Technology Mapping of Timed Asynchronous Circuits

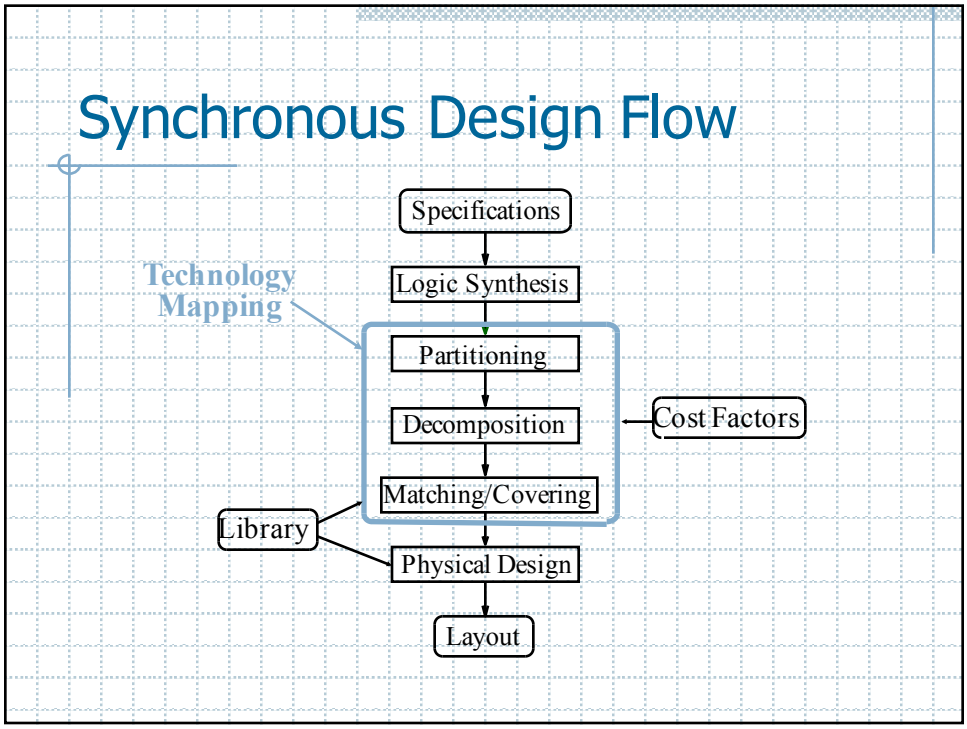
Curtis A. Nelson

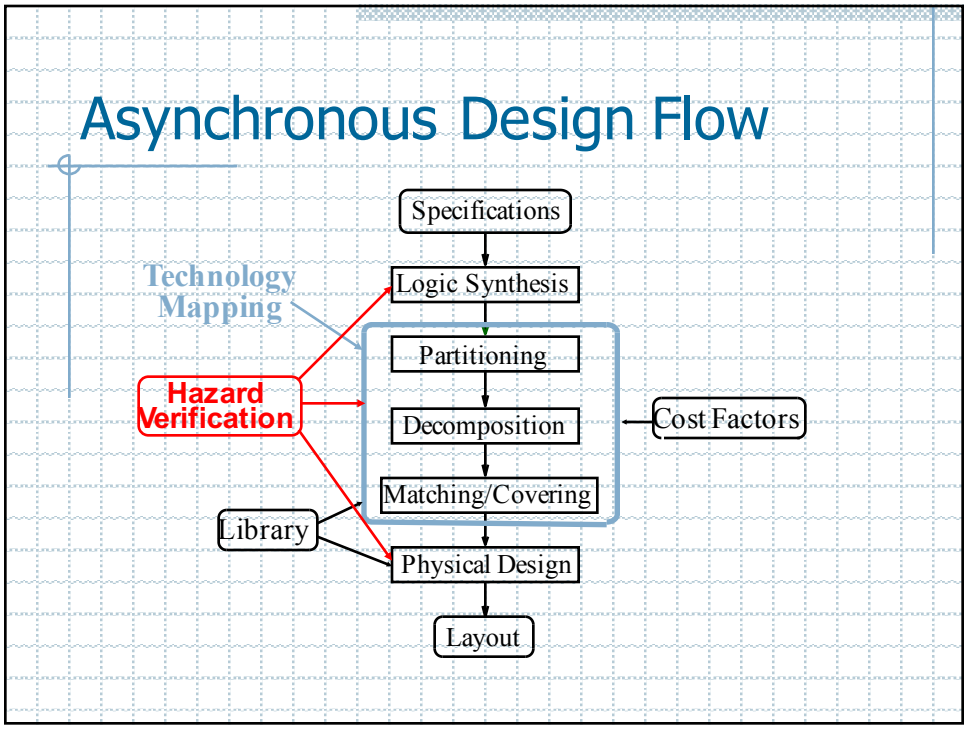
University of Utah

August 26, 2004

Technology Mapping

- Process of implementing a synthesized design.
- Utilizes technology-specific libraries.
- Combines the steps of:
  - Partitioning
  - Decomposition
  - Matching
  - Covering


Synchronous Design Flow
Specifications
Technology Mapping
Logic Synthesis
Partitioning
Decomposition
Cost Factors
Matching/Covering
Library
Physical Design
Layout


Asynchronous Design Flow
Specifications
Technology Mapping
Logic Synthesis
Partitioning
Hazard Verification
Decomposition
Cost Factors
Matching/Covering
Library
Physical Design
Layout

Timed Asynchronous Circuits

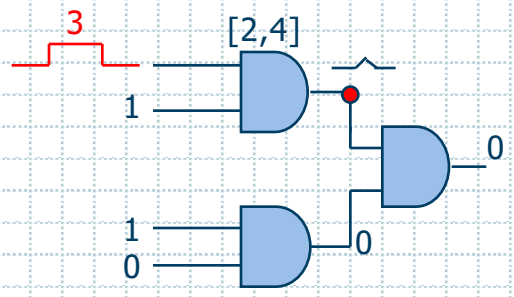
- *Timed circuits* are a class of asynchronous circuits that use explicit timing information.
- Can potentially reduce required circuitry as compared with *speed-independent* circuits.
- Used in Intel RAPPID design which was 3 times faster than synchronous design.
- As in all asynchronous circuits, timed circuit design is complicated by hazards.

Hazards

- Conditions that may manifest as glitches.
- Caused by structure or timing of the circuit.
- May result in incorrect behavior.
- Must be detected and eliminated.
- Two types of hazards:
  - *Acknowledgment.*
  - *Monotonicity.*

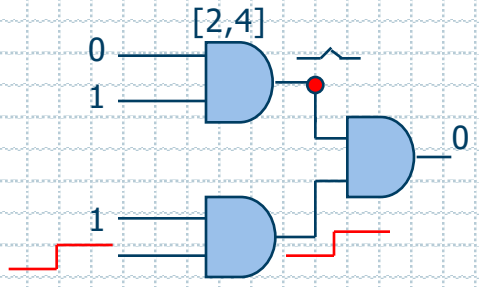
Acknowledgement Hazard

- Occurs when inputs to a gate change evaluation before the output stabilizes.



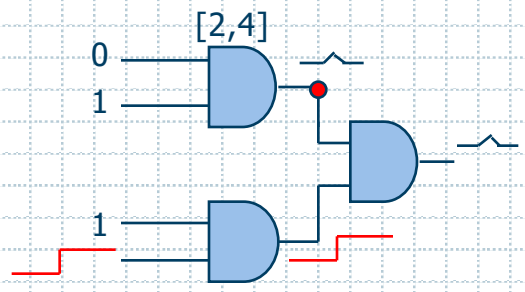
Acknowledgement Hazard

- Occurs when inputs to a gate change evaluation before the output stabilizes.



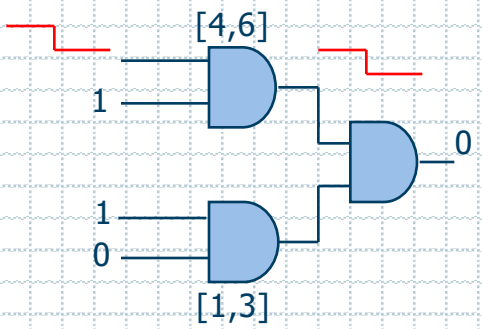
Acknowledgement Hazard

- Occurs when inputs to a gate change evaluation before the output stabilizes.



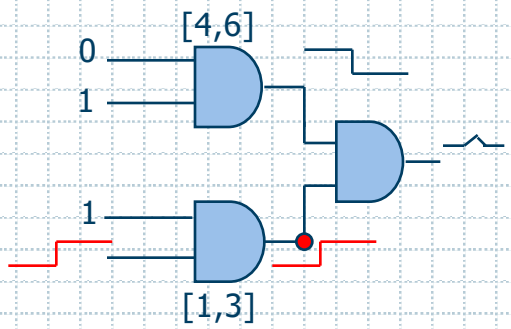
Monotonicity Hazard

- Occurs when an internal or output node:
  - Becomes excited to change when it should stay stable, or
  - Makes a transition non-monotonically.

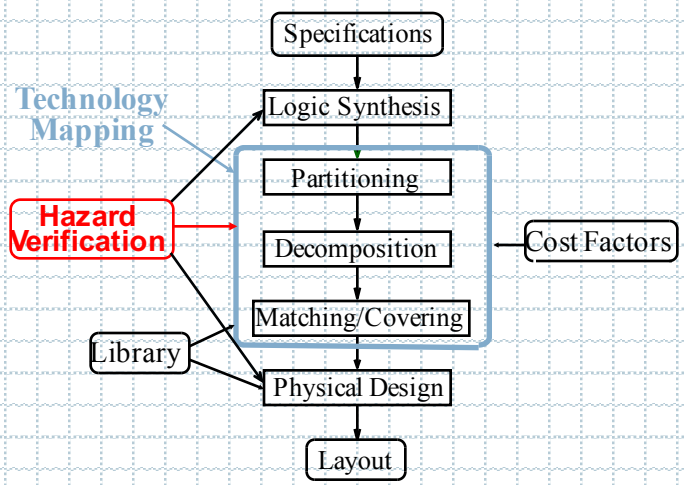


Monotonicity Hazard

- Occurs when an internal or output node:
  - Becomes excited to change when it should stay stable, or
  - Makes a transition non-monotonically.



Asynchronous Design Flow



Hazard Verification

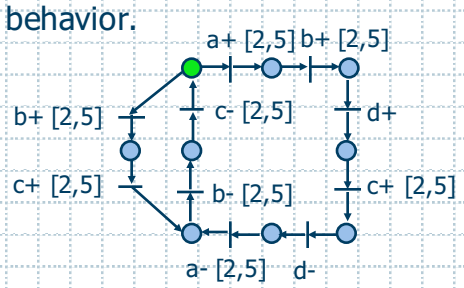
- Must check all reachable states for hazards.
- Size of state space is $O(2^{|I|} \times 2^{|O|} \times 2^{|N|})$.
- Beerel/Burch/Meng proposed using a *cube* to approximate internal signal behavior for speed-independent circuits.
- Reduces size of state space to $O(2^{|I|} \times 2^{|O|})$.
- We extend this application to timed circuits.

Hazard Verification Algorithm

- Input: *Time Petri Net* and *Gate Netlist.*
  - Check *complex gate equivalence*.
  - Determine stability of internal signals.
  - Check for acknowledgement hazards.
  - Check for monotonicity hazards.

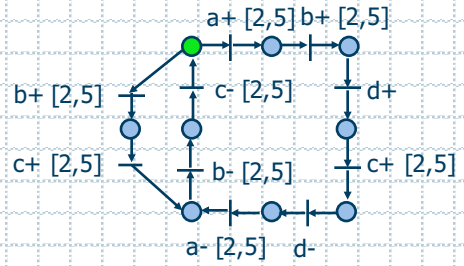
Time Petri Net

- Specification method for timed systems.
- Used to specify:
  - Environmental behavior.
  - Expected output behavior.



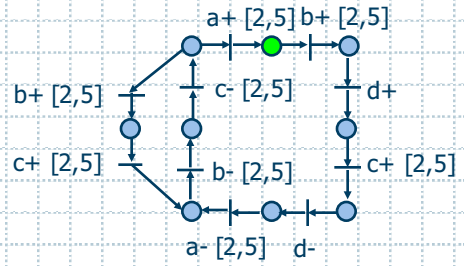
Time Petri Net

- State is a *marking* of places and *ages* of transitions.
- Transition *enabled* when all input places are marked.
- Transition *fires* after it has been enabled between [min,max] time units.

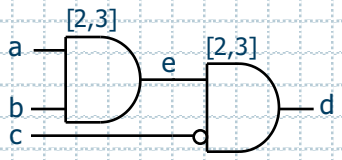


Time Petri Net

- After transition fires, marking removed from input places and added to output places.

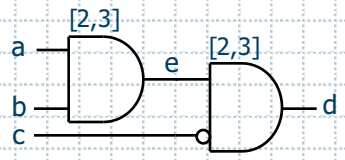


Gate Netlist

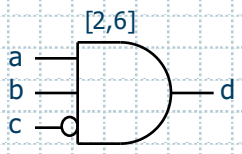


Gate-Level Netlist

Gate Netlist

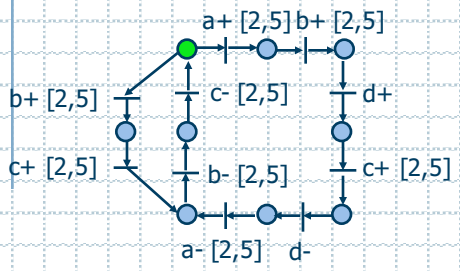


Gate-Level Netlist

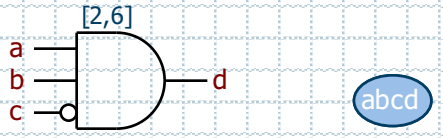


Complex Gate Equivalent (CGE)

Checking Equivalence

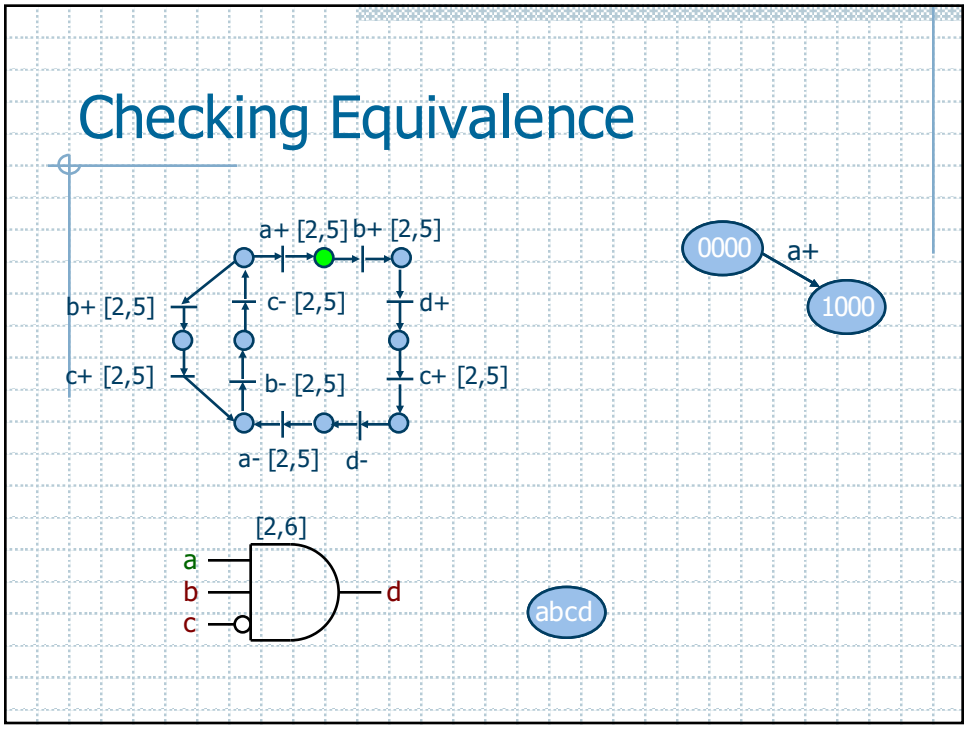


0000

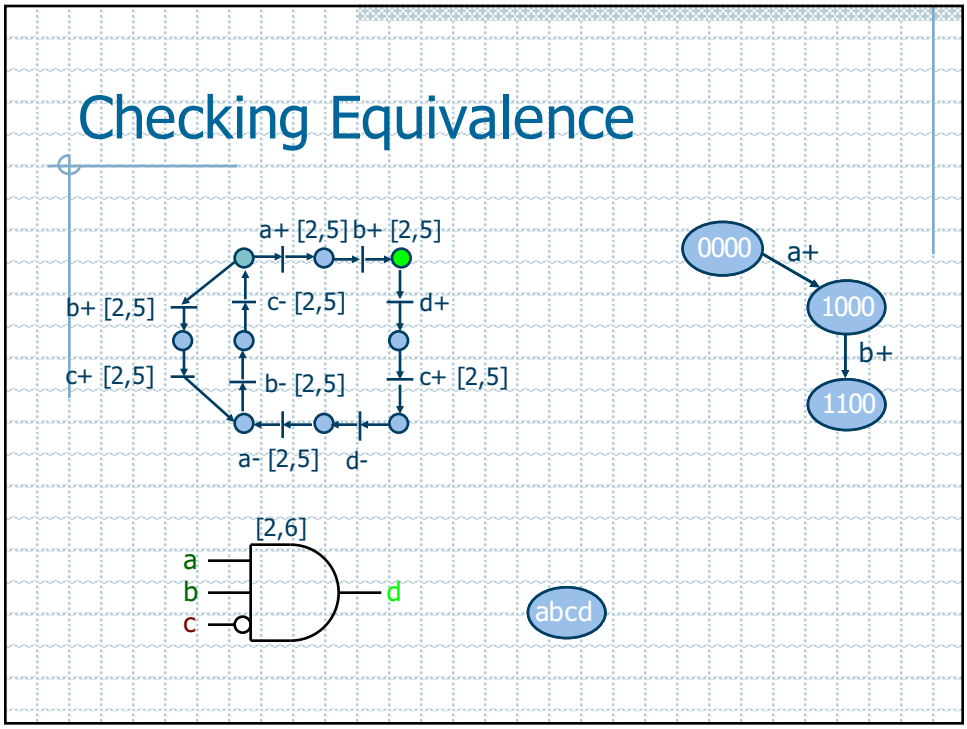


abcd

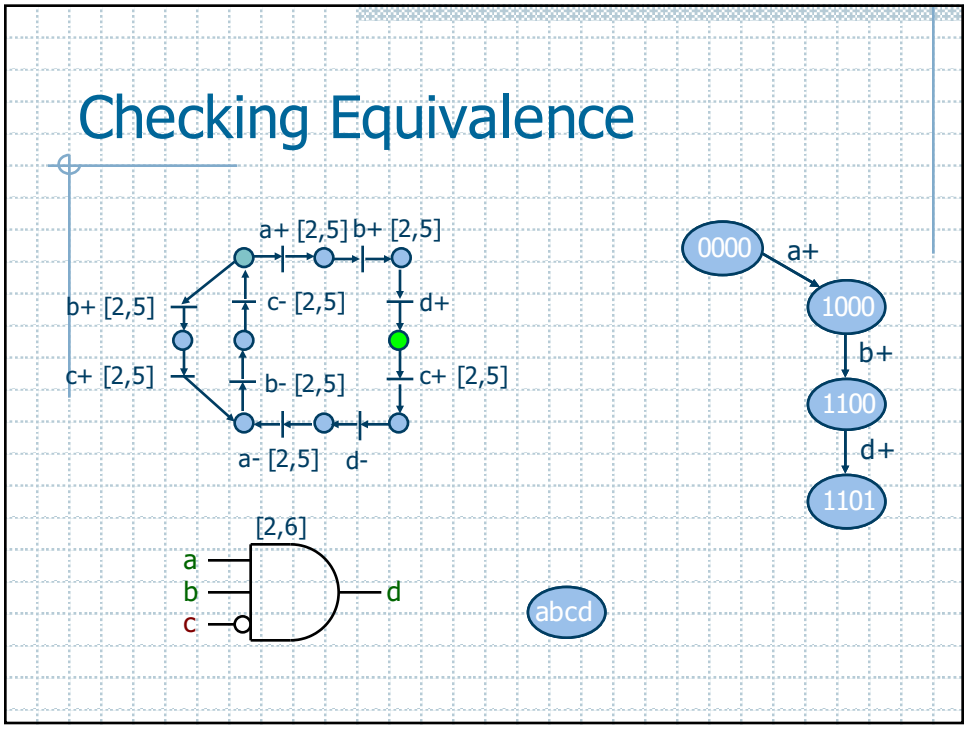
Checking Equivalence



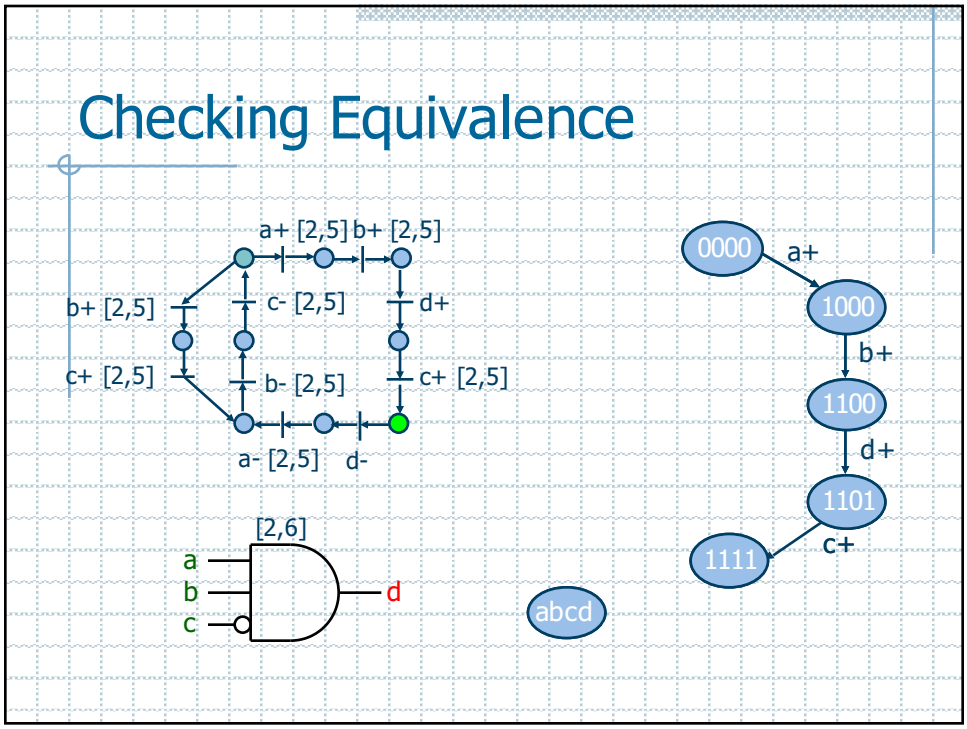
Checking Equivalence



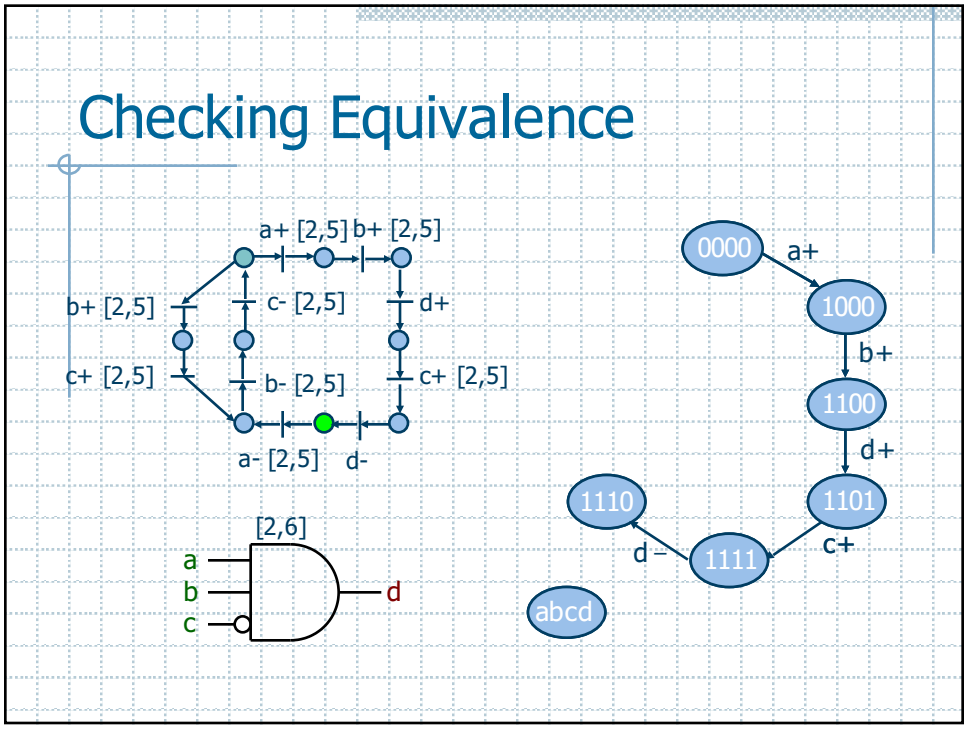
Checking Equivalence



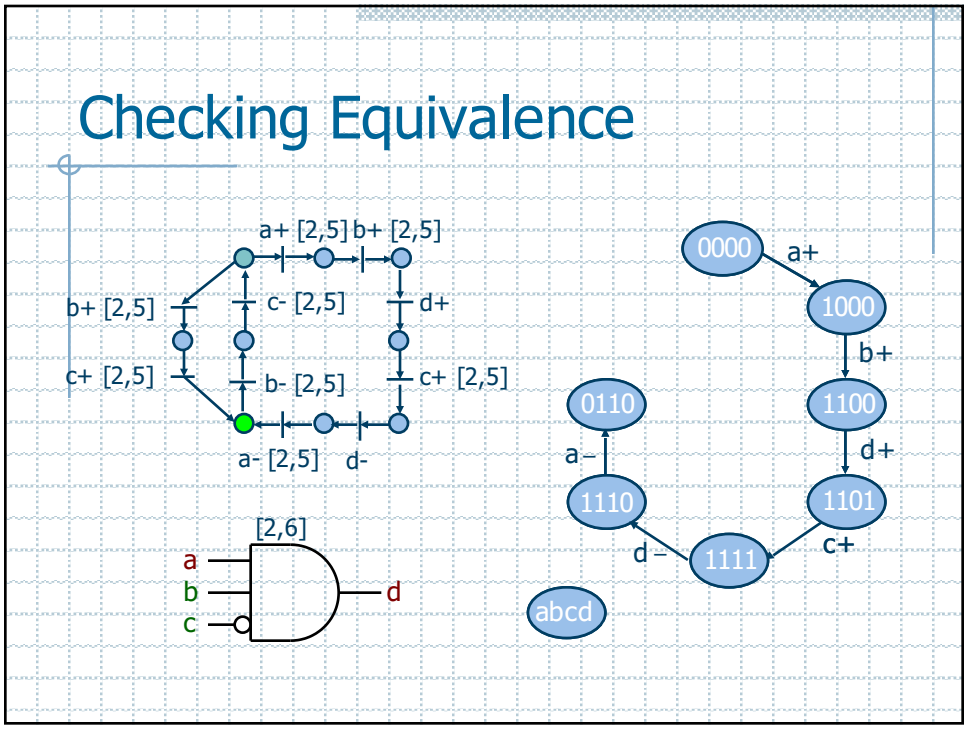
Checking Equivalence



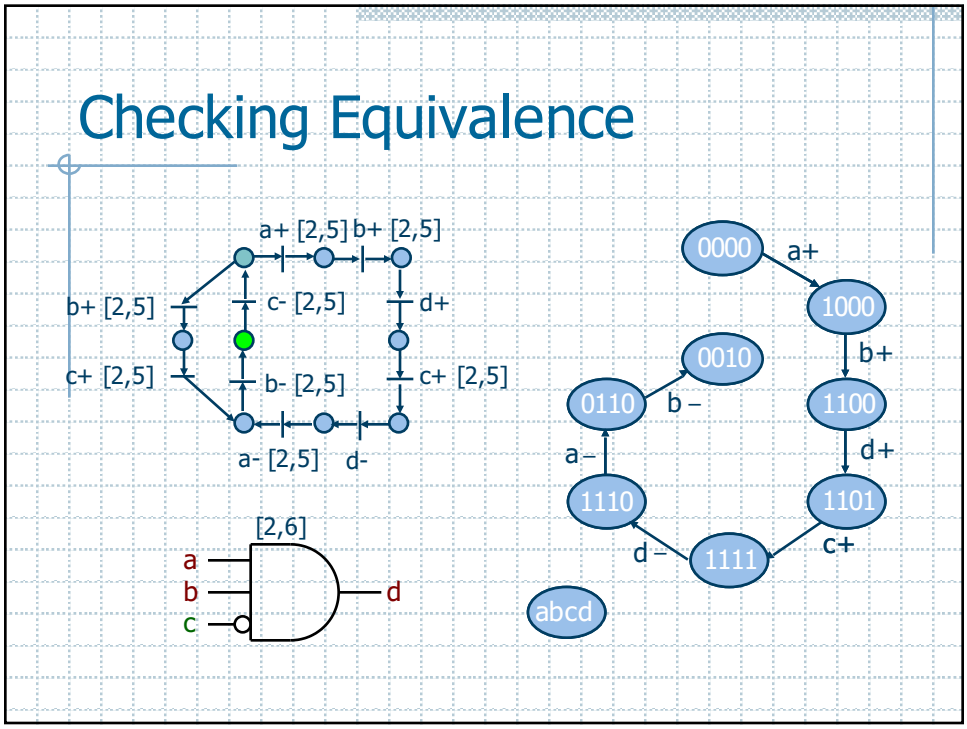
Checking Equivalence



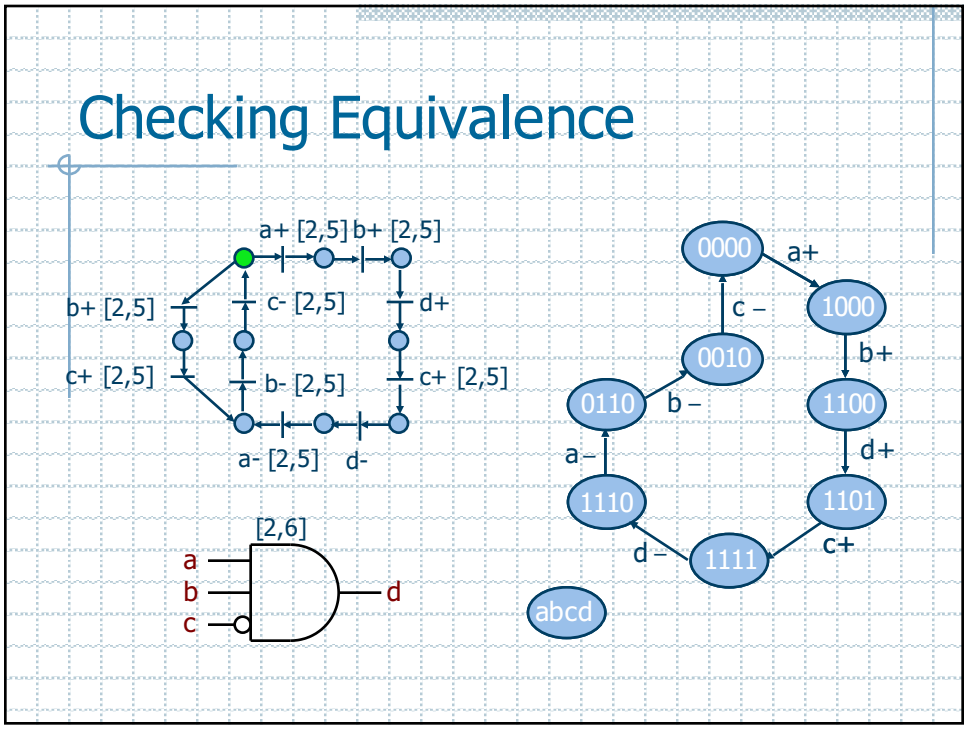
Checking Equivalence



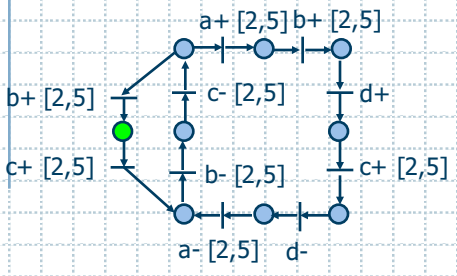
Checking Equivalence

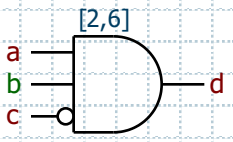


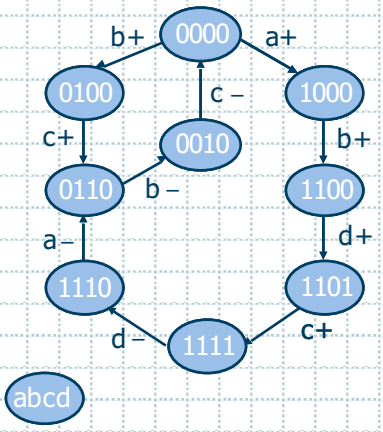
Checking Equivalence



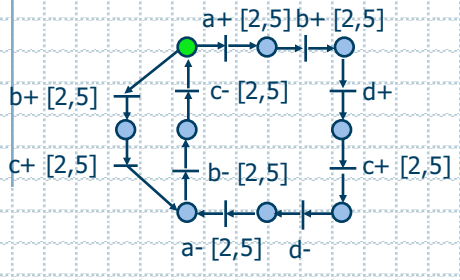
Checking Equivalence

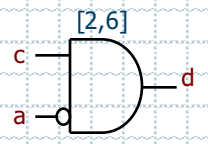




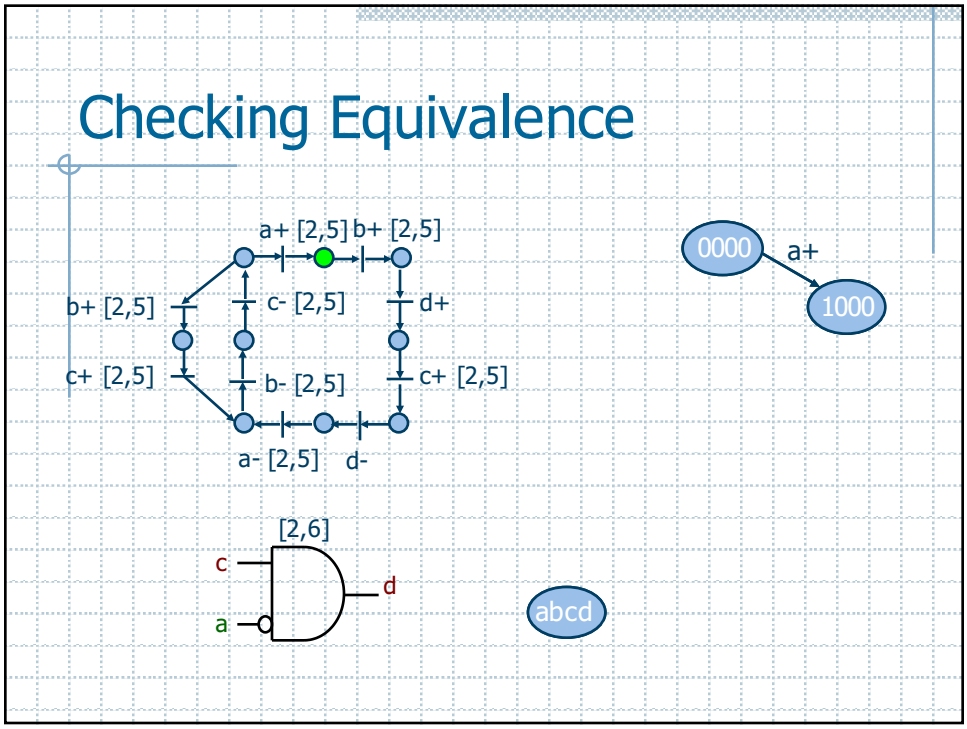


Checking Equivalence

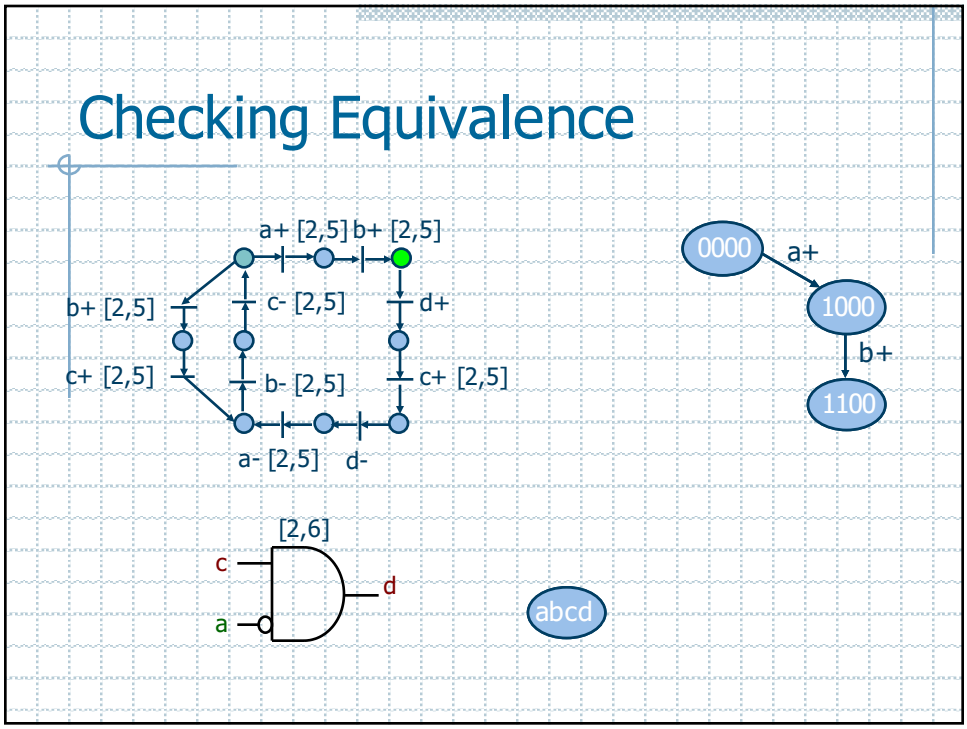




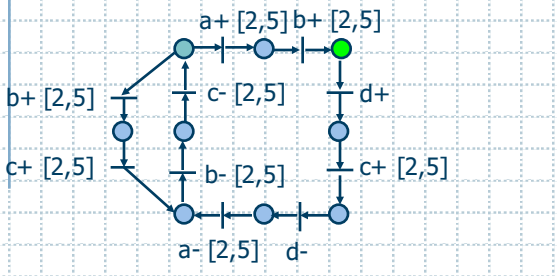
Checking Equivalence

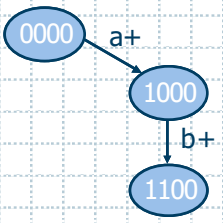


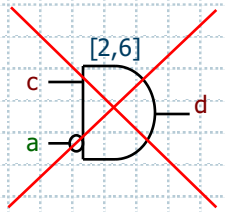
Checking Equivalence



Checking Equivalence

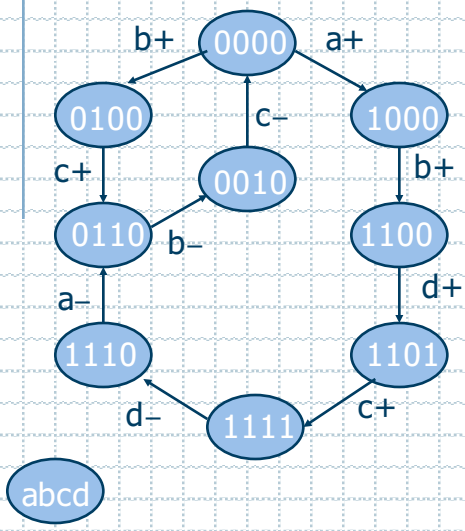


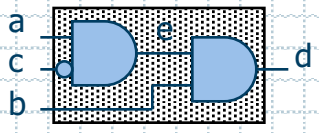




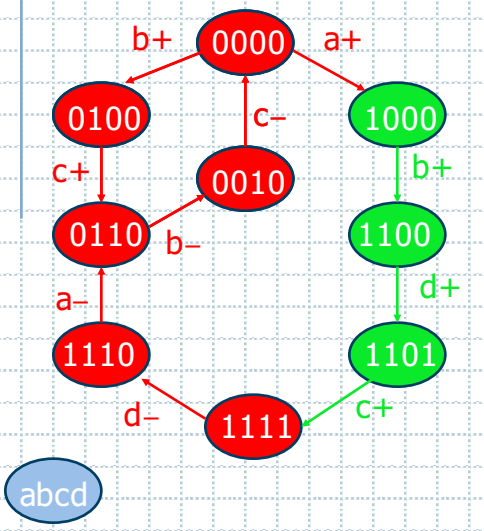
abcd

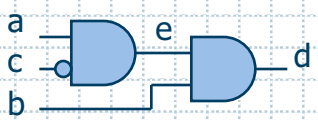
Finding Stable States





Finding Stable States

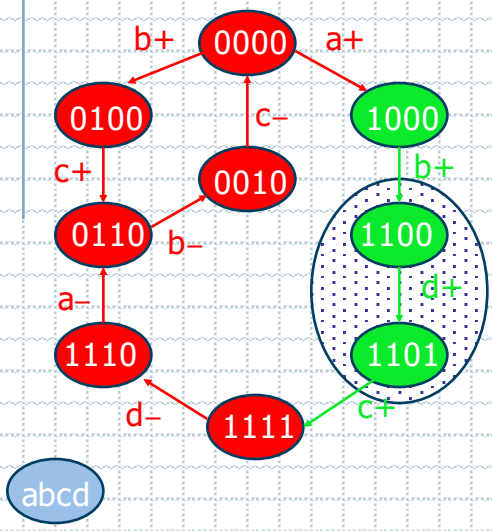


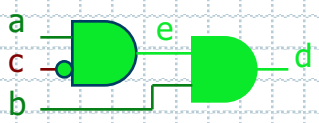


Step 1: Boolean evaluation of node e.

Rising
High
Falling
Low

Finding Stable States

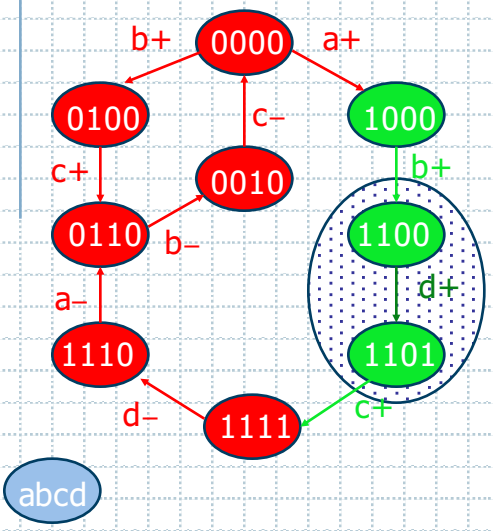


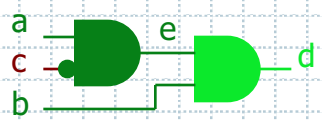


Step 2: Untimed stabilization of node e for d+

Rising
High
Falling
Low

Finding Stable States

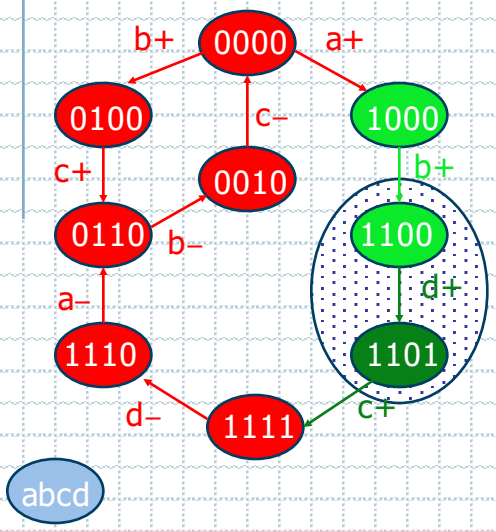


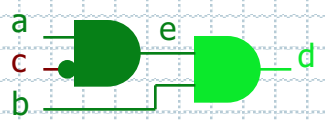


Step 2: Untimed stabilization of node e for d+

Rising
High
Falling
Low

Finding Stable States

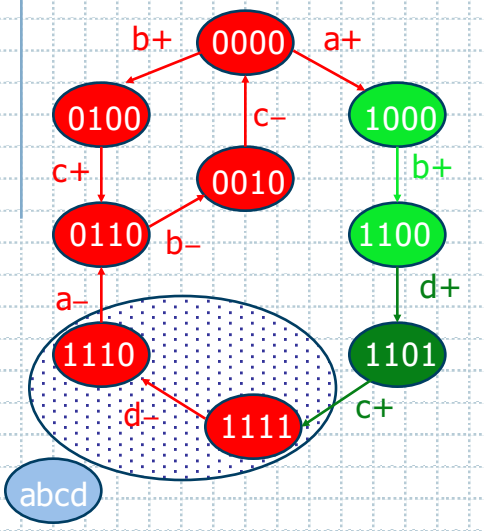


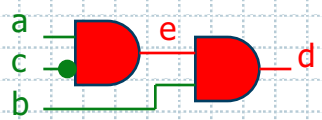


Step 2: State 1101 stable high.

Rising
High
Falling
Low

Finding Stable States

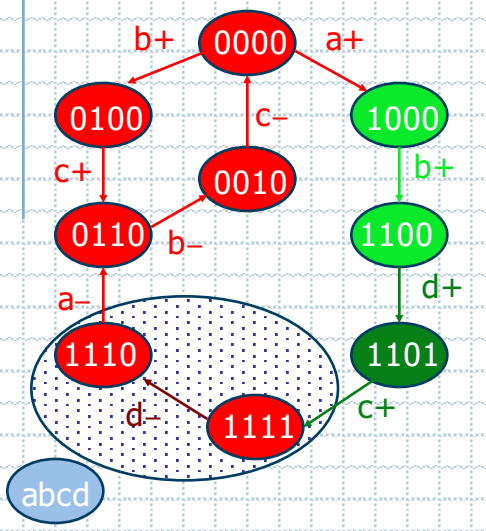


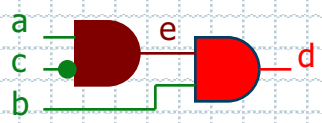


Step 3: Untimed stabilization of node e for d-

Rising
High
Falling
Low

Finding Stable States

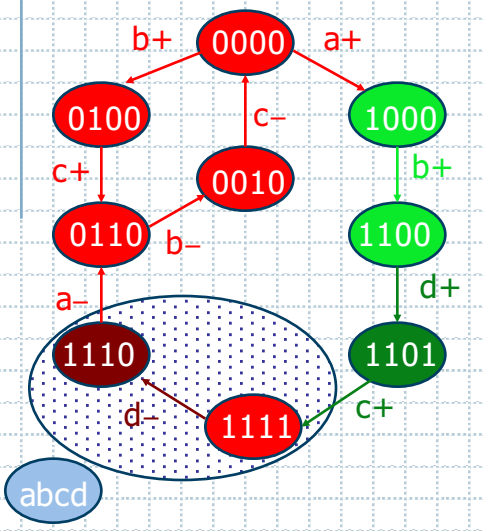


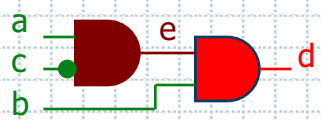


Step 3: Untimed stabilization of node e for d-

Rising
High
Falling
Low

Finding Stable States

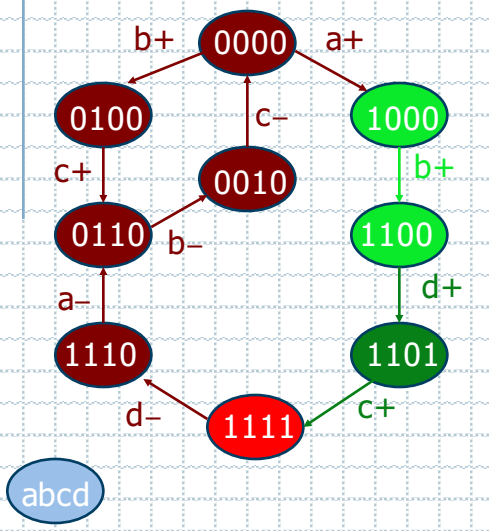


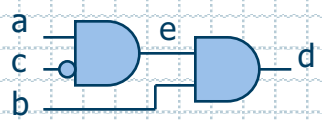


Step 4: Propagate stable states.

Rising
High
Falling
Low

Finding Stable States

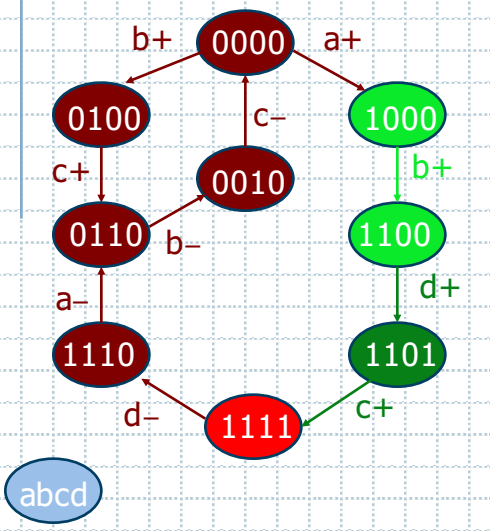


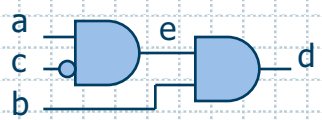


Step 4: Propagate stable states.

Rising
High
Falling
Low

Checking for Hazards

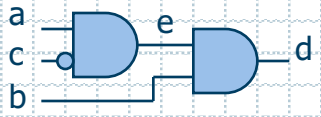




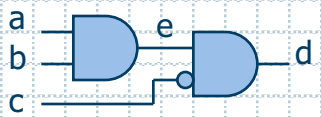
Node e is hazard-free!

Alternative Gate-Level Netlist

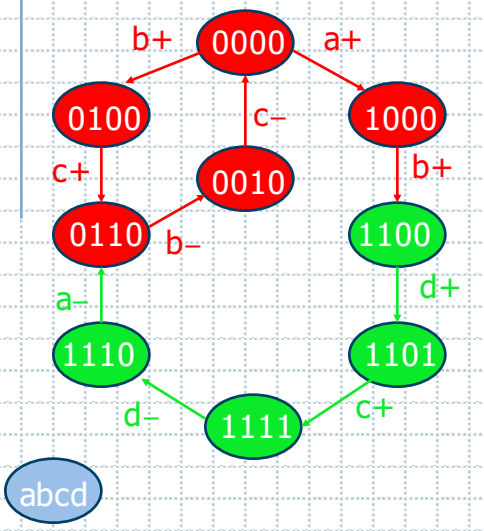
Try a different decomposition – Replace:

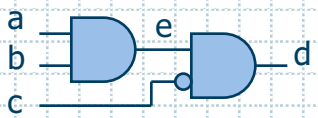


With:



Finding Stable States

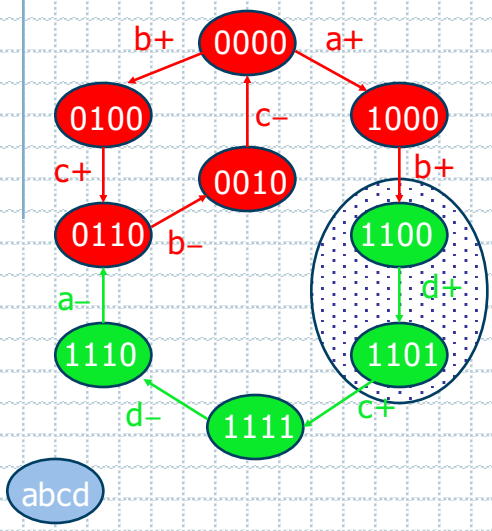


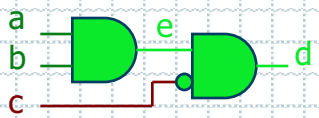


Step 1: Boolean evaluation of node e.

Rising
High
Falling
Low

Finding Stable States

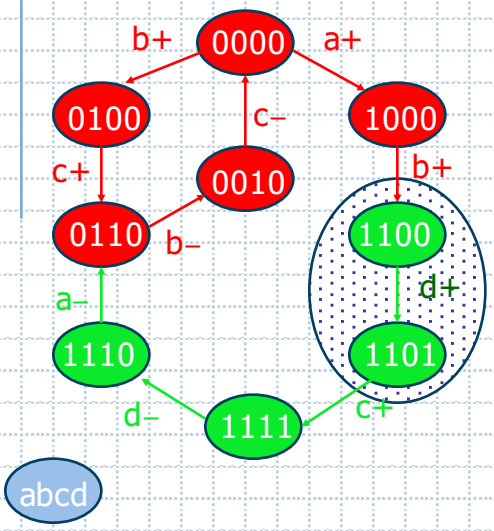


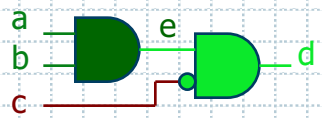


Step 2: Untimed stabilization of node e for d+

Rising
High
Falling
Low

Finding Stable States

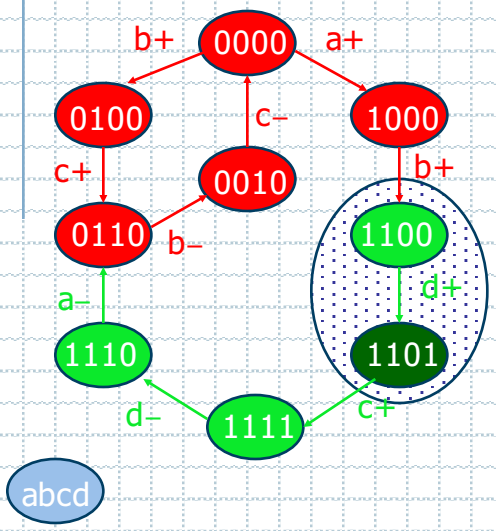


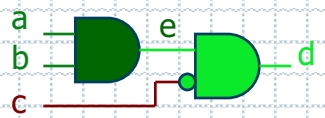


Step 2: Untimed stabilization of node e for d+

Rising
High
Falling
Low

Finding Stable States

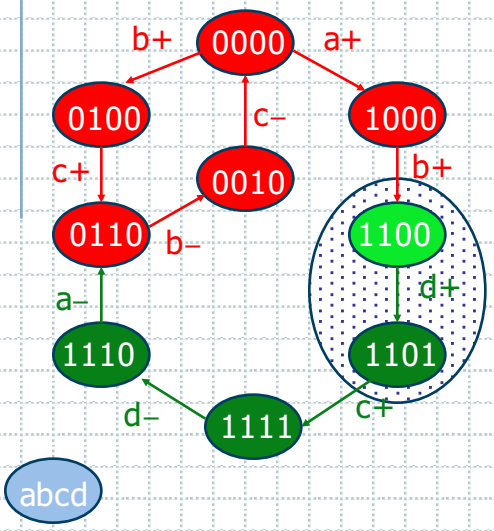


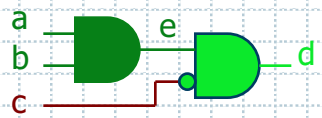


Step 3: Propagate stable states.

Rising
High
Falling
Low

Finding Stable States

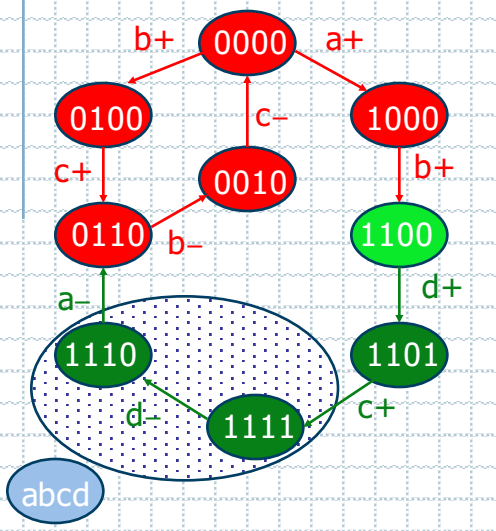


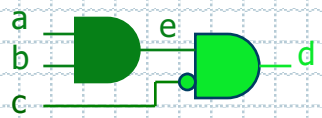


Step 3: Propagate stable states.

Rising
High
Falling
Low

Finding Stable States

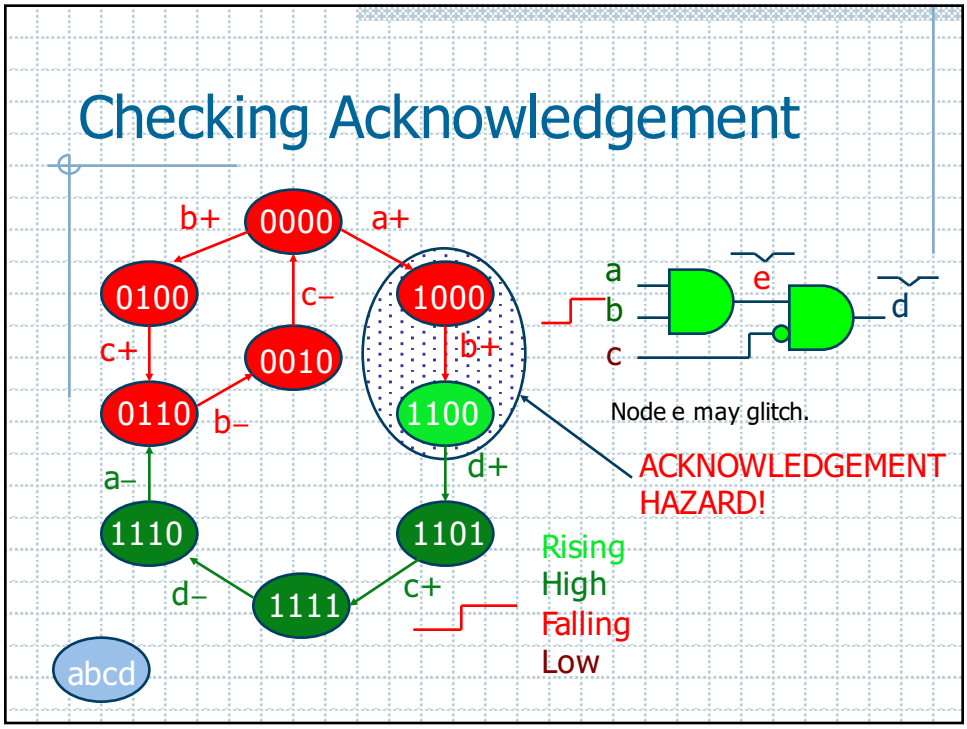




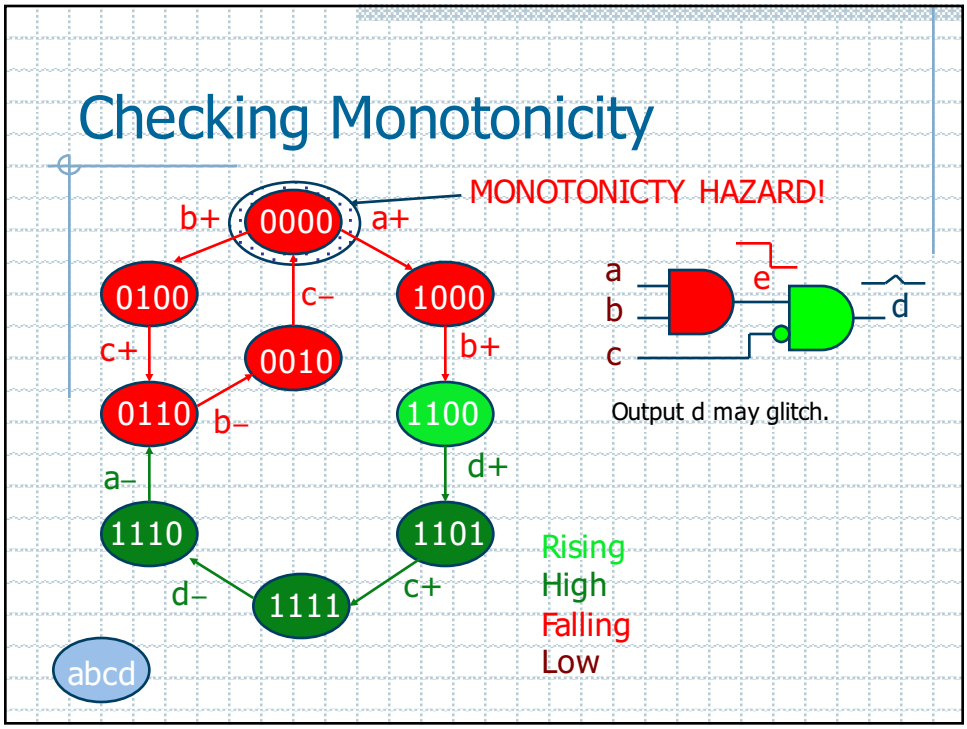
Step 4: Untimed stabilization of node e for d-

Rising
High
Falling
Low

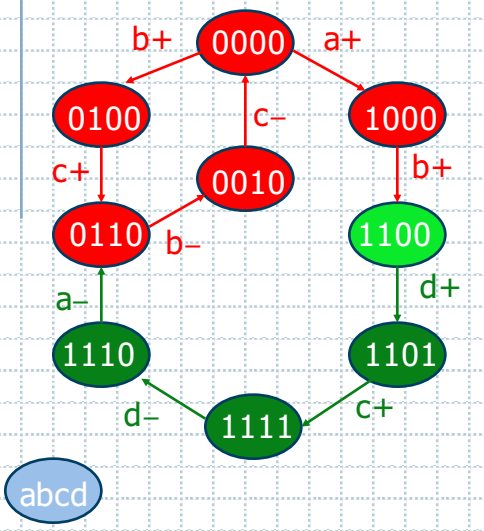
Checking Acknowledgement

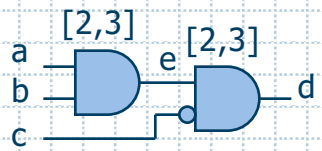


Checking Monotonicity



Timed Stabilization

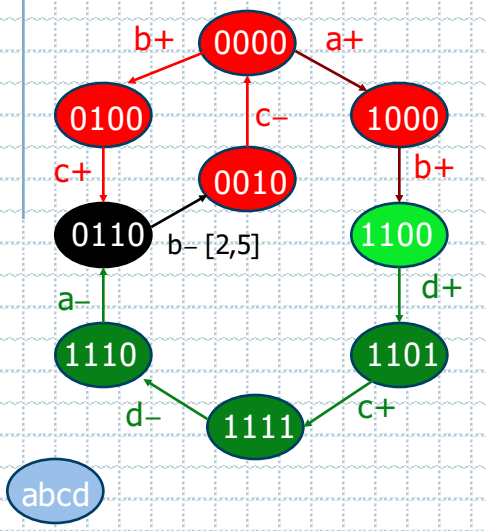


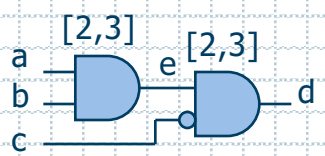


Consider Netlist timing.

Rising
High
Falling
Low

Timed Stabilization

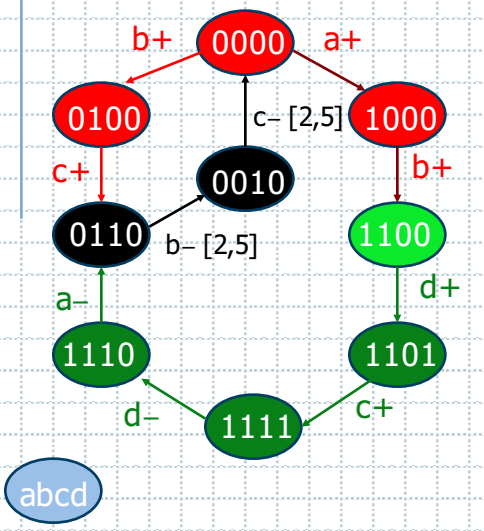


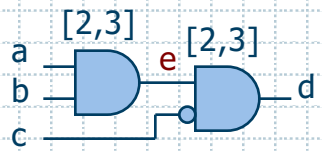


Minimum elapsed time: 2

Rising
High
Falling
Low

Timed Stabilization

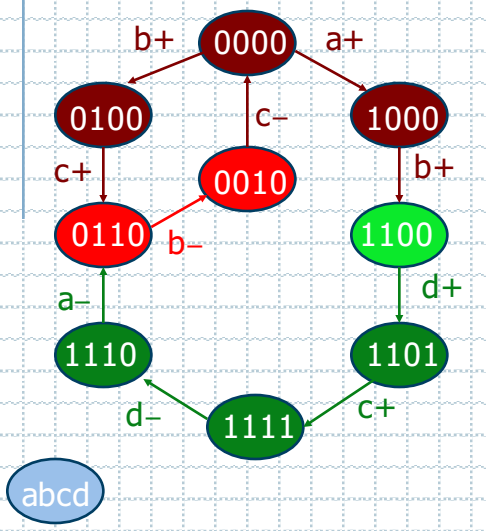


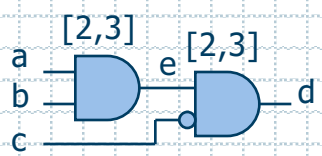


Minimum elapsed time: 4
Node e has stabilized low

Rising
High
Falling
Low

Timed Stabilization

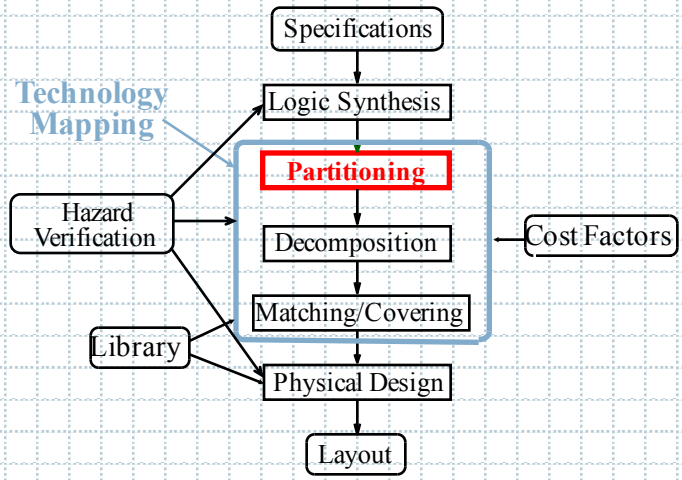




Propagate stable states.
Node e is now hazard-free!

Rising
High
Falling
Low

Asynchronous Design Flow



Partitioning

- Each output is a tree-based structure which is decomposed, matched, and covered individually.
- Makes matching and covering tractable.
- May reduce quality of final netlist because gate-sharing is eliminated between outputs.
- For our work, synthesis provides a set of equations already partitioned by primary output.

Decomposition: Background

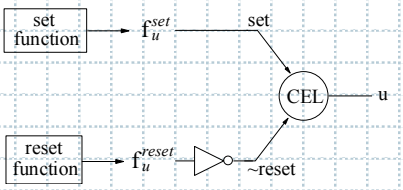
- Decomposition performed to guarantee a solution.
- Changing the circuit structure may create hazards.
- Synchronous approach:
  - Decompose synthesized circuit to *base functions*, typically inverters, 2-input NANDs, and memory elements.
- Asynchronous approaches are to:
  - Decompose directly to library gates without creating hazards (Burns, Beerel, Siegel, Kondratyev, Myers, etc).
  - Decompose without regard to hazard creation and remove hazards by inserting delay elements (Lavagno).

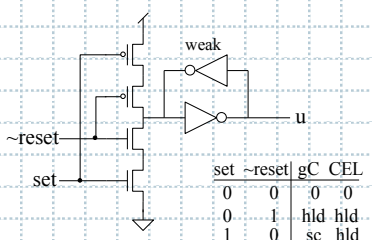
Decomposition: Our Approach

- Follow synchronous approach, decompose into base functions (inverters, 2-input NANDs, and C-element's).
  - Decompose without regard to hazard creation.
  - Perform gate-level hazard verification.
  - Do *hazard-guided* matching/covering later.
- Insert inverter pairs to increase matching options.
- Use unbalanced structure to allow for:
  - Consistent architecture.
  - Input pin re-ordering.

State-Holding Devices

- C-elements (CEL's).

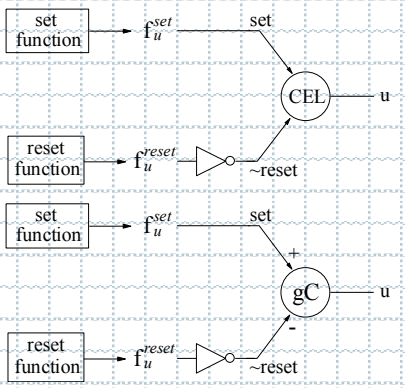


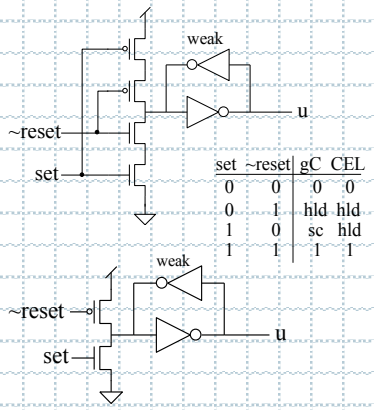


set	~reset	gC	CEL
0	0	0	0
0	1	hld	hld
1	0	sc	hld
1	1	1	1

State-Holding Devices

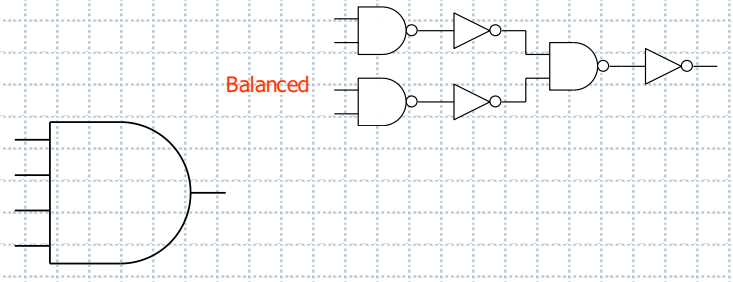
- C-elements (CEL's).
- Generalized C-elements (gC's).





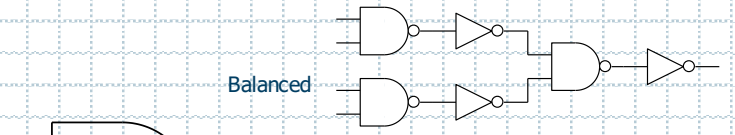
set	~reset	gC	CEL
0	0	0	0
0	1	hld	hld
1	0	sc	hld
1	1	1	1

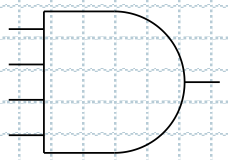
Decomposition Architectures

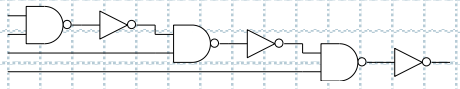


CGE Netlist

Decomposition Architectures

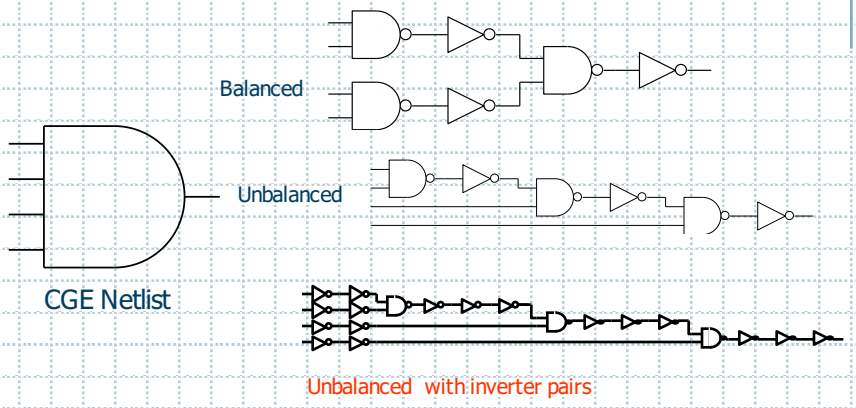






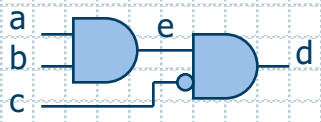
CGE Netlist

Decomposition Architectures



Input Pin Re-ordering

- Identify trigger, context signals.
- Place wisely in decomposition.

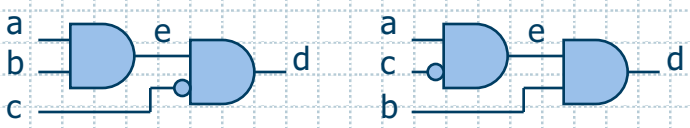


Untimed Hazardous.

Timed Hazard-free.

Input Pin Re-ordering

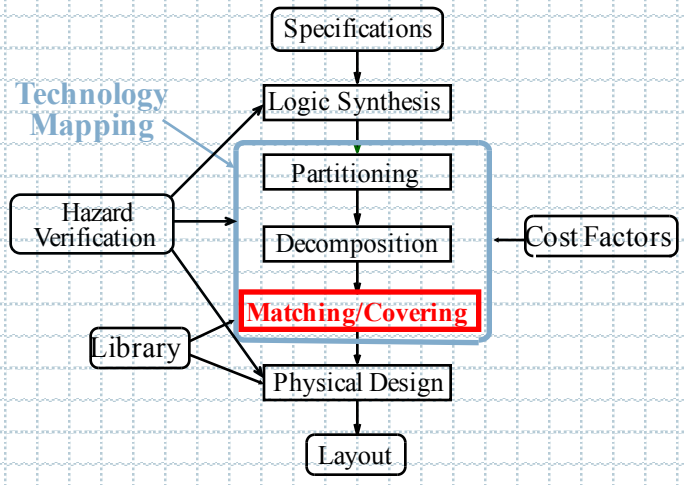
- Identify trigger, context signals.
- Place wisely in decomposition.



Untimed Hazardous.
Timed Hazard-free.

Untimed Hazard-free.
Timed Hazard-free.

Asynchronous Design Flow

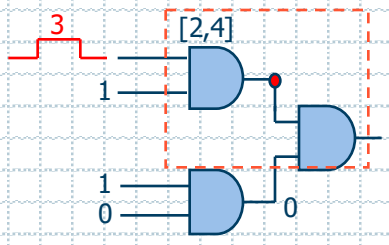


Matching and Covering

- Structurally matches library elements to circuit decomposition.
- Typically guided by area, delay, or power.
- Ours must be guided by hazard-freedom.
- Proposed hazard-aware matching algorithm:
  - Encapsulates acknowledgment hazards.
  - Finds *forcing* side-inputs for monotonicity hazards.

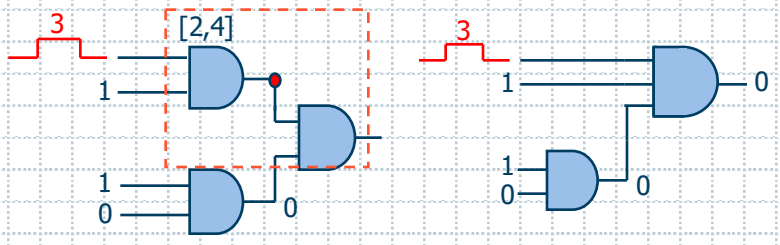
Acknowledgement Hazard

- Occurs when inputs to a gate change evaluation before the output stabilizes.
- Node *must be* encapsulated.



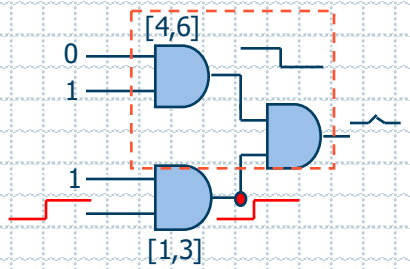
Acknowledgement Hazard

- Occurs when inputs to a gate change evaluation before the output stabilizes.
- Node *must be* encapsulated.



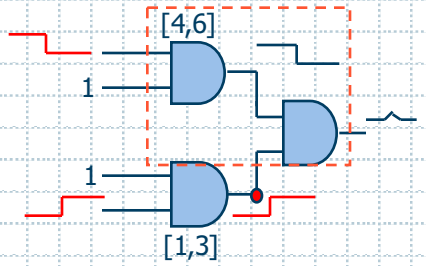
Monotonicity Hazard

- Occurs when an internal or output node becomes excited to change when it should stay stable.
- Prevented by forcing side inputs.



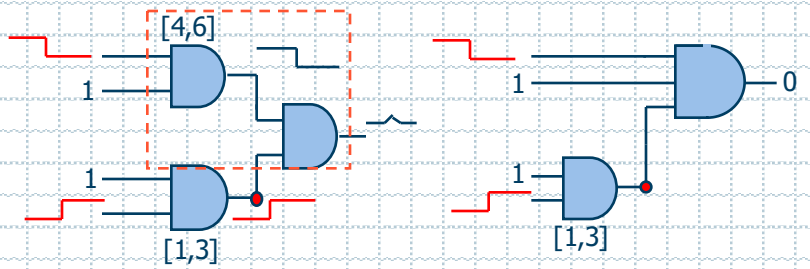
Monotonicity Hazard

- Occurs when an internal or output node becomes excited to change when it should stay stable.
- Prevented by forcing side inputs.
- Fanin node causing hazard must *not be* encapsulated.



Monotonicity Hazard

- Occurs when an internal or output node becomes excited to change when it should stay stable.
- Prevented by forcing side inputs.
- Fanin node causing hazard must *not be* encapsulated.



Hazard-Aware Matching

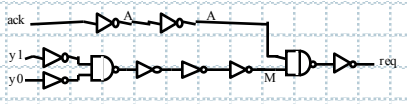
- Each match must undergo cost analysis.
- Objective is to minimize cost.
- Heuristic equation to determine hazard cost:

$hazcost = (W_1)monohaz + (W_2)ackhaz$

W_1, W_2 are coefficients.
monohaz is the number of exposed monotonicity hazards.
ackhaz is the number of exposed acknowledgment hazards.

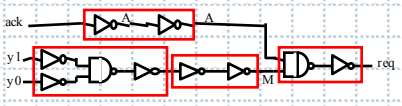
- $W_1, W_2 = 0$: No hazard awareness.
- $W_1, W_2 = 1$: Encourage encapsulation.
- $W_1, W_2 = -1$: Discourage encapsulation.
- $W_1, W_2 = -1, 1$: Discourage monotonicity encapsulation. Encourage acknowledgment encapsulation.

Hazard-Aware Matching Example

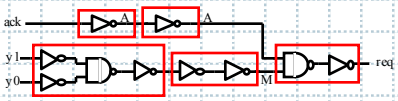


Hazard-Aware Matching Example

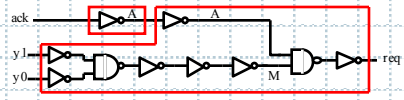
- W_1, W_2 = -1,1: Optimum.



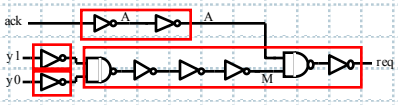
- W_1, W_2 = -1: Discourage encapsulation.



- W_1, W_2 = 0: No hazard awareness.



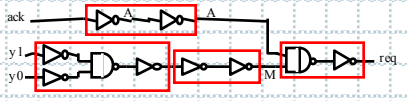
- W_1, W_2 = 1: Encourage encapsulation.



*** Untimed synthesis, verification.**

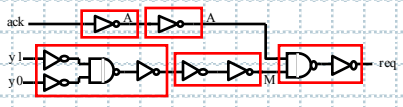
Hazard-Aware Matching Example

- W_1, W_2 = -1,1: Optimum.



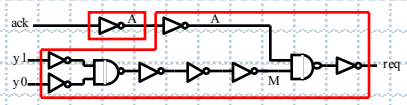
No Hazards.

- W_1, W_2 = -1: Discourage encapsulation.



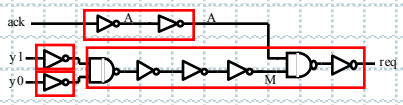
2 Ack hazards.
Mono hazard on req caused by node M.

- W_1, W_2 = 0: No hazard awareness.



1 Ack hazard.
Mono hazards on req caused by y0,y1.

- W_1, W_2 = 1: Encourage encapsulation.



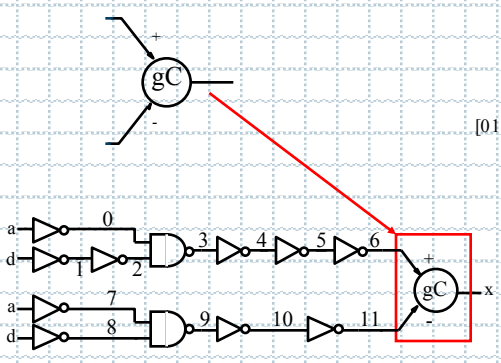
No Hazards.

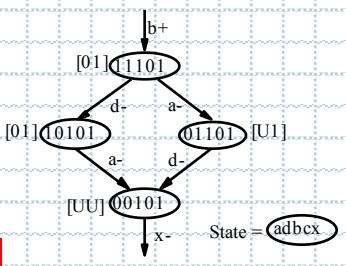
Short-Circuit Issues

- May occur when a gC is mapped to a portion of decomposition.

Short-Circuit Issues

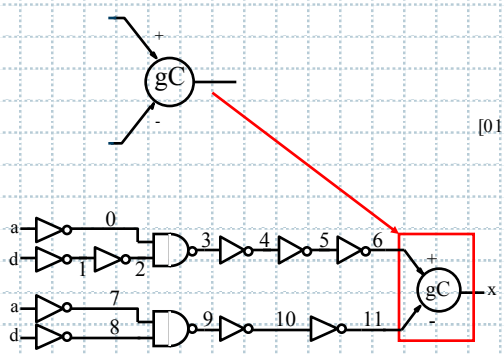
- May occur when a gC is mapped to a portion of decomposition.
- Example: Map a plain gC element to a netlist.

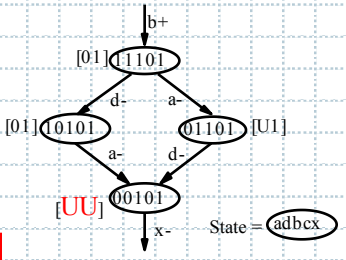




Short-Circuit Issues

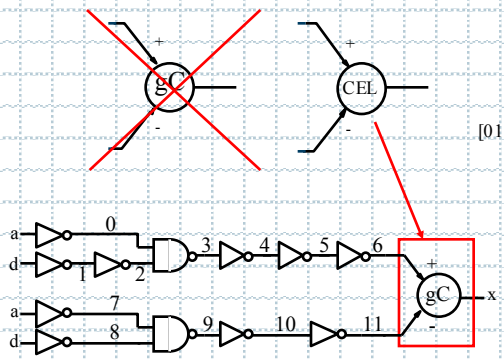
- Potential short-circuit in state 00101.

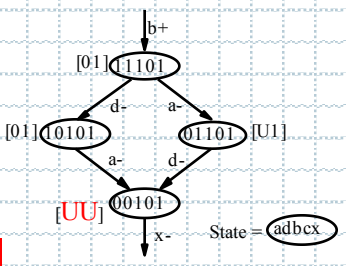




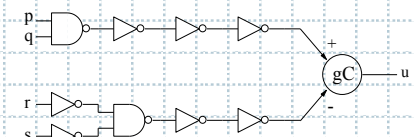
Short-Circuit Issues

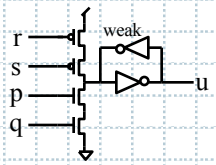
- Potential short-circuit in state 00101.
  ⇒ reject gC and replace with CEL

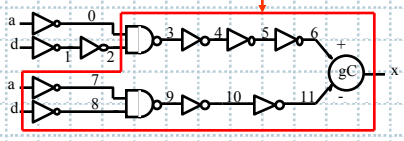


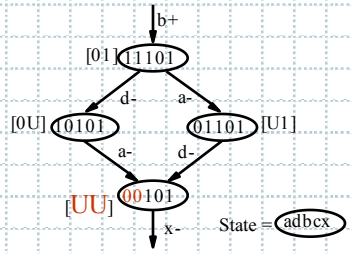


Short-Circuit: Example 2

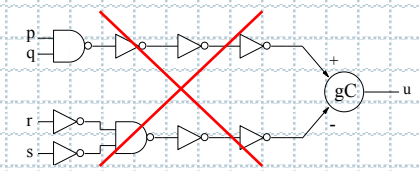




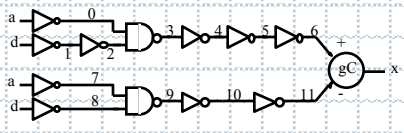


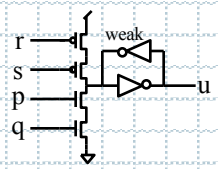


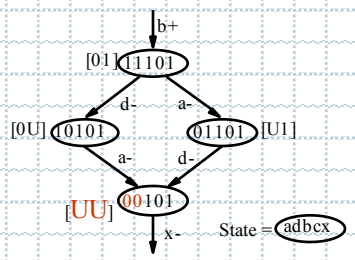
Short-Circuit: Example 2



Reject gC.







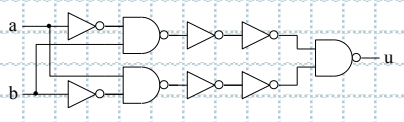
Common-Input Matching

- Must be addressed when a library cell has a common-input on two or more leaves.
- Requires leaves to be driven by equivalent sub-networks.
- Prohibits short-circuit problems if the common-input is found in both set and reset networks of a gC.
- Can increase size of library dramatically.

Common-Inputs: Example 1

- XOR: Non-tree based

$$u = a\bar{b} + b\bar{a}$$

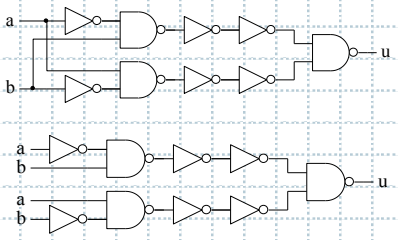


Common-Inputs: Example 1

- XOR: Non-tree based

$$u = a\bar{b} + b\bar{a}$$

- XOR: Tree based

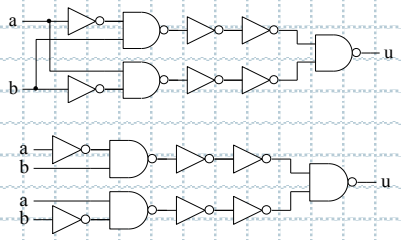


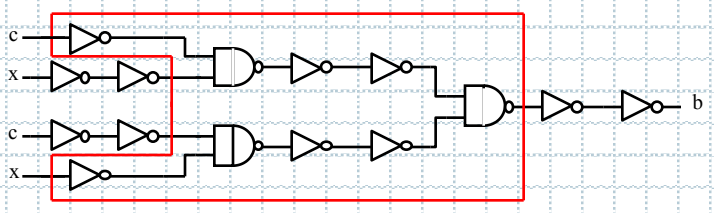
Common-Inputs: Example 1

- XOR: Non-tree based

$$u = a\bar{b} + b\bar{a}$$

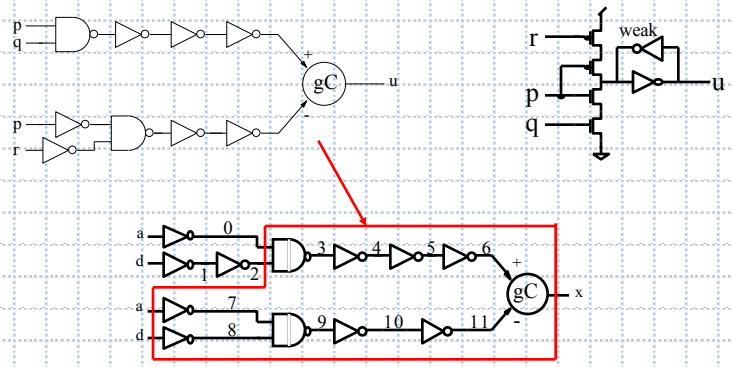
- XOR: Tree based





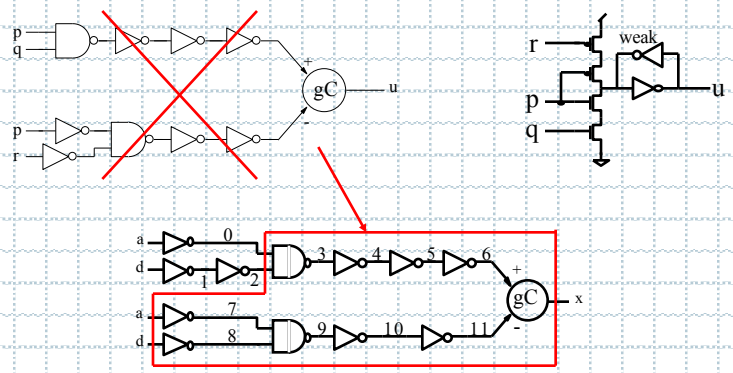
Common-Inputs: Example

- gC22 with 1 common-input (p)



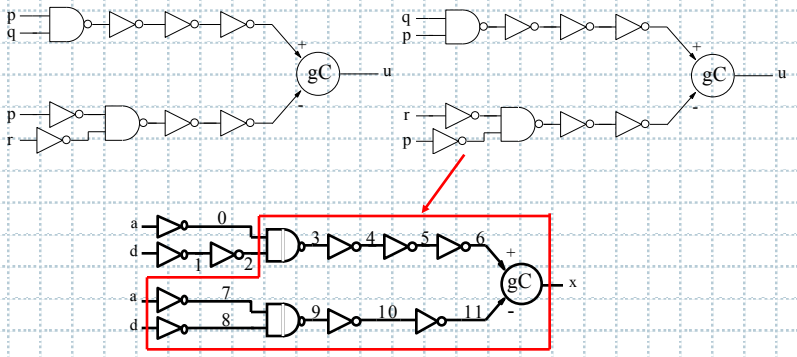
Common-Inputs: Example

- gC22 with 1 common-input (p)



Common-Inputs: Example

- gC22 with 1 common-input (p)



Experimental Results

- Algorithms implemented in 11,545 lines of C/C++ code within the ATACS software tool.
- Tests results presented for:
  - Gate-level verification.
  - Matching and covering.
    - Timed synthesis.
    - Untimed synthesis.
    - Hazard-aware matching.

Gate-level Verification

- Compared our verifier with:
  - KRONOS – timed automata tool, only checks conformance, not hazards.
  - Pena's tools – conservative approximation method, stops after finding first hazard.
  - ATACS – explicit state timing verifier.

Gate-level Verification

Example	Gates	Kronos CPU Time	Pena CPU Time	ATACS CPU Time	New CPU Time	Hazards
converta	12	0.19	12	0.24	0.11	2/2
alloc-outbound	11	0.09	3	0.33	0.09	0/0
chu133	9	0.63	1	0.16	0.11	1/1
ebergen	9	0.14	1	0.15	0.13	3/3
half	7	0.41	1	0.13	0.08	1/1
rpdft	8	2.93	2	0.30	0.10	1/2
sbuf-ram-write	17	31.77	415	0.32	0.20	1/2
ram-read-sbuf	17	>678	550	0.34	0.18	0/0
trimos-send	24	>580	127	10.7	4.87	5/5

Gate-level Verification

Example	Gates	Kronos CPU Time	Pena CPU Time	ATACS CPU Time	New CPU Time	Hazards
converta	12	0.19	12	0.24	0.11	2/2
alloc-outbound	11	0.09	3	0.33	0.09	0/0
chu133	9	0.63	1	0.16	0.11	1/1
ebergen	9	0.14	1	0.15	0.13	3/3
half	7	0.41	1	0.13	0.08	1/1
rpdft	8	2.93	2	0.30	0.10	1/2
sbuf-ram-write	17	31.77	415	0.32	0.20	1/2
ram-read-sbuf	17	>678	550	0.34	0.18	0/0
trimos-send	24	>580	127	10.7	4.87	5/5

Gate-level Verification

Example	Gates	ATACS Time(s)	ATACS Mem (MB)	New Time(s)	New Mem (MB)	Hazards
scsiSV	18	1.35	7.9	0.13	1.3	0/0
slatch	29	33.5	53.4	0.15	1.8	0/0
lapbsv	37	20.0	41.5	0.17	1.3	0/0
elatch	38	183	229	0.28	1.8	0/0
cnt3	80	>1000	>256	0.24	1.7	--/15
srgate	85	>1000	>256	0.29	2.3	--/0
selopt	164	>2000	>256	0.90	3.3	--/46
cnt11	213	>2000	>256	1.20	4.8	--/52

Gate-level Verification

Example	Gates	ATACS Time(s)	ATACS Mem (MB)	New Time(s)	New Mem (MB)	Hazards
scsiSV	18	1.35	7.9	0.13	1.3	0/0
slatch	29	33.5	53.4	0.15	1.8	0/0
lapbsv	37	20.0	41.5	0.17	1.3	0/0
elatch	38	183	229	0.28	1.8	0/0
cnt3	80	>1000	>256	0.24	1.7	--/15
srgate	85	>1000	>256	0.29	2.3	--/0
selopt	164	>2000	>256	0.90	3.3	--/46
cnt11	213	>2000	>256	1.20	4.8	--/52

Gate-level Verification

Example	Gates	ATACS Time(s)	ATACS Mem (MB)	New Time(s)	New Mem (MB)	Hazards
scsiSV	18	1.35	7.9	0.13	1.3	0/0
slatch	29	33.5	53.4	0.15	1.8	0/0
lapbsv	37	20.0	41.5	0.17	1.3	0/0
elatch	38	183	229	0.28	1.8	0/0
cnt3	80	>1000	>256	0.24	1.7	--/15
srgate	85	>1000	>256	0.29	2.3	--/0
selopt	164	>2000	>256	0.90	3.3	--/46
cnt11	213	>2000	>256	1.20	4.8	--/52

Matching: Timed Synthesis

Example	Simple	Combo	S-Stack	Com-Inputs
alloc-outbound	54/0	13/0	13/0	13/0
chu133	34/0	8/0	7/0	6/0
cnt3	80/12	21/0	21/0	22/0
cnt11	213/52	37/0	26/0	26/0
ebergen	43/7	9/2	9/2	6/0
ram-read-sbuf	70/0	18/0	17/0	13/0
sbuf-send-ctl	51/5	12/0	11/0	11/0
trimos-send	87/21	21/3	21/3	18/0

Matching: Timed Synthesis

Example	Simple	Combo	S-Stack	Com-Inputs
alloc-outbound	0			
chu133	0			
cnt3	12			
cnt11	52			
ebergen	7			
sbuf-send-ctl	5			
trimos-send	21			
selopt	128			

Matching: Timed Synthesis

Example	Simple	Combo	S-Stack	Com-Inputs
alloc-outbound	0	0		
chu133	0	0		
cnt3	12	0		
cnt11	52	0		
ebergen	7	2		
sbuf-send-ctl	5	0		
trimos-send	21	3		
selopt	128	19		

Matching: Timed Synthesis

Example	Simple	Combo	S-Stack	Com-Inputs
alloc-outbound	0	0	0	
chu133	0	0	0	
cnt3	12	0	0	
cnt11	52	0	0	
ebergen	7	2	2	
sbuf-send-ctl	5	0	0	
trimos-send	21	3	3	
selopt	128	19	19	

Matching: Timed Synthesis

Example	Simple	Combo	S-Stack	Com-Inputs
alloc-outbound	0	0	0	0
chu133	0	0	0	0
cnt3	12	0	0	0
cnt11	52	0	0	0
ebergen	7	2	2	0
sbuf-send-ctl	5	0	0	0
trimos-send	21	3	3	0
selopt	128	19	19	17

Matching: Untimed Synthesis

Example	Simple	Combo	S-Stack	Com-Inputs
alloc-outbound	14			
chu133	26			
cnt3	60			
cnt11	177			
ebergen	26			
sbuf-send-ctl	17			
trimos-send	121			
selopt	*			

Matching: Untimed Synthesis

Example	Simple	Combo	S-Stack	Com-Inputs
alloc-outbound	14	0		
chu133	26	4		
cnt3	60	15		
cnt11	177	17		
ebergen	26	4		
sbuf-send-ctl	17	4		
trimos-send	121	26		
selopt	*	*		

Matching: Untimed Synthesis

Example	Simple	Combo	S-Stack	Com-Inputs
alloc-outbound	14	0	0	
chu133	26	4	4	
cnt3	60	5	5	
cnt11	177	17	17	
ebergen	26	4	4	
sbuf-send-ctl	17	4	4	
trimos-send	121	26	26	
selopt	*	*	*	

Matching: Untimed Synthesis

Example	Simple	Combo	S-Stack	Com-Inputs
alloc-outbound	14	0	0	0
chu133	26	4	4	4
cnt3	60	5	5	4
cnt11	177	17	17	15
ebergen	26	4	4	3
sbuf-send-ctl	17	4	4	4
trimos-send	121	26	26	28
selopt	*	*	*	*

Matching: Untimed Synthesis

Example	Simple	Combo	S-Stack	Com-Inputs
alloc-outbound	14	0	0	0
chu133	26	4	4	4
cnt3	60	5	5	4
cnt11	177	17	17	15
ebergen	26	4	4	3
sbuf-send-ctl	17	4	4	4
trimos-send	121	26	26	28
selopt	*	*	*	*

Hazard-Aware Matching

Example	(0,0)	(-1,0)	(0,1)	(1,1)	(-1,1)	(-1,2)
alloc-outbound	13/7	13/3	14/0	16/0	14/0	14/0
chu133	6/4	13/10	6/4	6/4	6/4	6/4
converta	15/11	24/16	16/5	16/2	16/5	16/5
ebergen	9/5	13/8	9/3	9/5	9/3	9/3
half	6/4	10/8	5/2	5/2	5/2	5/2
nowick	15/4	18/2	15/2	15/2	17/2	17/2
ram-read-sbuf	17/7	26/12	18/2	18/2	18/2	18/2
rpdft	7/4	10/9	8/4	8/4	8/5	8/4
sbuf-ram-write	13/5	22/13	13/4	13/5	13/4	13/4
sbuf-send-ctl	10/2	20/13	10/2	10/2	10/4	10/4
trimos-send	33/24	56/45	37/28	33/24	37/28	33/24
cnt3	14/11	24/20	13/6	15/4	13/4	13/4
cnt11	28/20	51/40	26/15	28/17	26/15	26/15

Hazard-Aware Matching

- Test conditions for 21 circuits:
  - Untimed synthesis.
  - Untimed verification.
  - Common-input library implementation.

Hazard-Aware Matching

- Test conditions for 21 circuits:
  - Untimed synthesis.
  - Untimed verification.
  - Common-input library implementation.
- Compared matching using various W_1,W_2:
  - W_1,W_2 = 0,0: No hazard awareness.
  - W_1,W_2 = 0,1: Ignore mono, encap ack.
  - W_1,W_2 = -1,1: Discourage mono encap, encap ack.

Hazard-Aware Matching

- Results (win means fewer hazards in final netlist):
  - 0,1 compared to –1,0 19 wins, 0 losses
  - 0,1 compared to 0,0 13 wins, 2 losses
  - 0,1 compared to 1,1 7 wins, 3 losses
  - 0,1 compared to –1,1 4 wins, 1 loss
  - 0,1 compared to –1,2 3 wins, 2 losses

Hazard-Aware Matching

- Results (win means fewer hazards in final netlist):
  - 0,1 compared to –1,0 19 wins, 0 losses
  - 0,1 compared to 0,0 13 wins, 2 losses
  - 0,1 compared to 1,1 7 wins, 3 losses
  - 0,1 compared to –1,1 4 wins, 1 loss
  - 0,1 compared to –1,2 3 wins, 2 losses
- Observations:
  - Must encapsulate acknowledgement hazards.

Hazard-Aware Matching

- Results (win means fewer hazards in final netlist):
  - 0,1 compared to –1,0 19 wins, 0 losses
  - 0,1 compared to 0,0 13 wins, 2 losses
  - 0,1 compared to 1,1 7 wins, 3 losses
  - 0,1 compared to –1,1 4 wins, 1 loss
  - 0,1 compared to –1,2 3 wins, 2 losses
- Observations:
  - Must encapsulate acknowledgement hazards.
  - Give more emphasis to acknowledgment hazards.

Hazard-Aware Matching

- Results (win means fewer hazards in final netlist):
  - 0,1 compared to –1,0 19 wins, 0 losses
  - 0,1 compared to 0,0 13 wins, 2 losses
  - 0,1 compared to 1,1 7 wins, 3 losses
  - 0,1 compared to –1,1 4 wins, 1 loss
  - 0,1 compared to –1,2 3 wins, 2 losses
- Observations:
  - Must encapsulate acknowledgement hazards.
  - Give more emphasis to acknowledgment hazards.
  - Perhaps ignore monotonicity hazards altogether.

Hazard-Aware Matching

- Results (win means fewer hazards in final netlist):
  - 0,1 compared to –1,0 19 wins, 0 losses
  - 0,1 compared to 0,0 13 wins, 2 losses
  - 0,1 compared to 1,1 7 wins, 3 losses
  - 0,1 compared to –1,1 4 wins, 1 loss
  - 0,1 compared to –1,2 3 wins, 2 losses
- Observations:
  - Must encapsulate acknowledgement hazards.
  - Give more emphasis to acknowledgment hazards.
  - Perhaps ignore monotonicity hazards altogether.
  - Try multiple solutions and pick the best.

Conclusions

- Synchronous technology mapping flow is adaptable to timed circuit technology mapping.
- Gate-level hazard verifier using timing is feasible.
  - Results show number of false hazards is small.
  - Method scales well for large examples.
- gC's work well when short-circuit and common-input issues are considered.
- Timed synthesis and verification, hazard-aware matching, in most cases, produce hazard-free circuits.

Conclusions

- Synchronous technology mapping flow is adaptable to timed circuit technology mapping.
- Gate-level hazard verifier using timing is feasible.
- gC's work well when short-circuit and common-input issues are resolved.
- Timed synthesis and verification, hazard-aware matching, in most cases, produce hazard-free circuits.

Contributions

- Adapting synchronous technology mapping flow to timed circuits.
- Efficient gate-level hazard verification using timing.
- Hazard-aware matching and covering utilizing gC's in circuit solutions.
- Evaluating library complexity for implementation.

Contributions

- Adapting synchronous technology mapping flow to timed circuits.
- Efficient gate-level hazard verification using timing.
- Hazard-aware matching and covering algorithms utilizing gC's in circuit solutions.

Future Work

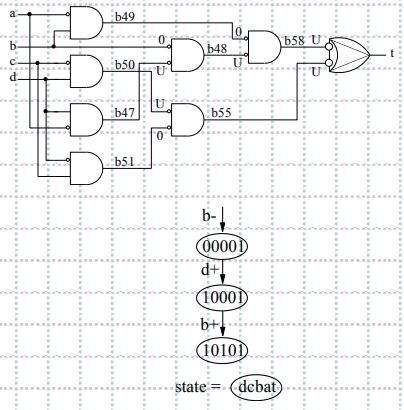
- Investigate:
  - Whether or not hazards are false.
  - Hazard reduction via input re-ordering.
  - Hazard reduction via inertial delay models.
  - Hazard-free outputs with internal hazardous behavior.
  - Circuits with internal cycles.
  - Using other specification forms to increase example suite.
  - Monotonicity hazard creation during hazard-aware matching.

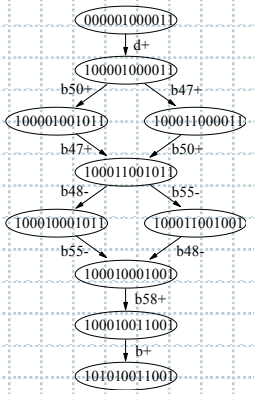
Case Studies

- False Hazards.
- Non-Propagating Acknowledgment Hazard.
- Non-Optimum Covered Circuit.

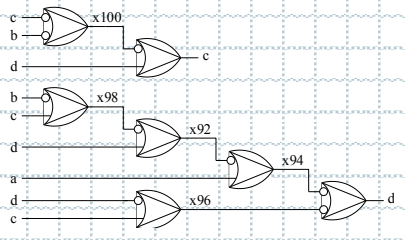
False Hazards

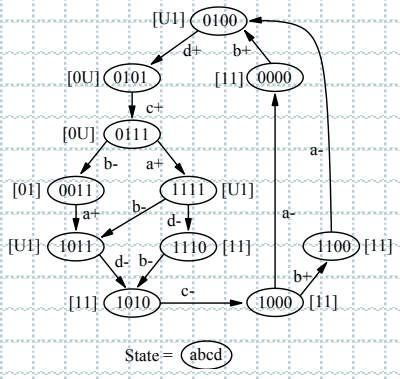
- Mono hazard on output *t* caused by *b58* in state *10001*.



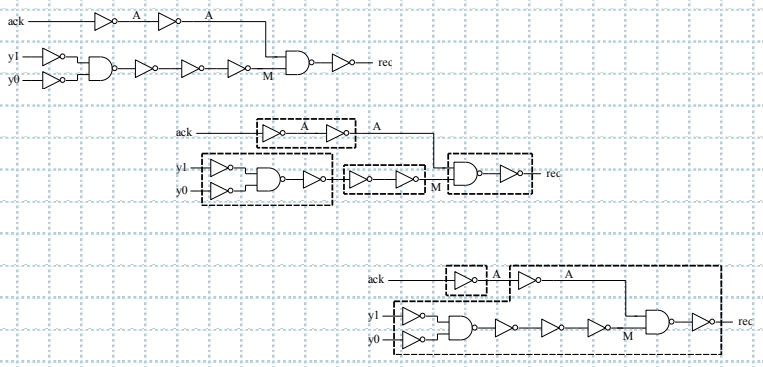


Non-Propagating Ack. Hazard





Non-Optimum Covered Circuit



Simple Example