A CAD Toolset for the Synthesis and Optimization of Asynchronous Threshold Networks

Cheoljoo Jeong Steven M. Nowick
Computer Science Department
Columbia University

This work was partially supported by NSF ITR Award No. NSF-CCR-0086036 and by a subcontract to Boeing under the DARPA "CLASS" program.

Introduction: ATN_OPT Toolset

- ATN_OPT Toolset
  - Optimization tools for robust asynchronous threshold networks
- Includes two optimization tools
  - ATN_RELAX for relaxation
  - ATN_MAP for technology mapping and cell merger
- Supports common modeling languages:
  - Verilog, VHDL, BLIF formats
- Supports an extensible target cell library:
  - GENLIB format

Introduction: ATN_OPT Toolset

Download site:

http://www1.cs.columbia.edu/~nowick/asynctools

- ATN_OPT is part of the "CaSCADE" tool package, a set of asynchronous tools and libraries developed by Columbia and USC

Related publications (see download site):

- ATN_RELAX for relaxation

  C. Jeong and S.M. Nowick, "Optimization of Robust Asynchronous Circuits by Local Input Completeness Relaxation." In Proceedings of the IEEE Asia and South-Pacific Design Automation Conference (ASPDAC-07)," Yokohama, Japan (Jan. 2007).

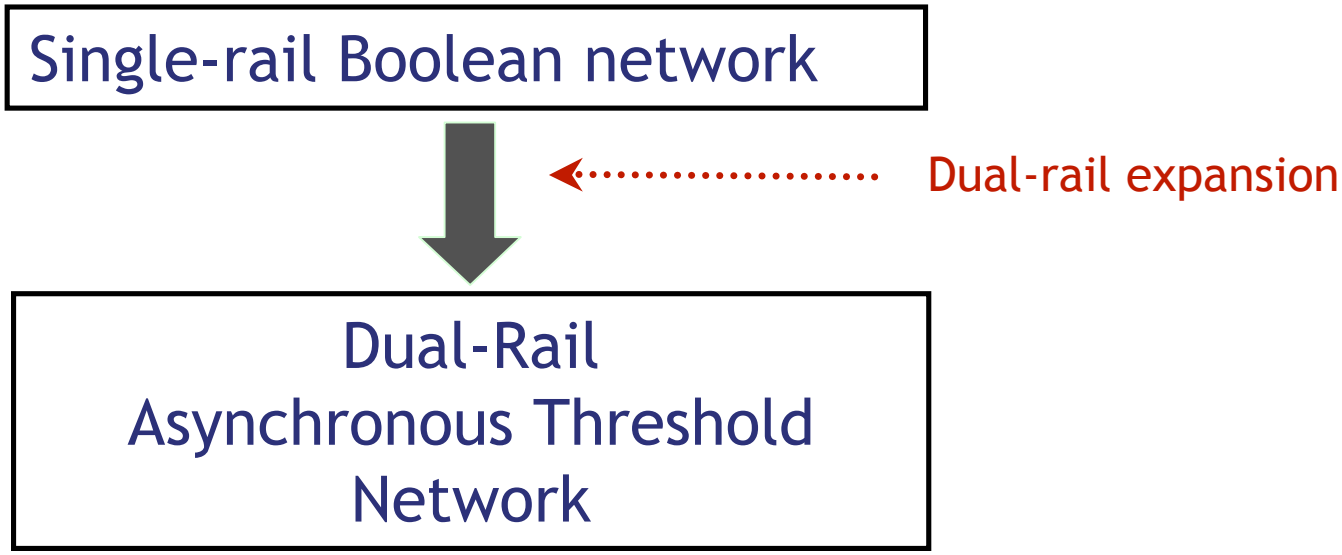
- ATN_MAP for technology mapping and cell merger

  C. Jeong and S.M. Nowick, "Optimal Technology Mapping and Cell Merger for Asynchronous Threshold Networks." In Proceedings of the IEEE International Symposium on Asynchronous Circuits and Systems (Async-06)," Grenoble, France (March 2006).

  C. Jeong and S.M. Nowick, "Technology Mapping and Cell Merger for Asynchronous Threshold Networks." IEEE Transactions on Computer-Aided Design (TCAD) (*to appear,* approx. February 2008).

Asynchronous Threshold Networks

- Asynchronous threshold networks
  - One of the most robust asynchronous circuit style
  - Based on delay-insensitive encoding
- Traditional synthesis flow for asynchronous threshold networks

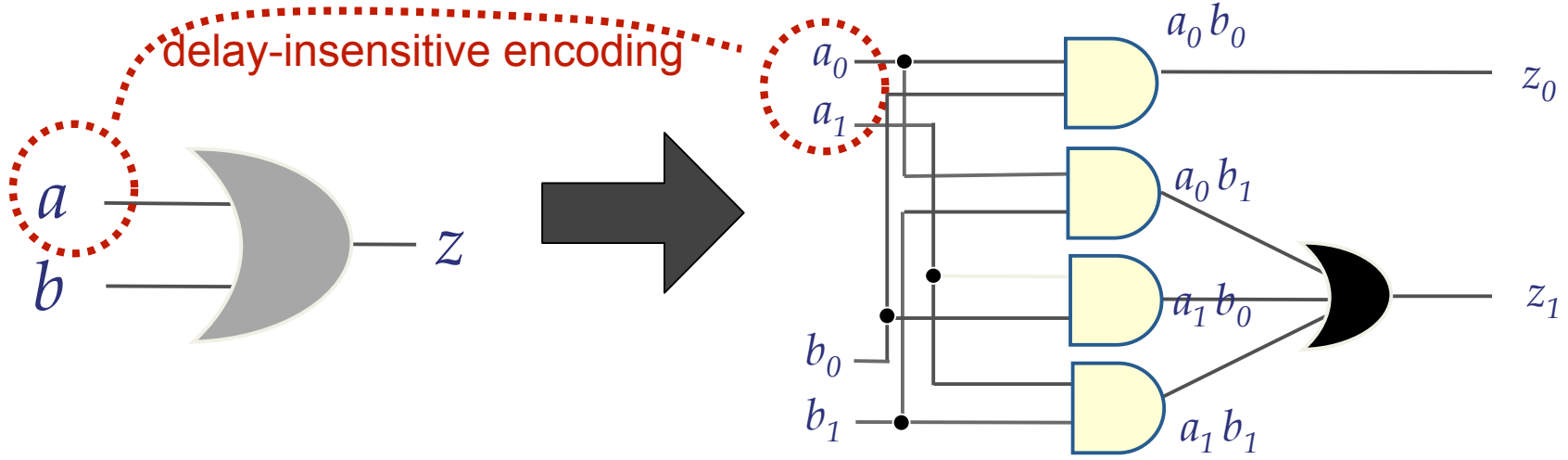


Asynchronous Threshold Networks (cont.)

- Delay-Insensitive Encoding:

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

- Example:

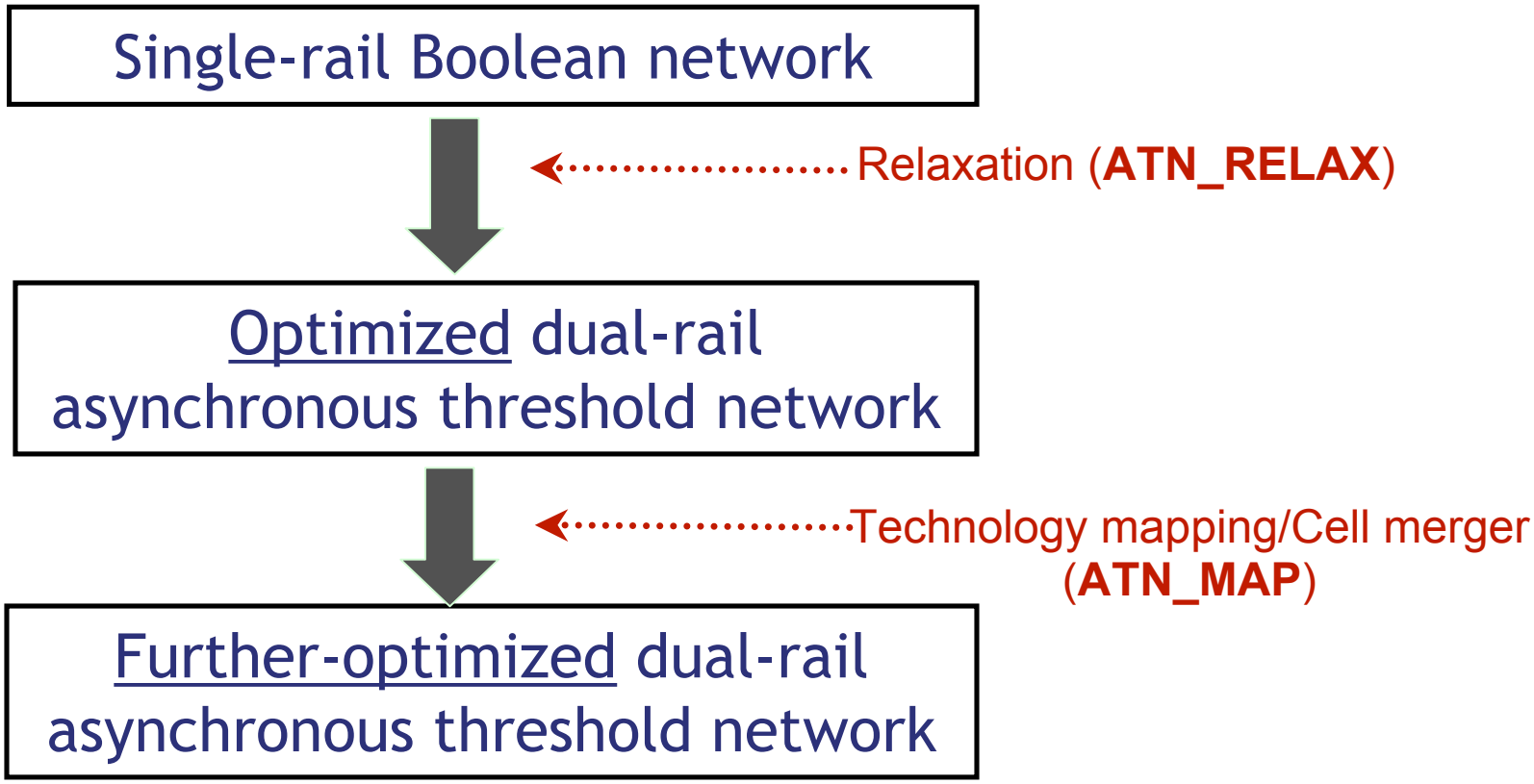


Combinational OR gate
(single-rail Boolean gate)

Equivalent asynchronous dual-rail asynchronous threshold network
in DIMS (delay-insensitive minterm synthesis)-style

ATN_OPT Tool Flow

- Overview of proposed approach:

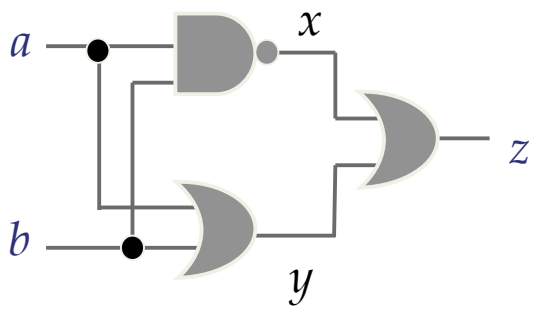


Relaxation

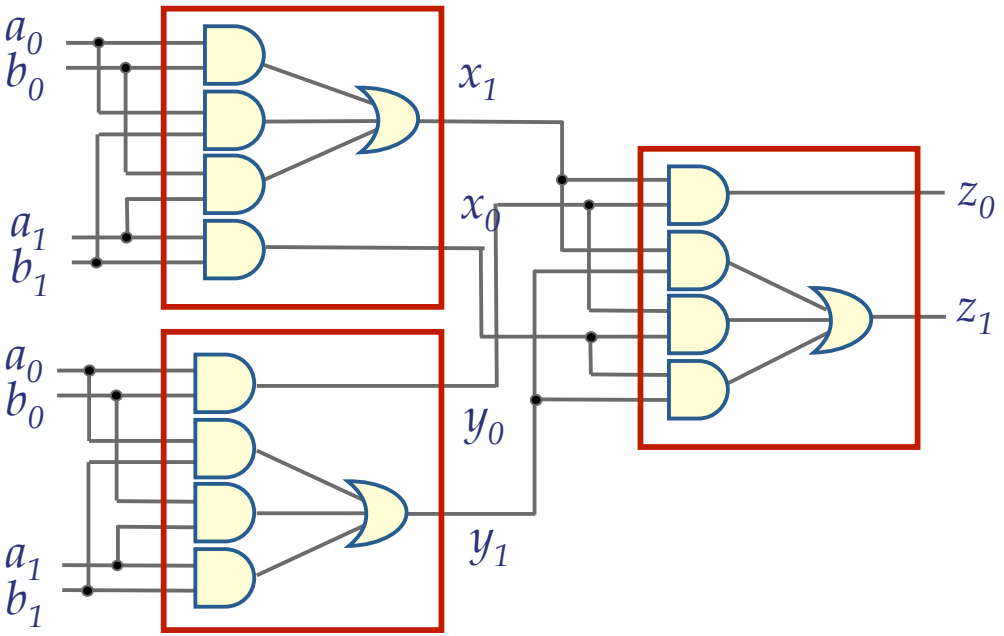
- Relaxation algorithm
  - Optimization technique for asynchronous threshold network which allows “eager” dual-rail logic
  - Relaxes local robustness without destroying global timing-robustness
  - The relaxation tool is called ATN_RELAX

Relaxation (cont.)

- Traditional synthesis flow for asynchronous threshold networks
  - Starting point: Boolean single-rail logic network
  - Goal: Map every gate into "robust" dual-rail asynchronous threshold logic
  - Dual-rail circuit operation: 2 "PHASES"
    - EVAL: new dual-rail inputs arrives/compute dual-rail outputs
    - RESET: inputs return to all-0 ("NULL")/outputs return to all-0



Initial Boolean circuit
(single-rail)



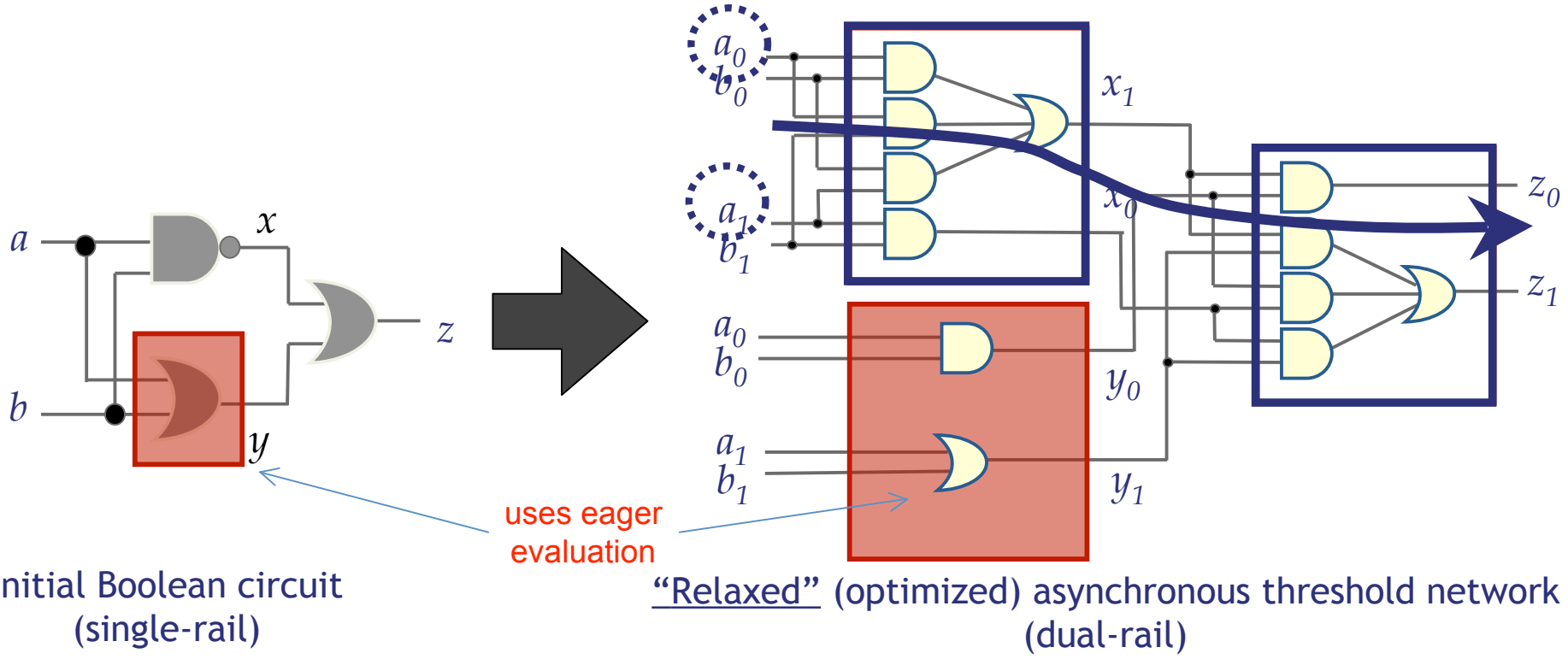
Unoptimized asynchronous threshold network
("DIMS"-style, dual-rail)

Relaxation (cont.)

- Relaxation of asynchronous threshold networks

  Observation: As long as every signal (primary input + internal signals) has at least one "robust path" to a primary output (i.e. path *with no eager blocks*), all remaining blocks can be eager.

  – Each signal still “robustly observed” at the output (through these robust paths).



Technology Mapping/Cell Merger

- Proposed technology mapping approach (ATN_MAP):
  - First general and systematic approach for dual-rail asynchronous threshold networks
  - Handles entire asynchronous circuits with 10,000+ gates very efficiently, which was not possible before:
    - function blocks, registration, completion detectors
  - Handles sophisticated library characterization:
    - input slew rate, individual I/O paths, rise vs. fall transitions, cell output loading
    - delay, area, power
  - Leverages efficient synchronous mapping techniques:
    - dynamic programming, load-binning, network decomposition to base functions, etc.

Technology Mapping/Cell Merger (cont.)

- Proposed cell merger approach (ATN_MERGE):
  - Proposed as a restricted version of technology mapping:
    - *only* merges adjacent nodes of the original circuit (no decomposition step used)
  - Faster than technology mapping, but less optimization capabilities

Technology Mapping/Cell Merger (cont.)

- Overview of the proposed algorithm: four steps
  - *Gate-orphan-free decomposition (NEW)*
    - Decompose original netlist into gate-orphan-free netlist
  - Partitioning (use existing techniques)
  - *Pattern graph generation (NEW)*
    - Use new positive monotonic finite basis (= AND2/OR2)
    - Must include some complex irreducible nodes
  - Matching and covering (use existing techniques)

Gate-Orphan-Free Decomposition

- Basic Idea: decomposing nodes in initial netlist
  - Decompose each node into simple base functions: AND2 (C-elt.) / OR2
    - only if no "gate orphans" are introduced! (i.e. preserve timing robustness)
  - Otherwise: node *not decomposed*

- Overview of gate-orphan-free decomposition
  - AND2 (i.e. C-elt.)/OR2 gates: *not* decomposed
    - already "primitive base functions"
  - Large OR cells: can safely be decomposed into network of OR2 cells
  - Large AND cells: *not* decomposed -- decomposition unsafe
  - *Otherwise*, do not decompose nodes -- decomposition unsafe

Pattern Graph Generation

- After decomposition, subject graphs consist of:
  - simple base functions (AND2/OR2)
  - complex base functions (irreducible)
- Overview of pattern graph generation:
  - Library cell functions: decomposed using simple finite basis
    - *threshold-AND2* (i.e. 2-input C-element), *threshold-OR2* (i.e. OR2 gate)
  - If library cell function = complex base function, then:
    - special pattern graph is added = single irreducible node

ATN_OPT Toolset

- Supports two optimization algorithms:
  - technology mapping/cell merger, relaxation
- Supports multiple netlist formats:
  - Verilog netlist, Structural VHDL, BLIF
- Supports one cell library format:
  - GENLIB

ATN_OPT Toolset (cont.)

- Includes several user-selectable options:
  - Target single cost functions:
    - area, fixed delay, load-dependent delay, power, etc.
  - Target hybrid cost function = ‘delay-area tradeoff’
    - minimize area under given worst-case (hard) timing constraint
  - Specify # of “load bins”:
    - determines precision of delay-oriented technology mapping
  - Support detailed library characterization:
    - distinguishing I/O timing path in cells, rise/fall delays, power, area
  - Verification *(currently not activated in release v0.1):*
    - checks initial netlist against final circuit

Performing Optimizations

- To run an optimization tool on the current netlist using the current technology library, you can use one of the following commands
  - **relax**: performs relaxation
  - **map**: performs technology mapping
  - **merger**: performs cell merger

Measuring the resulting circuit quality

- To measure the quality of the resulting circuits, type one of the following commands in a shell
  - **area**: returns the area of the netlist
  - **fixdelay**: returns the fixed delay of the netlist
  - **delay**: returns the load-dependent delay of the netlist
  - **power**: returns the power of the netlist

Cost Functions

- **For relaxation:** *use "set cost" to update*
  - **area**: minimizes the area of the circuit
  - **fixdelay:** minimizes the fixed delay of the circuit
  - **delay:** minimizes the load-dependent delay of the circuit
  - **power**: minimizes the power of the circuit
  - **norlx:** maximizes the number of relaxed nodes
  - **cktgen:** no relaxation (i.e. does not perform optimization)
    - generates an unoptimized dual-rail asynchronous threshold network

Cost Functions (cont.)

- **For tech mapping/cell merger:** *use "set cost" to update*
  - **area**: minimizes the area of the circuit
  - **fixdelay:** minimizes the fixed delay of the circuit
  - **delay:** minimizes the load-dependent delay of the circuit
  - **power**: minimizes the power of the circuit
  - **tradeoff:** minimizes area under a given hard timing constraint

Writing Resulting Netlists

- After performing optimization, the resulting circuit is set to be the "current" netlist.
- To write the current netlist into a file, use one of the following commands:
  - **write_blif** [filename]: writes the current netlist in BLIF format into a file named "filename"
  - **write_verilog** [filename]: writes the current netlist in Verilog format into a file named "filename"
  - **write_vhdl** [filename]: writes the current netlist in VHDL format into a file named "filename"

Setting Execution Options

- For optimization tools, there are several user-specified options:
  - **iopath**: if this option is ON, delay computation distinguishes each input-to-output timing arc in a gate
  - **risefall**: if this option is ON, delay computation distinguishes rise delay and fall delay
  - **loadbins**: user can set the number of load bins to be used in technology mapping for load-dependent delay
  - **reqtime**: user can set the required time to be used in technology mapping for delay-area tradeoff

Hands-On Tutorial

#0. Setup: running the 'atn_opt' shell

#1. Basics: starting from a single-rail logic network

- Synthesize a robust dual-rail async threshold network
- **Covered:**
  - (a) setup: reading in a netlist/library
  - (b) area run
  - (c) delay run
  - (d) specialized runs (skipping steps, etc.)

#2. Basics: starting from a dual-rail logic network

- Further optimize it (using tech map/merge)
- **Covered:**
  - (a) setup: reading in a netlist/library
  - (b) area run

#3. Advanced: 'delay-area' tradeoff for area recovery

- **Covered:**
  - (a) tradeoff run

#0. Setup: Running the ATN_OPT Shell

- First, set up a new “demo-1” subdirectory:
  - Go to root directory (“dist”) of the unpacked atn_opt tool (use ‘cd’)
  - Create a new subdirectory there *(or modify steps to create it elsewhere):*

    mkdir demo-1
  - Copy the files from “examples” into “demo-1”:

    cp examples/* demo-1
  - Go to the “demo-1” subdirectory:

    cd demo-1

#0. Setup: Running the ATN_OPT Shell

- Next, start up the “atn_opt” tool:
  - Type "**atn_opt**”: a prompt (#) will appear

    Asynchronous Threshold Network Optimization v.0.1

    #
- Type "help" to list supported commands:

    # **help**

#1. Basics: starting from a single-rail logic network

(a) Setup: reading in a netlist/library

- To perform optimization, a cell library and a netlist must be read into the shell
- Read the cell library first, since a netlist usually requires definitions of cells:

```
# read_genlib thr.lib
library read: # cells = 29
```

- Next, read in the example netlist:
  - We will use **xor.blif**, which is a single-rail Boolean logic network

```
# read_blif xor.blif
netlist read: i/o/g = 2/1/5
```

Example Run #1b. (area optimization)

[skip first 2 commands: already completed on previous slide]

[**# read_genlib thr.lib**
library read: # cells = 29
 # read_blif xor.blif
netlist read: i/o/g = 2/1/5]

 # set cost area
targeted cost function: area
 # relax
relax: 1 relaxed out of 3 (33.3333% relaxed)
 # map

[queries to find cost of resulting dual-rail asynchronous netlist: area, power, worst-case delay (under fixed-delay model)]

 # area
area: 56.3
 # power
power: 65.2
 # delay
max delay:13.54 at z_1

[save netlist in blif format]

 # write_blif tmp

 # more tmp

```
.model tmp
.inputs i1_0
.inputs i1_1
.inputs i2_0
.inputs i2_1
.outputs z_0
.outputs z_1
.subckt g12x0 and1 a=i1_0 b=i2_1 O=and1_0
.subckt g22x0 and2 a=i1_0 b=i2_1 O=and2_1
.subckt g22x0 z a=and1_0 b=and2_1 O=z_c1
.subckt g22x0 and1 a=i1_1 b=i2_0 O=and1_1
.subckt g22x0 and2 a=i1_1 b=i2_0 O=and2_c0
.subckt g22x0 and2 a=i1_1 b=i2_1 O=and2_c1
.subckt g22x0 and2 a=i1_0 b=i2_0 O=and2_c2
.subckt g13x0 and2 a=and2_c0 b=and2_c1
      c=and2_c2 O=and2_0
.subckt g22x0 z a=and1_1 b=and2_0 O=z_c2
.subckt g22x0 z a=and1_1 b=and2_1 O=z_c3
.subckt g13x0 z a=z_c1 b=z_c2 c=z_c3 O=z_1
.subckt g22x0 z a=and1_0 b=and2_0 O=z_0
.end
```

Note: “AND” cells in the thr.lib library are used to represent corresponding C-elements in the resulting dual-rail circuit, while OR cells are used as themselves.

Digression: How to Model Sequential Threshold Cells in the Library?

- **Goal:** to define "threshold cells". Examples:
  - **1-of-N threshold cell** *("g1N"* in library) = "N-input OR-gate"
  - **N-of-N threshold cell** *("gNN"* in library) = "N-input C-element"

    (see given cell library: thr.lib)

- **Modeling Approach**:
  - **Combinational Threshold Cells (e.g. 1-of-N)**: specified directly in library
    - e.g. OR2 gate *("g12"):* ***modeled by OR2 gate***
      - indicates correct Boolean function: "O(utput) = a+b"
  - **Sequential Threshold Cells (e.g. N-of-N)**: specified implicitly in library
    - e.g. 2-input C-element *("g22"):* ***modeled by AND2 gate***
      - model using corresponding combinational library cell
      - only indicates 'set' (i.e. evaluate) function: "O(utput) = a*b" ***(like "AND2"!)***
      - implicit meaning: *g22 is* sequential; output O resets only when inputs a=b=0

Example Run #1c. (delay optimization)

[can skip first command, library already read in]

[**# read_genlib thr.lib**
library read: # cells = 29]

 # read_blif xor.blif

netlist read: i/o/g = 2/1/5

 # set cost delay

targeted cost function: load-dependent delay

 # relax

relax: 1 relaxed out of 3 (33.3333% relaxed)

 # map

[queries to find cost of resulting dual-rail asynchronous netlist: area, power, worst-case delay (under load-dependent delay model)]

 # area

area: 98.1

 # power

power: 91.3

 # delay

max delay:10.6 at z_1

[NOTE: much better delay than Run #1b, but at the expense of increased area/power]

[save netlist in blif format]

 # write_blif tmp

 # more tmp

```
.model tmp
.inputs i1_0
.inputs i1_1
.inputs i2_0
.inputs i2_1
.outputs z_0
.outputs z_1
.subckt g12x1 and1 a=i1_0 b=i2_1 O=and1_0
.subckt g22x3 and2 a=i1_0 b=i2_1 O=and2_1
.subckt g22x3 z a=and1_0 b=and2_1 O=z_c1
.subckt g22x3 and1 a=i1_1 b=i2_0 O=and1_1
.subckt g22x3 and2 a=i1_1 b=i2_0 O=and2_c0
.subckt g22x3 and2 a=i1_1 b=i2_1 O=and2_c1
.subckt g22x3 and2 a=i1_0 b=i2_0 O=and2_c2
.subckt g13x0 and2 a=and2_c0 b=and2_c1
      c=and2_c2 O=and2_0
.subckt g22x3 z a=and1_1 b=and2_0 O=z_c2
.subckt g22x3 z a=and1_1 b=and2_1 O=z_c3
.subckt g13x0 z a=z_c1 b=z_c2 c=z_c3 O=z_1
.subckt g22x3 z a=and1_0 b=and2_0 O=z_0
.end
#
```

Note: "AND" cells in the thr.lib library are used to represent corresponding C-elements in the resulting dual-rail circuit, while OR cells are used as themselves.

Example Run #1d. (specialized runs)

- Usual runs (#1b-c): include two steps
  - **(i) relax:** expand single-rail to dual-rail circuit
    - optimization = expand some single-rail gates into "eager" dual-rail blocks
  - **(ii) map:** further optimize (I.e. re-map) dual-rail circuit

- Sometimes, skipping steps can be useful:

Case #1. Skip "map"

- **Goal: isolate/evaluate benefits of "relax" optimization**
  - expand each single-rail gate optimally:
    - to either robust or eager dual-rail block
  - no further optimization with "map"
- **Steps:** after reading in library and single-rail netlist

  relax

Example Run #1d. (specialized runs, cont.)

- Sometimes, skipping steps can be useful …:

Case #2. Skip "relax"

- **Goal: isolate/evaluate benefits of "tech map"**
  - no relaxation:
    - expand each single-rail gate directly to robust (non-eager) dual-rail block
    - result = unoptimized "DIMS-style" circuit
  - optimize dual-rail circuit using "map"

- **Steps:** after reading in library and single-rail netlist

```
set cost cktgen
relax                [… 'cktgen' skips optimization, just produces dual-rail circuit…]
set cost area […or delay, fixdelay, …]  [restore desired cost function for map]
map
```

Example Run #1d. (specialized runs, cont.)

- Sometimes, skipping steps can be useful …:

Case #3. Skip both "relax" and "map"

– **Goal: read in single-rail network, simply convert it to dual-rail**
  - avoid all optimization!
  - result is unoptimized "DIMS-style" circuit

– **Steps:** after reading in library and single-rail netlist

```
set cost cktgen
relax                [… 'cktgen' skips optimization, just produces dual-rail circuit…]
```

NOTE: cost function is now 'cktgen'. Immediately after typing 'relax', you should change it back ("set cost …") to your desired usual cost function (area, delay, etc.). (If you don't, your next step will still target 'cktgen'!)

#2. Basics: starting from a dual-rail logic network

(a) Setup: reading in a netlist/library

- As in PART #1, before performing optimization, a cell library and a netlist must be read into the shell

  NOTE: ***unlike PART #1 examples (which were single-rail), these examples are already dual-rail!***

- **Goal:** further optimize these dual-rail circuits by "re-mapping" them

- **Steps:** as before, read in cell library first
  - it is used for both single-rail and dual-rail circuits

```
# read_genlib thr.lib
library read: # cells = 29
```

#2. Basics: starting from a dual-rail logic network

(b) Area run

– Next, read in the example netlist:

- We will use **tr.blif**, which is a dual-rail Boolean logic network

```
# read_blif tr.blif
netlist read: i/o/g = 414/338/8716
```

– Now, run “technology mapping” (**map**) or “cell merger” (**merger**):

```
# set cost area                    […set desired cost function…]
targeted cost function:  area
# map                                    […do tech map only …]
# write_verilog  tr-opt.v          […write results to a new file…]
```

Example Run #3a. (delay-area tradeoff):

how to recover area in delay-oriented runs

Step #1. Do "fixed delay" run

(to obtain worst-case delay bound)

```
# read_genlib thr.lib
library read: # cells = 29
# read_blif dalu.blif
netlist read: i/o/g = 150/60/4047
# set cost fixdelay
targeted cost function: fixed delay
# map
# fixdelay
max delay: 105.85 at 2_0
# area
area: 31639.9
```

Two-Step Strategy: (i) find "max delay" in delay-only run; then **start over**: (ii) use this delay as "reqtime" in delay-area tradeoff run to obtain reduced area (while maintaining same targeted max delay)

Step #2. Do "delay-area" tradeoff run

(using this worst-case delay bound)

```
# read_genlib thr.lib
library read: # cells = 29
# read_blif dalu.blif
netlist read: i/o/g = 150/60/4047
# set cost tradeoff
targeted cost function: delay-area tradeoff
(valid only for techmap/cell merger)
# set reqtime 105.85
reqtime set to: 105.85
# map
# fixdelay
max delay: 105.85 at 2_0
# area
area: 26619
```

obtain worst-case fixdelay and use it as required time

area further reduced while maintaining same worst-case delay

Note: the above example only performs tech map ("map"), given an unoptimized dual-rail circuit (dalu.blif). However, a similar strategy can be used if starting from a single-rail circuit (first use "relax", then "map".)

Alternative Approach for Synthesis Runs: Executing A Script File

- A script file, which consists of shell commands can be created and used either on a UNIX shell or on a ATN_OPT shell.
  - To execute a script file on a UNIX shell, use following command:

    > **atn_opt <script_file>**
  - To execute a script file inside a ATN_OPT shell, use "source" shell command:

    # **source <script_file>**

Other ATN_OPT Commands (1): Setting Parameters

- ATN_OPT allows updating of synthesis parameters:
  - **use “set” command**
- Default values:
  - **cost:** load-dependent delay
  - **required time:** not specified (-1)
  - **no. of loadbins:** 20
  - **distinguish cell i/o path:** true
  - **distinguish rise/fall delay:** true
  - **verbose mode:** false
- At any time, can check current settings of parameters:
  - **type: “print_options”**

Other ATN_OPT Commands (2): Setting Parameters

- IMPORTANT NOTE:
  - ***Once a parameter is "set" (or at its default value), its value "persists" (i.e. remains set) until changed!***
- Advantage:
  - You can do multiple synthesis runs with same parameters (i.e. target costs, verbose mode, etc.), without explicitly setting them for each run
- Danger:
  - You may use old (stale) parameter settings for new runs, if you are not careful!
- How to Avoid Danger…:
  - check current parameter settings regularly: use "**print_options**"

Other ATN_OPT Commands (3): Setting Parameters

- An Addendum: an issue using "**set reqtime**"
  - **Default value** = "not specified (-1)"
  - **ATN_OPT tool limitation:**
    - ***Once you modify 'reqtime', you can never reset it back to default '-1'!***
  - **Solution:**
    - ***To reset 'reqtime' to 'not specified', set it to a very large positive number (so it will have no effect on results)***

Cell Library (1): Modeling Single-Rail vs. Dual-Rail Circuits

- ATN_OPT uses the same library to model both single- and dual-rail circuits!
  - **for single-rail circuits:** ATN_OPT ignores most cell parameters
    - only parameter used: cell's "function" [O(utput) = …]
    - cells simply used to structurally model initial single-rail circuit
    - **each cell =** treated as a combinational (i.e. Boolean) function
      - e.g. g22 = "AND2 gate"
  - **for dual-rail circuits:** ATN_OPT uses all cell parameters
    - parameters: characterize actual VLSI cells to be used in final dual-rail circuit
    - algorithms (relax/map/merger) *directly target* dual-rail cell parameters
      - to obtain optimal dual-rail implementation
    - **each cell =** treated as a threshold function
      - e.g. g22 = … now viewed as a "2-input C-element"

Cell Library (2): How to Extend and Update

- ATN_OPT v0.1 includes one cell library: **thr.lib**
- ATN_OPT can flexibly handle different libraries:
  - User can:
    - modify parameters of cells in 'thr.lib'
    - add new cells/delete old cells in 'thr.lib':
      - ***can include complex non-inverting gates: AND-OR, OR-AND, etc.***
    - create whole new libraries: must use "GENLIB" format
  - Restrictions:
    - **cell parameters:** must follow same field types and ordering as in 'thr.lib'
    - **cell types:** must follow "*restriction on cell library*" (see slide #43):
      - negative logic gates: only INVERTER allowed (for use in single-rail)
      - *otherwise ATN_OPT will not guarantee correct relax/map steps!*

Current Restrictions (1): Single-Rail Logic Networks

- Currently, two restrictions on initial single-rail network:

  (a) negative logic: only inverters allowed!

  – no NAND, NOR, AOI, etc. gates allowed in initial single-rail network

  (b) gate fan-in restrictions: all gates must have fan-in of 3 or fewer!

  – "RELAX" currently cannot handle networks with larger fan-in gates

- **If initial single-rail network violates (a) or (b)?**

  Option #1. Create simple automated pre-processing script -- run on initial netlist:

  - break NAND into "AND + INV" (NOR into "OR + INV", etc.)
  - break N-input AND (N >3) into network of smaller AND's, etc.

  Option #2. Re-derive initial single-rail network -- now use restricted cell library:

  - library: has restricted fan-in gates (3 or fewer) + only INV for negative logic
  - … re-derive netlist with SIS or commercial tools targeting this library

Current Restrictions (2): Cell Libraries

- Currently, one restriction on cell library:

  negative logic: only inverters allowed!

  - NAND, NOR, AOI, OAI, etc. not allowed
  - **For single-rail logic network:**
    - » only negative logic gate used = INVERTER
  - **For dual-rail logic network:**
    - » no negative logic gates used
    - » dual-rail asynchronous threshold networks: only use 'positive unate' gates

  (see given cell library: thr.lib)

Conclusions

- ATN_OPT for optimizing robust asynchronous threshold networks
  - Supports two optimization algorithms
    - Relaxation
    - Technology mapping /cell merger
  - Supports various cost functions
    - single cost functions: area, load-dependent delay, fixed delay, power
    - hybrid cost functions: delay-area tradeoff
  - Supports multiple execution modes
  - Supports common circuit representations: VHDL, Verilog, BLIF
  - Supports extensible/modifiable cell library: GENLIB format