Logic synthesis from concurrent specifications

Jordi Cortadella

Universitat Politecnica de Catalunya
Barcelona, Spain

In collaboration with M. Kishinevsky,
A. Kondratyev, L. Lavagno and A. Yakovlev

Outline

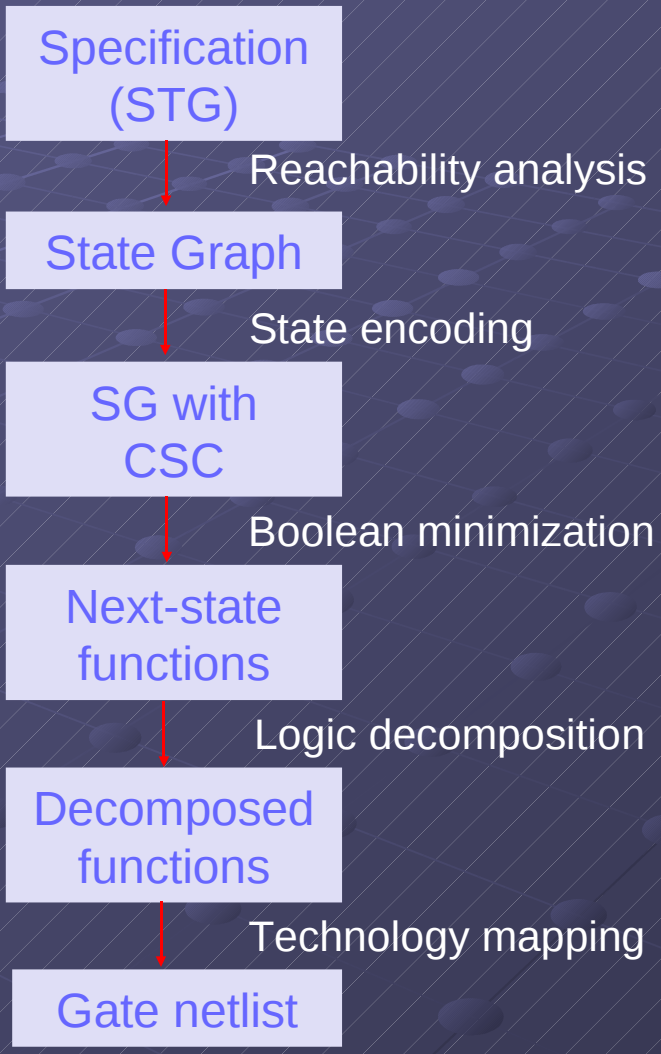
- Overview of the synthesis flow
- Specification
- State graph and next-state functions
- State encoding
- Implementability conditions
- Speed-independent circuit
  - Complex gates
  - C-element architecture
- Review of some advanced topics

Book and synthesis tool

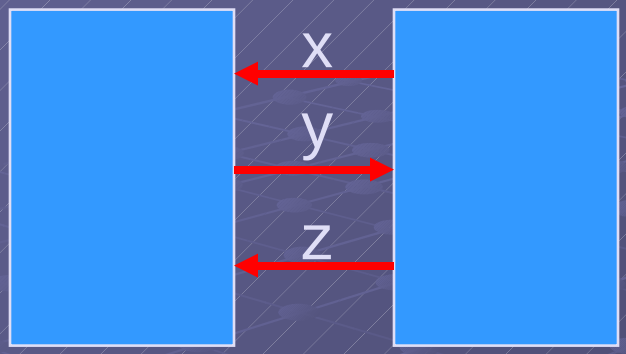
- J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, Logic synthesis for asynchronous controllers and interfaces, Springer-Verlag, 2002

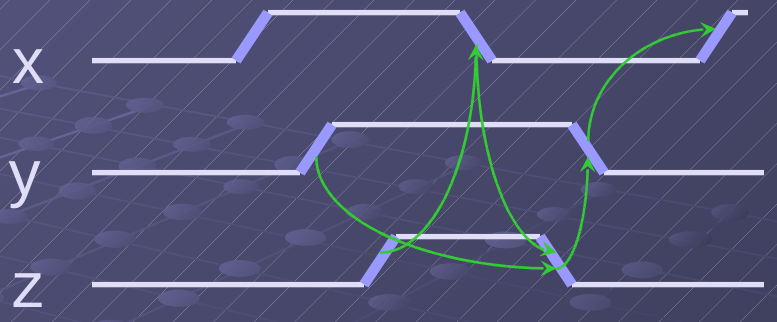
- petrify: http://www.lsi.upc.es/petrify

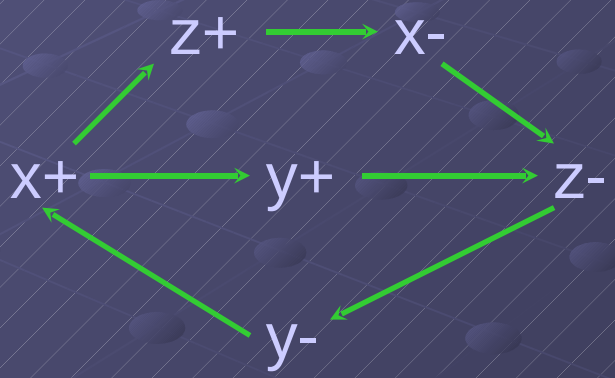
Design flow


Specification (STG)
Reachability analysis
State Graph
State encoding
SG with CSC
Boolean minimization
Next-state functions
Logic decomposition
Decomposed functions
Technology mapping
Gate netlist

Specification

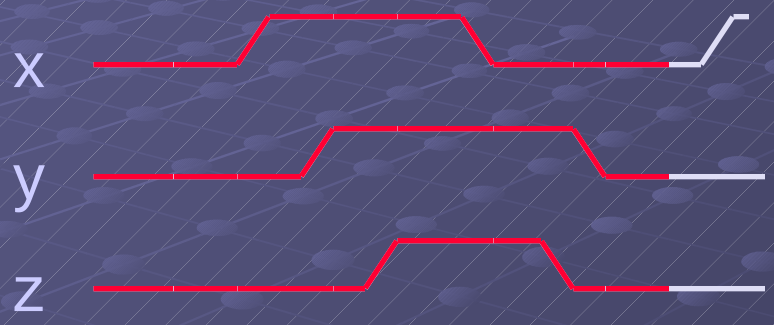


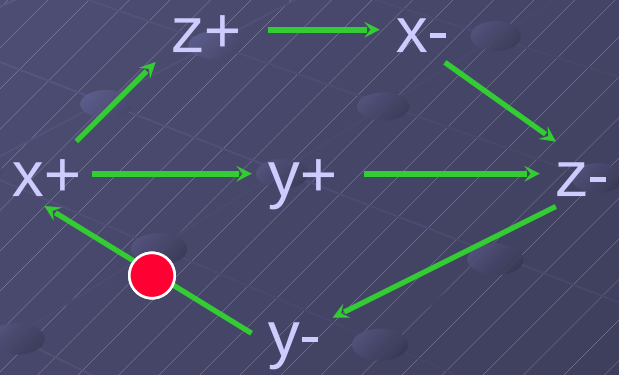




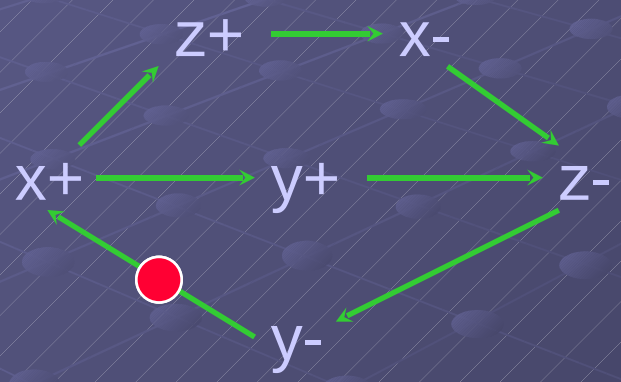
Signal Transition Graph (STG)

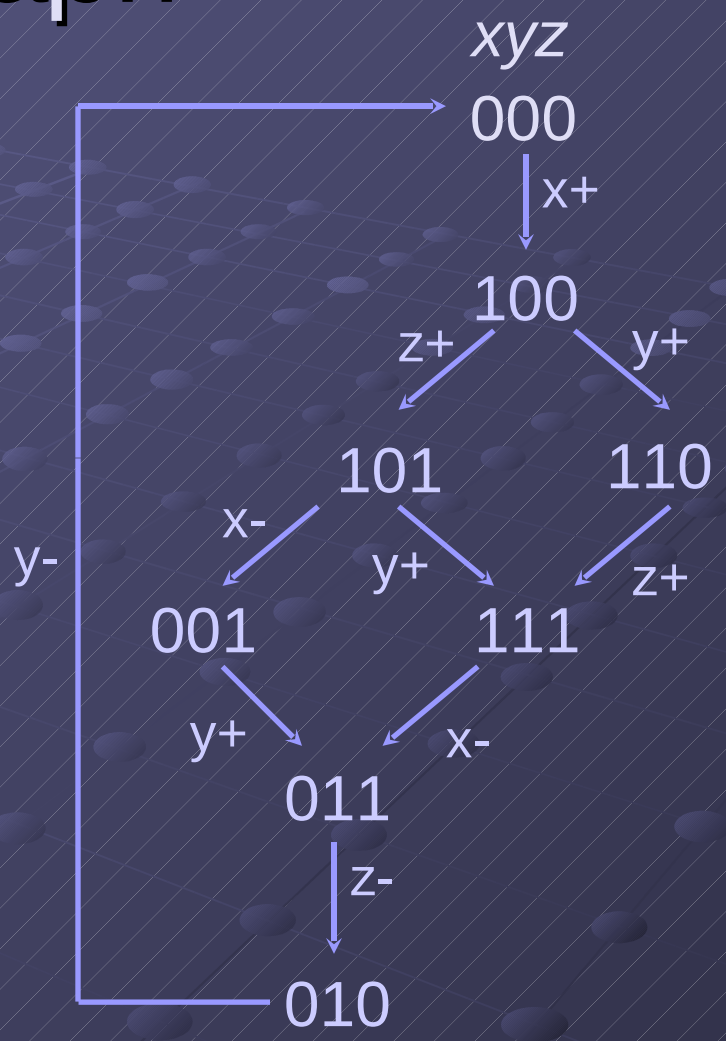
Token flow





State graph



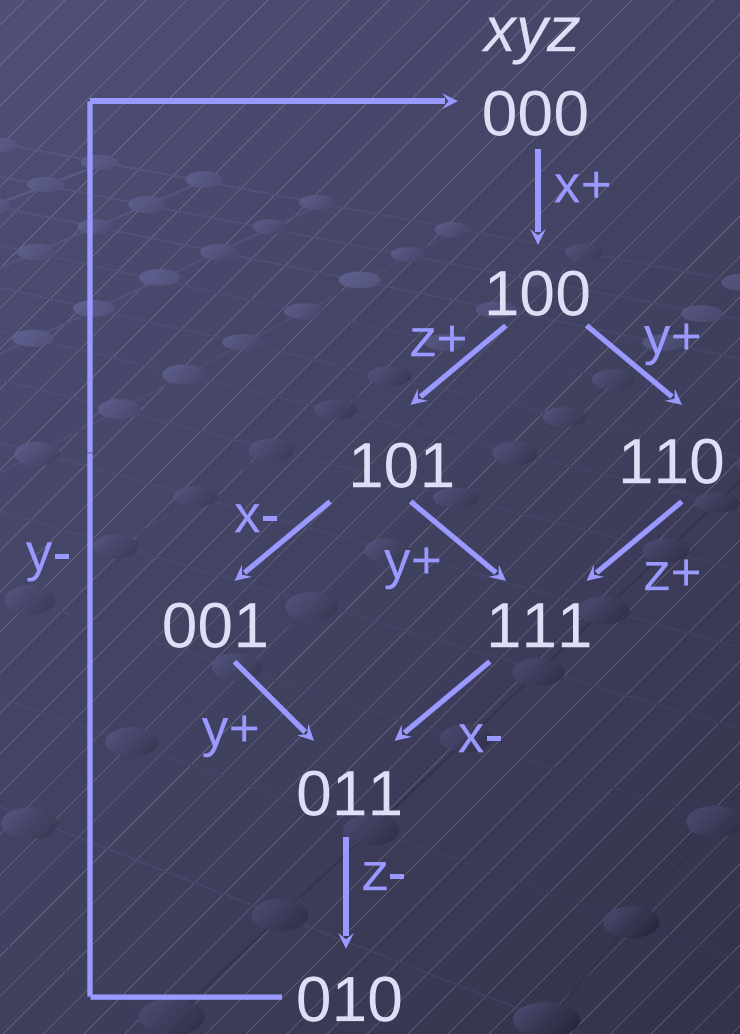


Next-state functions

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

$$y = z + x$$

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

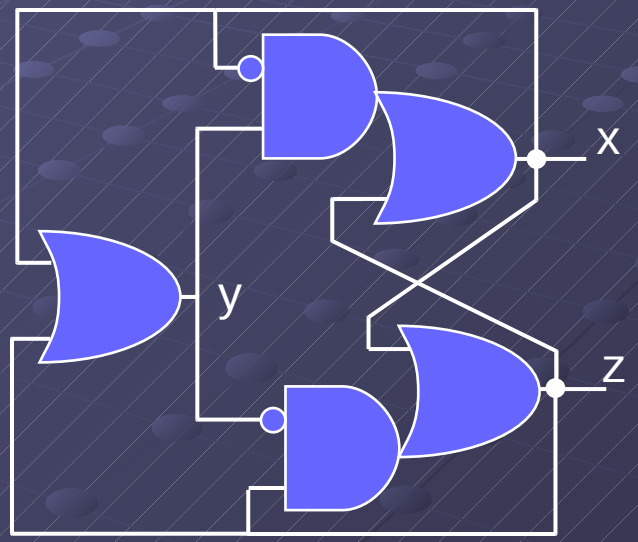


Gate netlist

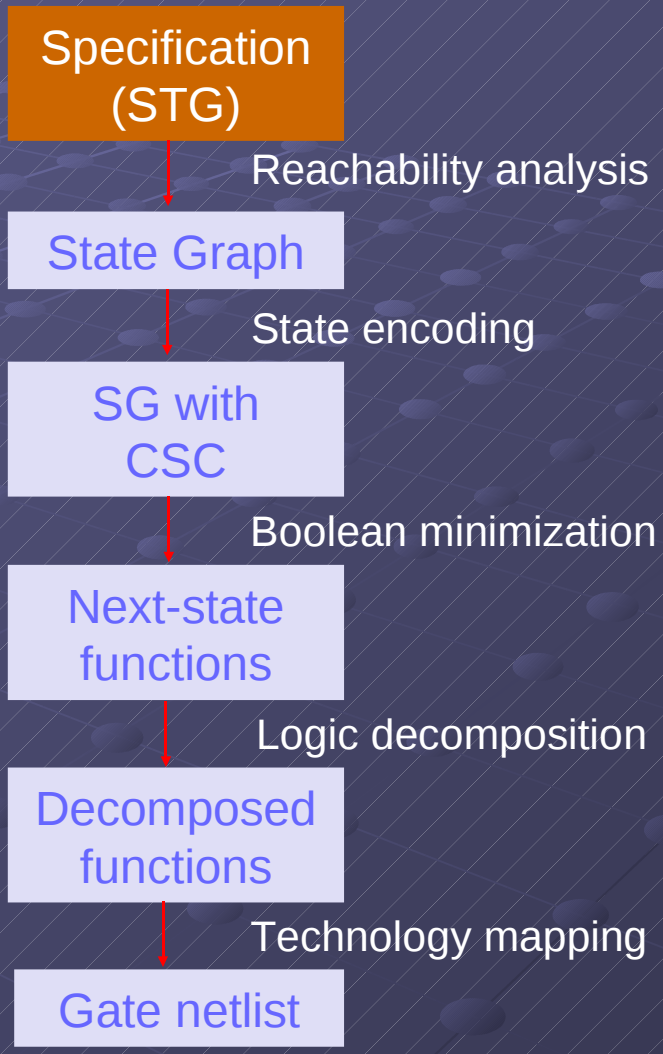
$$x = \bar{z} \cdot (x + \bar{y})$$

$$y = z + x$$

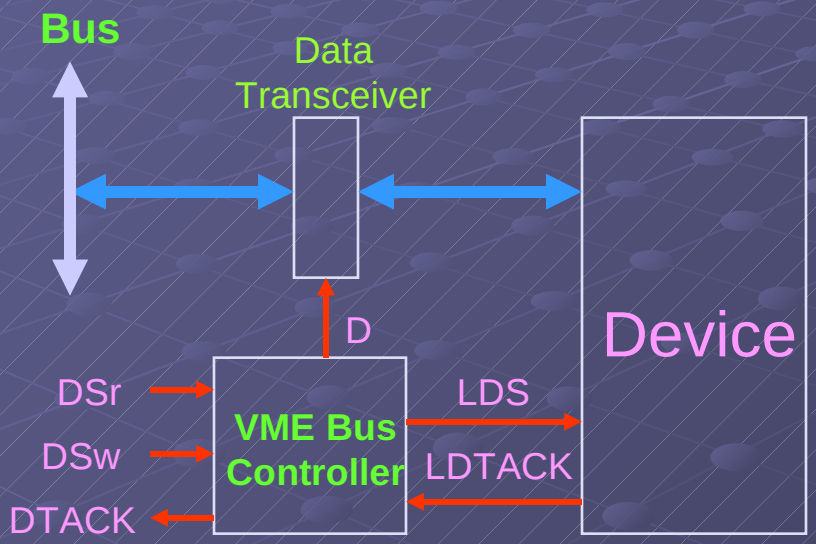
$$z = x + \bar{y} \cdot z$$

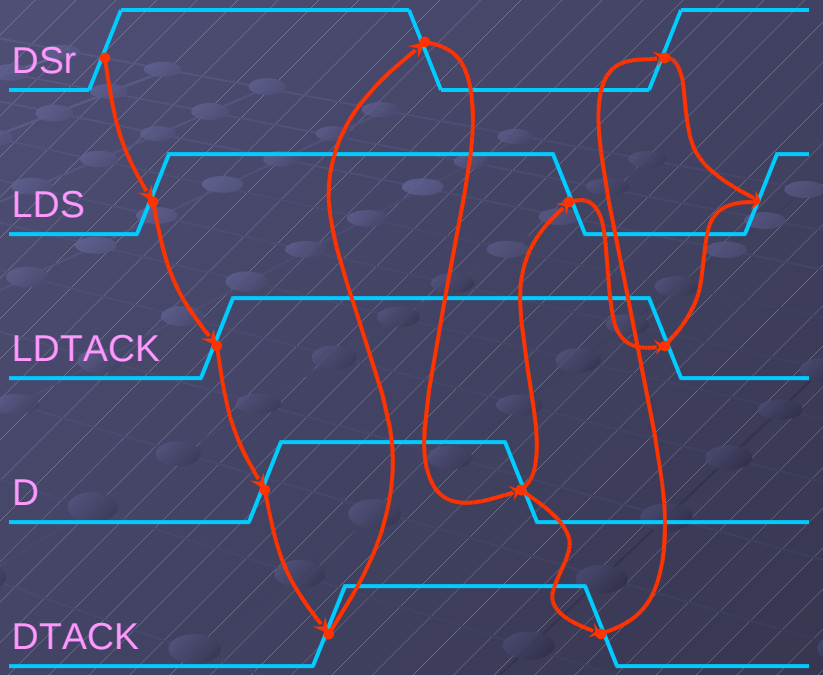


Design flow


Specification (STG)
Reachability analysis
State Graph
State encoding
SG with CSC
Boolean minimization
Next-state functions
Logic decomposition
Decomposed functions
Technology mapping
Gate netlist

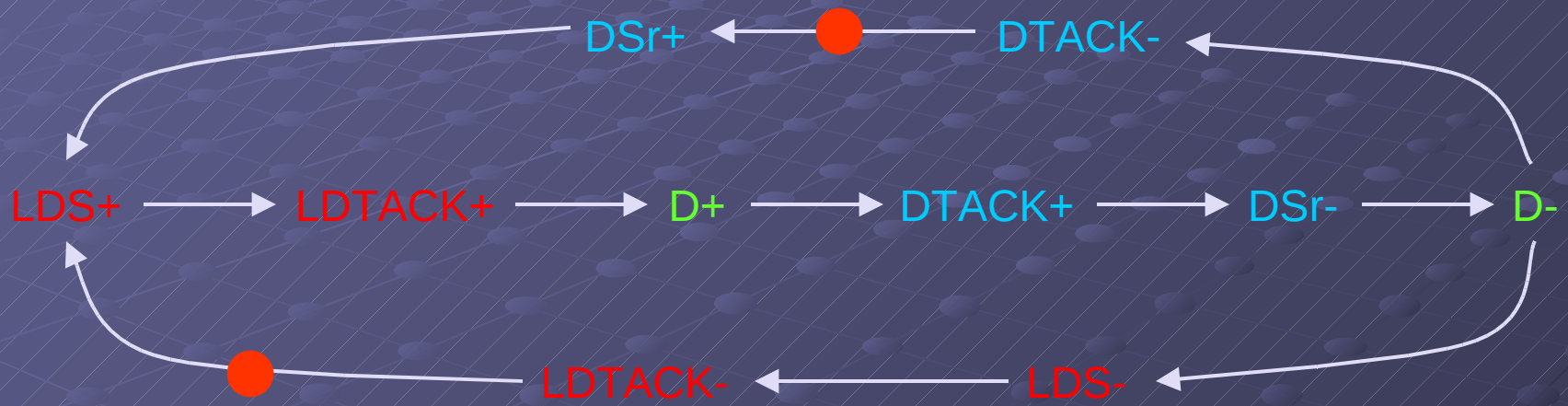
VME bus

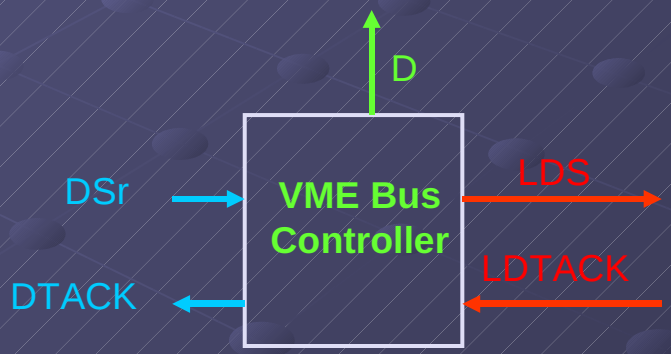




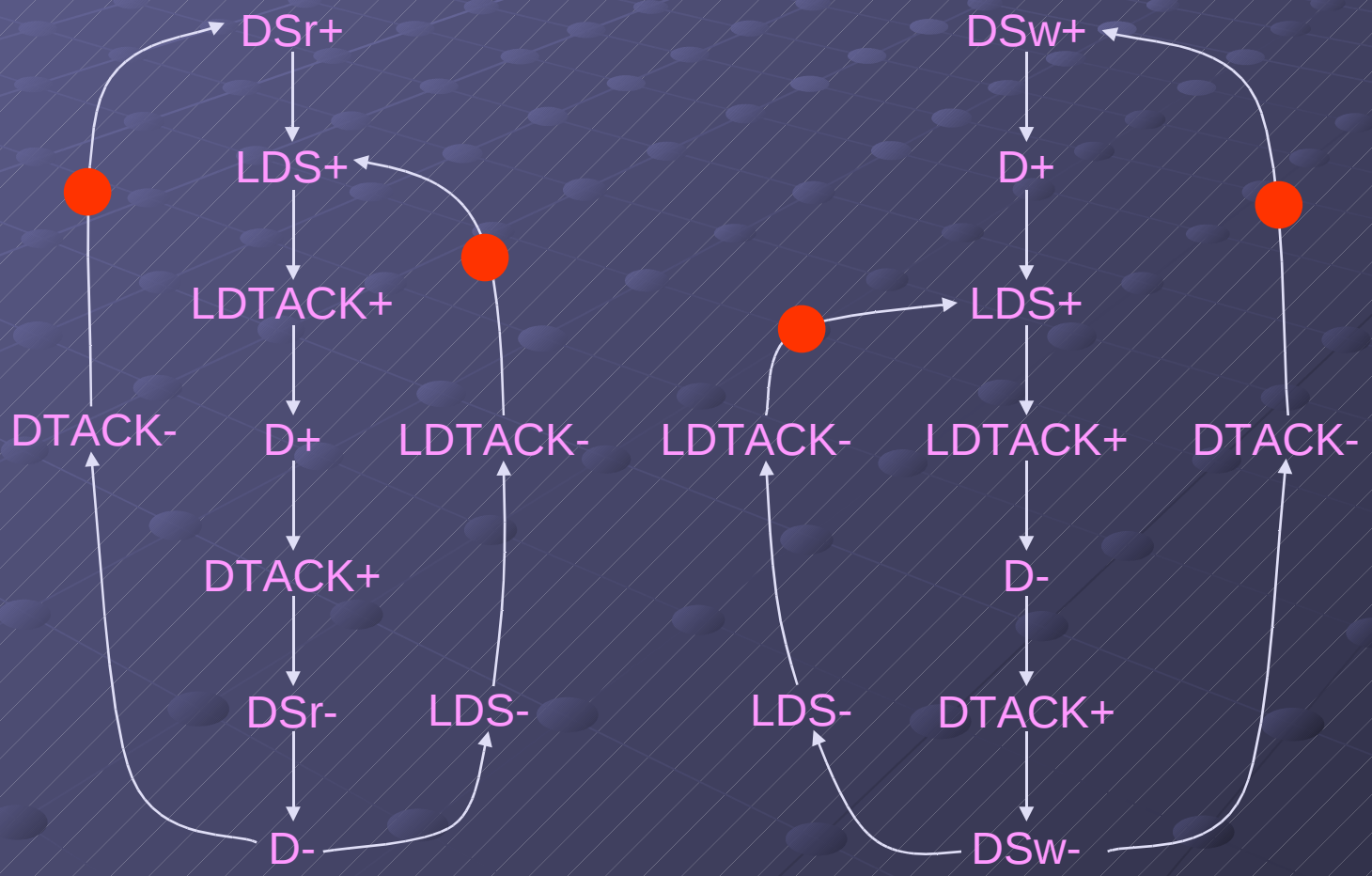
Read Cycle

STG for the READ cycle

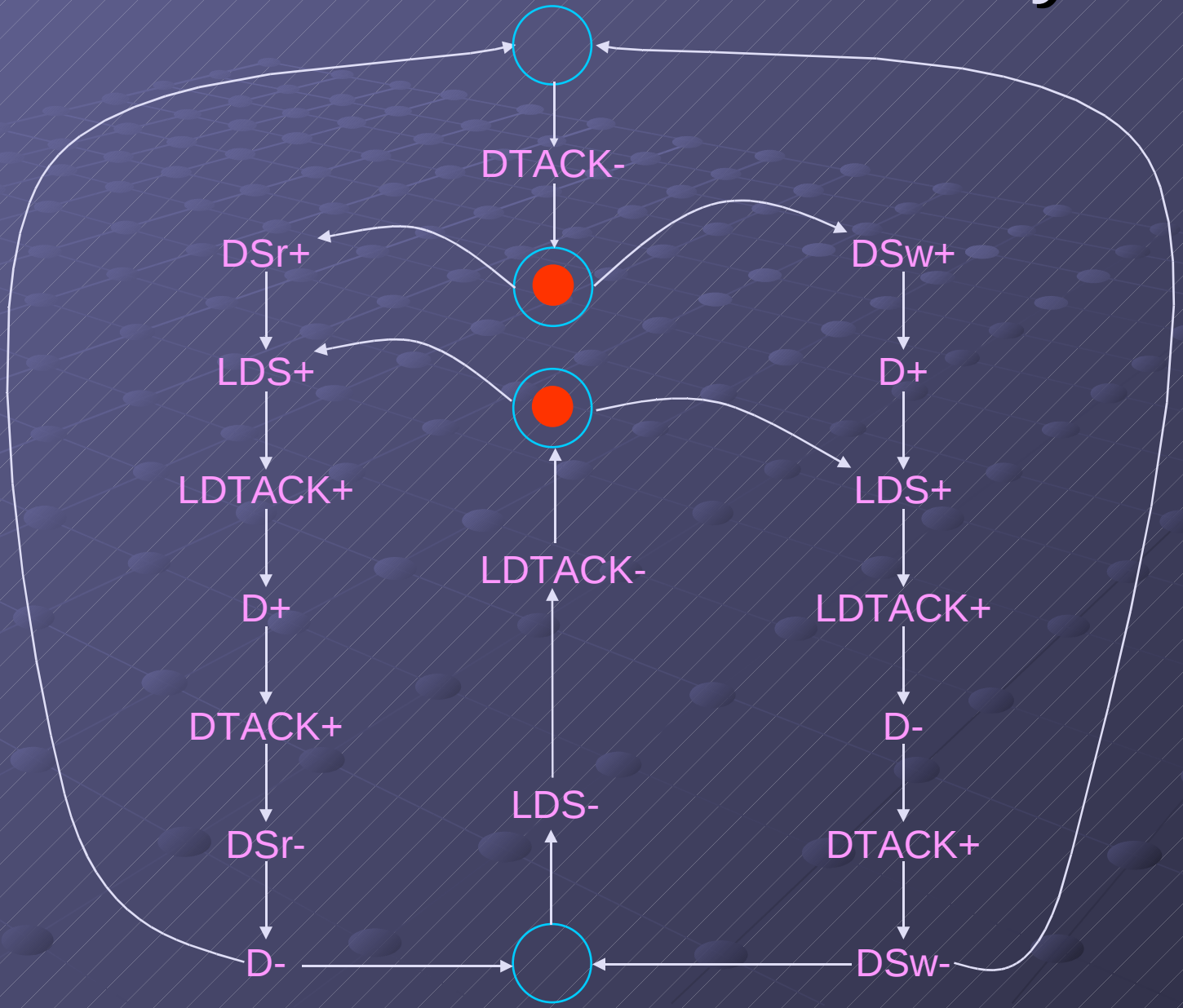




Choice: Read and Write cycles



Choice: Read and Write cycles



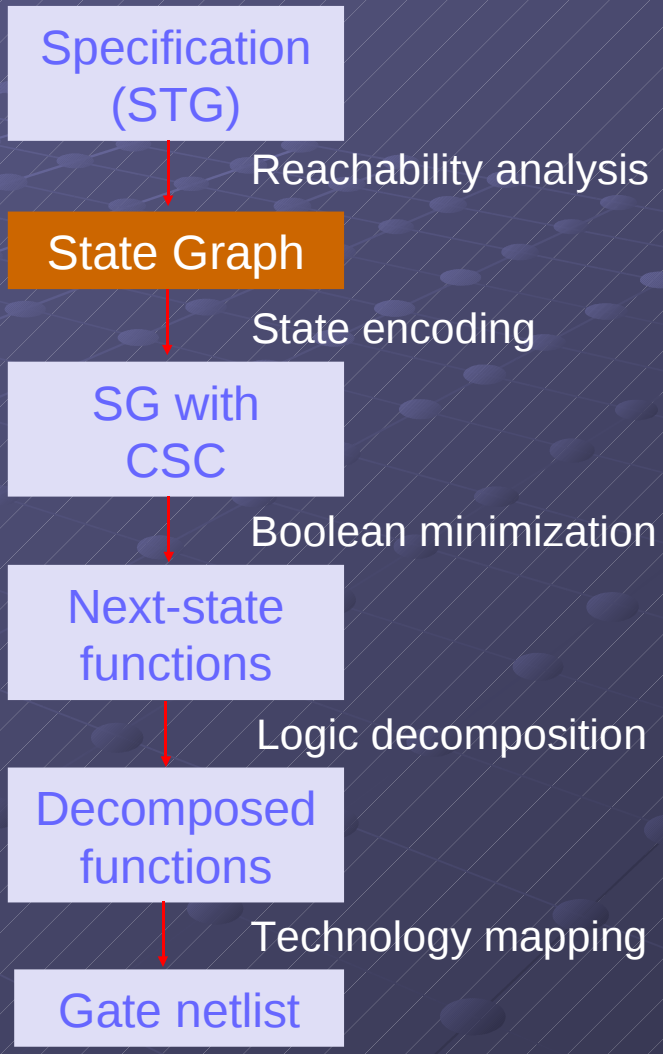
Circuit synthesis

- Goal:
  - Derive a hazard-free circuit under a given delay model and mode of operation

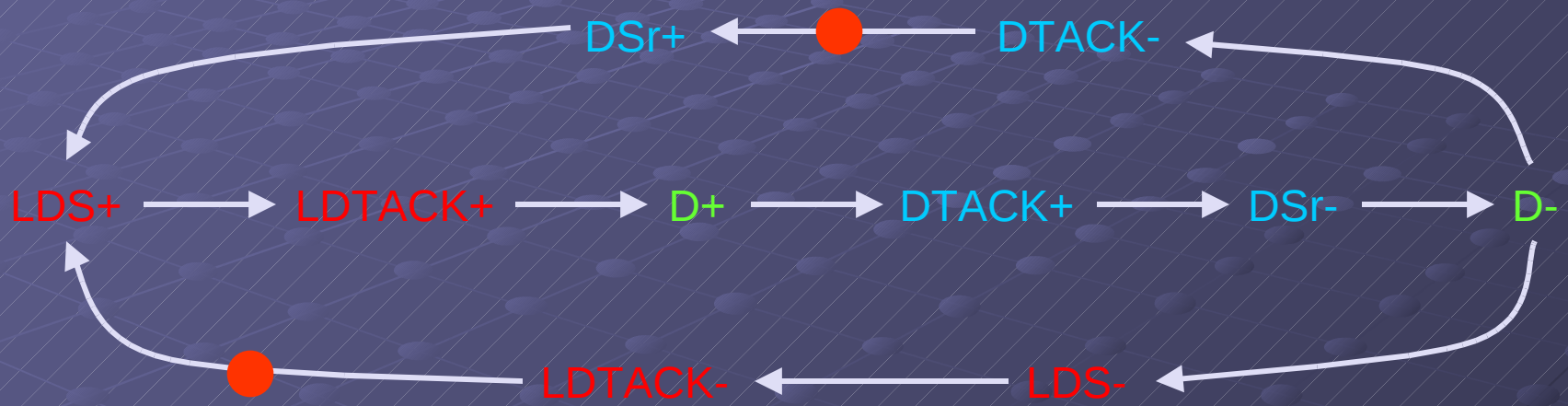
Speed independence

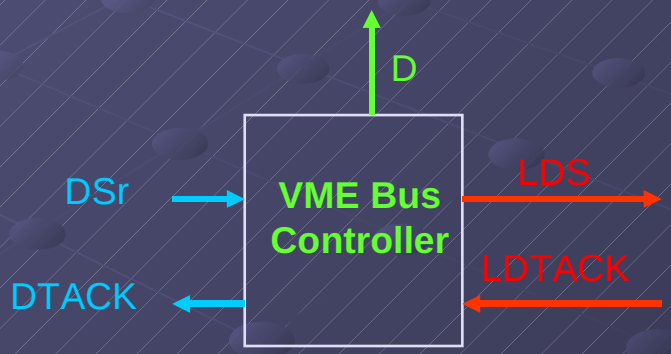
- Delay model
  - Unbounded gate / environment delays
  - Certain wire delays shorter than certain paths in the circuit
- Conditions for implementability:
  - Consistency
  - Complete State Coding
  - Persistency

Design flow

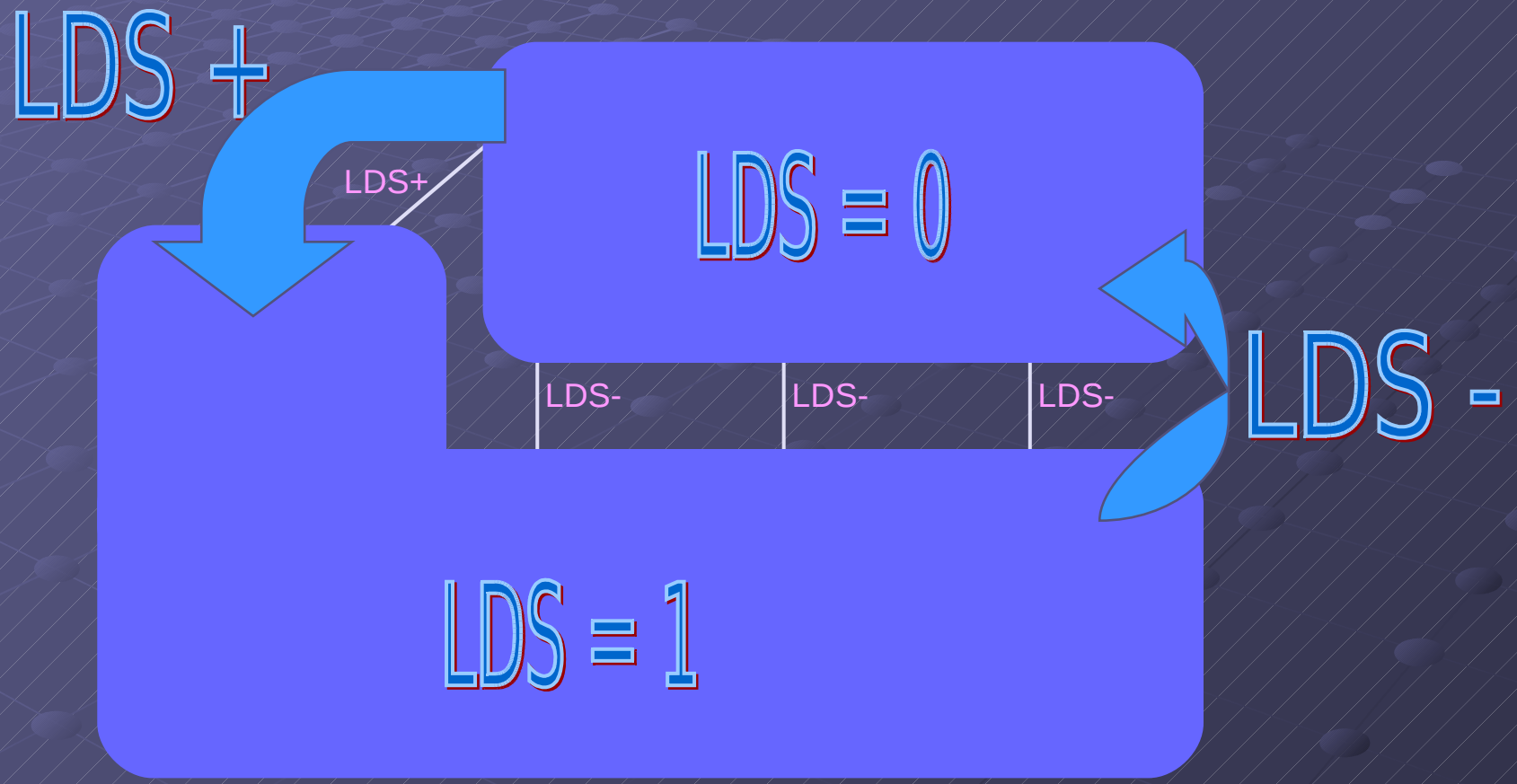

Specification (STG)
Reachability analysis
State Graph
State encoding
SG with CSC
Boolean minimization
Next-state functions
Logic decomposition
Decomposed functions
Technology mapping
Gate netlist

STG for the READ cycle

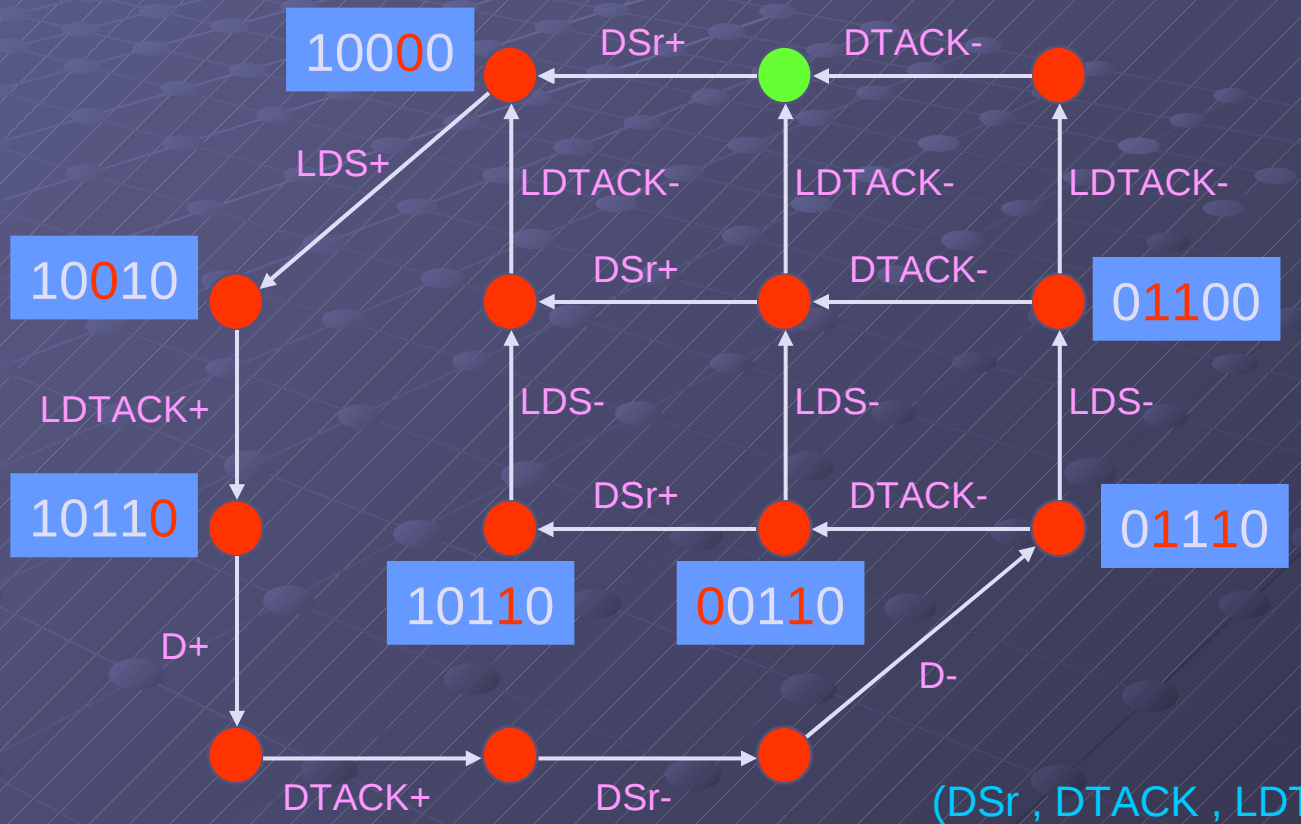




Binary encoding of signals

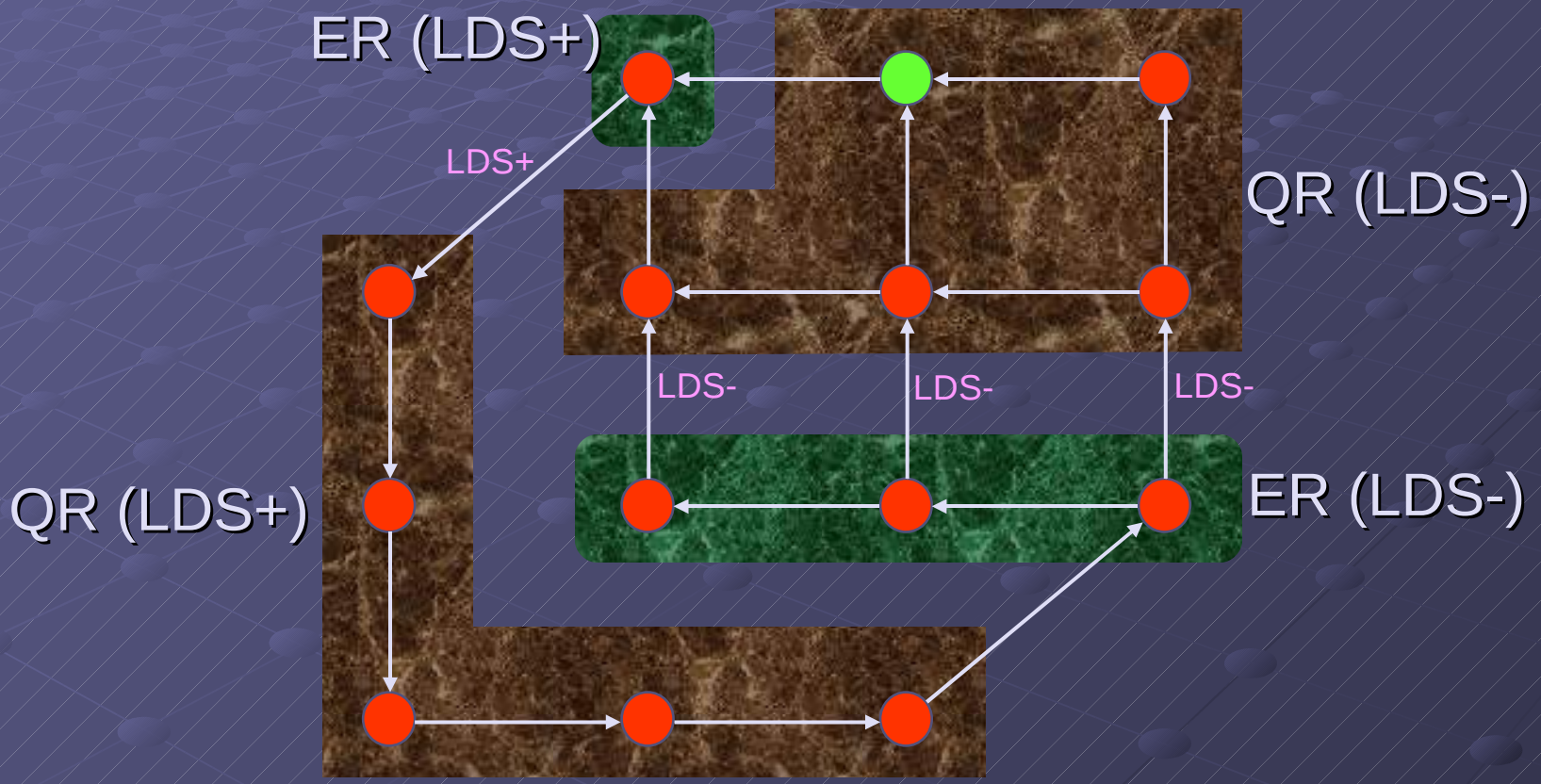


Binary encoding of signals

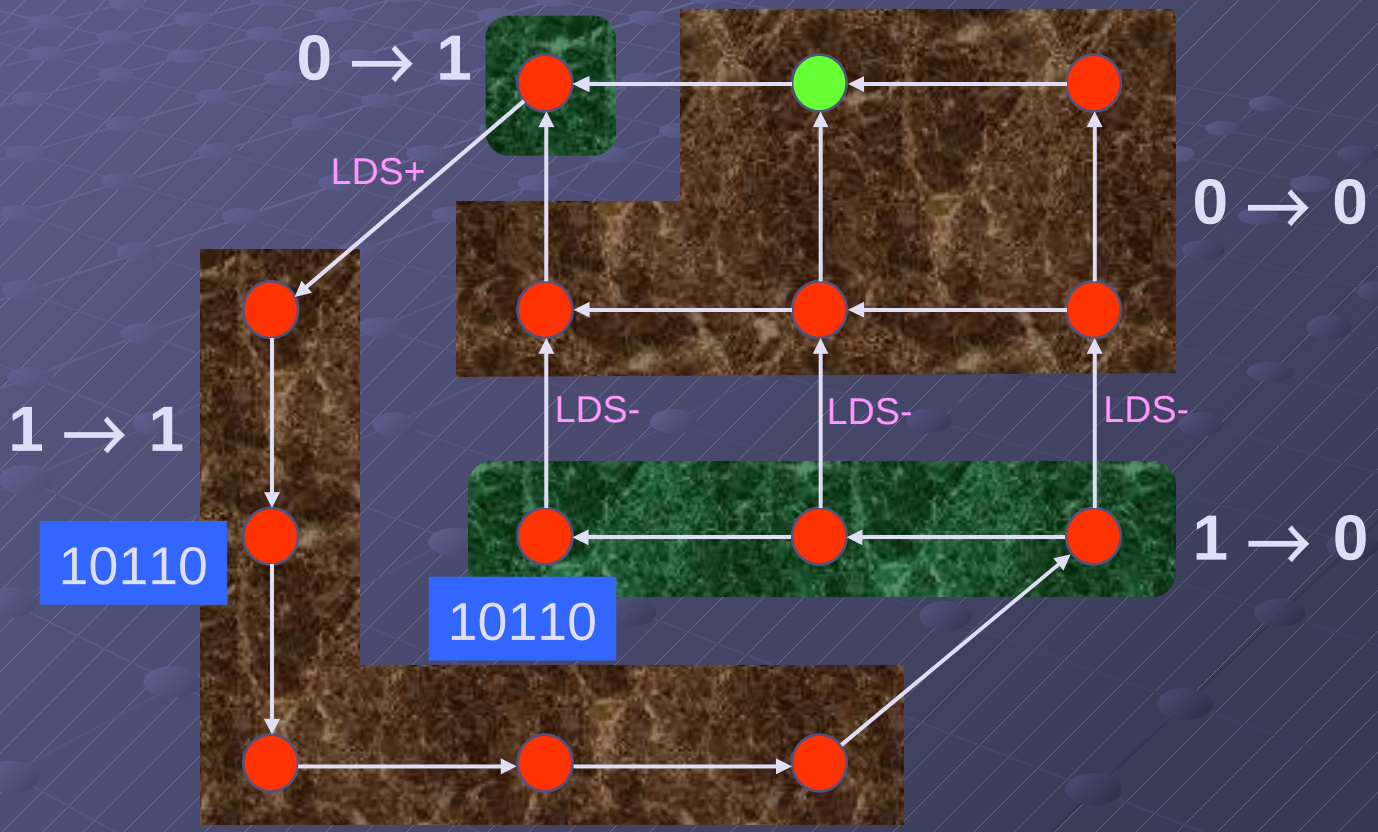


(DSr , DTACK , LDTACK , LDS , D)

Excitation / Quiescent Regions



Next-state function



Karnaugh map for LDS

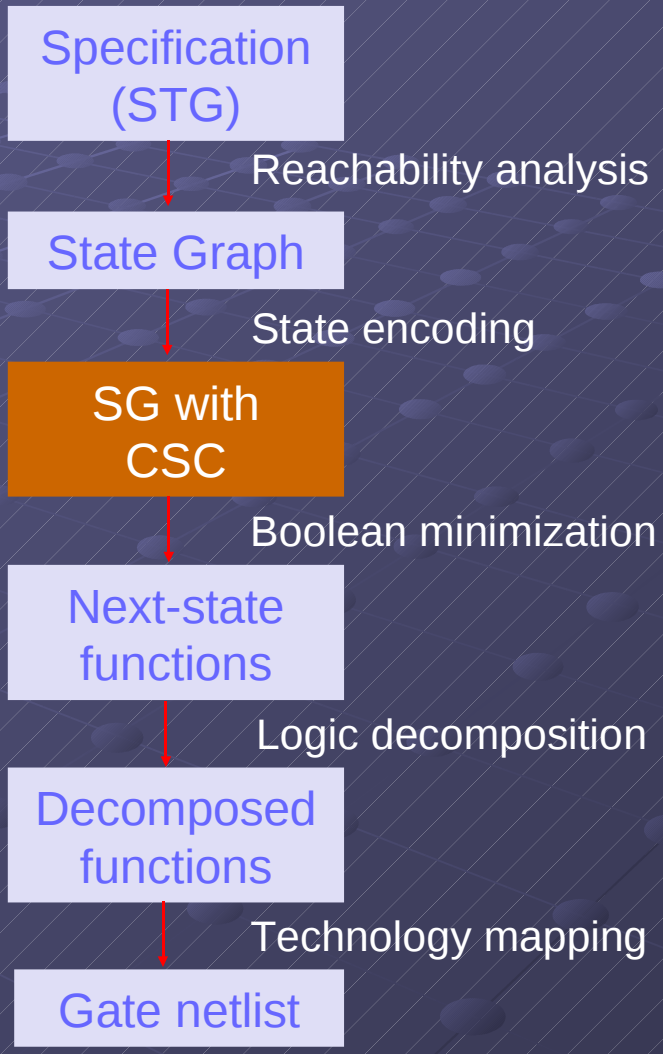
LDS = 0

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

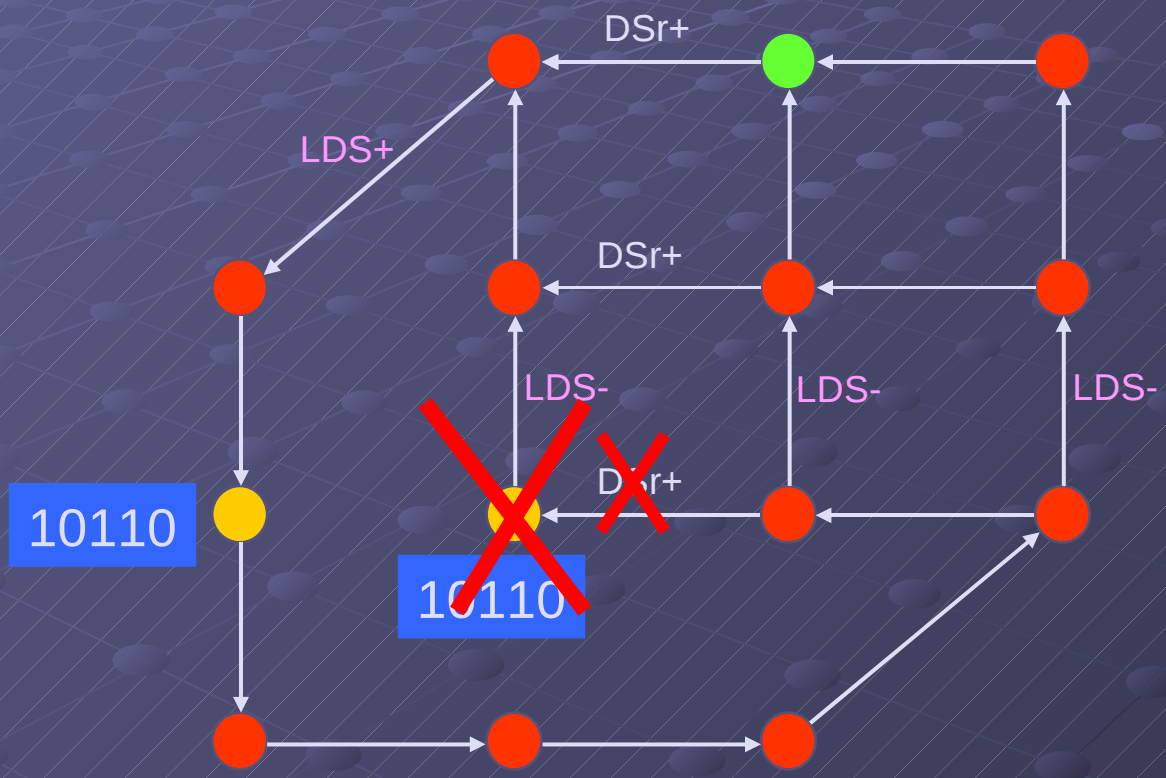
LDS = 1

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

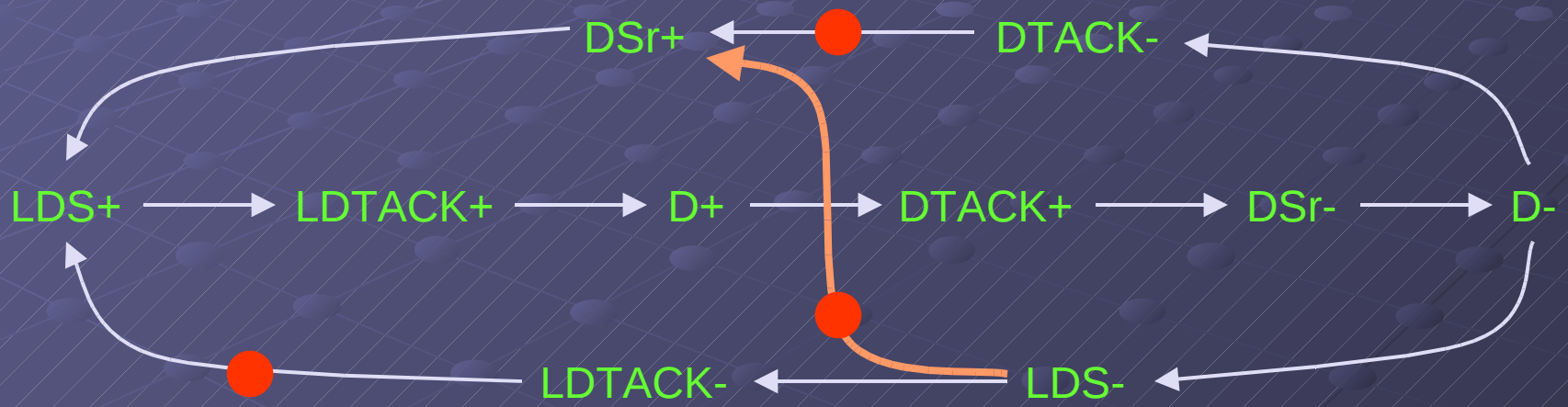
Design flow


Specification (STG)
Reachability analysis
State Graph
State encoding
SG with CSC
Boolean minimization
Next-state functions
Logic decomposition
Decomposed functions
Technology mapping
Gate netlist

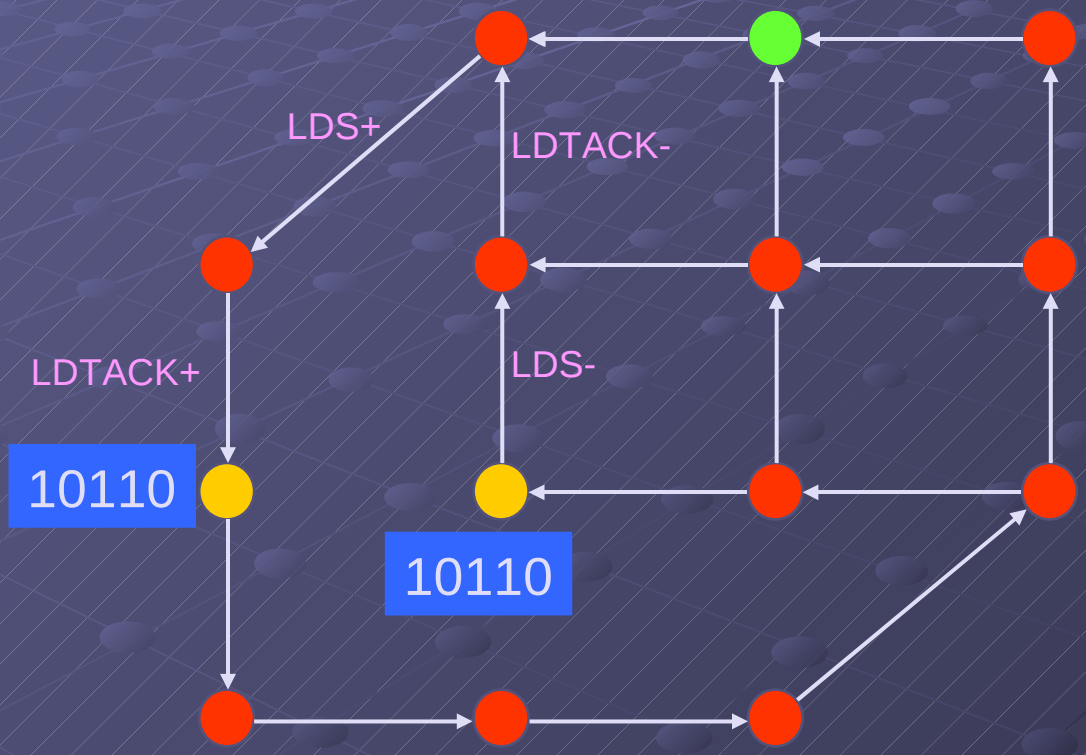
Concurrency reduction



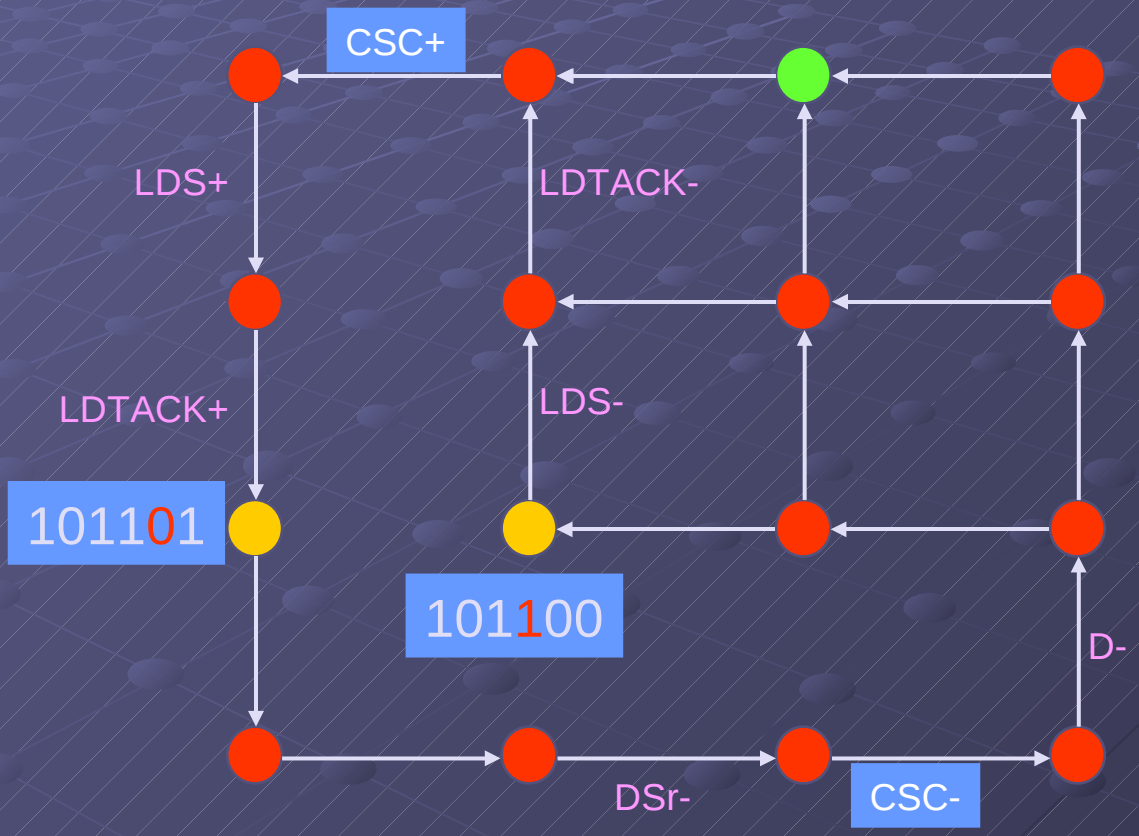
Concurrency reduction



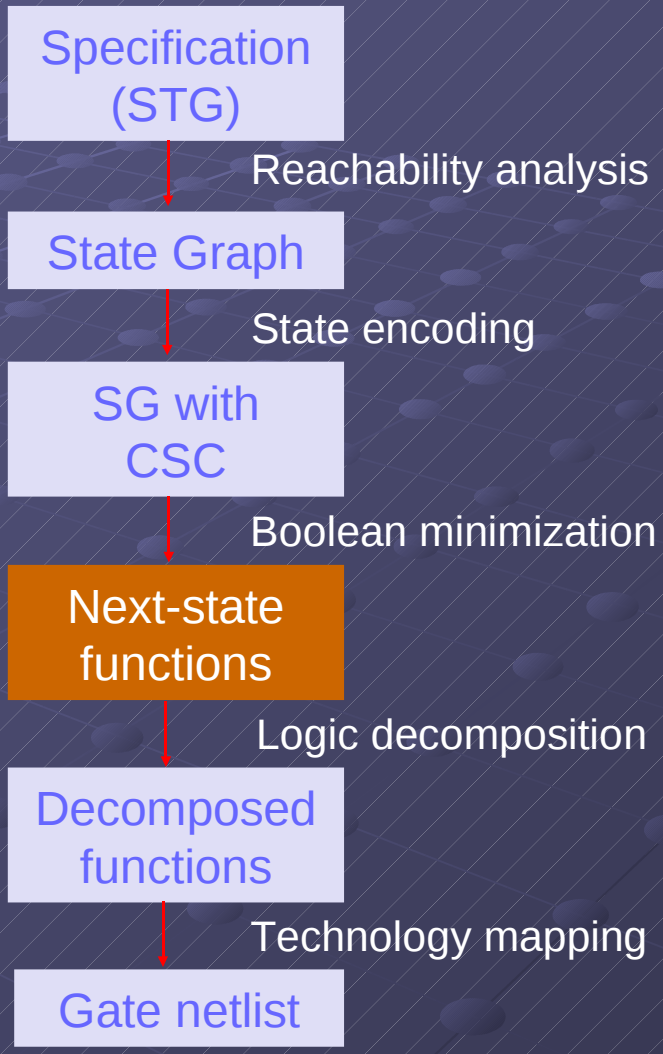
State encoding conflicts



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

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

$$DTACK = D$$

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

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

Implementability conditions

- 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)

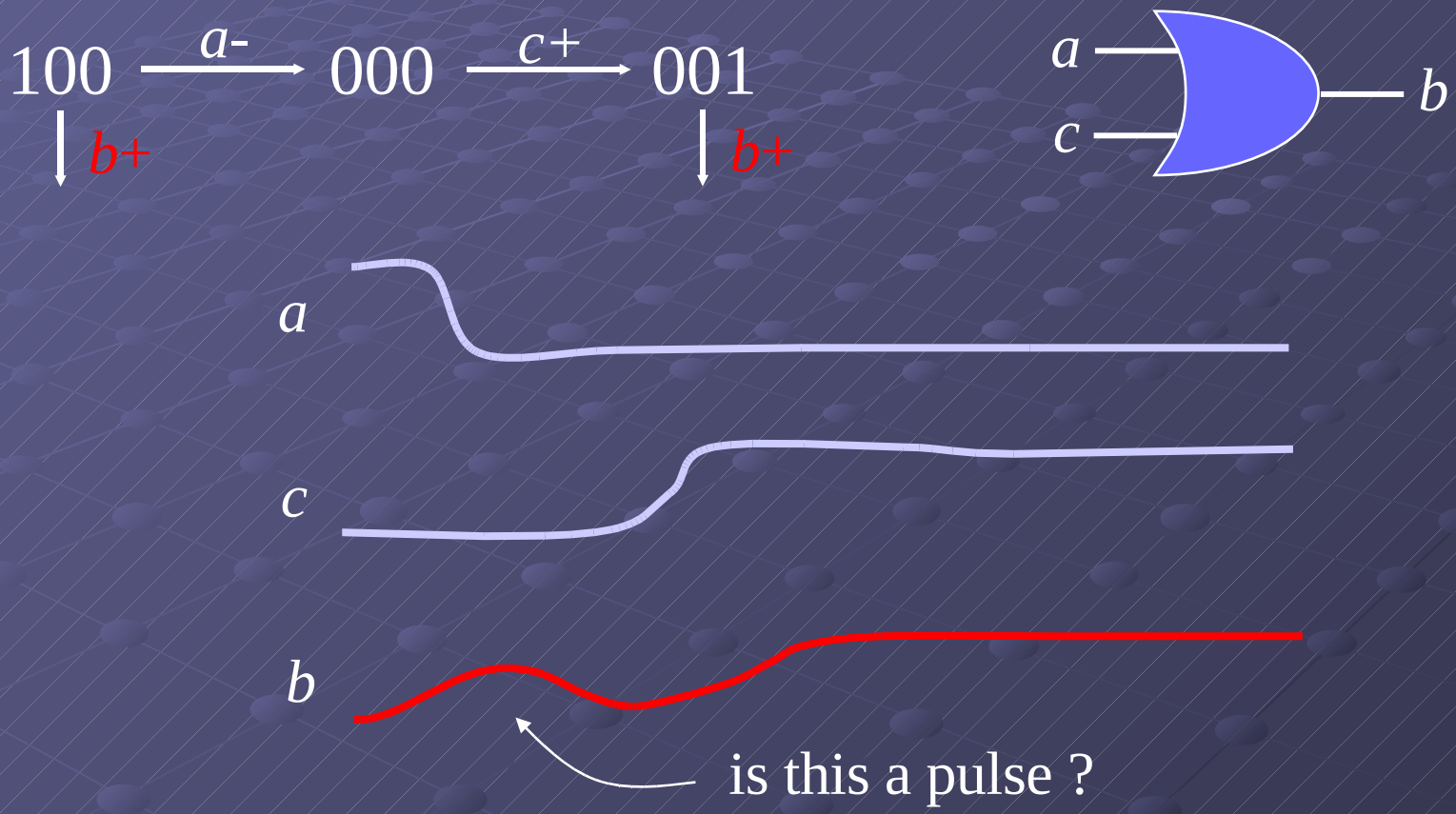
Implementability conditions

- Consistency + CSC + persistency

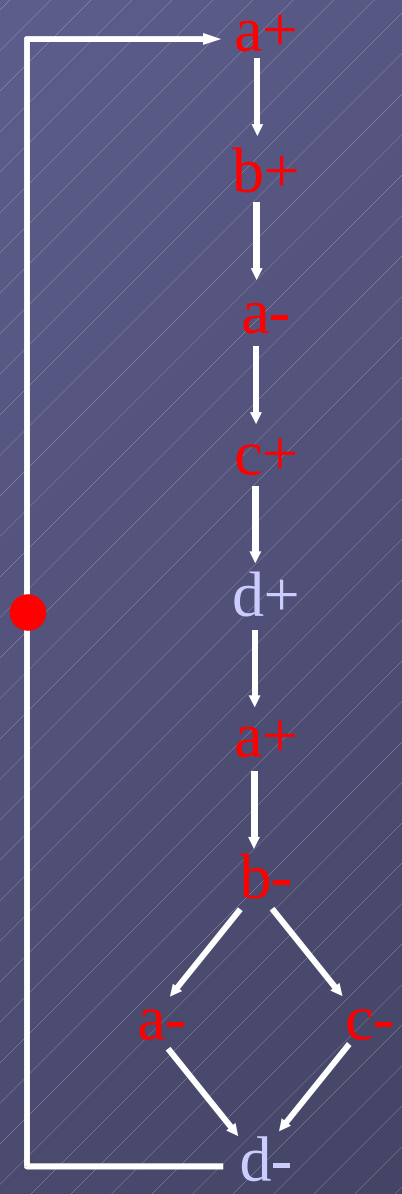
- There exists a speed-independent circuit that implements the behavior of the STG

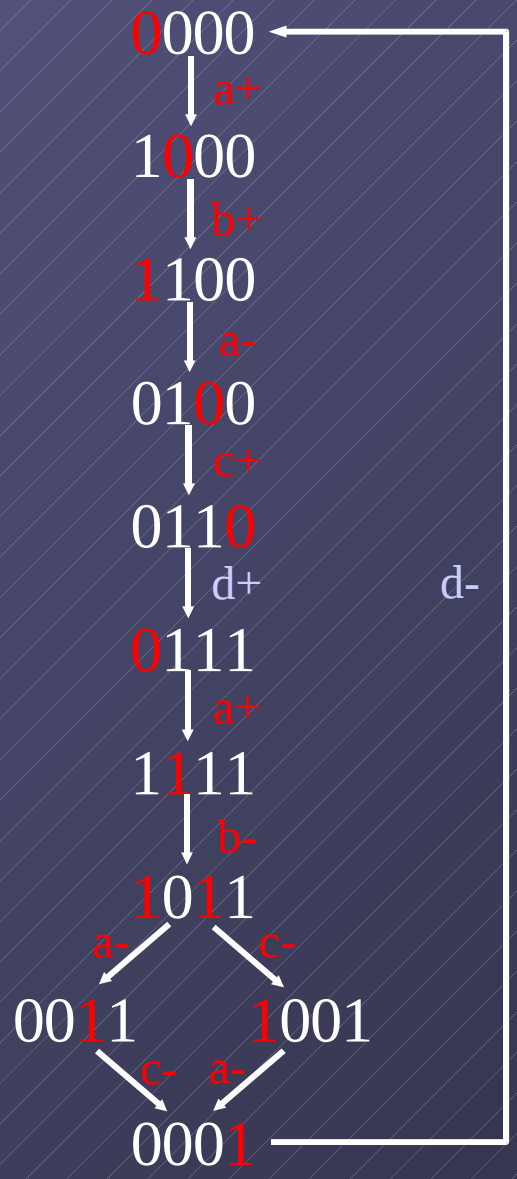
(under the assumption that ay Boolean function can be implemented with one complex gate)

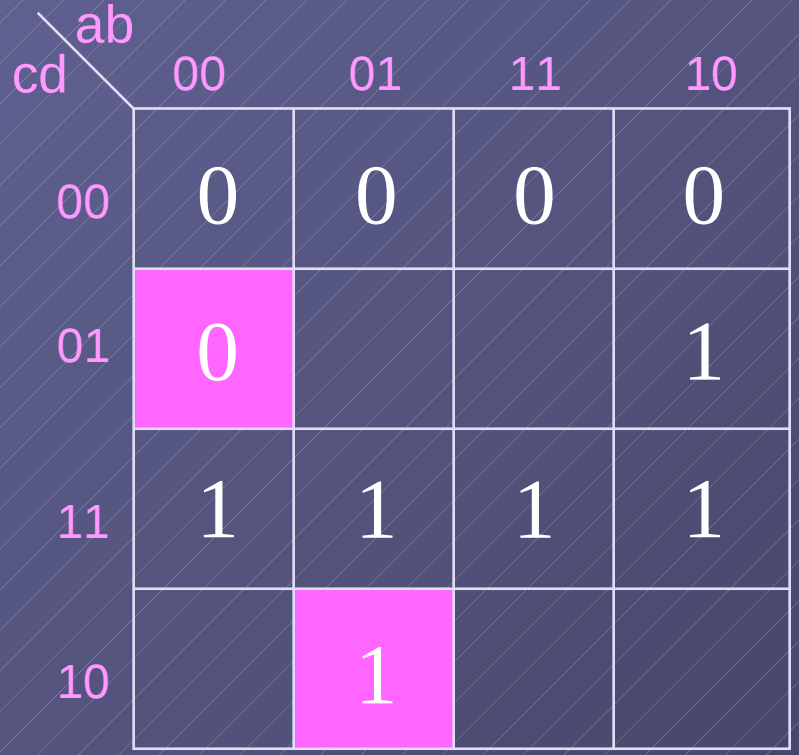
Persistency

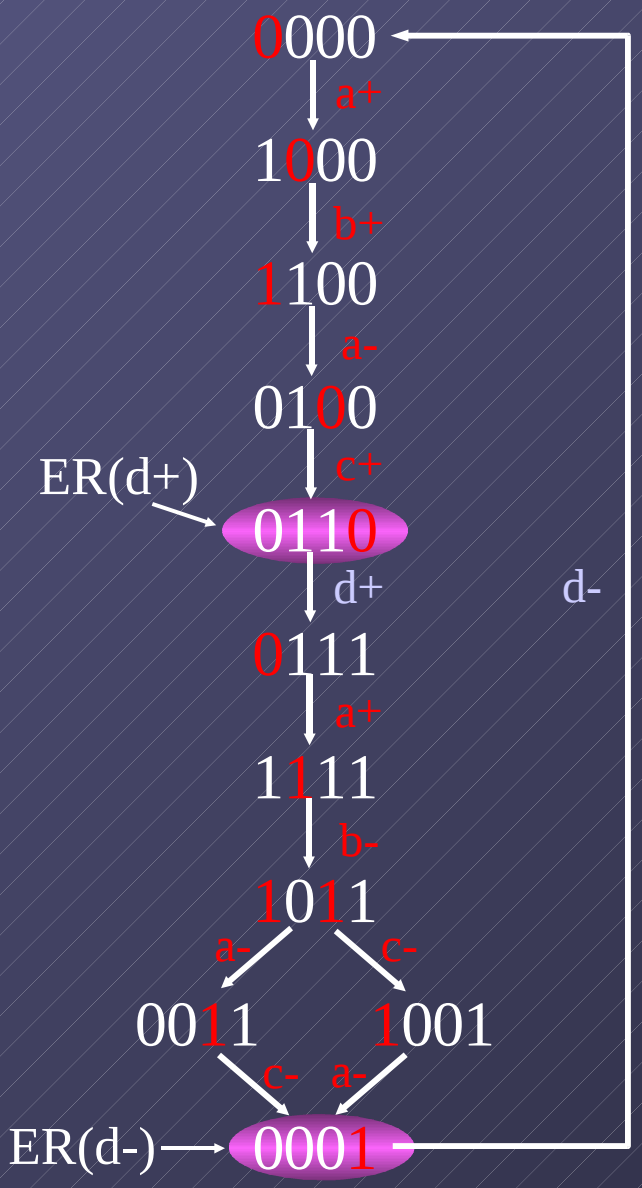


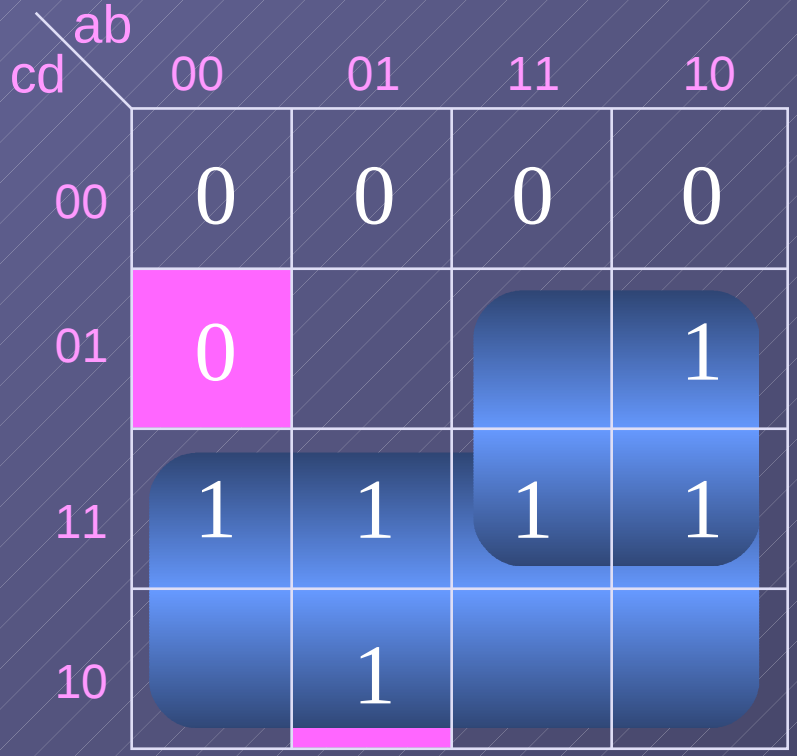
Speed independence ⇒ glitch-free output behavior under any delay


a+
b+
a-
c+
d+
a+
b-
a-
c-
d-


0000
a+
1000
b+
1100
a-
0100
c+
0110
d+
d-
0111
a+
1111
b-
1011
a-
c-
0011
1001
c-
a-
0001

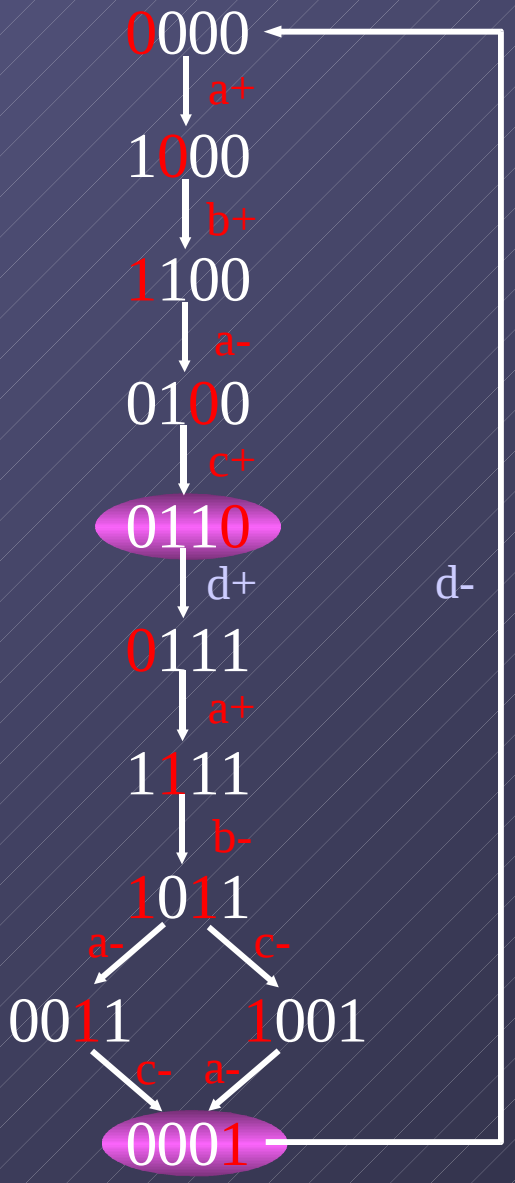

ab
cd
00
01
11
10
00
01
11
10
0 0 0 0
0 1
1 1 1 1
1


0000
a+
1000
b+
1100
a-
0100
c+
ER(d+)
0110
d+
d-
0111
a+
1111
b-
1011
a-
c-
0011
1001
c-
a-
ER(d-)
0001

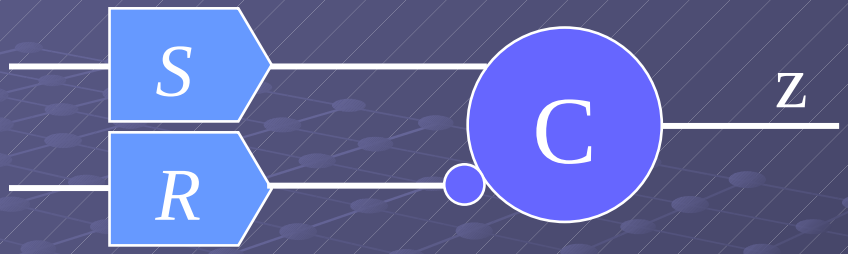


$$d = c + ad$$

Complex gate

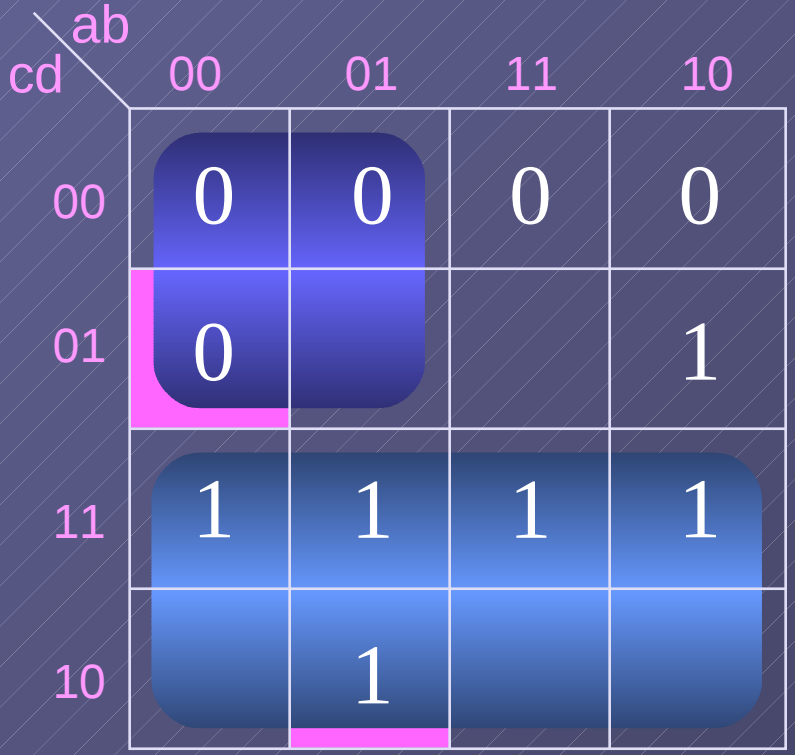


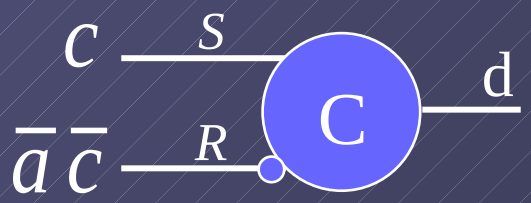
Implementation with C elements

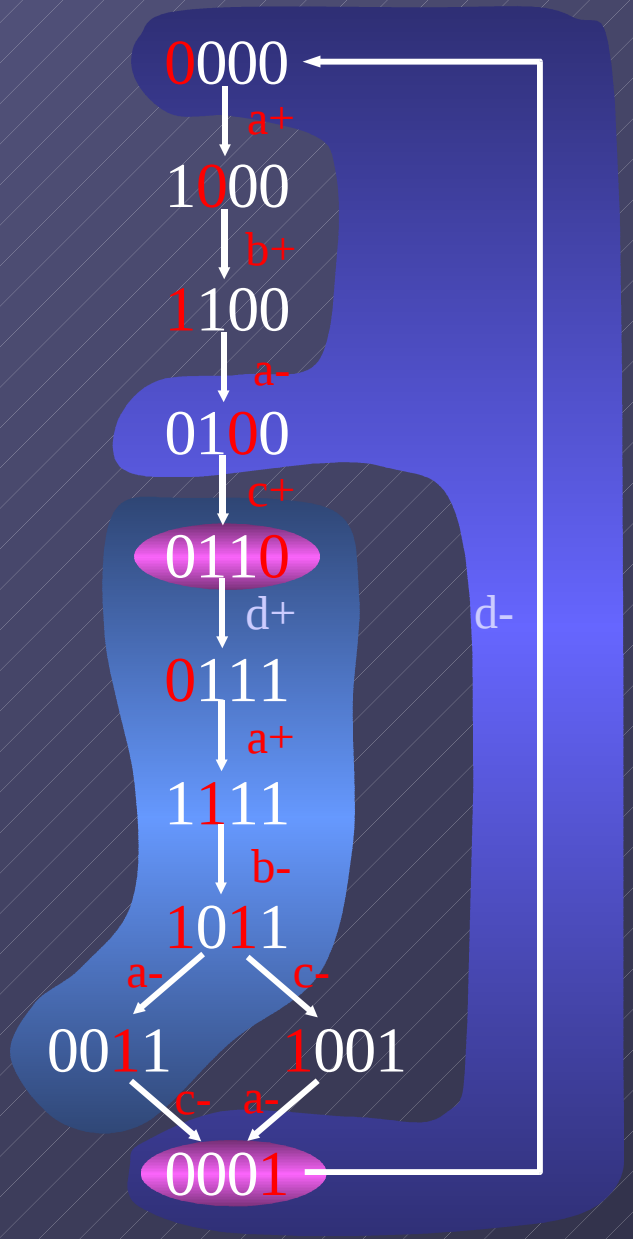


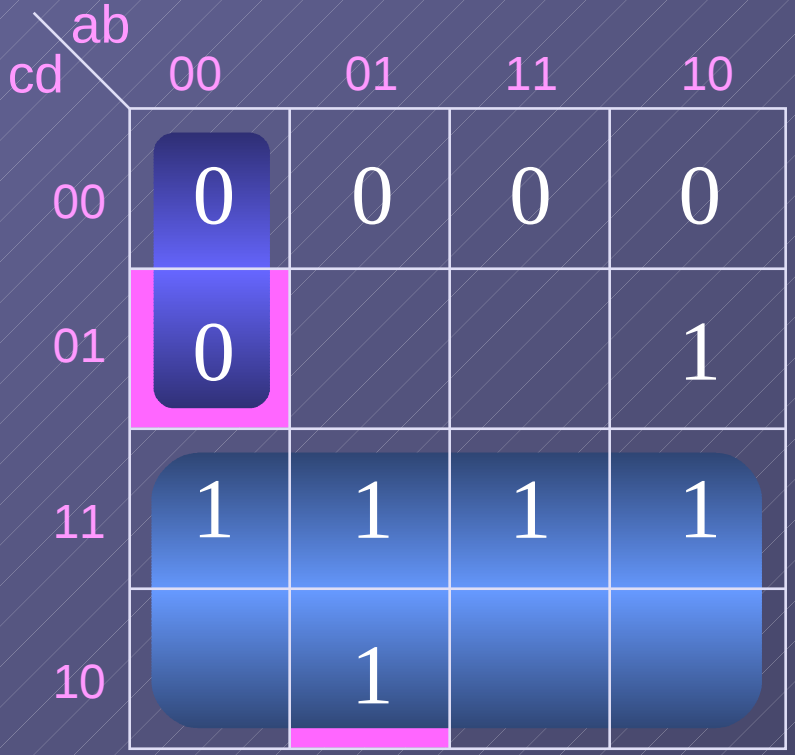
$\bullet\bullet\bullet \rightarrow S+ \rightarrow z+ \rightarrow S- \rightarrow R+ \rightarrow z- \rightarrow R- \rightarrow \bullet\bullet\bullet$

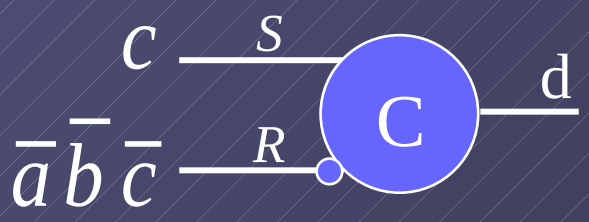
- *S* (set) and *R* (reset) must be mutually exclusive
- *S* must cover $ER(z+)$ and must not intersect $ER(z-) \cup QR(z-)$
- *R* must cover $ER(z-)$ and must not intersect $ER(z+) \cup QR(z+)$

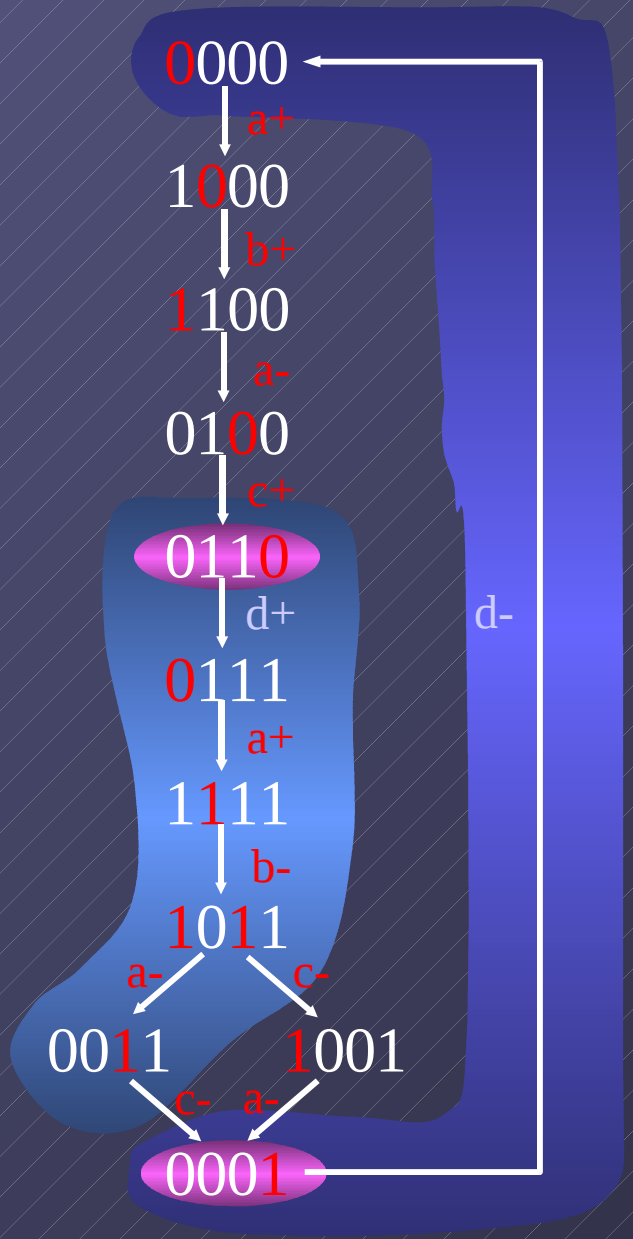

ab
cd
00
01
11
10
00
01
11
10
0 0 0 0
0 1
1 1 1 1
1


c
S
$\overline{a}\,\overline{c}$
R
C
d


0000
a+
1000
b+
1100
a-
0100
c+
0110
d+
0111
a+
1111
b-
1011
a-
c-
0011
1001
c-
a-
0001
d-

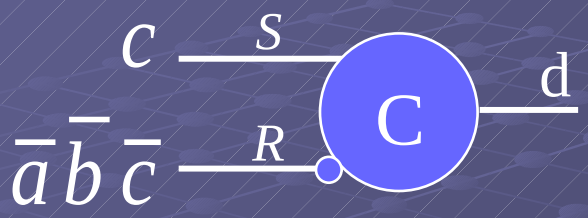


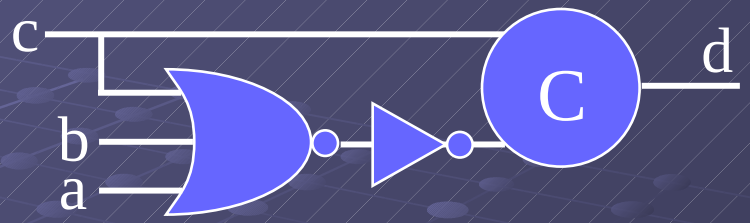


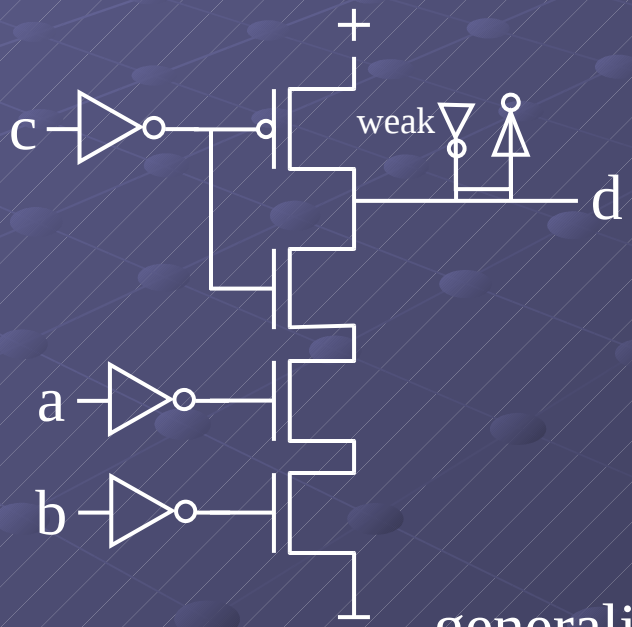


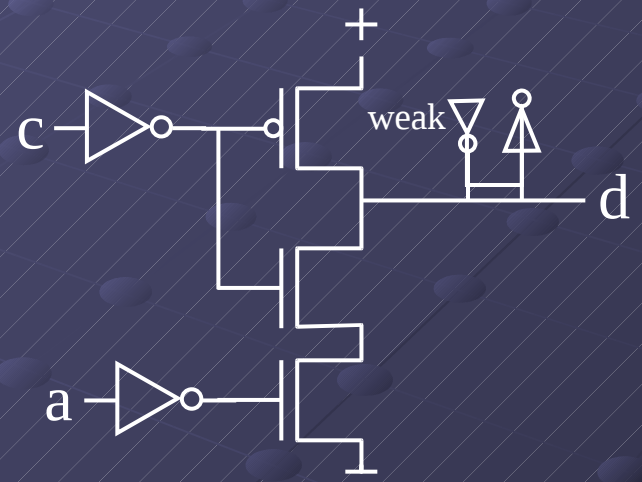
Monotonic covers

C-based implementations







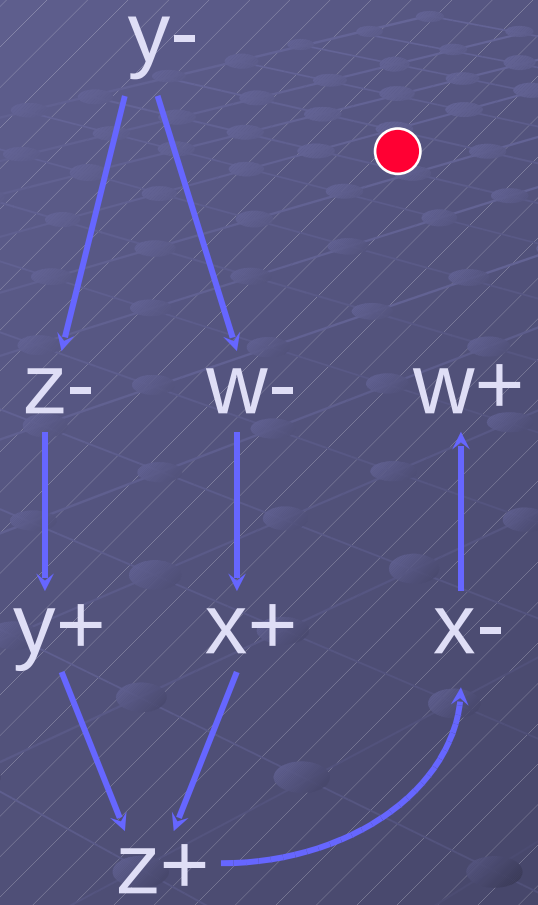


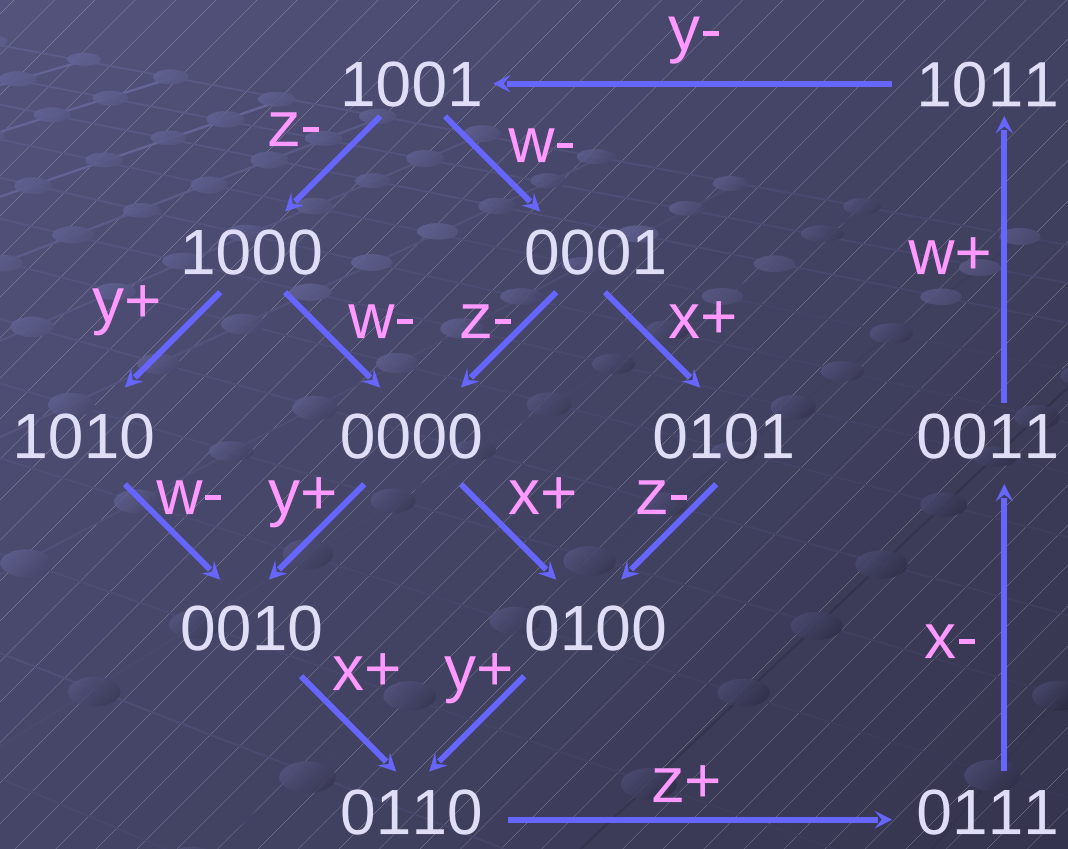
generalized C elements (gC)

Speed-independent implementations

- Implementability conditions
  - Consistency
  - Complete state coding
  - Persistency
- Circuit architectures
  - Complex (hazard-free) gates
  - C elements with monotonic covers
  - ...

Synthesis exercise



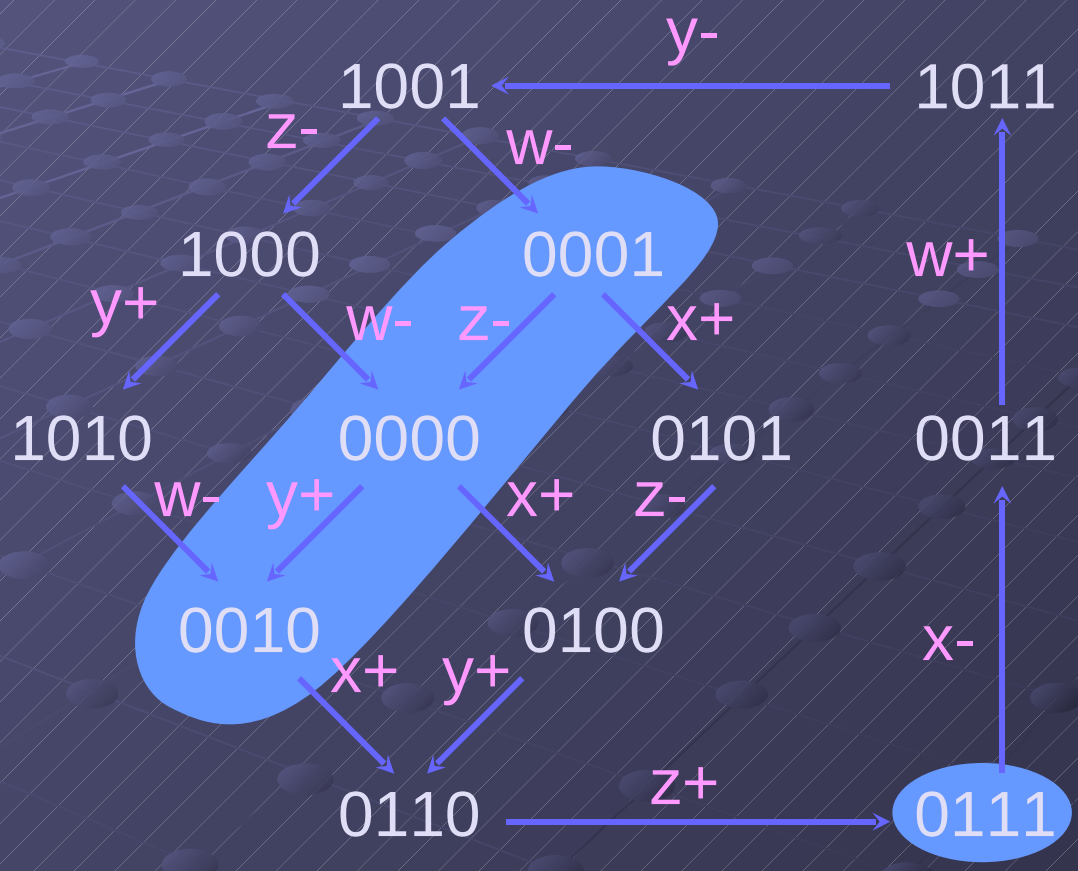


Derive circuits for signals *x* and *z* (complex gates and monotonic covers)

Synthesis exercise

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

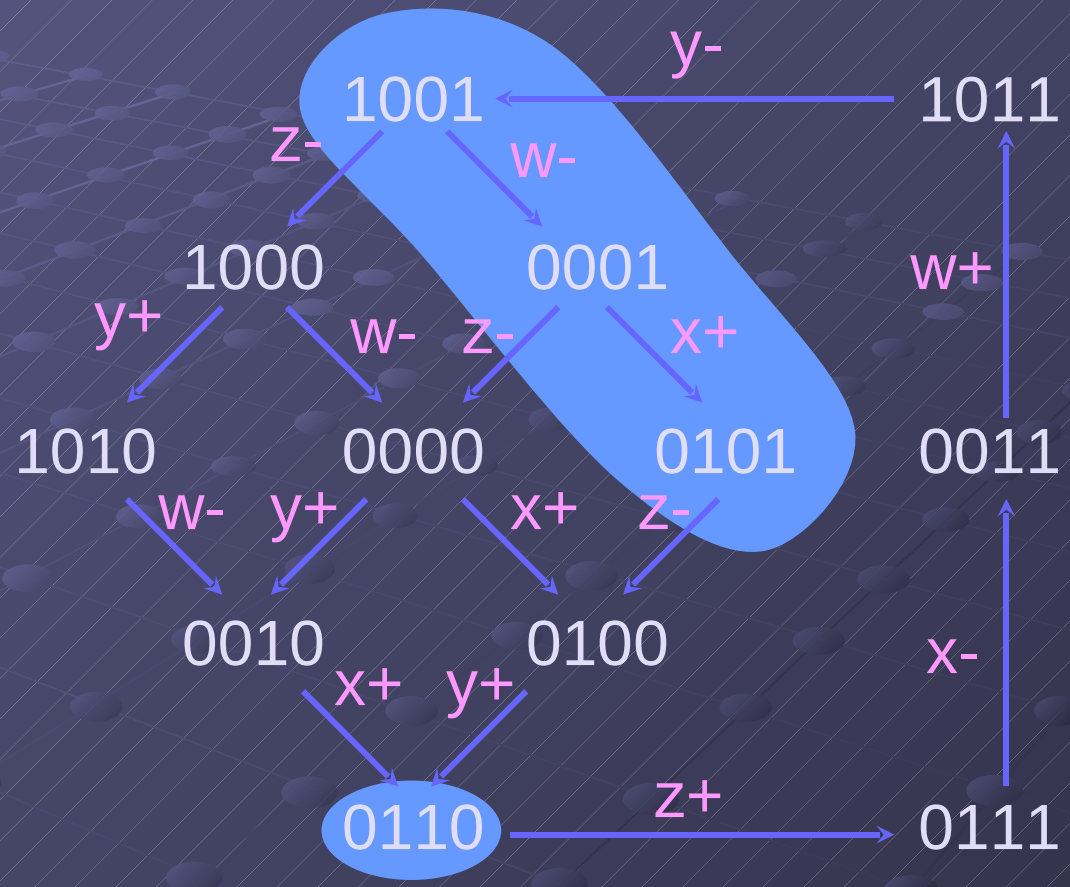
Signal *x*



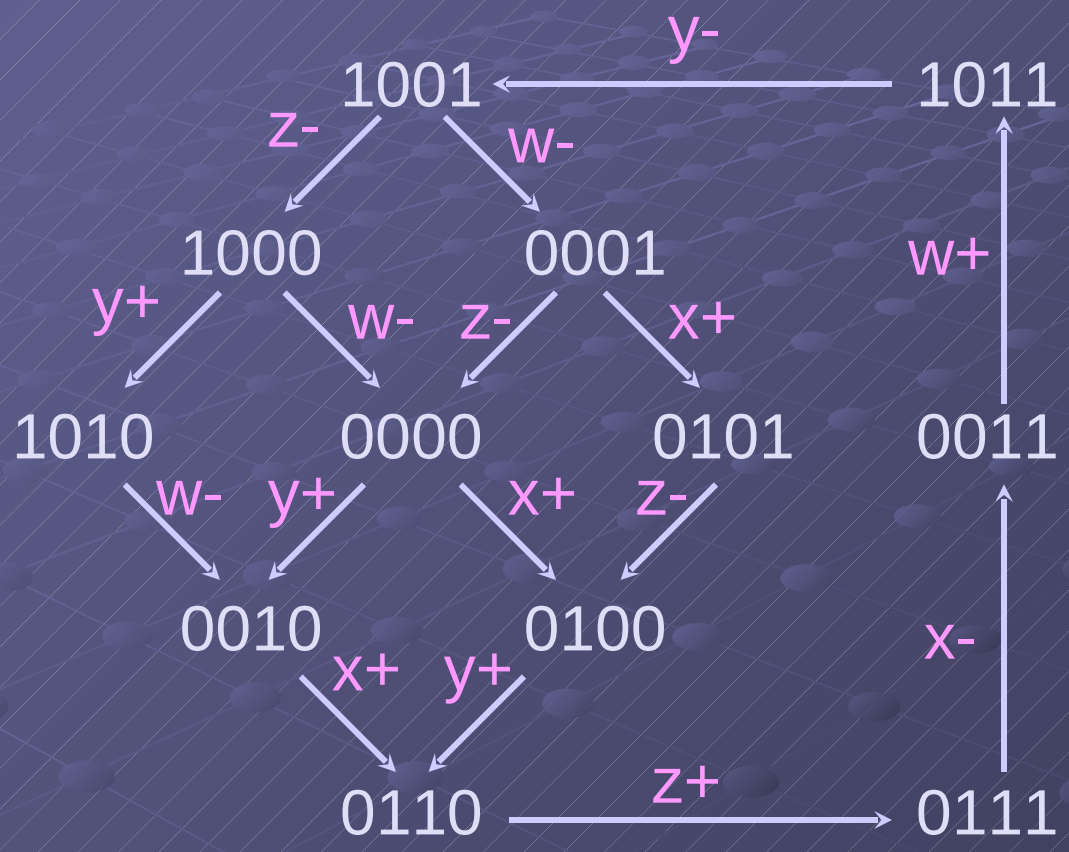
Synthesis exercise

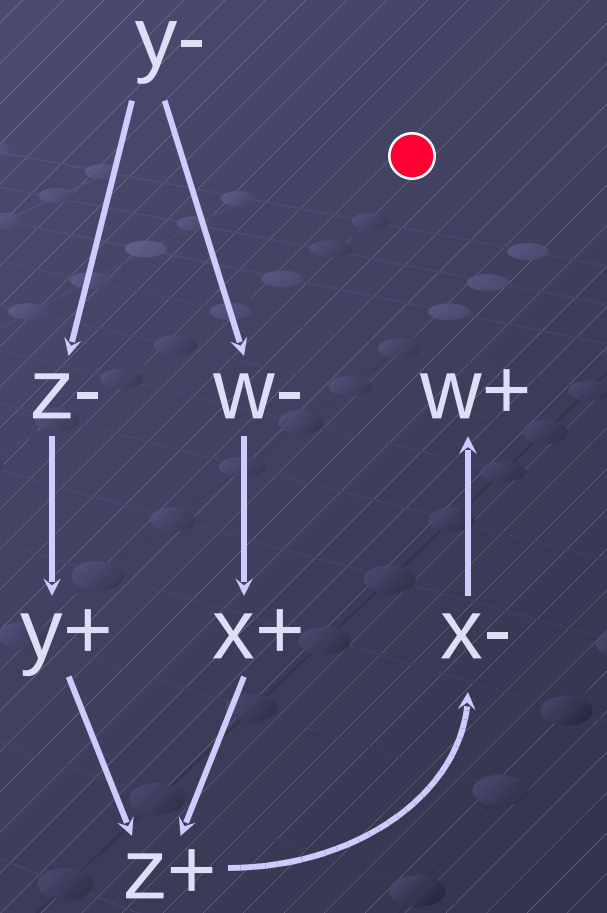
yz \ wx	00	01	11	10
00	0	0	-	0
01	0	0	-	0
11	1	1	-	1
10	0	1	-	0

Signal *z*

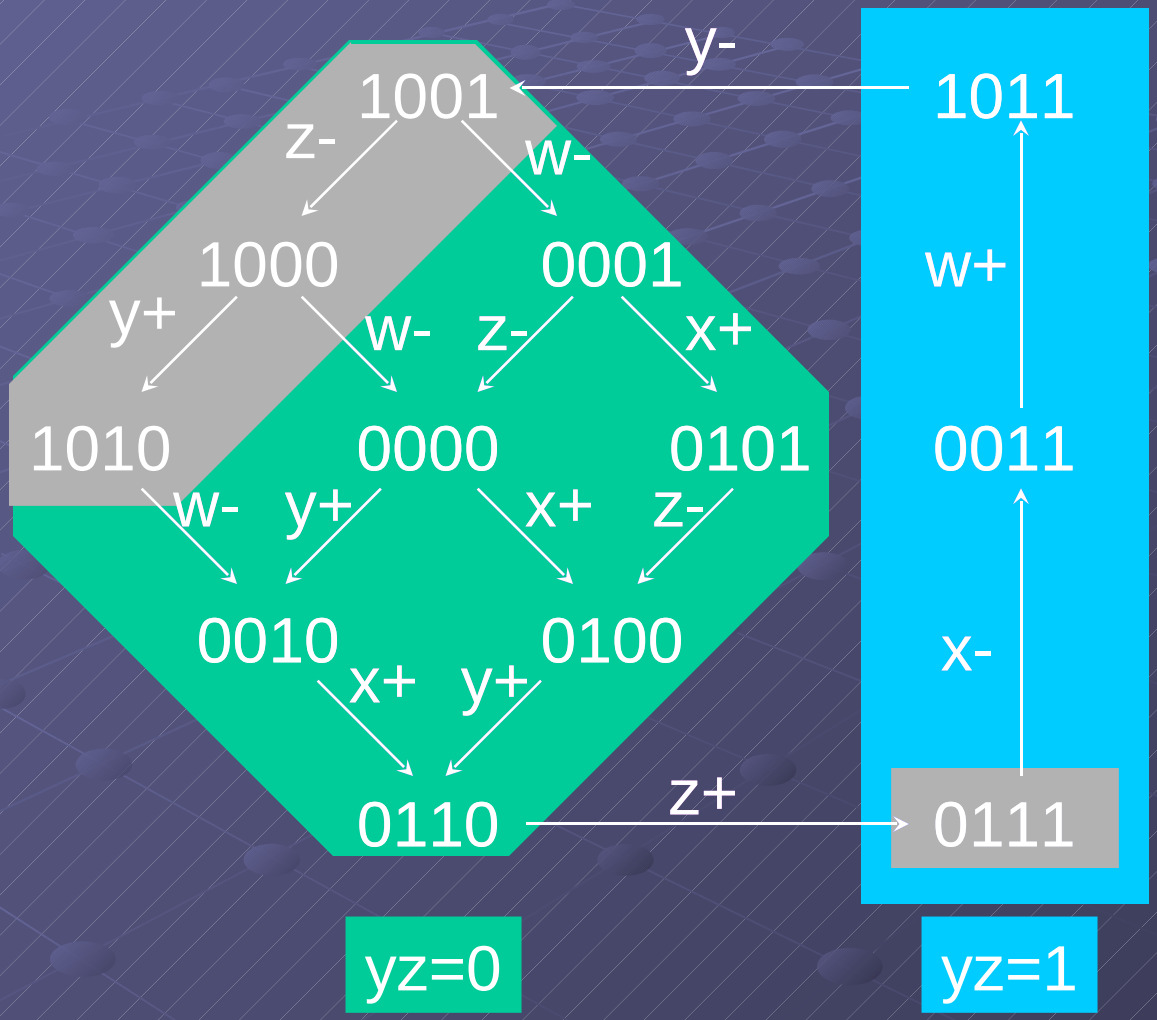


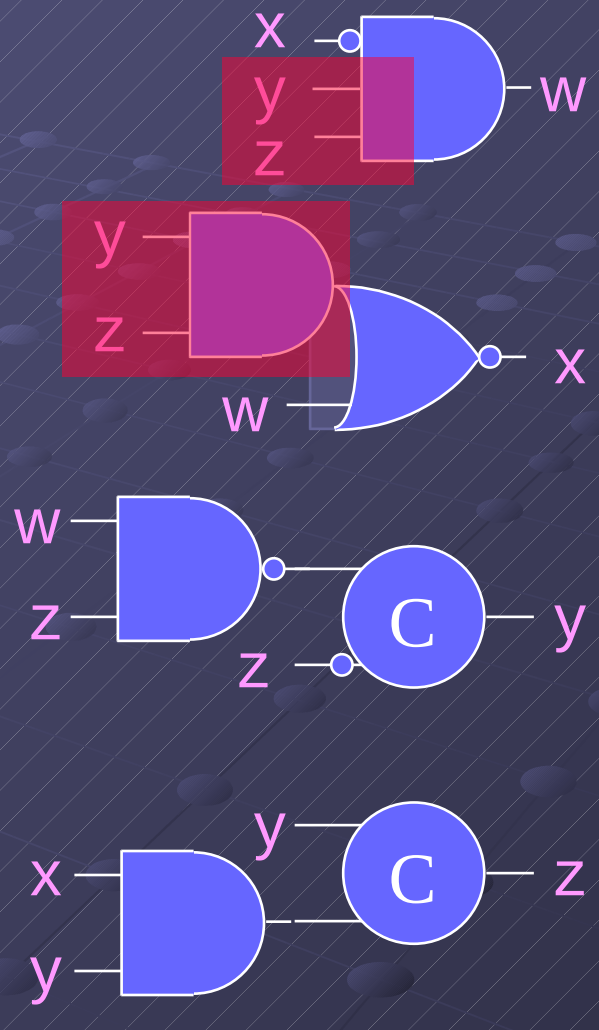
Logic decomposition: example



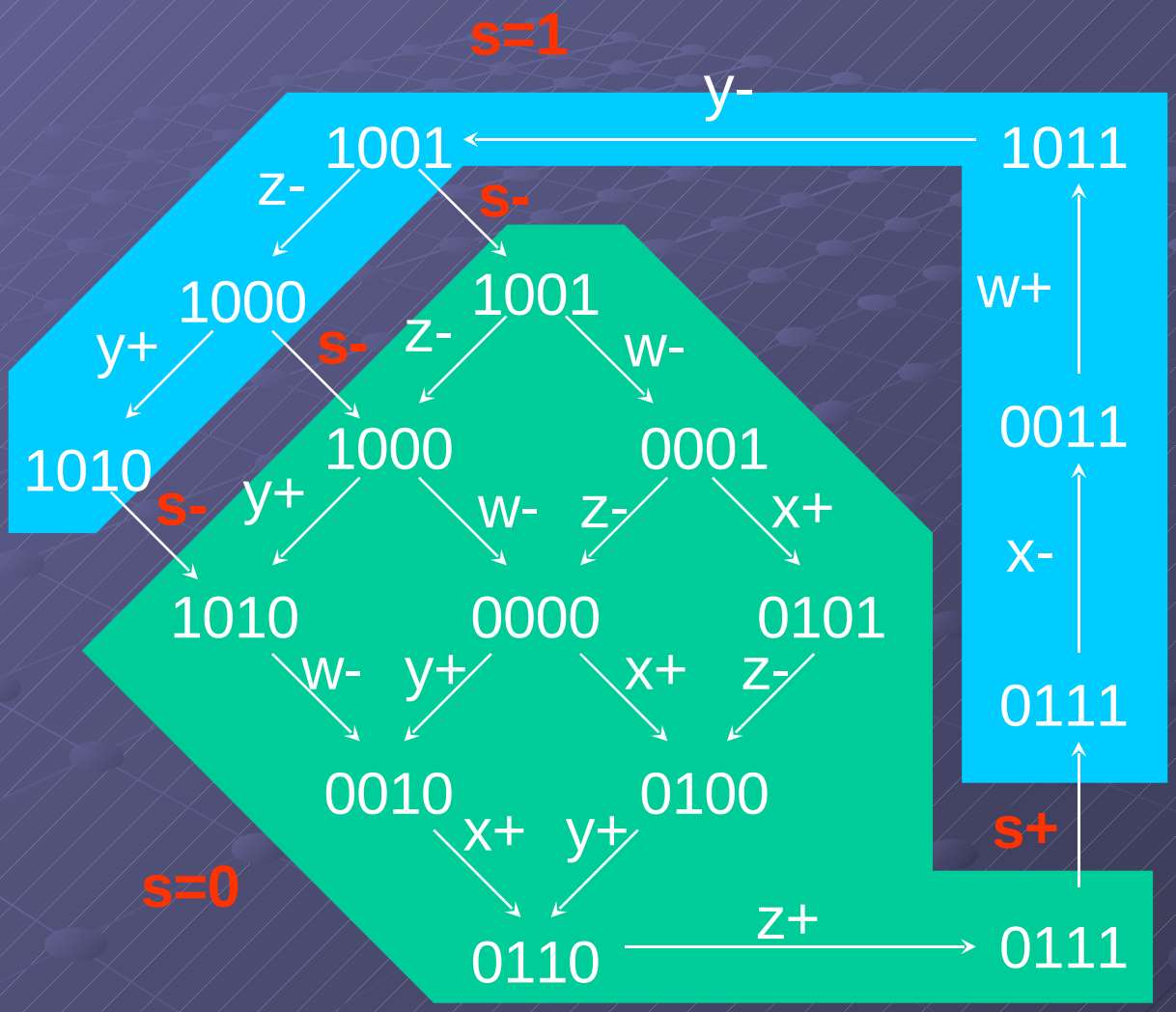


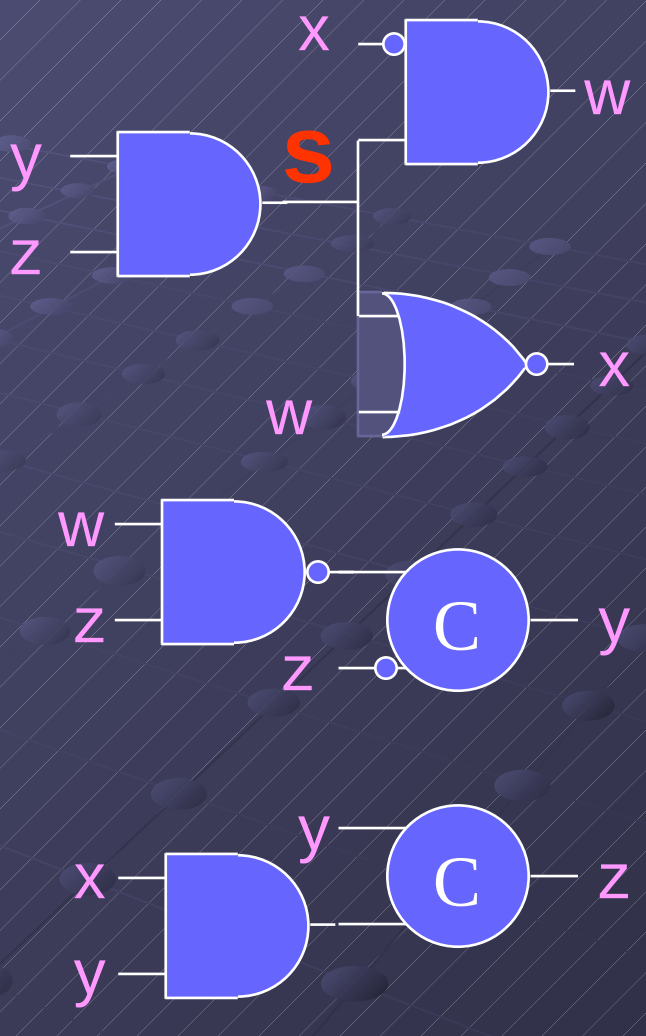
Logic decomposition: example



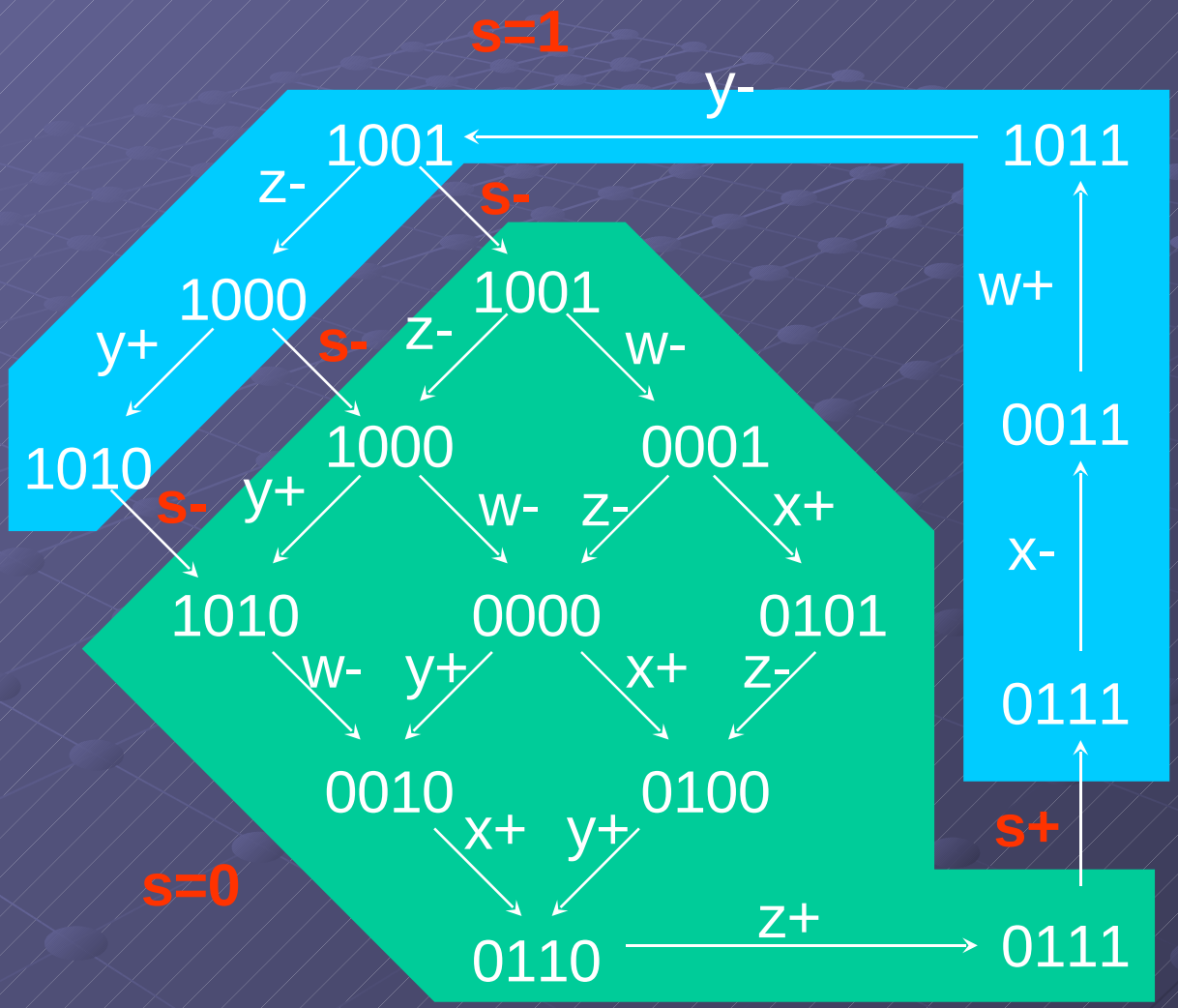


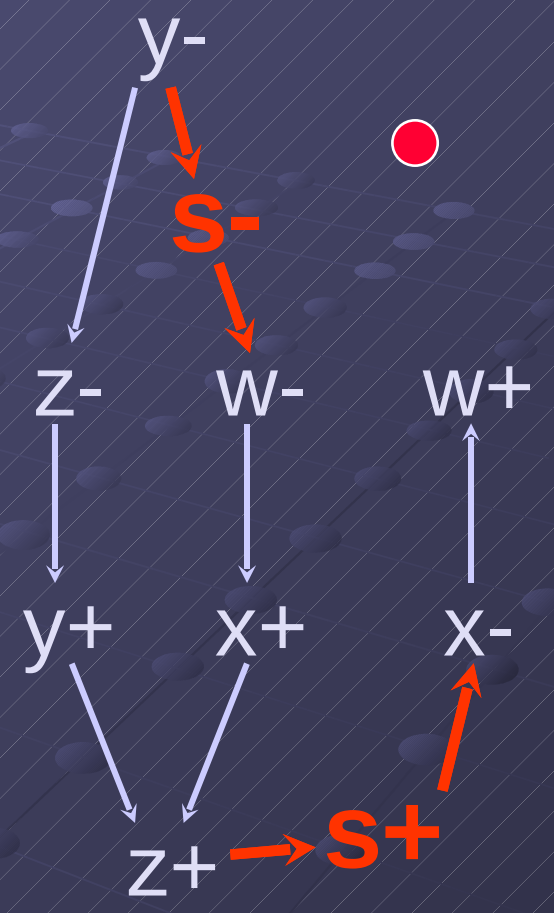
Logic decomposition: example



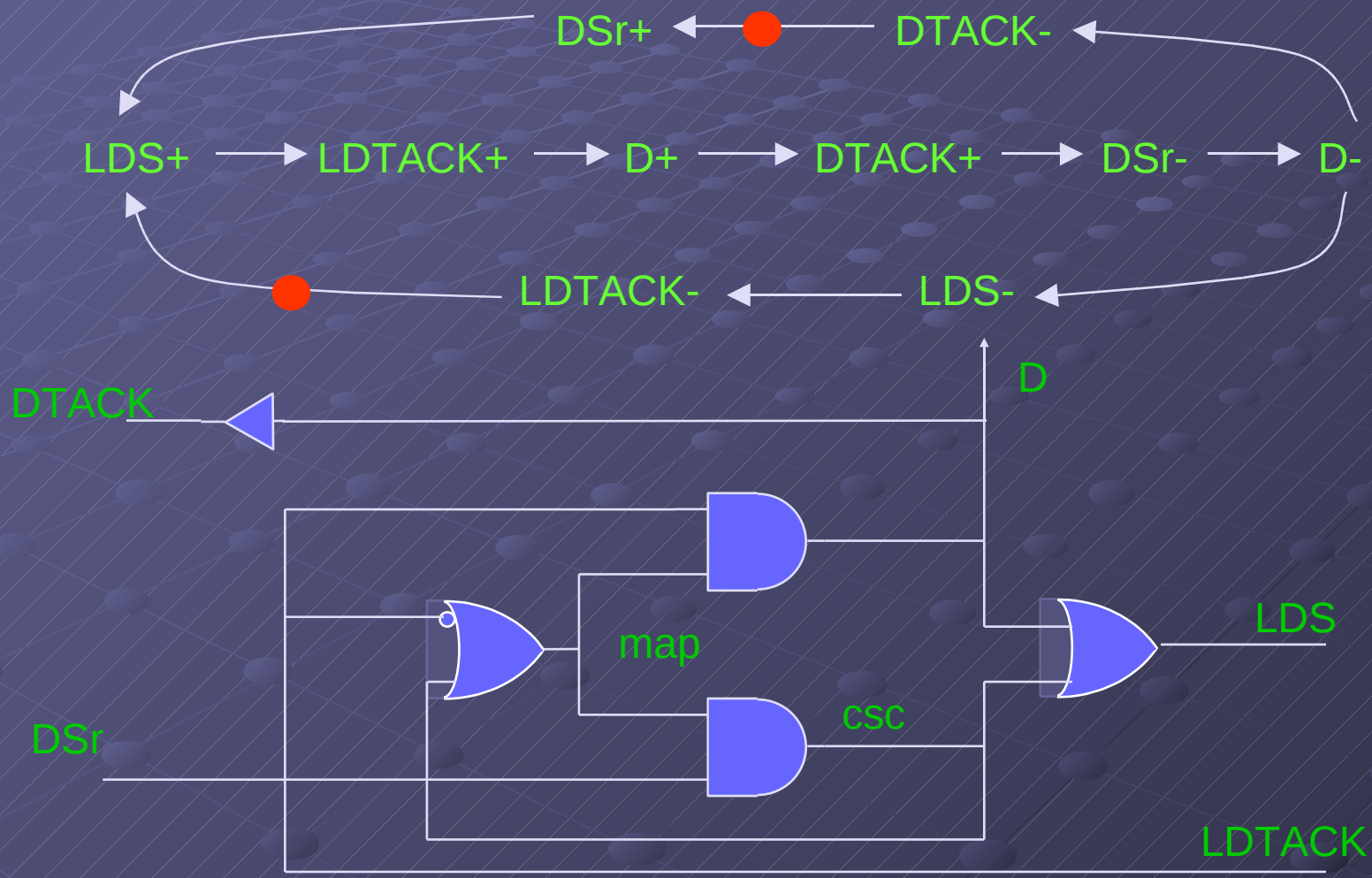


Logic decomposition: example

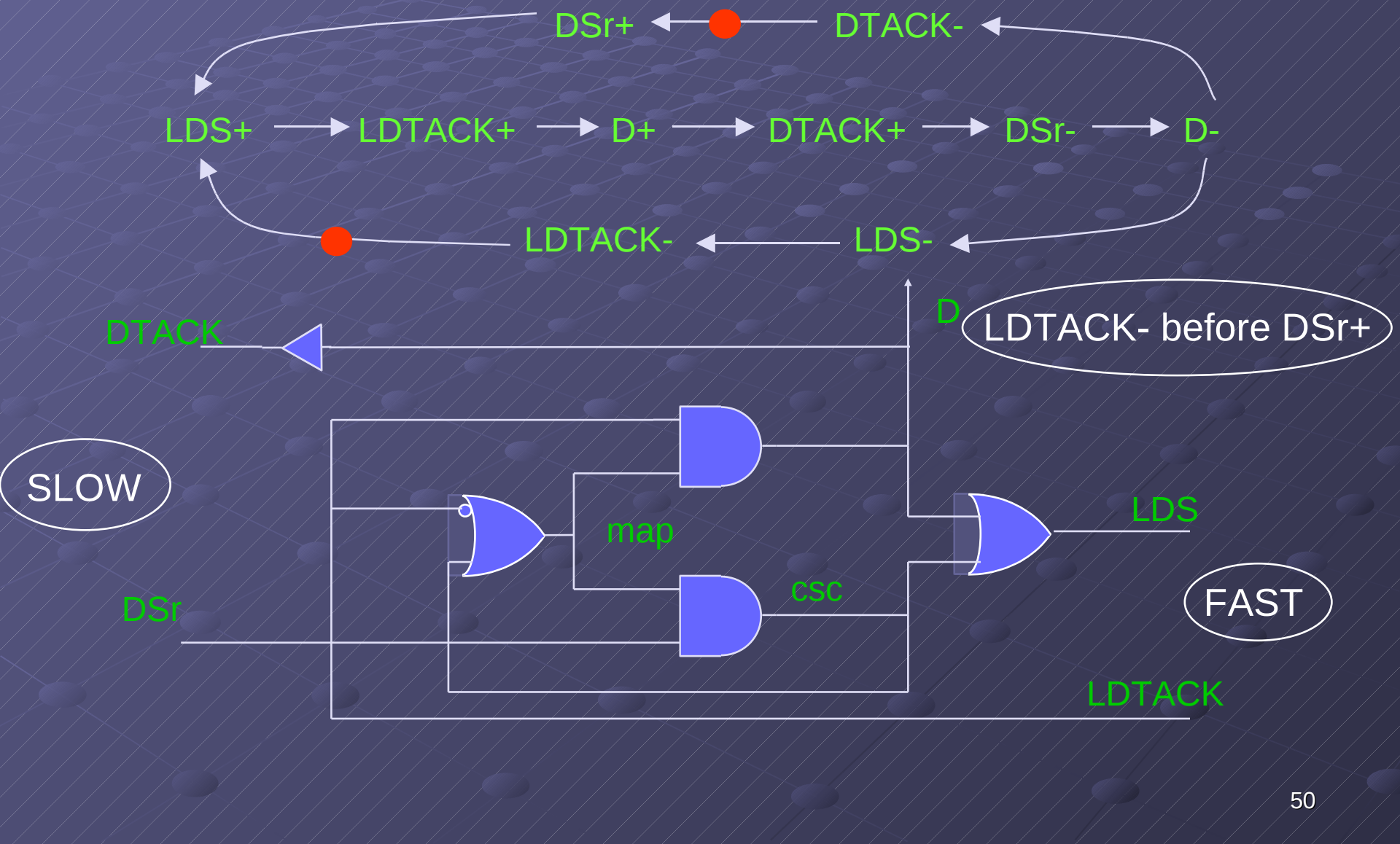




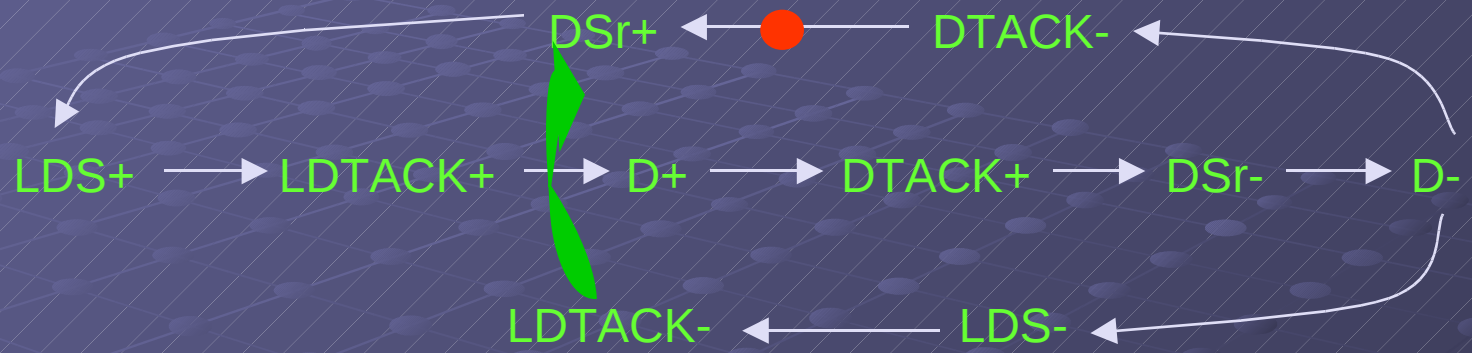
Speed-independent Netlist

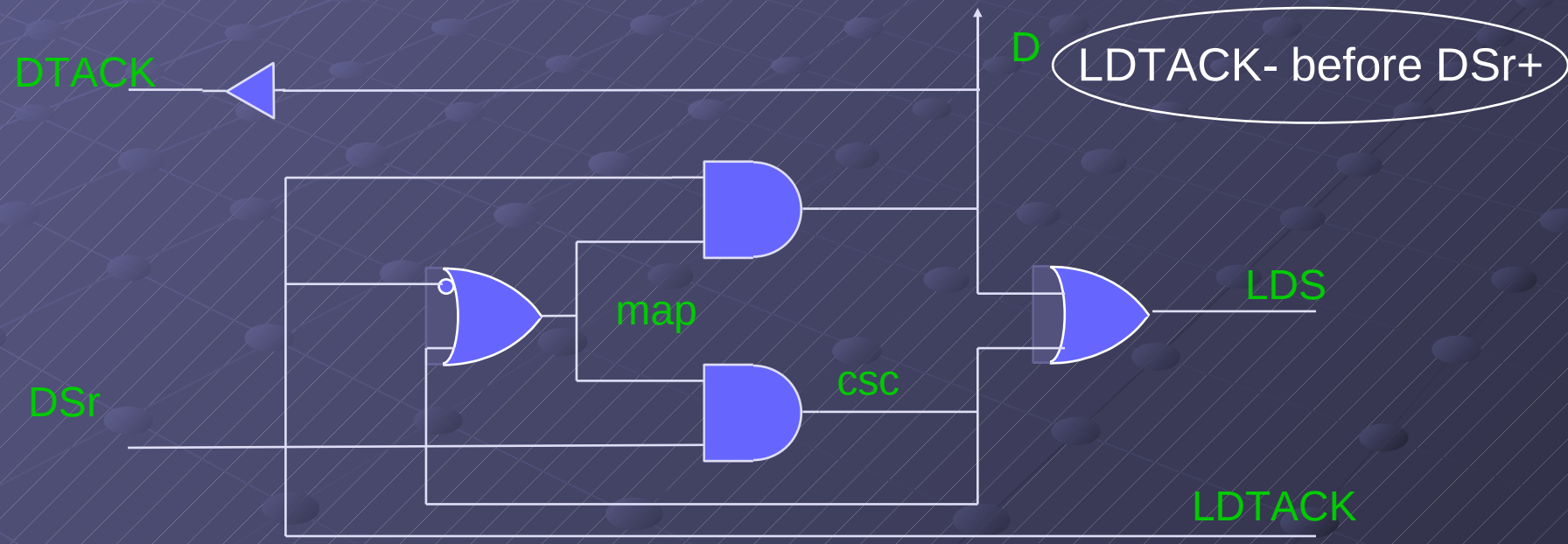


Adding timing assumptions

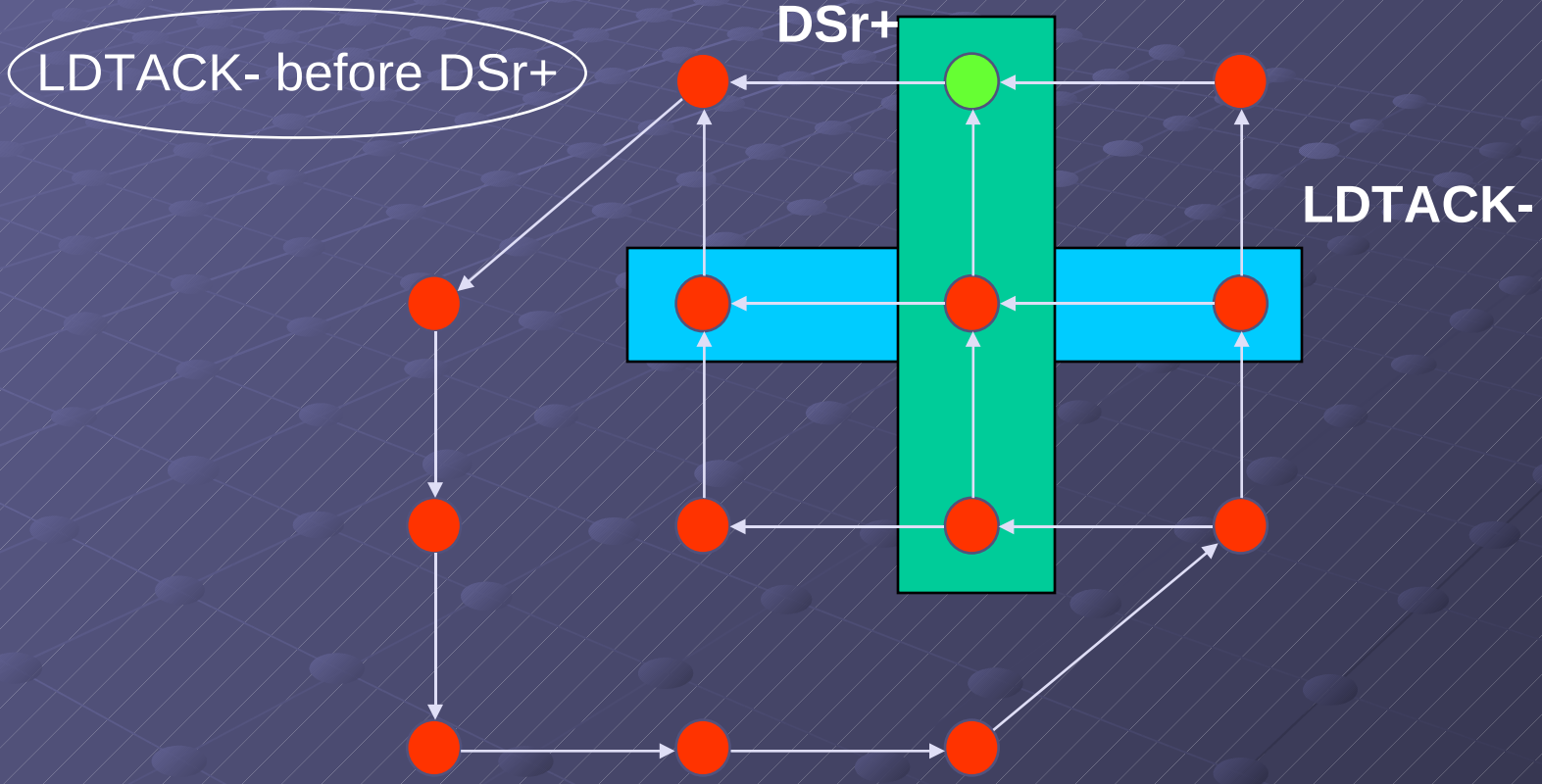


Adding timing assumptions

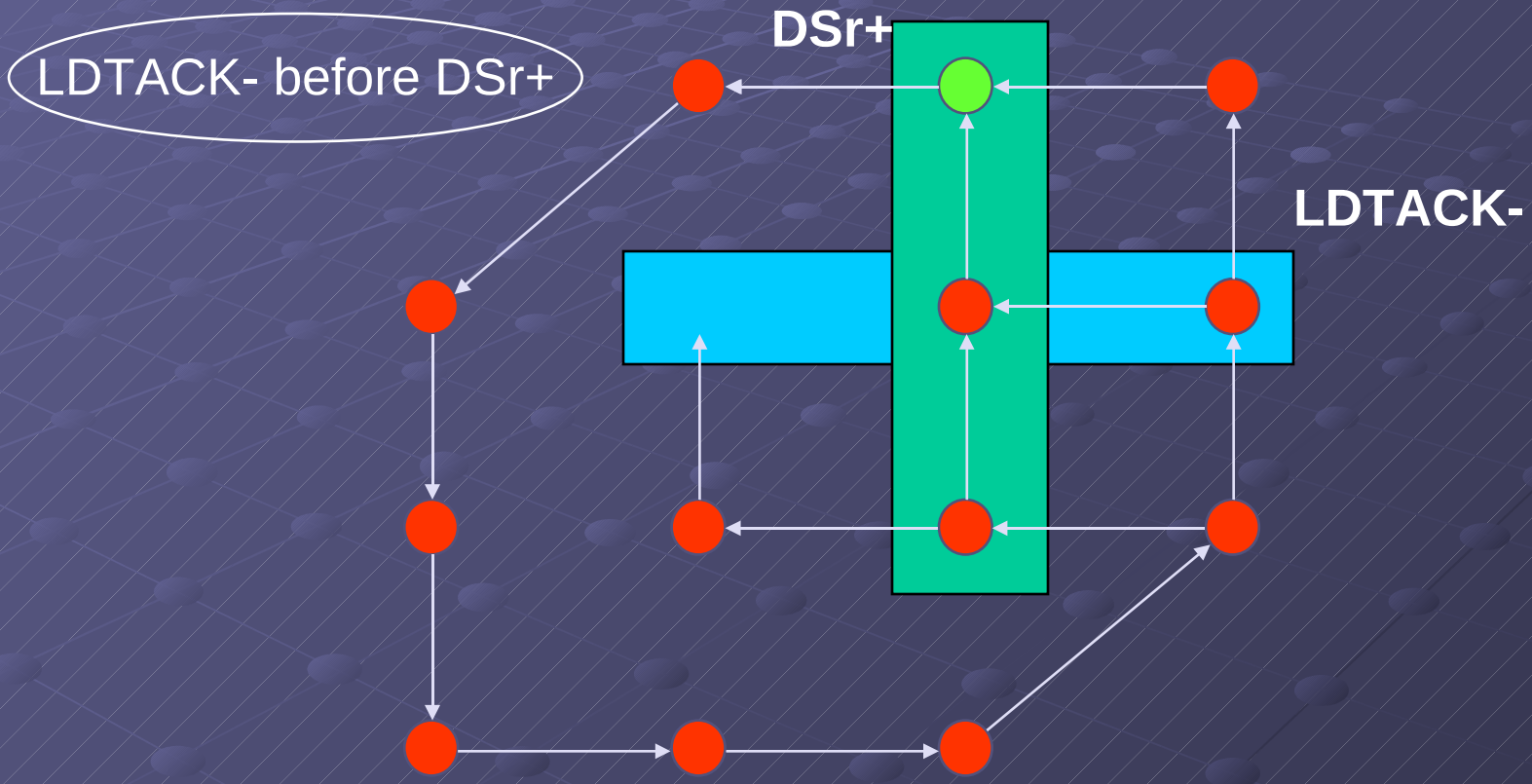




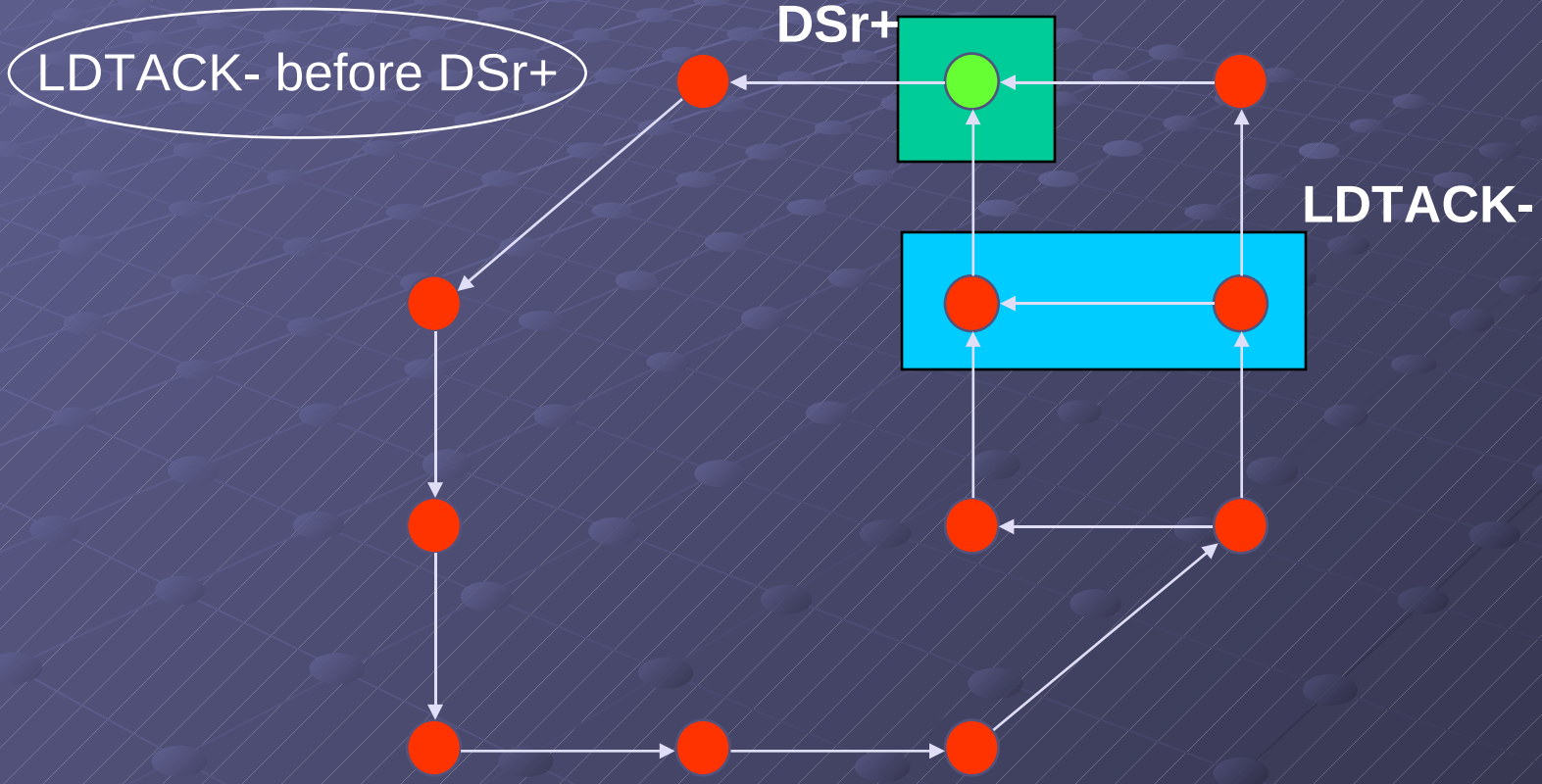
State space domain



State space domain



State space domain



Two more unreachable states

Boolean domain

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?

Boolean domain

LDS = 0

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

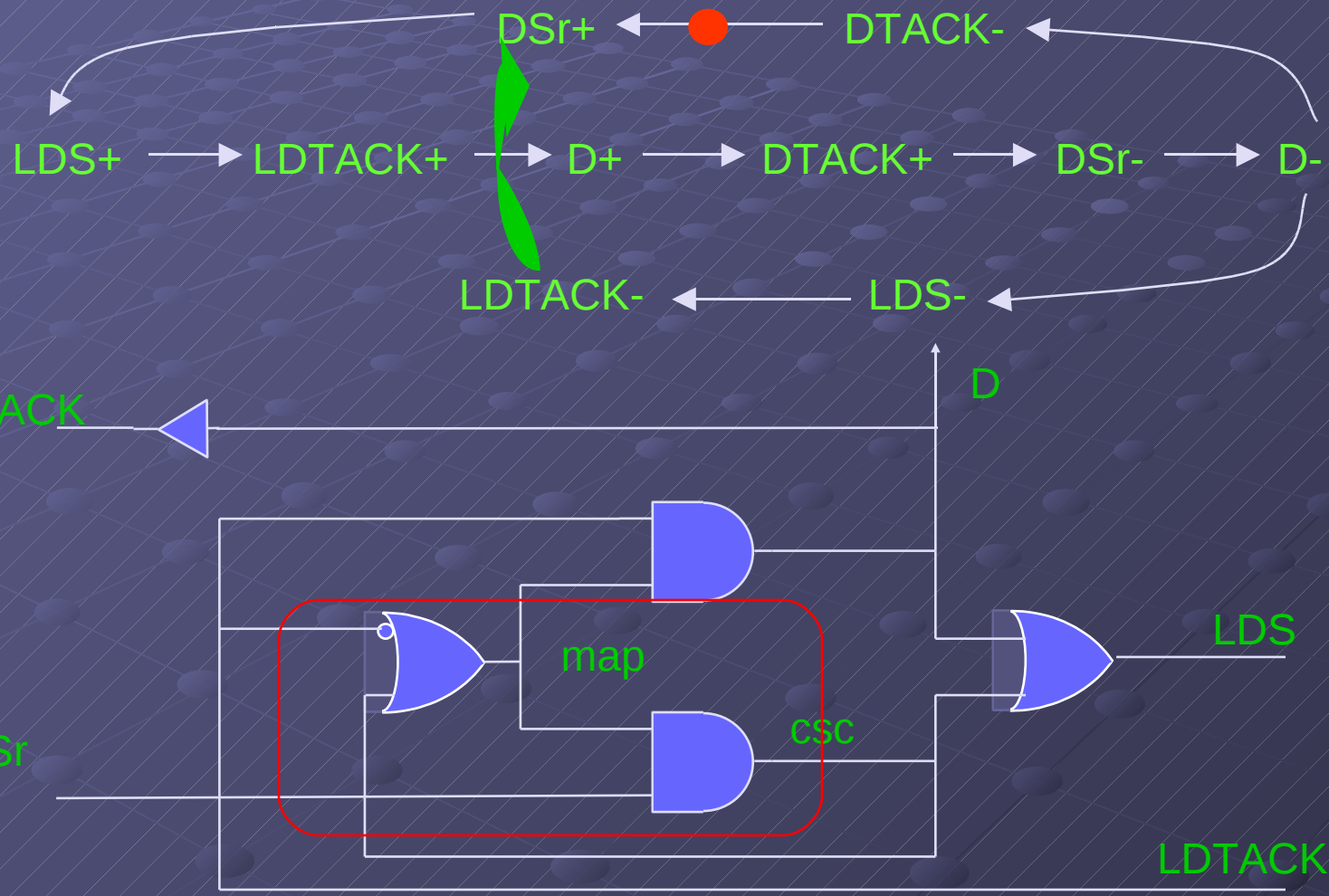
LDS = 1

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

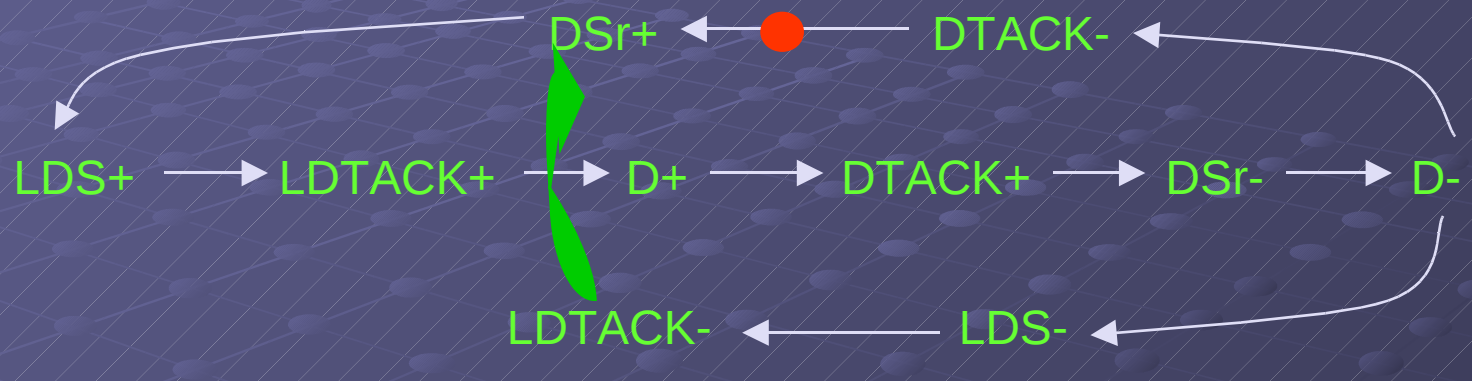
One more DC vector for *all* signals

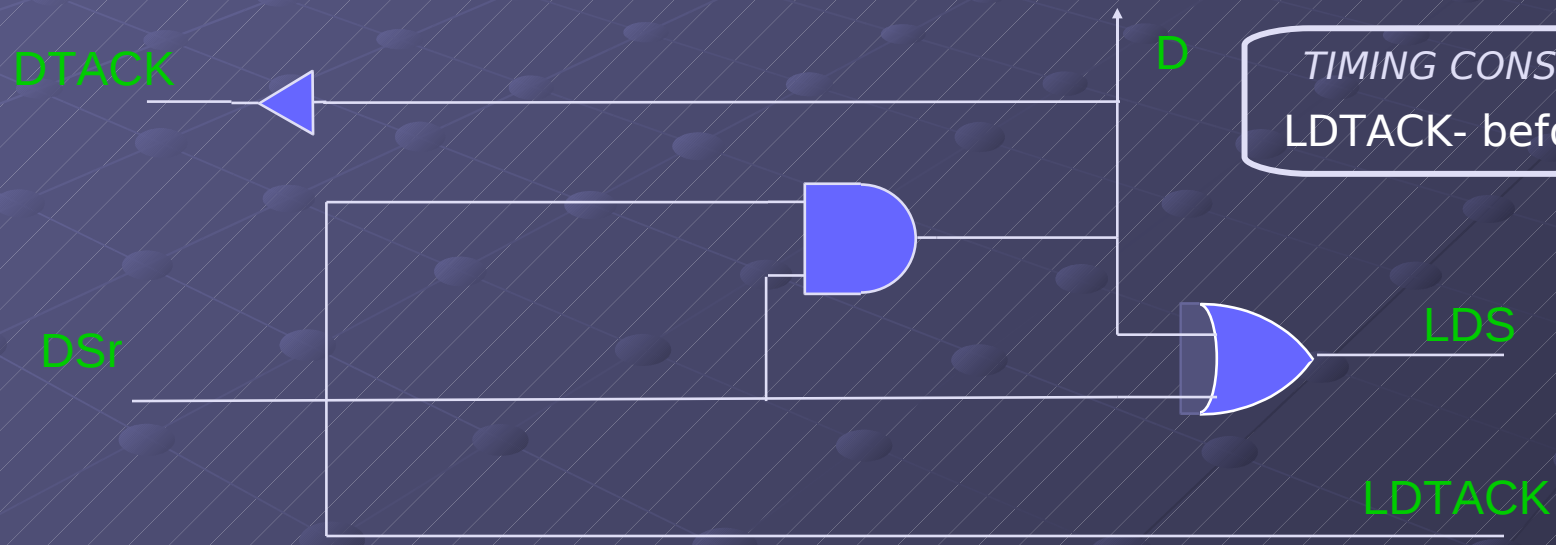
One state conflict is removed

Netlist with one constraint



Netlist with one constraint





TIMING CONSTRAINT

LDTACK- before DSr+

Conclusions

- STGs have a high expressiveness power at a low level of granularity (similar to FSMs for synchronous systems)
- Synthesis from STGs can be fully automated
- Synthesis tools often suffer from the state explosion problem (symbolic techniques are used)
- The theory of logic synthesis from STGs can be found in:

J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, *Logic Synthesis of Asynchronous Controllers and Interfaces*, Springer Verlag, 2002.