis Flow
(Theseus Logic/Camgian Networks)

Single-rail Boolean network ⟵ *Considered as abstract multi-valued circuit*

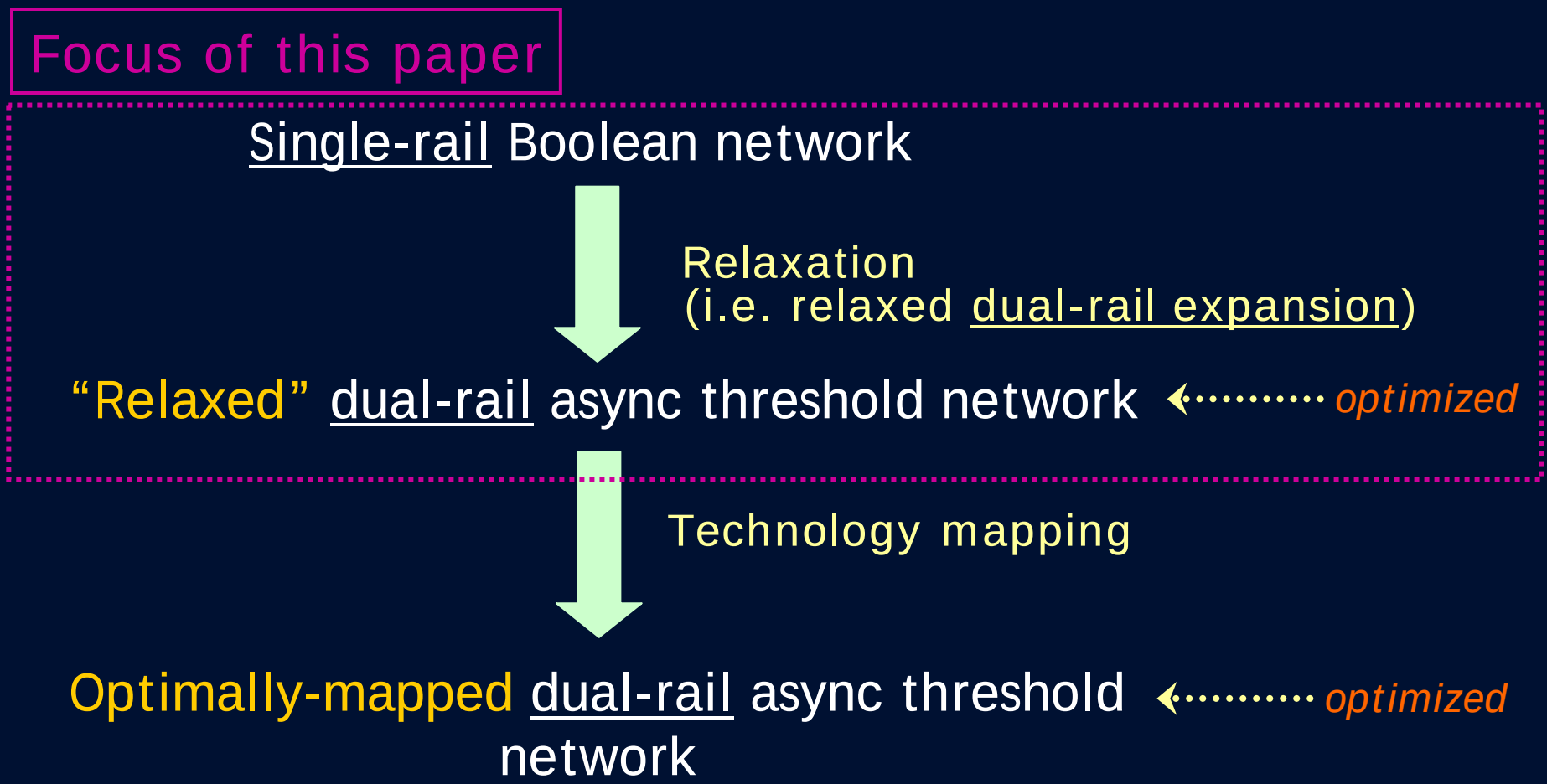
simple dual-rail expansion
(delay-insensitive encoding)

Dual-rail async threshold network ⟵ *Instantiated Boolean circuit (robust, unoptimized)*

New Optimized Synthesis Flow



New Optimized Synthesis Flow

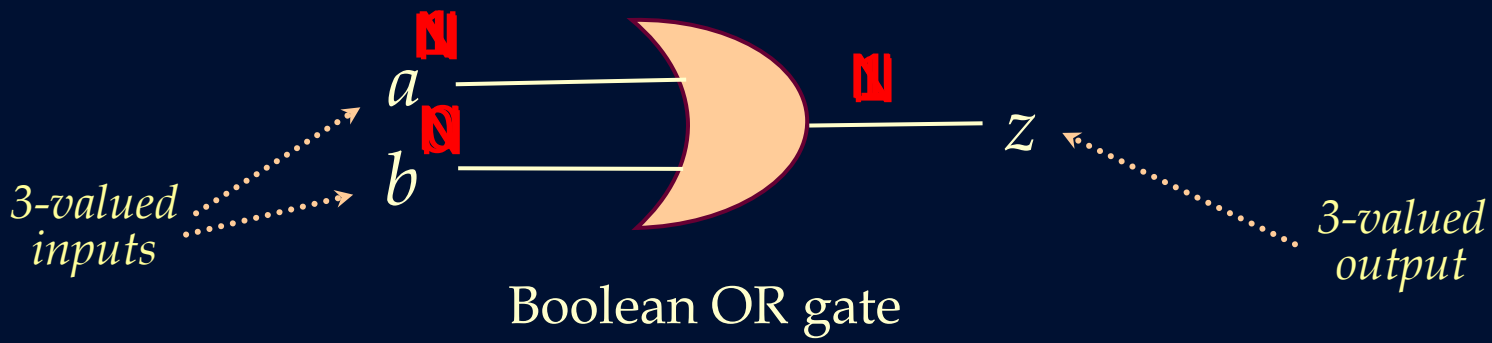


Outline

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*

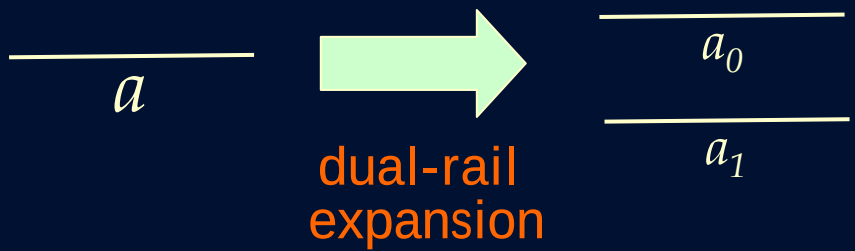


Boolean OR gate

  - 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



a_1	a_0	a
0	0	NULL
0	1	0
1	0	1
1	1	*Not allowed*

spacer

valid data

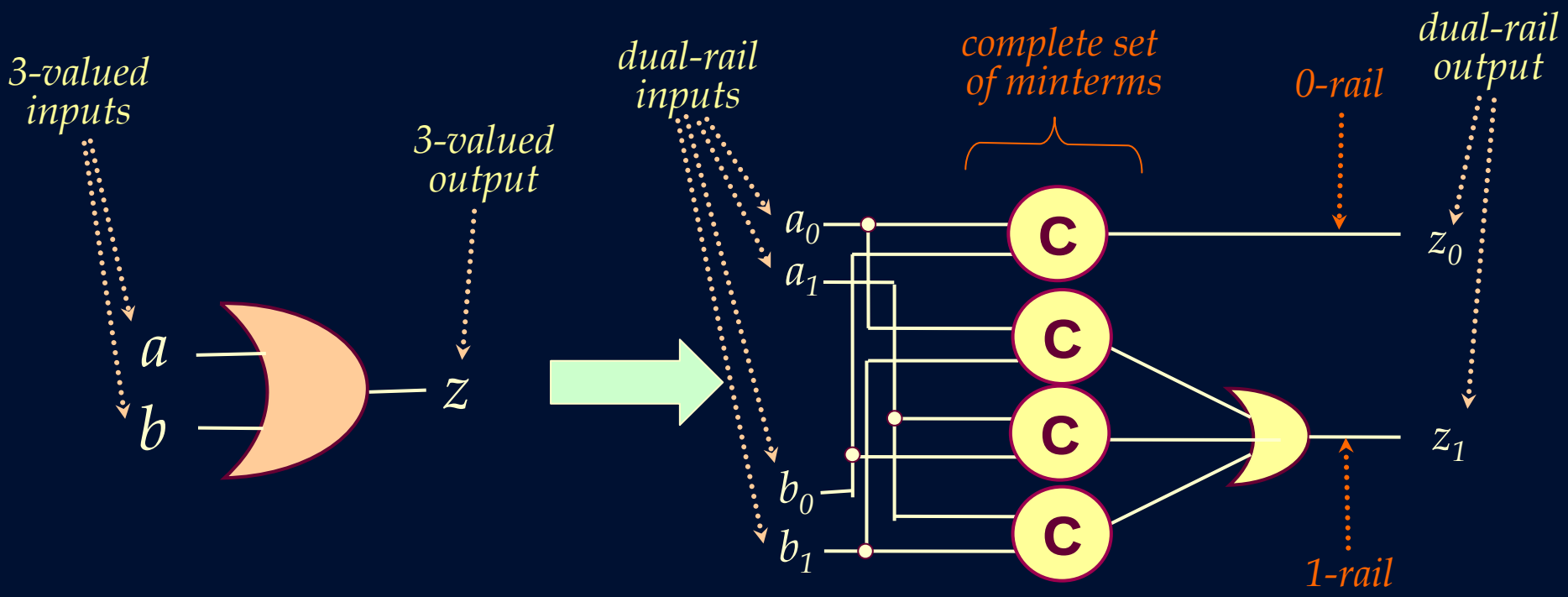
invalid

Encoding table

- Motivation: robust data communication

Dual-Rail Expansion

Single Boolean gate: expanded into dual-rail network

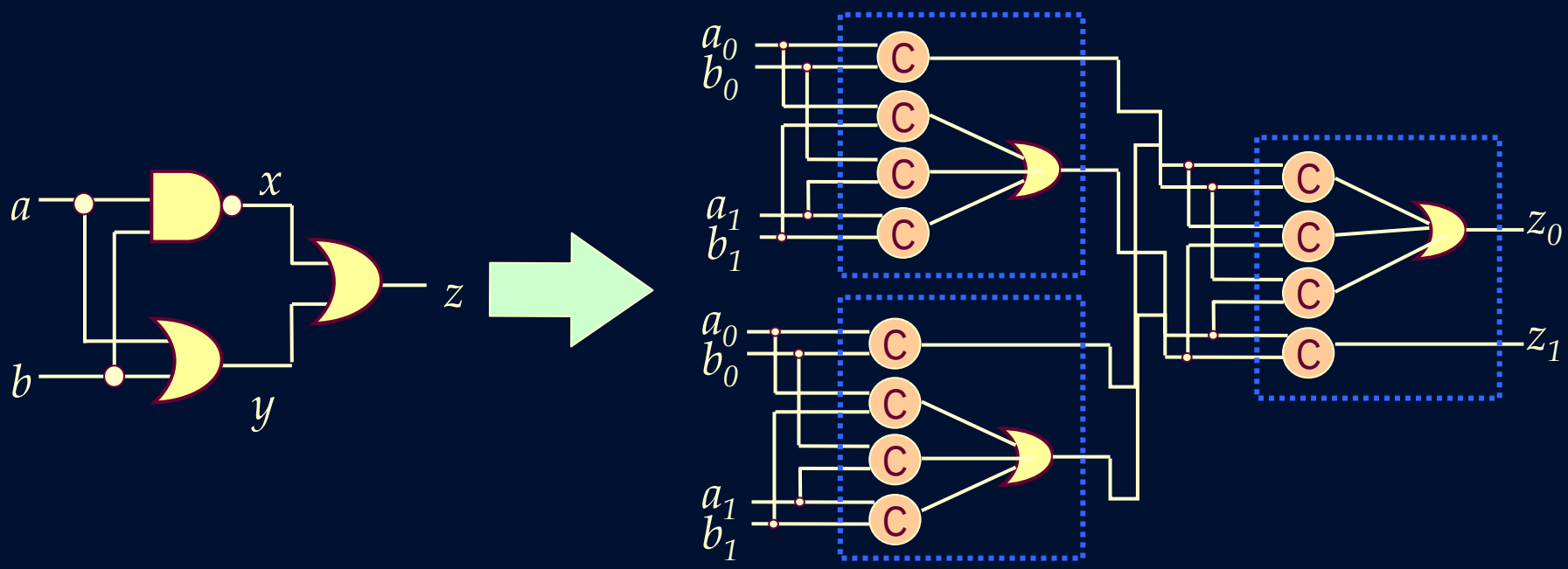


Boolean OR gate

"DIMS"-style dual-rail OR circuit

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



Boolean logic network

Dual-rail asynchronous threshold network

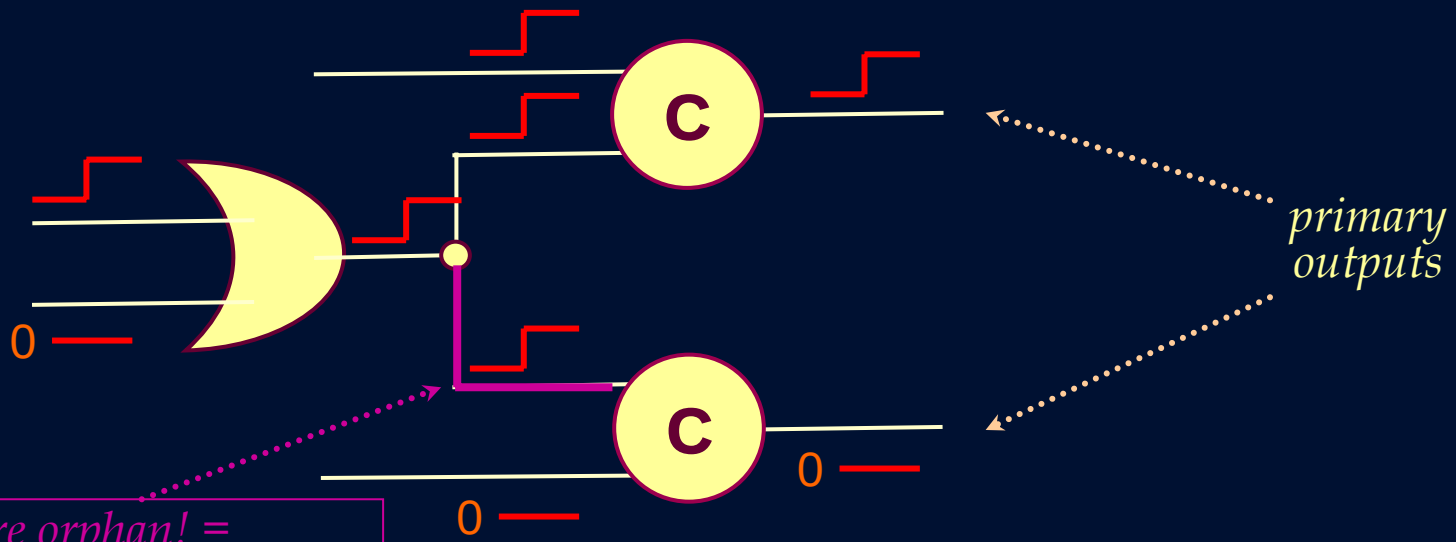
Hazard Issues

- 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:

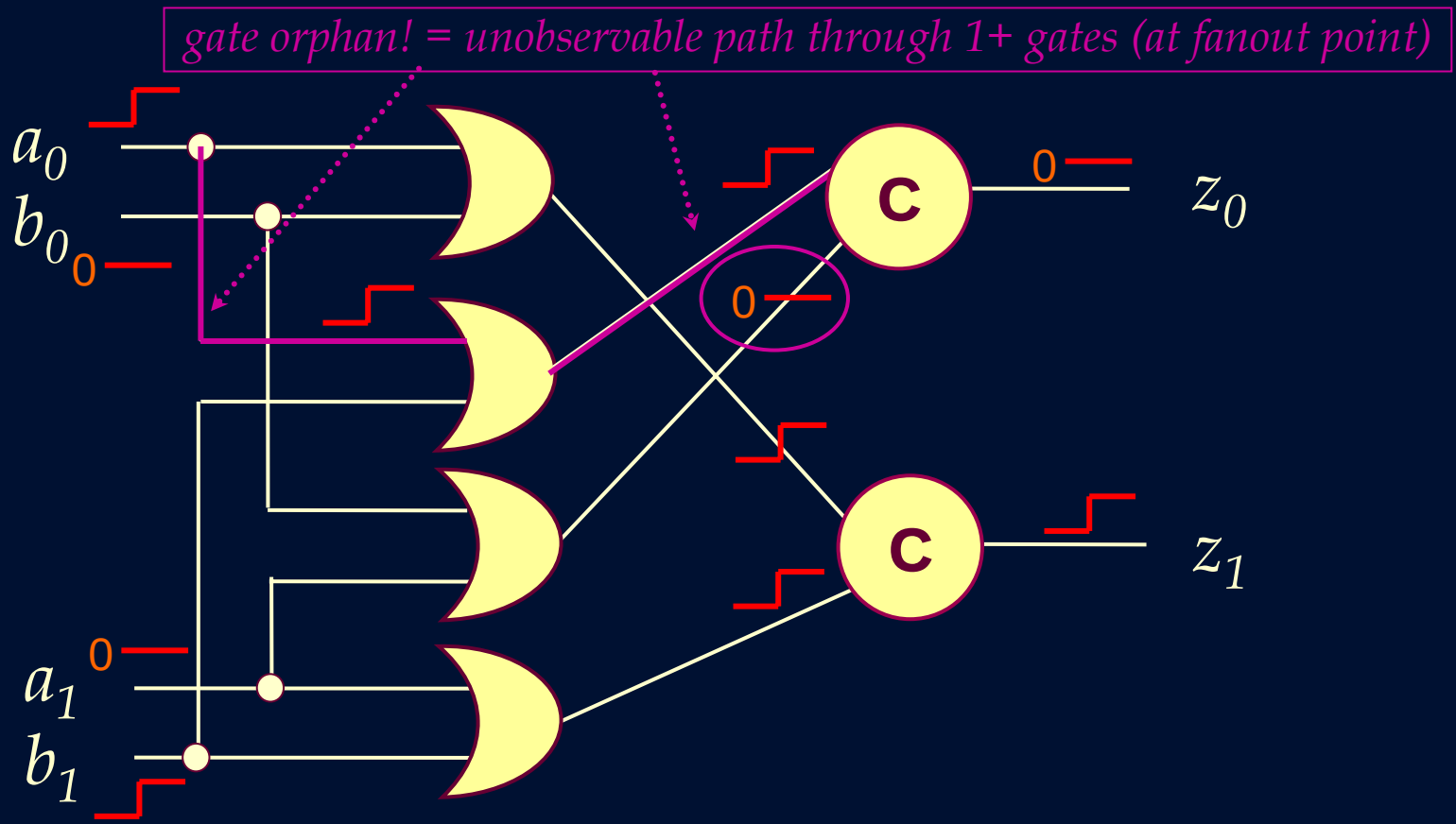


Wire orphan example

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

Hazard Issues

- Gate orphan example:



Gate orphan example

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

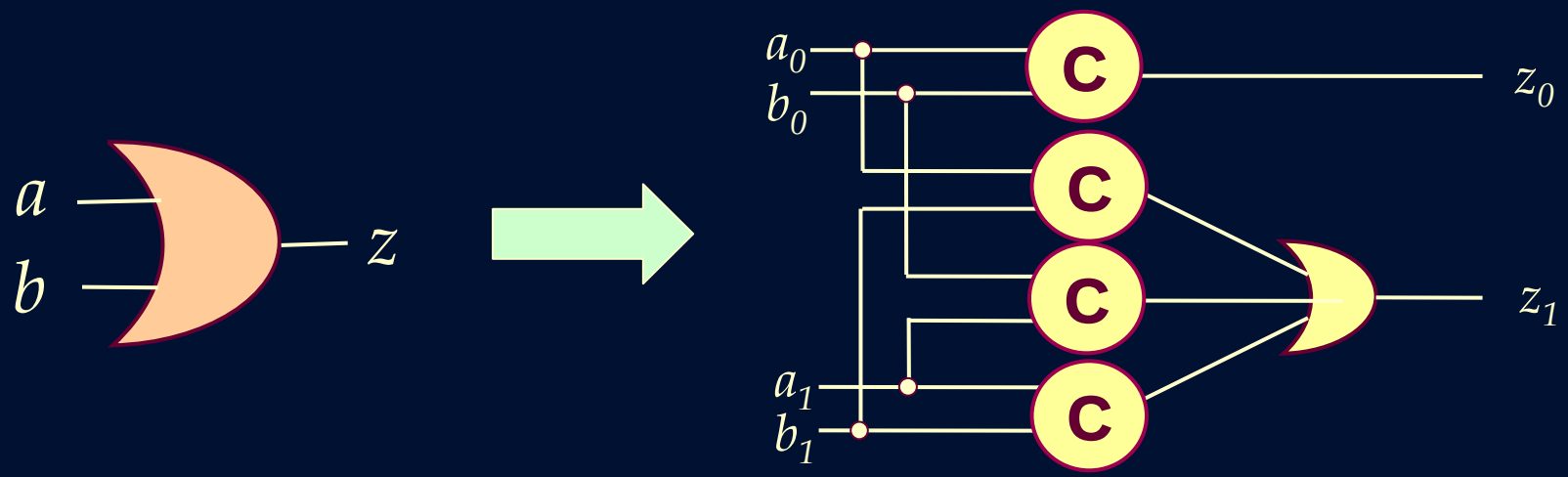
Single-rail Boolean network

Relaxation

Relaxed dual-rail async threshold network ⟵ *optimized*

Input Completeness

- 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.



Boolean OR gate

Input-complete dual-rail OR network

(input complete w.r.t. input signals a and b)

Enforcing input completeness for every gate is the traditional synthesis approach to avoid hazards (i.e. gate orphans).

Input Incompleteness

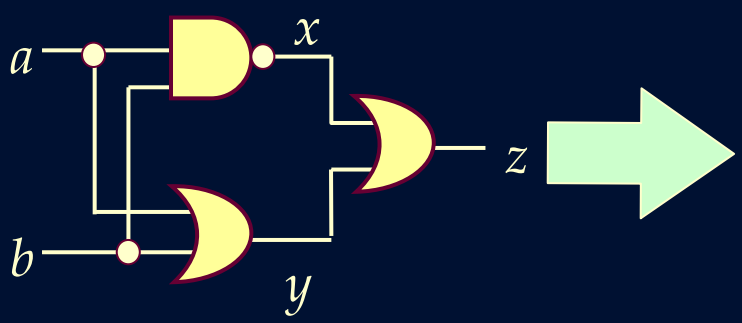
- A dual-rail implementation of a Boolean gate is input-incomplete w.r.t. its input signals ("eager-evaluating"), if the output can change *before* all inputs arrive.

Boolean OR gate

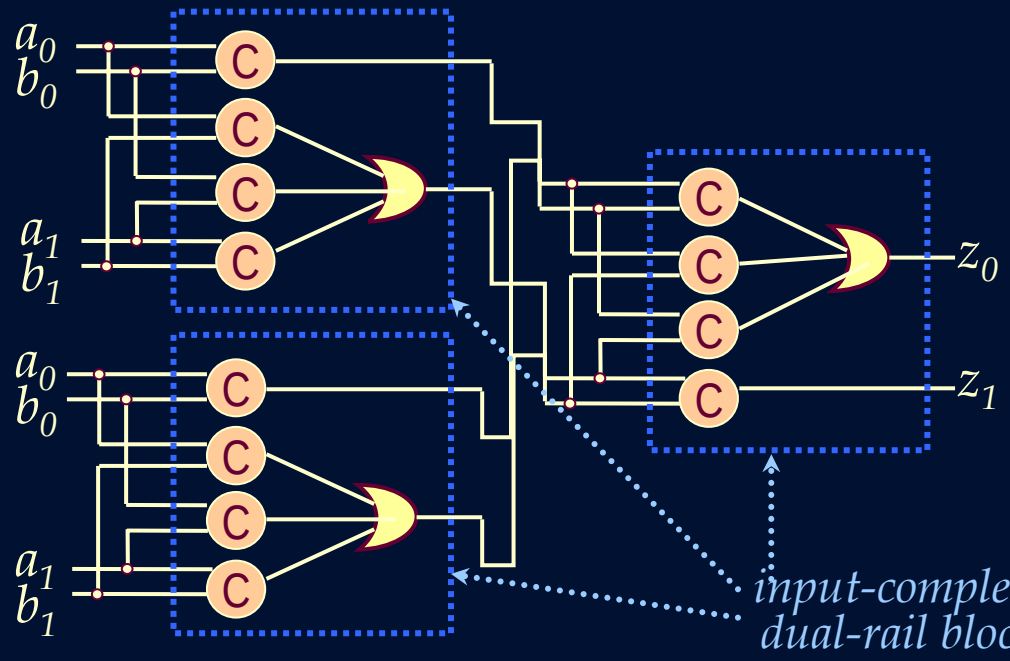
Input-incomplete dual-rail OR network

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.



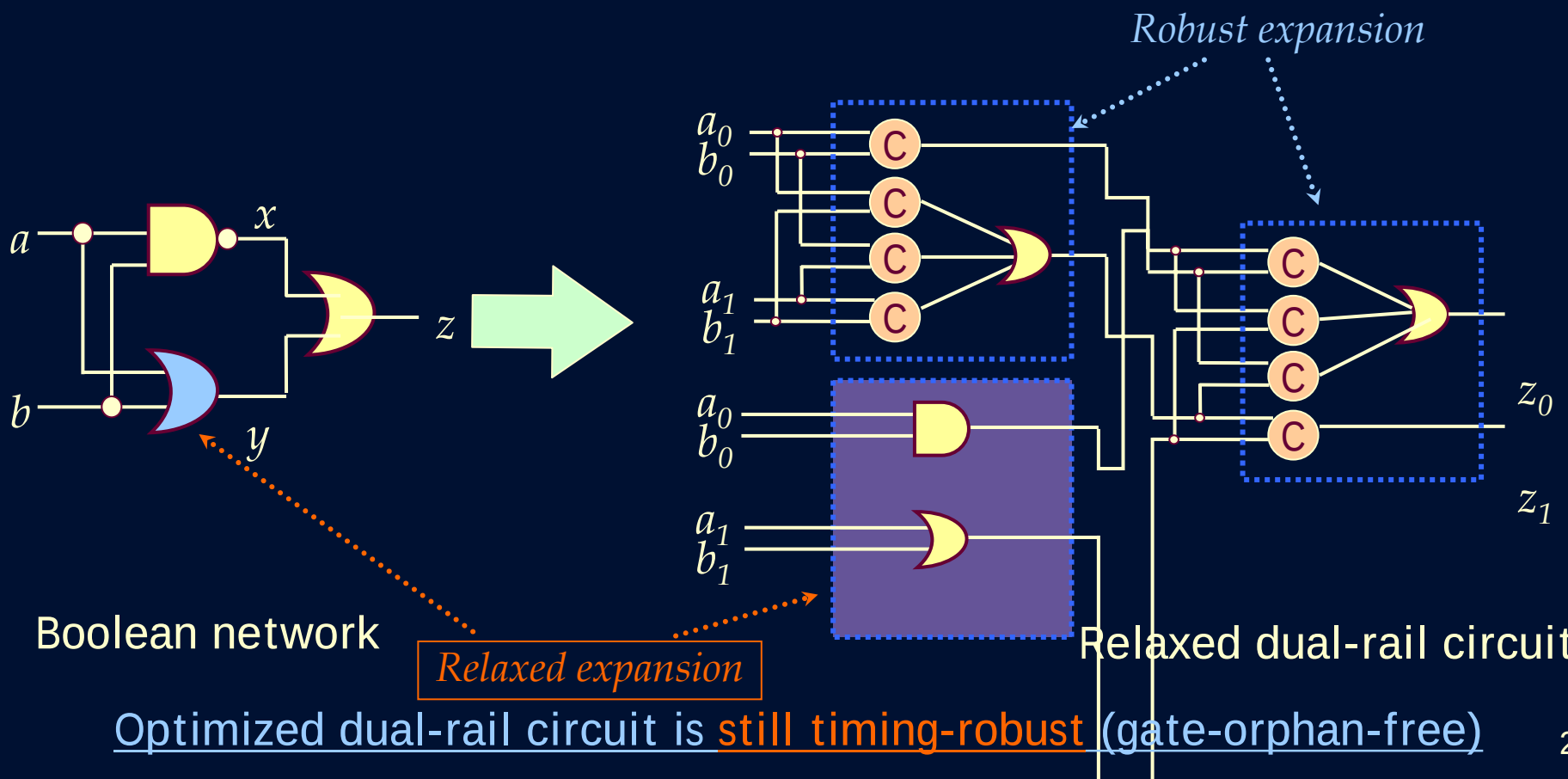
Boolean network



Dual-rail circuit with full expansion (no relaxation)

Gate-Level Relaxation Example #1 (cont.)

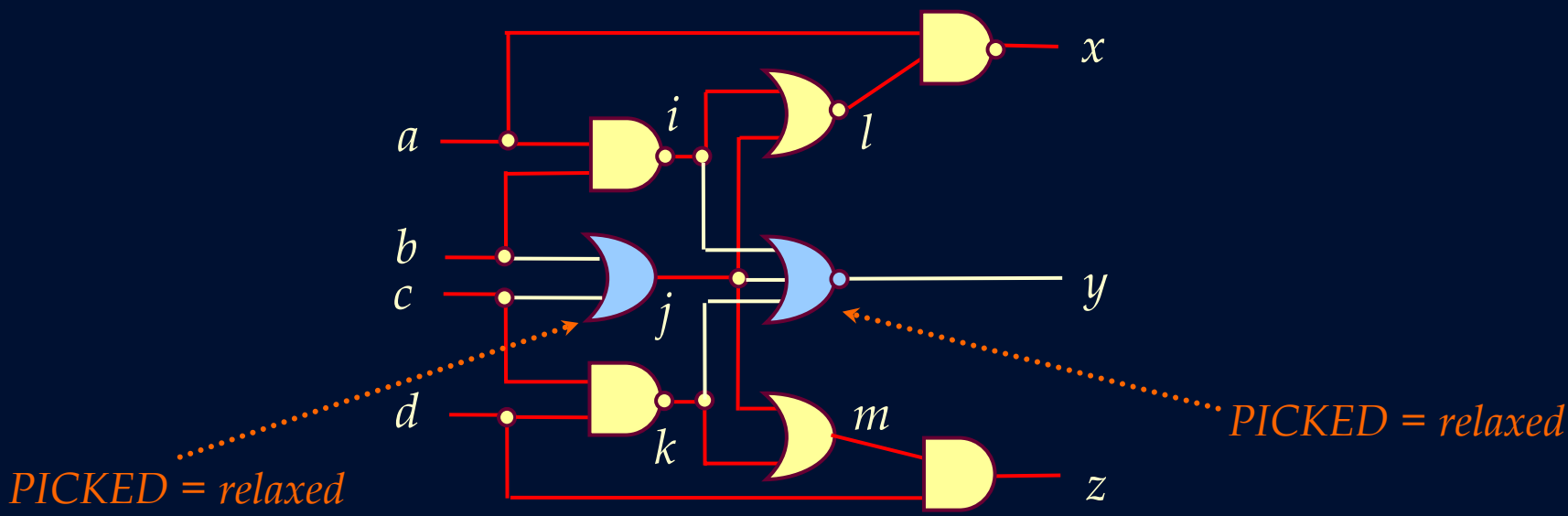
- Not every Boolean gate needs to be expanded into input-complete block.



Optimized dual-rail circuit is still timing-robust (gate-orphan-free)

Gate-Level Relaxation Example #2

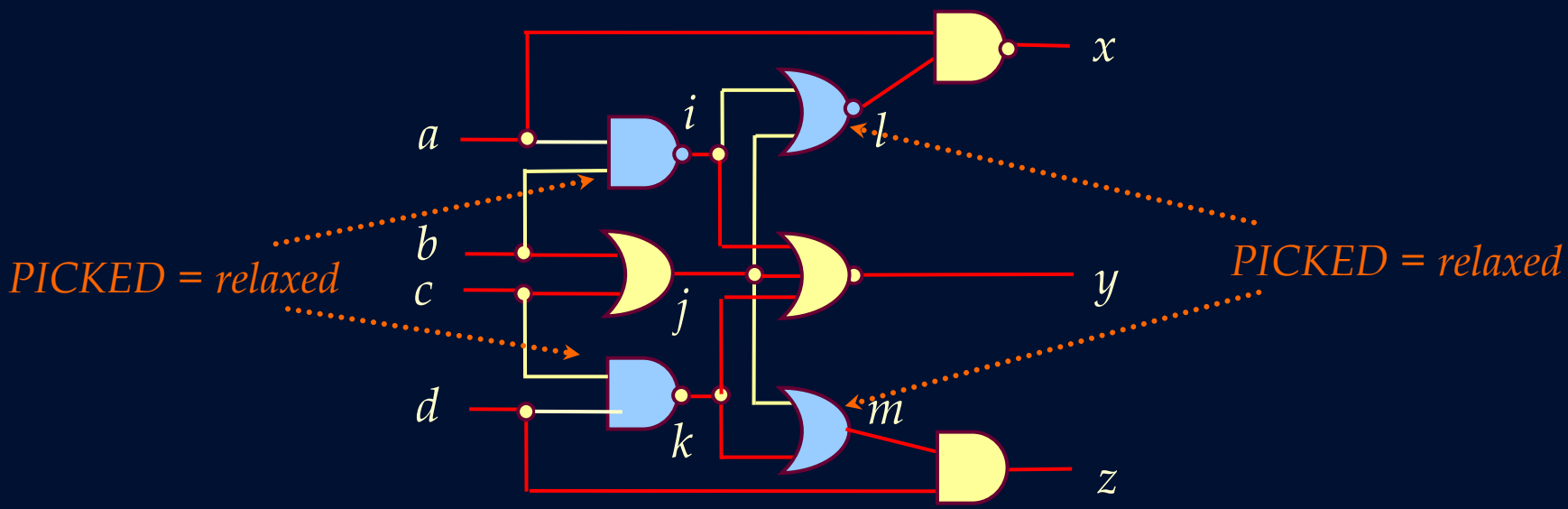
- Different choices may exist in relaxation.



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)

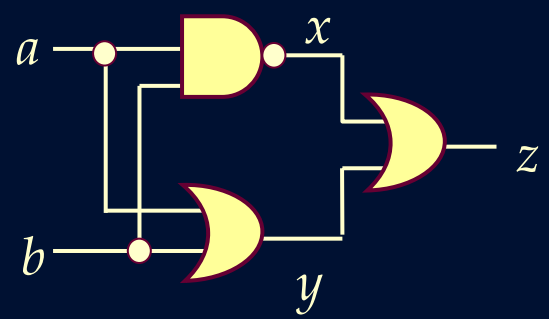
Resulting implementation has no loss of timing robustness (remains "gate-orphan-free")

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:



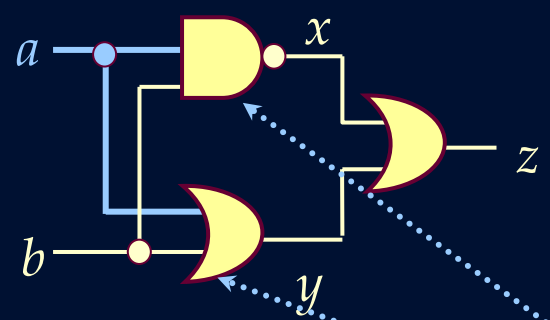
Boolean network

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

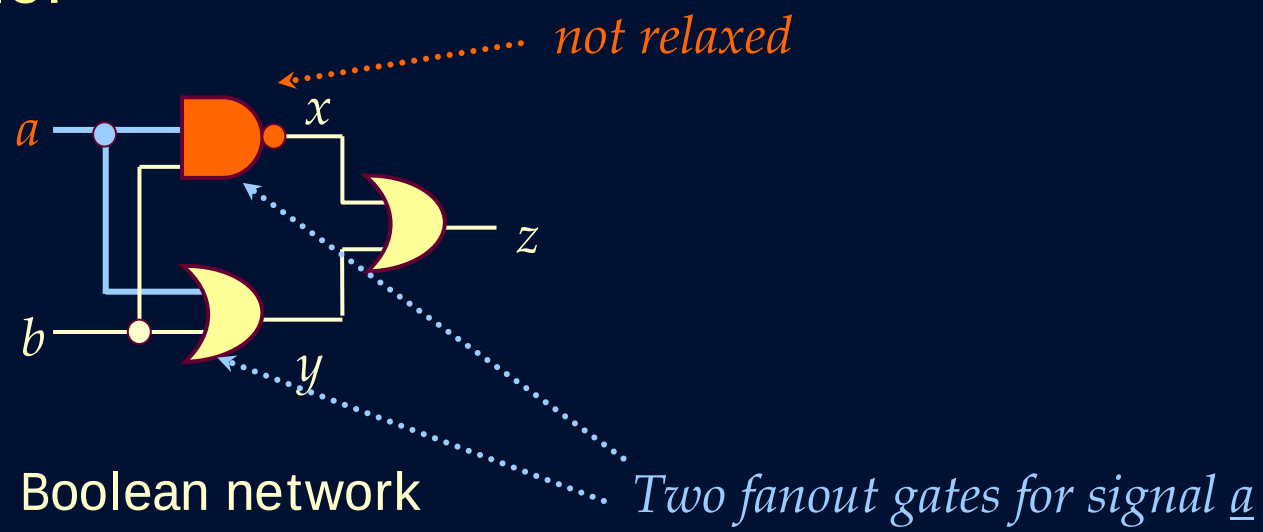
Two fanout gates for signal a

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:



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

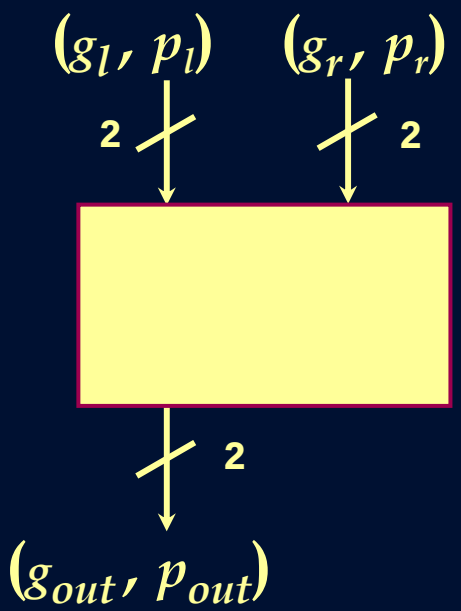
Outline

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

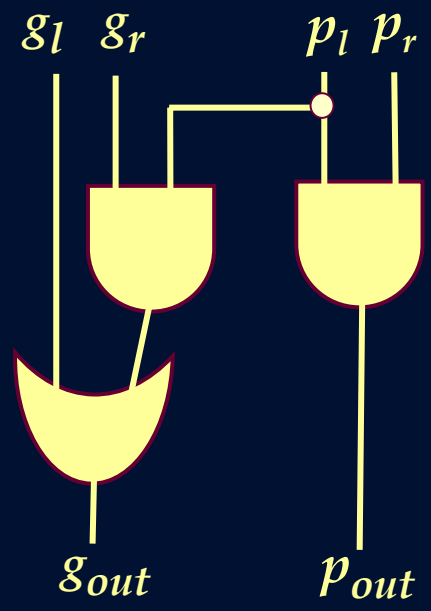
Block-Level Relaxation

- Block-level vs. Gate-level circuits

Block-level circuit	Gate-level circuit
Consists of large granularity blocks	Consists of simple gates
Blocks have *multiple* outputs	Gates have *single* output



P/G block in prefix adders



Gate-level implementation of P/G block

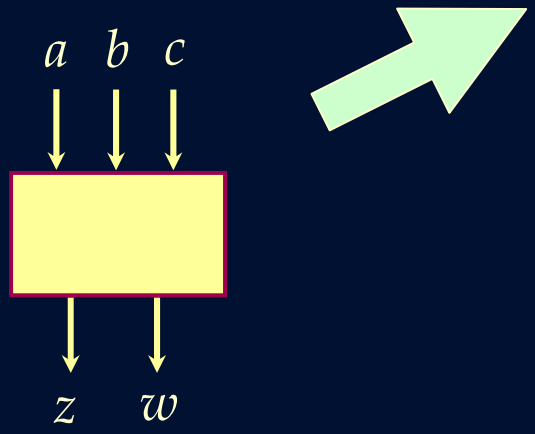
Why Relaxation at Block-Level?

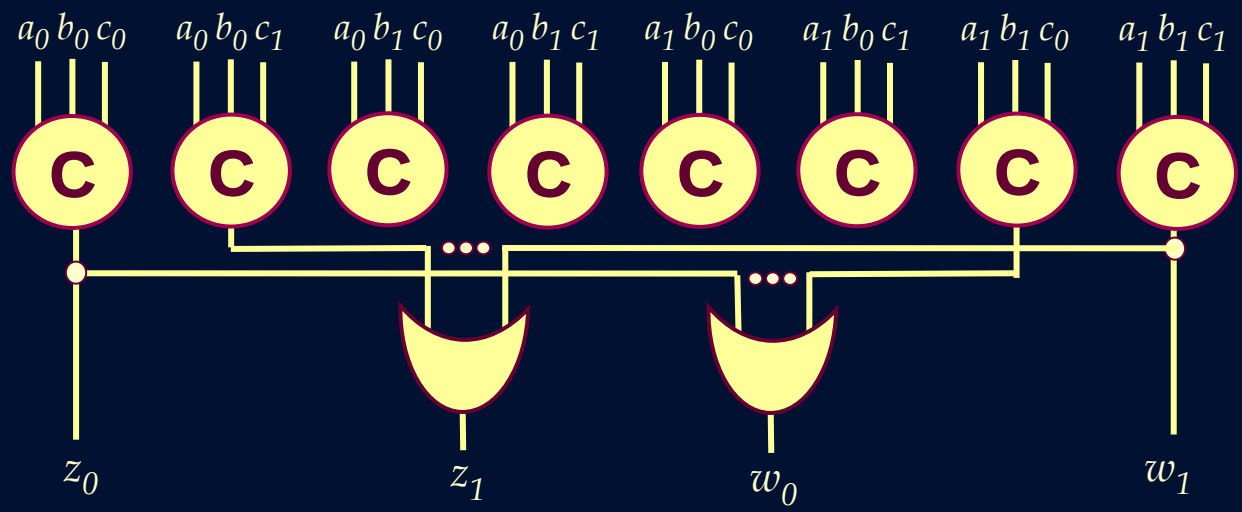
- Like gate-level relaxation: blocks are either
  - input complete: wait for all inputs to arrive
  - relaxed: eager, do not wait for all inputs to arrive
- New idea: 3rd possibility
  - “partially-eager”:
    - input complete: each input vector acknowledged on *some output*
    - partially-eager: allows some outputs to fire early

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)





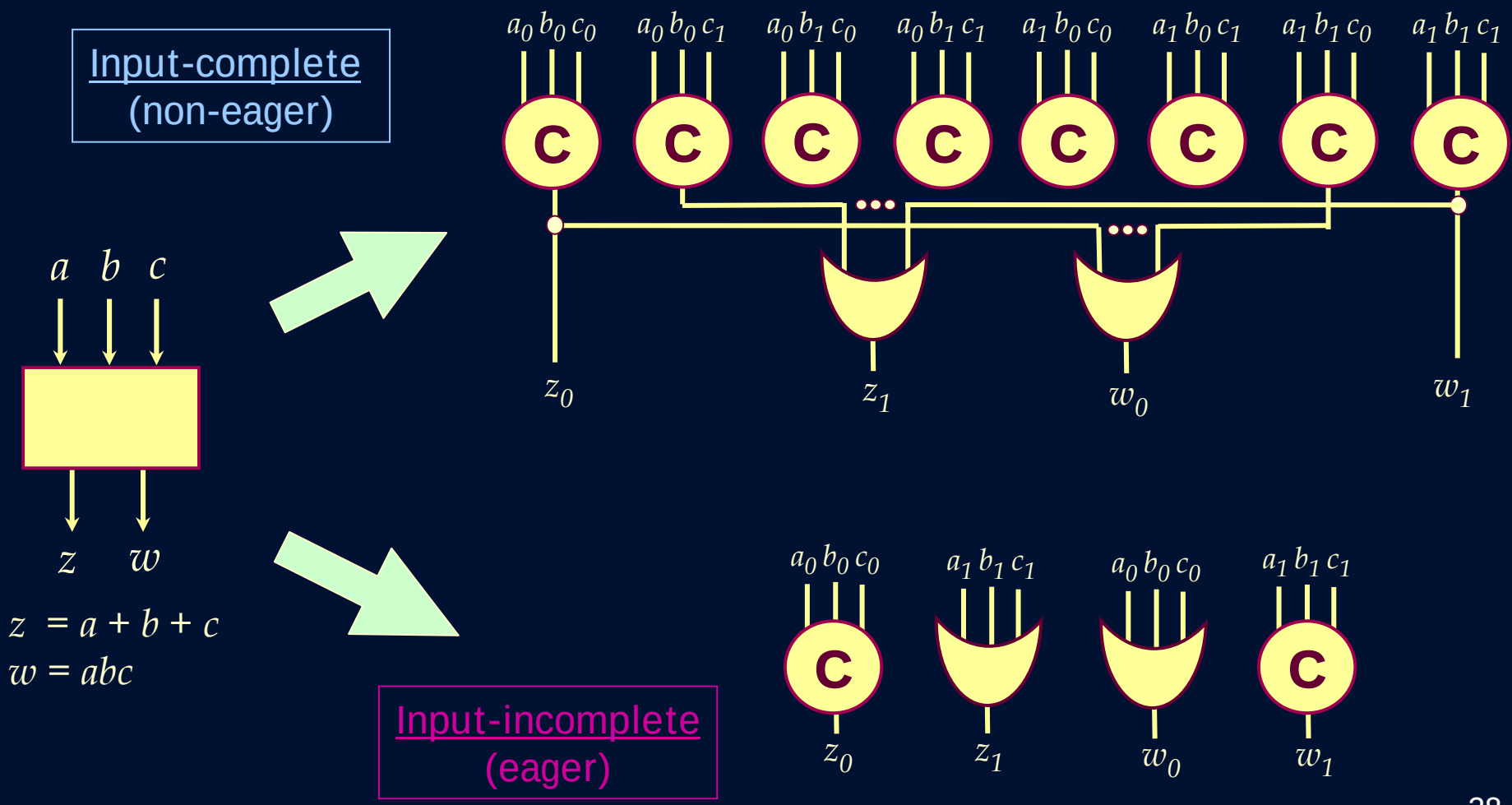
$z = a + b + c$

$w = abc$

Block example

Block-Level Relaxation Example

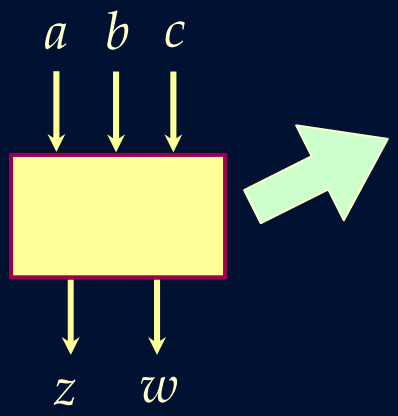
- Basic approach = direct extension of gate-level relaxation
  - No output in robust block fires before all inputs arrive



Block-Level Relaxation Example

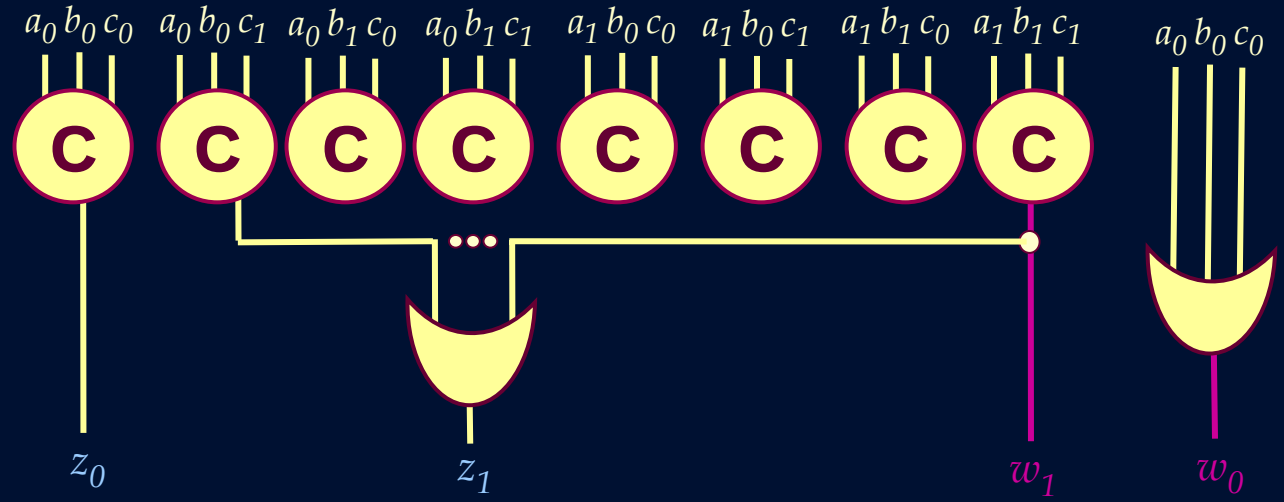
- New Option #1: "Biased Approach"
  - In biased implementation of blocks, only one output is implemented in a robust way; other outputs are eager-evaluating

Input-complete block
(and partially eager!)



$z = a + b + c$
$w = abc$

Block example

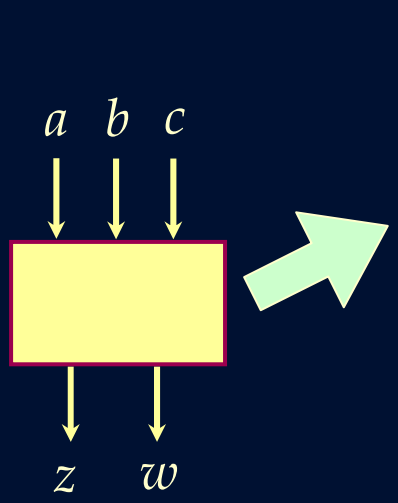


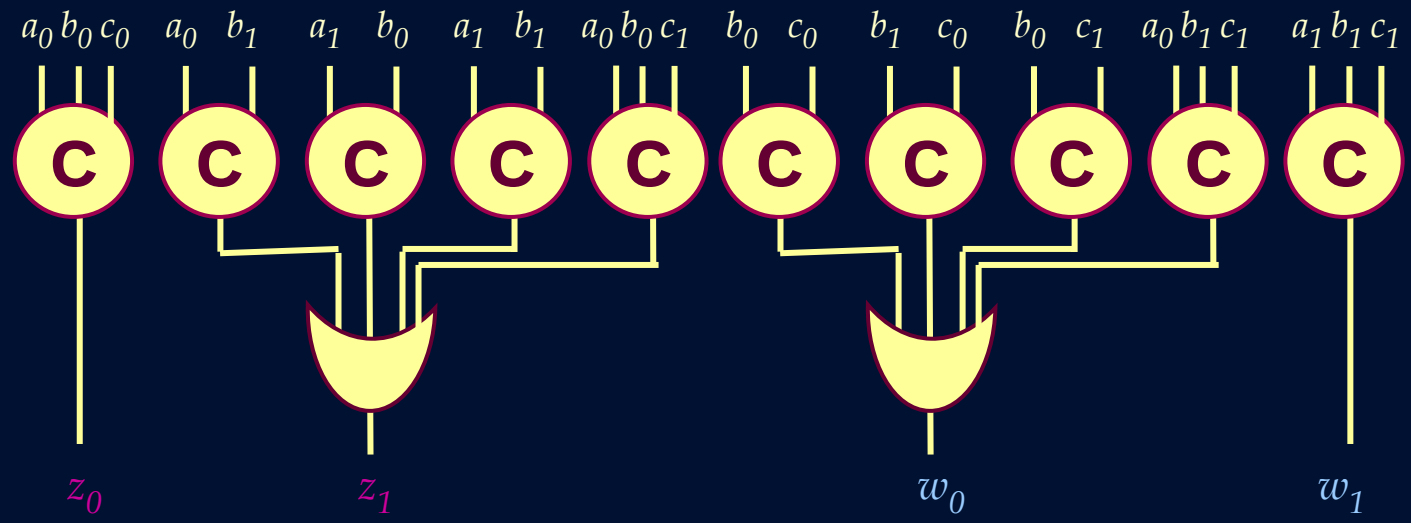
Output z: waits for all inputs ("non-eager")
Output w: early evaluating ("eager")

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!)



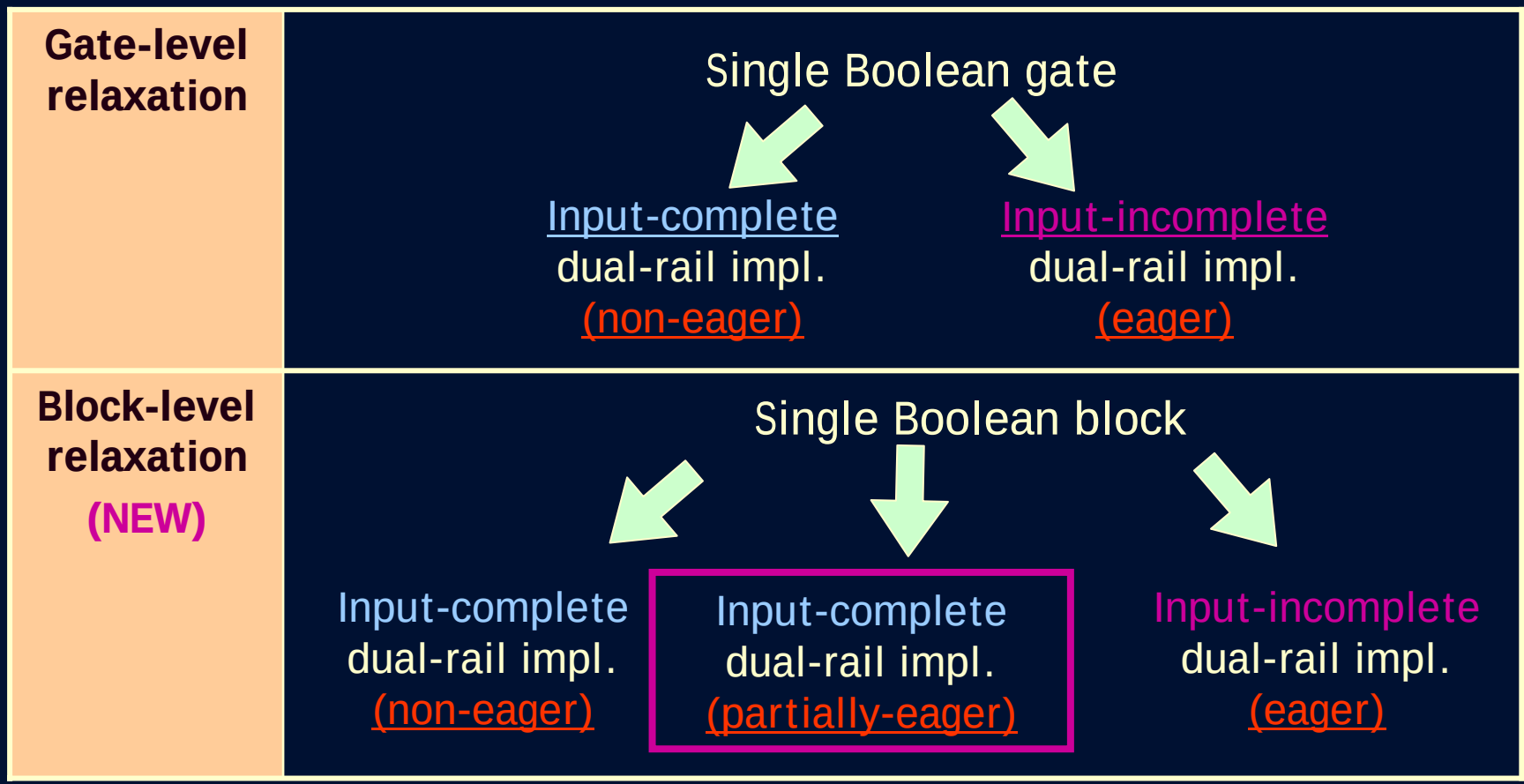


$z = a + b + c$
$w = abc$

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?



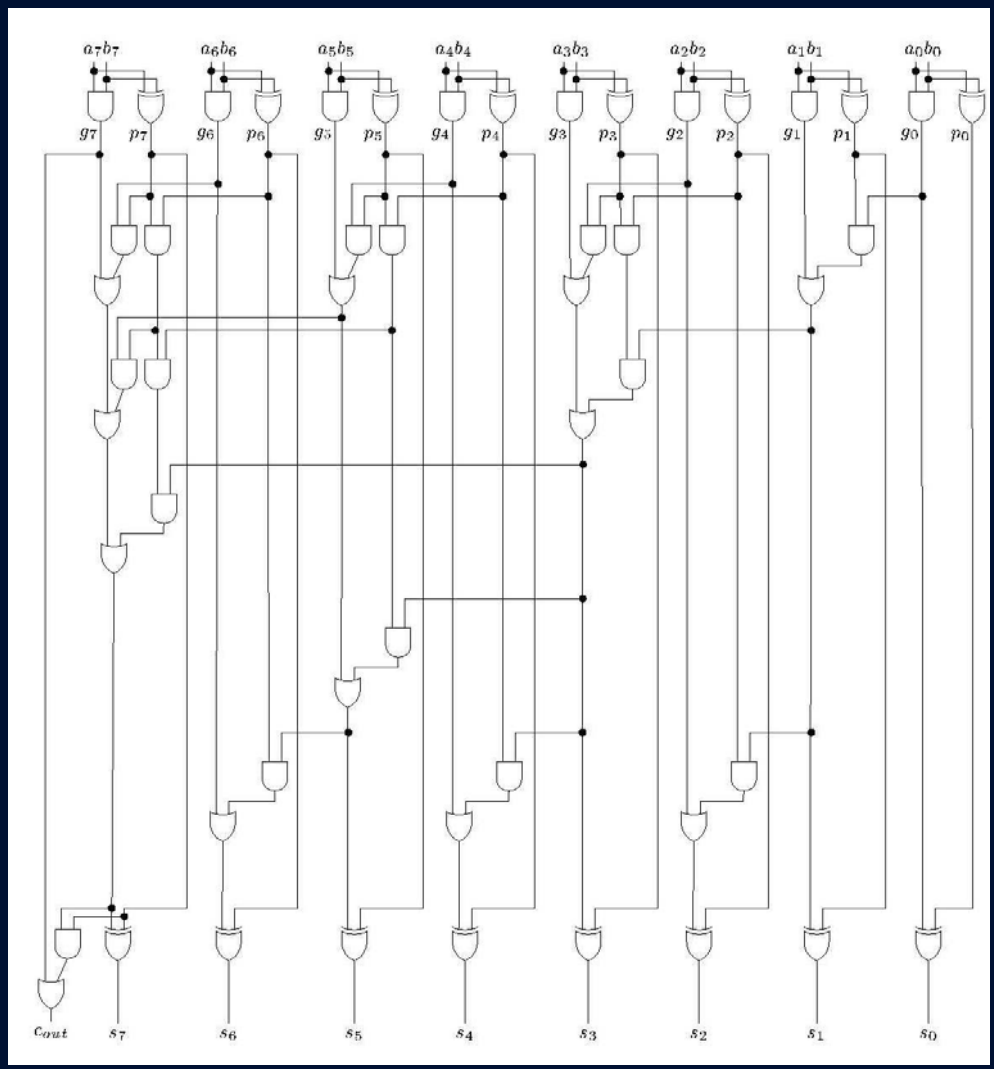
More optimization opportunities + larger design space

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

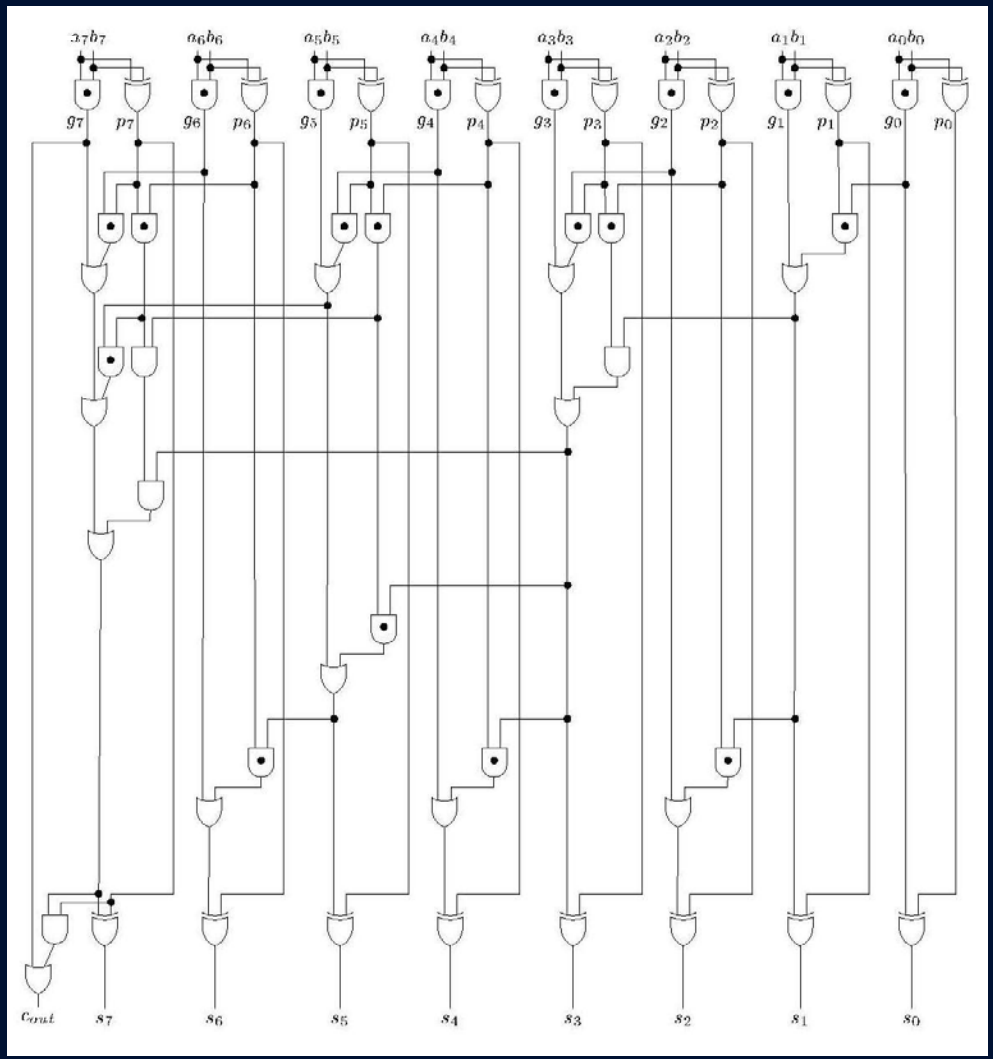
- Gate-level relaxation example



Gate-level 8-bit Brent-Kung adder circuit (Initial Boolean network)

Block- vs Gate-Level Relaxation Example

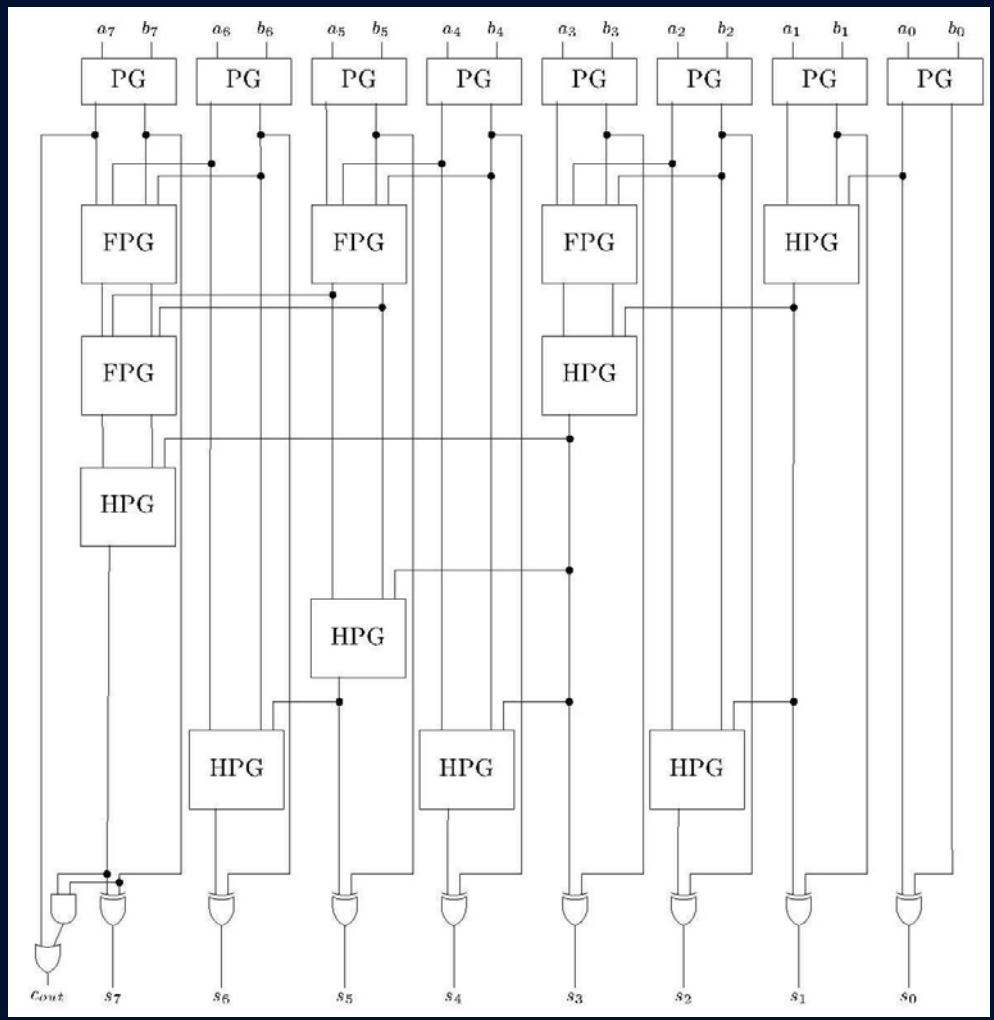
- Gate-level relaxation example



Gate-level 8-bit Brent-Kung adder circuit w/ relaxed gates marked

Block- vs Gate-Level Relaxation Example

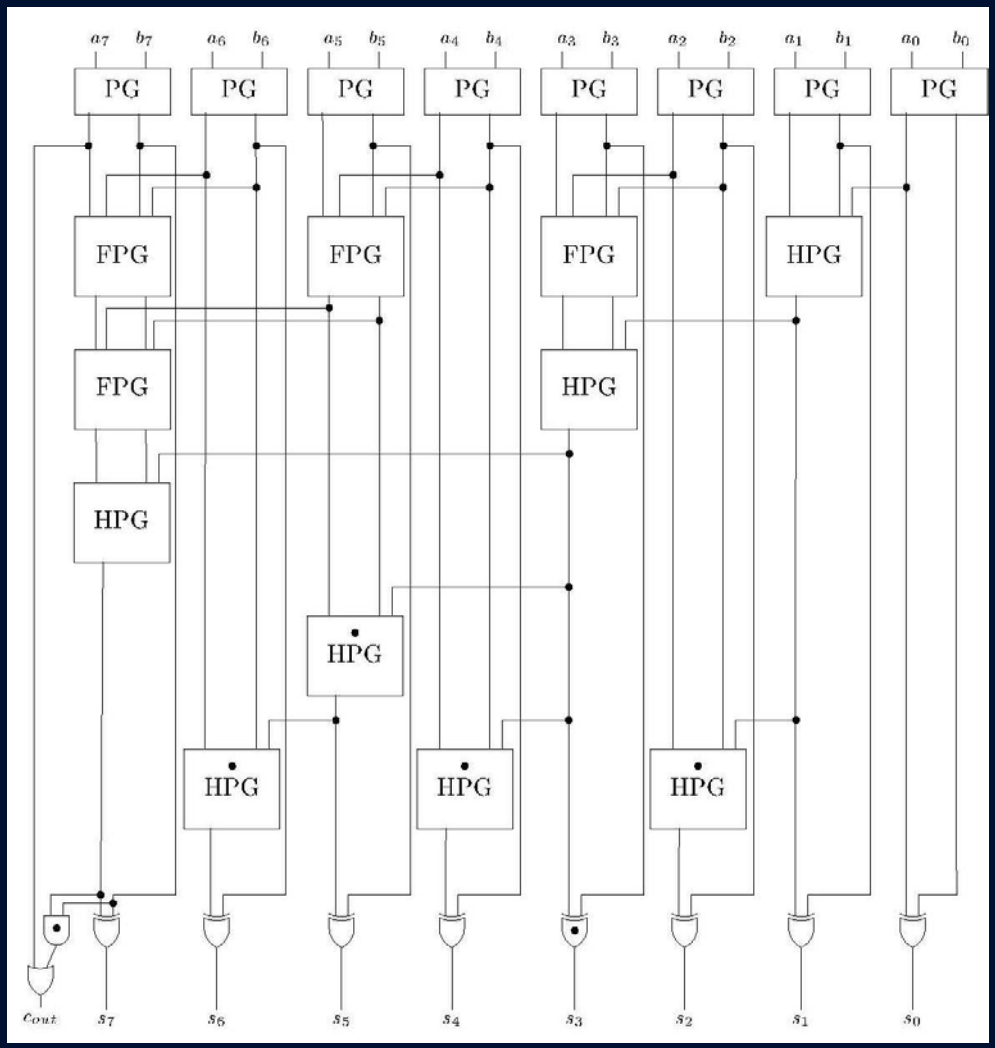
- Block-level relaxation example



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

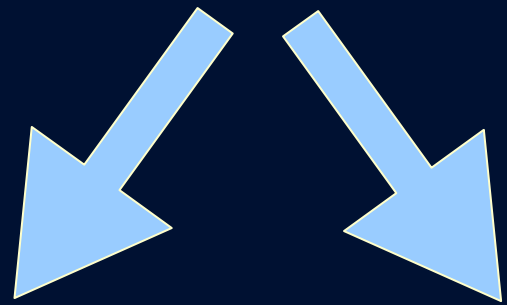
Outline

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

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

compared

Relaxed dual-rail arithmetic circuit

Experimental Results (cont.)

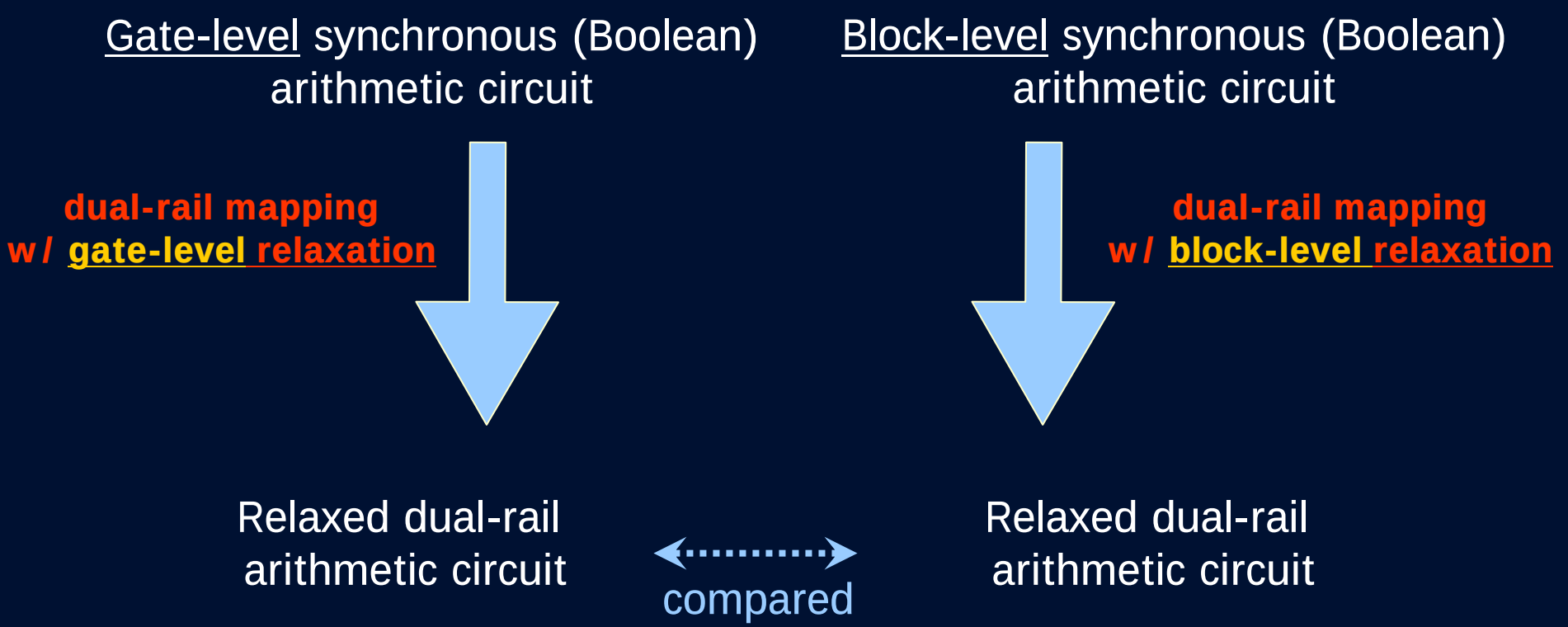
Experiment #1: Effectiveness of block-level relaxation

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

Original block-level network		Unoptimized block-level dual-rail circuit		Relaxed block-level dual-rail circuit	
name	#i/#o/#g	area	critical delay	area	critical delay
8-b Brent-Kung	32/18/49	9020.2	8.45	6094.1	6.64
16-b Brent-Kung	4/34/110	21599.9	12.19	13587.8	9.65
8-b Kogge-Stone	32/18/67	16208.6	7.68	9624.9	5.84
16-b Kogge-Stone	64/34/179	44916.0	13.36	22596.4	7.57
8-b unopt. mult	32/16/323	29231.2	25.01	24998.4	23.52
16-b unopt. mult	64/32/1411	126786.0	53.78	108728.0	52.29
8-b opt. mult	32/16/320	28984.4	17.66	24745.0	15.44
16-b opt. mult	64/32/1408	126538.0	37.02	108474.0	32.97
Average percentage				72.8%	86.9%

Experimental Results (cont.)

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



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

Original Boolean network		Relaxed gate-level dual-rail circuit		Relaxed block-level dual-rail circuit	
name	#i/#o/#g	area	critical delay	area	critical delay
8-b Brent-Kung	32/18/49	4688.6	7.48	6094.1	6.64
16-b Brent-Kung	4/34/110	10396.8	10.69	13587.8	9.65
8-b Kogge-Stone	32/18/67	6341.8	5.57	9624.9	5.84
16-b Kogge-Stone	64/34/179	16571.5	6.99	22596.4	7.57
8-b unopt. mult	32/16/323	28828.4	25.69	24998.4	23.52
16-b unopt. mult	64/32/1411	125915.0	55.87	108728.0	52.29
8-b opt. mult	32/16/320	28523.1	20.98	24745.0	15.44
16-b opt. mult	64/32/1408	125610.0	46.70	108474.0	32.97
Average percentage				110.8%	91.2%

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

Original Boolean network		Relaxed gate-level dual-rail circuit		Relaxed block-level dual-rail circuit	
name	#i/#o/#g	area	critical delay	area	critical delay
8-b Brent-Kung	32/18/49	4688.6	7.48	6094.1	6.64
16-b Brent-Kung	4/34/110	10396.8	10.69	13587.8	9.65
8-b Kogge-Stone	32/18/67	6341.8	5.57	9624.9	5.84
16-b Kogge-Stone	64/34/179	16571.5	6.99	22596.4	7.57
8-b unopt. mult	32/16/323	28828.4	25.69	24998.4	23.52
16-b unopt. mult	64/32/1411	125915.0	55.87	108728.0	52.29
8-b opt. mult	32/16/320	28523.1	20.98	24745.0	15.44
16-b opt. mult	64/32/1408	125610.0	46.70	108474.0	32.97
Average percentage				110.8%	91.2%

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

Original Boolean network		Relaxed gate-level dual-rail circuit		Relaxed block-level dual-rail circuit	
name	#i/#o/#g	area	critical delay	area	critical delay
8-b Brent-Kung	32/18/49	4688.6	7.48	6094.1	6.64
16-b Brent-Kung	4/34/110	10396.8	10.69	13587.8	9.65
8-b Kogge-Stone	32/18/67	6341.8	5.57	9624.9	5.84
16-b Kogge-Stone	64/34/179	16571.5	6.99	22596.4	7.57
8-b unopt. mult	32/16/323	28828.4	25.69	24998.4	23.52
16-b unopt. mult	64/32/1411	125915.0	55.87	108728.0	52.29
8-b opt. mult	32/16/320	28523.1	20.98	24745.0	15.44
16-b opt. mult	64/32/1408	125610.0	46.70	108474.0	32.97
Average percentage				110.8%	91.2%

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