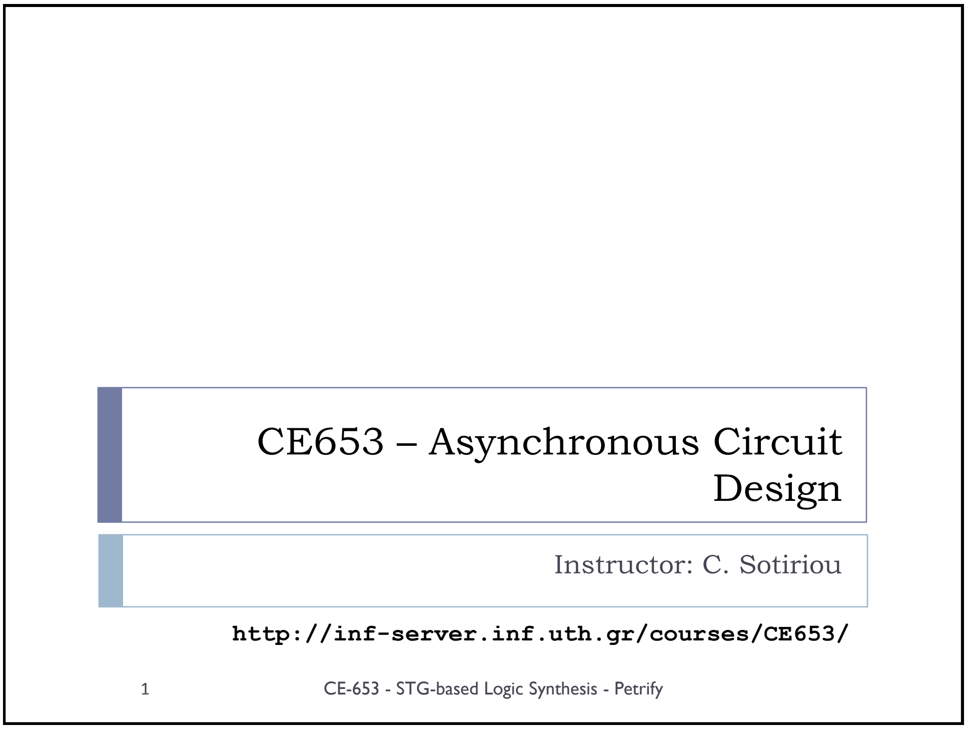
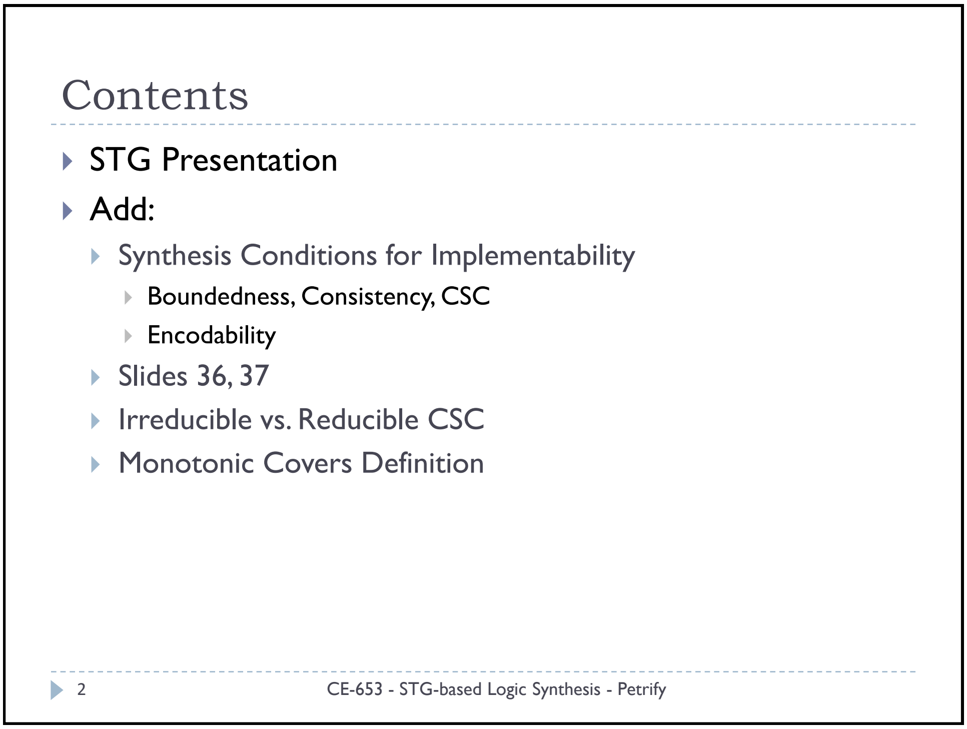


CE653 – Asynchronous Circuit Design

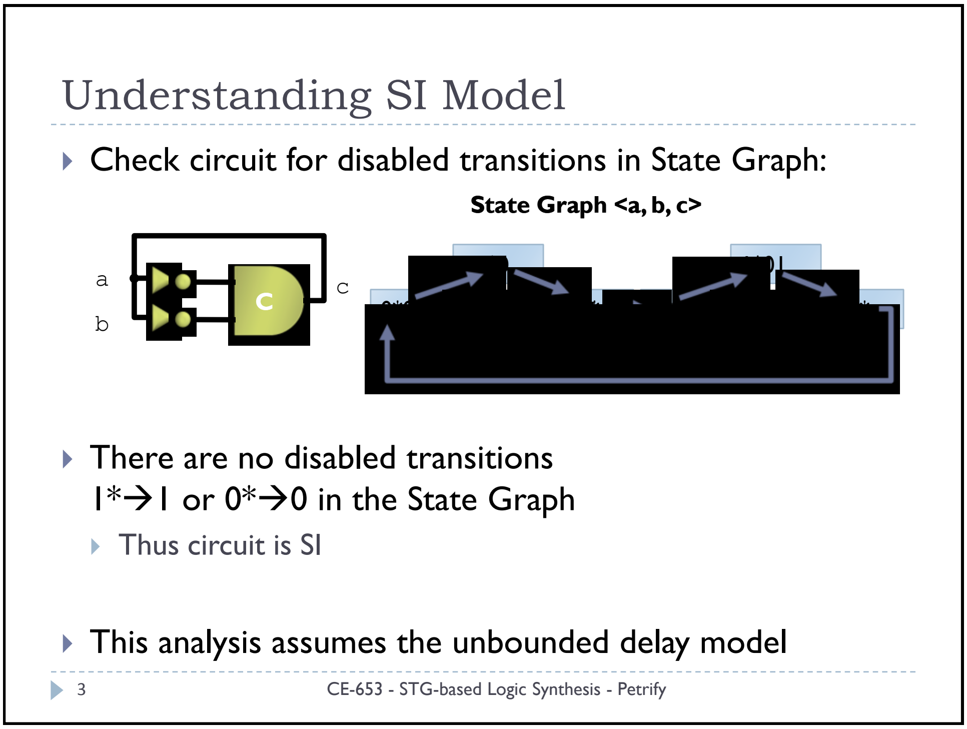
Instructor: C. Sotiriou

http://inf-server.inf.uth.gr/courses/CE653/



Contents

- STG Presentation
- Add:
  - Synthesis Conditions for Implementability
    - Boundedness, Consistency, CSC
    - Encodability
  - Slides 36, 37
  - Irreducible vs. Reducible CSC
  - Monotonic Covers Definition

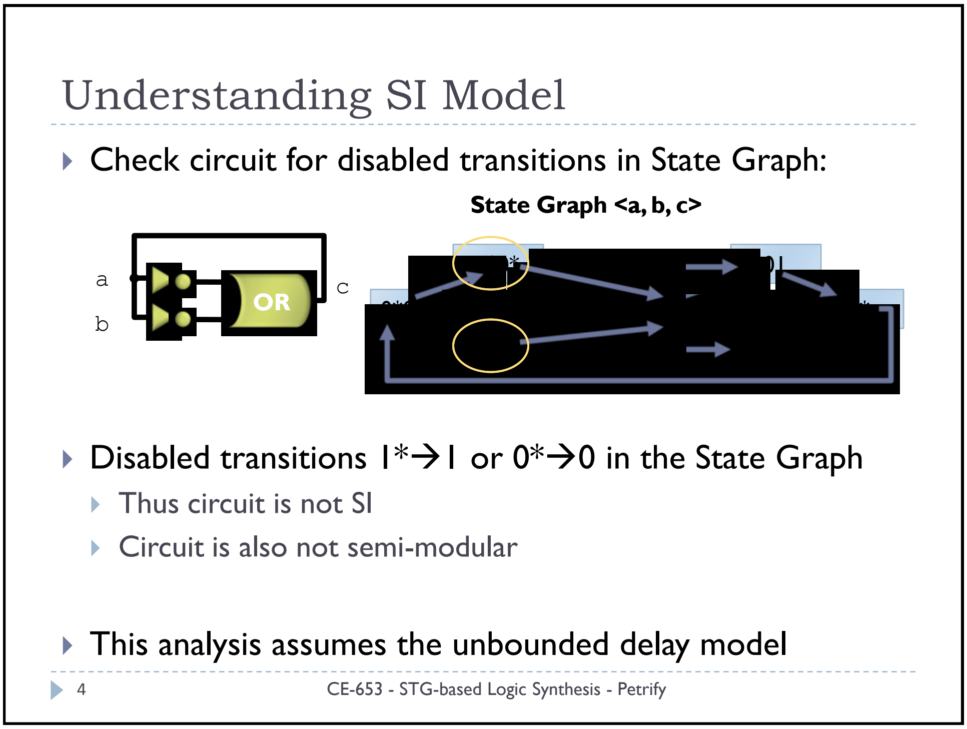


Understanding SI Model

- Check circuit for disabled transitions in State Graph:

State Graph <a, b, c>

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



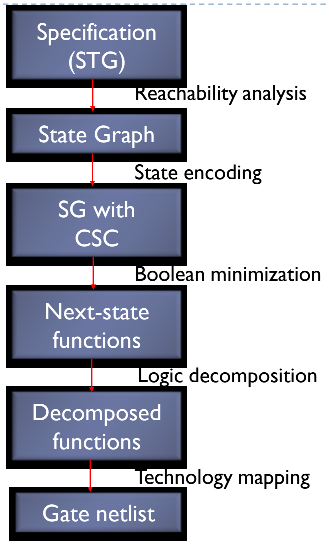
Understanding SI Model

- Check circuit for disabled transitions in State Graph:

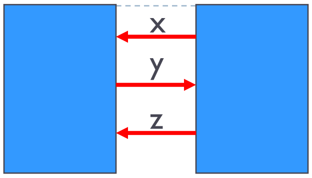
State Graph <a, b, c>

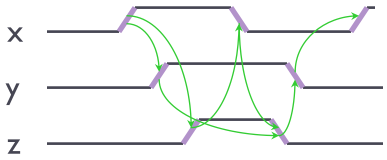
- Disabled transitions I*→I or 0*→0 in the State Graph
  - Thus circuit is not SI
  - Circuit is also not semi-modular
- This analysis assumes the unbounded delay model

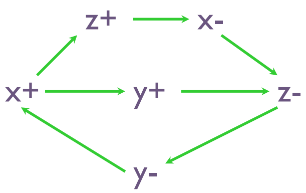
Design flow



Specification

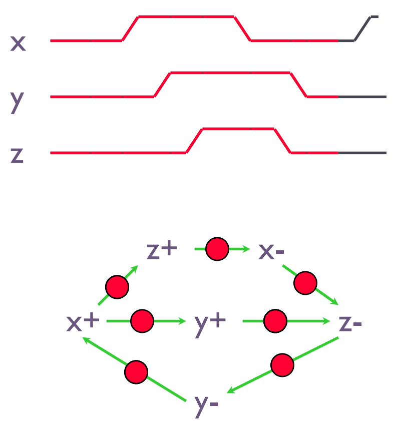




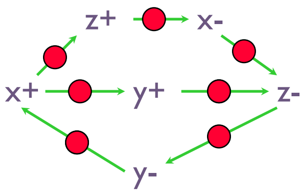


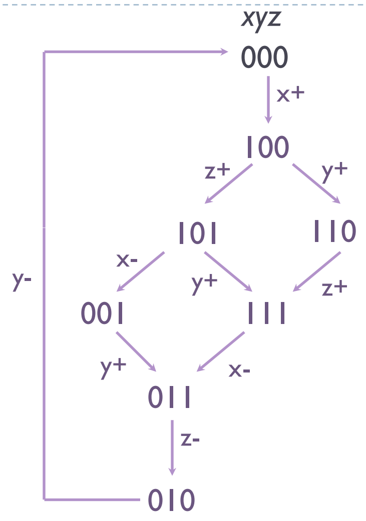
Signal Transition Graph (STG)

Token flow



State graph



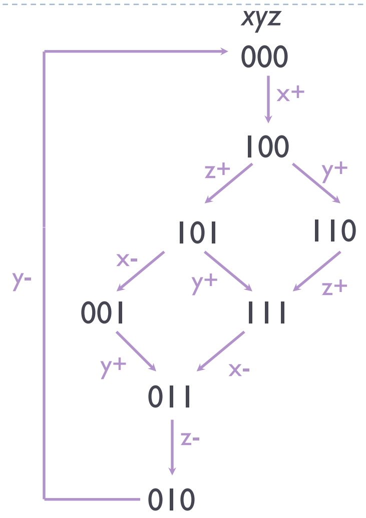


Next-state functions

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

$y = z + x$

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

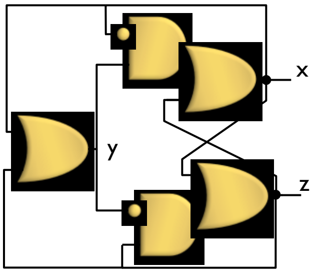


Gate netlist

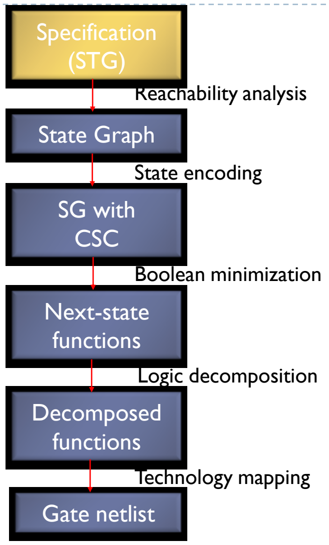
$x = \bar{z} \cdot (x + \bar{y})$

$y = z + x$

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



Design flow



Specification (STG)

Reachability analysis

State Graph

State encoding

SG with CSC

Boolean minimization

Next-state functions

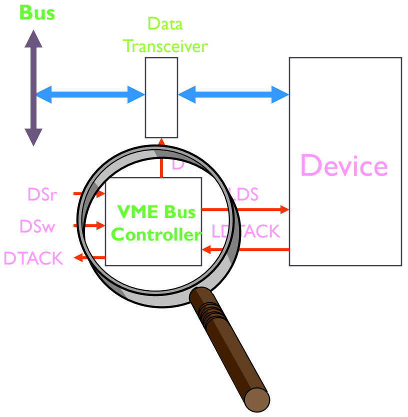
Logic decomposition

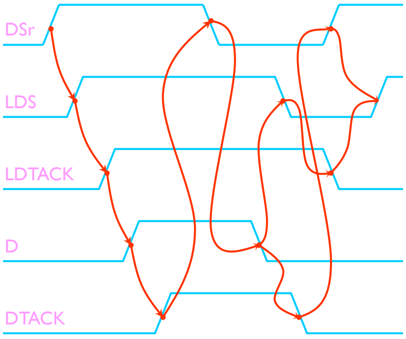
Decomposed functions

Technology mapping

Gate netlist

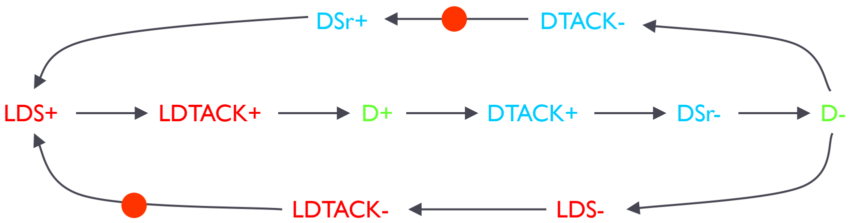
VME Bus Example

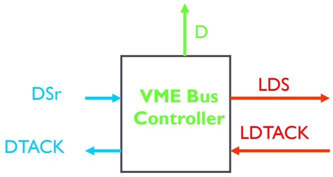




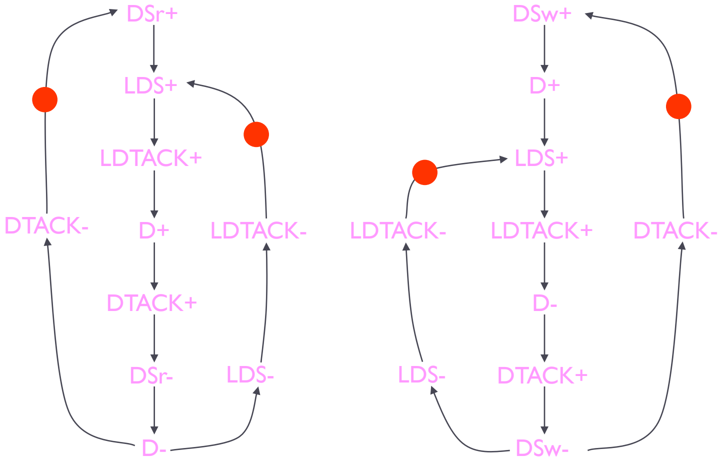
Read Cycle

STG for READs

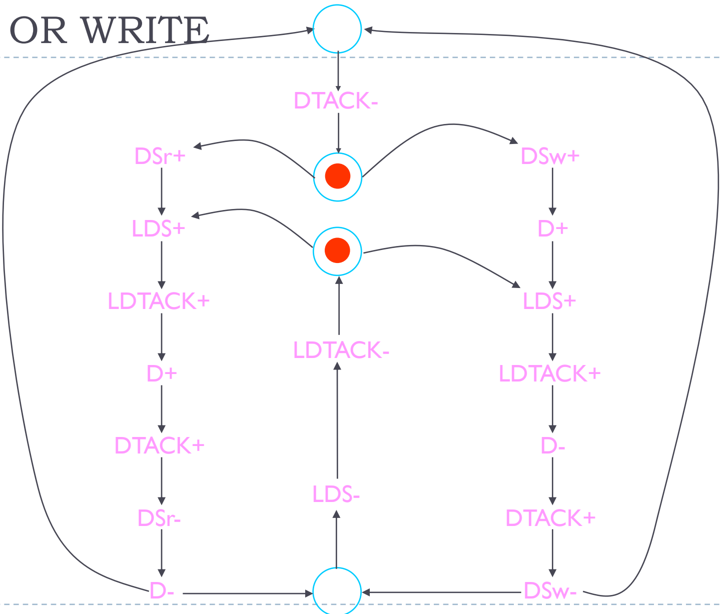




NEED Choice to select between READ OR WRITE

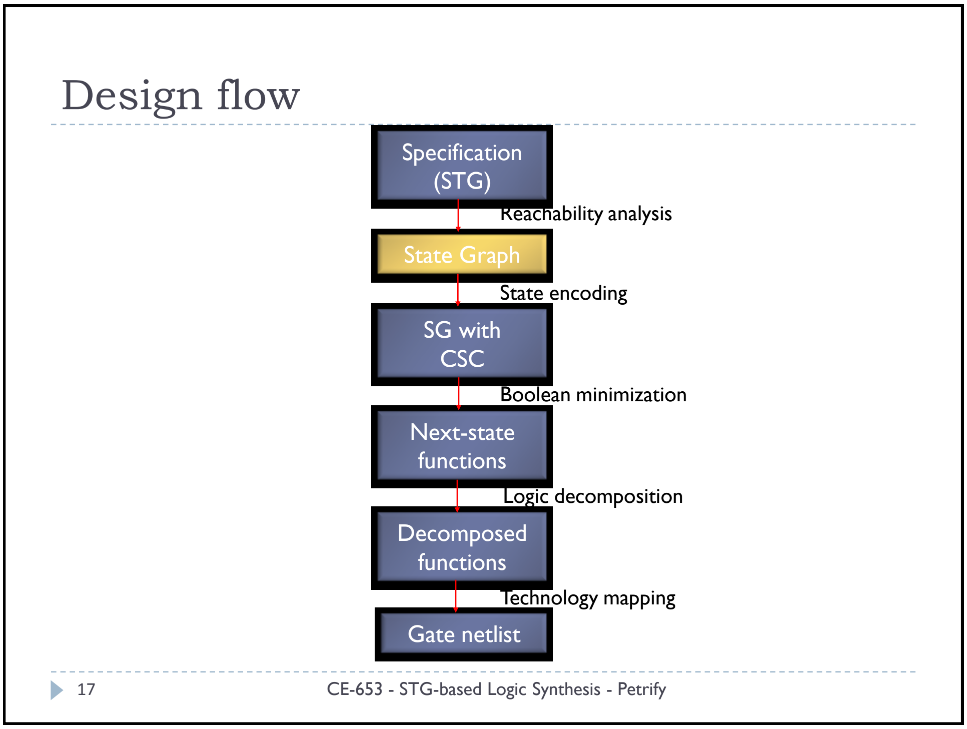


NEED Choice to select between READ OR WRITE



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

Specification (STG)

Reachability analysis

State Graph

State encoding

SG with CSC

Boolean minimization

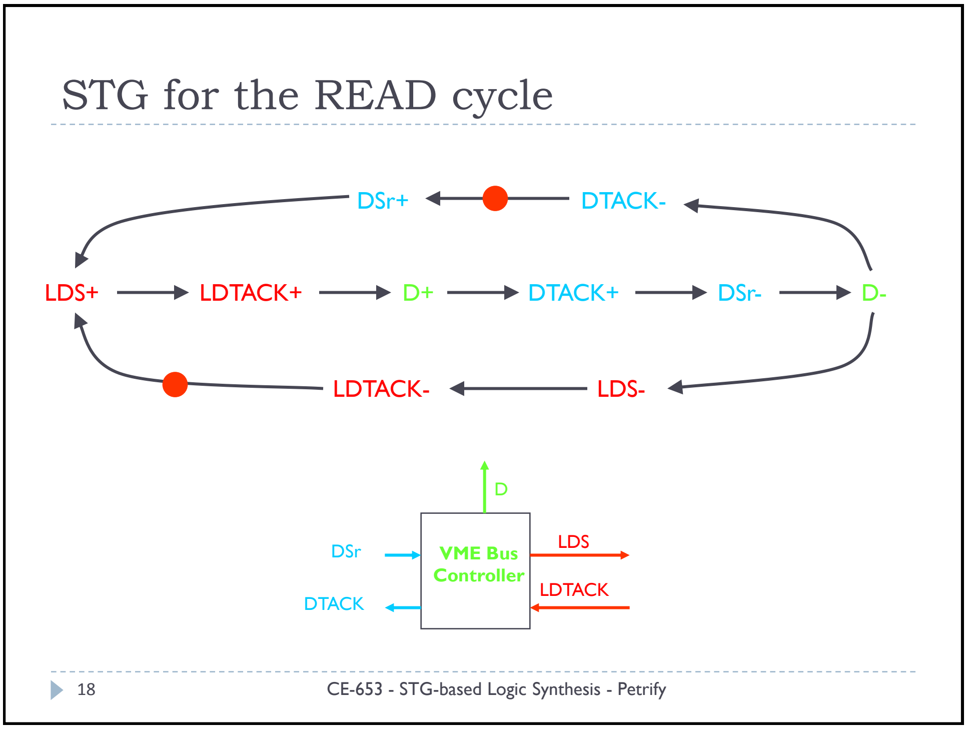
Next-state functions

Logic decomposition

Decomposed functions

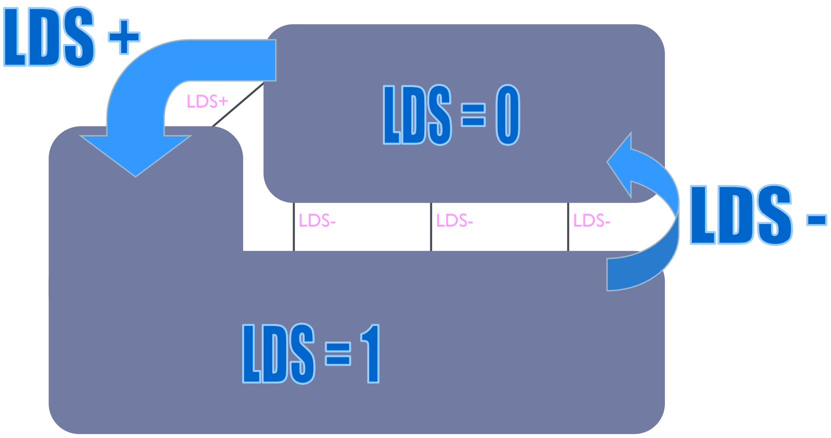
Technology mapping

Gate netlist

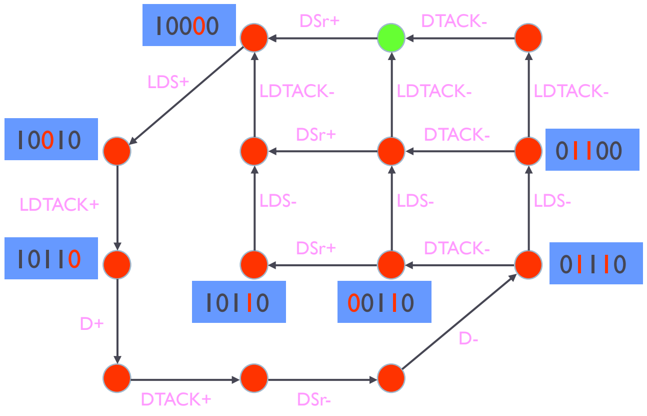


STG for the READ cycle

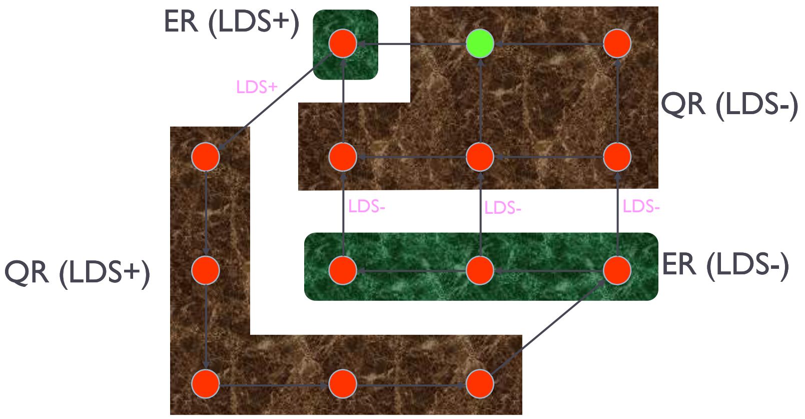
Reachability Graph – Binary Encoding



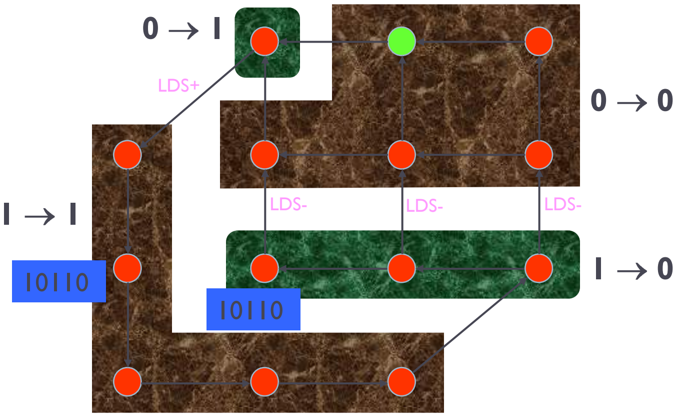
Reachability Graph – Binary Encoding



Defining Excitation and Quiescent Regions



Forming the Next State Function



Extracting the Boolean Expression of the Next State Function

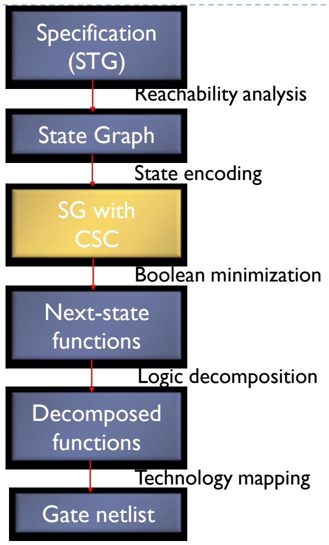
LDS = 0

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

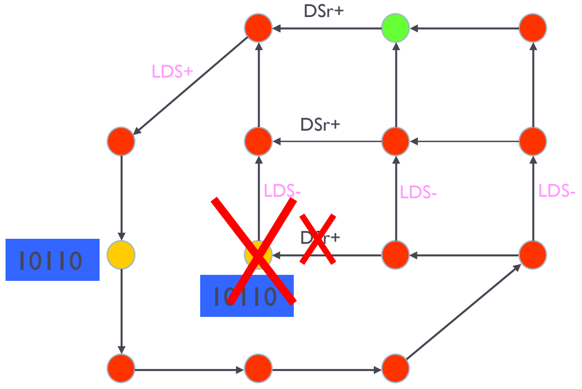
LDS = 1

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

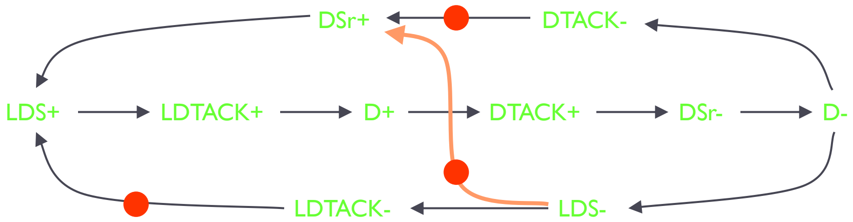
Design flow

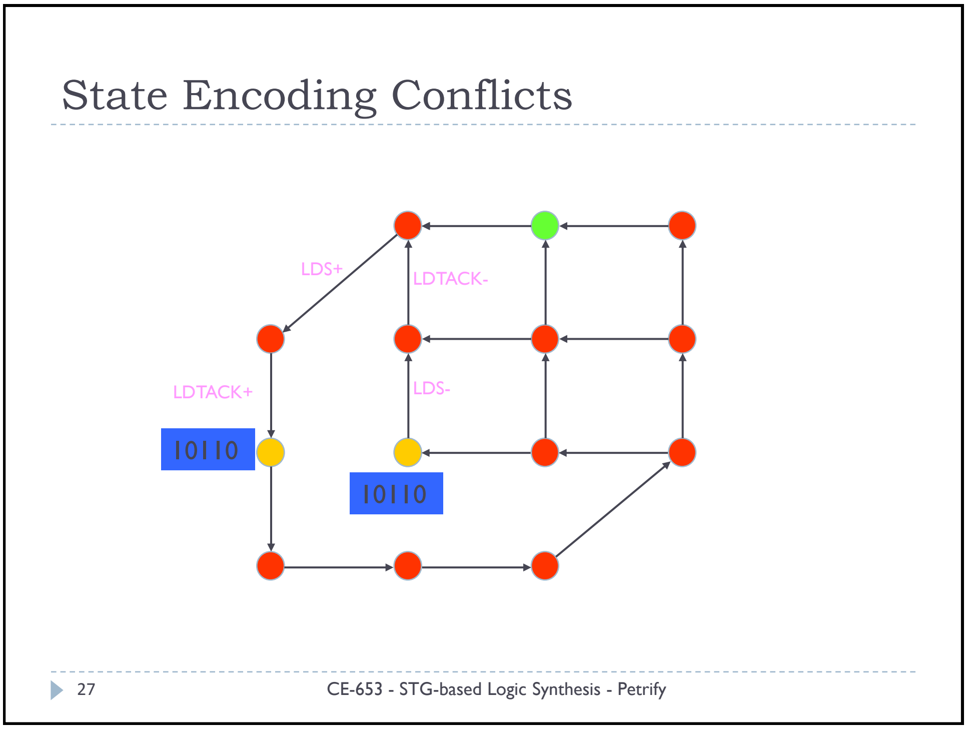


Concurrency Reduction (Manual/Automatic) at State Graph Level



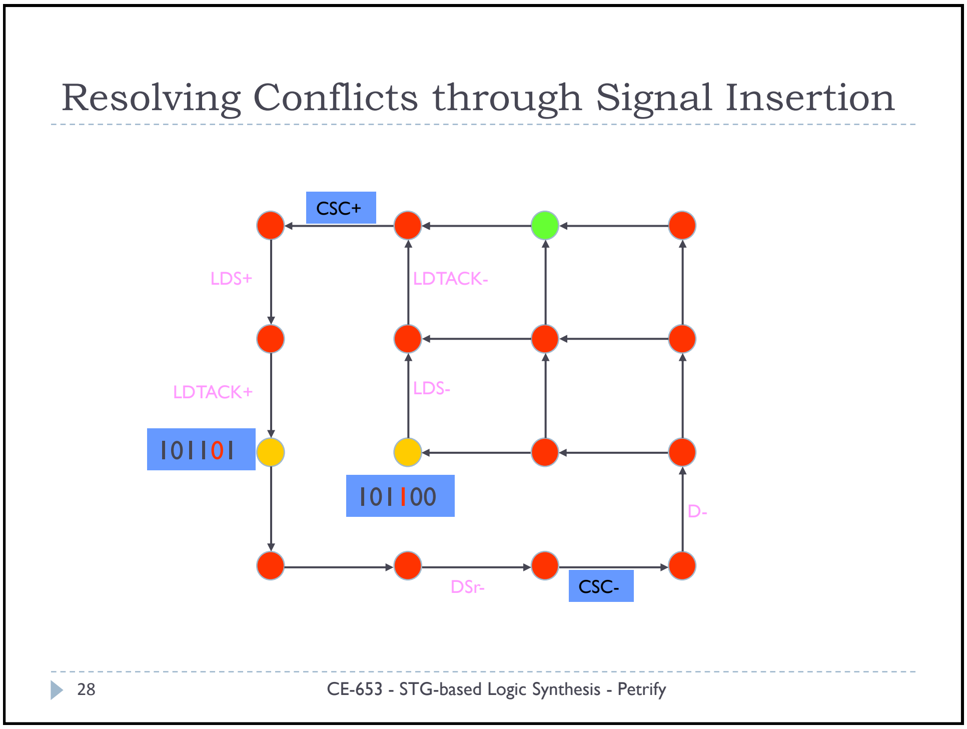
Concurrency Reduction – Migration to STG/PTnet Level





State Encoding Conflicts

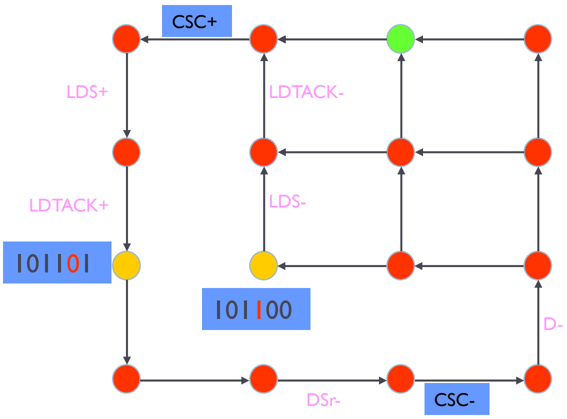
27 CE-653 - STG-based Logic Synthesis - Petrify



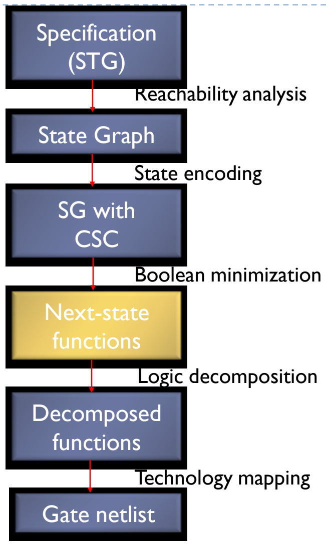
Resolving Conflicts through Signal Insertion

28 CE-653 - STG-based Logic Synthesis - Petrify

Signal Insertion



Design flow


Specification (STG)

Reachability analysis

State Graph

State encoding

SG with CSC

Boolean minimization

Next-state functions

Logic decomposition

Decomposed functions

Technology mapping

Gate netlist

Complex-Gate Implementation

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

$$DTACK = D$$

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

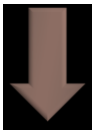
$$\mathrm{csc} = DSr \cdot (\mathrm{csc} + \overline{LDTACK})$$

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)

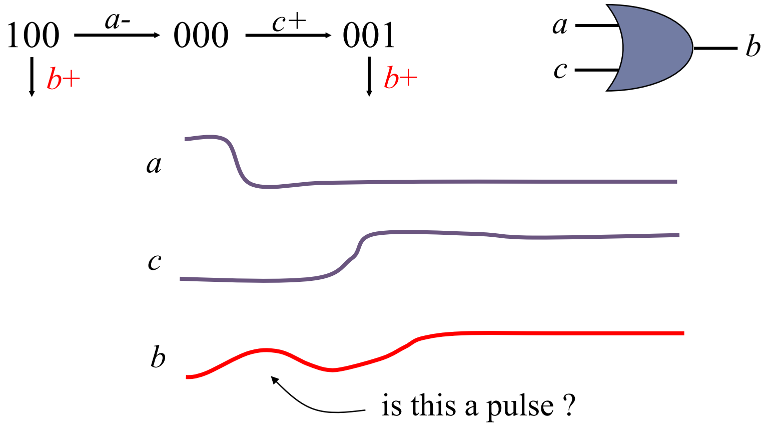
Implementability Conditions - Revisited

- Consistency + CSC + persistency

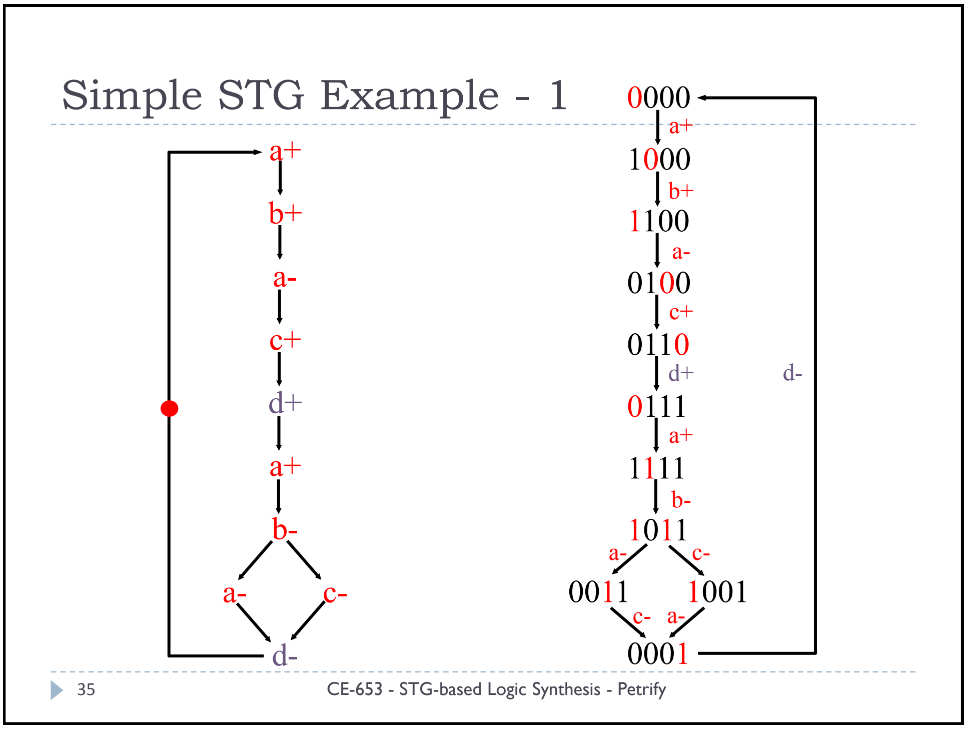


- There exists a speed-independent circuit that implements the behavior of the STG
  - under the assumption that any Boolean function can be implemented with one complex gate

Understanding Persistency

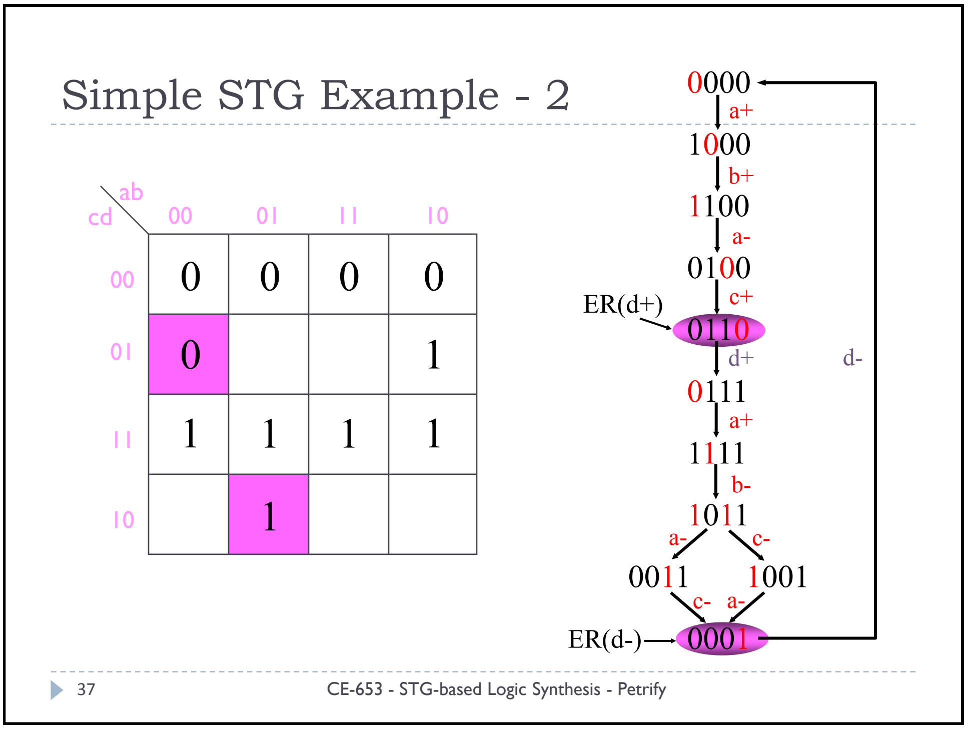


Speed independence ⇒ glitch-free output behavior under any delay



Simple STG Example - 1

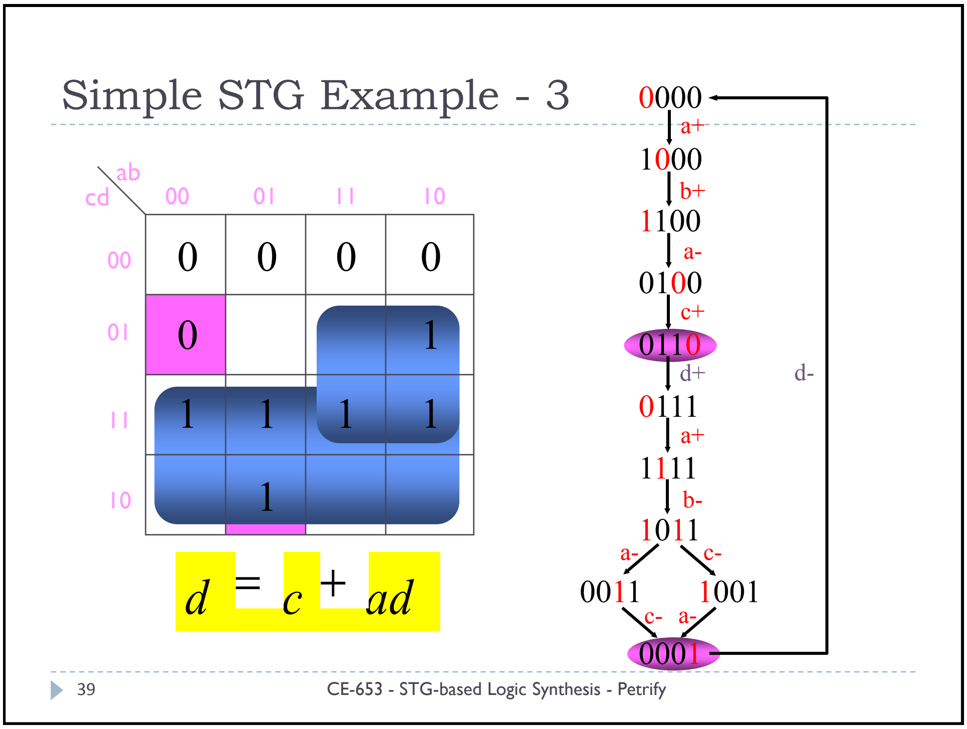
35 CE-653 - STG-based Logic Synthesis - Petrify



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		

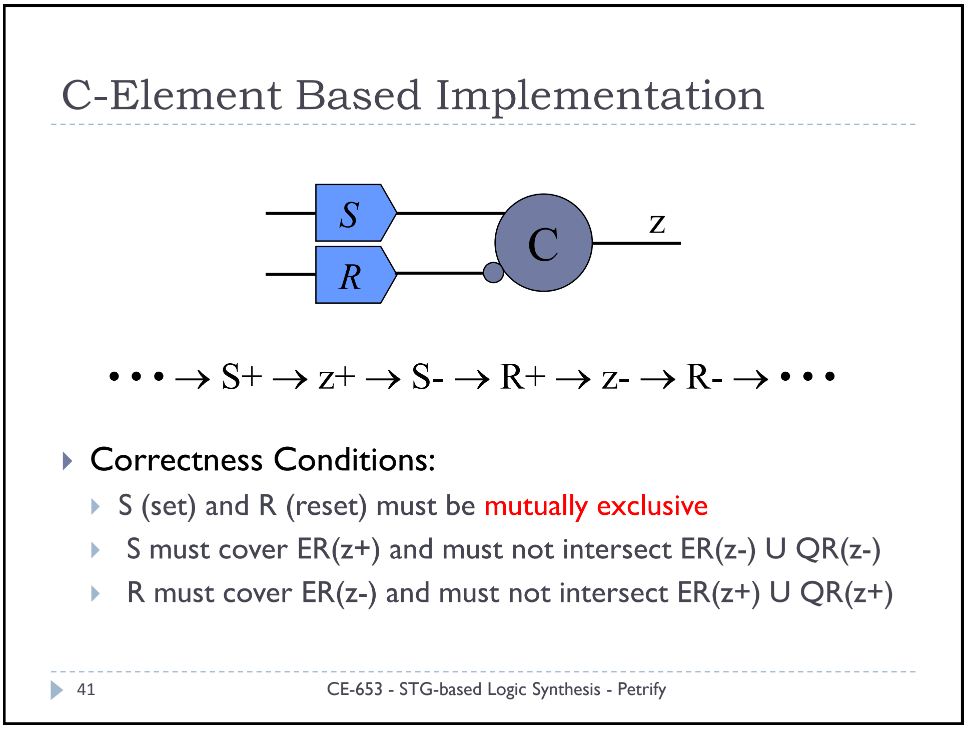
37 CE-653 - STG-based Logic Synthesis - Petrify



Simple STG Example - 3

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

$d = c + ad$



C-Element Based Implementation

• • • → S+ → z+ → S- → R+ → z- → R- → • • •

- Correctness Conditions:
  - S (set) and R (reset) must be mutually exclusive
  - S must cover ER(z+) and must not intersect ER(z-) U QR(z-)
  - R must cover ER(z-) and must not intersect ER(z+) U QR(z+)

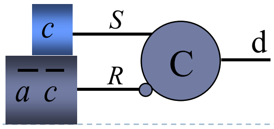
Monotonic Covers

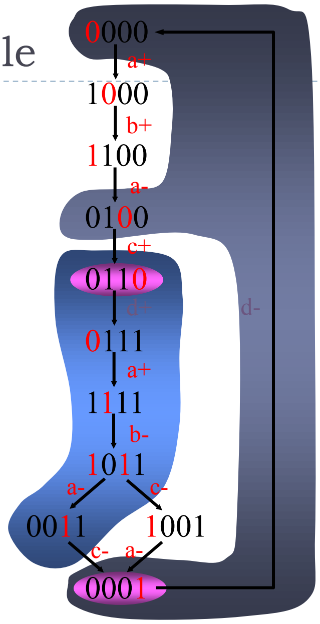
- Definition[Monotonic Cover]
  - Cover Cube C is a monotonic cover for ER(a*) iff:
    - C covers all states ER(a*)
    - C covers no states outside ER(a*) U QR(a*)
    - C changes only once inside QR(a*)

- A Monotonic Cover ensures SI implementation using simple gates

C-Element Based Example

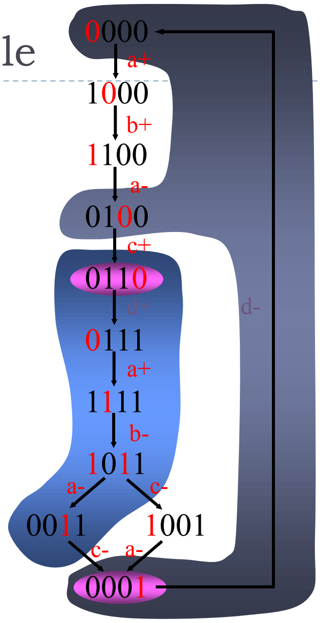
cd \ ab	00	01	11	10
00	0	0	0	0
01	0			1
11	1	1	1	1
10		1		

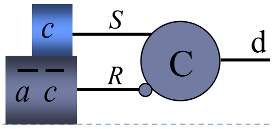




C-Element Based Example

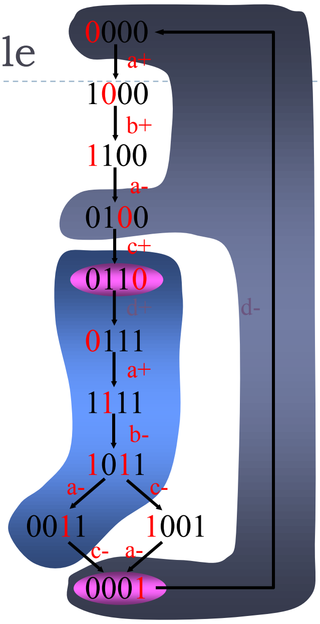
- If the Reset R = a′c′ has an unbounded delay
- Then, starting from state 0000:
  - a+;R-;b+;a-;c+;S+;d+;
  - The a-, c+ transition can cause a hazard at the Reset logic

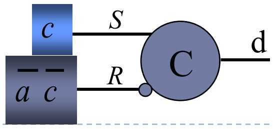


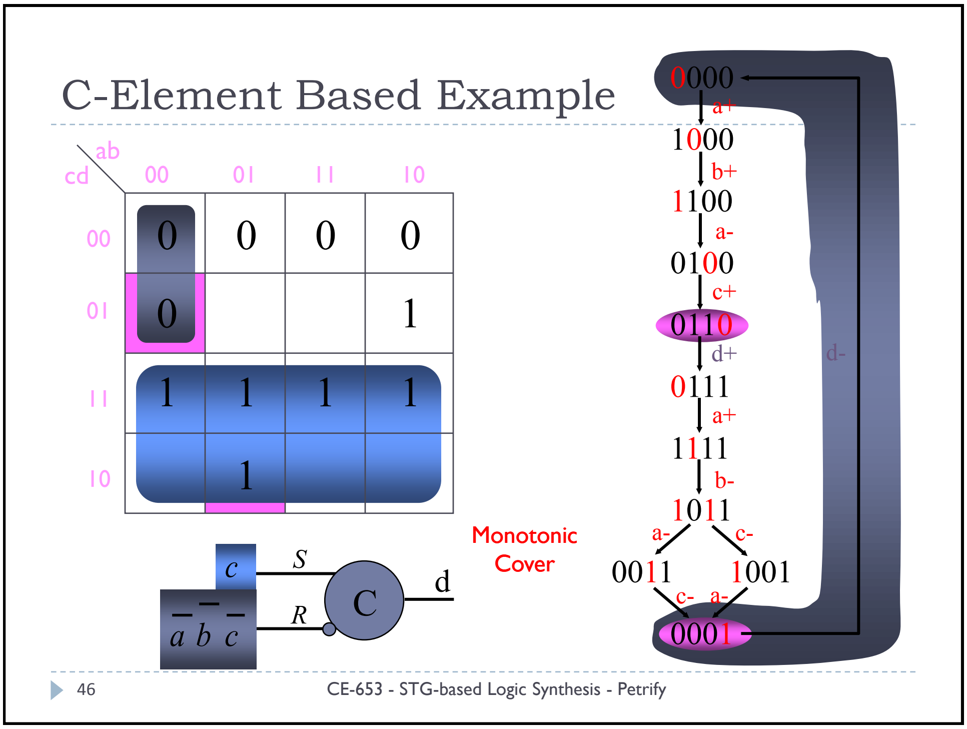


C-Element Based Example

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





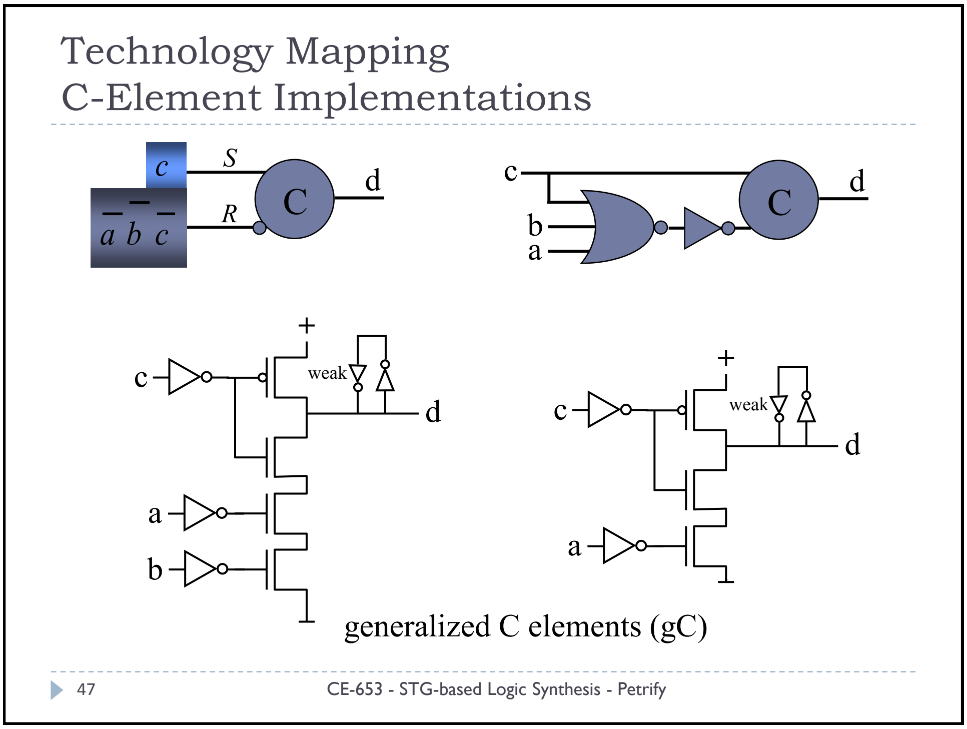


C-Element Based Example

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

Monotonic Cover

46 CE-653 - STG-based Logic Synthesis - Petrify



Technology Mapping C-Element Implementations

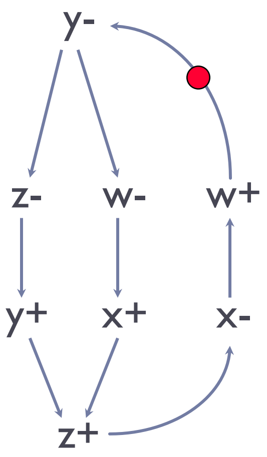
generalized C elements (gC)

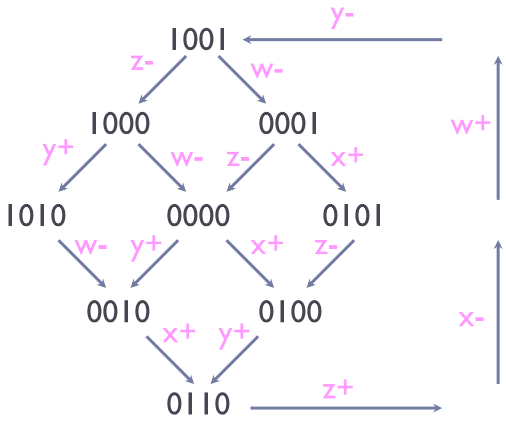
47 CE-653 - STG-based Logic Synthesis - Petrify

Speed-Independence - Summary

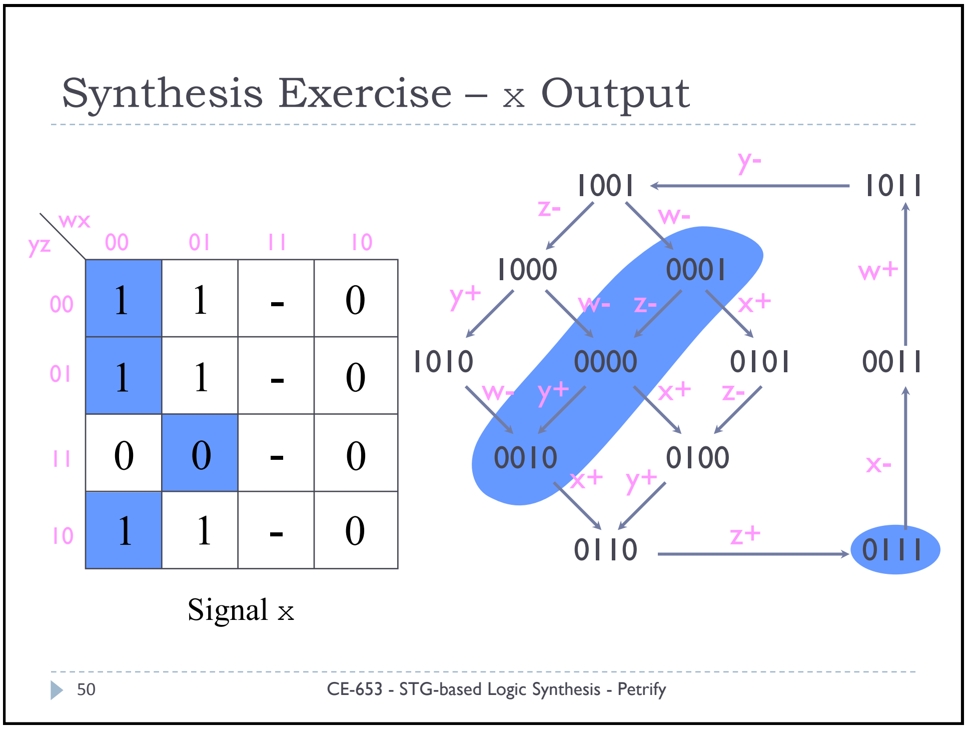
- Implementability conditions
  - Consistency
  - Complete State Coding (CSC)
  - Persistency
- Circuit architectures
  - Complex (hazard-free) gates
  - C elements with monotonic covers

Synthesis Exercise





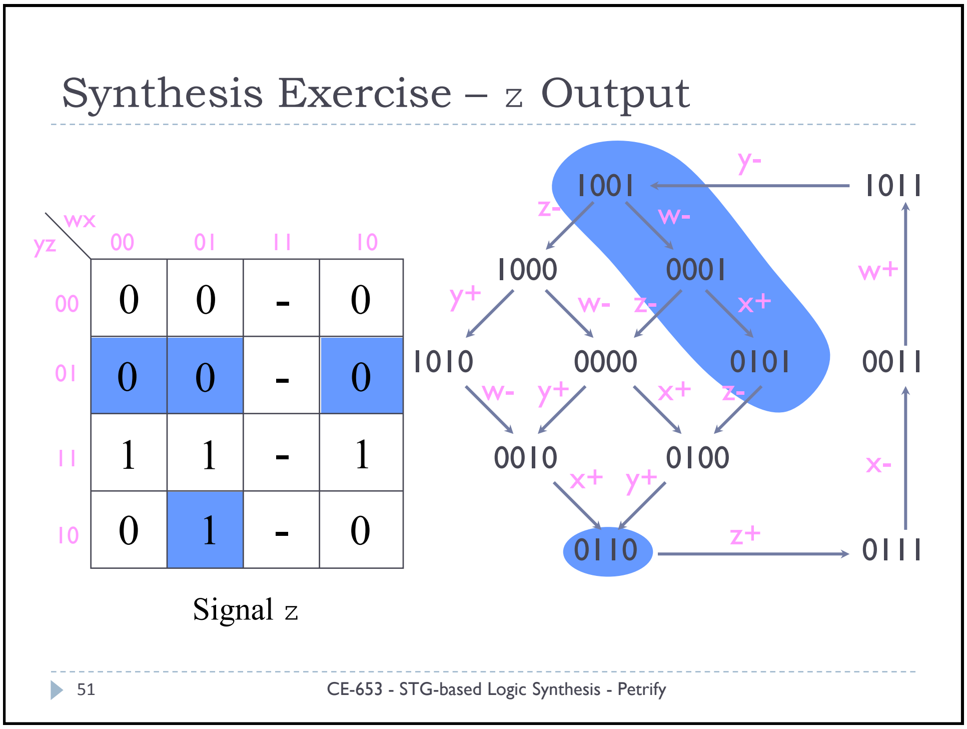
- Derive circuits for outputs x and z
  - Both complex gate and C-element based implementations



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

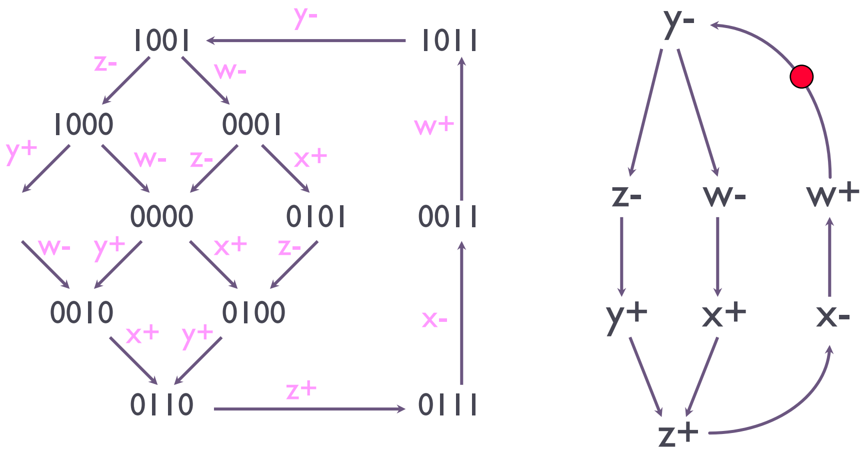


Synthesis Exercise – z Output

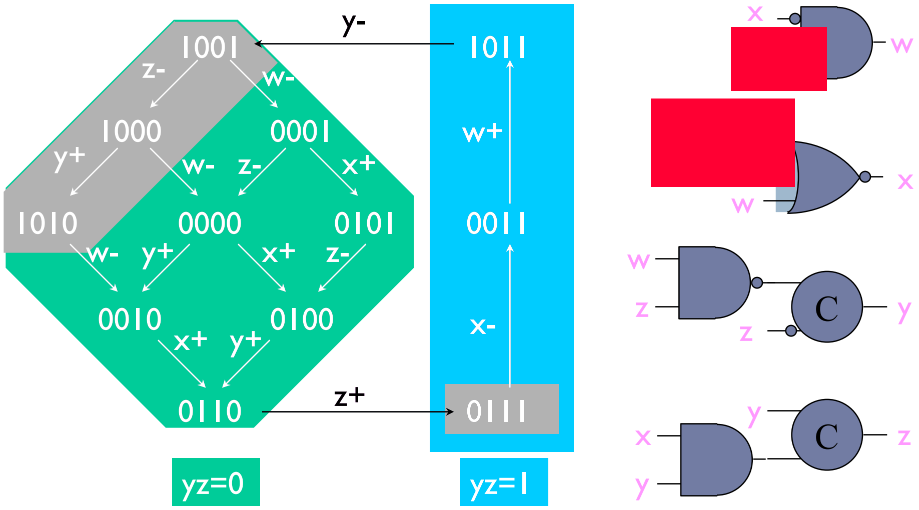
yz \ wx	00	01	11	10
00	0	0	-	0
01	0	0	-	0
11	1	1	-	1
10	0	1	-	0

Signal z

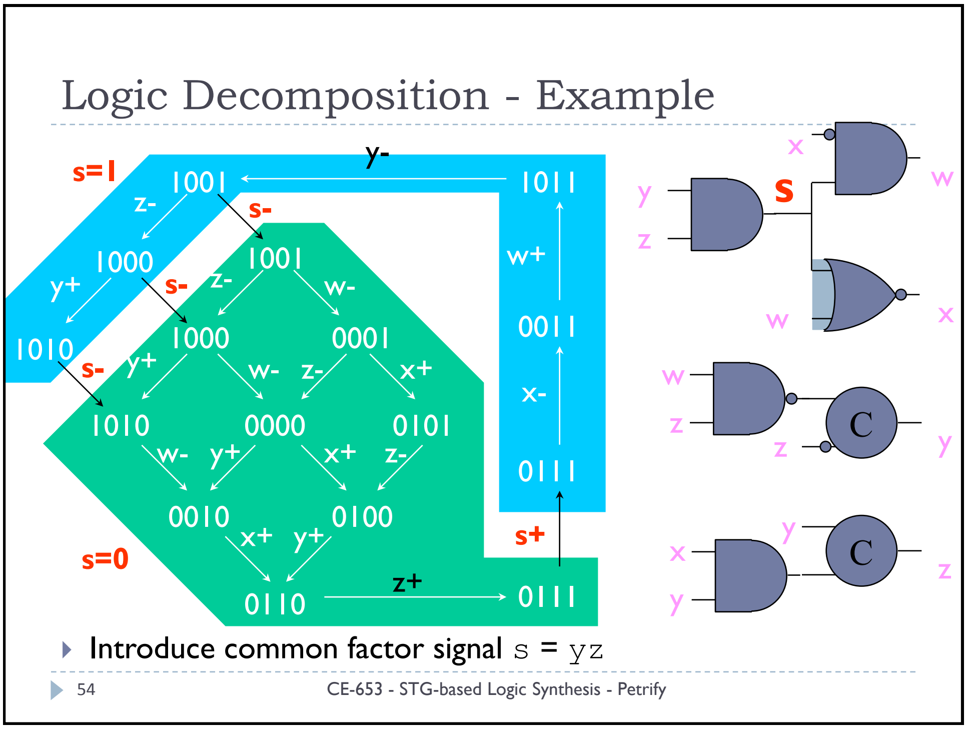
Logic Decomposition - Example



Logic Decomposition - Example

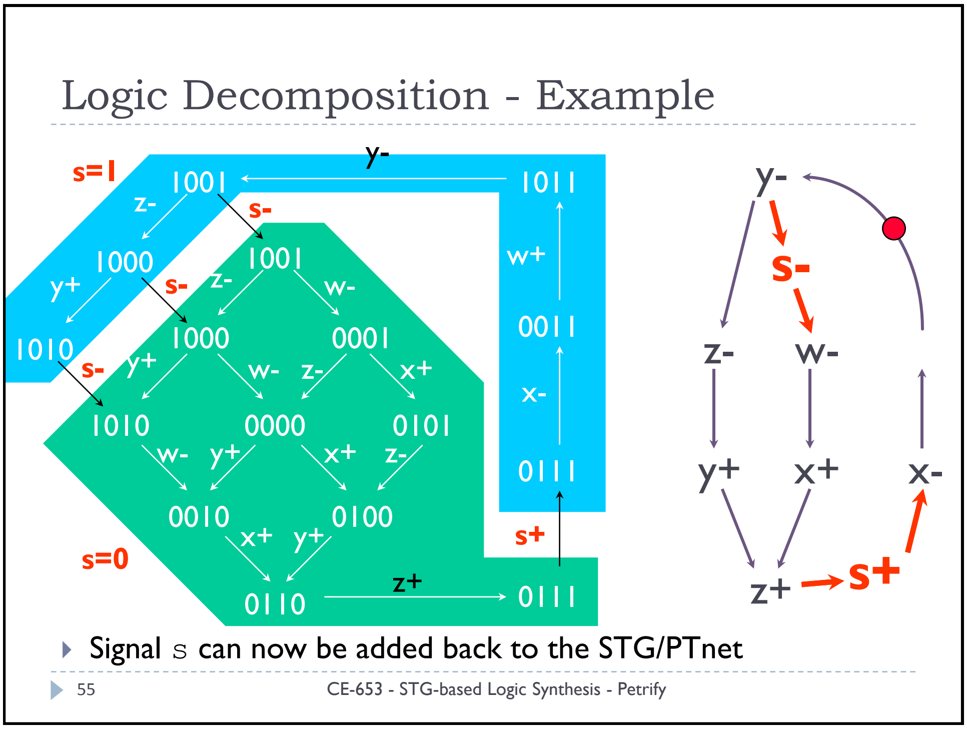


- Can we decompose yz into an independent AND gate?



Logic Decomposition - Example

- Introduce common factor signal s = yz



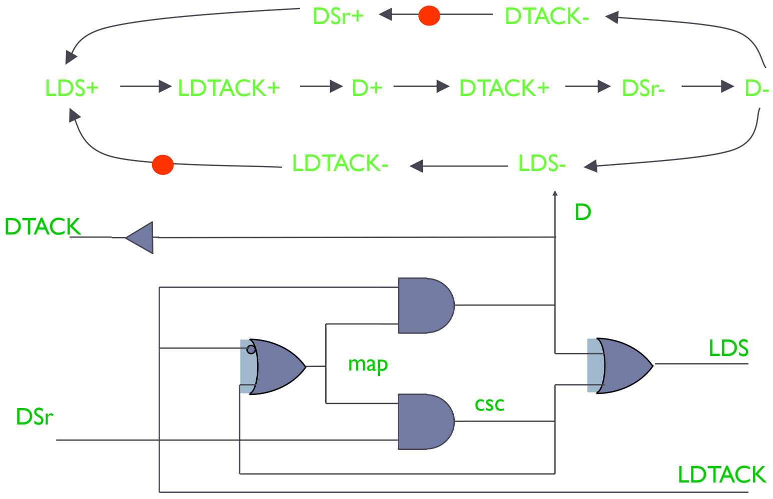
Logic Decomposition - Example

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

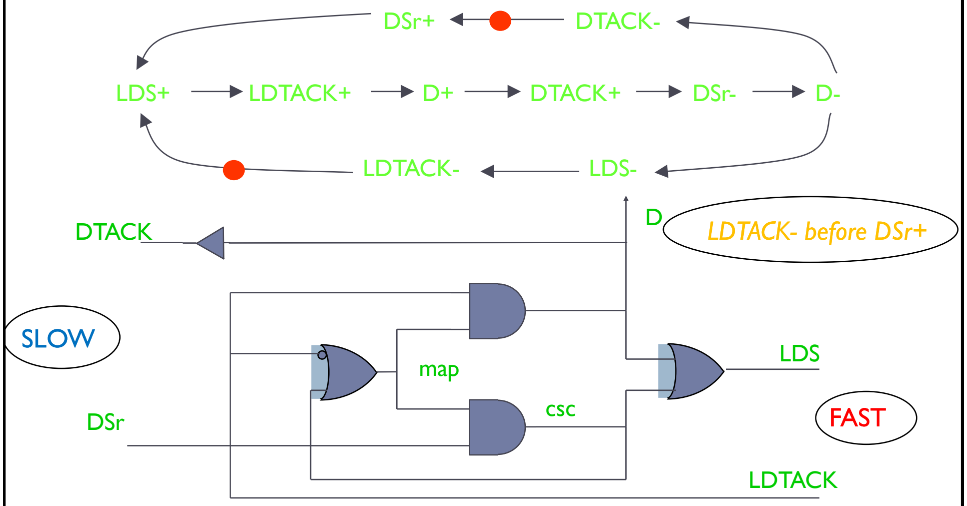
Timing Assumptions

- 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

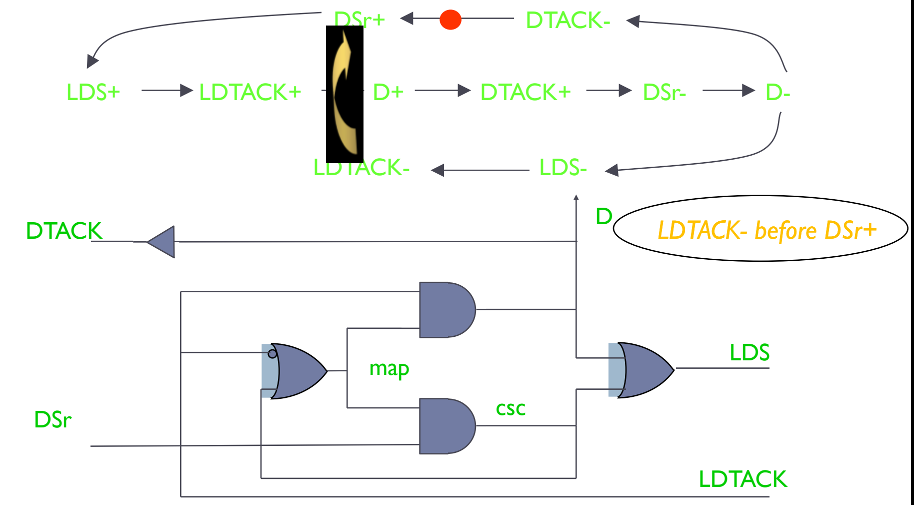


LDTACK- before DSr+

SLOW

FAST

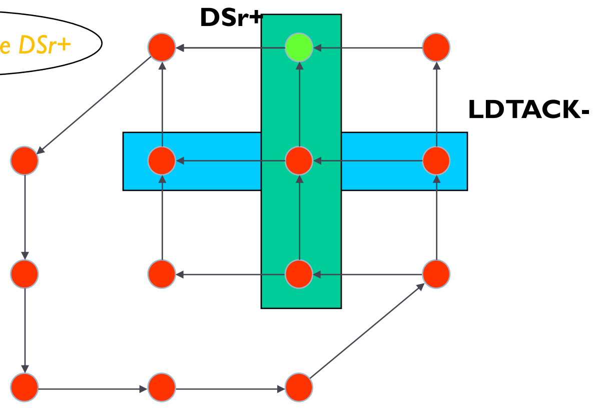
Timing Assumptions Example – SI Netlist – Adding Timing Assumptions



LDTACK- before DSr+

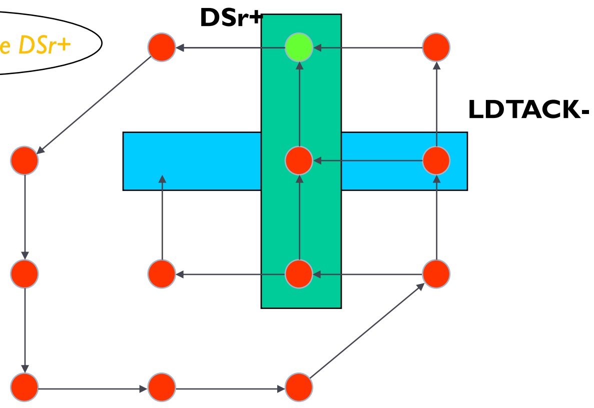
Timing Assumptions Example – SI Netlist – Adding Timing Assumptions – State Graph

LDTACK- before DSr+



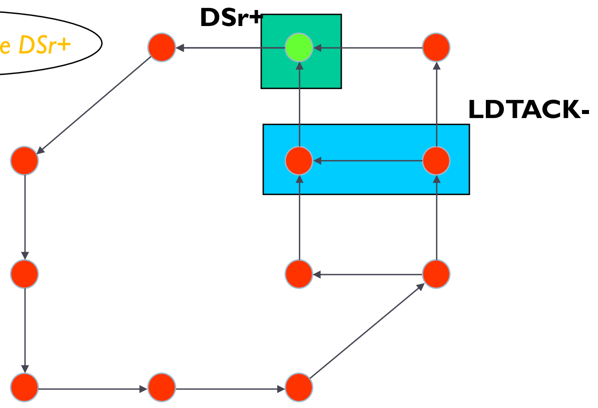
Timing Assumptions Example – SI Netlist – Adding Timing Assumptions – State Graph

LDTACK- before DSr+



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

LDTACK- before DSr+



Two more unreachable states

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

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?

- Original Circuit had CSC issue!!!

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

LDS = 0

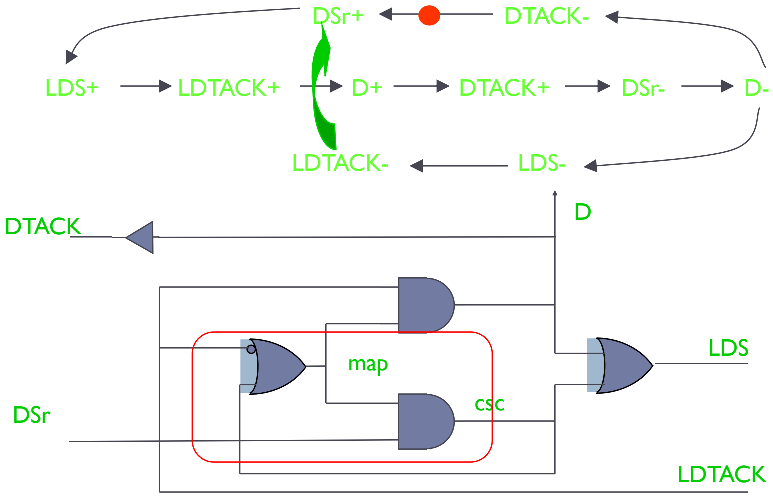
D LDTACK \ DTACK DSr	00	01	11	10
00	0	0	-	1
01	-	-	-	-
11	-	-	-	-
10	0	0	-	-

LDS = 1

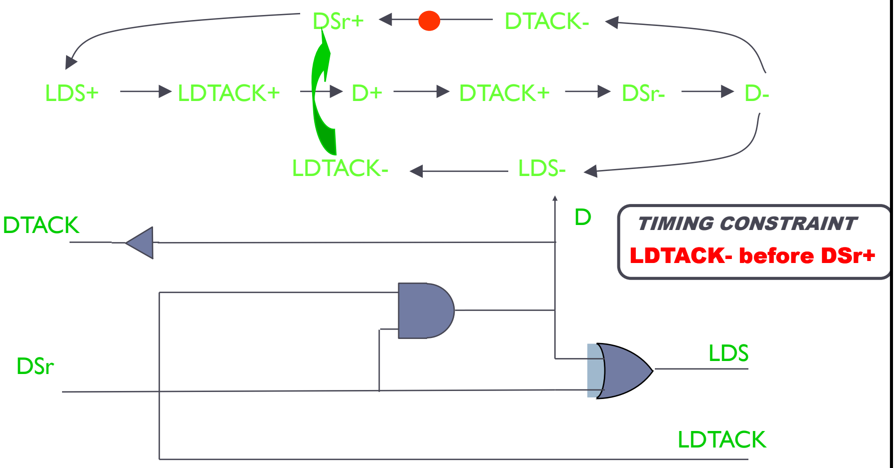
D LDTACK \ DTACK DSr	00	01	11	10
00	-	-	-	1
01	-	-	-	-
11	-	1	1	1
10	0	0	-	1

- Timing assumptions add DC and resolve CSC!!!

Timing Assumptions Example – SI Netlist with Timing Constraint



Timing Assumptions Example – SI Netlist with Timing Constraint



TIMING CONSTRAINT

LDTACK- before DSr+

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