An Introduction to Burst-Mode Controllers and the MINIMALIST CAD Package

(Release v2.0)

Steven M. Nowick

Columbia University

(nowick@cs.columbia.edu)

November 24, 2007

MINIMALIST: Funding Acknowledgments

v2.0 Release (NEW):

This work was supported by NSF ITR Award No. NSF-CCR-0086036.

v0.9 and v1.0-v1.2 Releases (1994-2001):

This work was supported by NSF Award Nos. NSF-CCR-99-88241, NSF-MIP-95-01880 and NSF-MIP-9308810; a research grant from IBM Corporation; and an Alfred P. Sloan Research Fellowship.

The MINIMALIST v2.0 Package: Introduction

MINIMALIST: developed at Columbia University [1994-]

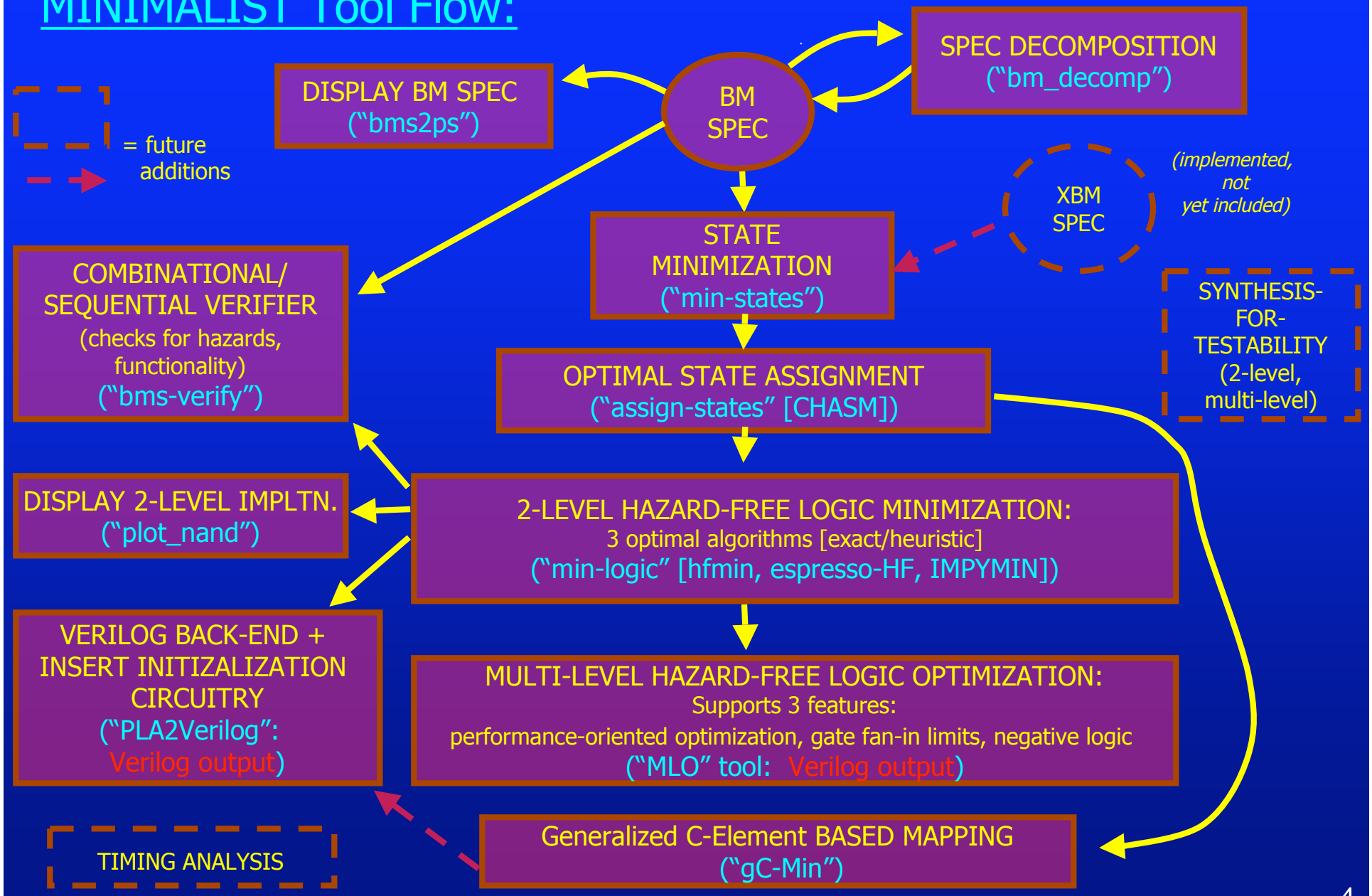
- extensible "*burst-mode*" synthesis package
- integrates synthesis, testability and verification tools

Synthesis flow for individual asynchronous controllers:

- Includes several optimization tools:
  - State Minimization
  - CHASM: optimal state assignment
  - 2-Level Hazard-Free Logic Minimization: exact/heuristic
  - Multi-level logic optimizer tool ("MLO"): includes performance-oriented decomposition
  - Decomposing large specifications ("bm_decomp")
- Other practical features:
  - Automated scripts, manual command-line interface
  - Verilog back-end + auto insertion of initialization circuitry ("pla2verilog")
  - Top-to-bottom verifier ("bms-verify")
  - "GC-Min": mapping to generalized C-elements
  - Graphical display ("bms2ps": specifications; "pla2nand": implementations)

The MINIMALIST v2.0 Package: Overview

MINIMALIST Tool Flow:



The MINIMALIST v2.0 Package: New Features (highlights)

- Multi-Level Optimizer (MLO) Tool: Comprehensive Package
  - Stand-alone back-end translator, from two-level (.sol/.pla) to multi-level (.v)
  - Key features: can "mix-and-match"
    - performance-oriented multi-level logic decomposition:
      reduces critical input-to-output paths (auto/manual modes)
    - gate fan-in restriction:
      user can specify fan-in limits to gates
    - target negative logic gates:
      map only to negative logic gates
  - Verilog translator: produces multi-level Verilog output (.v file)
  - Runs directly in Linux shell (outside of Minimalist)
  - Current restriction:
    - cannot run after PLA2Verilog (so cannot include initialization circuitry in multi-level output)

See separate "MLO" package:
tutorial, examples, documentation, etc.

The MINIMALIST v2.0 Package: New Features (highlights, cont.)

- bm_decomp: decomposition of Burst-Mode specifications
  - Stand-alone front-end translator
    - Input: a single (monolothic) BM specification (.bms file)
    - Output:
      - a set of several interacting BM specifications, implementing the same behavior
      - some auxiliary hardware indicated (must be manually inserted):
        input latches + latch controllers, output generators
  - Potential benefits: especially for large controllers
    - Runtime: (often) much faster to synthesize smaller decomposed controllers
    - Low Power: (potentially) only one smaller controller active at a time
    - Timing Assumptions: smaller next-state logic ==> narrower fundamental mode window
  - Runs directly in Linux shell (outside of Minimalist)

See separate "bm_decomp" package: tutorial, examples, documentation, etc.

The MINIMALIST v2.0 Package: New Features (highlights, cont.)

- PLA2Verilog: comprehensive Verilog back-end
  - Verilog back-end translator:
    - translates Minimalist output (2-level circuit [PLA file]) to Verilog
  - Automatic insertion of initialization circuitry (hazard-free)

- bms-verify: top-to-bottom verification tool
  - Compares original BM specification directly against final 2-level implementation
    - Sequential + combinational verification:
      - exhaustively simulates entire BM spec, and checks against 2-level logic impltn.
    - Checks for: functional correctness, hazard-freedom

- bms2ps: graphical display of BM specifications
  - Translates BM specification (.bms file) to Postscript graphics (.ps file)
  - Runs directly in Linux shell (outside of Minimalist)
  - Much improved quality over (non-supported) earlier "plot_qt" tool

MINIMALIST Developers

Principal Architects: [1994-present]

- Robert M. Fuhrer: system designer & primary implementer
- Steven M. Nowick: project leader
- Tiberiu Chelcea: coordinating v2.0 updates

Documentation:

–Overview Chapter (includes a good readable introduction to Minimalist, see section on "burst-mode controllers"):

Luciano Lavagno and Steven M. Nowick,
"Asynchronous Control Circuits", chapter 10 of
"Logic Synthesis and Verification",
(editors S. Hassoun and T. Sasao),
Kluwer Academic Publishers, Boston, MA

–Book: Robert M. Fuhrer and Steven M. Nowick,
"Sequential Optimization of Asynchronous and Synchronous Finite-State Machines: Algorithms and Tools",
Kluwer Academic Publishers,
Boston, MA (2001), ISBN 0-7923-7425-8.

–PhD Thesis: Robert M. Fuhrer, *<same title as book>,*
Columbia University, Dept. of Computer Science, May 1999.

Other Contributors

Current and Former PhD Students:

- Melinda Agyekum:

  "bm_decomp" Tool [v2.0 feature]: decomposes BM specifications.
  A standalone front-end tool, which takes a single BM controller specification and decomposes it into set of equivalent interacting BM specifications.
  These interacting controllers can then be synthesized using Minimalist (some added auxiliary hardware required).

  [see separate tutorial + docs]

- Tiberiu Chelcea:

  "pla2nand": two-level circuit display

  "pla2verilog" [v2.0 feature]: Verilog backend for 2-level logic, also performs automatic insertion of initialization logic

- Michael Theobald: advanced 2-level minimization: espresso-HF, IMPYMIN

- Luis Plana: state minimization w/fedback outputs (an initial contributor)

Other Contributors

Current MS Students:

–Walter Dearing:

Multi-Level Optimizer ("MLO") Tool [v2.0 feature]:

MLO is a standalone back-end tool which takes a 2-level circuit produced by Minimalist and maps to multi-level Verilog output.

Features: supports …

(i) performance-driven multi-level logic optimization ("CEO"),

(ii) gate fan-in limitations, and

(iii) negative logic gates

[see separate tutorial + docs]

Other Contributors

Former Undergraduates:

- Charles O'Donnell:

  "bms-verify" [v2.0 feature]: complete BM spec-to-implementation verification check

  "minxbm": XBM support (completed, in preparation for future release)

- Alexander Shapiro: gC-min, gC-CHASM, phase optimization

- Tao Wu: espresso-HF (contributor)

MINIMALIST: Download Site

Minimalist is part of the "CaSCADE" Release of Async CAD tools

Accessible on the Web from:

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

Currently, one version: Linux*

*NOTE: the v2.0 Minimalist release only supports Linux, earlier versions for SPARC Solaris have not been updated and are not fully supported

Includes:

complete tutorial (text + PDF slides)
benchmark examples
other documentation

Outline

PART I: Technical Overview

- The MINIMALIST CAD Package: Introduction
- Optimization Algorithms
  - 2-Level Hazard-Free Logic Minimization
  - Optimal State Assignment
- New MINIMALIST Features
- User-Selectable Modes
- Results, Evaluation and Conclusions

PART II: Tutorial

- Design Examples + Hands-On Tutorial

The MINIMALIST Package: Earlier Features (highlights)

Other Features:

- Graphical Interfaces:
  - displays:
    - state-machine specification
    - circuit implementation
  - menu-based input: (… currently under development)
- Extensible Package:
  - easily accommodates new "plug-in" tools
  - "MinShell": interactive user shell
  - provides: on-line help, command-completion, ...
  - class library for burst-mode manipulation
  - C++ implementation, ~45,000 lines of code

The MINIMALIST Package: Earlier Features (highlights, cont.)

Unlike most other asynchronous packages, MINIMALIST offers designers flexibility:

- Fully-automated synthesis using scripts:
  - Target speed, area, runtime
    - e.g. 'minimalist-speed-suite', 'minimalist-area', etc.
  - Options: produces one vs. multiple implementations (user selects best one)
- Advanced operation: custom synthesis with command-line interface ("MinShell"):
  - Allows advanced users to custom-select each synthesis step
    - targeted machine style (fedback outputs, no fedback outputs)
    - logic implementation style (how much sharing of logic between outputs)
    - cost functions
    - varying encoding lengths (in state assignment)
    - output "phase optimization" (inverted or non-inverted outputs)
    - alternative heuristics for steps:

      skipping state minimization step, CRF-only state encoding (avoid optimal algorithms), etc.

The MINIMALIST Package

Includes some highly-optimized existing *(non-asynchronous!)* CAD tools to solve compute-intensive sub-problems:

- dichot: exact dichotomy solver [Saldanha 91]
- NOVA: simulated annealing -heuristic dichotomy solver [Villa 89]
- espresso (Berkeley SIS): prime implicant generation
- Scherzo: unate/binate covering [Coudert 94]

"Burst-Mode" Controllers

Synthesis style for individual asynchronous FSM's:

- Mealy-type
- allows:
  - *multiple-input changes*
  - concurrent behavior
- target technology: normal synchronous cell libraries
- optimization algorithms: comprehensive set

- Brief History:...
  - Based on informal approach at HP Labs:
    - Davis, Coates, Stevens [1986-, and earlier]
  - Formalized and constrained at Stanford: Nowick/Dill [91]
    - Finalize formal Burst-Mode specifications
    - Nowick/Dill first to develop a correct synthesis method

Burst-Mode: Implementation Style

"Huffman Machine": async machine, no explicit latches

Burst-Mode: Implementation Style

Burst-Mode Behavior: inputs in a user-specified *'input burst'* arrive, in any order (glitch-free)

Burst-Mode: Implementation Style

Burst-Mode Behavior: inputs in a user-specified *'input burst'* arrive, in any order

Burst-Mode: Implementation Style

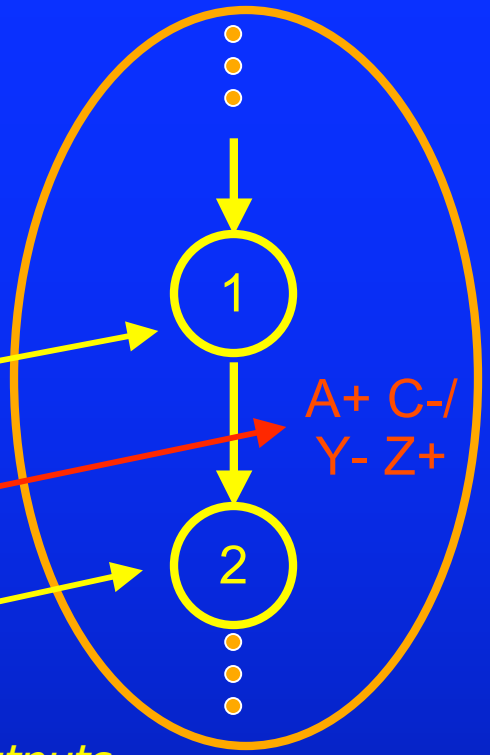
Burst-Mode Behavior: once *'input burst'* is complete, machine generates a (glitch-free) *'output burst'* ...

Burst-Mode: Implementation Style

… and *(sometimes!)* a concurrent (and glitch-free) *state change* to a new state….

Burst-Mode Specifications

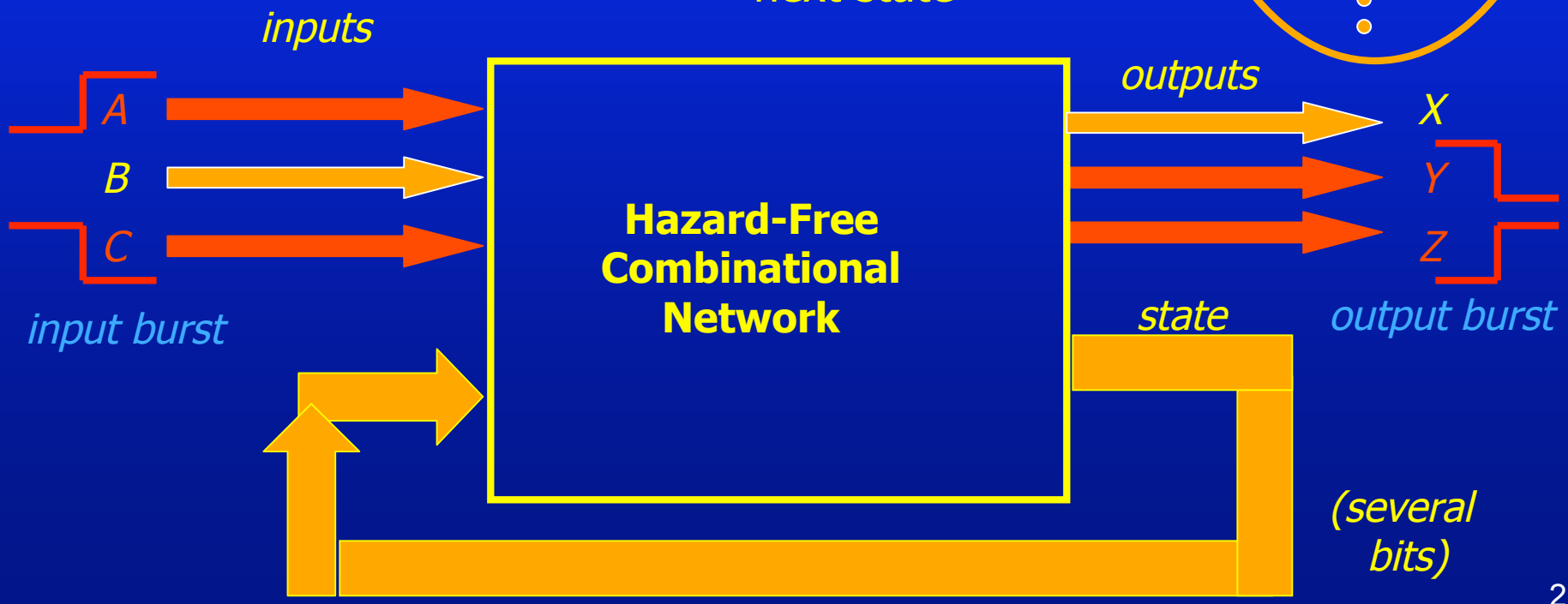
How to specify "burst-mode" behavior?:



current state

input burst/
output burst

next state



Burst-Mode Specifications

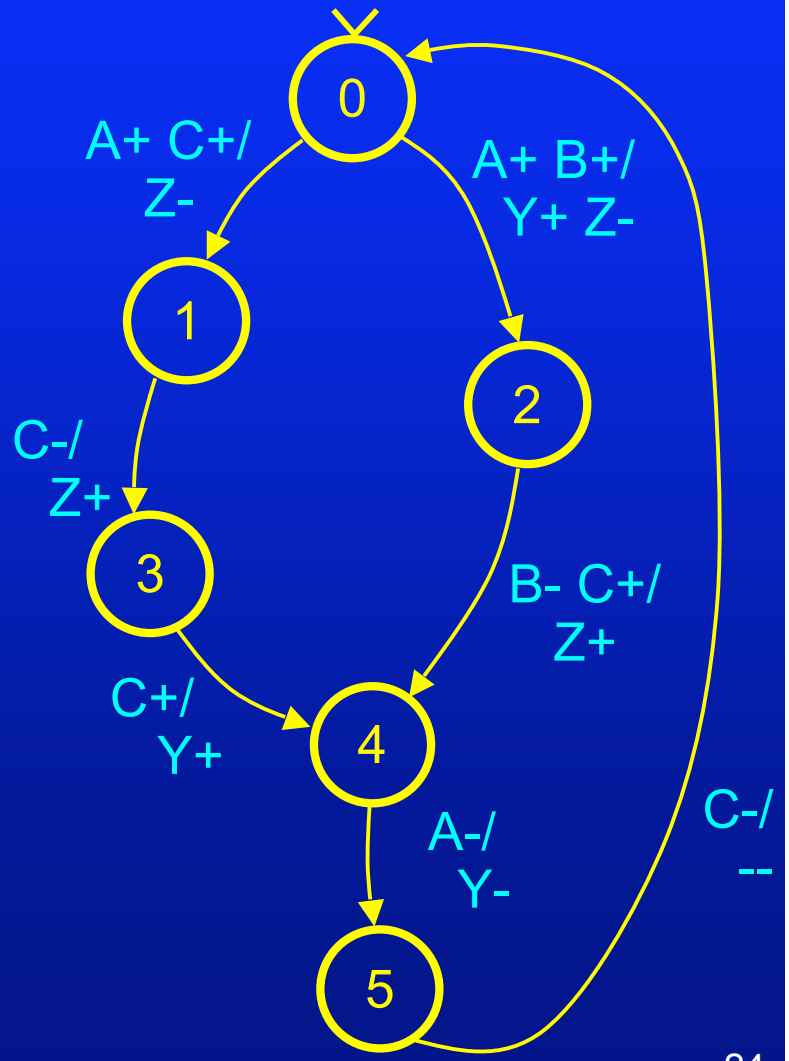
Example: Burst-Mode (BM) Specification:

Initial Values:
ABC = 000
YZ = 01

- Inputs in specified *"input burst"* can arrive in any order and at any time

- After all inputs arrive, generate *"output burst"*

Note:
-input bursts: must be non-empty
(at least 1 input per burst)
-output bursts: may be empty
(0 or more outputs per burst)



Burst-Mode Specifications

"Burst-Mode" (BM) Specs: 2 Basic Requirements

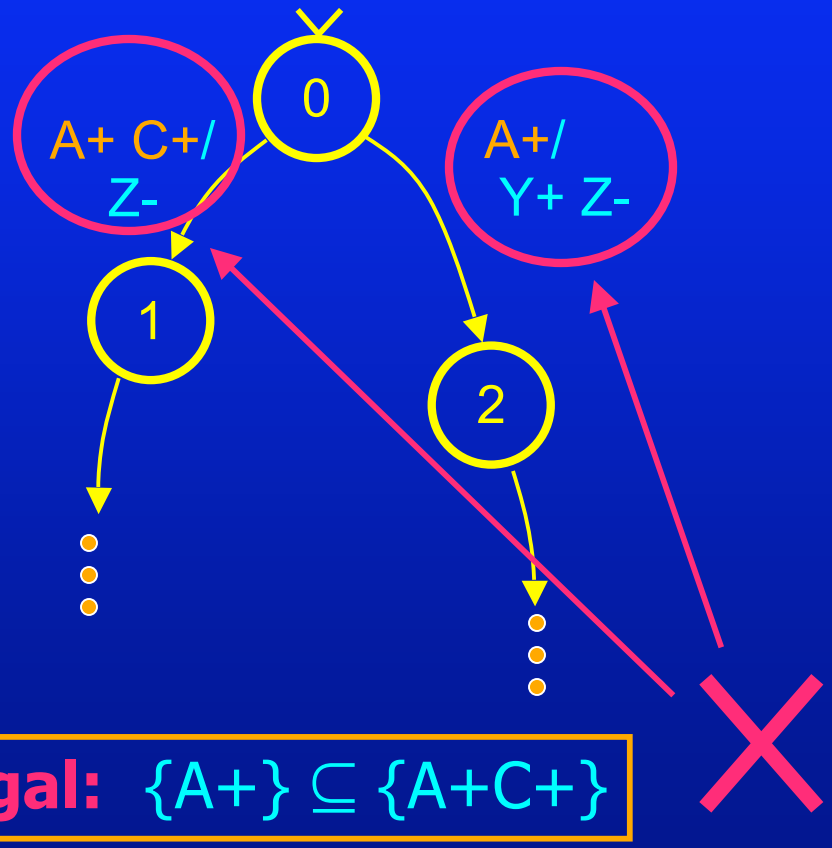
- requirements introduced by Nowick/Dill [ICCD'91,ICCAD'91]
- *… guarantee hazard-free synthesis!*

1. *"maximal set property":* in each specification state, no input burst can be a subset of any other input burst

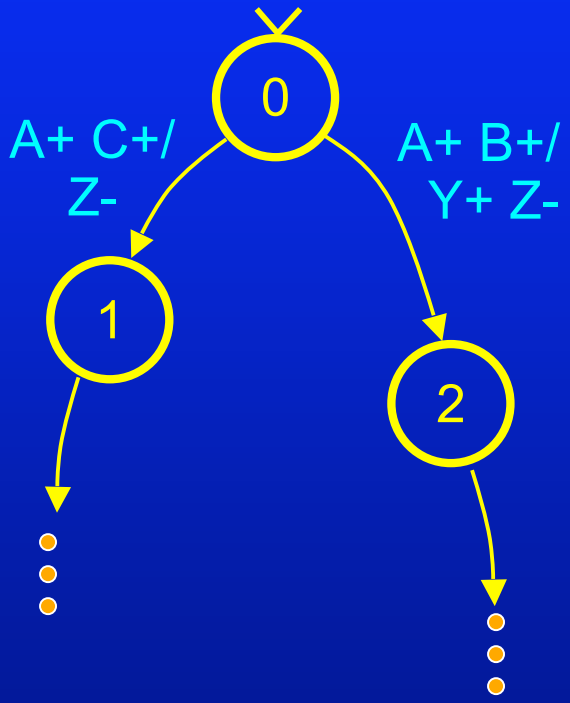
2. *"unique entry point":* each specification state must be entered at a 'single point'

Burst-Mode Specifications

1. "maximal set property": in each specification state, no input burst can be a subset of another input burst



illegal: {A+} ⊆ {A+C+}

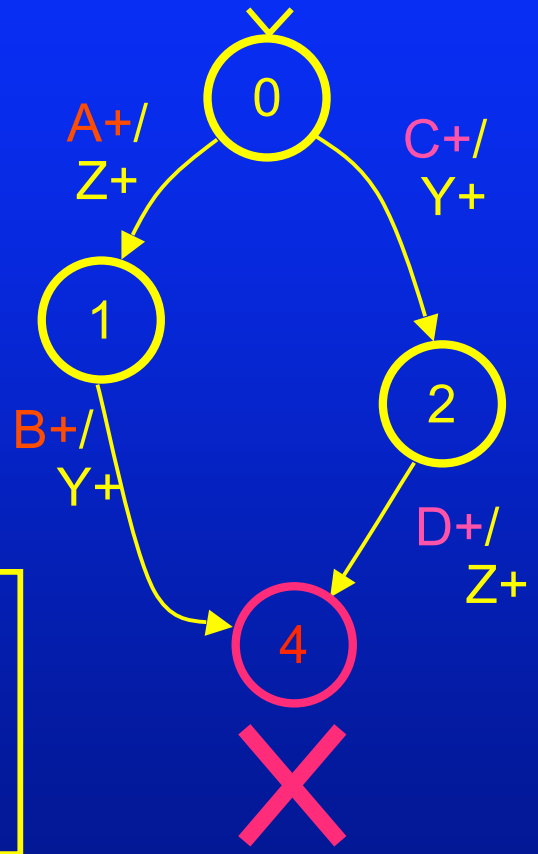


legal

…meaning is ambiguous: what to do when only input A+ arrives?:
- wait for C+? or output Y+ Z-??

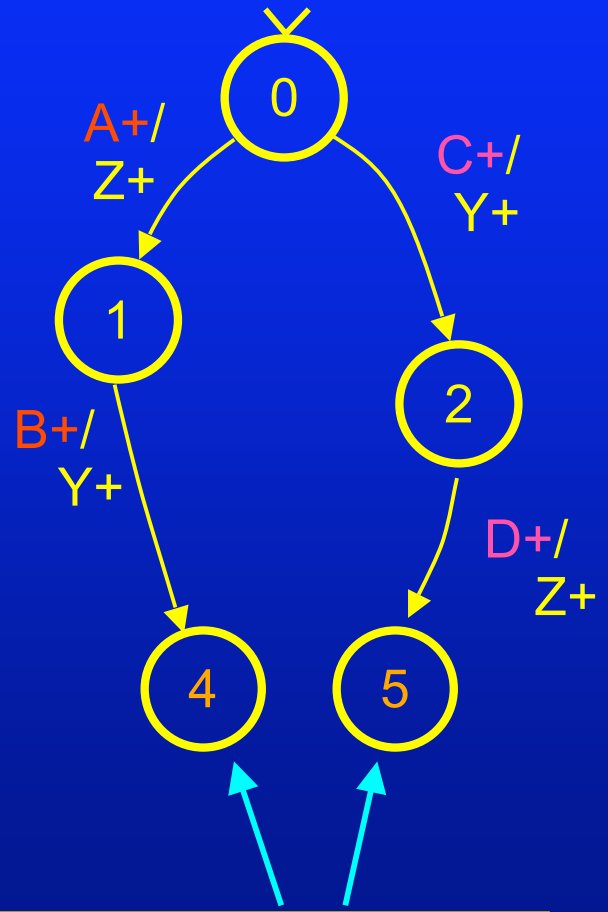
Burst-Mode Specifications

2. "unique entry point": each specification state must be entered at a 'single point' *(guarantees hazard-free synthesis)*



Entering State 4:
- from State 1: **ABCD = 1100 (YZ=11)**
- from State 2: **ABCD = 0011 (YZ=11)**

illegal: 2 different input/output values when entering state 4



legal:
Solution = split state 4

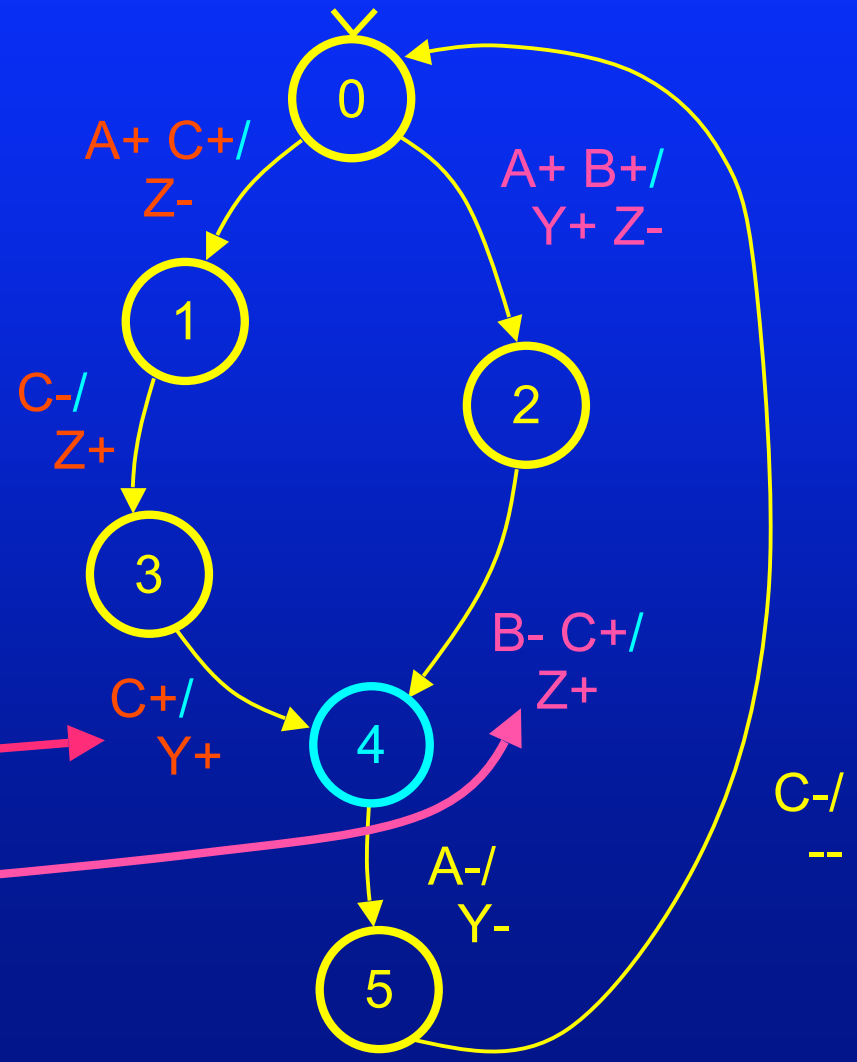
Burst-Mode Specifications

2. "unique entry point" (cont.):

State 0: Initial Values
ABC = 000
YZ = 01

Another Example:

this is legal:
state 4 -- entered with
the same input/output values
on both 'incoming arcs'



Entering State 4:
- from State 3: ABC = **101** **(YZ=11)**
- from State 2: ABC = 101 (YZ=11)

… so, "unique entry point" property is satisfied.

Burst-Mode Specifications

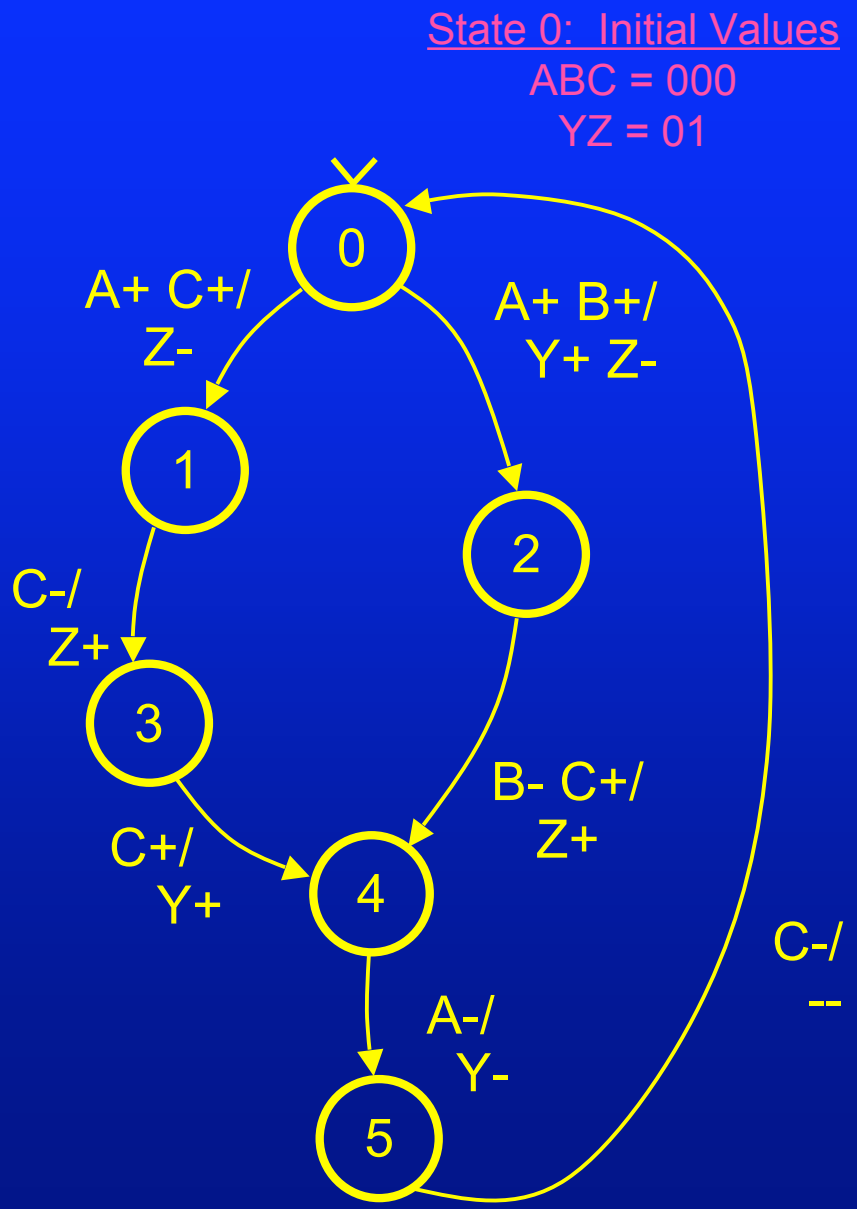
Final observation:

Burst-Mode specs must explicitly indicate *all "expected events"*

Missing input burst:
means "cannot occur"!

EXAMPLE: in State #0…

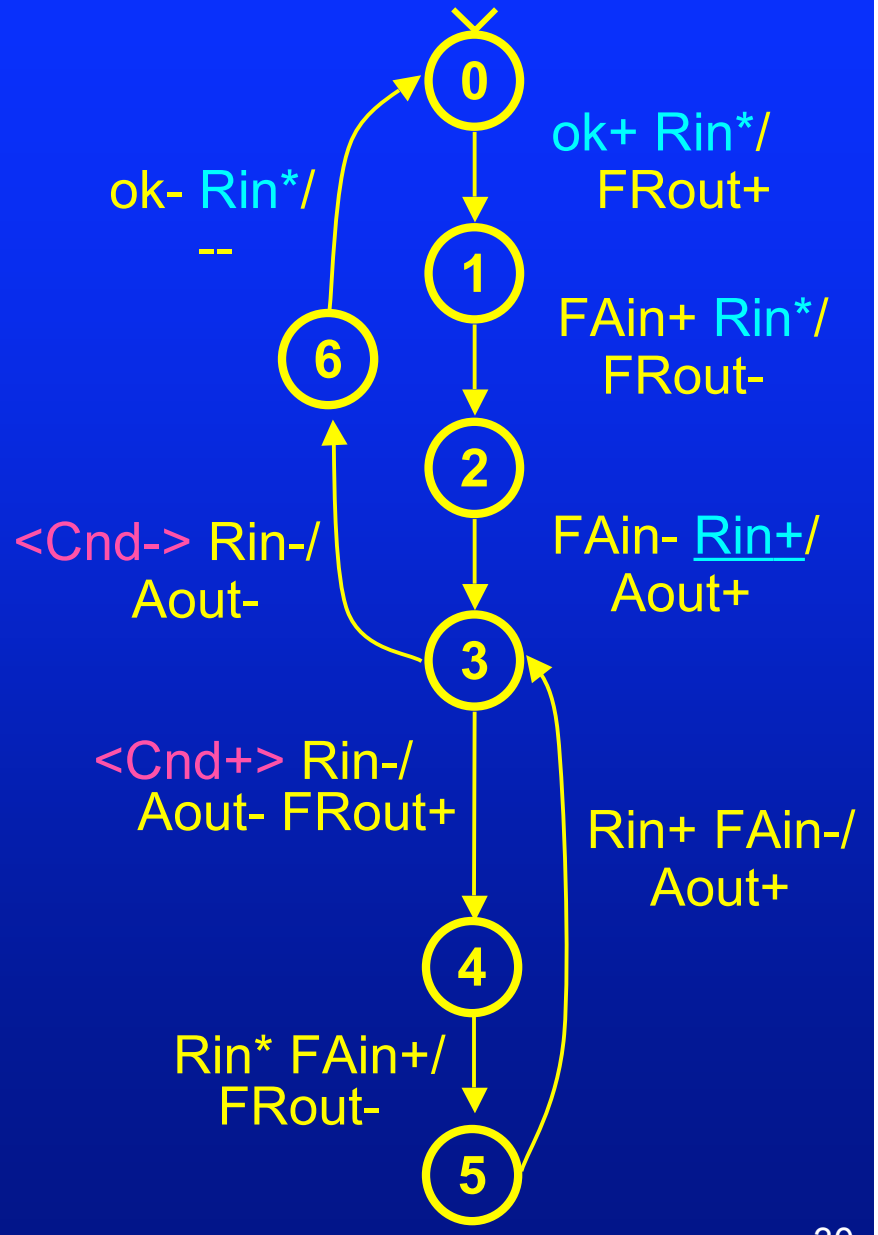
- this specification indicates (implicitly) that input burst B+C+ should never occur
  … since this event is not specified!



Burst-Mode Specifications

"Extended Burst-Mode" (XBM):

[Yun/Dill ICCAD-93/95]



Additional Features:

1. *"directed don't cares" (Rin*):*
   allow concurrent inputs & outputs
2. *"conditionals" (<Cnd>):*
   allow "sampling" of level signals

Handles glitchy inputs,
mixed sync/async inputs, etc.

(… not yet supported by MINIMALIST, expected in future releases)

One-Sided Timing Requirements

#1. Fedback State Change: must *not* arrive at inputs until *previous input burst* has been *fully processed* …

- add: *1-sided delay* to feedback path
- usually negligible delay: often no extra delay needed

One-Sided Timing Requirements (cont.)

#2. Next Input Burst: must *not* arrive until machine *has stabilized* from *previous input+state change* …

- often satisfied: environment usually "slow enough"
- if not: add small delays to outputs

One-Sided Timing Requirements (cont.)

#2. Next Input Burst (cont.): must not arrive until entire machine has stabilized …

"Generalized Fundamental Mode": after each input burst arrives, a machine 'hold time requirement' must be satisfied, before environment can apply the next input burst.

… similar to notion of 'hold-time' for a latch or flip-flop, but now extended to an entire small asynchronous controller.

An Example: "PE-SEND-IFC" Controller (HP Labs)

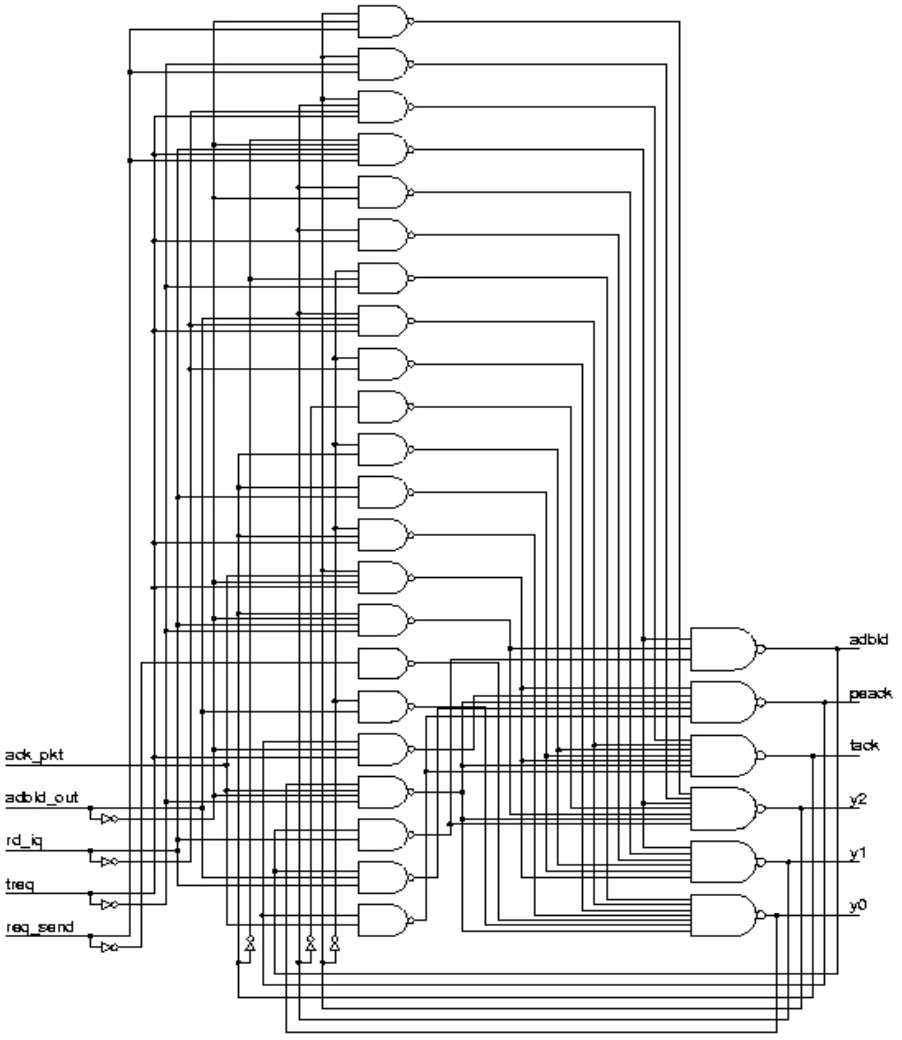
Inputs:
req-send
treq
rd-iq
adbld-out
ack-pkt

Outputs:
tack
peack
adbld

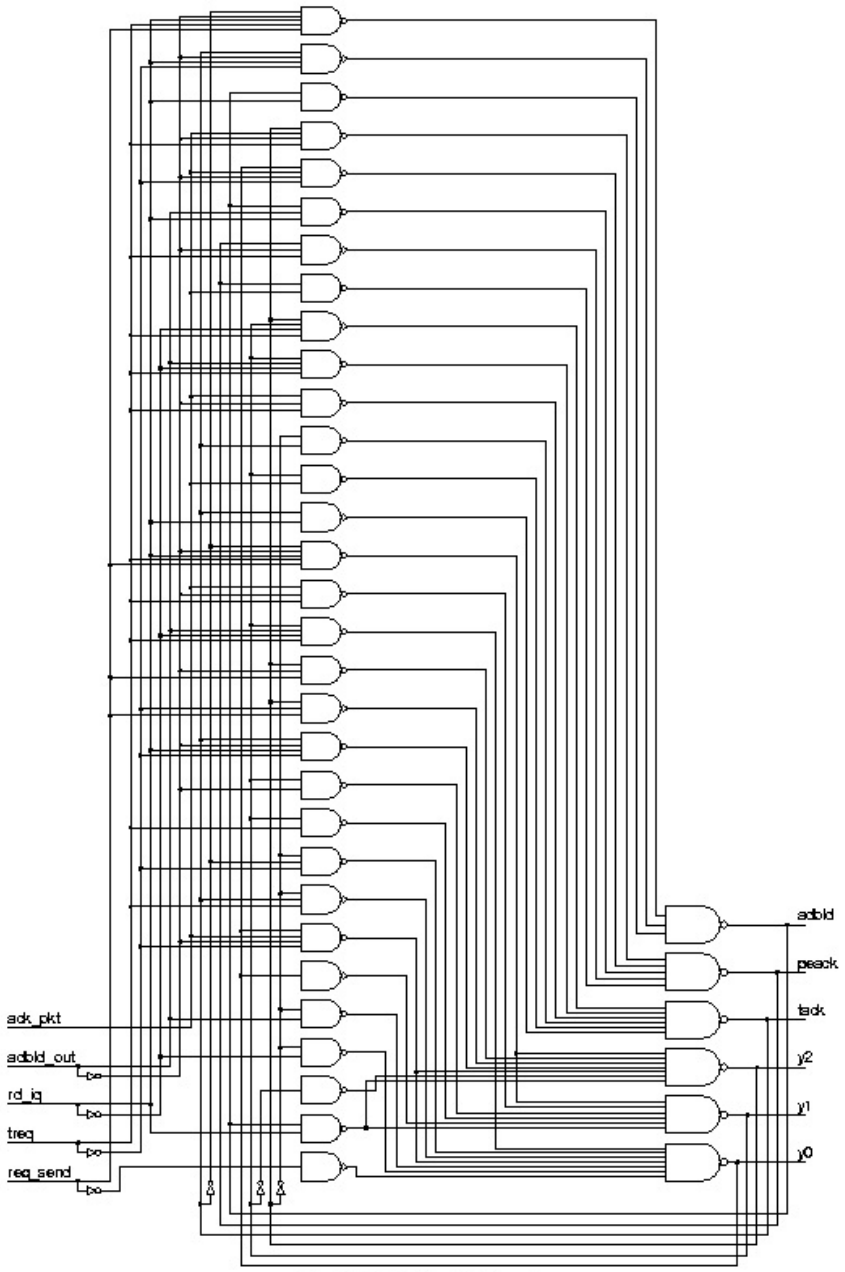
From HP Labs
"Mayfly" Project:
B.Coates, A.Davis, K.Stevens,
"The Post Office Experience: Designing a Large Asynchronous Chip",
INTEGRATION: the VLSI Journal, vol. 15:3, pp. 341-66 (Oct. 1993)

An Example (cont.)

Design-Space Exploration
using MINIMALIST:
optimizing for area vs. speed



PE_SEND_IFC-FL



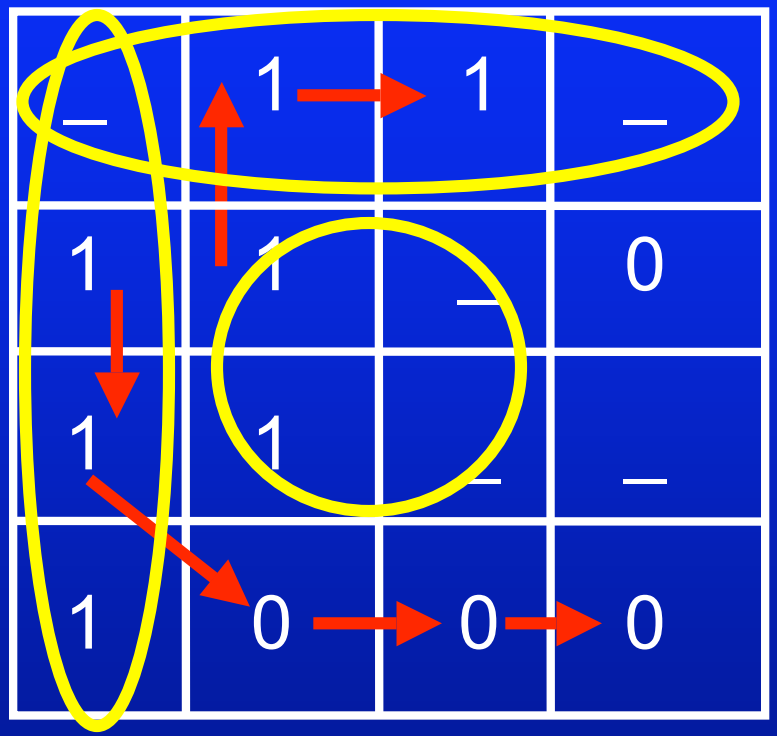
PE_SEND_IFC-Fs

Some Technical Details: Optimization Algorithms

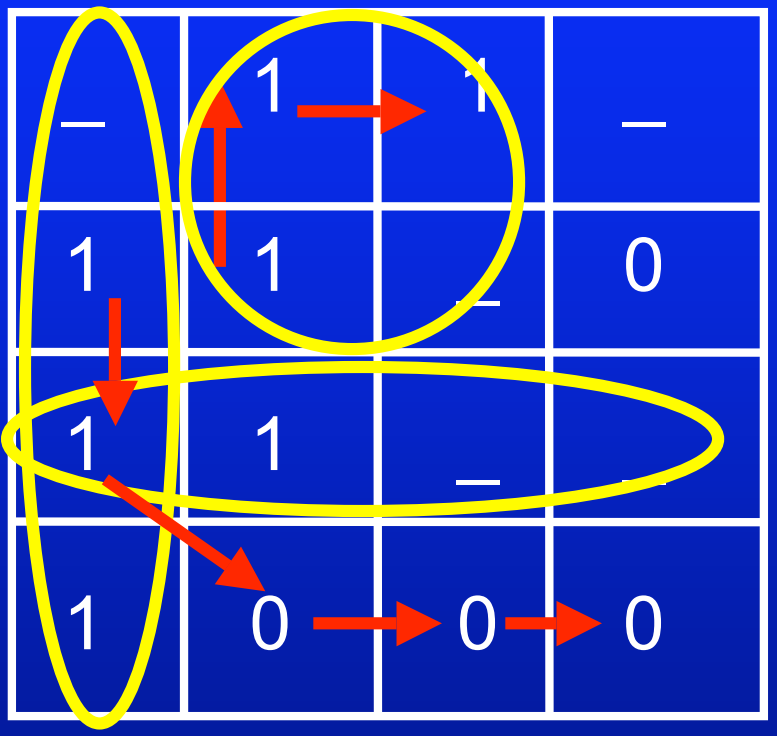
A large set of CAD synthesis, optimization and verification algorithms and tools have been developed:

- 2-Level Hazard-Free Logic Minimization ("min-logic": several modes)
- Optimal State Assignment ("CHASM")
- Multi-Level Logic Optimization ("MLO")
- Controller Decomposition ("bm_decomp")
- Inserting Initialization Circuitry ("pla2verilog")
- Verification: functionality/hazard-freedom ("bms-verify")

2-Level Hazard-Free Logic Minimization: An Example



Non-hazard-free:
min cost = 3 products



Hazard-free:
min cost = 3 products

2-Level Hazard-Free Logic Minimization

Have developed 4 hazard-free logic minimizers:

- **Basic Method:** first exact solver for this problem
  - S.M. Nowick/D.L. Dill:

    (a) ICCAD-92 (IEEE International Conference on Computer-Aided Design),

    (b) IEEE Trans. on Computer-Aided Design, vol. 14:8, pp.986-997 (Aug. 95)
- **HFMIN:** binary and symbolic exact minimizer
  - R.M. Fuhrer/S.M. Nowick, in ICCAD-95.
- **Espresso-HF:** fast heuristic minimizer
- **IMPYMIN:** fast exact minimizer
  - M. Theobald/S.M. Nowick, IEEE Trans. on Computer-Aided Design, vol. 17:11, pp.1130-1147 (Nov. 98)

2-Level Hazard-Free Logic Minimization

HFMIN:

Unlike original (Nowick/Dill ['92]) algorithm:

- handles both binary and symbolic ('multi-valued') inputs

Used in industry and academia:

- academia: in 3D (UCSD), ACK (Utah) and UCLOCK tools
- Intel: used for async instruction-length decoder
- HP Labs (Stetson project: infrared communications chip)
- AMD (SCSI controller)

2-Level Hazard-Free Logic Minimization

IMPYMIN:

Fast algorithm for exact minimization:

- introduces novel method for generating DHF-primes:
  - re-formulates as a *synchronous prime generation problem*
- uses compact *"implicit" data structures*: BDDs/ZBDDs
- calls highly-optimized existing synchronous CAD tools
  - Scherzo [Coudert]

IMPYMIN vs. HFMIN: Results

	I/O	#C (#prods)	HFMIN (in seconds)	IMPYMIN
cache	20/23	97	impossible	301
pscsi	16/11	77	1656	105
sd	18/22	34	172	52
Stetson1	32/33	60	>72000	813
Stetson2	18/22	37	151	49

2-Level Hazard-Free Logic Minimization

ESPRESSO-HF:

Fast heuristic minimization algorithm:

- based loosely on synchronous "ESPRESSO" algorithm
- solves all existing async benchmarks
- *up to 32 inputs/33 outputs:* < 2 minutes
- typical runtime: < 3 seconds

For large examples, usually within 3% of exact solution

CHASM: Optimal State Assignment

[Fuhrer/Lin/Nowick, ICCAD-95]

Overview:

- First general/systematic "optimal state encoding" algorithm for asynchronous state machines
- Based on an "*input encoding model*"
- Modifies synchronous "KISS" algorithm [DeMicheli '85] to insure:
  - *critical race-free encoding*
  - *minimum-cost hazard-free logic*

CHASM: Optimal State Assignment

Special Feature: "Output-Targeted" State Assignment

Observation:

- output logic often determines latency in an async FSM

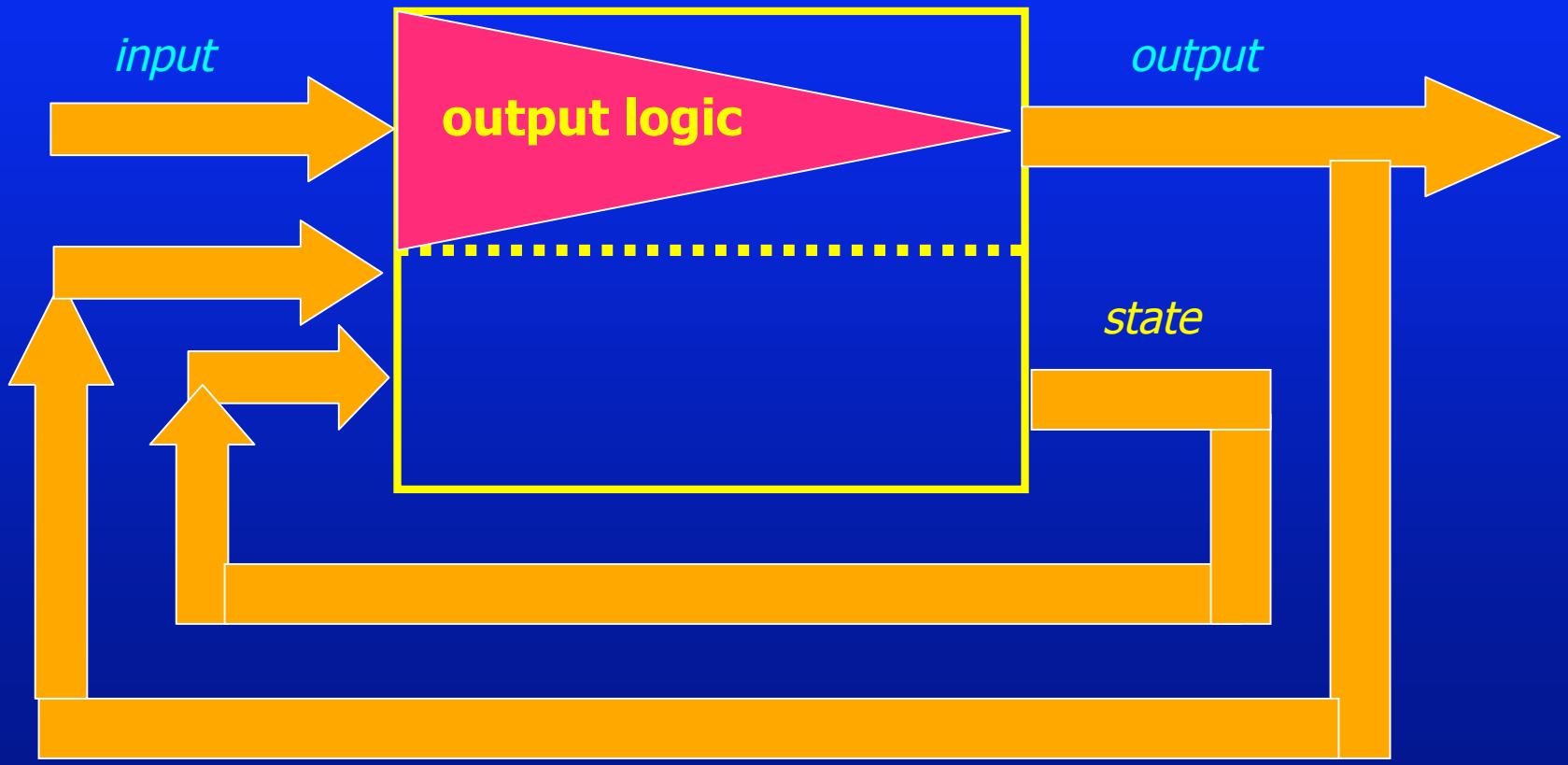
Goal:

- pick state assignment which yields *best output logic*
- … while still insuring "correct" next-state logic (critical race-free)

CHASM produces *exact (i.e. optimal) solution* for this problem

Asynchronous FSMs and Critical Paths

Observation: output logic often critical for async FSM latency



critical path is often *input-to-output*

MINIMALIST v2.0: New Features

Several useful features added to MINIMALIST in release v2.0:

1. Multi-Level Optimizer ("MLO"): multi-level logic optimization (w/Verilog output)
   - *see separate "MLO" tutorial* (Minimalist download site)
2. "bm_decomp": decomposition of large BM specifications
   - *see separate "bm-decomp" tutorial + docs* (Minimalist download site)
3. "pla2verilog": Verilog back-end (2-level) + inserts initialization circuitry
4. "bms2ps": graphical display of BM specifications (creates Postscript)
5. "bms-verify": top-to-bottom verifier, checks BM spec against final implementation
   - *for #3-5, see part II of this Minimalist tutorial (demo part)*

Other Advanced MINIMALIST Features

Several other advanced features previously added to MINIMALIST:

From Earlier Releases:

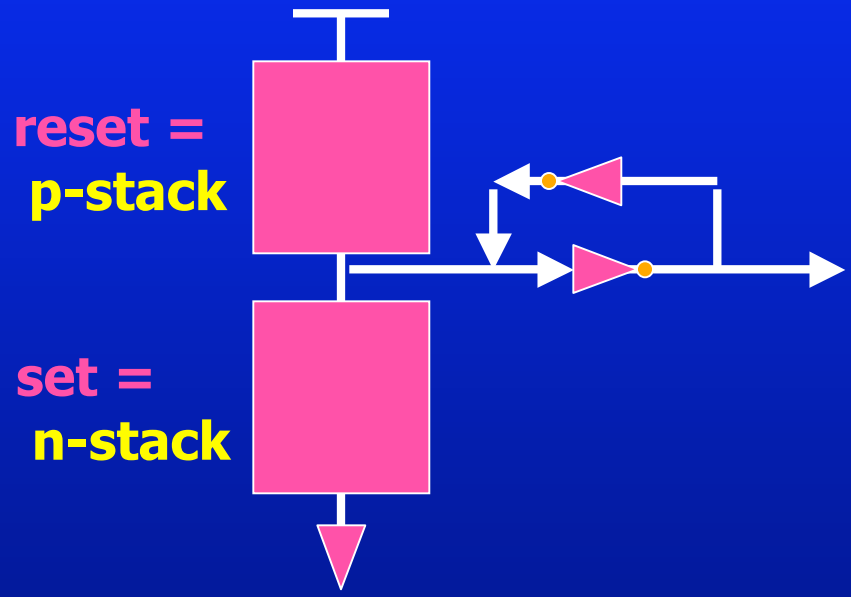
1. Technology Mapping: to "generalized C-elements"
2. Phase Optimization

Other Advanced MINIMALIST Features

#1. gC-Based Technology Mapping

Target = "Generalized C-element":

async sequential component implementing "set" (n-stack) and "reset" (p-stack) conditions

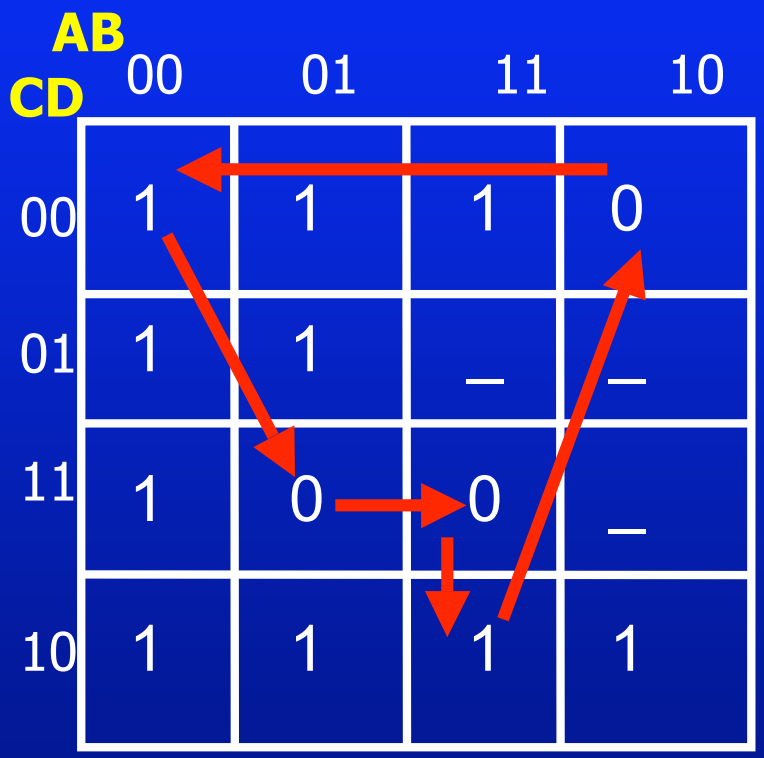


New exact hazard-free gC logic minimizer: *"gC-min"*
[Alexander Shapiro/S. Nowick '00]

Other Advanced MINIMALIST Features

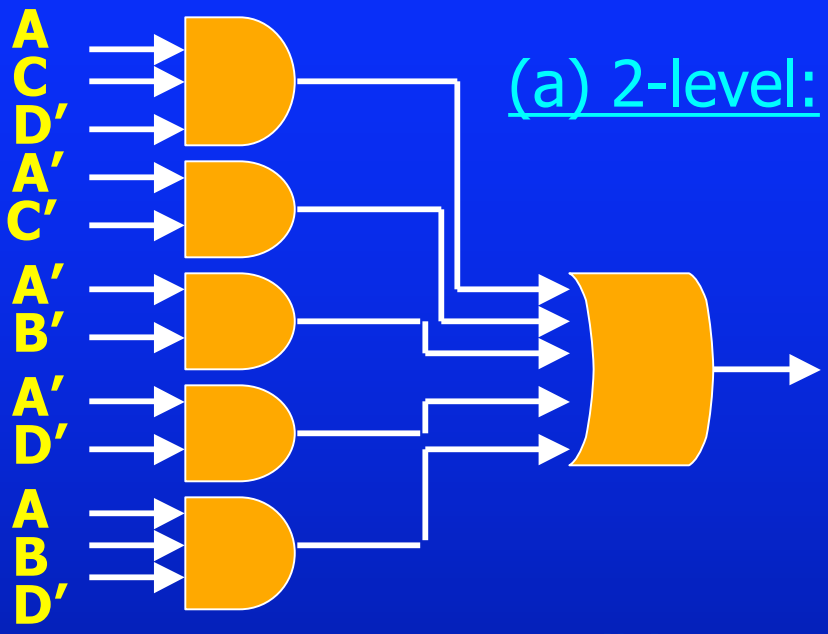
#1. gC-Based Technology Mapping: an Example

(hazard-free synthesis)



Boolean function + specified input transitions

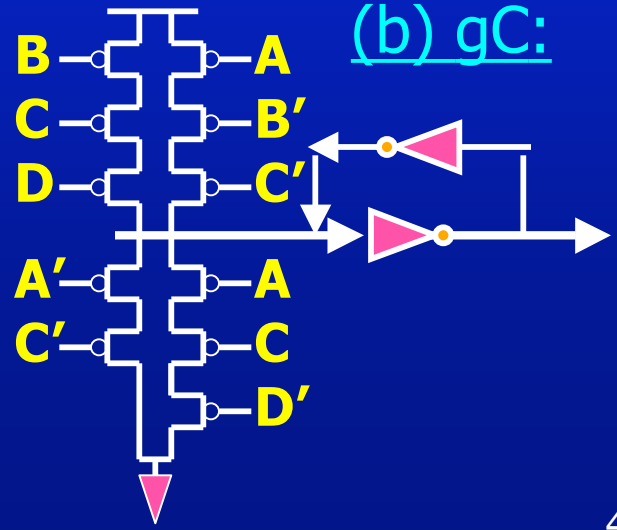
(a) 2-level:



(b) gC:

reset = p-stack

set = n-stack



Other Advanced MINIMALIST Features

#2. Phase Optimization [Alexander Shapiro, S. Nowick '00]

Goal: for each output and next-state function x...

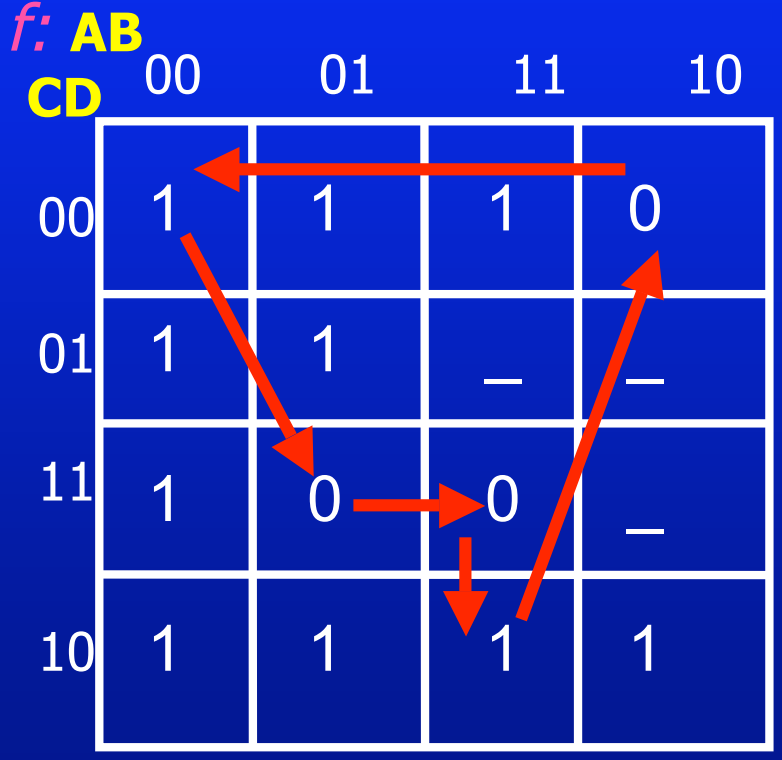
- implement both x and x'
- select best result = *"phase optimization"*
  - if x' selected: add output inverter

Now included in several MINIMALIST steps:

- logic minimization: both 2-level and gC-min
- optimal state assignment:
  - CHASM: target state assignment to selected phase choices

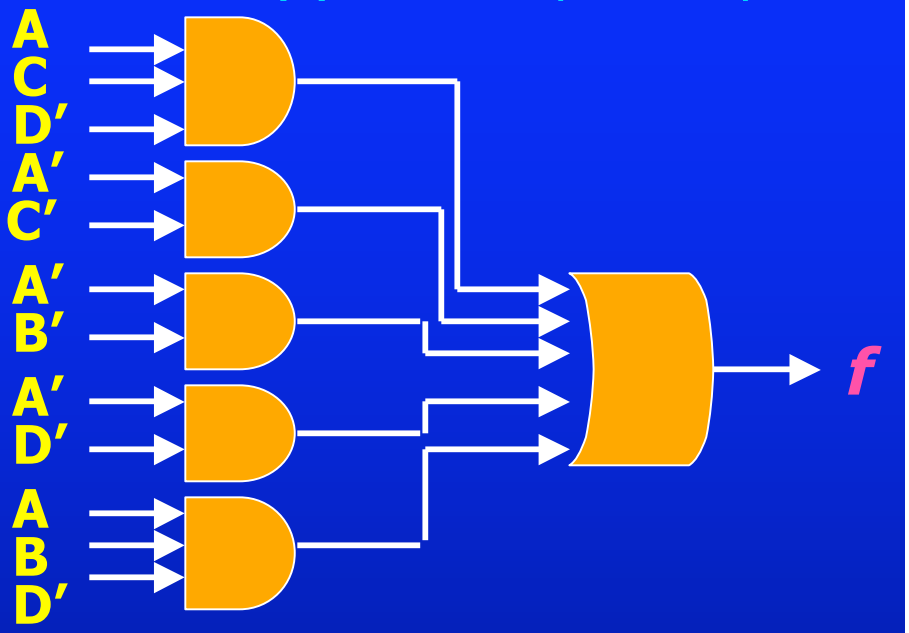
Other Advanced MINIMALIST Features

#2. Phase Optimization:

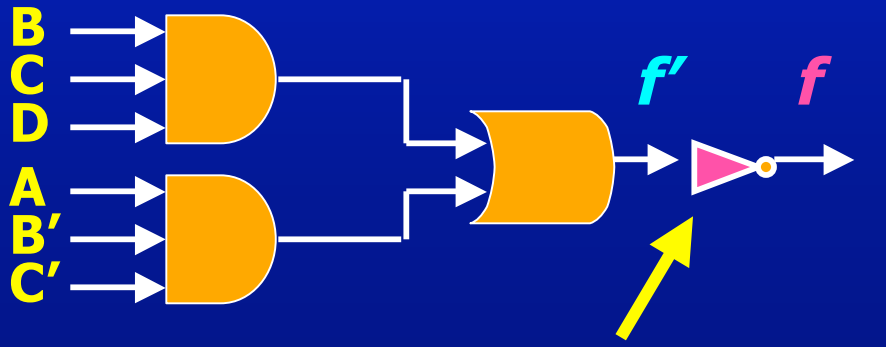


Boolean function +
specified input transitions

(a) Without phase optzn.:



(b) With phase optzn.:



Add inverter:

User-Selectable Modes

MINIMALIST provides several options for *user "design-space exploration":*

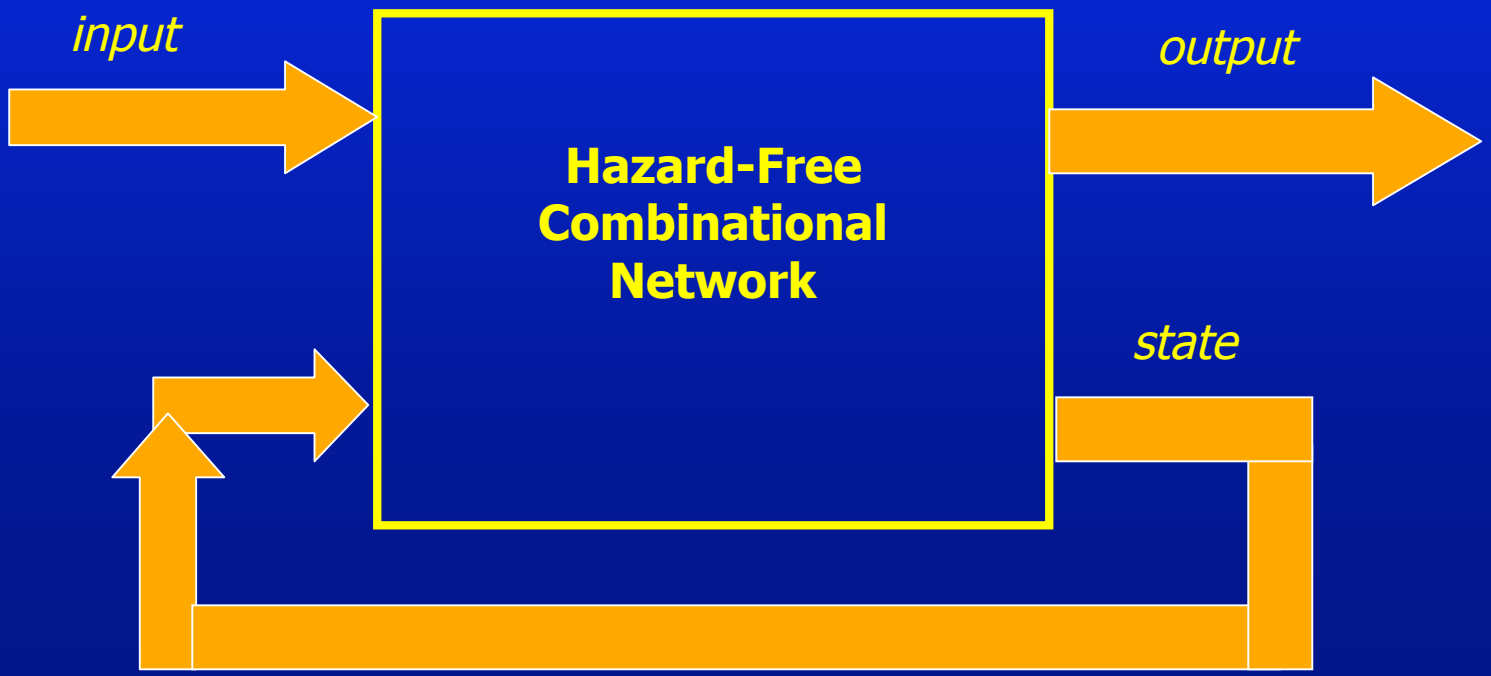
- machine style
- logic implementation style
- cost function
- state assignment style

User-Selectable Modes

Many choices to allow user "design space exploration":

#1. Machine Style: *"fedback outputs" vs. none*

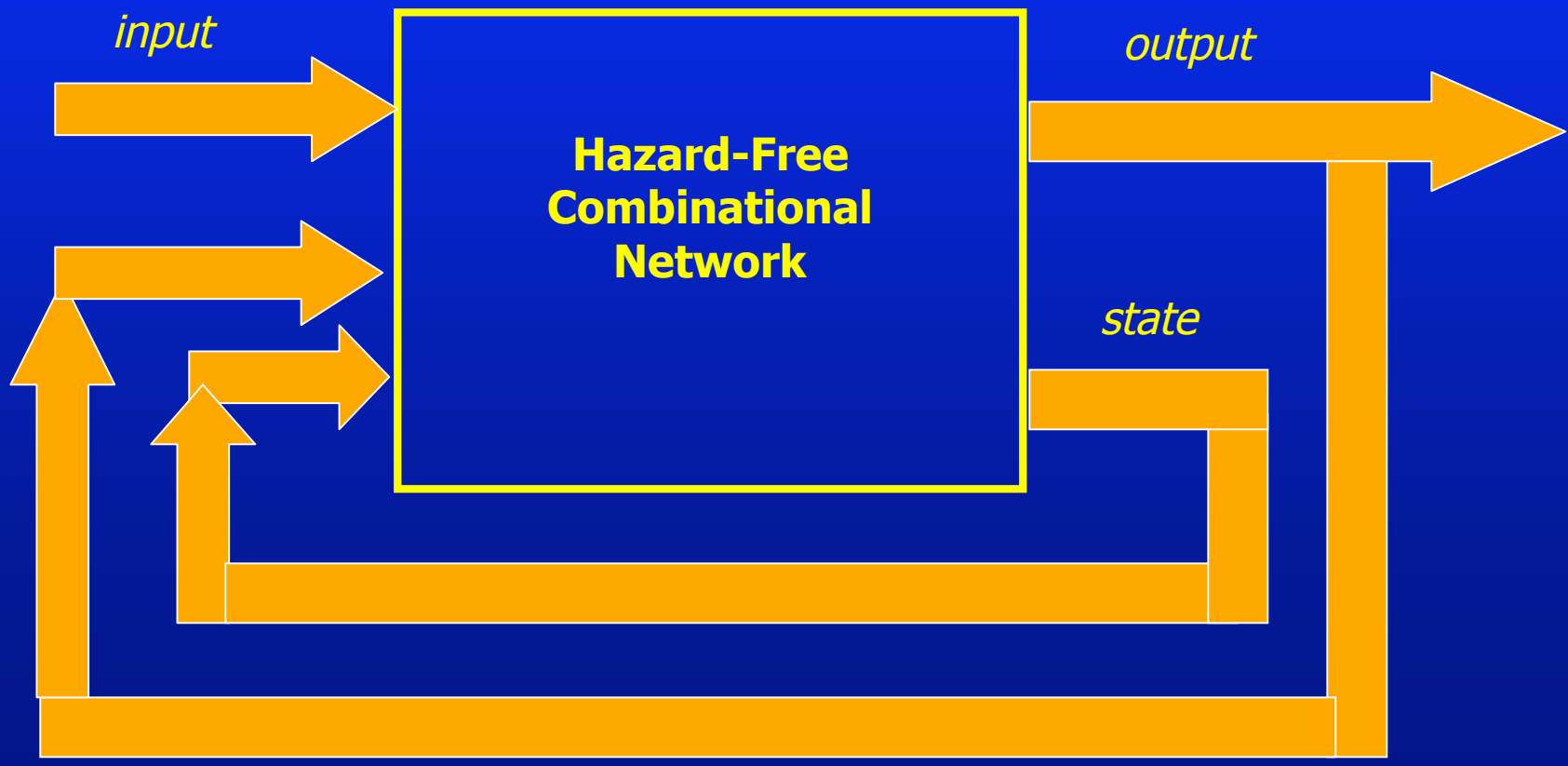
(a) No Fedback Outputs: benefit = sometimes smaller output loads (lower latency)



User-Selectable Modes

#1. Machine Style (cont.)

(b) Fedback Outputs: benefit = sometimes less area/fewer state bits



User-Selectable Modes

#2. Logic Implementation Styles: 3 Choices

(a) "Multi-Output":

- *share products* across all outputs + next-state
  - goal: area

(b) "Single-Output":

- *no shared products:* implement each function separately
  - goal: performance

(c) "Output-Disjoint":

- share products only: (i) among outputs, and (ii) among next-state
  - goal: balanced

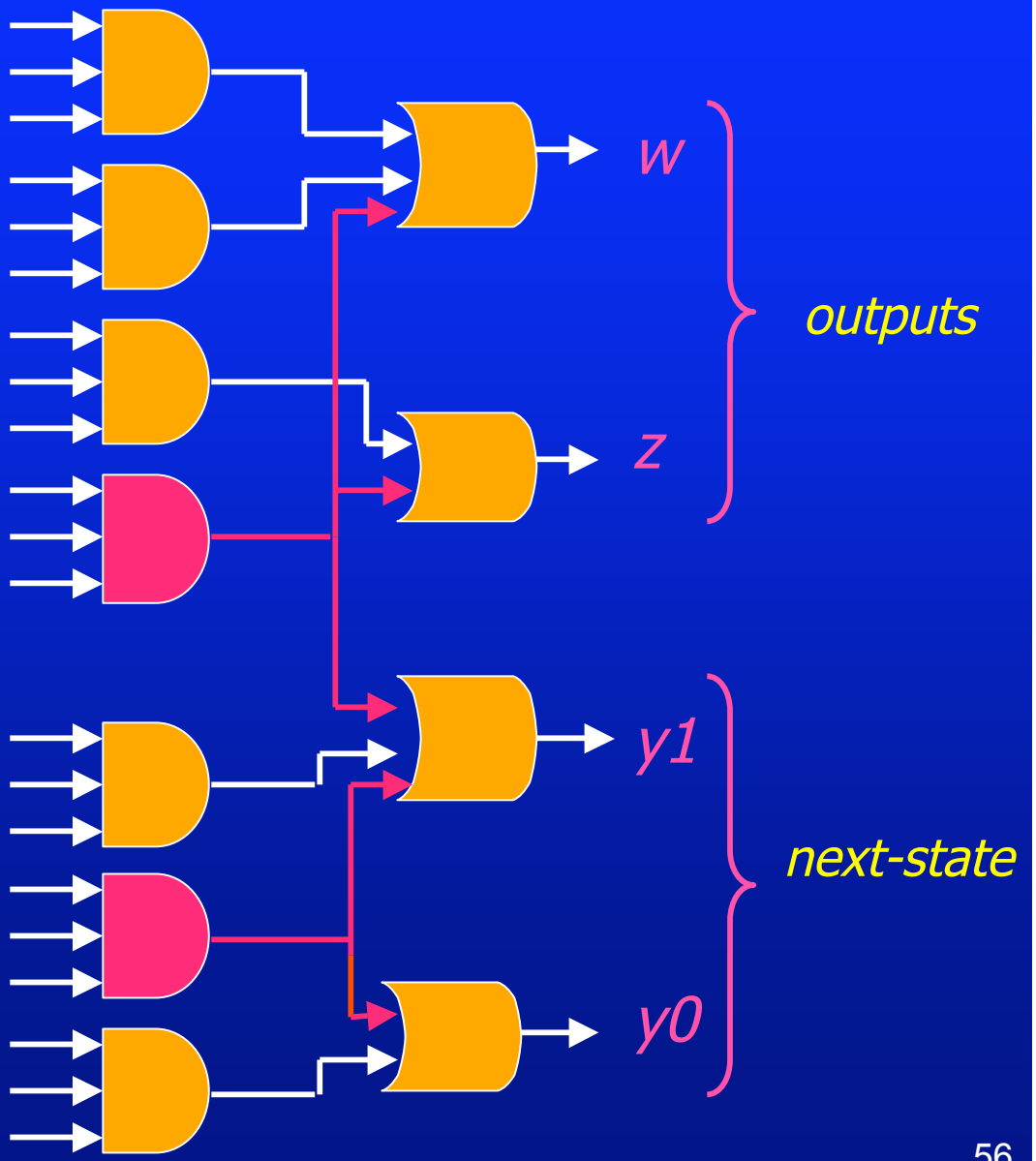
User-Selectable Modes

#2. Logic Styles:

(a) "Multi-Output":

share products between outputs + next-state

benefit:
- *area (sometimes)*



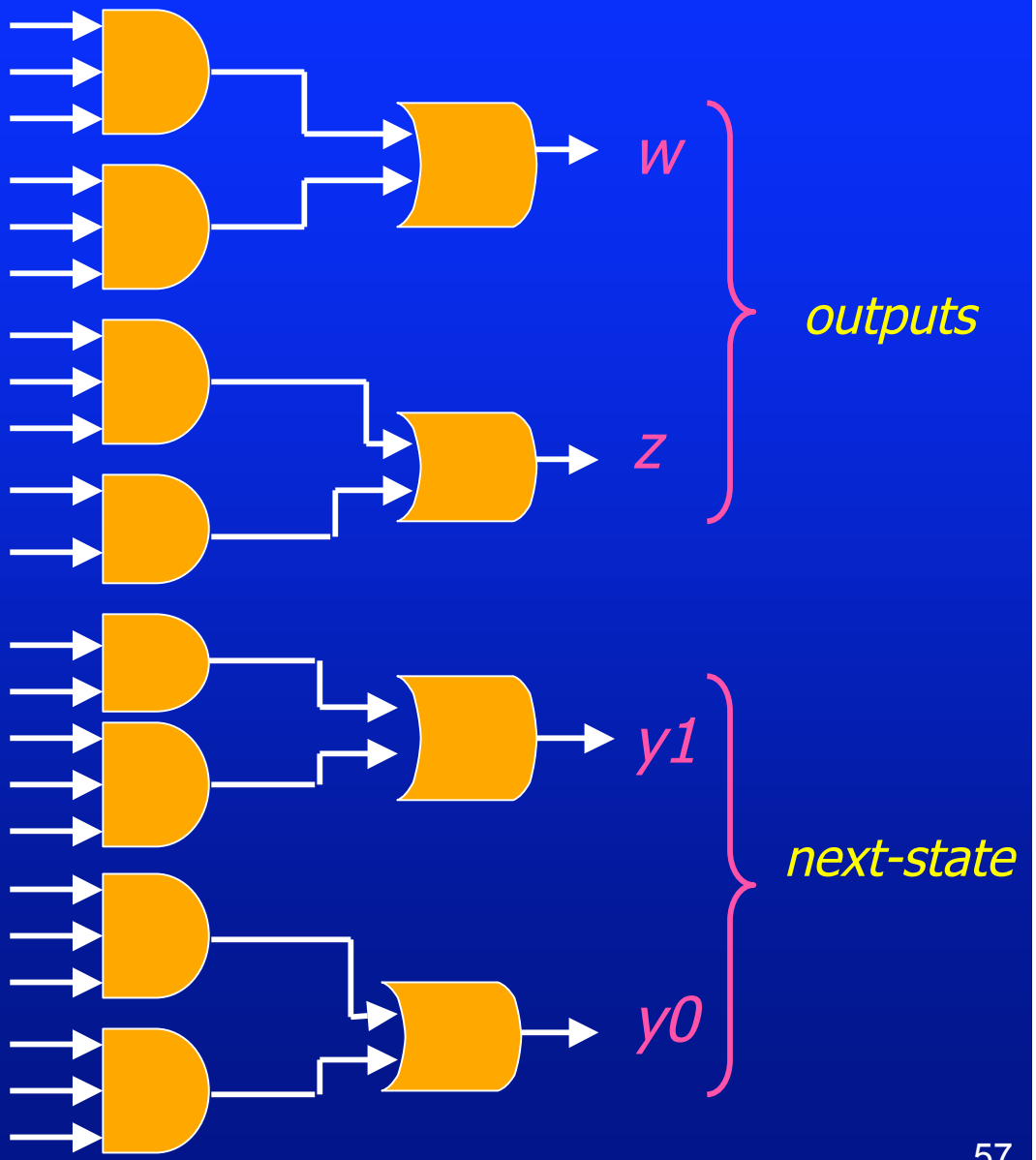
User-Selectable Modes

#2. Logic Styles:

(b) "Single-Output":

do not share products!

benefit:
- *speed (sometimes)*



User-Selectable Modes

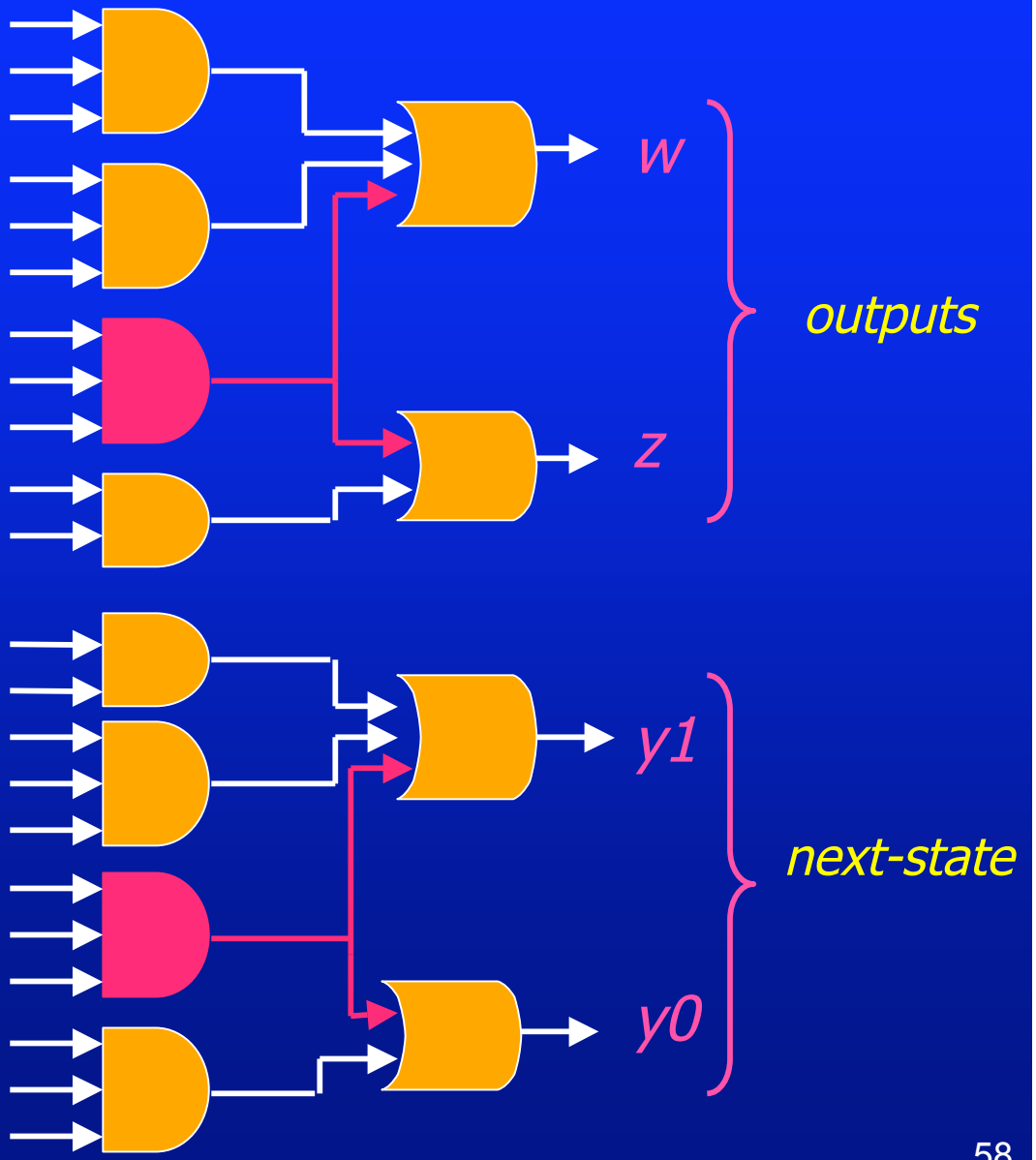
#2. Logic Styles:

(c) "Output-Disjoint":

share products only:
- *-among outputs*
- *-among next-state*

benefit:
- *-balanced approach (speed/area -- sometimes)*

algorithmic feature:
- *- the opt. state assignment method can ensure optimal sharing of products among primary outputs*



User-Selectable Modes

#3. Cost Function:

What to minimize…?:

(a) # products

(b) # literals

(c) # "primary I/O literals": *on critical input-to-output paths*

Motivation of (c): *lower latency*...

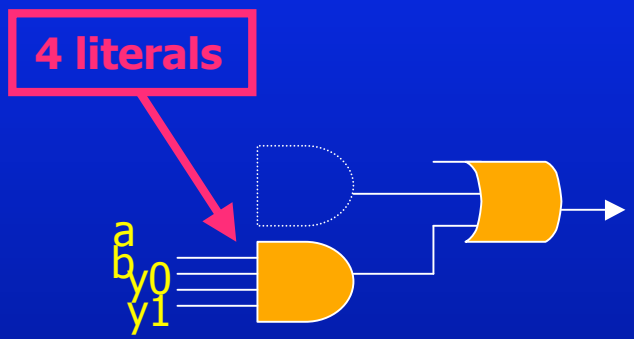
- primary inputs/primary outputs: (often) form the critical path
- state changes: (often) non-critical, occur in background mode

User-Selectable Modes

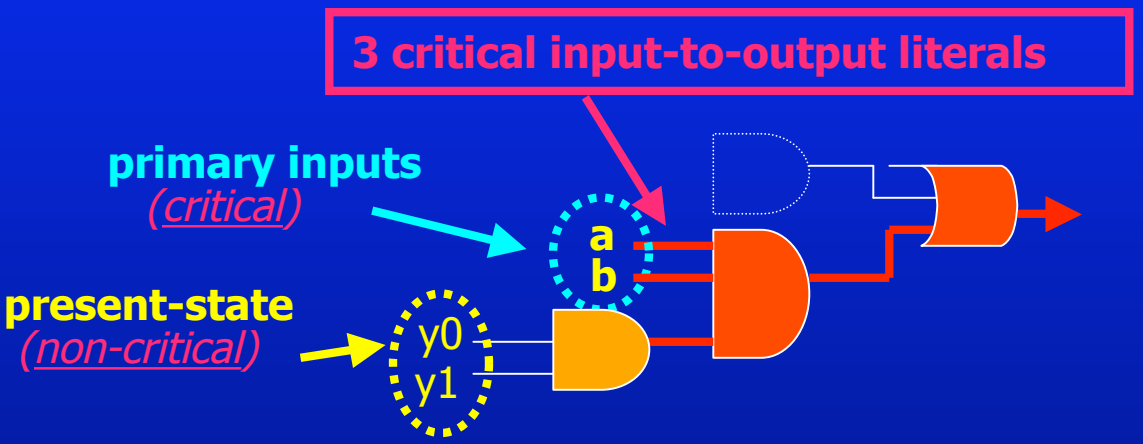
#3. Logic Cost Function (cont.): (c) "Primary I/O Literals"

EXAMPLE:

Initial 2-Level Circuit



After Multi-Level Decomposition:
"extract out" all present-state literals to improve machine latency



2-LEVEL CIRCUIT:
1st Option

DERIVING MULTI-LEVEL CIRCUIT:
apply logic decomposition to speed up primary input-to-output path *(3 "critical" literals)*

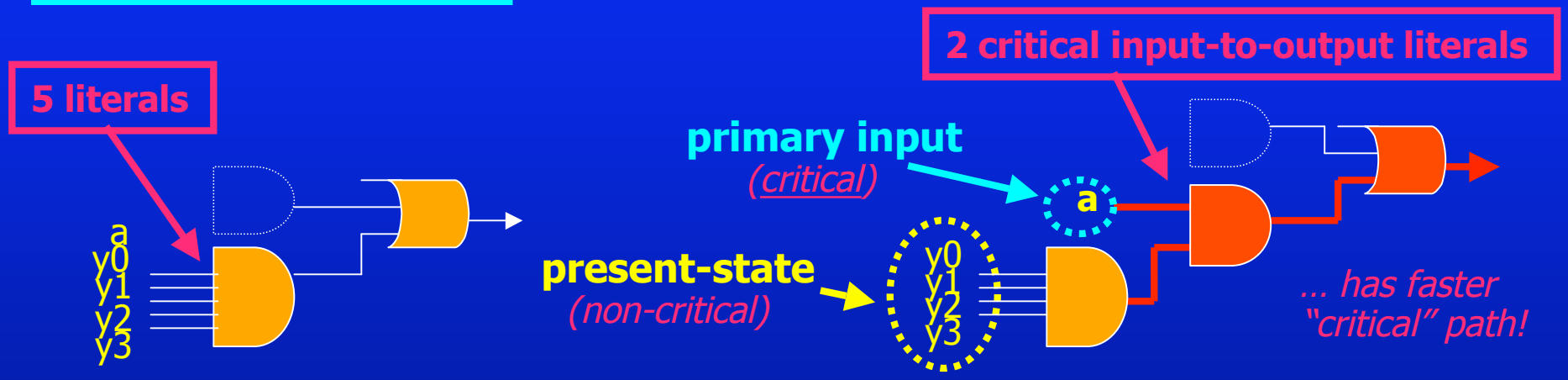
User-Selectable Modes

#3. Logic Cost Function (cont.): (c) "Primary I/O Literals"

EXAMPLE (cont.):

Initial 2-Level Circuit:

After Multi-Level Decomposition:
"extract out" all present-state literals to improve machine latency



2-LEVEL CIRCUIT:
2nd Option
...has more literals (5) than 1st option, yet yields better multi-level circuit!

DERIVING MULTI-LEVEL CIRCUIT:
apply logic decomposition to speed up primary input-to-output path *(only 2 "critical" literals!)*

User-Selectable Modes

#3. Logic Cost Function (cont.): (c) "Primary I/O Literals"

Conclusion: Pick 2-Level Circuit "Option #2"

> it has more literals,

> ...yet results in a faster multi-level circuit

(after multi-level logic decomposition)

Cost Function "Primary Input/Output Literals"...:

>produces 2-level circuit with fewest (primary) input literals for each primary output

>next: apply multi-level logic decomposition *(automatically with "MLO", or manual):*

- *factor out* "present-state literals" (non-critical)

>result: multi-level circuit with optimized (critical) primary input-to-output paths

User-Selectable Modes

#4. State Assignment Style: Several Options

(a) Critical Race-Free (basic):

- no optimization

(b) Optimal (CHASM): *exact solution*

- solve *all* optimality constraints

(c) Optimal (CHASM): *"fixed-length" encoding*

- *"heuristic mode"*
- *partially* solve optimality constraints (fewer state bits)

(d) Optimal (CHASM): *"output-only" mode*

- outputs: exact min-cost solution
- next-state: ignore/just insure they are critical race-free

MINIMALIST: Experimental Results
(from earlier release v1.2)

Design		3D			MINIMALIST					
					Performance (single-output)			Area (multi-output)		
	i/s/o	Prods	lits	olits	FBO	prods	olits	FBO	prods	lits
Pscsi-isend	4/9/3	31	105	44		31	38	Yes	18	67
Scsi-isend-bm	5/10/4	26	87	59	Yes	23	56	Yes	17	63
Scsi-tsend-bm	5/11/4	28	92	67		24	48		19	75
It-control	5/10/7	21	73	56	Yes	19	56	Yes	13	54
Sc-control	13/33/14	122	512	301	Yes	93	275	Yes	3	267
Sd-control	8/27/12	50	217	155	Yes	43	98	Yes	28	125
Stetson-p1	13/33/14	87	371	209	Yes	61	179	Yes	45	199
Stetson-p2	8/25/12	46	194	148	Yes	45	130	Yes	30	140
Oscsi	10/45/5	129	529	187	Yes	89	185	Yes	65	334

Performance: avg. reduction of 11% (up to 37%) in output literals.

Area: avg. reduction of 33% (up to 48%) in total literals.

Evaluation and Summary

Some Characteristics, and Potential Advantages, of Burst-Mode Circuits:

1. Simple Timing Requirements:

- Combinational Logic: highly robust
  - *"forward" logic path:* always hazard-free, regardless of gate/wire delays
    - + *no* timing assumptions (to process one "input burst" through comb. logic)
    - + *no* "isochronic forks" required
      - » hazard-free regardless of gate or wire delays
- Sequential Operation:
  - must satisfy simple *one-sided timing requirements*

Evaluation and Summary

2. Provides "Complete" Synthesis Path: always succeeds

- Given: any legal burst-mode specification (BM/XBM)
- Apply:
  - *(constrained)* state minimization
  - *(critical race-free)* state assignment
  - *(hazard-free)* 2-level logic minimization
- Result: always guaranteed *hazard-free "gate-level circuit":*
  2-level/multi-level implementation
  - *no backtracking/iteration ever required*

Evaluation and Summary

3. Hazard-Free Logic Decomposition:

Wide range of safe *"hazard-non-increasing" transformations*:

- associative law
  - not generally safe for 'speed-independent' (QDI) circuits!
- DeMorgan's law, and many other algebraic transformations
- technology mapping: simple to ensure correctness

Unlike speed-independent circuits, no need for:

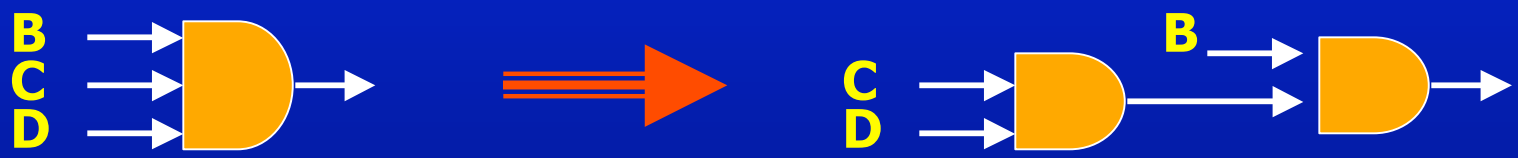
- "acknowledgment forks", state-holding elements within combinational logic, etc.

Evaluation and Summary

3. Hazard-Free Logic Decomposition (cont.):

Example: Associative Law

– in burst-mode circuits, never introduces hazards



(This transform may introduce hazards in "speed-independent" or "QDI" circuits.)

Evaluation and Summary

4. Optimal Synthesis Algorithms:

- Several "provably-optimal" algorithms (e.g. CHASM)
- Incorporates/exploits several highly-optimized synchronous CAD tools for sub-steps (Scherzo, espresso, dichot, etc.)

Evaluation and Summary

5. Generalized Fundamental-Mode Operation:

- BM Target: designing controllers with low latency (i.e. input-to-output response time)
- Tradeoff: must allow machine to "settle" (i.e. hold-time) between input events, before next input can be applied
- In practice, this timing assumption *often satisfied:*
  - fundamental mode timing "window": usually very small settling time
    - + typical BM controllers are small: only 2-3 gates "deep" in logic
    - + "settling time": usually much less than time for environment to send new inputs
  - industrial experiences: NASA Goddard (2006-7), HP Labs ("Stetson" 94), etc.
    - + … confirm practicality of handling these timing constraints for many applications
- If cannot be satisfied: can always add extra delay to primary outputs
  - will allow machine to settle before outputs are sent to environment

Conclusions

"Burst-Mode" Asynchronous Controllers:

- used effectively in several experimental industrial designs:
  - Recent (2006-2007): NASA Goddard Space Flight Center
  - Earlier: Intel, AMD, HP Labs

New "MINIMALIST" Asynchronous CAD Package:

- CHASM: optimal state assignment
- HFMIN, IMPYMIN, espresso-HF: hazard-free 2-level logic min
- gC-Min: gC-based tech-map
- Verifier: combinational + sequential (hazards, functionality)
- Verilog output, graphic interfaces

Overall Goals:

- providing many user options: *"design space exploration"*
- *globally-optimal algorithms*