Logic Decomposition of Asynchronous Circuits in WORKCRAFT

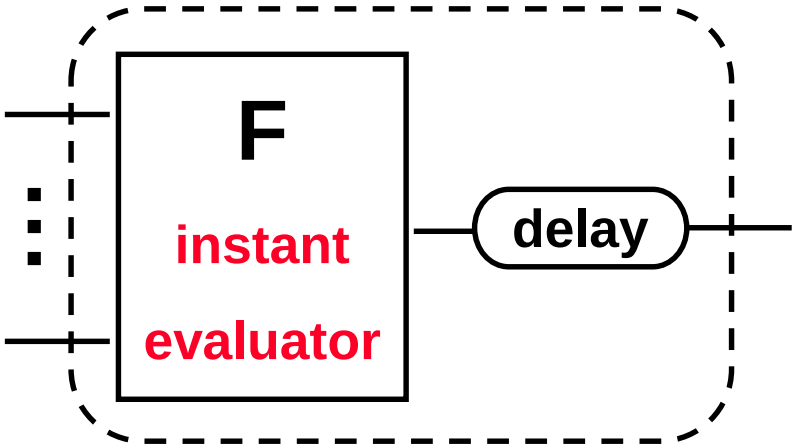
Victor Khomenko, Danil Sokolov, Alex Yakovlev

Motivation

- **Logic decomposition is one of the most difficult tasks in the design flow**
- **Much more difficult than for synchronous circuits – no guarantee of success**
- **The quality of the resulting circuit (in terms of area and latency) depends to a large extent on the way logic decomposition was performed**

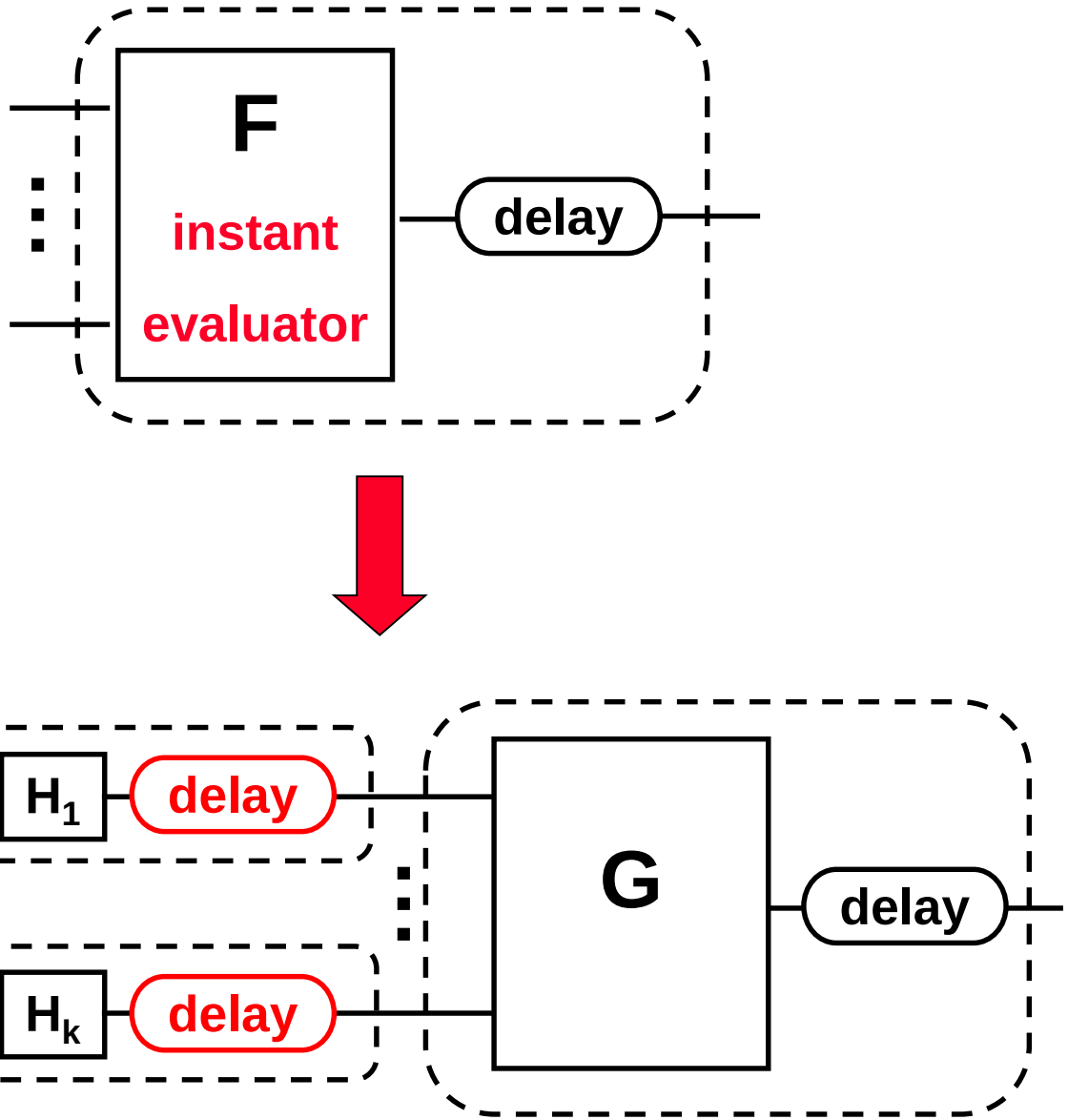
Speed-independency assumptions

- Gates are ***atomic*** (so no internal hazards)

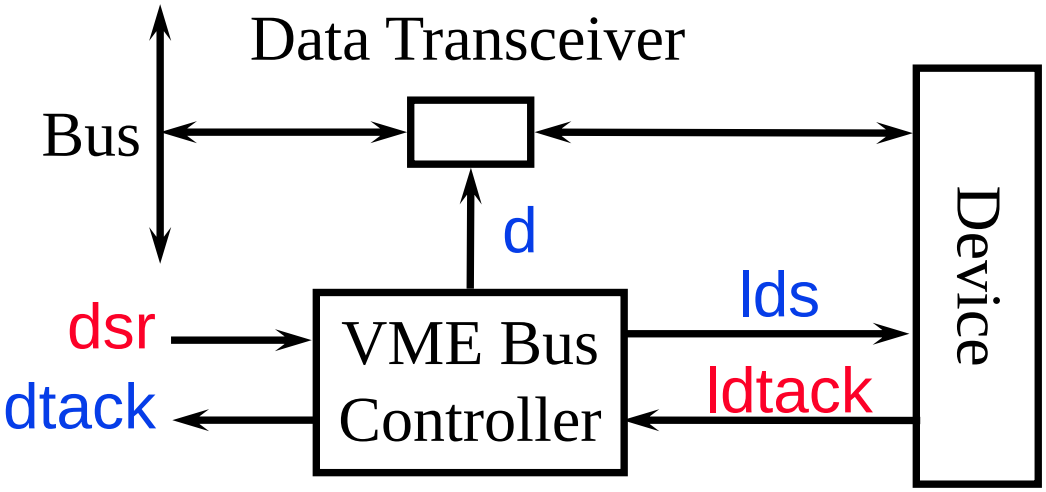


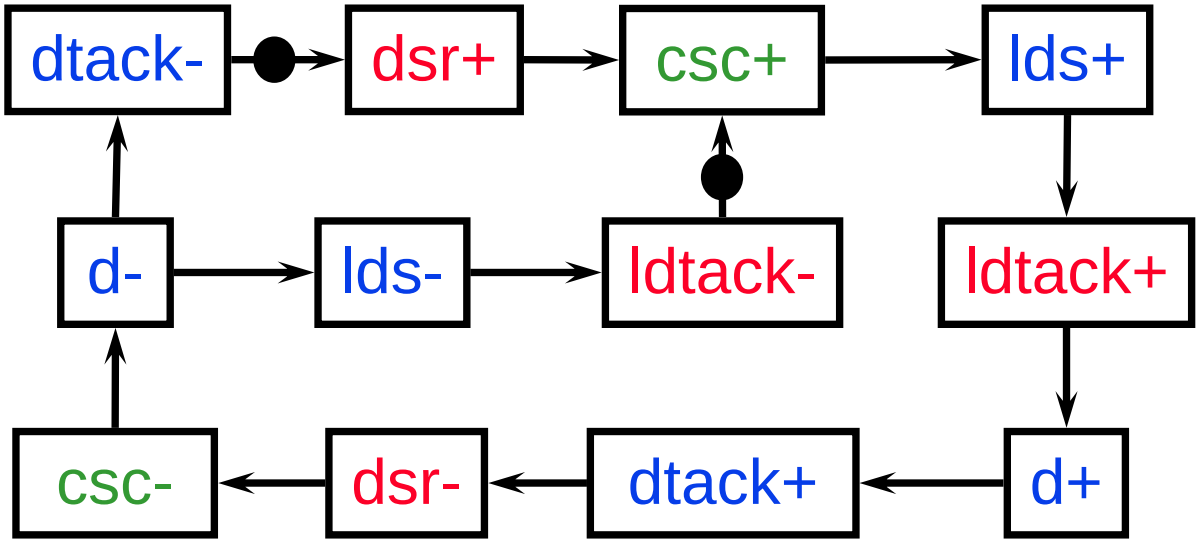
- Gates' delays are positive and unbounded (and perhaps variable)
- Wire delays are negligible (SI) or, alternatively, wire forks are isochronic (QDI)

Speed-independent decomposition

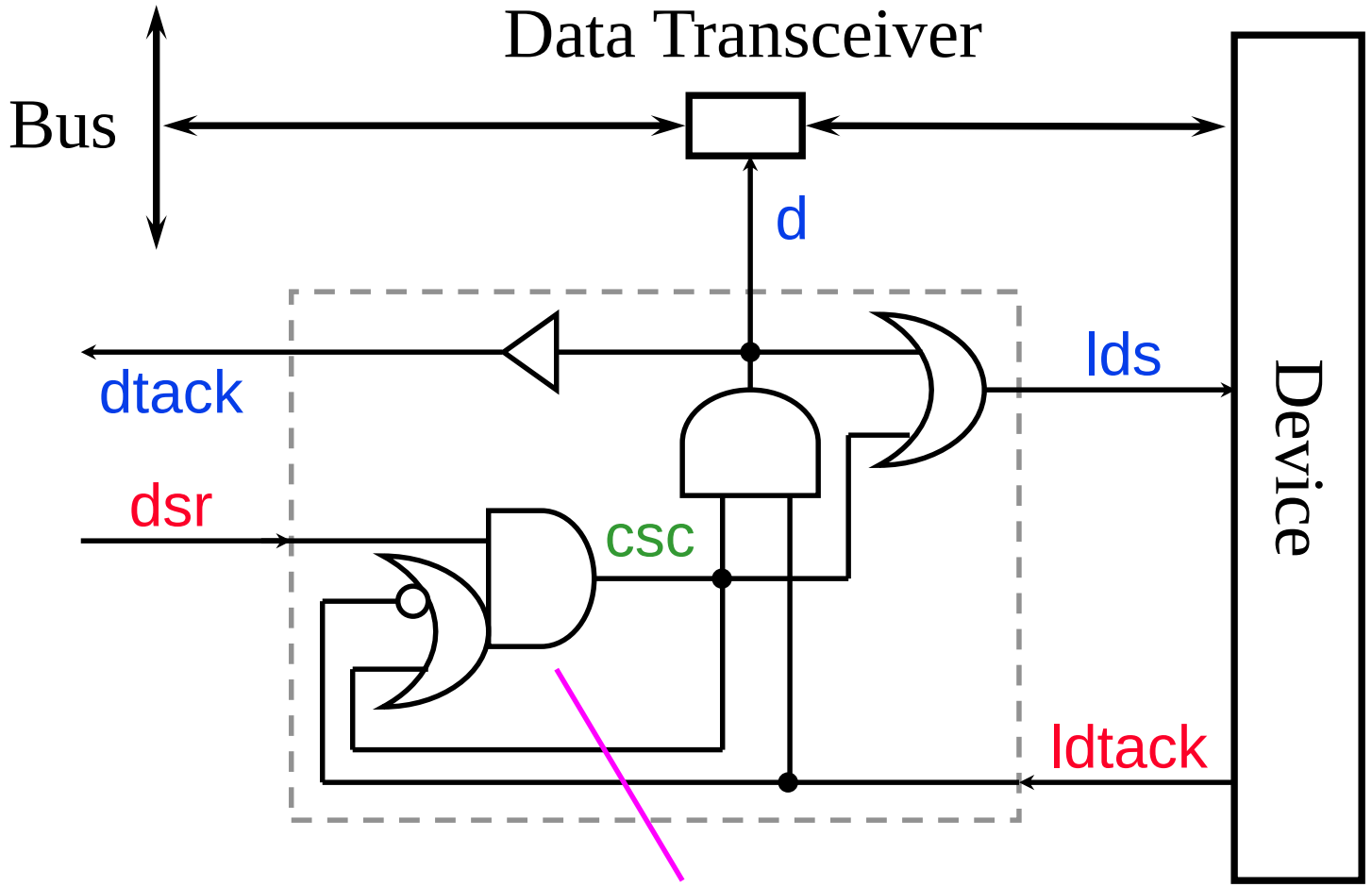


VME Bus Controller

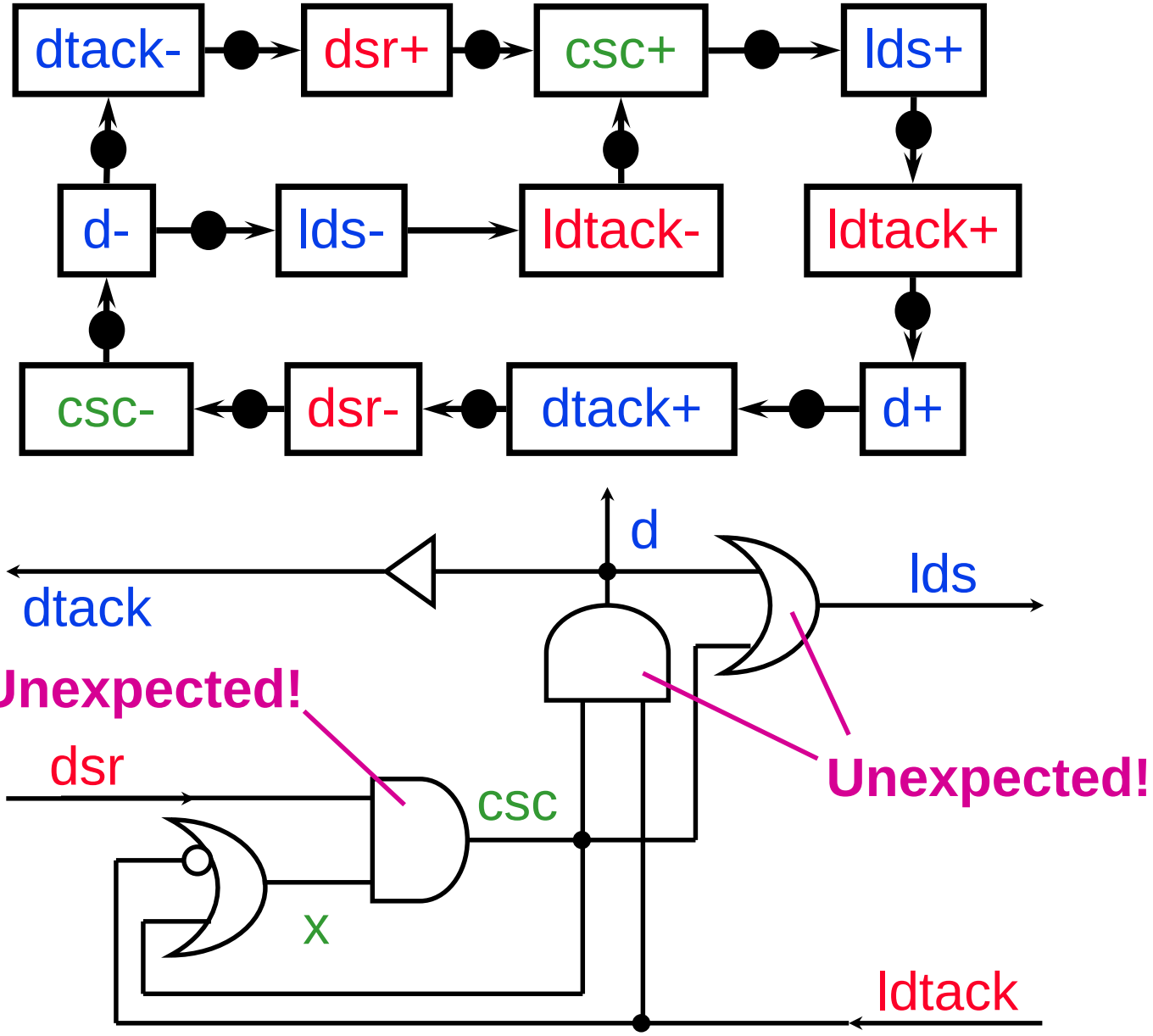




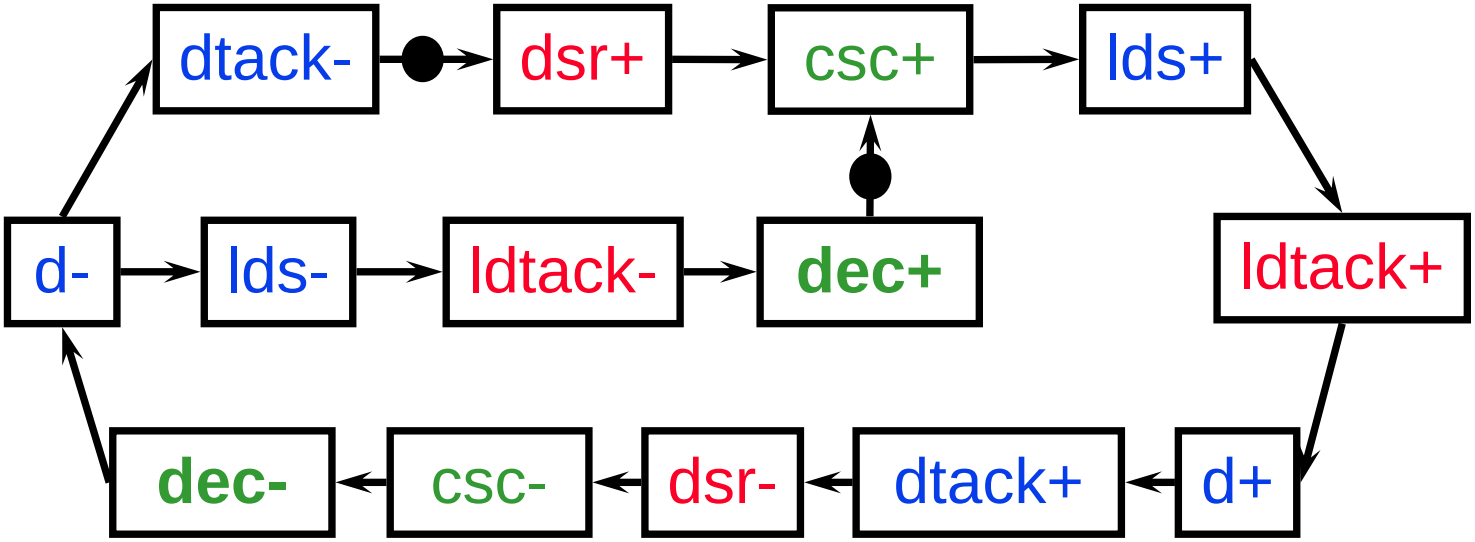
Complex-gate implementation

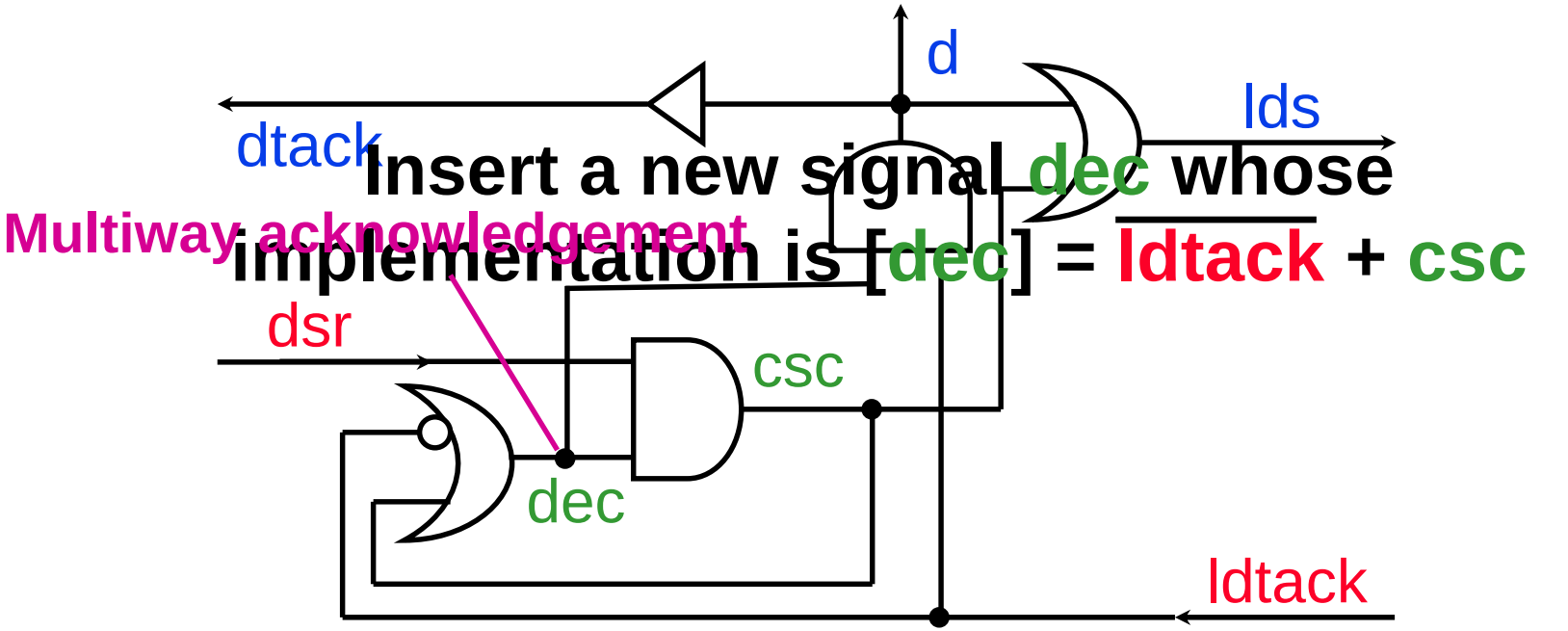


Naïve decomposition is hazardous



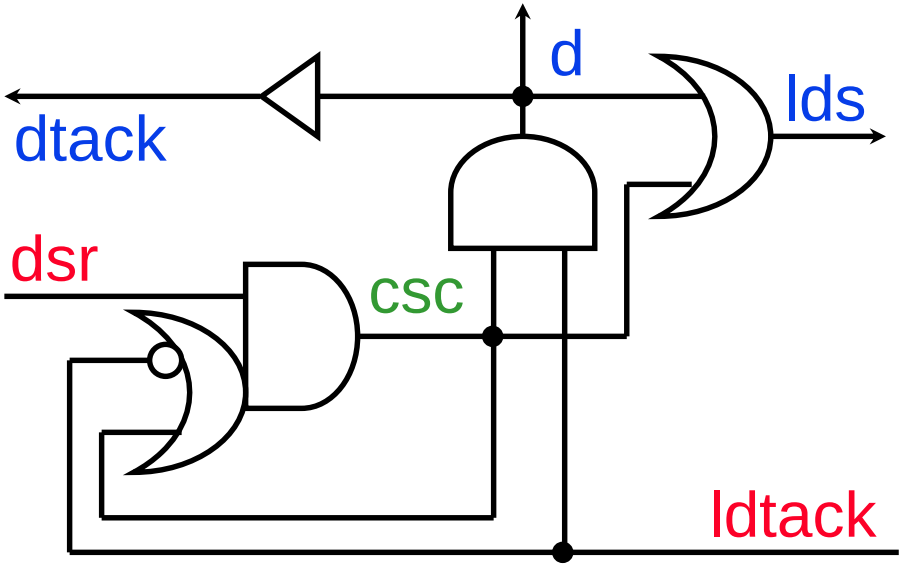
Decompose at the level of STG

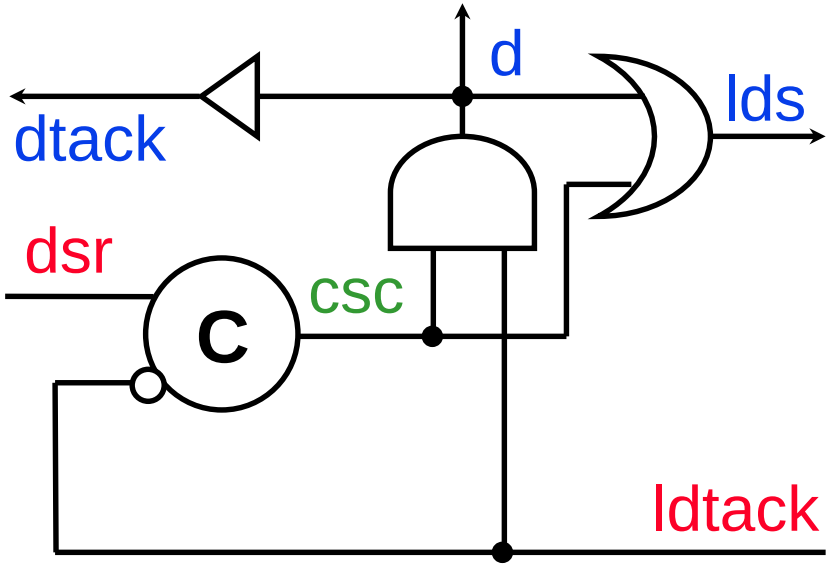




Insert a new signal dec whose implementation is [dec] = $\overline{\text{ldtack}}$ + csc

Latch utilisation





Only possible because there is no globally reachable state at which dsr=ldtack=0 and csc=1

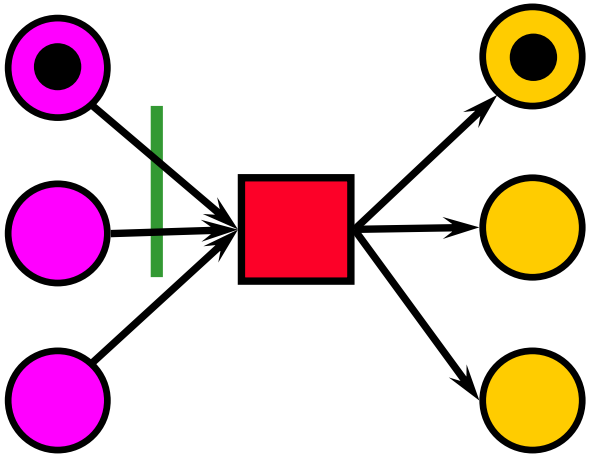
Logic decomposition algorithm

- Synthesise the circuit from the STG (several complex-gate and standard-C implementations are considered for each signal)
- Heuristically select a non-mappable gate, and a decomposition of this gate
- Insert a new signal into the STG for the sub-function in the selected decomposition
- Repeat the above steps until all gates are mappable or no further progress is possible

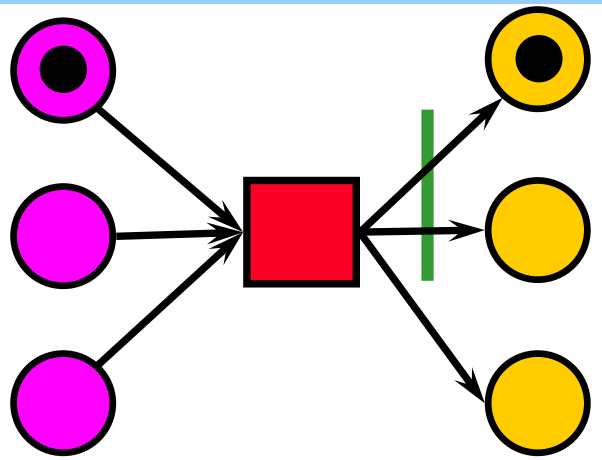
Function-guided signal insertion

Problem: given a Boolean function **F**, insert a new signal **dec** (i.e. a set of new transitions labelled **dec+** or **dec-**) with the implementation **[dec]=F** into the STG

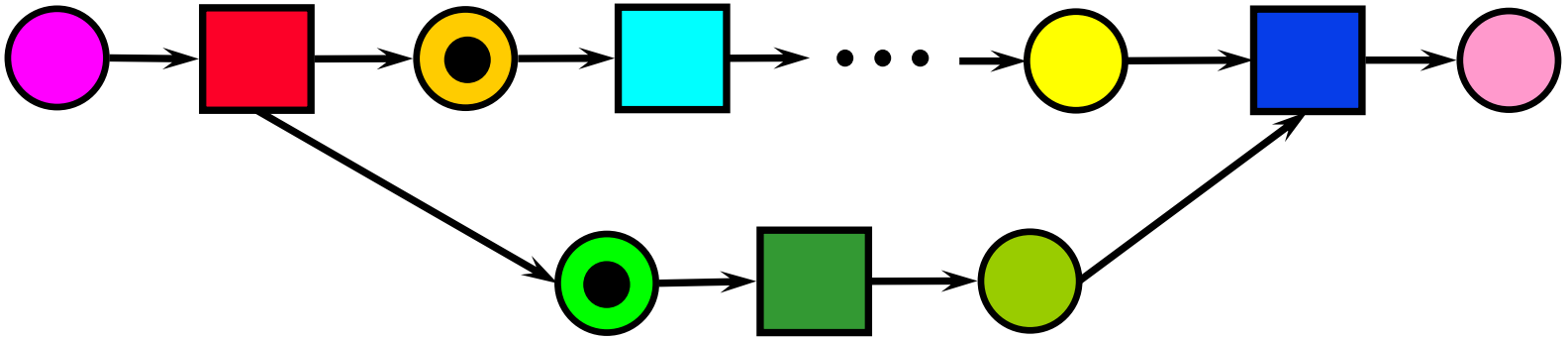
Transition insertions



Sequential pre-insertion

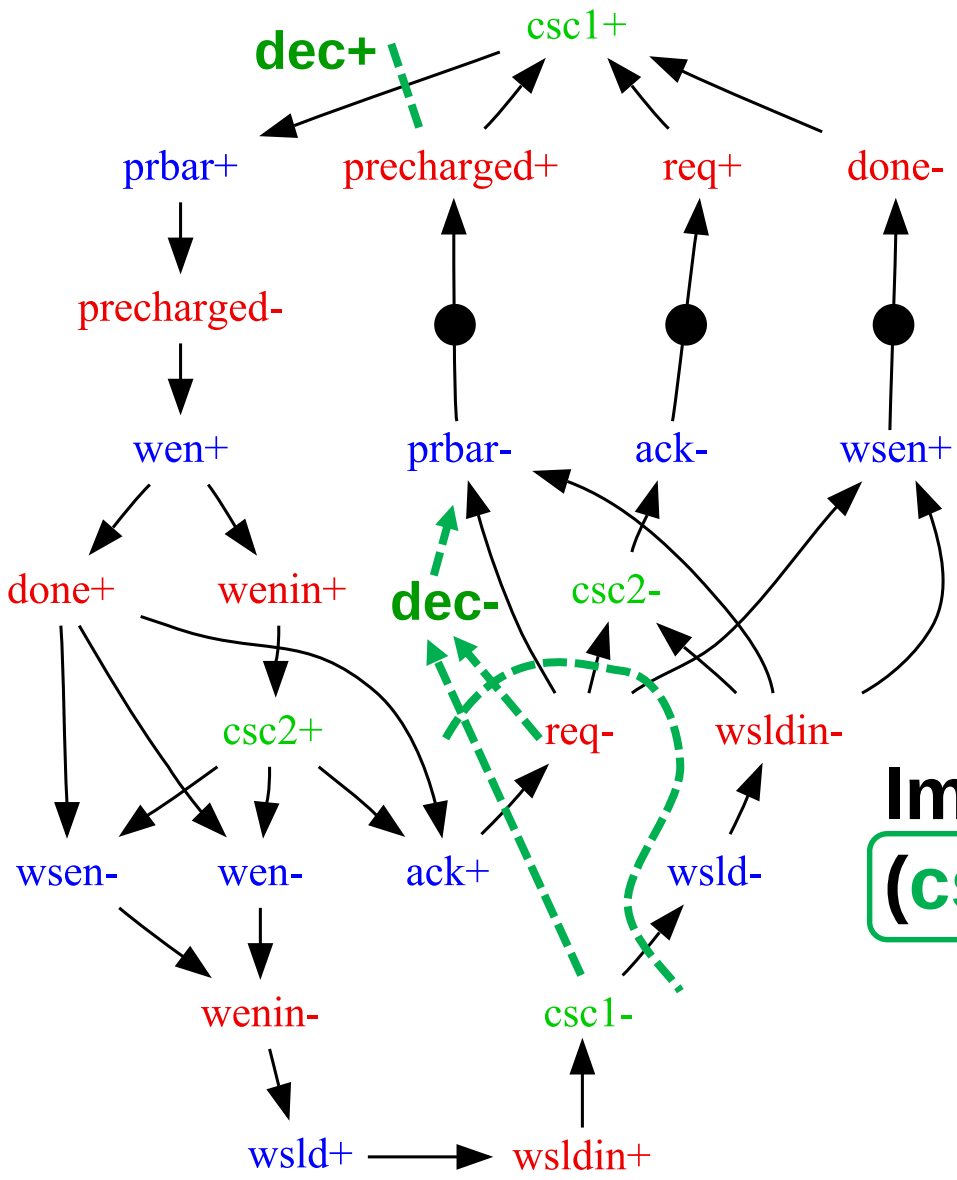


Sequential post-insertion

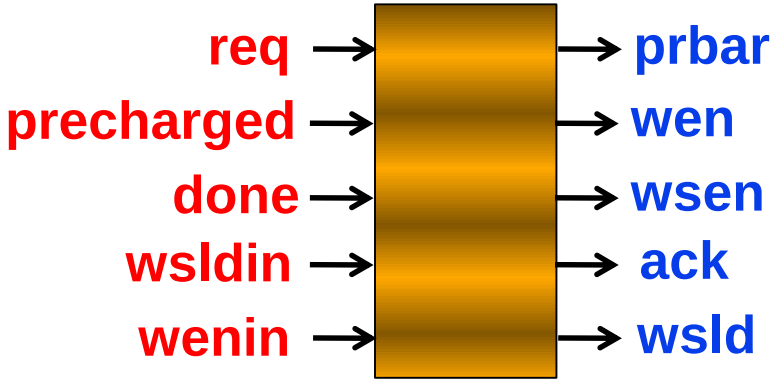


Concurrent insertion

Example: imec-sbuf-ram-write



imec-sbuf-ram-write

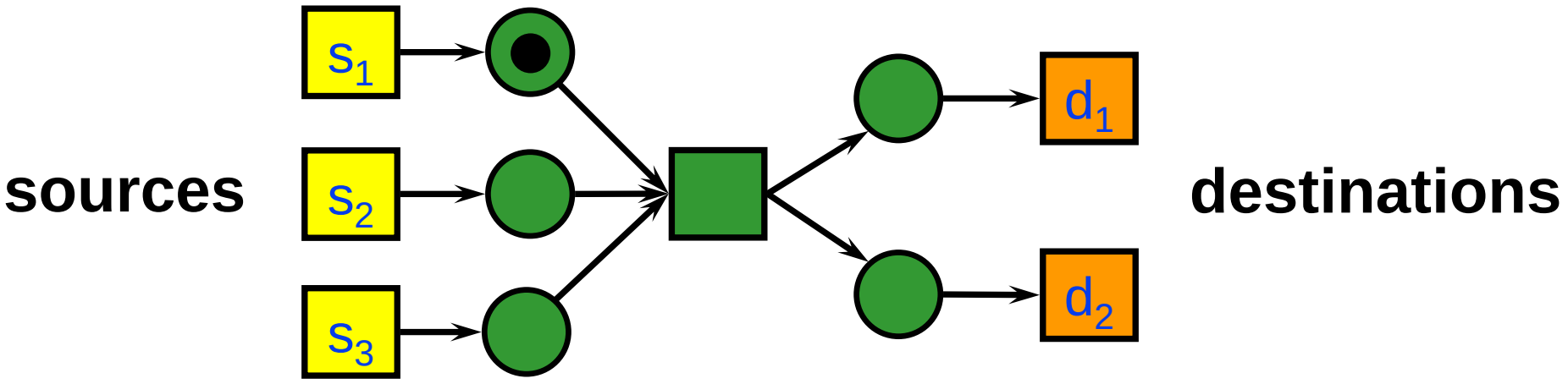


Implementation of prbar:

$$(csc2 \wedge req) \vee csc1 \vee wsldin$$

dec

Generalised transition insertion



Sources and destinations are locked

Cost function

Parameterised by the user; takes into account:

- **the delay introduced by the insertion**
- **the number of syntactic triggers of all non-input signals**
- **the number of inserted transitions of a signal**
- **the number of signals which are not locked with the newly inserted signal**
- **…**

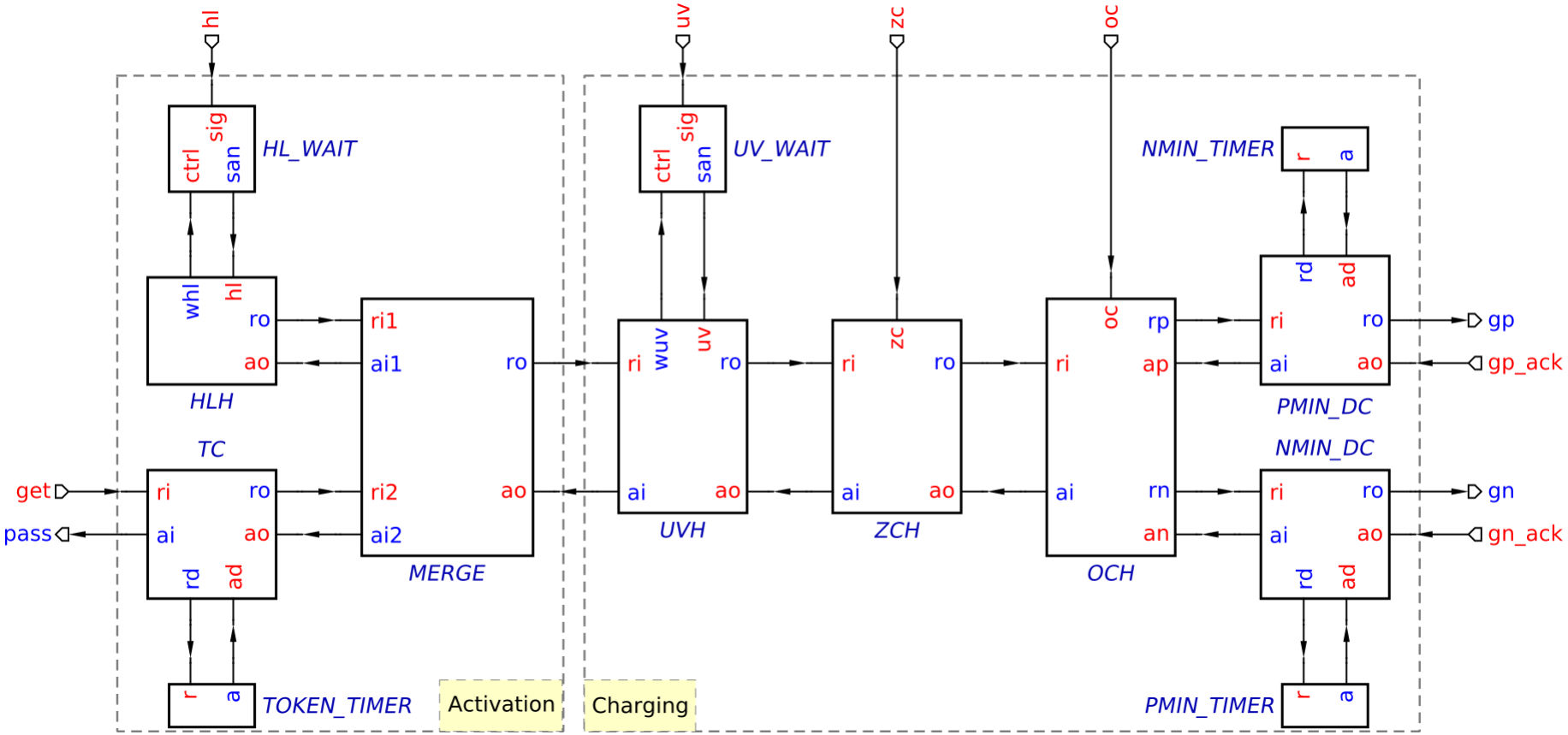
Overcoming mapping failure

- Logic decomposition is not guaranteed to succeed, so tools occasionally fail
- May need to help the tools:
  - methods & tricks
  - "think outside the box" – knowledge of the environment, capacity to redesign the system and its environment
  - "high-level understanding of the design" – knowing the causal dependencies between the signals, which environment signals are fast/slow (useful for concurrency reduction), etc.
  - relative timing assumptions

0 Prevention is better than cure

- Large monolithic STGs are difficult, both for humans and for tools
- Hierarchical design:
  - architectural decomposition into modules
  - … until each module is small, say ~10 signals (this size is about right for humans* and tools)
  - Advantages: human- and tool-friendly, more predictable, module re-use (within and between designs), easy to document and maintain, etc.
- Workcraft has support for hierarchical designs

Example: stage of multiphase buck



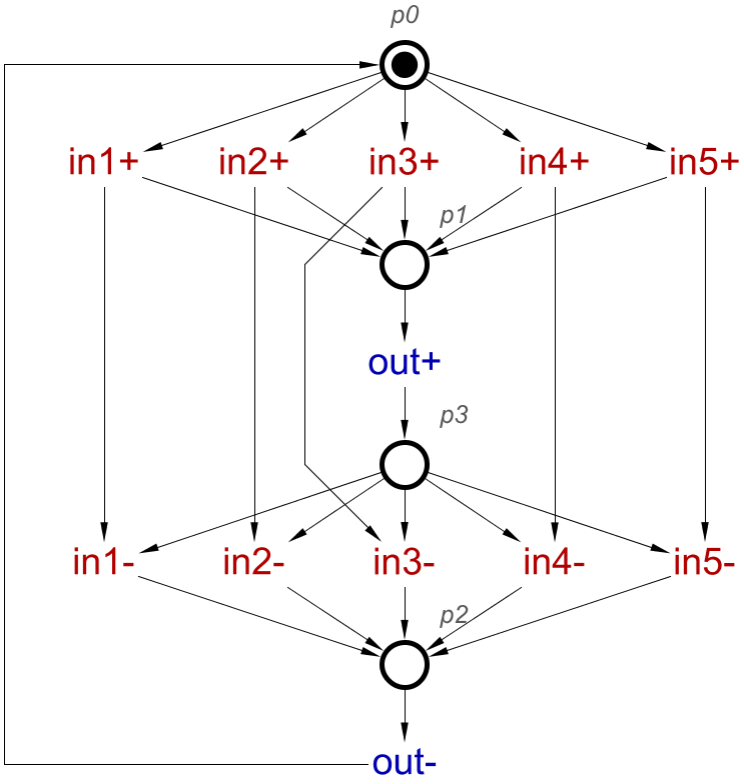
1 Expanding gate library

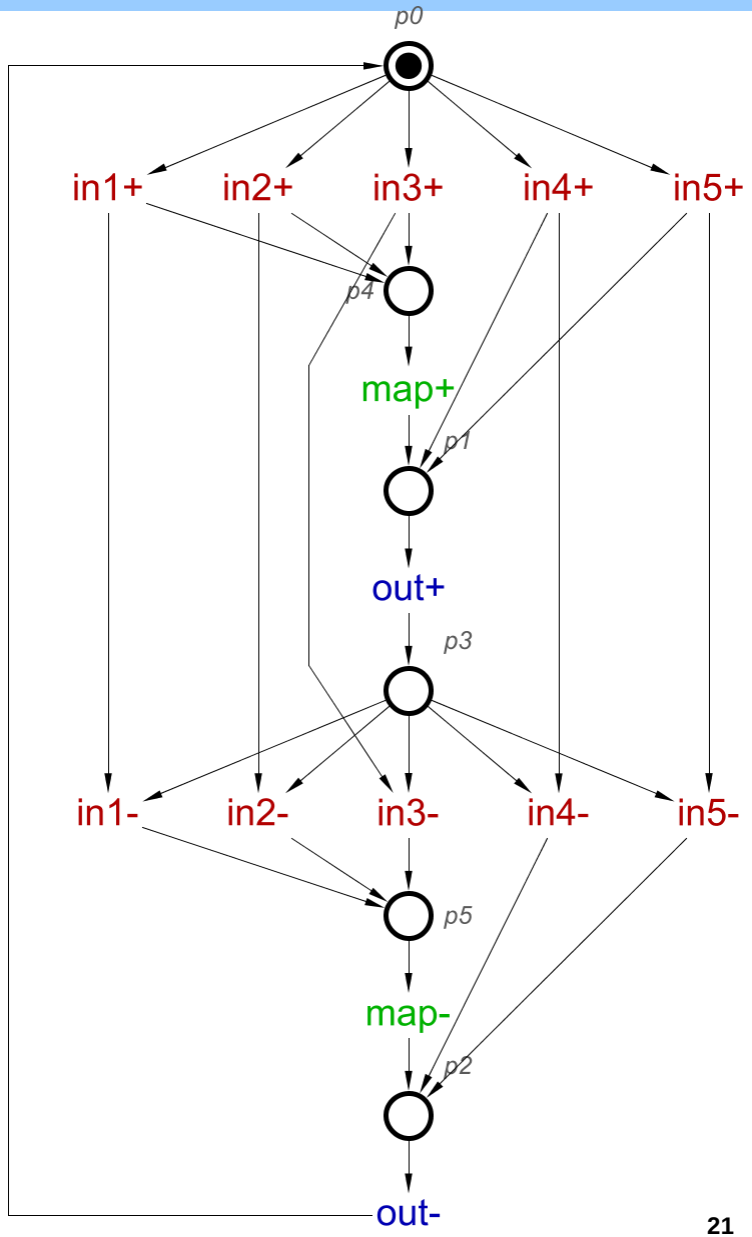
- **Add a missing gate to the library**
- **Usually not an option ☹**

2 Inserting a useful signal

- Tools often fail because:
  - some heuristic selects a bad sub-function
  - there is no structural signal insertion to implement a useful sub-function
- One can help the tool by inserting an internal signal implementing a useful sub-function

Example: OR5

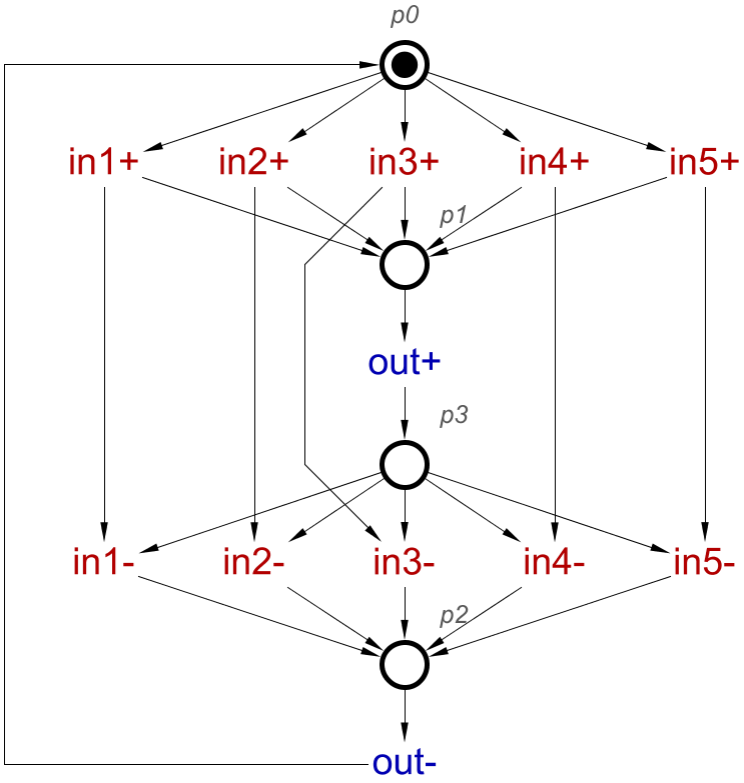

p0
in1+
in2+
in3+
in4+
in5+
p1
out+
p3
in1-
in2-
in3-
in4-
in5-
p2
out-

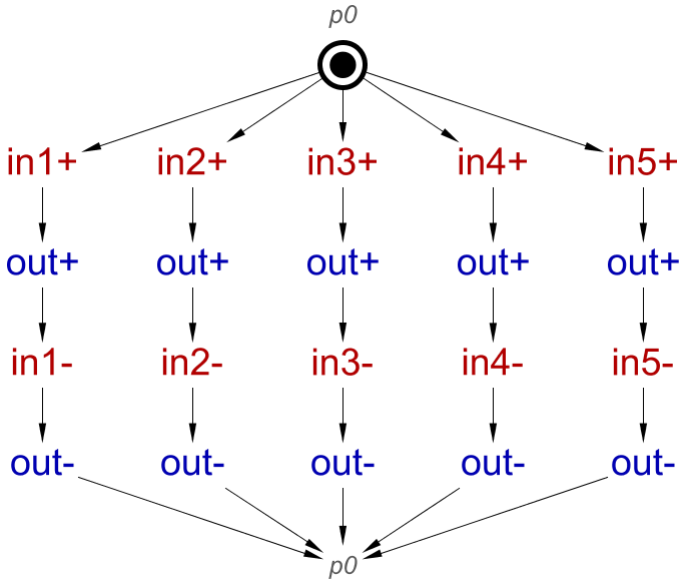

p0
in1+
in2+
in3+
in4+
in5+
p4
map+
p1
out+
p3
in1-
in2-
in3-
in4-
in5-
p5
map-
p2
out-

3.1 Simplifying the STG structure

- **If the STG has complicated structure, it may be impossible to insert a signal structurally (e.g. one would have to merge and then split some choice branches for that)**
- **Try to simplify the STG structure by reducing the number of choice and merge (i.e. explicit) places, in particular controlled choices can often be removed**

Example: OR5


p0
in1+
in2+
in3+
in4+
in5+
p1
out+
p3
in1-
in2-
in3-
in4-
in5-
p2
out-


p0
in1+
in2+
in3+
in4+
in5+
out+
out+
out+
out+
out+
in1-
in2-
in3-
in4-
in5-
out-
out-
out-
out-
out-
p0

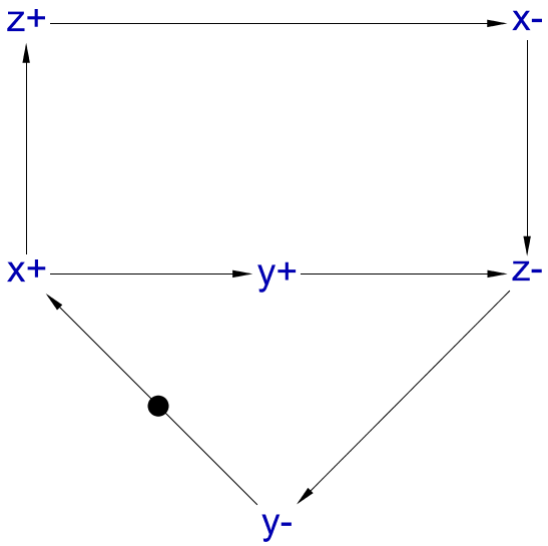
3.2 STG re-synthesis

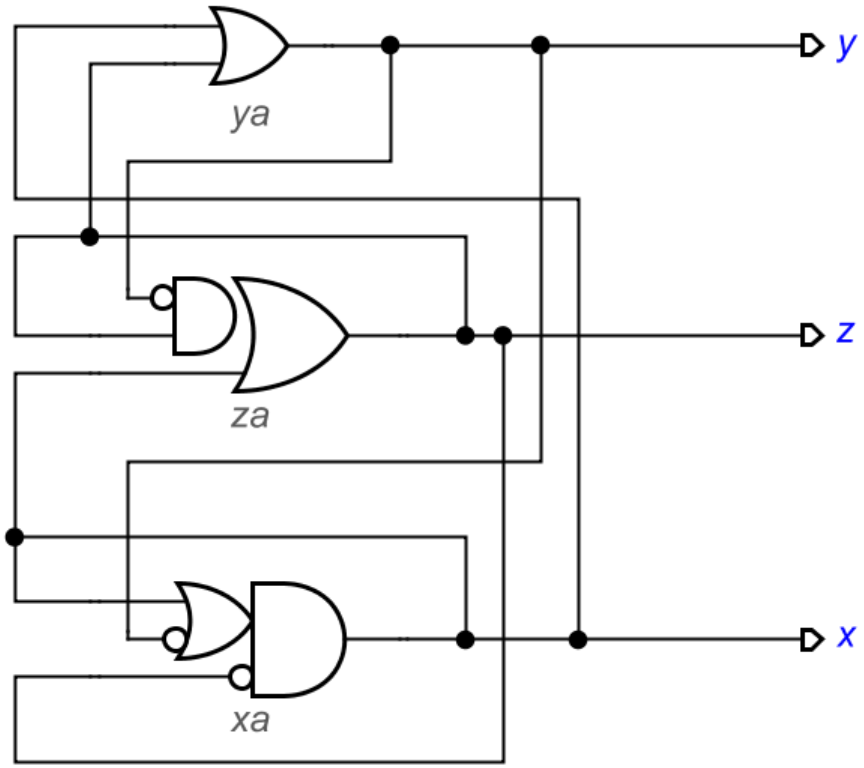
- Re-synthesis builds the state graph and then derives an equivalent STG from it, often with simpler structure
- Fully automatic, so easy to try if technology mapping fails
- Try various command-line options

4 Concurrency reduction

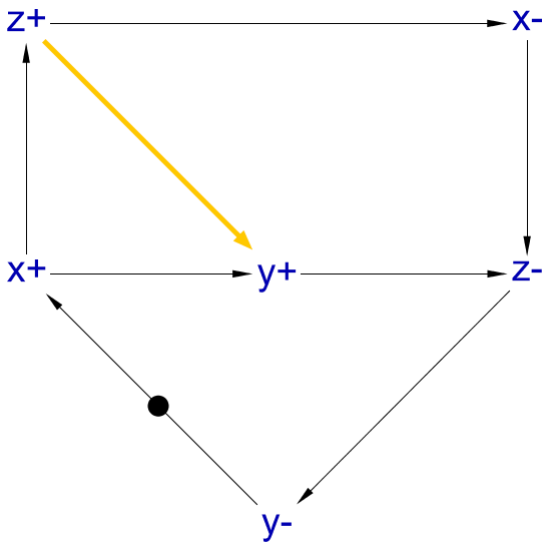
- **CR does not necessarily decrease performance – though events are less concurrent, the gates become smaller and some internal signals may become unnecessary**
- **CR may change the contract with the environment and introduce a deadlock or global deterioration of performance that is difficult to debug**

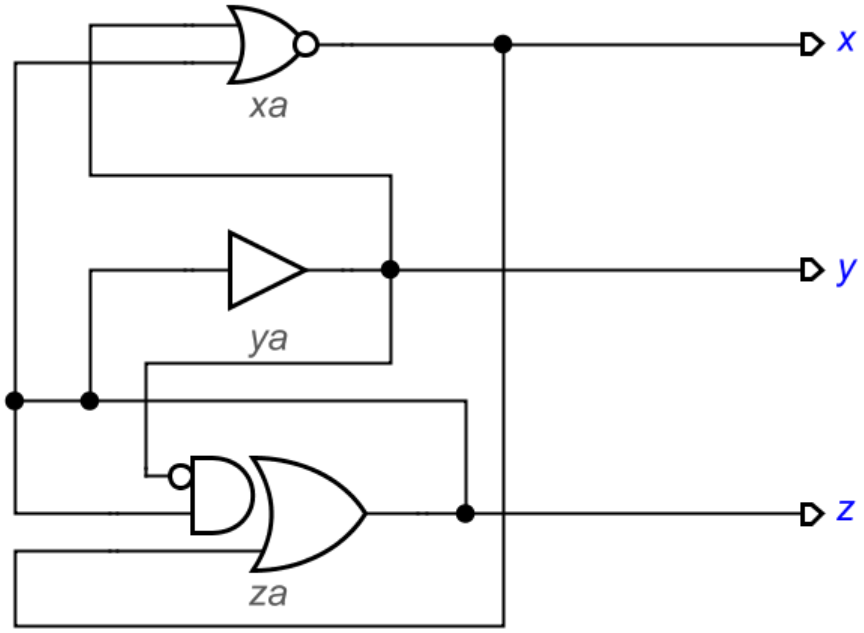
Example: xyz



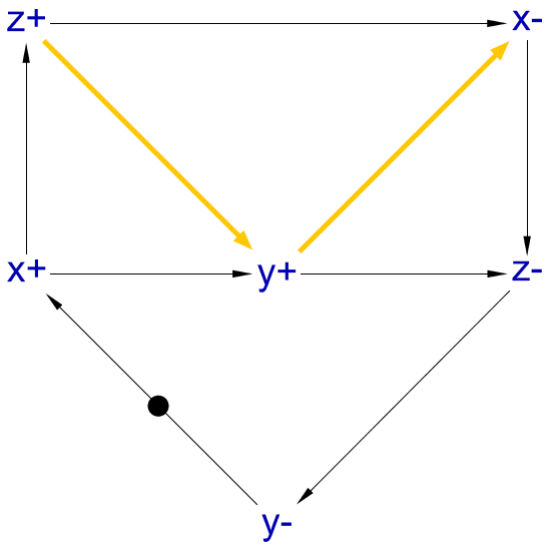


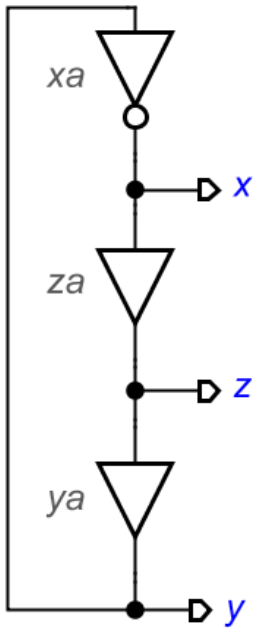
Example: xyz with CR





Example: xyz with more CR

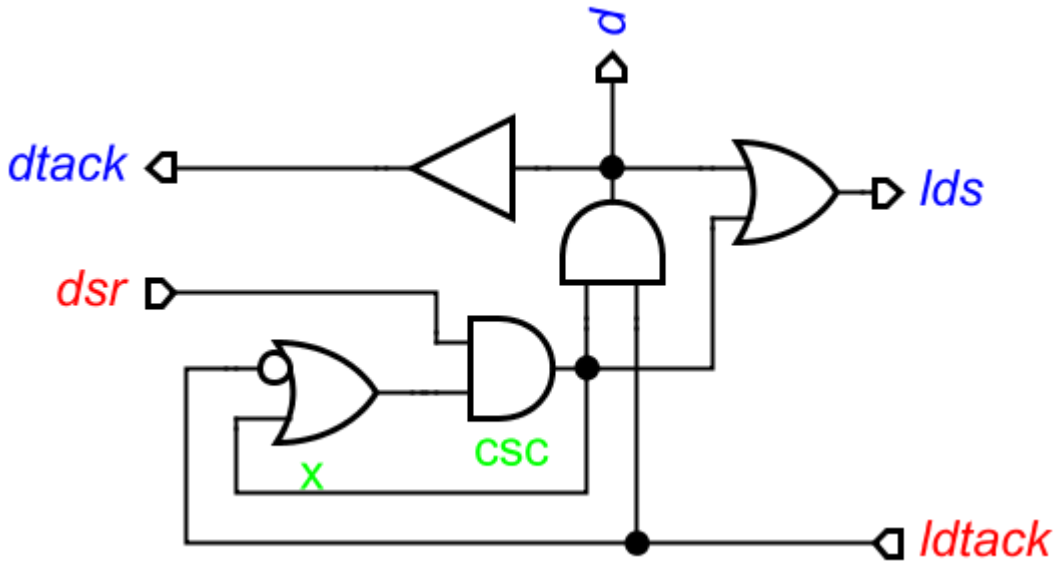




5 Relative timing assumptions

- Occasionally, the described techniques still fail to yield a solution
- Breaking up a large gate yields a non-speed-independent decomposition
- The correct operation can then be ensured by relative timing assumptions
- This has implications for place&route
- Easy to make a mistake, need tool support

Example: VME read phase



$$\mathtt{MaxDelay(x-)} < \mathtt{MinDelay(d-} \rightarrow \mathtt{lds-)}$$

$$\mathtt{MaxDelay(x-)} < \mathtt{MinDelay(d-} \rightarrow \mathtt{dtack-} \rightarrow \mathtt{dsr+)}$$

Thank you!
Any questions?