HY 590.20

Χρονισμός Ψηφιακών Συστημάτων
Χρήστος Σωτηρίου
7

Control Circuits

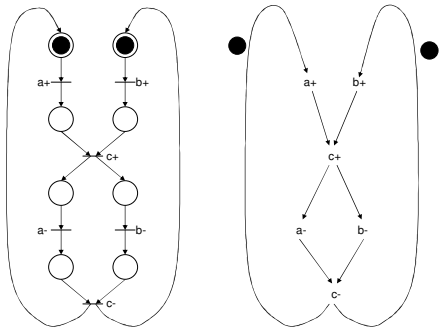
- Many methods, tools, assumptions, frameworks
- Balsa generated both control and data-path
- We now consider one method / tool for control-only spec, verification and synthesis:
  - Assumption: Speed independent control for speed independent bundled data
  - Method: Signal Transition Graph
    - A special type of Petri net : FC-PTnet
  - Tool: **Petrify** (mostly from Jordi Cortadella at UPC)

Delay Models

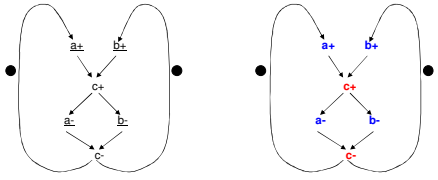
- Fixed delay: d=c
- Min-max delay: choose d∈[m,M]
  - Max delay: d∈[0,M]
  - Low-bounded delay: d∈[m,∞)
- Unbounded delay: d∈[0,∞)

- Inertial delay: Glitches are filtered.
  - We don't assume intertial delays in async control design

Petri net, Signal Transition Graph (STG)



Separating Inputs, Outputs and Internals

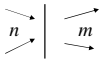


In the STG:

- PN transition are signal transitions
- PN places and arcs are causal relations
- Simple places (one arc in, one arc out) are omitted
- MARKING
  - ≡ assignment of tokens to places
  - ≡ state of the circuit

Token Preservation (from Lecture #1)

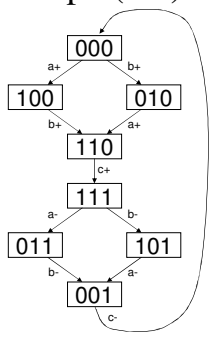
- Tokens do not disappear
- Tokens do not appear (from nowhere)
- One token does not overtake another
- A transition with *n* inputs and *m* outputs:
  - waits for *n* tokens on inputs
  - Generates *m* tokens on outputs



7

State Graph (SG)

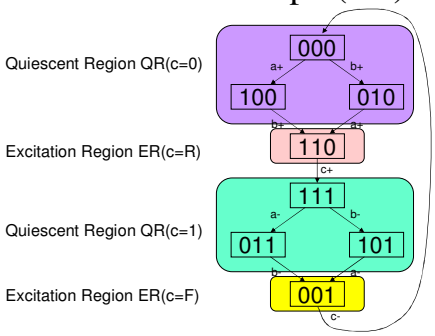
State: abc



- SG more complex than STG
- Bad for design spec
- Needed for synthesis

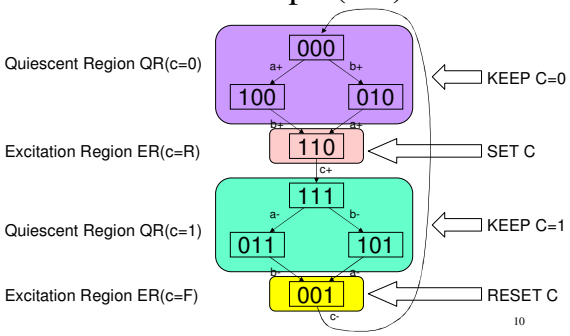
8

State Graph (SG)



9

State Graph (SG)



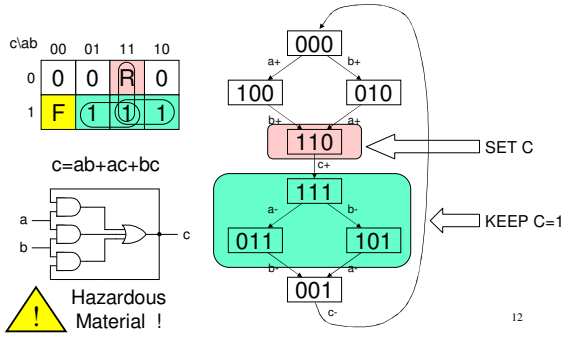
10

Synthesis: C Element

- Two methods:
  - Using gates
  - Using SR latch

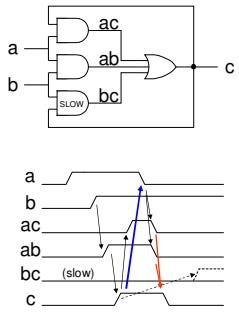
11

Synthesis: C Element using gates



12

Hazardous Conditions



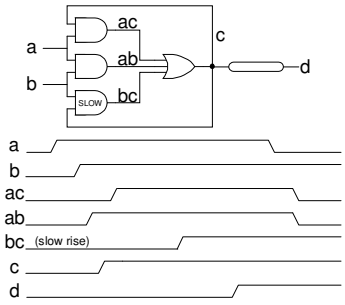
13

Hazardous Conditions

- Hazards are not an issue in sync circuits
- **Hazards can be deadly in (some) async circuits**
- We **can** build hazard-free circuits
- We can impose timing assumptions and add delays

14

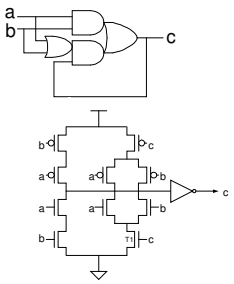
Hiding Hazards Behind Delays



Do you like slowing your circuit ?

15

Avoiding Hazards with Complex Gates

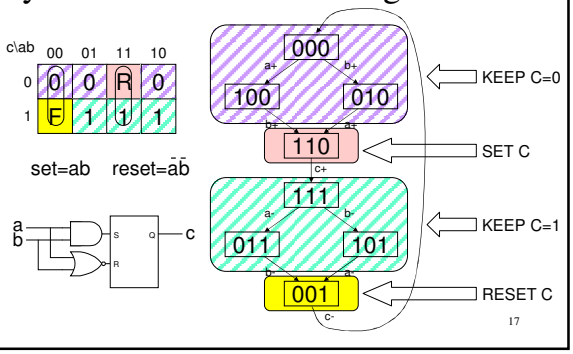


•Theoretically, same hazard problem (e.g. T1 slow)

•Must use caution or delay output

16

Synthesis: C Element using SR Latch



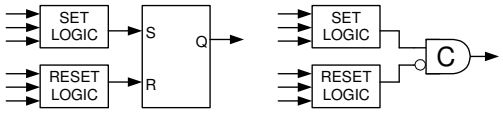
17

Synthesis using SR latches

- Needs SET, RESET regions
- May use KEEP 0, KEEP 1 regions
- May use unreachable states (more later…)
- Area / performance / power may be more or less than gate circuits
- May use C elements instead of SR latches
- SR latches enable implementation with standard libraries

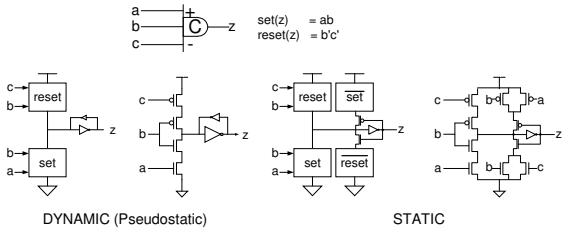
18

Implementation using Latches or C elements



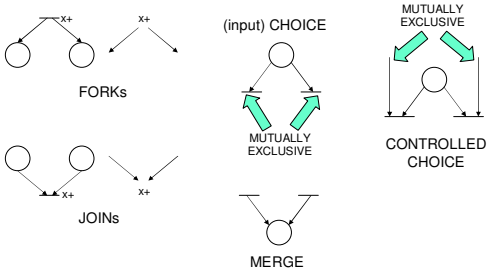
19

Generalized C Element



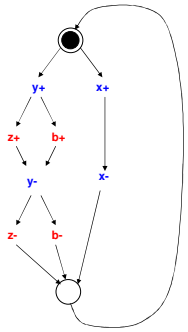
20

More interesting PNs and STGs



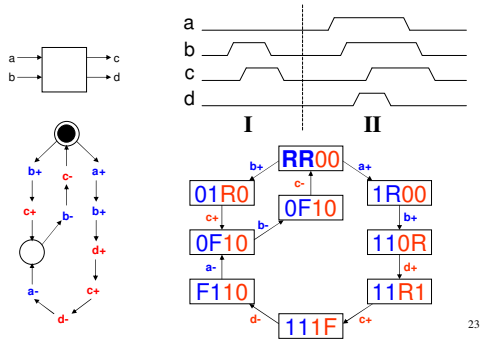
21

STG with Input Choice



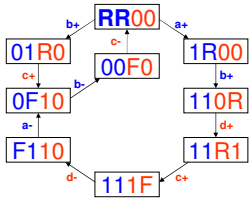
22

A "Simple" Choice Net



23

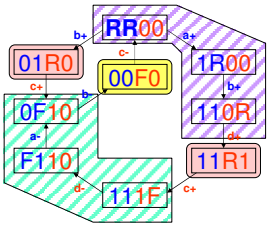
Unreachable States



cd\ab	00	01	11	10
00				
01	X	X		X
11	X	X		X
10				X

24

Regions and Maps



For c

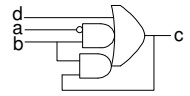
cd\ab	00	01	11	10
00	0	R	0	0
01	x	x	R	x
11	x	x	1	x
10	F	1	1	x

c=d+a'b+bc

set(c)=d+a'b

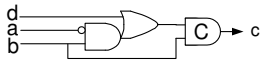
reset(c)=b'

c=d+a'b+bc

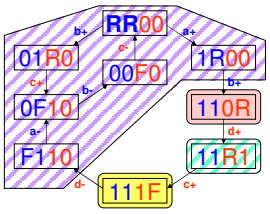


set(c)=d+a'b

reset(c)=b'



Regions and Maps



For d

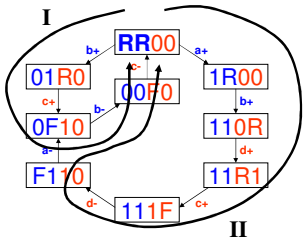
cd\ab	00	01	11	10
00	0	0	R	0
01	x	x	1	x
11	x	x	F	x
10	0	0	0	x

d=abc'

set(d)=abc'

reset(d)=c

Walks on SG and KM



cd\ab	00	01	11	10
00				
01	x	x		x
11	x	x		x
10				x

I II

A decomposed gate implementation is not speed-independent due to hazards. Explain, using the walks.

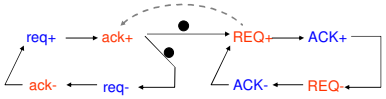
STG Rules

- Any STG:
  - Input free-choice—Only mutex inputs may control the choice)
  - 1-Bounded—Maximum 1 token per place
  - Liveness—No deadlocks
- STGs for Speed Independent circuits:
  - **Consistent state assignment**
    - Signals strictly alternate between + and –
  - **Persistency**
    - Excited signals fire, namely they cannot be disabled by another transition
- Synthesizable STG:
  - **Complete state coding or Unique state coding**
    - Different markings must represent different states (USC), or
    - CSC + separation of next state by environment signals

- We use the following circuit to explain STG rules:

1-Bounded (Safety)

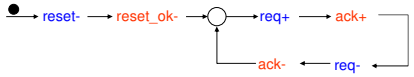
- STG is *safe* if no place or arc can ever contain more than one token
- Often caused by one-sided dependency



STG is not safe:
If left cycle goes fast and right cycle lags,
then arc ack+ → REQ+ accumulates tokens.
(REQ+ depends on both ack+ *and* ACK-)
Possible solution: stop left cycle by right cycle

Liveness

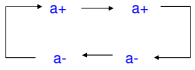
- STG is *live* if from every reachable marking, every transition can eventually be fired



- The STG is not live:
  - Transitions reset, reset_ok cannot be repeated.
- But non-liveness is useful for initialization

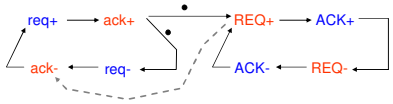
Consistent State Assignment

- The following subset of STG makes no sense:



Persistency

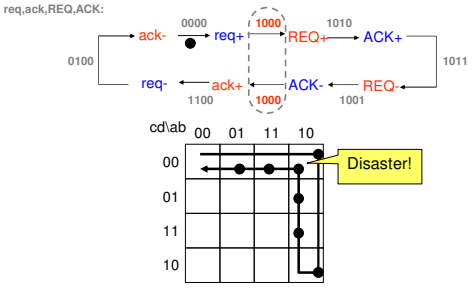
- STG is *persistent* if for all arcs a* → b*, other arcs ensure that b* fires before opposite transition of a* (* is either + or -)
- Non-persistency may be caused by one-sided relations



STG is not persistent (in addition to being unsafe):
If left cycle goes fast and right cycle lags,
then ack+ → ack- before REQ+ .
Danger: Logic design may be REQ+ = ack+
Exception: If a* → b*, assume that the environment assures persistency.
Possible solution: stop left cycle by right cycle.

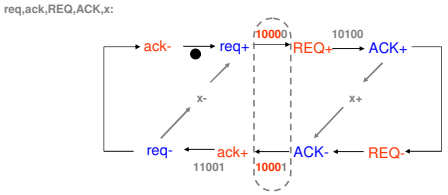
Complete State Coding

- STG has a *complete state coding* if no two different markings have identical values for all signals.



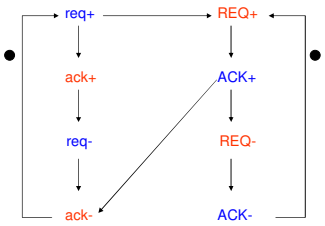
Complete State Coding

- Possible solution: Add an internal state variable



A faster STG?

- Does it need an extra variable?



37