

VLSI Design

Lecture 17: Sequential Machine Design and Implementation

Shaahin Hessabi

Department of Computer Engineering

Sharif University of Technology

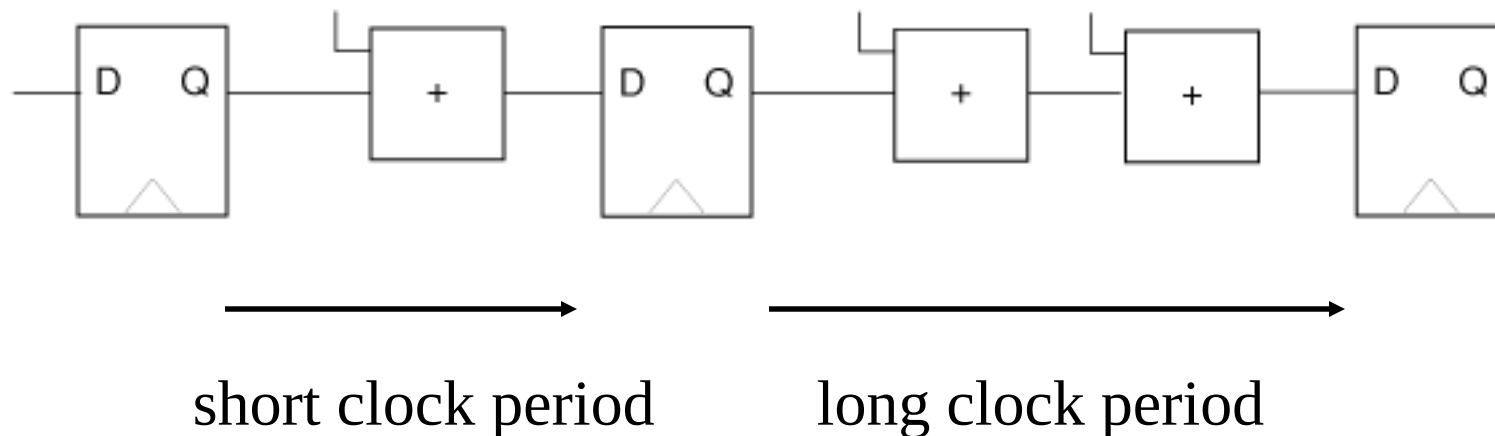
Adapted, with modifications, from lecture notes prepared by the
author (from Prentice Hall PTR)

Clock period

- For each phase, phase period must be longer than sum of:
 - combinational delay;
 - latch propagation delay.
- Phase period depends on longest path.

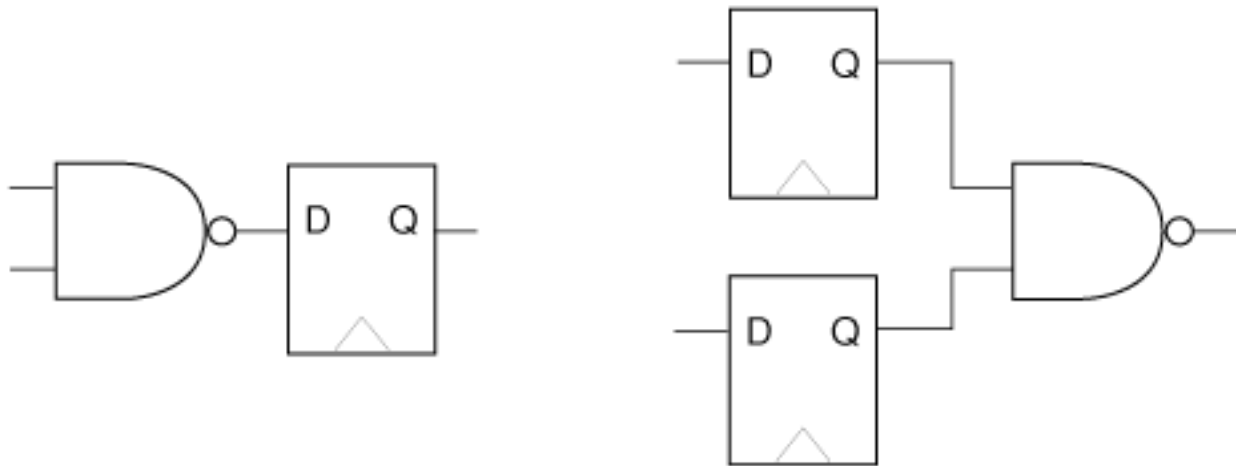
Unbalanced delays

Logic with unbalanced delays leads to inefficient use of logic:



Retiming

Retiming moves memory elements through combinational logic:



Retiming properties

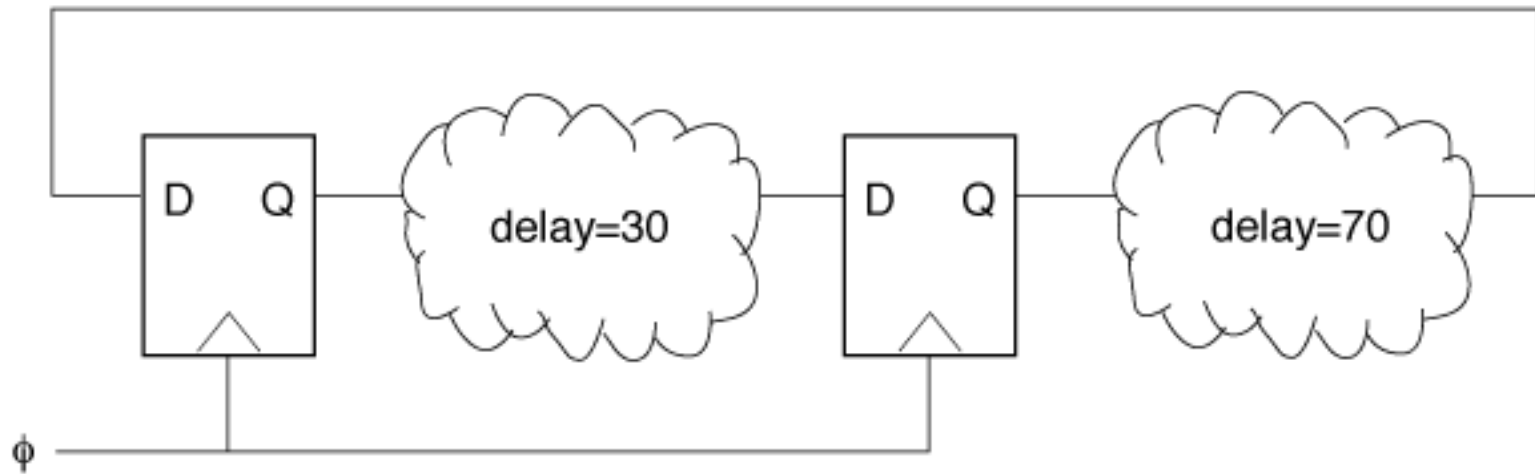
- Retiming changes encoding of values in registers, but proper values can be reconstructed with combinational logic.
- Retiming may increase number of registers required.
- Retiming must preserve number of latches around a cycle; may not be possible with reconvergent fanout.

Advanced performance analysis

- Latch-based systems always have some idle logic.
- Can increase performance by blurring phase boundaries (time borrowing).
 - Results in cycle time closer to average of phases.

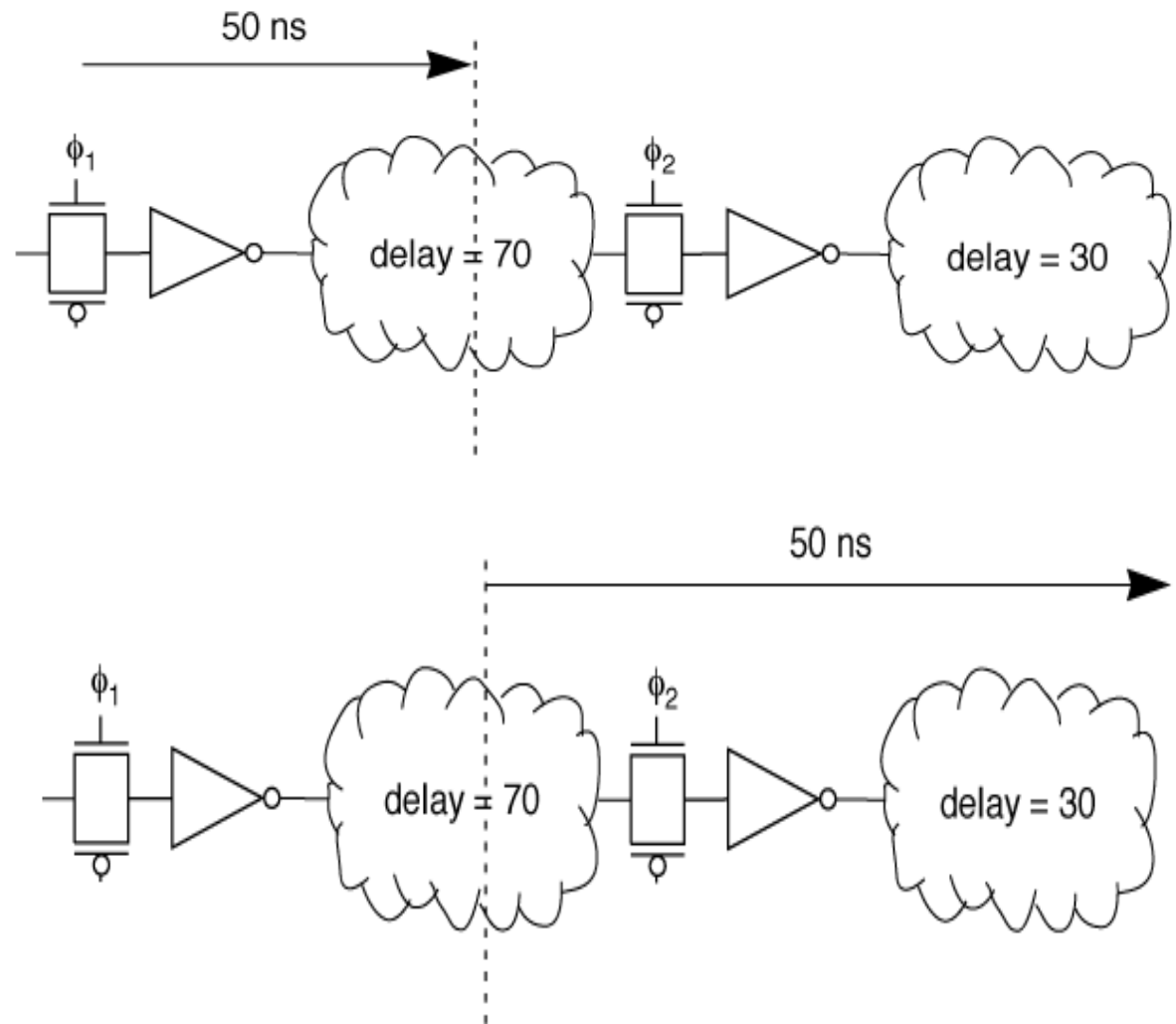
Example with unbalanced phases

One phase is much longer than the other:



Spreading out a phase

- Compute only part of long paths in one phase (Φ_1):
- Use other phase (Φ_2) for end of long logic block and all of short logic block:



Problems

- Feasible for latches, not FFs.
- Hard to debug: can't stop the system.
- Hard to initialize system state
 - since some of that state is stored on wires in the combinational logic.
- More sensitive to process variations.

Sequential machine design

- Two ways to specify sequential machine:
 - structure: interconnection of logic gates and memory elements.
 - function: Boolean description of next-state and output functions.
- Best way depends on type of machine being described.

Counter

- Easy to specify as one-bit counter.
- Harder to specify n-bit counter behavior.
 - Can specify n-bit counter as structure made of 1-bit counters.

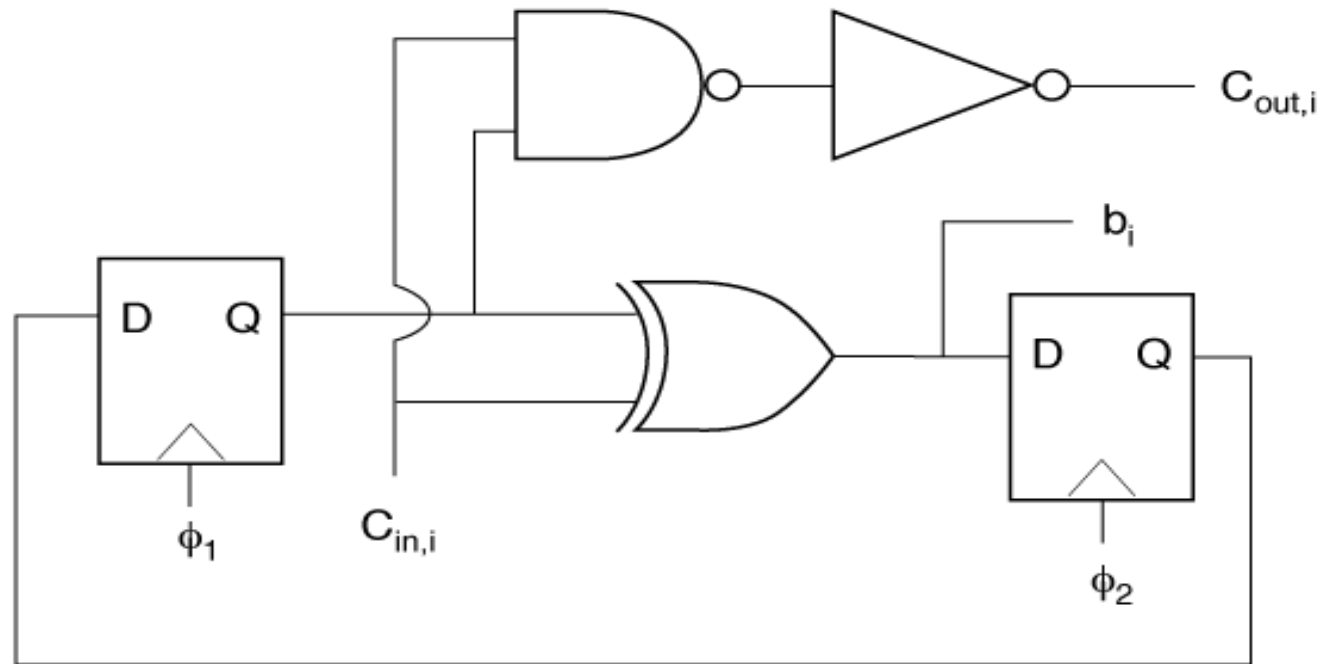
One-bit counter

Truth table:

count	C_{in}	next	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

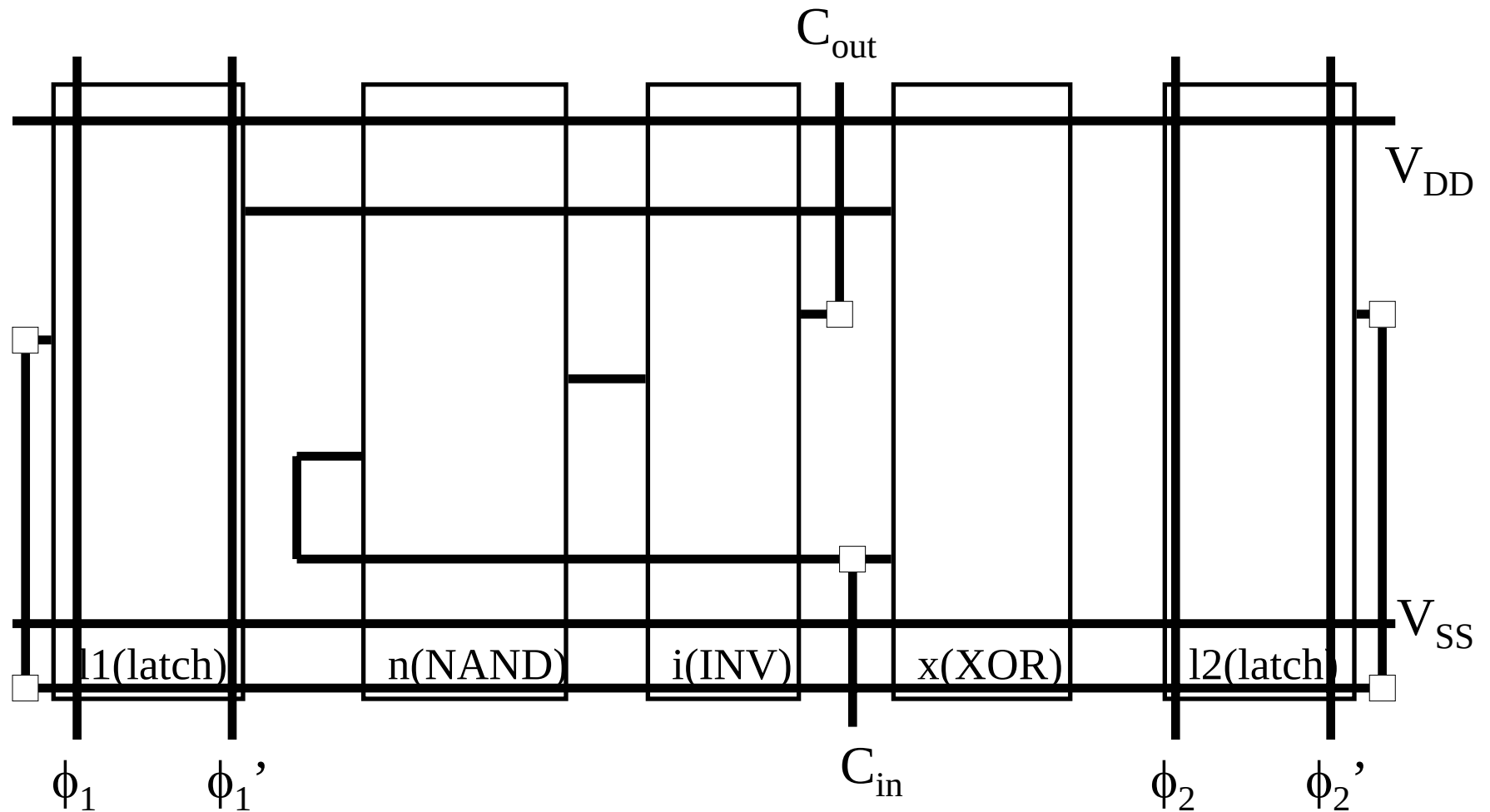
$$\Rightarrow \begin{aligned} \text{next} &= \text{count} \oplus C_{in} \\ C_{out} &= \text{count} \& C_{in} \end{aligned}$$

One-bit counter (implementation and operation)

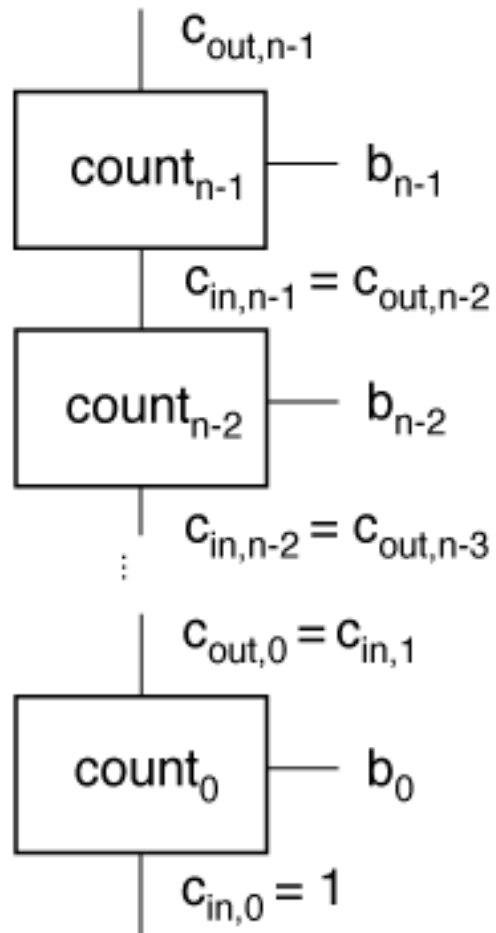


- All operations are performed as $s\phi_2$.
- XOR computes next value of this bit of counter.
- NAND/inverter compute carry-out.

One-bit counter stick diagram



n-bit counter structure



State transition graphs/tables

- Basic functional description of FSM.
- Symbolic truth table for next-state, output functions:
 - no structure of logic;
 - no encoding of states.
- State transition graph and table are functionally equivalent.

01 string recognizer

Behavior of machine which recognizes “01” in continuous stream of bits:

time	0	1	2	3	4	5
input	0	0	1	1	0	1
state	bit1	bit2	bit2	bit1	bit1	bit2
next	bit2	bit2	bit1	bit1	bit2	bit1
output	0	0	1	0	0	1

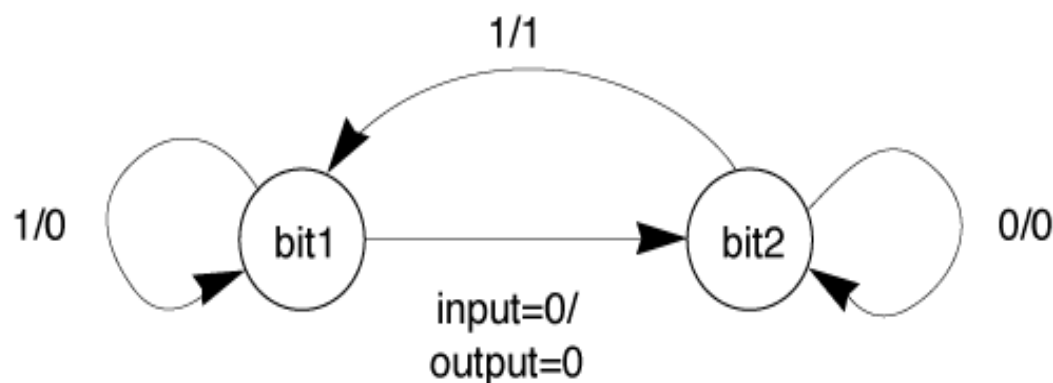
- Waits for 0 to appear in state *bit1*.
- Goes into separate state *bit2* when 0 appears.
- If 1 appears immediately after 0, can't have a 01 on next cycle, so can go back to wait for 0 in state *bit1*.

State transition table and graph

Symbolic state transition table:

input	present	next	output
0	bit1	bit2	0
1	bit1	bit1	0
0	bit2	bit2	0
1	bit2	bit1	1

State transition graph (equivalent to state transition table):



State assignment

- Must find binary encoding for symbolic states: state assignment.
- Choice of state assignment directly affects both the next-state and output logic:
 - area;
 - delay.
- May also encode some machine inputs/outputs.

01 recognizer encoding

Choose *bit1* = 0, *bit2* = 1:

input	present	next	output
0	0	1	0
1	0	0	0
0	1	1	0
1	1	0	1

Logic implementation

After encoding, truth table can be implemented in gates:

