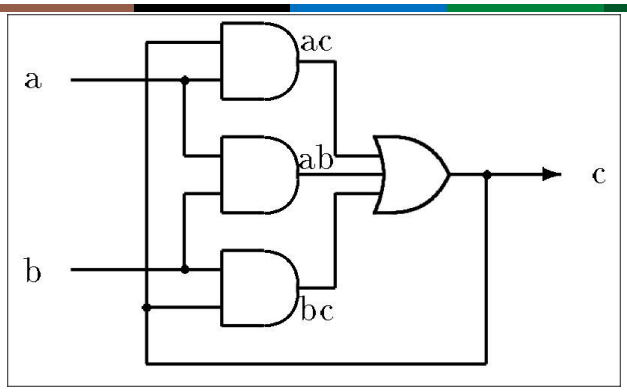
Asynchronous Design Seminar at University of Verona – Lecture Notes 4

Asynchronous Control Circuits – Hazards and Logic Synthesis

Hazards

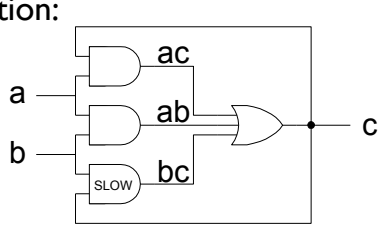
Motivation - C-Element Hazard Example

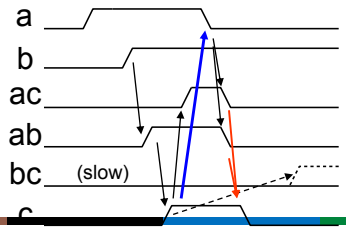


- Consider:
  1. (a, b) = 11 ➔ ab = 1 ➔ c = 1 (before ac, bc = 1)
  2. (a, b) = 10 ➔ ab = 0 ➔ c = 0, ac = 1 ➔ c = 1 (static 1 hazard)

Motivation - C-Element Hazard Example

- Hazard Animation:



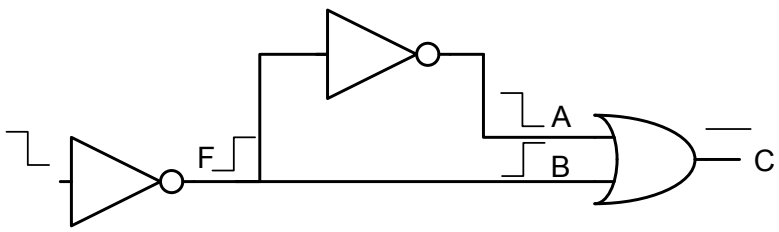


Asynchronous Circuits - Classes

- **Dimension 1: Delay Model**
  - Measure of robustness of control to variations in delays of gates and wires
  - Assumption about delays of gates necessary to ensure design works as dictated by specification
    - Most robust = Arbitrary gate and wire delay
      - Design will work as specified even if delays are random(0,infinity)
      - Larger delays just means control is slower
    - Least robust = Bounded delay on gates and wires
      - If delays are outside these bounds, glitches may occur at outputs or output simply may not transition as expected
- **Dimension 2: Environmental Model**
  - Essentially these are assumptions/restrictions on how fast environment can be for circuit to work

QDI Model – Isochronic Fork

- **Isochronic fork**
- If fork at F is isochronic
  - can assume B fires high before A
  - translates to relative timing assumption about long and short paths

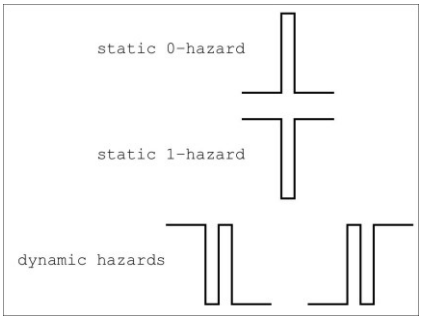


Asynchronous Circuits - Classes

- Timing Model (or Class) is used to define specific timing assumptions with respect to correct circuit operation
  - DI
    - Arbitrary gate and wire delays (unbounded)
  - QDI
    - DI except for Isochronic Forks
      - No need to acknowledge fanouts
  - SI (or Muller) circuits
    - Arbitrary gate delays, bounded wire delays
    - Closed system implementation (gate + environment)
  - Fundamental Mode (Huffman) circuits
    - "Fundamental Mode" Operation:
    - **Outputs and State (local) stabilise before new input change**

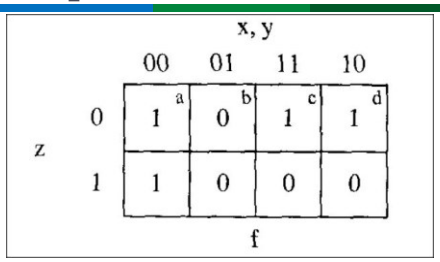
Hazard Types

- Static 0 or 1
- Function Hazard
  - f(A) = f(B), where:
  - A = $(a_1, \ldots, a_p, a_{p+1}, \ldots, a_n)$
  - B = $(a_1', \ldots, a_p', a_{p+1}, \ldots, a_n)$
  - A➔B input vector transition contains 0's and 1's in function cubes
- Logic Hazard
  - Combinational Network static hazard caused by gate delays
- Dynamic
  - Static + Output Change



Function Hazard Example

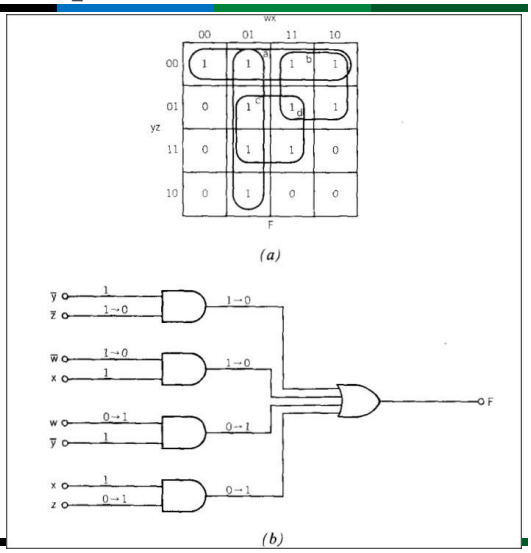
- Consider input transitions:
  000→110 for function f



- Function f contains potential function hazards:
  - for input changes between minterms a→c, a→d, etc. – static 1
  - for input changes between other minterms, e.g. 010→111 – static 0

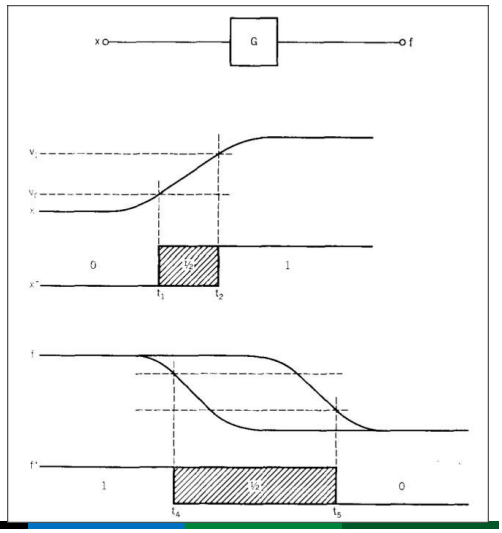
Logic Network Example

- Logic Networks may be free of function hazards but not of logic hazards
- Consider transitions between minterms:
  - a→d, d→a, c→b, b→c
- **Theorem:**
  A 2-level SOP function f is free of logic hazards, iff it contains all Primes (PIs) of f.

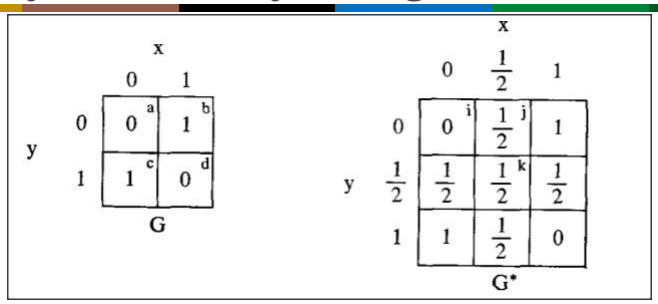


Ternary Approximation to Binary Signals

- A third value, **1/2**, or **X**, may be used to signify the transitive state of a signal
- 3-Valued Algebra may be used to detect and eliminate hazards

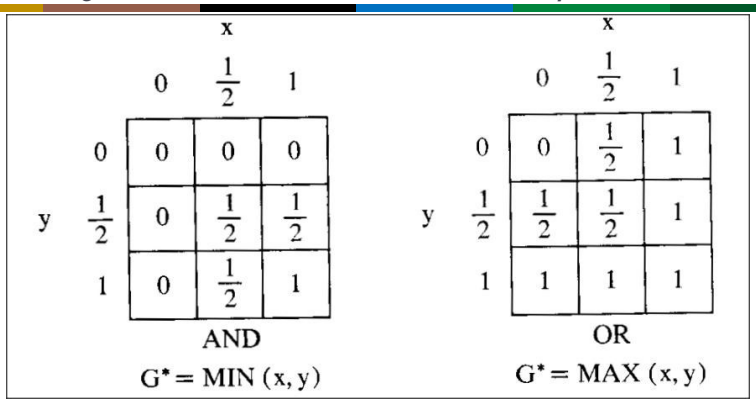


Binary and Ternary XOR gate Truth Table



- Original truth table of logic function used to determine ternary truth table, whereas **½ = 0 OR 1**, e.g. for XOR:
  - 0 (+) ½ = 0 (+) (0 or 1) =1/2, as 0 (+) 0 and 0 (+) 1 produce different outputs
  - ½ (+) ½ = (0 or 1) (+) (0 or 1) = ½
- if p of n inputs are ½, output is 0 or 1 if all 2^p output entries agree

Ternary Truth tables for AND/OR



y \ x	0	1/2	1
0	0	0	0
1/2	0	1/2	1/2
1	0	1/2	1

AND

$G^* = MIN(x, y)$

y \ x	0	1/2	1
0	0	1/2	1
1/2	1/2	1/2	1
1	1	1	1

OR

$G^* = MAX(x, y)$

- AND, OR truth tables easier to derive due to their controlling values:
  - 0 . ½ = 0, 1 + ½ = 1

Ternary Gate Functions Properties

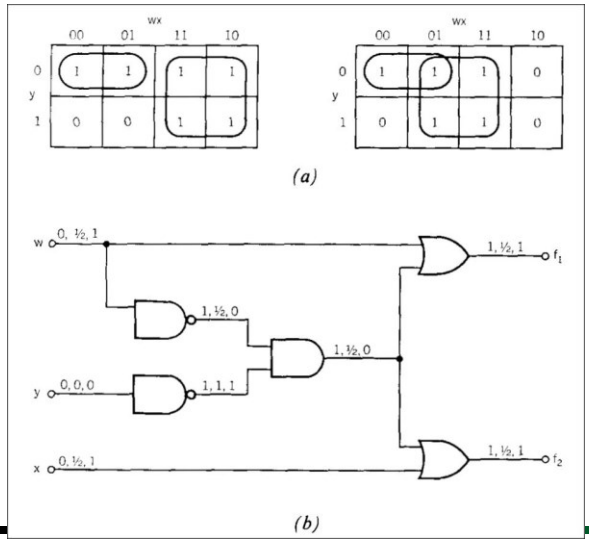
- Property 1:
  - If one or more ternary gate or logic network inputs are changed 1→1/2, or 0→1/2, the ternary outputs will either remain unchanged or change to 1/2
- Property 2:
  - If one or more ternary gate or logic network inputs are changed 1/2→1, or 1/2 → 0, the ternary outputs will either remain unchanged or change to 0 or 1
- Proof:
  - E.B. Eichelberger – *Hazard Detection in Combinational and Sequential Switching Circuits*, IBM Journal, 1965.

Hazard Detection using Ternary Algebra

- Theorem 1:
  - A combinational logic network contains a hazard for an input vector transition from A to B, where:
    - A = $(a_1, \ldots, a_p, a_{p+1}, \ldots, a_n)$
    - B = $(a_1, \ldots, a_v, a_{p+1}, \ldots, a_n)$
    - A/B = $(1/2, \ldots, 1/2, a_{p+1}, \ldots, a_n)$
  - iff (if and only if)
    1. **f(A) = f(B) ≠1/2**
    2. **f(A/B) = 1/2**
- Proof:
  - Based on previous properties

Hazard Detection using Ternary Algebra

- For functions f1, f2
- Consider input change:
  - w′x′y′ → wxy′
- is a hazard produced for f1, f2?
- Must determine:
  - f(w′, x′, y′), f(½ , ½, y′), f(w, x, y′)
- Outcome:
  - (1, ½, 1)
  - f1, f2 both contain a hazard



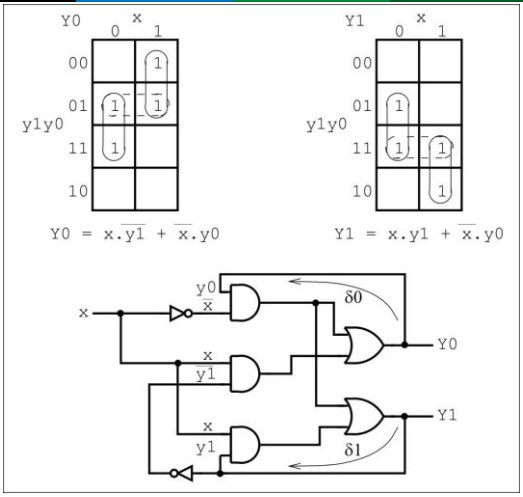
Additional Hazards in Sequential Circuits

- Critical Race
  - If order of changes in state variables affects final state, race is critical
  - 11 → 10 → 00, or, 11→ 01 → 00
- Essential Hazard
  - Critical race between input and feedback change – must add delay to fix
  - Property of FSM specification
- Essential Hazard Detection:
  - for input vector A→B,
  - If single change (A, B) produces different states and output to three changes, *i.e.* (A→B, A, B) circuit contains an **Essential Hazard**

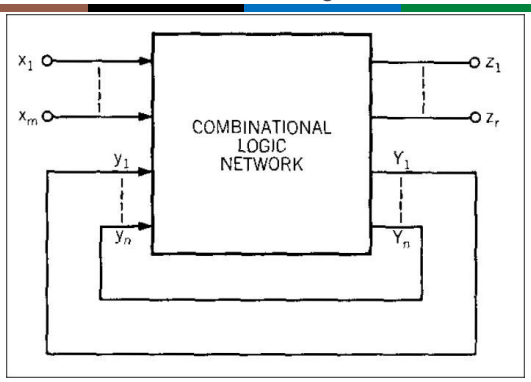
Essential Hazard Example

	x	
Y1Y0	0	1
1 (00)	1, 0	2, 0
2 (01)	3, 0	2, 0
3 (11)	3, 0	4,0
4 (10)	1,1	4,1

- δ0, δ1 are feedback delays for state signals `Y1, Y0`
  - `Y1, Y0` → `y1, y0`



Sequential Hazard Analysis



- Sequential Circuit Model:
  - Input vector ($x_1 \ldots x_m$) changes
  - Next State signals ($Y_1 \ldots Y_n$) change as a response to input change
  - Current State signals ($y_1 \ldots y_n$) change as a response to next state change

Sequential Hazard Analysis

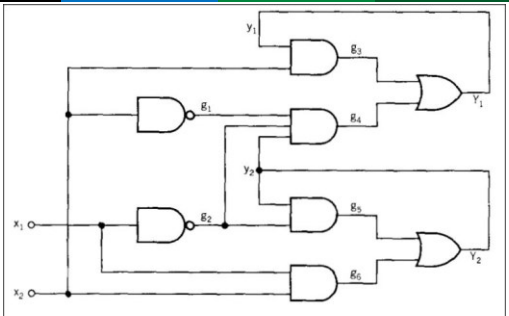
- Eichelberger's Two-Step Approach:
  - Procedure/Step 1 – Determine all ***changing*** Y signals
    1. Set changing input vector signals to intermediate 1/2 values, and all other x or y signals to their previous values
    2. Evaluate Y_i functions to determine changes from 1 or 0 to 1/2
    3. Propagate any 1/2 Y_i change to corresponding y_i change and repeat process until no further changes to Y_i occur
  - Procedure/Step 2 – Determine all ***stabilising*** Y signals
    - Set changing input vector signals to their final values, 1 or 0, and all other x or y signals to their previous values, as determined by Procedure 1
    - Evaluate Y_i functions to determine changes from 1/2 to 1 or 0
    - Propagate any 0 or 1 Y_i change to corresponding y_i change and repeat process until no further changes to Y_i occur

Sequential Hazard Analysis

- Theorem 2:
  - If Y_k = 1(0) after applying Procedures A and B to a sequential circuit for a given input change starting from a given internal state, then the Y_k signal must stabilise at 1(0) for this transition, regardless of the values of the finite delays of the logic gates
- Proof:
  - Based on previous Theorem (Theorem 1)

Sequential Hazard Analysis - Example

- Determine whether 00→11 change in x_1, x_2 inputs results in indeterminate final state



(a)

	x_1	x_2	y_1	y_2	g_1	g_2	g_3	g_4	g_5	g_6	Y_1^*	Y_2^*
1	0	0	0	0	1	1	0	0	0	0	0	0
2	½	½	0	0	½	½	0	0	0	½	0	½
3	½	½	0	½	½	½	0	½	½	½	½	½
4	1	1	½	½	0	0	½	0	½	1	½	1
5	1	1	½	1	0	0	½	0	1	1	½	1

(b)

Ternary Simulation Characteristics

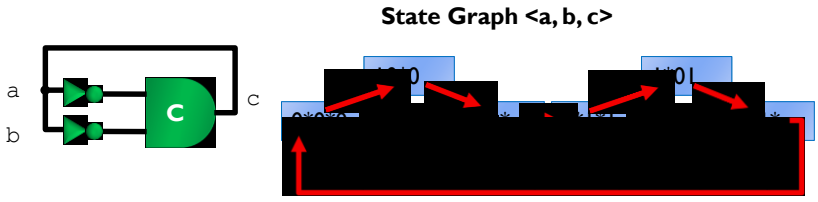
- For n feedback lines, at most 2 x n evaluations are required
- Hazards and Races are detected automatically
- ***Optimisations***
  - During Procedure A, any gate with output at 1/2 need not be further considered, since output cannot change further
  - During Procedure B, any gate with output different from 1/2 need not be further considered, since again its output cannot change further
    - In both cases remove gate from simulation queue

Signal Transition Graph (STG) -based Logic Synthesis

of Asynchronous Control

Understanding SI Model

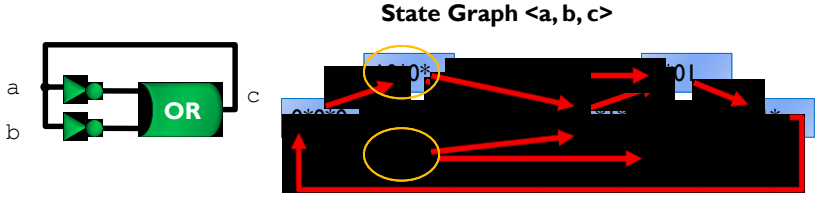
- Check circuit for disabled transitions in State Graph:



- There are no disabled transitions
  1*→1 or 0*→0 in the State Graph
  - Thus circuit is SI
- This analysis assumes the unbounded delay model

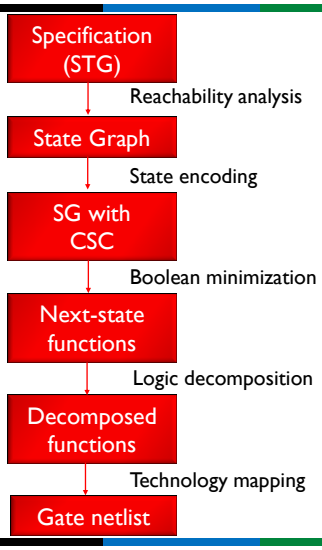
Understanding SI Model

- Check circuit for disabled transitions in State Graph:

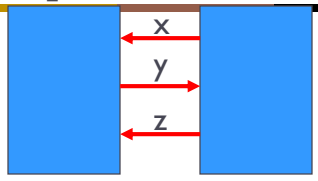


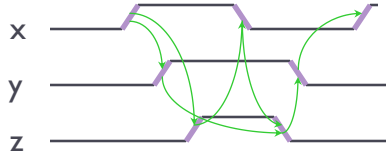
- Disabled transitions 1*→1 or 0*→0 in the State Graph
  - Thus circuit is not SI
  - Circuit is also not semi-modular
- This analysis assumes the unbounded delay model

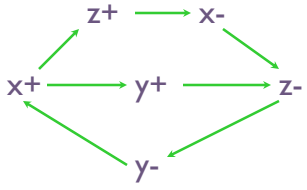
Design flow



Specification

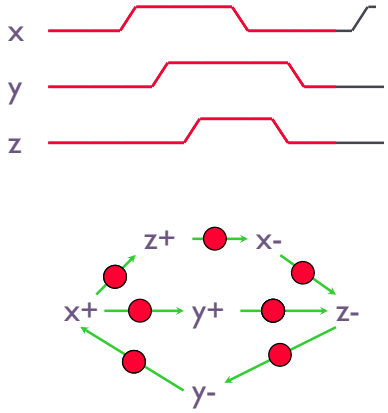




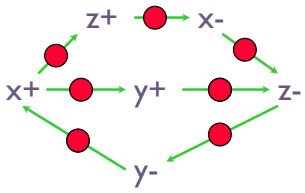


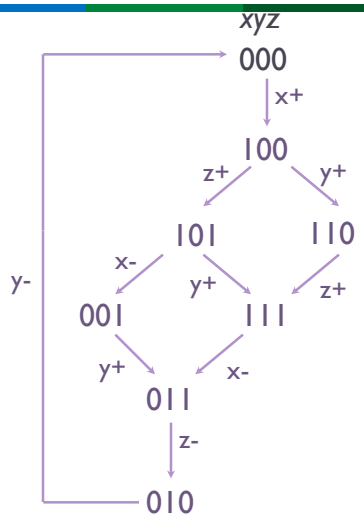
Signal Transition Graph (STG)

Token flow



State graph



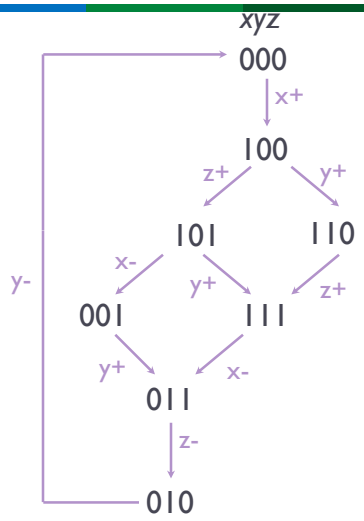


Next-state functions

$$x = \bar{z} \cdot (x + \bar{y})$$

$$y = z + x$$

$$z = x + \bar{y} \cdot z$$

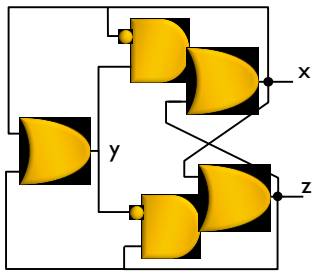


Gate netlist

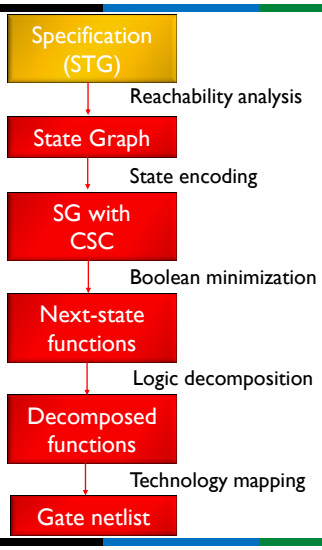
$$x = \bar{z} \cdot (x + \bar{y})$$

$$y = z + x$$

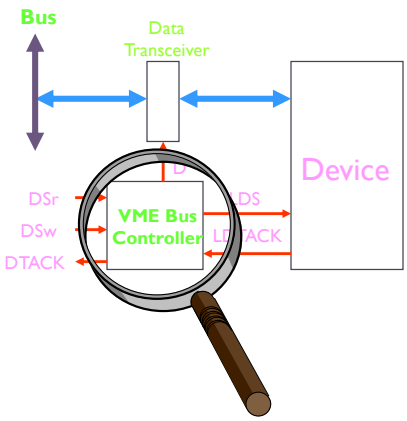
$$z = x + \bar{y} \cdot z$$

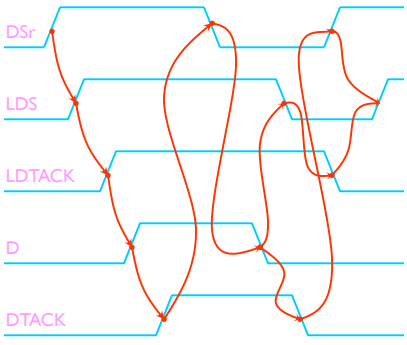


Design flow



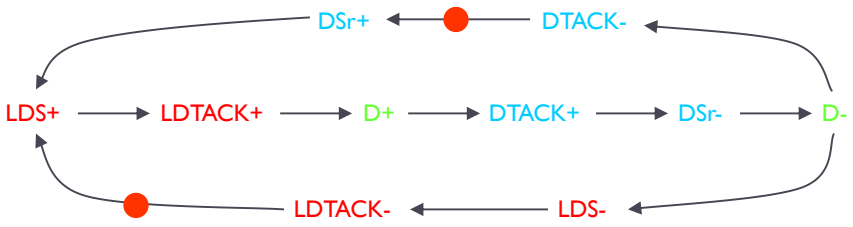
VME Bus Example

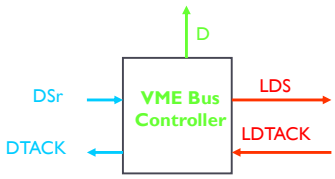




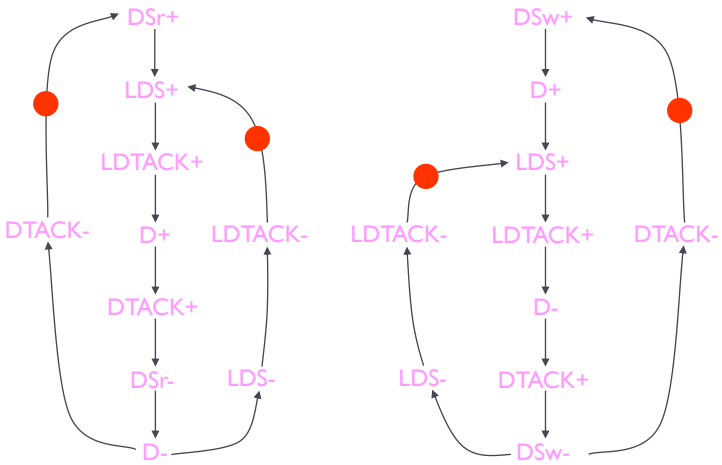
Read Cycle

STG for READs

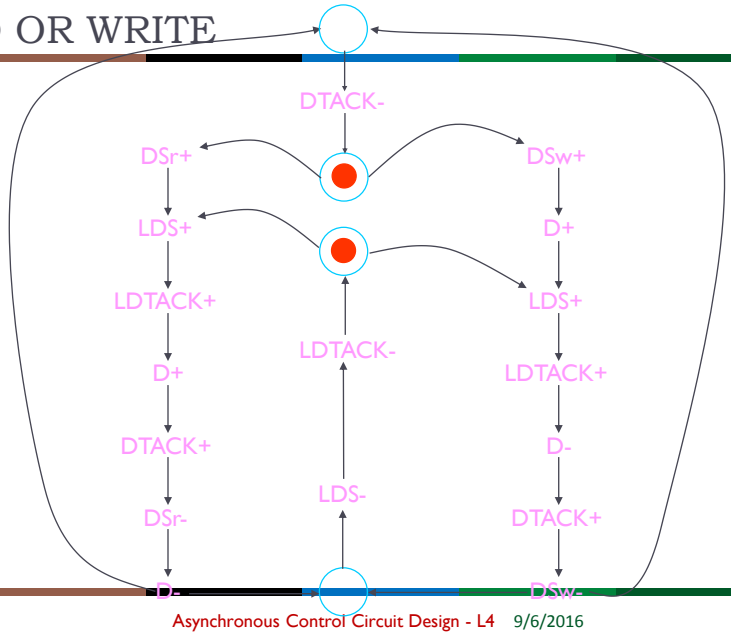




NEED Choice to select between READ OR WRITE



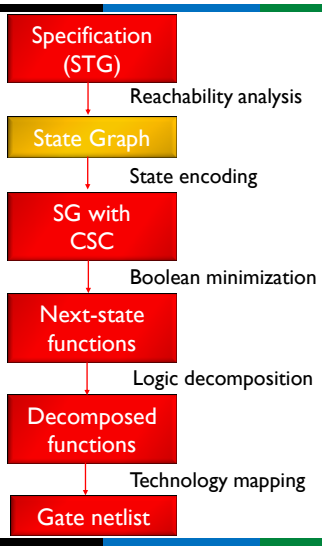
NEED Choice to select between READ OR WRITE



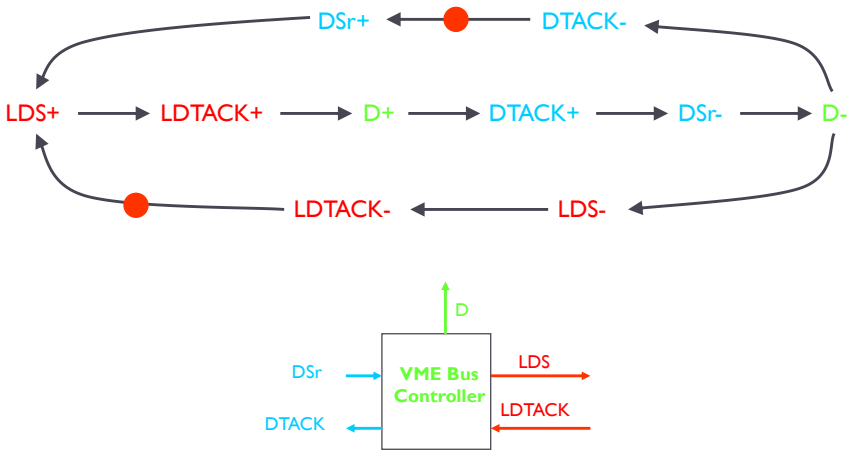
SI Asynchronous Circuit Synthesis

- Goal:
  - Derive a hazard-free circuit under a given delay model and mode of operation
- Speed Independence
  - Unbounded gate / environment delays
  - Certain wire delays shorter than certain paths in the circuit
    - Wires LONGER than GATES!!!
- SI Implementability Conditions
  - Consistency
    - Signal transitions alternate in all PTnet paths and thus Reachability Graph
  - Complete State Coding (CSC)
    - Each pair of Reachability Graph States have different state encoding, or if the share the same encoding, they enable different non-input (output) signals ➔ distinguishable
  - Persistency ➔ Semi-Modularity
    - Outputs cannot be disabled once enabled, Inputs cannot be disabled by Outputs

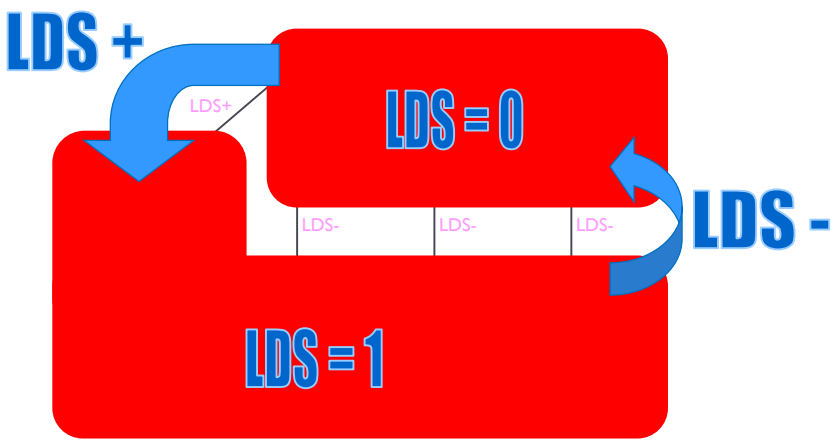
Design flow



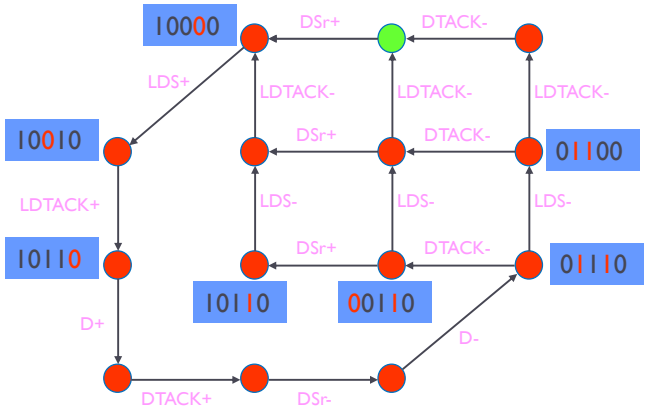
STG for the READ cycle



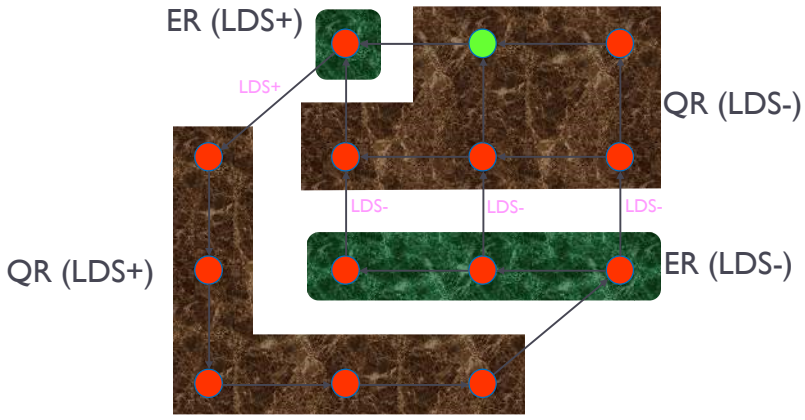
Reachability Graph – Binary Encoding



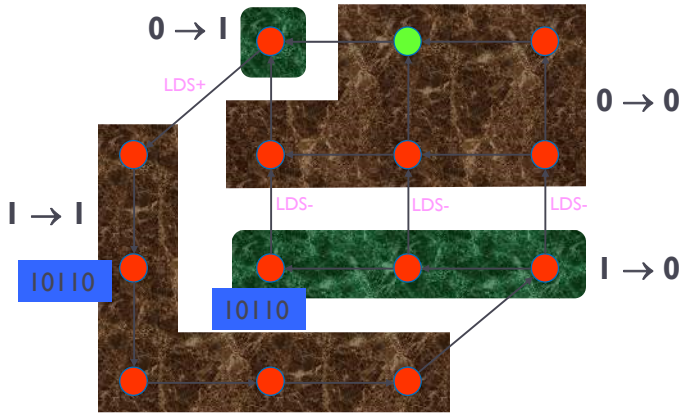
Reachability Graph – Binary Encoding



Defining Excitation and Quiescent Regions



Forming the Next State Function



Extracting the Boolean Expression of the Next State Function

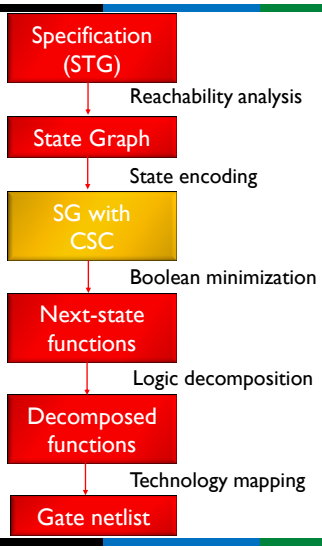
LDS = 0

D LDTACK \ DTACK DSr	00	01	11	10
00	0	0	-	1
01	-	-	-	-
11	-	-	-	-
10	0	0	-	0

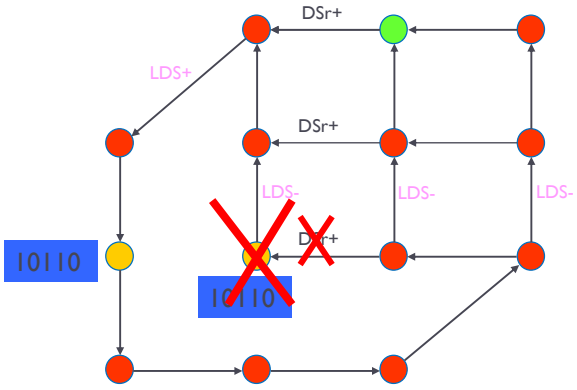
LDS = 1

D LDTACK \ DTACK DSr	00	01	11	10
00	-	-	-	1
01	-	-	-	-
11	-	1	1	1
10	0	0	-	0/1?

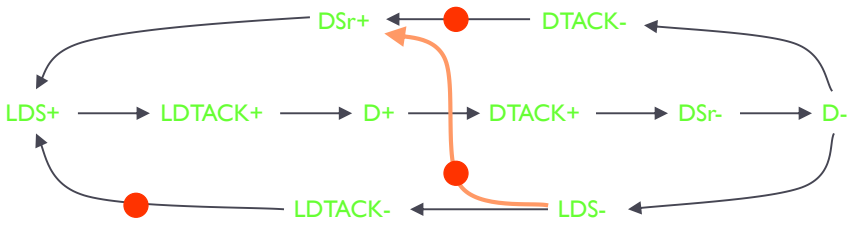
Design flow



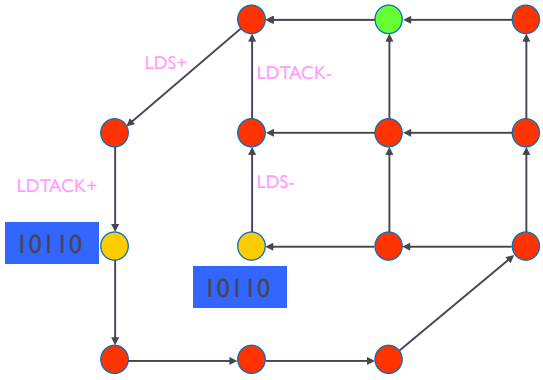
Concurrency Reduction (Manual/Automatic) at State Graph Level



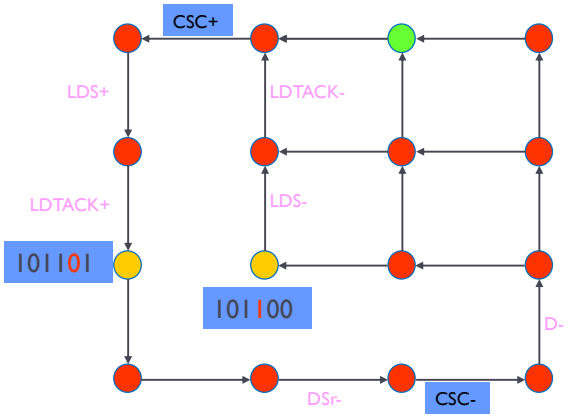
Concurrency Reduction – Migration to STG/PTnet Level



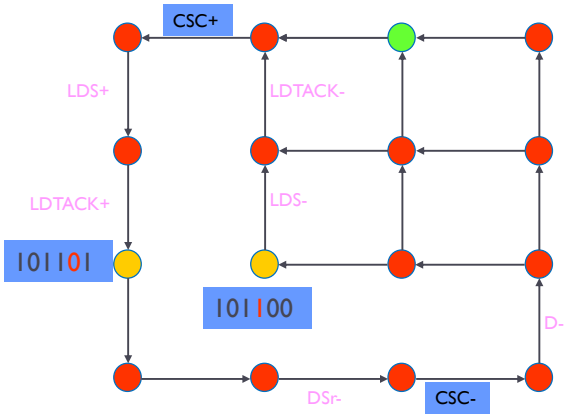
State Encoding Conflicts



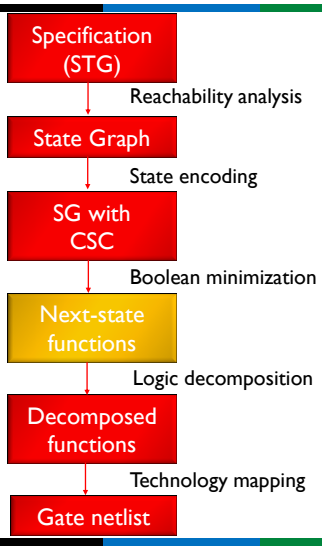
Resolving Conflicts through Signal Insertion



Signal Insertion



Design flow



Complex-Gate Implementation

$$LDS = D + \text{csc}$$

$$DTACK = D$$

$$D = LDTACK \cdot \text{csc}$$

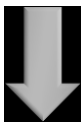
$$\text{csc} = DSr \cdot (\text{csc} + \overline{LDTACK})$$

Implementability Conditions - Revisited

- Consistency
  - Rising and falling transitions of each signal alternate in any trace
- Complete state coding (CSC)
  - Next-state functions correctly defined
- Persistency
  - No event can be disabled by another event (unless they are both inputs)

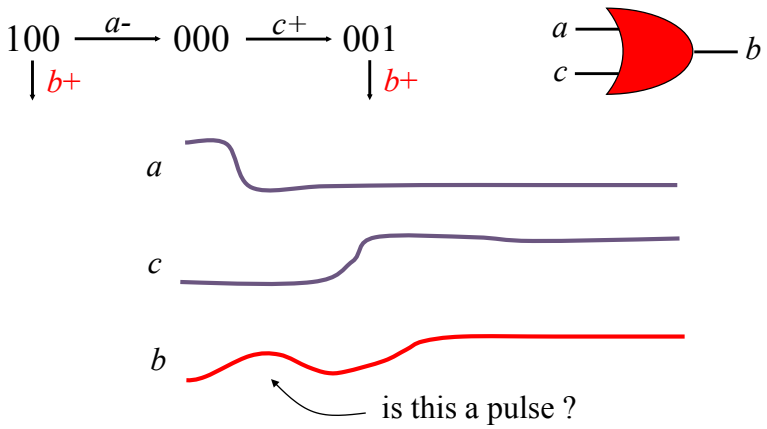
Implementability Conditions - Revisited

- Consistency + CSC + persistency



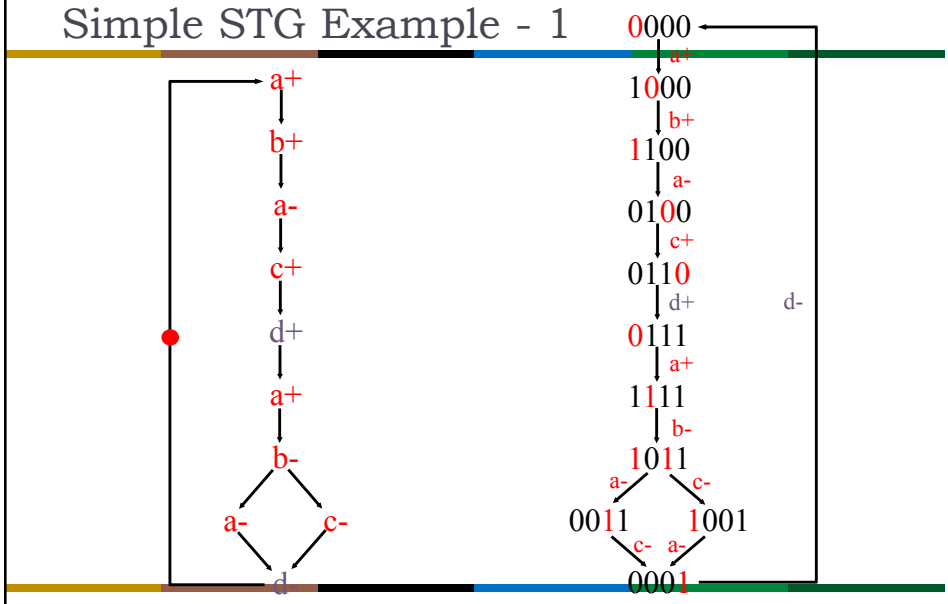
- There exists a speed-independent circuit that implements the behavior of the STG
  - under the assumption that any Boolean function can be implemented with one complex gate

Understanding Persistency



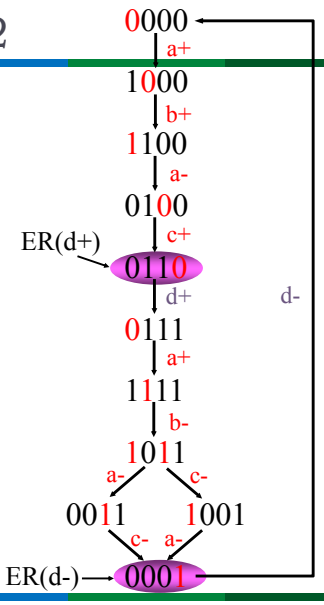
Speed independence ⇒ glitch-free output behavior under any delay

Simple STG Example - 1

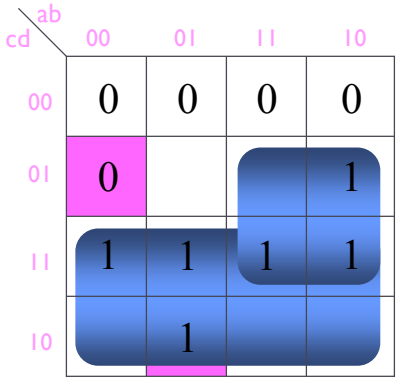


Simple STG Example - 2

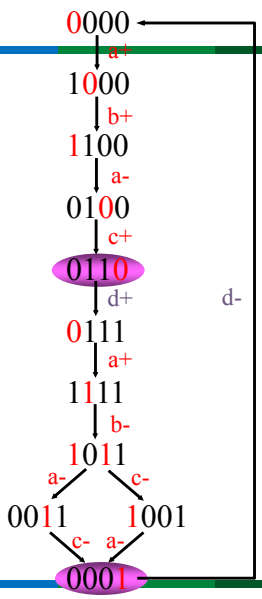
cd \ ab	00	01	11	10
00	0	0	0	0
01	0			1
11	1	1	1	1
10		1		



Simple STG Example - 3



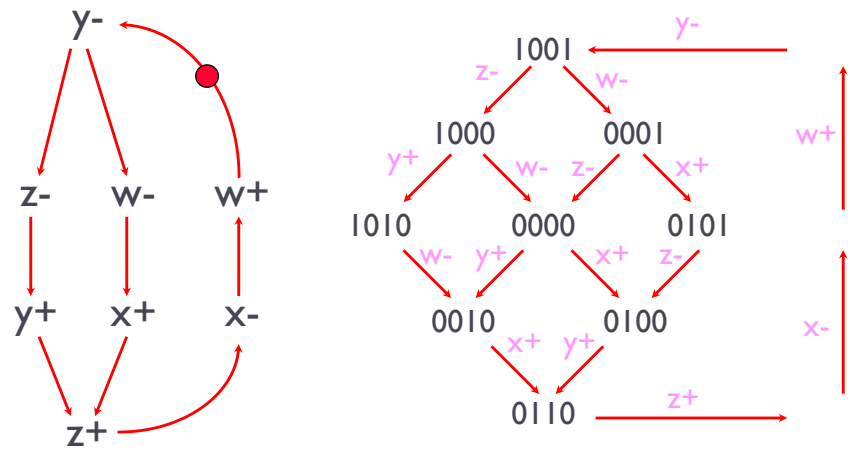
$$d = c + ad$$



Speed-Independence - Summary

- Implementability conditions
  - Consistency
  - Complete State Coding (CSC)
  - Persistency
- Circuit architectures
  - Complex (hazard-free) gates
  - C elements with monotonic covers

Synthesis Exercise

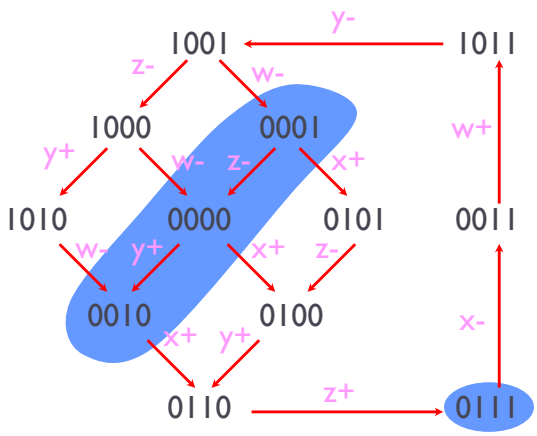


- Derive circuits for outputs x and z
  - Both complex gate and C-element based implementations

Synthesis Exercise – x Output

yz \ wx	00	01	11	10
00	1	1	-	0
01	1	1	-	0
11	0	0	-	0
10	1	1	-	0

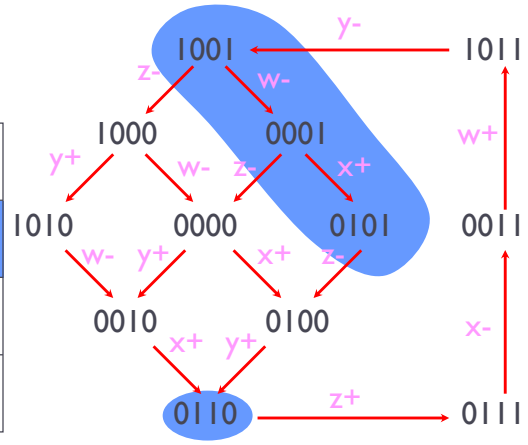
Signal x



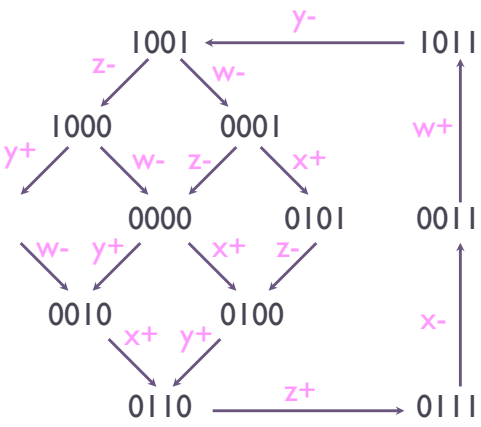
Synthesis Exercise – z Output

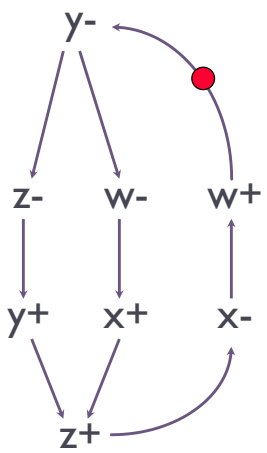
yz \ wx	00	01	11	10
00	0	0	-	0
01	0	0	-	0
11	1	1	-	1
10	0	1	-	0

Signal z

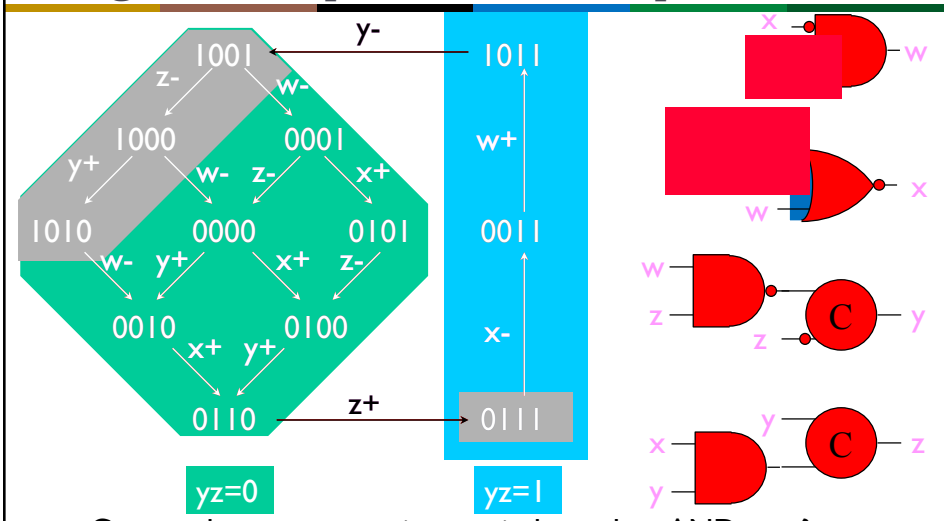


Logic Decomposition - Example



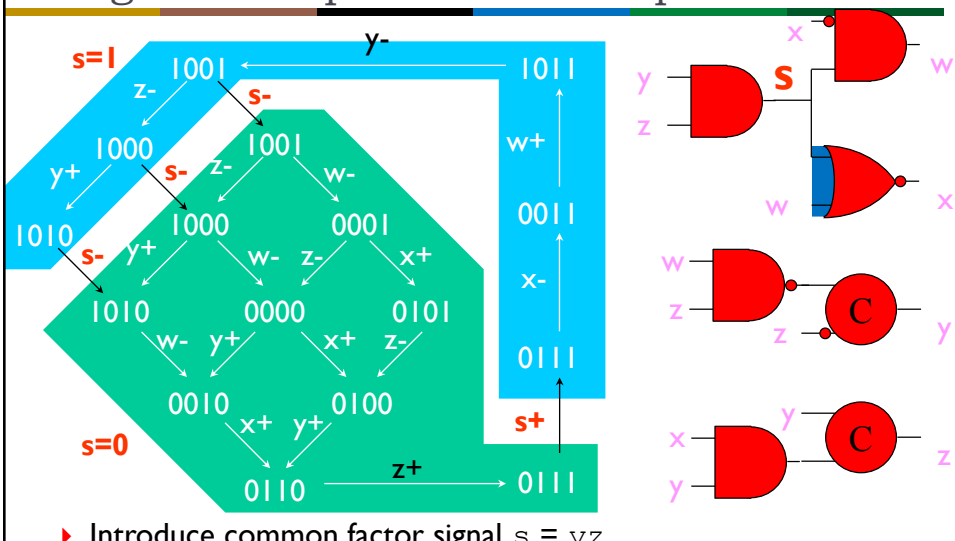


Logic Decomposition - Example



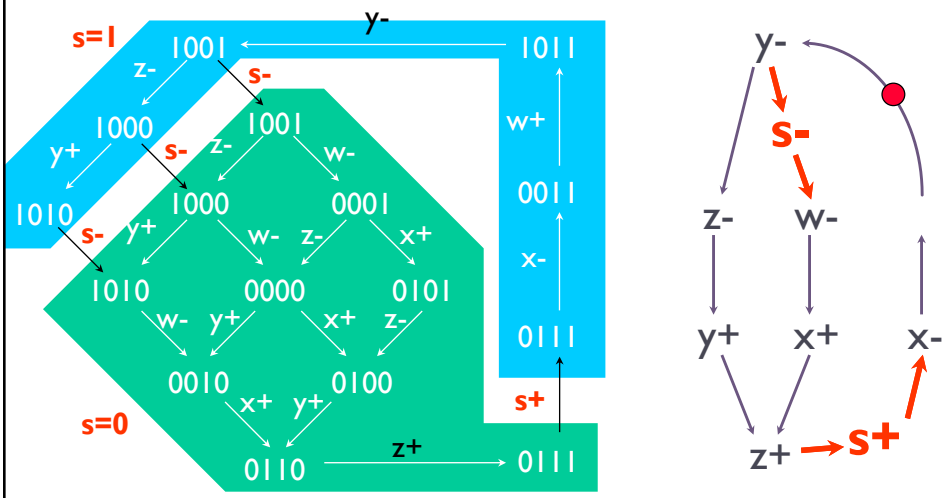
- Can we decompose yz into an independent AND gate?

Logic Decomposition - Example



- Introduce common factor signal s = yz

Logic Decomposition - Example

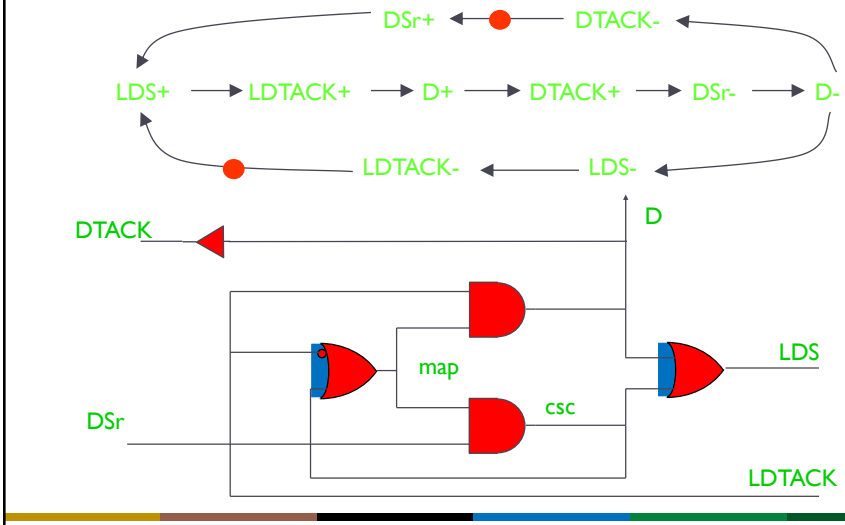


- Signal `s` can now be added back to the STG/PTnet

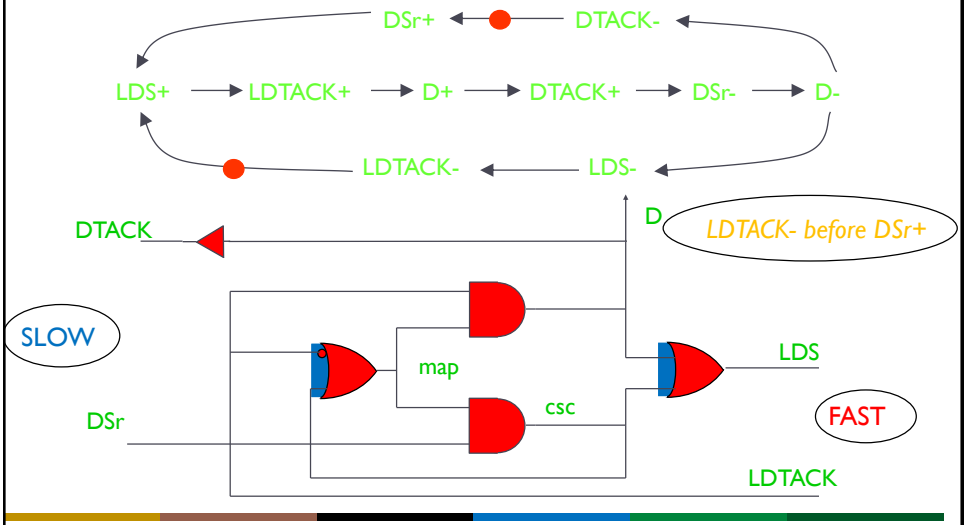
Timing Assumptions

- Relative Timing Assumptions can ***significantly*** reduce circuit complexity
  - Timing assumptions effectively remove or make redundant PTnet/STG edges
  - Extreme example:
    - `Ain` is not necessary, as controller and receiver are faster than sender
  - Each timing assumption must be guaranteed by timing constraints at schematic or physical level or even system level
- Relative Timing Assumptions can be used to optimise timing by a great deal!

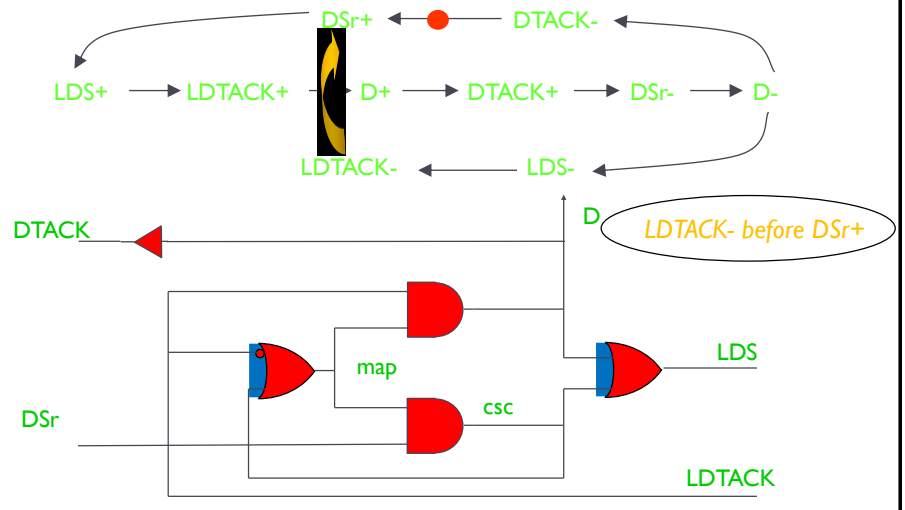
Timing Assumptions Example – SI Netlist



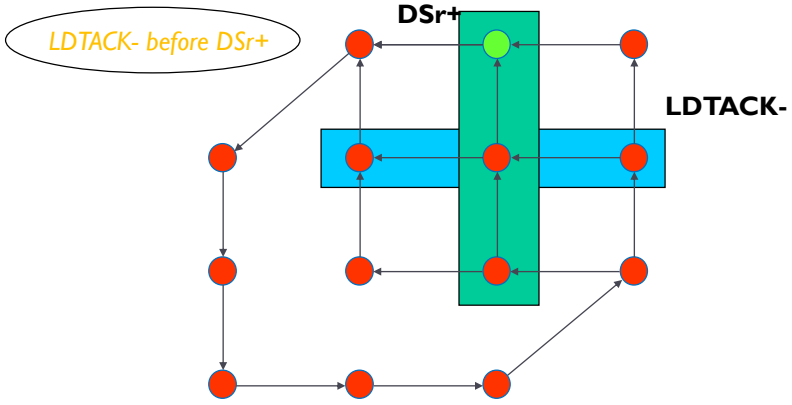
Timing Assumptions Example – SI Netlist – Adding Timing Assumptions



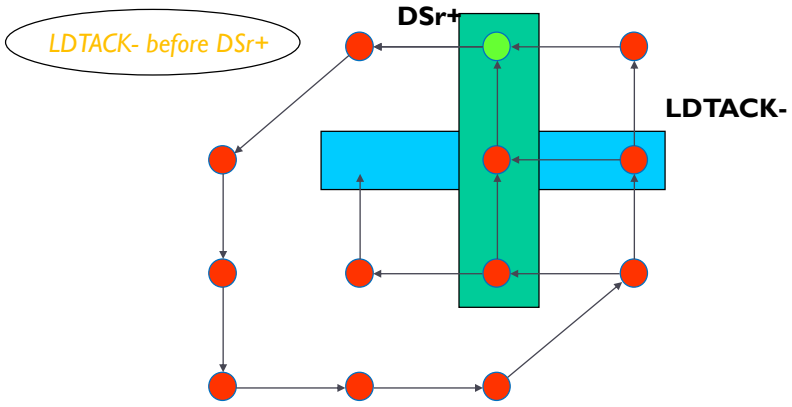
Timing Assumptions Example – SI Netlist – Adding Timing Assumptions



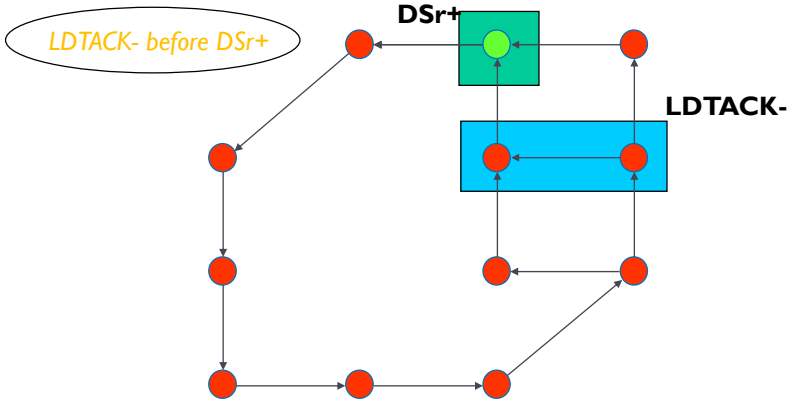
Timing Assumptions Example – SI Netlist – Adding Timing Assumptions – State Graph



Timing Assumptions Example – SI Netlist – Adding Timing Assumptions – State Graph



Timing Assumptions Example – SI Netlist – Adding Timing Assumptions – State Graph



Two more unreachable states

Timing Assumptions Example – SI Netlist – Adding Timing Assumptions – Boolean Logic

LDS = 0

D \ DTACK DSr	00	01	11	10
00	0	0	-	1
01	-	-	-	-
11	-	-	-	-
10	0	0	-	0

LDS = 1

D \ DTACK DSr	00	01	11	10
00	-	-	-	1
01	-	-	-	-
11	-	1	1	1
10	0	0	-	0/1?

- Original Circuit had CSC issue!!!

Timing Assumptions Example – SI Netlist – Adding Timing Assumptions – Boolean Logic

LDS = 0

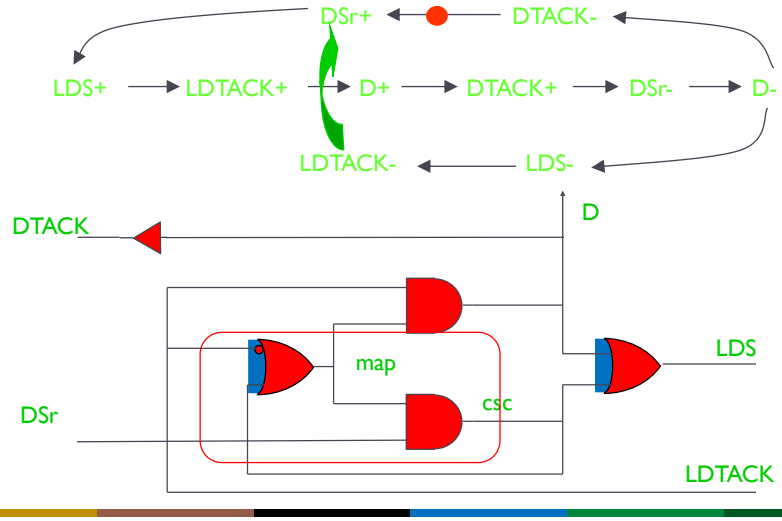
D \ DTACK DSr	00	01	11	10
00	0	0	-	1
01	-	-	-	-
11	-	-	-	-
10	0	0	-	-

LDS = 1

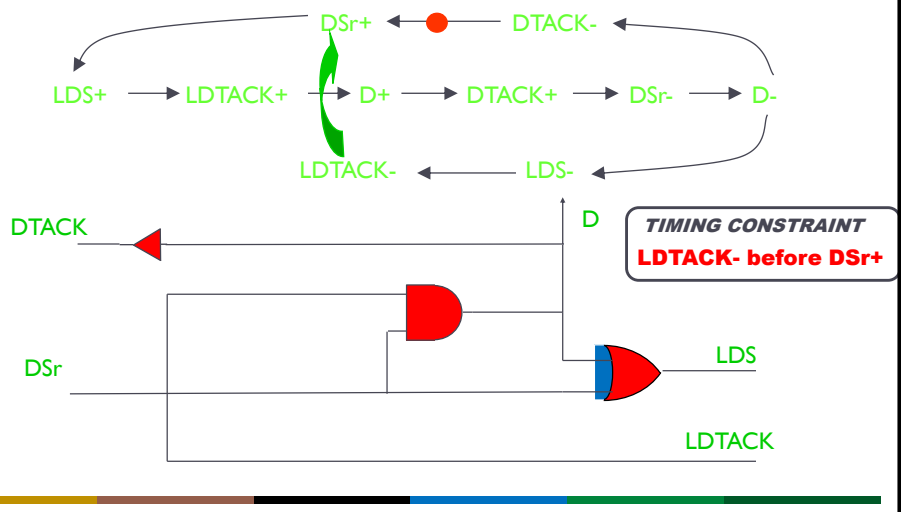
D \ DTACK DSr	00	01	11	10
00	-	-	-	1
01	-	-	-	-
11	-	1	1	1
10	0	0	-	1

- Timing assumptions add DC and resolve CSC!!!

Timing Assumptions Example – SI Netlist with Timing Constraint



Timing Assumptions Example – SI Netlist with Timing Constraint



STG Logic Synthesis - Conclusions

- STGs have a high expressiveness power at a low level of granularity (similar to FSMs for synchronous systems)
- Very effective approach for asynchronous control circuit design
- Not suitable for datapath design
- Circuits with choice require attention for determinism (no confusion!)
- Synthesis from STGs can be fully automated
- Synthesis tools often suffer from the state explosion problem (symbolic techniques are used)
  - State Space generation is exponential