CE653 – Asynchronous Circuit Design

Instructor: C. Sotiriou

http://inf-server.inf.uth.gr/courses/CE653/

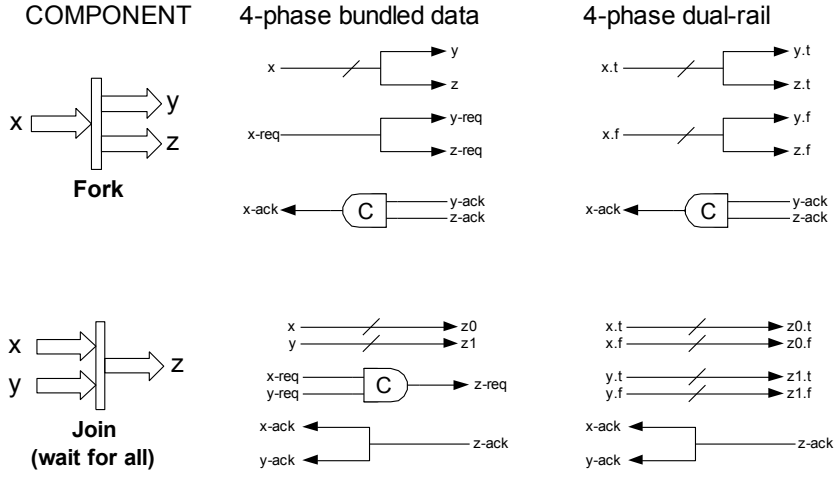
Contents

Simple 4-phase Templates

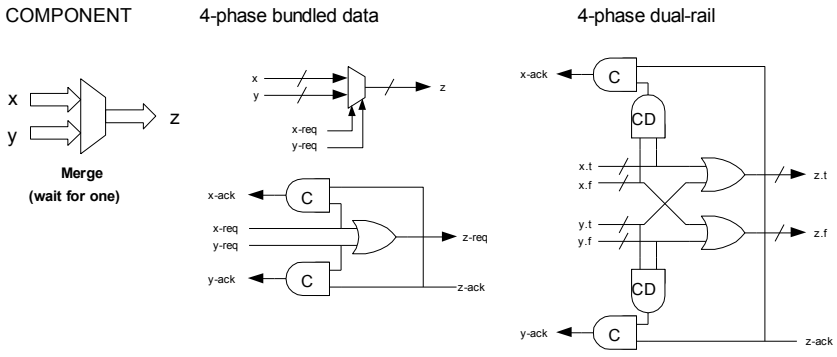
- 4-phase fork, join
- 4-phase merge
- 4-phase split (demux)

- NOTE:
  - Mutually exclusive signals must always be dual-rail encoded

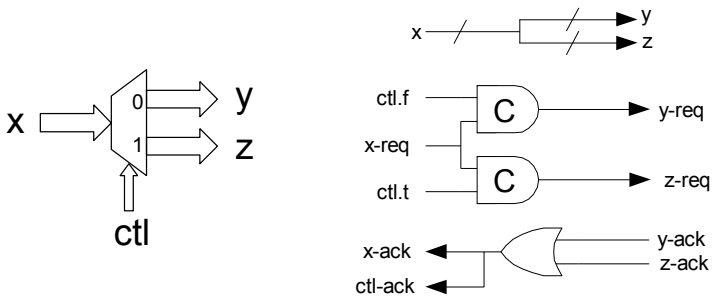
4-Phase Fork-Join



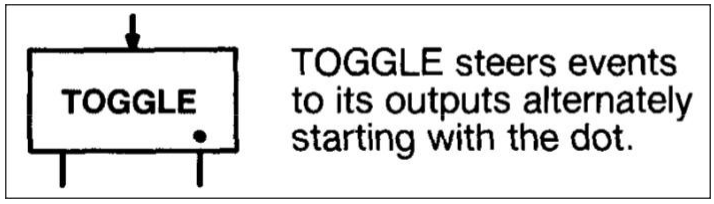
4-Phase Merge



4-Phase Split

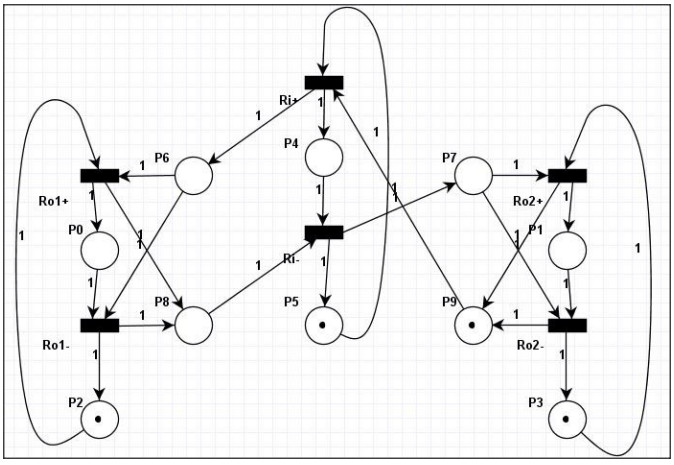


Toggle Element Specification



- An event is the rise or fall of the input
- A fall event must follow a rise event
- Thus, operation is as follows:
  - Input **rises** ➔ first output **rises**
  - Input **falls** ➔ second output **rises**
  - Input **rises** ➔ first output **falls**
  - Input **falls** ➔ second output **falls**, … etc.

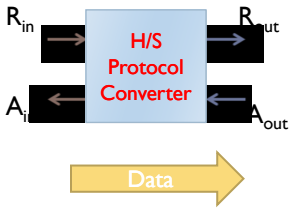
Toggle Element PTnet



- Need 2 choice places per output channel!

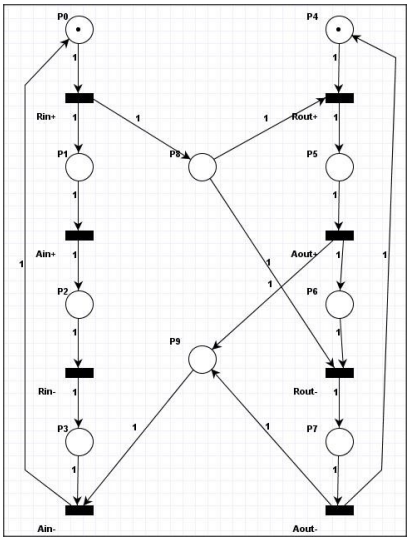
Handshake Protocol Converters

- 2-phase to 4-phase OR 4-phase to 2-phase
- Obvious specifications



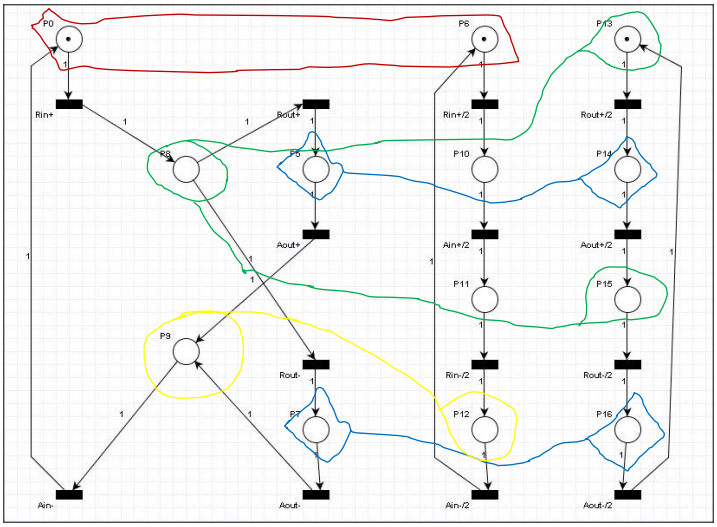
4-Phase to 2-Phase PTnet

- Causality Relationships
  - `Rin+` causes `Rout+` OR `Rout-`
  - `Aout+` OR `Aout-` cause `Ain-`

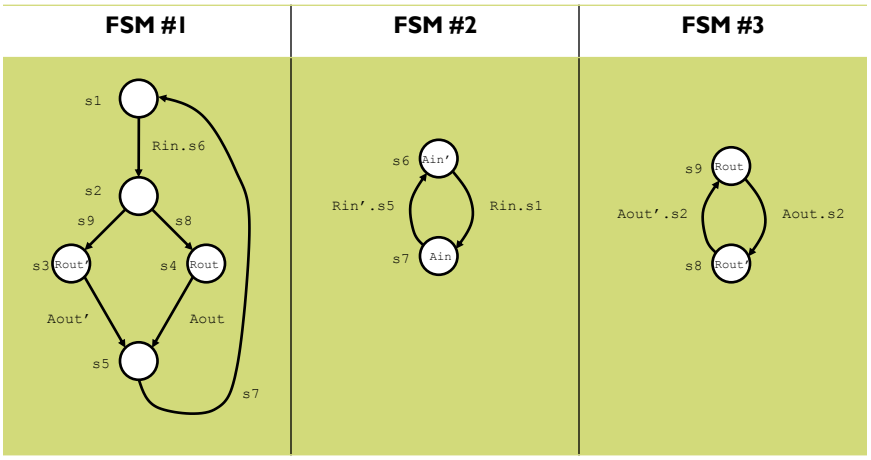


4-Phase to 2-Phase PTnet – S-Covers

- 3 SMs in S-Cover
- Can you identify where they come from?

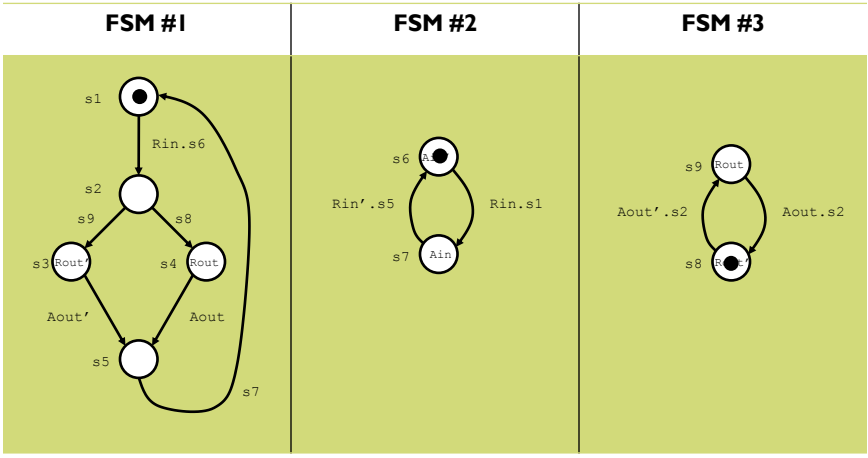


4-Phase to 2-Phase PTnet – MSFSMs



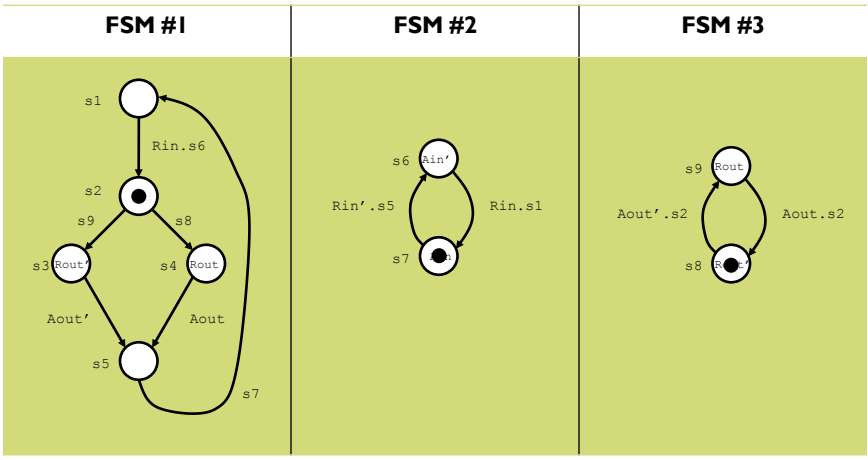
- `Ain`, `Rout` outputs
- `Rin`, `Aout` inputs

4-Phase to 2-Phase PTnet – MSFSMs



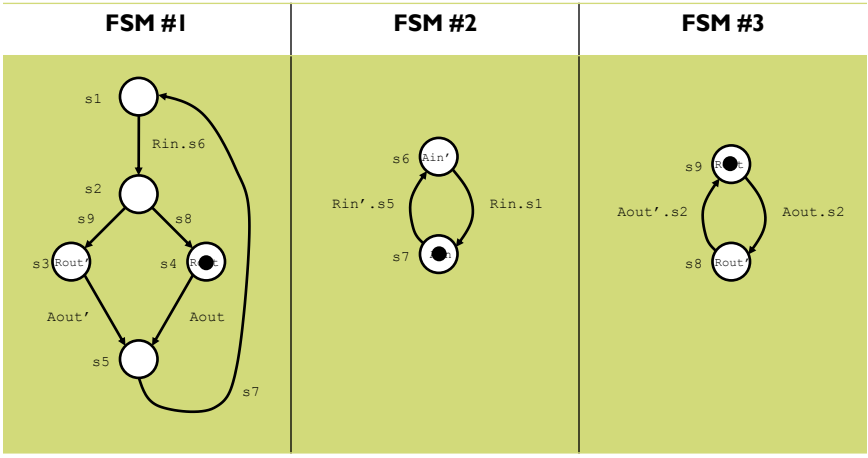
- `Ain, Rout` outputs
- `Rin, Aout` inputs

4-Phase to 2-Phase PTnet – MSFSMs



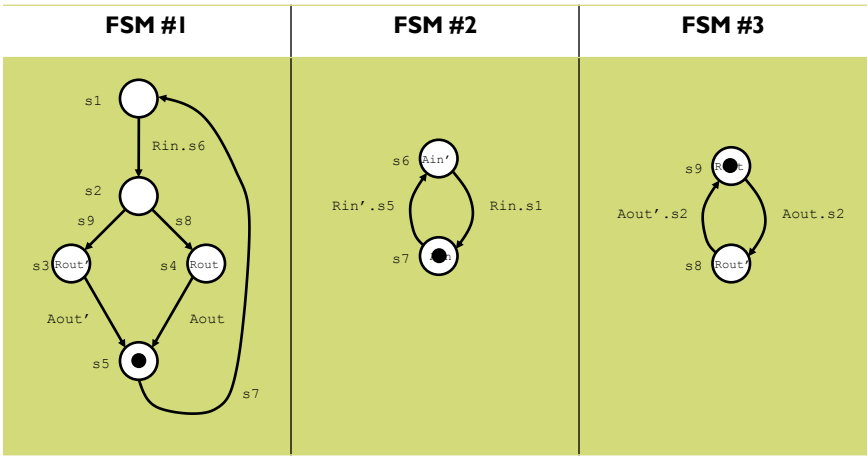
- `Ain, Rout` outputs
- `Rin, Aout` inputs

4-Phase to 2-Phase PTnet – MSFSMs



- `Ain, Rout` outputs
- `Rin, Aout` inputs

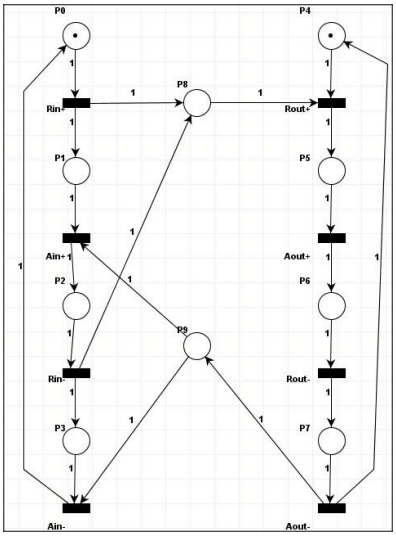
4-Phase to 2-Phase PTnet – MSFSMs



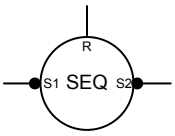
- `Ain, Rout` outputs
- `Rin, Aout` inputs

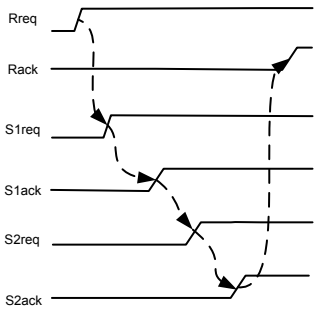
2-Phase to 4-Phase PTnet

- Causality Relationships
  - `Rin+` OR `Rin-` cause `Rout+`
  - `Aout-` causes `Ain+` OR `Ain-`



Sequencer

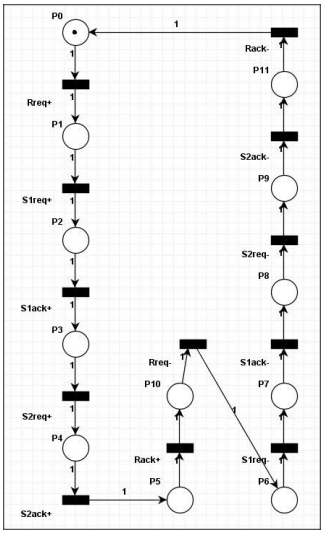




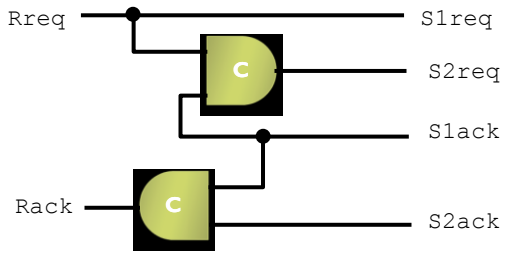
- Handshakes on S1 first then S2
  - Used to sequence operations associated with S1 and S2
- Both handshakes enclosed in handshake on R
  - Initiated by request on R
  - Terminated by acknowledgement of R

Sequencer Block - PTnet

- PTnet is FSM
  - No concurrency

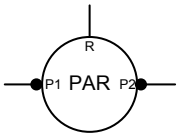


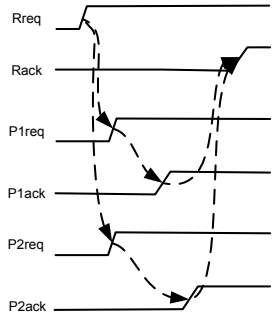
Sequencer Block – Implementation



- `S1req` identical to `Rreq`
- `S2req` is join of `Rreq`, `S1ack`
- Rack is join of `S1ack`, `S2ack`
- 2 C Elements ➔ 2 FSMs
  - where are these FSMs in Sequencer's specification?
  - are they concurrent, or sequential?

Parallel

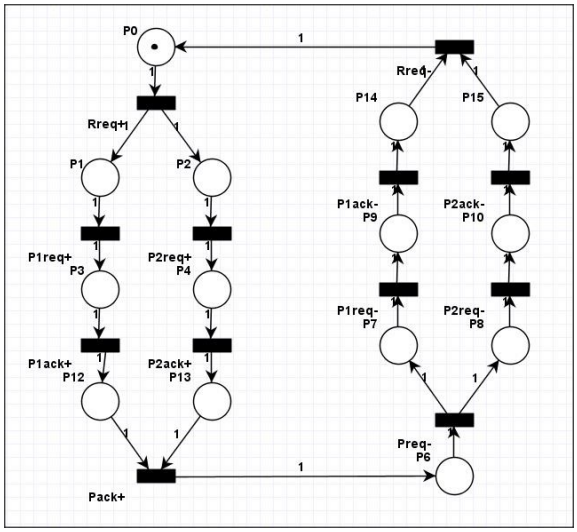




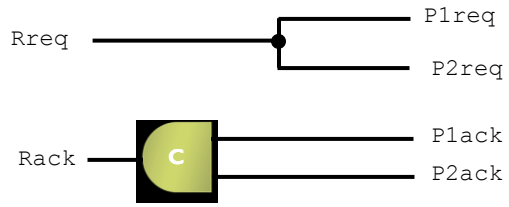
- Handshakes on P1 in parallel with P2
  - Used to execute operations associated with P1 and P2 in parallel
- Both handshakes enclosed in handshake on R
  - Initiated by request on R
  - Terminated by acknowledgement of R
    - *Both P1 and P2 must complete before R is acknowledged*

Parallel Block - PTnet

- Concurrency between two parallel handshakes
- You need two places after
  - P1ack+, P2ack+
  - P1ack-, P2ack-
  - Join, not return from choice!

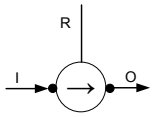


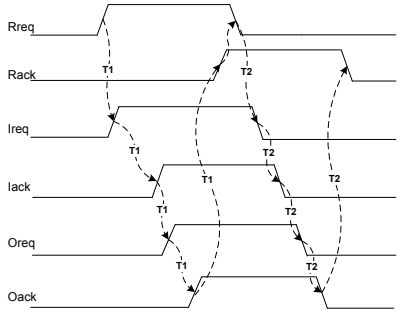
Parallel Block - Implementation



- Rreq simply forks to two requests
- Ack is join of acknowledgements
- 1 C Element ➔ 1 FSM
  - What of the concurrency in original Ptnet?
  - Where is it present in the single FSM?

Transferer

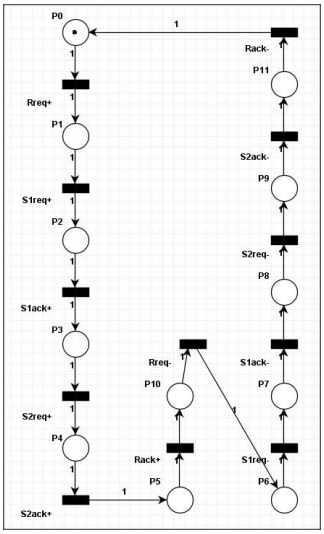




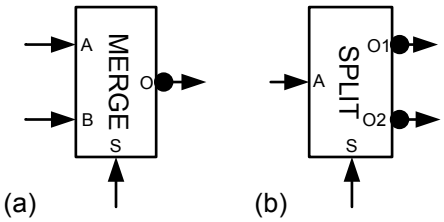
- Purpose
  - Pulls data from its input channel and pushes it onto its output channel
  - All enclosed in handshake on request channel R
- Operation
  - Waits for a request on its passive nonput port
  - Then initiates a handshake on its pull input port
  - The handshake on the pull input channel is relayed to the push output channel
  - Finally, it completes the handshaking on the passive nonput channel

Transferer PTnet

- No difference to Sequencer…

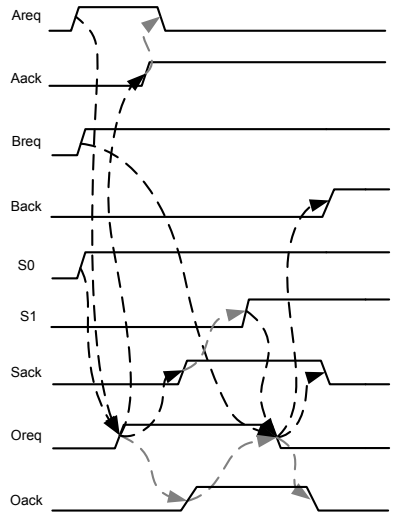


Conditional and Non-Linear Pipeines



- MERGE
  - Wait for token on S.
  - Depending on value, wait for token on either A or B and send onto O
- SPLIT
  - Wait for token on S and A.
  - Dependent upon value of S, send copy of token on A to O1 or O2

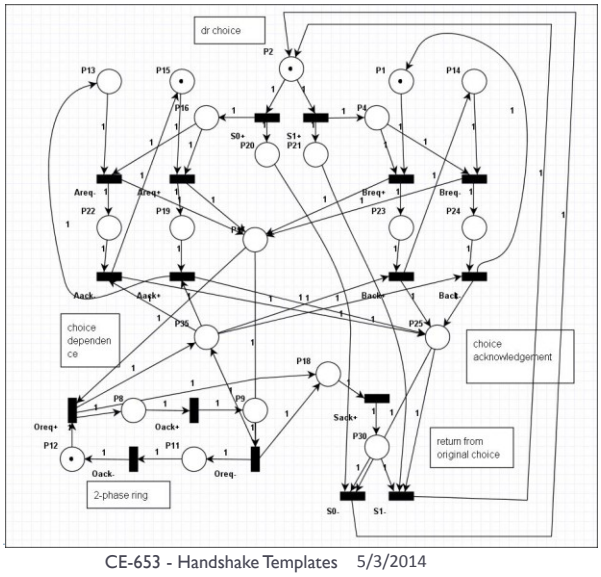
Timing Diagram of Merge



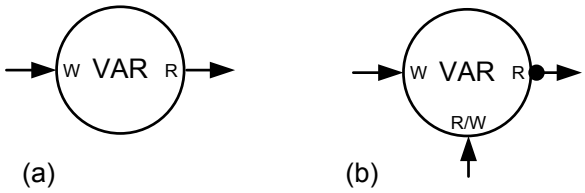
- Assumptions (in this example)
  - full-buffer two-phase handshaking
  - dual-rail select signal
- Functionality
  - Token on A consumed first
    - After token on S = 0
    - I.e., S0 changes
  - Token on B stalled until consumed second
    - After token on S = 1
    - I.e., Once S1 changes
  - Result: two tokens on O
    - First = Oreq+
    - Second = Oreq-

Merge PTnet

- **Why is it so complex?**
- PTnet must include:
  - 2-phase ring for `Oreq/Oack`
  - DR choice
  - DR return from choice
  - `Oreq*` dependence to `A/Back*`
  - DR return from choice dependence on `A/Back*`

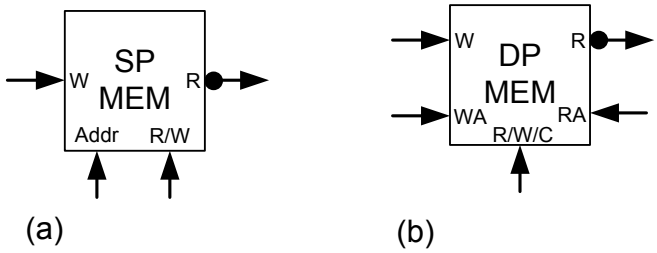


Variables



- Purpose
  - Store state of some program variable
  - W is a write port and R is a read port
- Type (a)
  - R and W are assumed to be mutually exclusive
- Type (b)
  - Waits for token on R/W 1-bit port
  - Dependent on value…
    - *Waits for request on W and store data value received or*
    - *Generates a token on R with the previously stored data value*

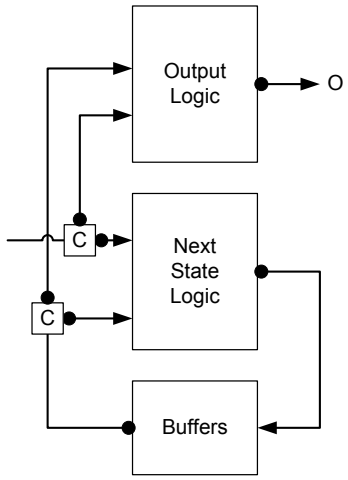
Multi-Bit Variables



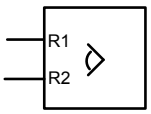
- Functionality
  - Store multi-bit state of some program variable
  - W is a write port and R is a read port
  - Addr is the address port
  - R/W controls read/write
  - C is simultaneous read and write to different addresses

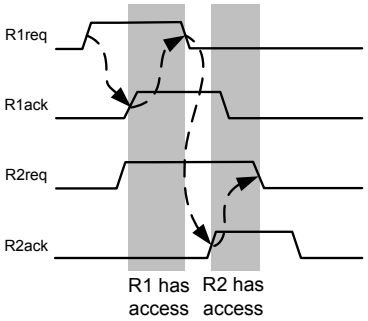
Channel-Based FSM

- Purpose
  - Implement FSM based on channels
- State
  - Stored in token in Buffers
  - Consumed every "cycle"
  - New state generated by Next State Logic
- Output
  - New output token generated in response to input token and current state
  - Copy cells needed to route input tokens and state to both NSL and OL.
- Buffers
  - One of the buffers must be token buffer, reflecting initial state of FSM



Basic 2-way Arbiter

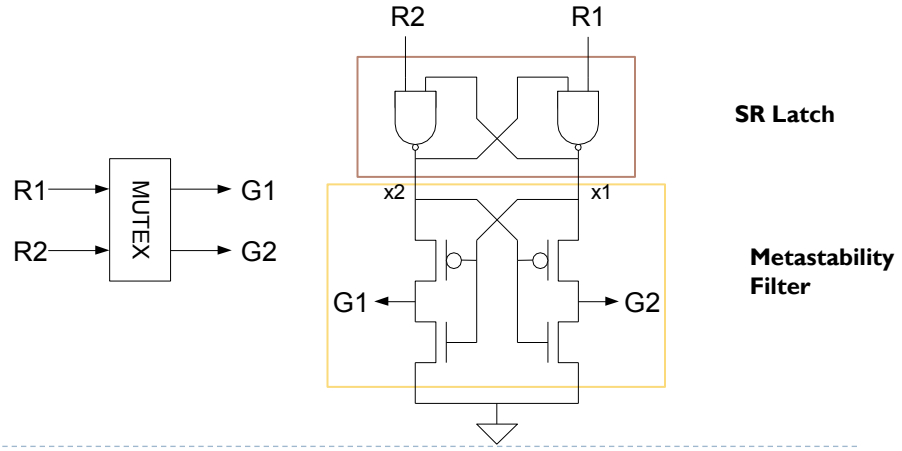




- Purpose
  - Used to control access to shared resource
- Approach
  - Acknowledge handshake on request port that arrives first, granting access
  - Requires four-phase protocol
    - winner maintains mutually-exclusive access of resource until it resets request
- Caveat
  - Make take an exponential amount of time to determine who came first when requests arrive very close together
- Sometimes called *slackless arbiter*

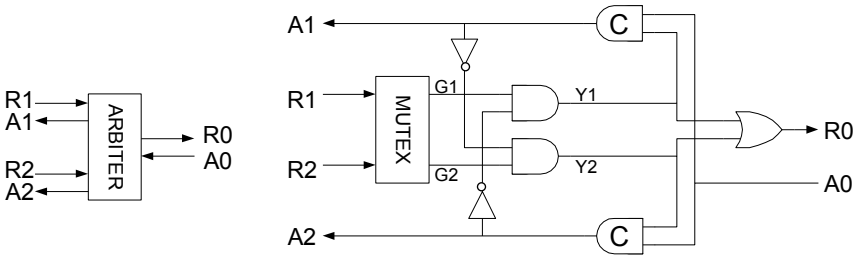
Basic 2-way Arbiter - 1

- Consists of an SR latch and a Metastability Filter
  - G1 – Grant 1, G2 – Grant 2



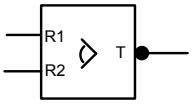
Basic 2-way Arbiter - 2

- MUTEX Cell and Handshaking Logic
- C gates used for Acknowledgements

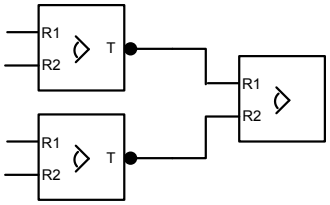


Slackless Tree Arbiters

- Slackless Tree Arbiter cell
  - Used to build multi-way arbiters

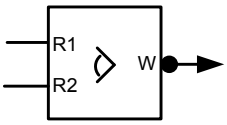


- Approach
  - Add T channel to normal arbiter
  - Delay ack of request channels until T channel acknowledged
    - Can send request on T channel as soon as any request arrives



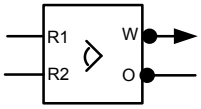
4-way Slackless Tree Arbiter

2-way (Pipelined) Arbiter



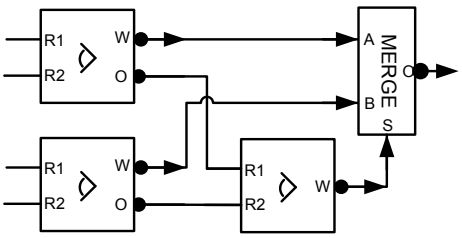
- Purpose
  - Used to control access to shared resource in a pipelined design
- Approach
  - Acknowledge of request not used to signify winner
  - Instead, additional W channel used to identify who won
  - Handshake on W simultaneously with acknowledging winning request
- Note
  - Still may take exponential time
  - In principle, this can use a two or four-phase protocol
- ***Arbiter + Pipeline Buffers all in one***

Pipelined Tree Arbiter Cells



- Tree Arbiter cell
  - Used to build multi-way pipelined arbiters
- Approach
  - Add synchronization channel O to 2-way (pipelined) arbiter
  - Send request on O channel as soon as any request arrives
- Question
  - How can use this cell as the basis of a 4-way pipelined arbiter?

Pipelined Tree Arbiter – Naïve Solution



N.B. Assume this Merge operates on 1-bit data channels

- The Problem Scenario
  - All 4 requests arrive at same time
  - Output generates 1-bit output
  - Which of the 4 requests does this 1-bit output identify?
    - Need notion of addresses to distinguish between 4 requests