UNIVERSITY OF THESSALY

TECHNION Israel Institute of Technology

USC Viterbi School of Engineering University of Southern California

Asynchronous Design Seminar at University of Verona – Lecture Notes 2

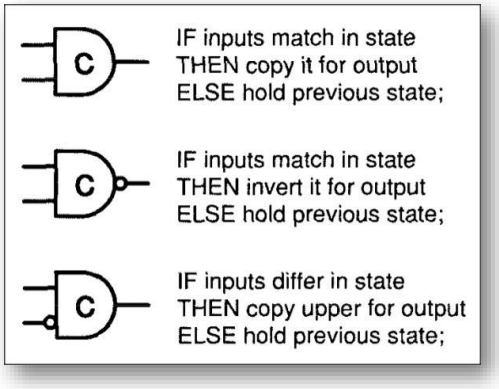
Asynchronous Design Fundamentals

Bundled-Data Pipelines

Micropipelines and Mousetrap

The C Element

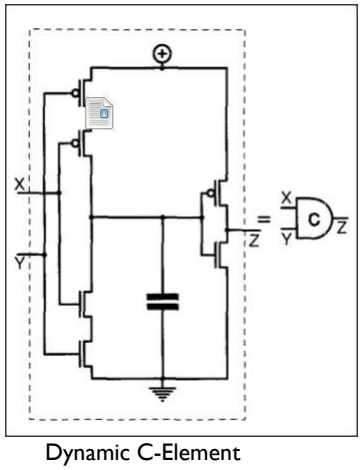
- AND gate for events



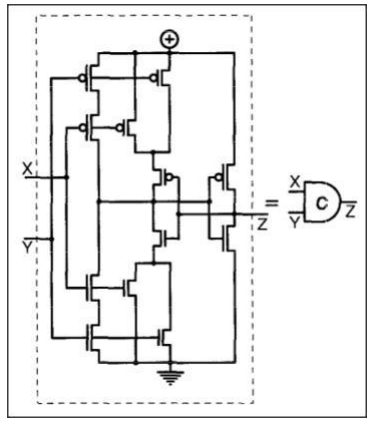
a	b	c
0	0	0
0	1	c′
1	0	c′
1	1	1

- What is the OR gate equivalent for events?

C-Element Implementations



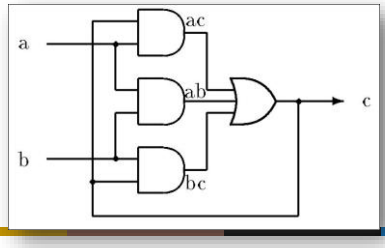
Dynamic C-Element

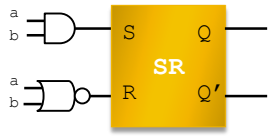


Static C-Element

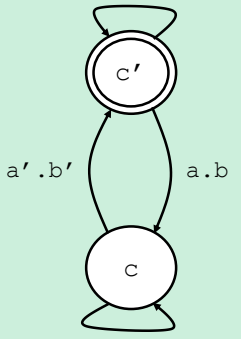
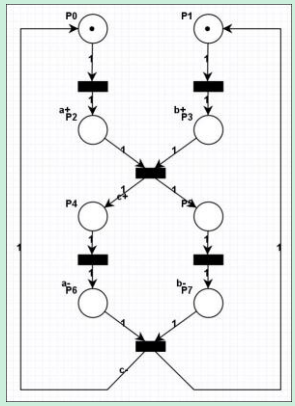
C-Element Implementations

- Using Standard Cells
  - 2-input C-Element's Logic Function is:
    $c = ab + c(a + b) = ab + bc + ac$
  - n-input C-Element is:
    $c = a_1a_2...a_n + c(a_1 + a_2 + ... + a_n)$
- Thus may be implemented using two level logic or using an SR latch



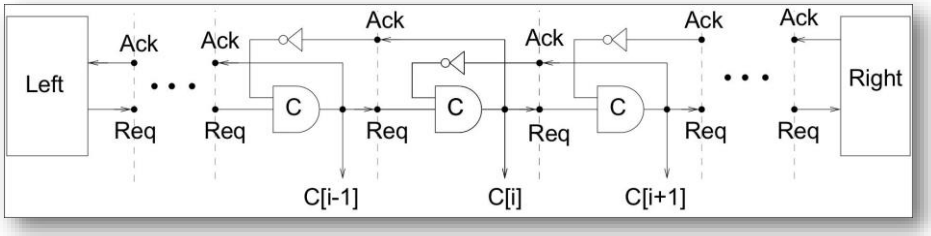


C Gate Formal Models

FSM	PTnet
	

- *Empty transitions* ➔ else stay in same state

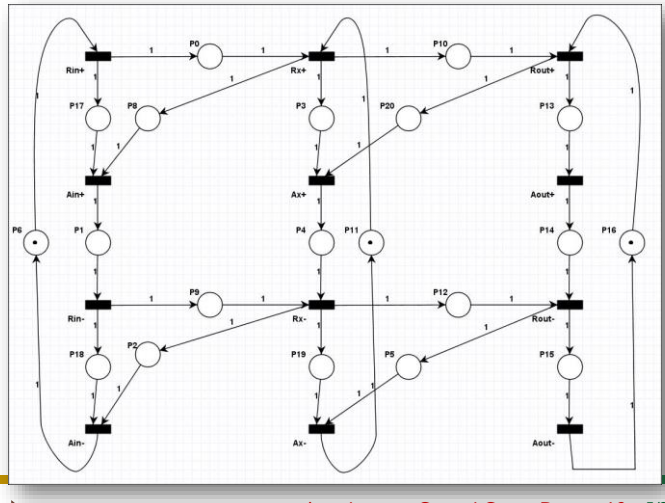
Micropipeline Control Structure

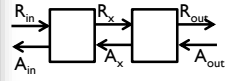


- Req/Ack are handshaking pairs
- C[i] accepts 1/0 from C[i-1] only if C[i+1]=0/1
- Think of 1010101.. as waves: ⌇0 ⌇0 ⌇0 ⌇..
- The C-elements propagate waves precisely
- Timing depends on local delays, may vary along the pipe
- If RIGHT is quiet, pipe will fill and stall
- Same for 4-phase, 2-phase

Micropipeline Analysis - PTnets

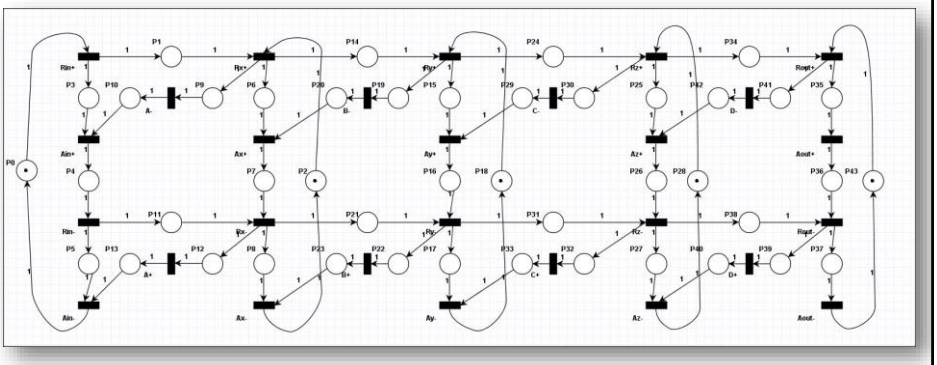
- Two-stage Micropipeline PTnet:





Micropipeline Analysis - PTnets

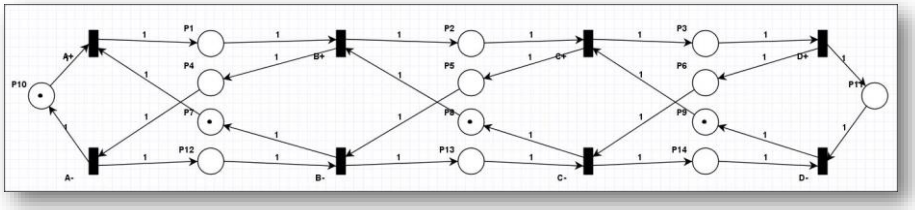
- Four-stage Micropipeline PTnet:



- Represents 4 C Elements and their environments
- ***Correct operation may be verified by possible signal orders***

Micropipeline Analysis - PTnets

- Four-stage Micropipeline Ptnet
  - Reduction to Latch Control Signals

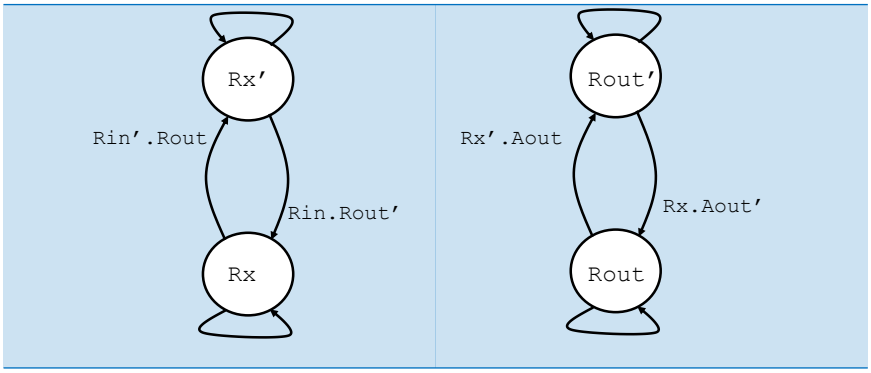


- Characteristic Pattern:

Micropipeline Analysis - MSFSMs

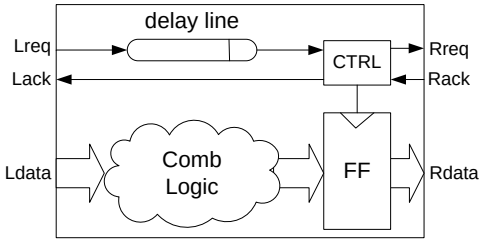
- Two-stage Micropipeline FSMs:

First stage FSM | **Second stage FSM**



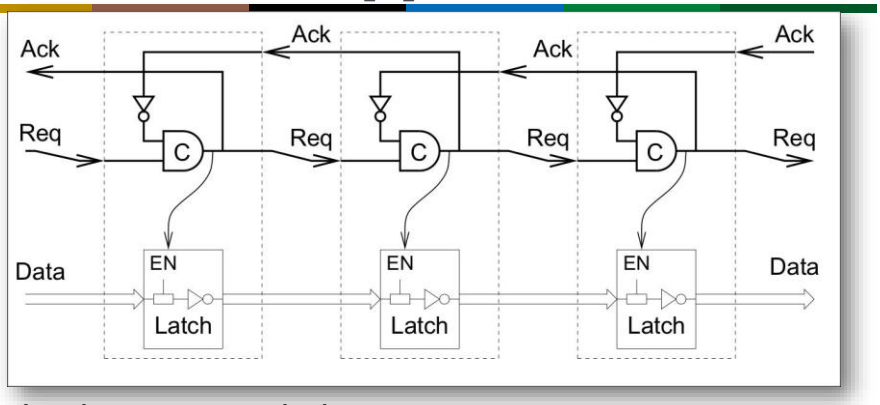
- *What about LHS/RHS FSM Interfaces?*
  - What are the two environment FSMs Rin/Ain, Rout/Aout?

Bundled-Data Concept



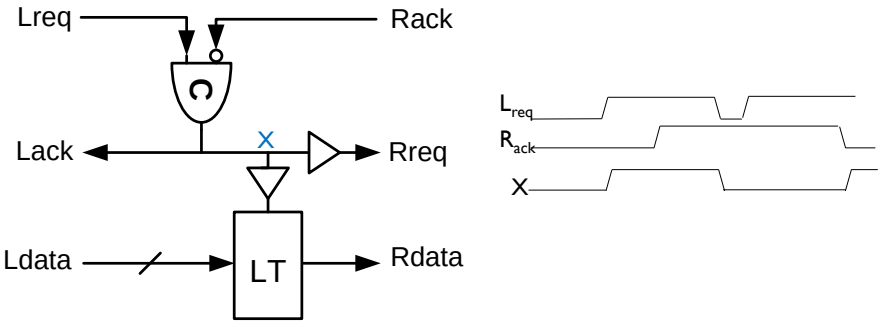
- Static combinational logic (typically) and standard FFs or latches
- Channels use bundled-data protocol
  - Data lines of the channel are "bundled" timing-wise to the REQ line, so:
    - time(valid,REQ) > time(valid,data)
  - Delay line matches worst-case delay of combinational logic
  - Margin limits performance, particularly because adds to forward latency
- Controller CTRL drives local clock to bank of FFs (or latches)
  - Designed using known templates or burst-mode controllers, signal-transition graphs, or syntax directed translation

Four-Phase Micropipeline



- Latches are active high
- Four-phase, so latches open on first ½ of h/s and close on second ½

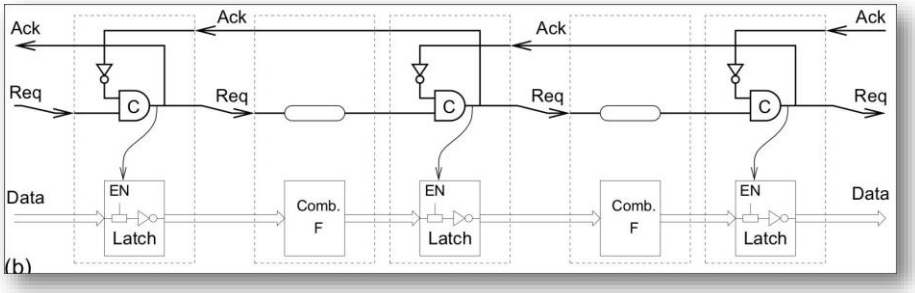
Four-Phase Micropipeline Stage



Note: Latch is level-sensitive

- Four-phase protocol on Lreq/Lack and Rreq/Rack
  - Latch is simple level sensitive traditional latch
  - This is a Half-Buffer!

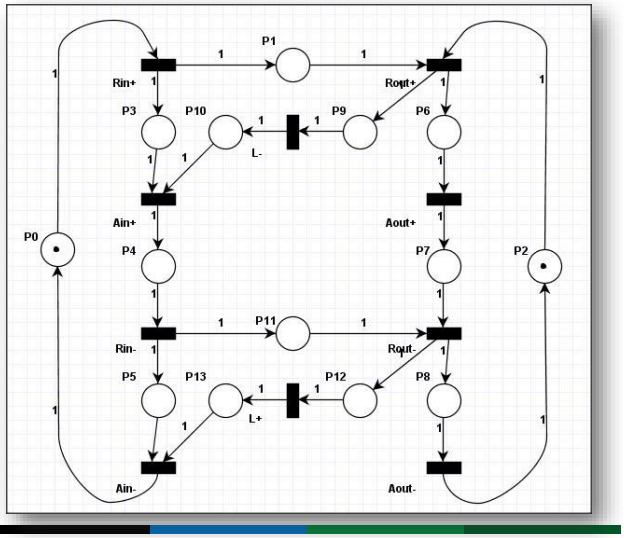
Four-Phase Micropipeline with C.L.



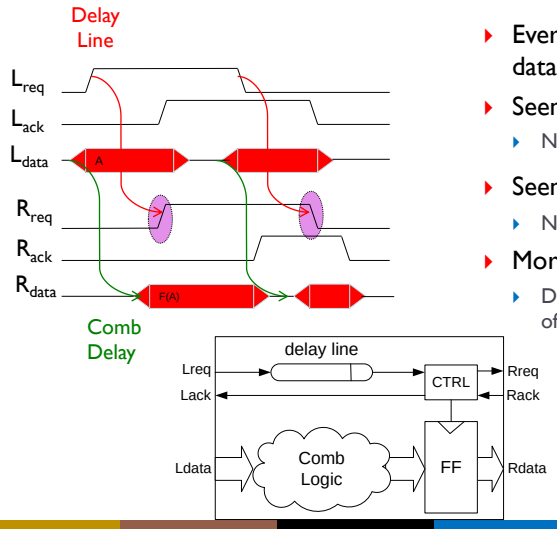
- Delay elements ***mimic*** worst-case critical path of Combinational Logic Cloud, F, of each pipe stage
- Bundled-Data Design (bundling assumption on Req, Data)

Micropipeline 4-Phase PTnet including Latch Enable

- L
  - **Active-high** in PTnet
- L-
  - Closes Latch
- L+
  - Opens Latch

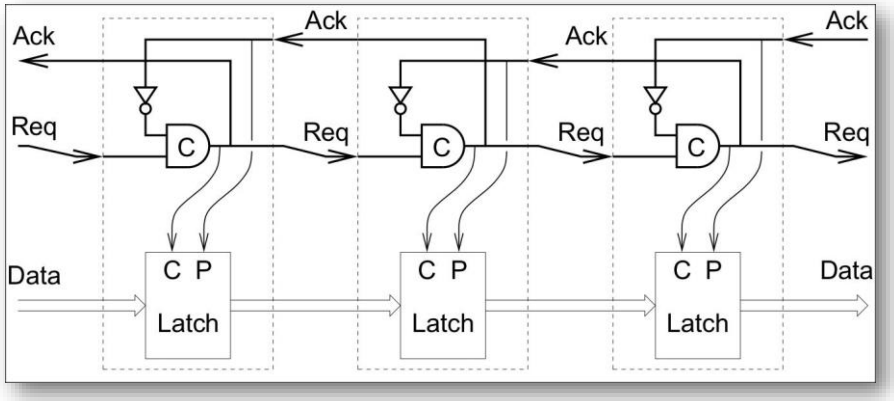


Two Phase Signaling



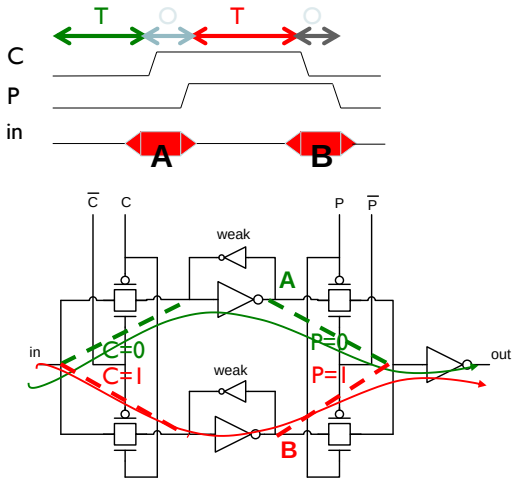
- Every Toggle on request => new data
- Seems to be faster (NRZ)
  - Not in general
- Seems to be low power
  - Not usually the case
- More complex Control
  - Data Validity coded in phase difference of Req & Ack

Two-Phase Micropipeline



- Simple version without Cd (C done), Pd (P done)
  - Little difference anyway in generating fake done signals (using delay elements)

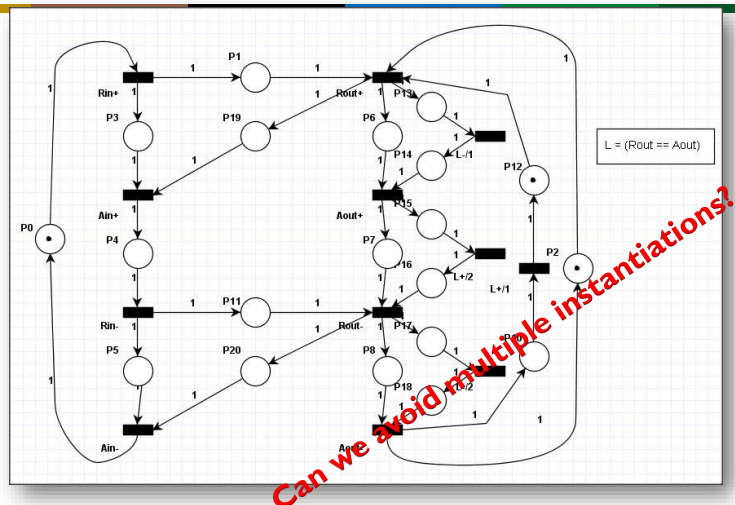
Capture-Pass Data Latch



- Two-phase data-latch
  - Becomes transparent when a Pass "P" event occurs
  - Becomes opaque when a Capture "C" event occurs
  - Store data alternatively in top and bottom inverter loops
- (C == P) ➔ open
- (C != P) ➔ closed
  - Transition signalling

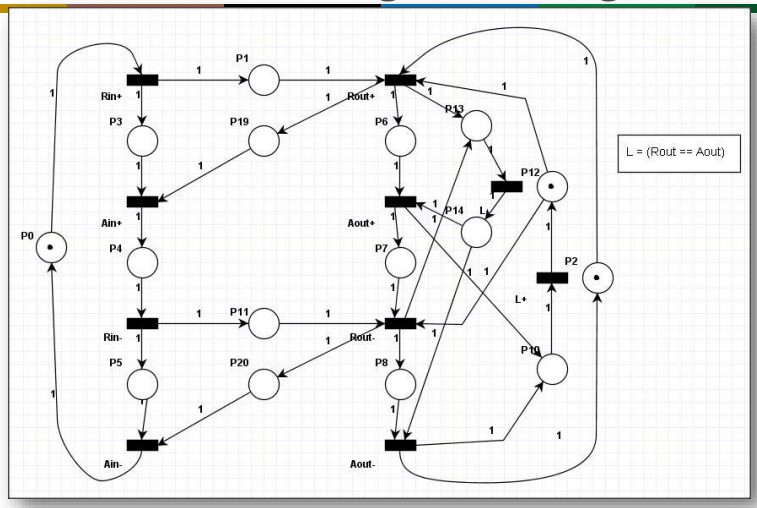
Too many Transistors

Micropipeline 2-Phase PTnet including Latch Enable



- Multiple instantiations of L signal (L/1, 2)

Micropipeline 2-Phase PTnet including Latch Enable and Single Latch Signal



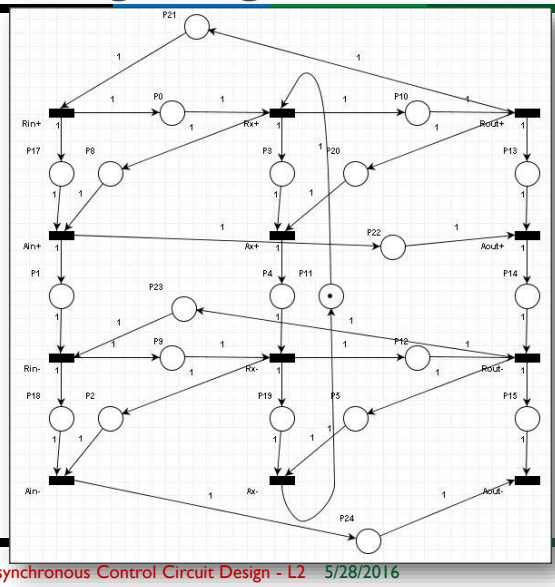
- Note the AC Return from Choice/Choice Needed

Micropipeline Rings

- So far focused on Linear Pipelines
- A Latch Controller Ring is a very useful structure
  - Data tokens go around the ring
  - Represents basic iterative computation
  - May include entry/exit points
  - How do I build a ring?
    - **Connect Rout/Aout of RHS controller to Rin/Ain of a LHS controller**
- Does it *always* work with micropipelines?

Micropipeline 2-Stage Ring

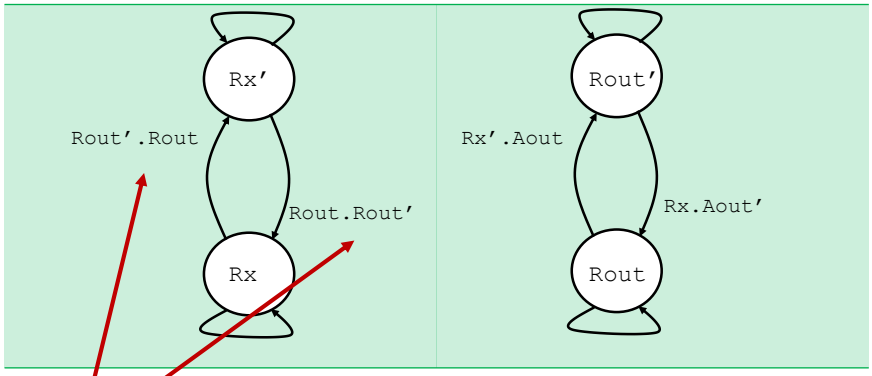
- PTnet Cycles with no tokens ➔ **Deadlock**!!!
- Commoner's Theorem:
  - **Deadlocked systems include an unmarked cycle**
- *Can I find a valid marking to make it **live**?*



Micropipeline Analysis - MSFSMs

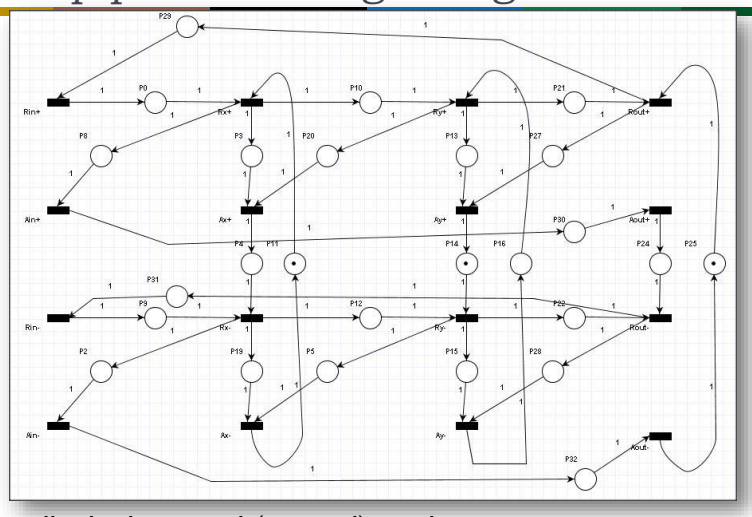
- Two-stage Micropipeline FSMs:

First stage FSM **Second stage FSM**



- *Ring* ➔ `Rin` = `Rout`, `Ain` = `Aout`
- ***Never LIVE!!!***

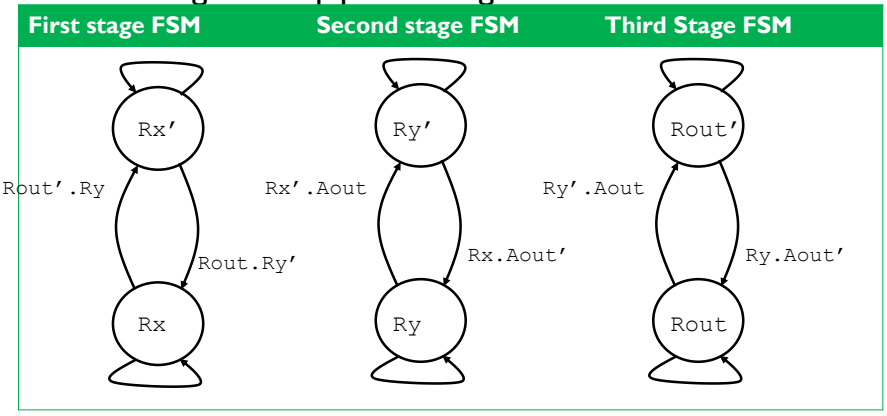
Micropipeline 3-Stage Ring



- Deadlocked at initial (original) marking

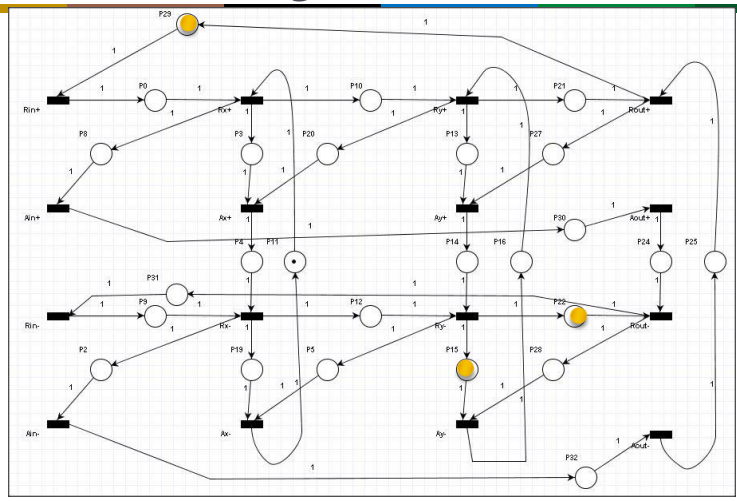
Micropipeline Analysis - MSFSMs

- Three-stage Micropipeline Ring FSMs:



- *Now:* `Rin` = `Rout`, `Ain` = `Aout`
- *Live when?* `Rout'`, `Ry'`, `Rx` *initial states*

Live Micropipeline 3-Stage Ring with Alternate Marking

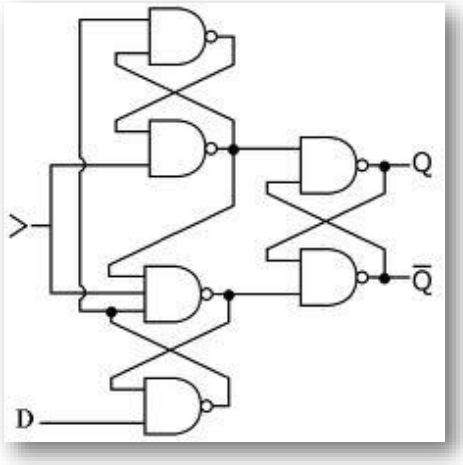


- Live Marking Shown: `Rin+`, `Rx+`, `Ry-`

Another interesting example of FSM-based Formal Analysis

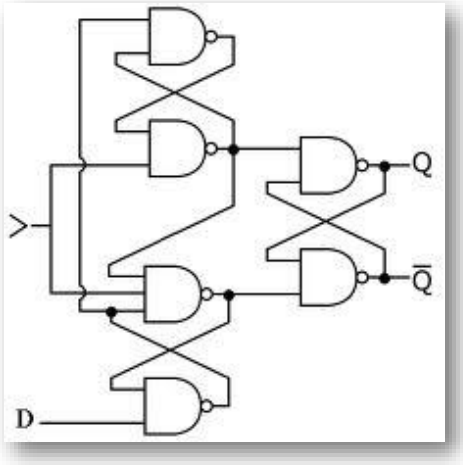
Edge-Triggered FF, Morris Mano Book

D-type FF from Mano's Book



- consists of 3 set-reset latches
- RHS latch produces Q, Q'
- Verbal functional explanation quite complex and non-intuitive
  - when **input D is high**, lower LHS latch is set whenever the clock is low
  - thus, the set input of the upper LHS latch is triggered, which sets the output latch (RHS) whenever the clock is high
  - when **input is D low**, lower LHS latch is reset, thus resetting the output latch (RHS), whenever the clock is high
- As a result, Q may only change state when clock makes a low→high transition

D-type FF from Mano's Book

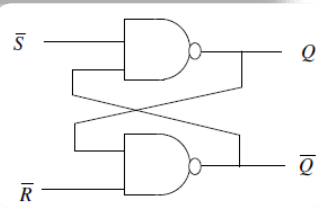


- How can we formally verify that output latch's inputs can never cause race?
  - are never 00, OR
  - can never transition from 00→11

Active-Low SR latch

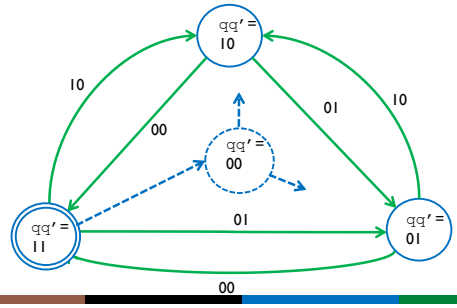
S	R	Q	Q_{next}	$\overline{Q_{next}}$
0	0	x	1	1
0	1	x	1	0
1	0	x	0	1
1	1	0	0	1
1	1	1	1	0

- QUESTION:
  How do we model the sequential SR latch behaviour?



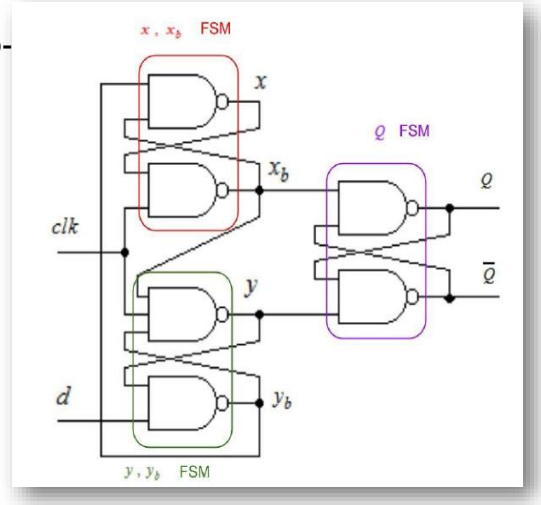
Single (Active-Low) SR latch FSM Model

- Choice between 2 states or 3 states
  - 2 states model hides possibility of SR = 00
- 00→11 transition result is uncertain, thus the fourth state QQ' = 11 does not have deterministic next states
- We assume initial state to be 11
  - May be different, e.g. 01
  - Transitions correspond to SR inputs

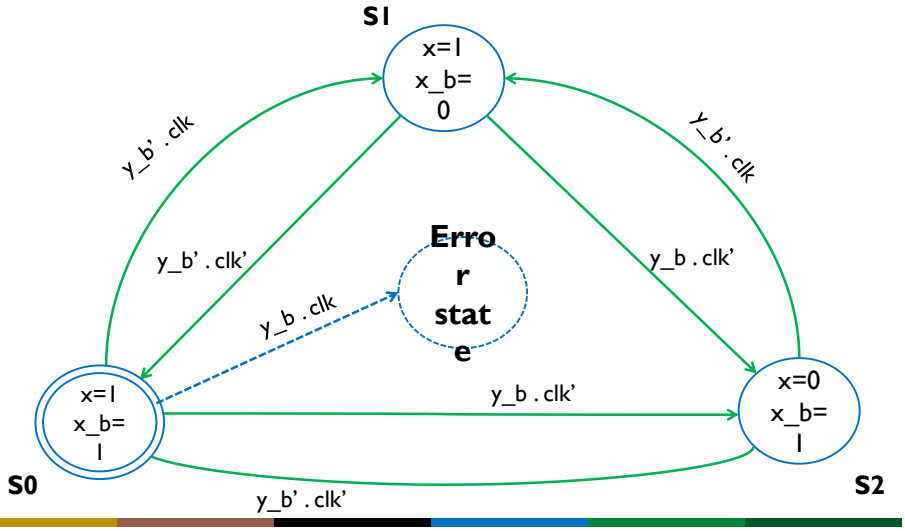


D-Type FF – inferring MSFSM model

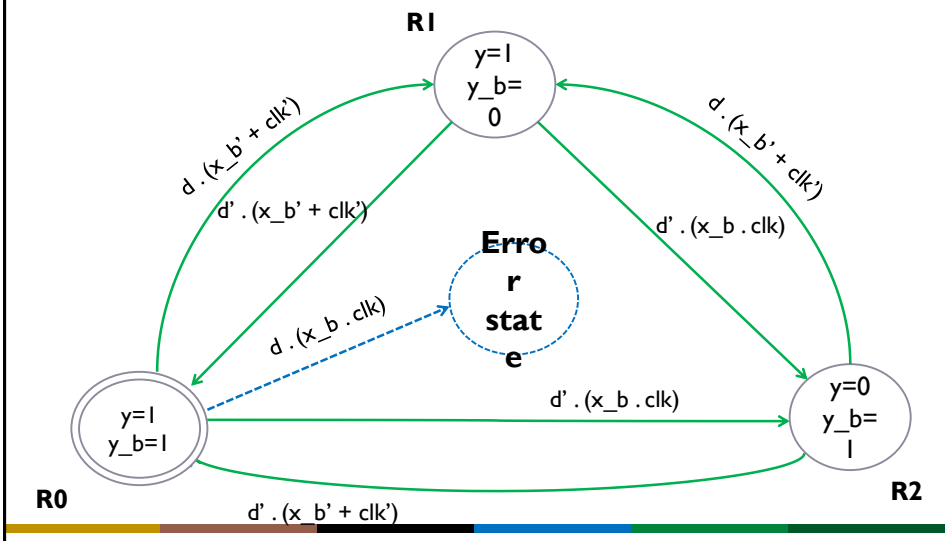
- We split the D-type FF circuit into 3 separate sub-circuits *(SR latches)*
- *Infer an FSM of each sub-circuit*
- Signals and states will transition asynchronously
- Goal:
  - Signals x_b, y may never assume 00, OR
  - may never transition from 00→11



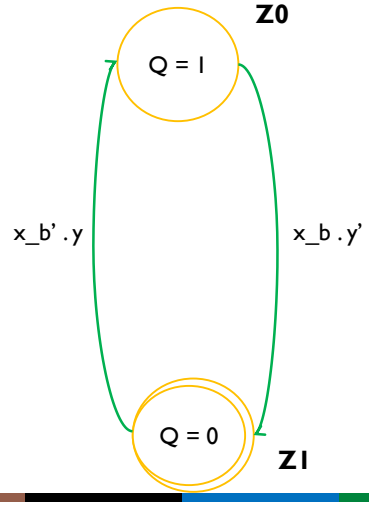
FSM #1 - x, x_b signal generation



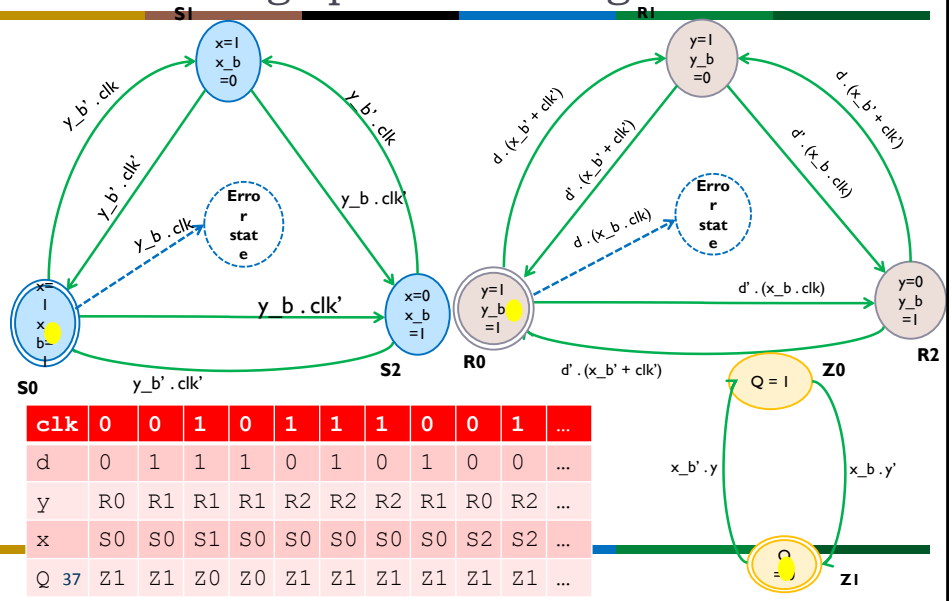
FSM #2 - y, y_b signal generation



FSM Q – RHS latch output Q



Simulating operation using MSFSMs



clk	0	0	1	0	1	1	1	0	0	1	...
d	0	1	1	1	0	1	0	1	0	0	...
y	R0	R1	R1	R1	R2	R2	R2	R1	R0	R2	...
x	S0	S0	S1	S0	S0	S0	S0	S0	S2	S2	...
Q	Z1	Z1	Z0	Z0	Z1	Z1	Z1	Z1	Z1	Z1	...

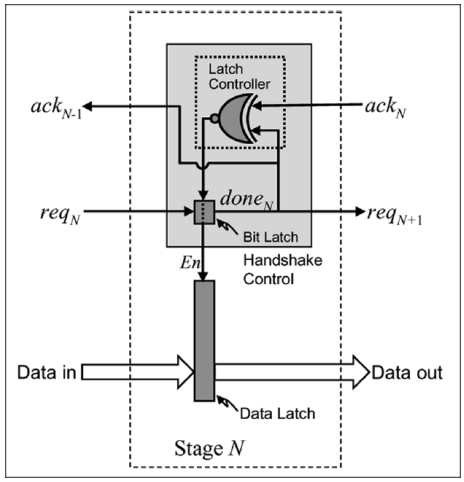
37

Conclusions

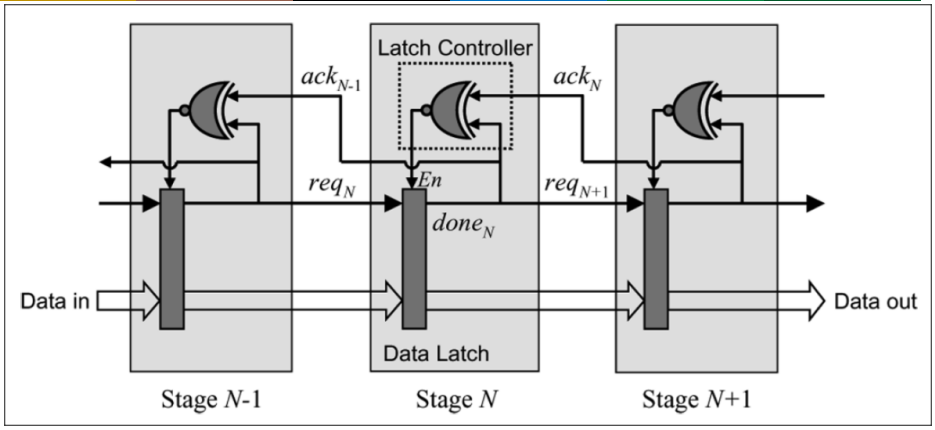
- The traditional way of verifying a model demands multiplication of FSMs, which leads to bigger state spaces
- Using interactive FSMs and based on an asynchronous change of signals, we escape the boundaries of a monolithic FSM and avoid state explosion.
- The state space of the formal analysis in our example was reduced by almost 55%

Mousetrap

- Mousetrap is one of the simplest and fastest latch controller implementations
- Consists of
  - XOR gate producing the Latch `EN` signal
  - Latch for Storing `Ri` state
- Latch is open (transparent) when
  - `Ro` == `Ao`
- Closes (opaque) when
  - `Ro` changes (1)
- Opens again when
  - `Ao` changes a second time (1)

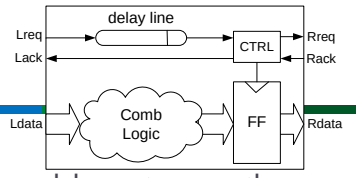


Mousetrap Pipeline

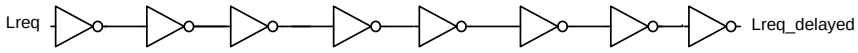


- Caveat
  - Does not support feedback dependencies in the Datapath

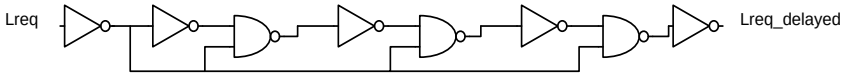
Delay Line Design



- **Symmetric** (rise/fall ~same)
  - Designed with an even # of inverters that model worst-case path delay through combinational logic

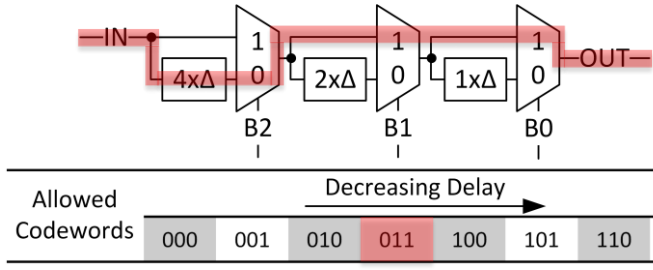


- **Asymmetric** (rise slow/fall fast or the opposite)
  - Faster reset implemented by replacing some INVs with NANDs



- **Other**
  - May also use re-create path of gates along critical path to better match delay (delay distribution)

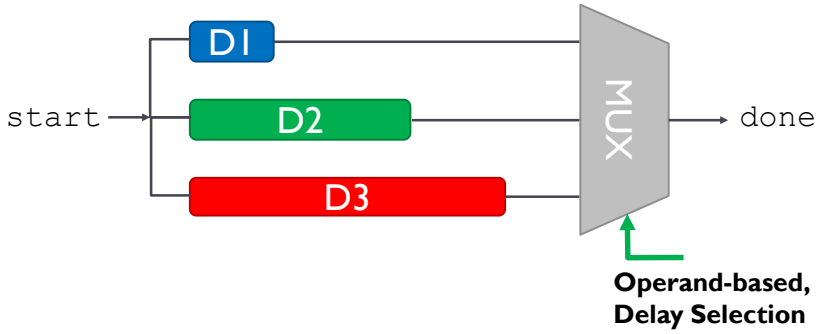
Programmable Delay Elements: MUX-DE



N. Mahapatra et al., "Comparison and Analysis of Delay Elements," in *MWSCAS*, 2002, pp. 473–476.

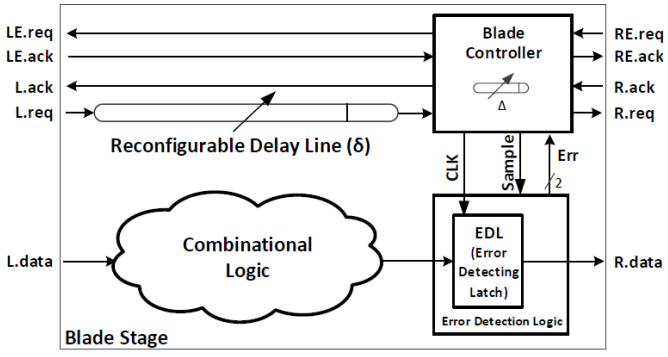
- Most popular programmable DE
- Delay controlled by number of buffers in the signal path
- Binary codewords

Speculative Completion Sensing – Variable Latency



- ***Enables data-dependent component delays in bundled-data environment***

Resilient Bundled Data

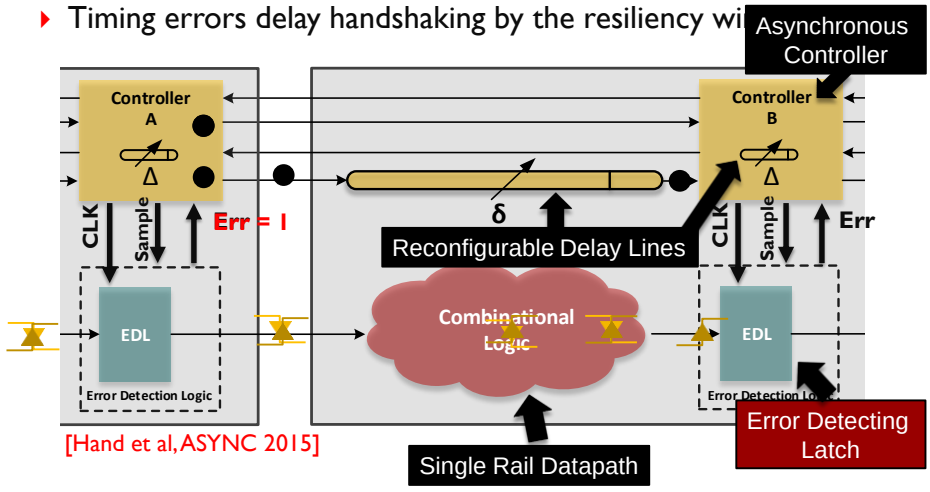


- Two delay elements
- Average case performance

D. Hand, M. Moreira, H. Huang, D. Chen, F. Butzke, Z. Li, M. Gibiluka, M. Breuer, N. Calazans and Peter Beerel, "Blade - A Timing Violation Resilient Asynchronous Template," in *ASYNC 2015*, pp. 21–28.

Resilient Bundled Data

- Timing errors delay handshaking by the resiliency wi



[Hand et al, ASYNC 2015]

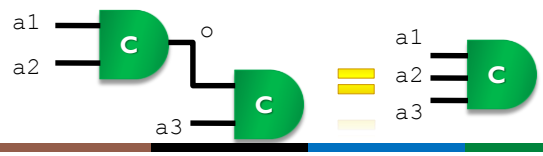
C Element Extensions and Generalizations

Boolean Function with Feedback

- **Define**:
  - SET, KEEP and RESET functions which are feedback free
  - RESET = U – SET (OFF-set of SET), **or**
  - RESET ∩ SET = ø
  - KEEP = RESET'
  - **Sole feedback of f is on the f line** (feeds back to input)
- Form 1 (Set and Keep):
  - **f** = (SET function) + **f** (KEEP function)
- Form 2 (Set and Reset):
  - **f** = (SET function) + **f** (RESET function)'
- **Example**
  - Asymmetric C Element:
  - **f = bc (SET) + f (ab')' (RESET)'**

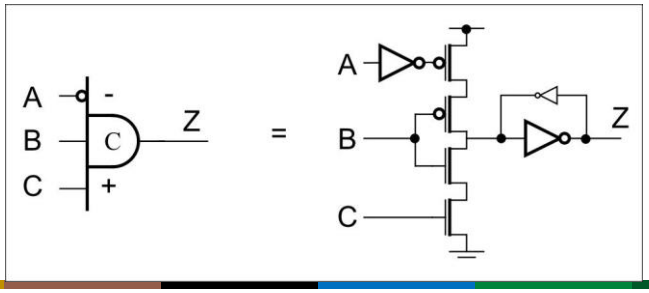
Composing C Elements

- Based on the C-Element Boolean Equation:
  - 2-input C-Element's Logic Function is:
    $c = ab + c(a + b) = ab + bc + ac$
  - n-input C-Element is:
    $c = a_1a_2\ldots a_n + c(a_1 + a_2 + \ldots + a_n)$
- It can be shown that C gates are composable using their ***set***, ***keep*** functions:
  - $c3_{set} = a1a2a3$, $c3_{keep} = a1 + a2 + a3$
  - $c2_{set} = a1a2$, $c2_{keep} = a1 + a2$
    - $o_{set} = a1a2$, $o2_{keep} = a1 + a2$
    - $output_{set} = (o_{set} \cdot a3)$, $output_{keep} = (o_{keep} + a3)$
- Thus:
  - **$o_{set} = c3_{set}$, $output_{keep} = c3_{keep}$**



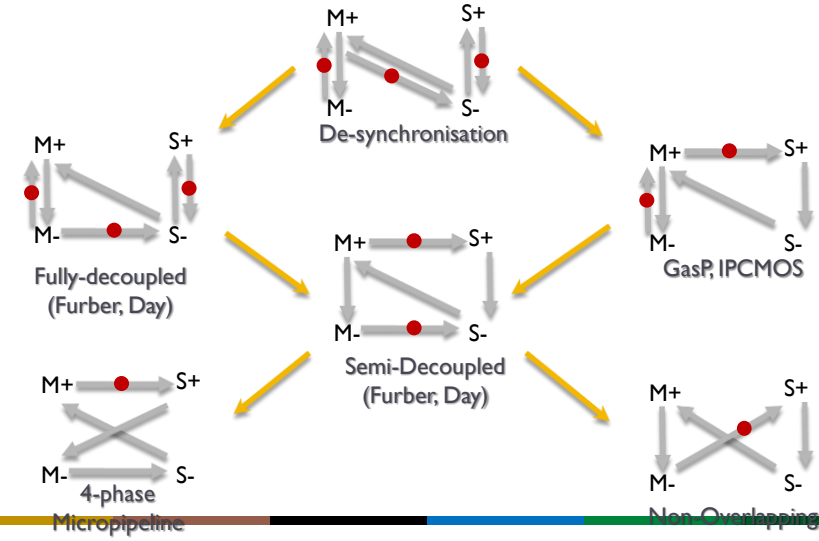
Asymmetric C Elements

- In certain cases, the Set and Reset logic of a C element is not identical
  - Obvious from PTnet specification of a controller
- Asymmetric C elements are an extension of the basic C
- Equivalent to SR Latch or Boolean feedback circuit as well

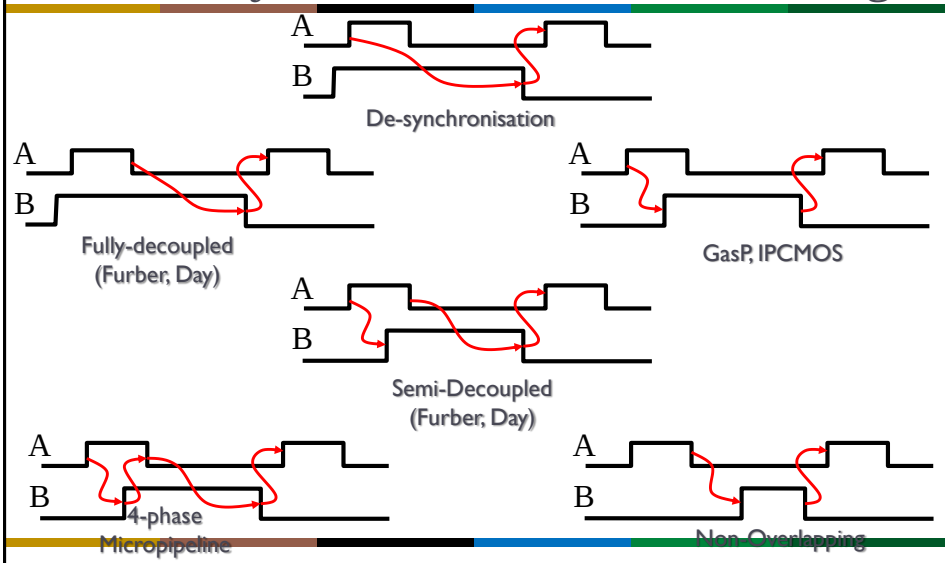


Taxonomy of Latch Controllers

Taxonomy of Latch Controllers - Signals

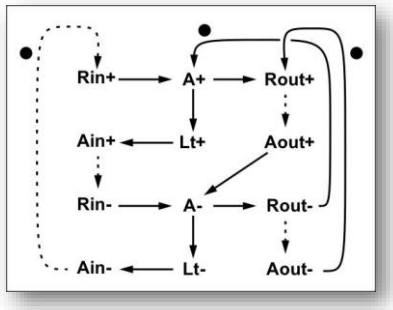


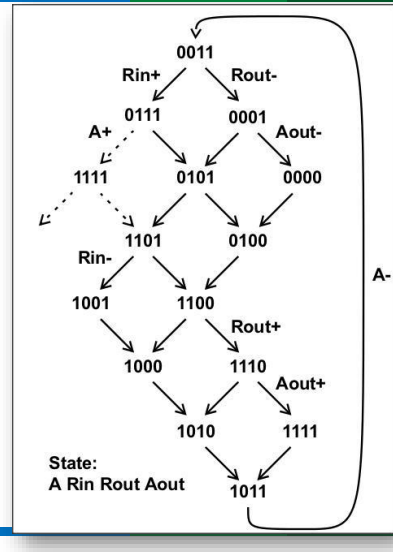
Taxonomy of Latch Controllers - Timing



Semi-Decoupled Latch Controller

Semi-Decoupled Latch Controller – STG and SG

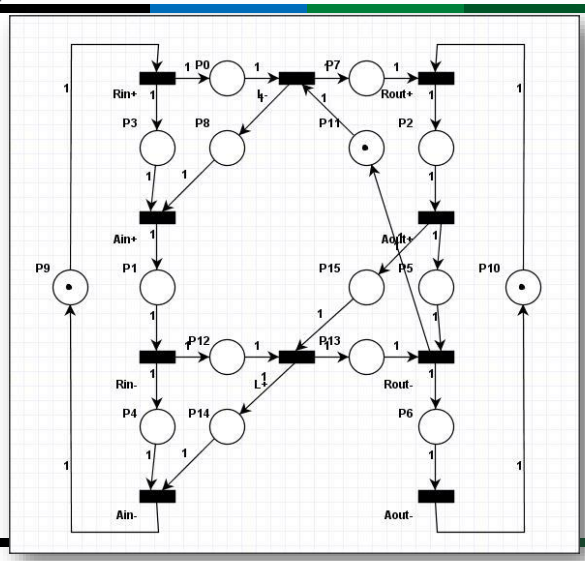




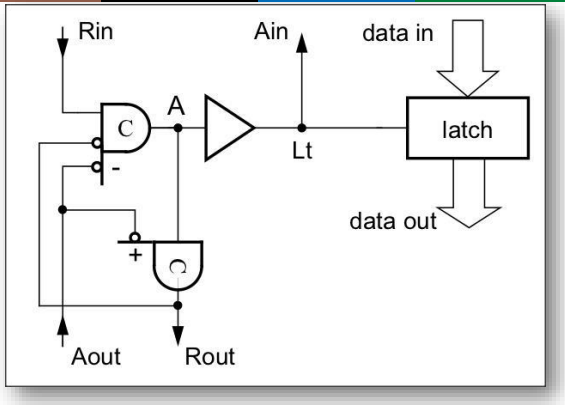
- A is redundant – ignore
  - pre-buffered Lt
- Lt is for active-low latch,

Semi-Decoupled Latch Controller PTnet

- Latch Control
  - L
  - L-, L+ in PTnet

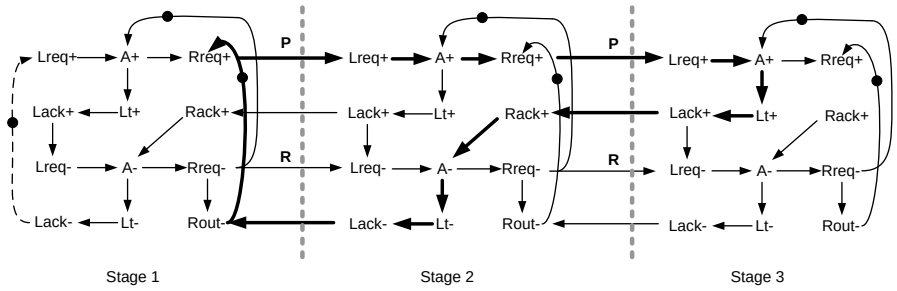


Semi-Decoupled Latch Controller Circuit



- C-gate Set, Reset functions may be inferred from STG/PTnet specification of the controller

Semi-Decoupled – PTnet for 3 Stages

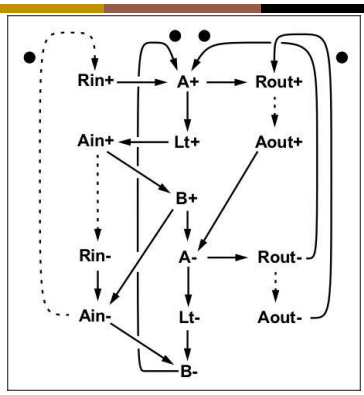


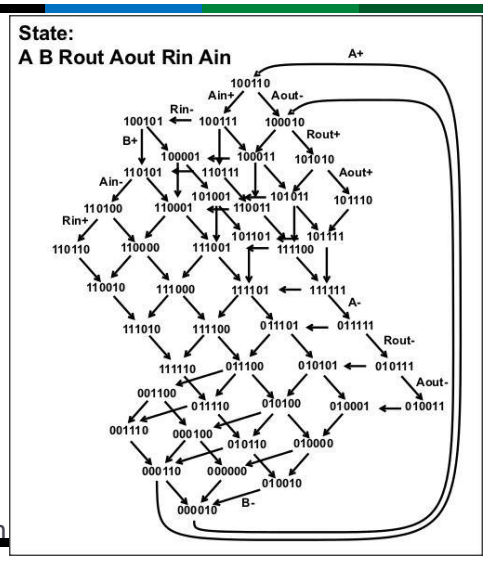
Notice how critical cycle contains two "P"s
(i.e., processing steps involving delay lines)

- **Is this an issue? Why?**

Fully-Decoupled Latch Controller

Fully-Decoupled Latch Controller – STG and SG

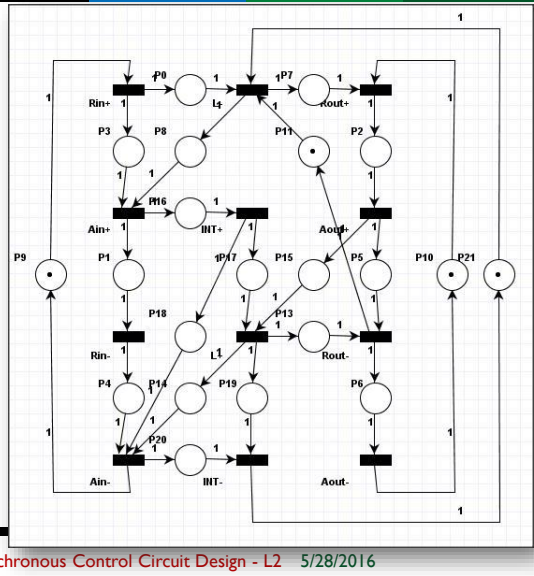




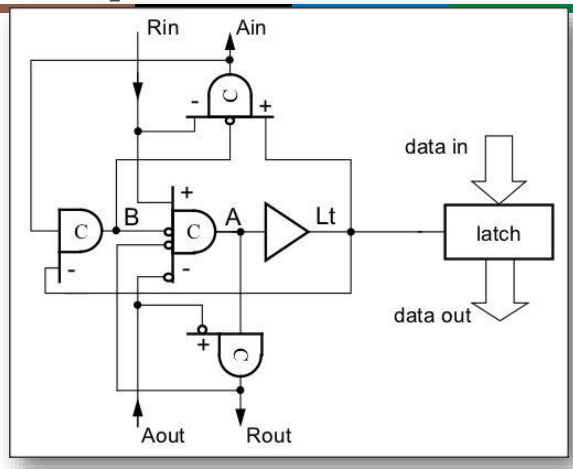
- A again is redundant
- Lt is for active-low latch,
- B is internal signal
  - Needed for implementation

Fully-Decoupled Latch Controller PTnet

- Latch Control
  - L
  - L-, L+ in PTnet
- Signal INT,
  - Transitions INT+/INT-
  - Internal
    - For implementation purposes only



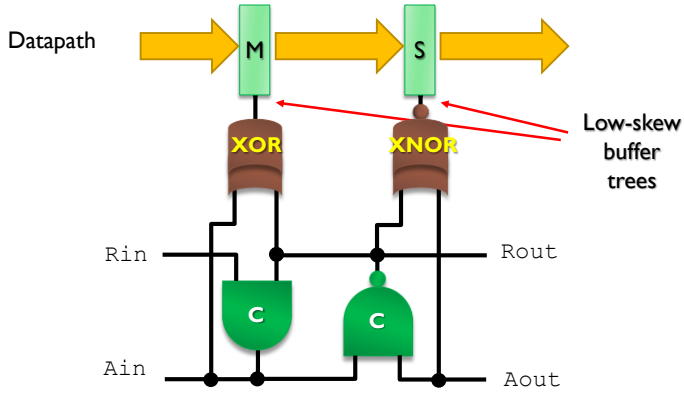
Fully-Decoupled Latch Controller Circuit



- Note Tradeoff between Concurrency and Circuit Complexity
- ***PTnet concurrency is not confluent with circuit***

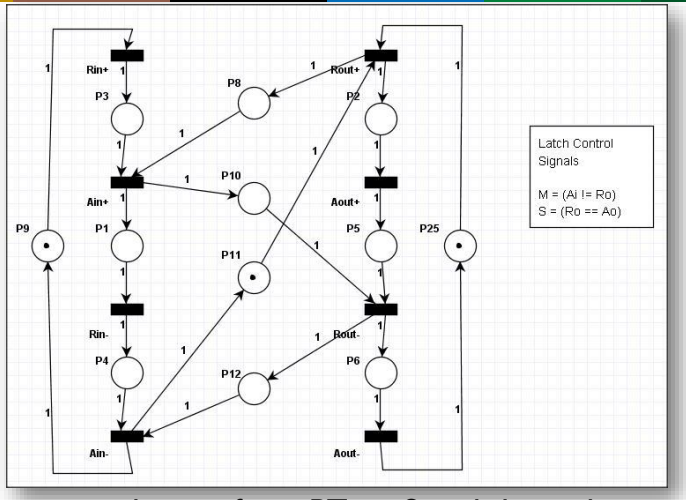
2 C-Element De-synchronisation Controller

2 C-Element De-Synchronisation Controller



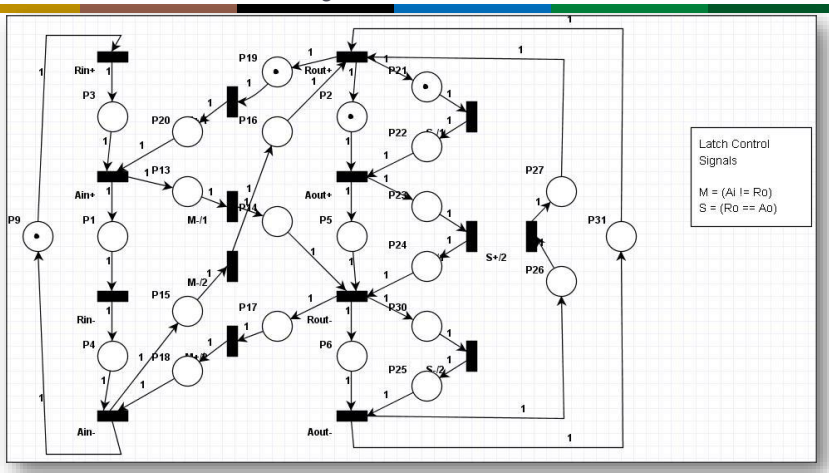
- Master Latch Enable: `M = (Ain != Rout)` [XOR gate]
- Slave Latch Enable: `S = (Rout == Aout)` [XNOR gate]

2 C-Element De-Synchronisation Controller – Handshake Signals PTnet



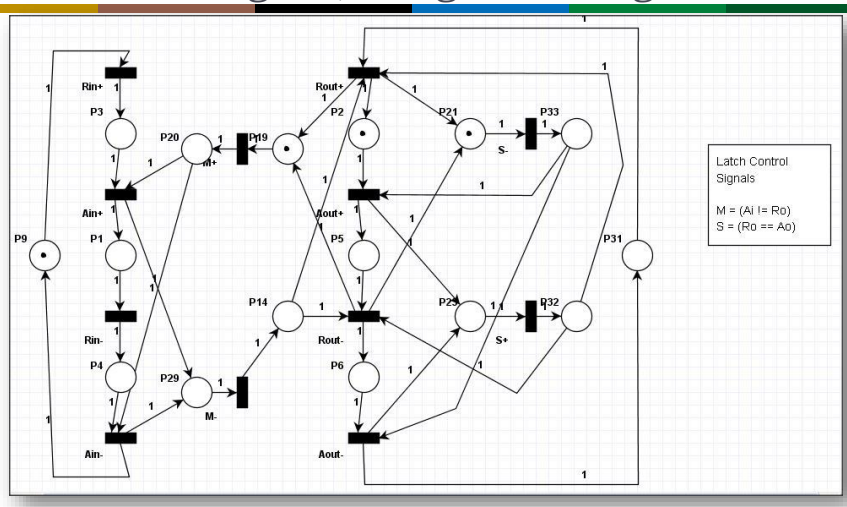
- C elements obvious from PTnet Signal dependencies

2 C-Element De-Synchronisation Controller



- Need multiple instantiations of M, S signals
  - Per control signal transition

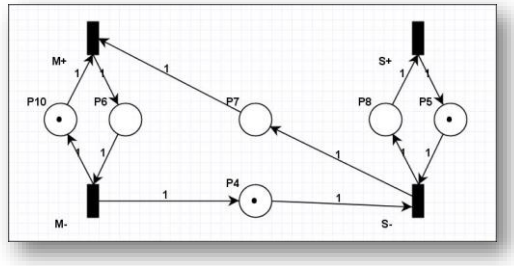
2 C-Element De-Synchronisation Controller – How to Merge M, S Signals using Choice



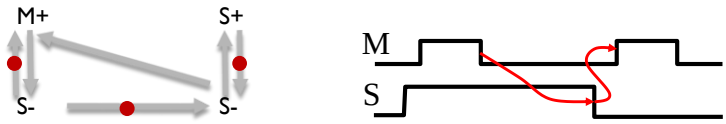
- Note: PTnet is now AC

2 C-Element De-Synchronisation Controller Analysis - PTnets

- Reduction to Latch Control Signals (Verify)

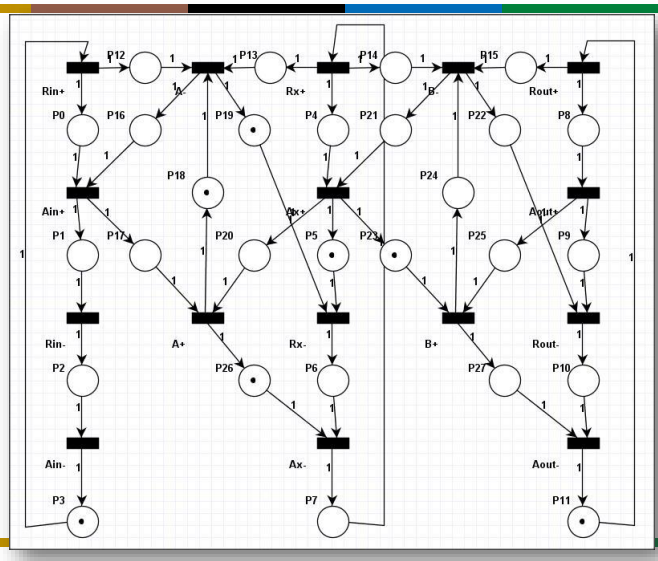


- Characteristic Pattern:



De-Synchronisation Maximum Concurrency Controller

De-Synchronisation Maximum Concurrency Controller



Bundled-Data Design: Summary

- Similar area to synchronous circuit
- Enables the use of
  - standard-cell cell libraries, conventional EDA tools
- Performance improvements due to
  - PVT variation tracking, with post-silicon tuning of delay lines
  - Pipeline stages can achieve average-case delay, using
    - **multiplexed delay lines** (operand-based delay), and/or
    - **error detecting latches**
- Power benefits
  - Dynamic voltage scaling
  - Conditional communication
- Workhorse of several Asynchronous start-ups
  - Nanochronous, Elastix, and REM