ACMs – taxonomy and use cases

EEE8087

Dr Fei Xia and Dr Alex Bystrov

Question from last session

- Organizing ACMs into types according to whether overwriting and re-reading are permitted
  - It's possible to organize ACMs into four main types

	No overwriting	Overwriting (asynchrony for writer)
No re-reading	FIFO buffers (channel type)	OW buffers (signal type)
Re-reading (asynchrony for reader)	RR buffers (message type)	OWRR buffers (pool type)

Channel type ACM

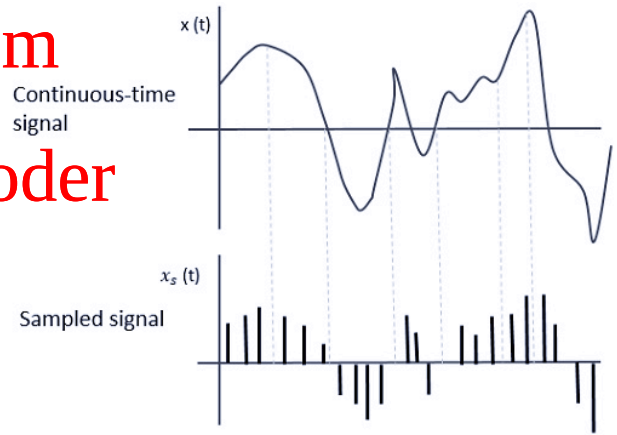
- Example: bounded FIFO buffer
  - The most widely-used type of device for asynchronous data communication
  - Exists in almost all digital systems
- Advantages
  - Many design libraries
  - Sophisticated and mature design methods
  - Directly suitable for traditional tasks in computing
- Disadvantages
  - Being used in many cases where it is not suitable or optimal

Channel type ACM

- Decouples the clock domains to some degree
  - Good for system-level design, plug and play
  - Good for real-time systems, to some extent
- Regulates the data flow
  - Smoothen the data flow
  - Make the execution on both sides more predictable and easier to organize
  - Reduce communication link congestion (better use of wires)
- Must use for transferring such data as computer programs, entertainment media, voice, etc.

Pool type ACM

- Offers full asynchrony to both writer and reader
- Good for sensing, monitoring, and control systems
- Digital mimic of an analogue wire



https://www.tutorialspoint.com/digital_communication/digital_communication_sampling.htm

Signal type ACM

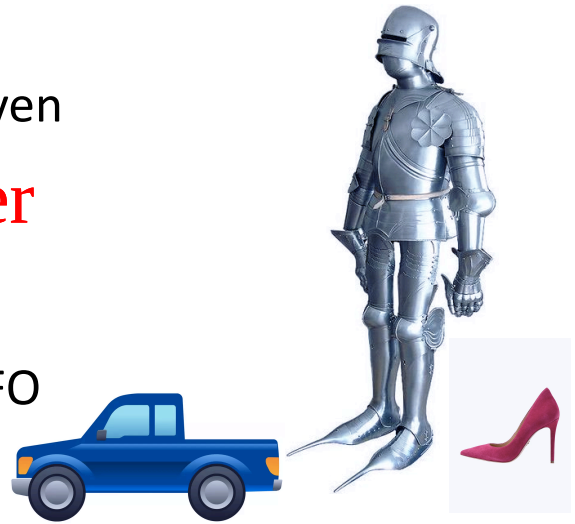
- Full asynchrony for the writer
- Reader must wait if buffer is empty
  - Does not re-use previous data items
  - Good for event-driven reader-side
  - When there is no new data, the reader-side subsystem may enter an idle state
  - Does not need to receive every item of data ever generated, only requires data freshness

Message type ACM

- Writer needs to know that every message has been acted on
  - Previous data item must have been read before a write can happen
  - Writer may enter idle mode if data in ACM has not been read
  - Effectively an event-driven writer
- Reader has full asynchrony, can re-read
  - No data freshness requirement, but a lose version of data sequencing is maintained

To compare

- All ACMs must maintain data coherence (the atomicity of each data item not to be broken)
- Signal and pool has data freshness
- Channel and message has data sequencing
- Channel: writer and reader may become event-driven
- Signal: reader event-driven
- Message: writer event-driven
- Pool: neither side event-driven
- Current reality: all of these are implemented by FIFO buffers at the bottom level – non-ideal

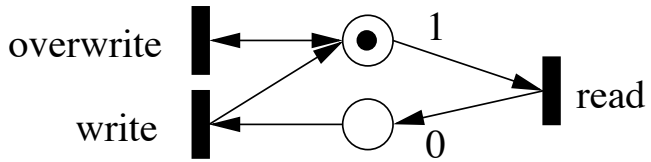


More about signal type ACM

- How many data slots do we need?
- Obviously a 4-slot protocol can be modified to realize a wonderfully safe signal but can we save on slots?
  - Try a 2-slot solution first maybe?
- Starting from the functional specification

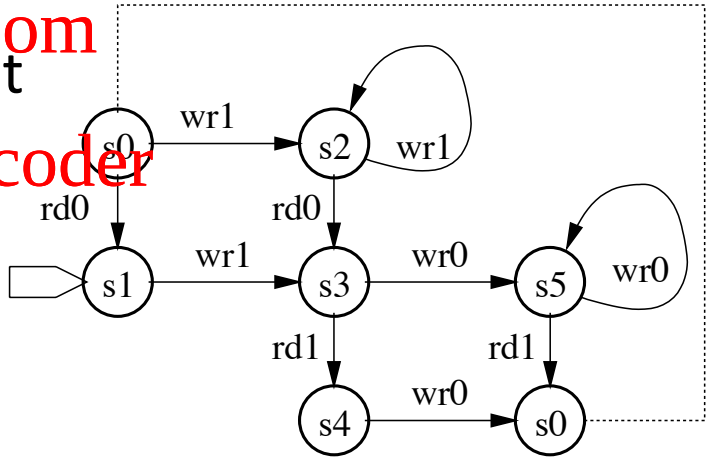
More about signal type ACM

- How many data slots do we need?
- Obviously a 4-slot protocol can be modified to realize a wonderfully safe signal but can we save on slots?
  - Try a 2-slot solution first maybe?
- Starting from the functional specification



2-slot signal

- From the functional specification, try and see if 2 slots are enough using a state graph
- Reader waiting states: s1 and s4
- Reader never repeats on the same slot
- Writer repeats on the same slot
- 2 slots potentially can work!

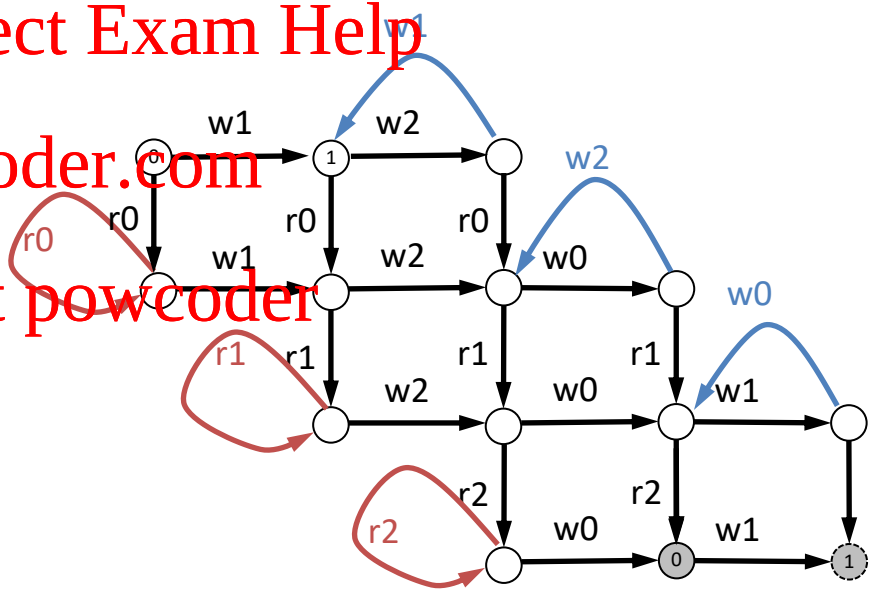


Would adding slots help?

- When writer is accessing slot 1 and reader has already read from slot 0, reader must wait
- However, writer might actually be overwriting slot 1, in which case the data being erased is more fresh (does not violate data freshness as the overwriting has not completed) than slot 0
- If there is a third slot, this would not keep the reader idle

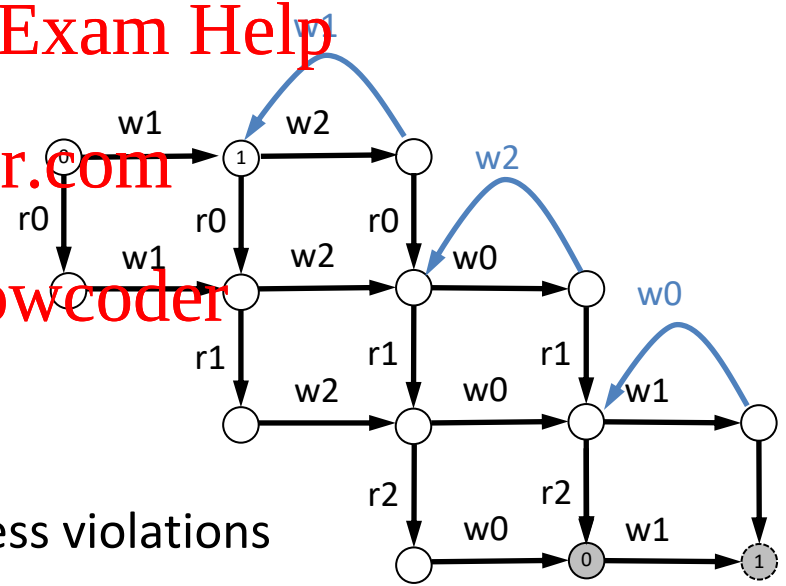
3-slot signal

- State graph can be obtained by modifying the 3-slot pool state graph
  - Pool is shown on the right ->
  - Note the red re-reading arcs



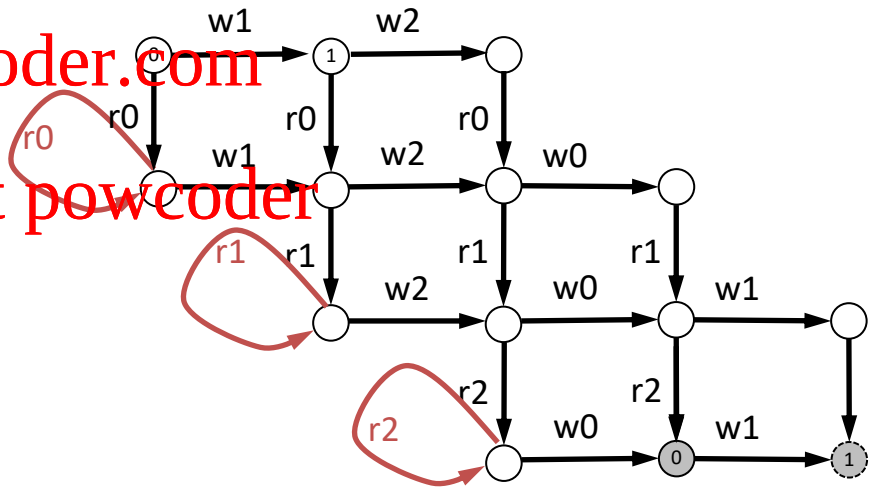
3-slot signal

- State graph can be obtained by modifying the 3-slot pool state graph
  - Just remove the re-reading arcs
  - Note that compared to the 2-slot signal, this version allows more reading accesses before it runs out of slots and needs to wait
  - Example: non-waiting r0 -> r1 -> r2
  - More communication without freshness violations



More about message

- Similar to signal, a message can be implemented using 2 slots
- But again 3 slots allows more asynchrony and less writer waiting:



What about a 3-space FIFO?

- A quiz:
- Try and derive the state graph specification for a 3-space FIFO, based on reading and writing actions
- Assume the data storage spaces in the channel ACM to be addressed in the same way as the slots in the pool, signal and message examples, i.e. spaces 0, 1, 2
- Can you list some fundamental differences in spite of similarities between the state graphs?