ACMs – the pool type

EE2307

Dr Fei Xia and Dr Alex Bystrov

Question from last session

- Why do we seem to only care for the settling of the signal/data (potentially unbounded delay) and not for the value it settles to (non-deterministic 0 or 1)?
- Answer:
  - Metastability may only happen if the data wire is changing value (from 0 to 1 or from 1 to 0) when it is sampled by a clock edge
  - Metastability settling to 0 or 1 is equivalent to sampling either just before this value change or after this value change
  - Therefore this is not an error
- However, what about multi-bit words?
  - Different bits could settle to different before/after, destroying data coherence

Reading materials

- Research papers related to these sessions

https://ieeexplore.ieee.org/document/1309114

https://ieeexplore.ieee.org/abstract/document/1134344

http://www.async.org.uk/comfort/publications/yak_xia_ps.gz

http://www.staff.ncl.ac.uk/i.g.clark/publications/ICACSD_2001.ps.gz

http://www.async.org.uk/comfort/publications/xia_async2001.pdf.gz

https://www.researchgate.net/publication/220444250_Automating_Synthesis_of_Asynchronous_Communication_Mechanisms

More than a single bit

- Large data items therefore must not be directly sampled through synchronizers
- Need a rethink
- Considering the FIFO example
  - Reader and writer are never required to be on the same clock
  - Only obliged to wait by the data state, i.e. buffer being empty or full
  - Indirect synchronization

ACMs

- An ACM includes some memory sited between a writer and a reader processes and accessible by both
- An ACM must provide access mechanisms or access procedures, through which
  - Writer can have writing access to the memory and
  - Reader can have reading access to the memory
- Memory is the necessary resource
- Access mechanisms are the control protocols

Pools

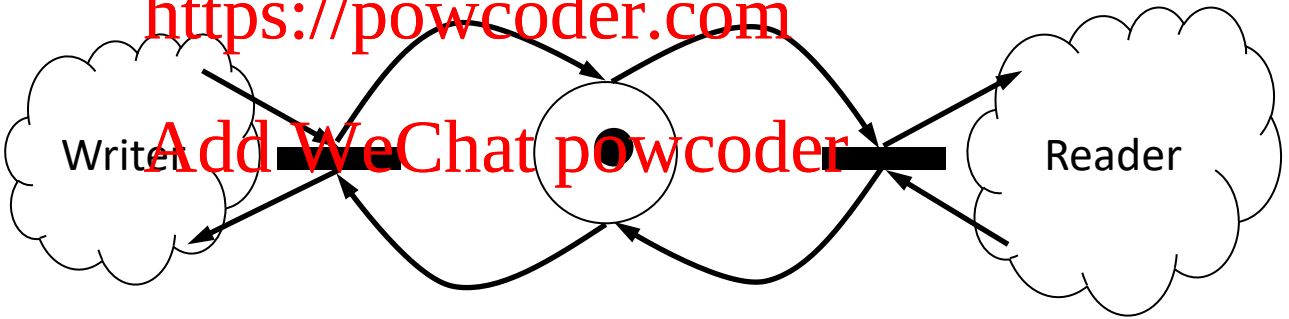
- Writer and reader may access the ACM fully asynchronously
  - No synchronization between the two processes
  - No waiting by either side
  - Overwriting
  - Re-reading
- Conceptual single-space buffer
  - Permanently holding a valid data item

Pool-type ACMs

- Enough memory to support the following access protocols
- Permanent valid data item in the memory
- Writer overwrites the memory, replacing (updating) the data item in the pool
- Reader may re-read the same data item it obtained from the previous round of reading access, if no newer item has since been updated
- Data coherence and data freshness required

Petri net model

- Always available to both writer and reader
- Reading and writing can happen in any order



Only a single physical space in the memory

- This is intuitively not enough
  - If there is no synchronization between the writer and reader, simultaneous accesses cannot be ruled out
  - Writer and reader could clash in time on the single item data memory in the pool
  - Data coherence and data freshness cannot be maintained
- What about two data spaces in the pool memory – enough to contain two data items?

Data slots

- A data slot is enough memory to hold a single item of data, designated to hold a single item of data, with an ACM's memory element
- An ACM may include multiple physical slots for storing a single valid data item between writer and reader
- The multiple slots serve to provide writer and reader with enough space to avoid simultaneous clashes on the same physical memory location
- A FIFO (channel type) buffer typically includes multiple data slots, which are used for storing multiple data items

How about data coherence and freshness?

- Data coherence
  - Complete data items written or read, not modified mid-transfer
  - No problem
- Data freshness
  - Reader does not obtain any data item that is older than the most recently fully written item in the ACM
  - How can this be maintained if there are multiple data slots, all claiming to store the same data item?

Imagine having two data slots

- If writer comes in and reader is not accessing, writer overwrites the slot containing the oldest data item
- If reader comes in and writer is not accessing, reader reads the slot containing the newest data item
  - May re-read
- If writer comes in and reader is accessing
  - Reader must be reading only one of the slots
  - Writer overwrites the other slot
- If reader comes in and writer is accessing
  - Writer must be overwriting only one of the slots
  - Reader reads the other slot

In other words

- Writer and reader never clash on the same slot
  - Data coherence is therefore maintained
- Reader always goes to the newest completely written slot and when there is a choice, writer always overwrites the slot containing the oldest data item, leaving the other slot for the reader
  - Data freshness is therefore maintained
- Or is it? Try setting up a Petri net model for a two-slot pool and verify with reachability analysis (advanced task)

Thinking a bit more

- Writer comes in and settles on slot 0 to write
- During this writing access reader comes in, has to go to slot 1
- Writer finishes writing slot 0, prepares next data item, comes back, finds reader still not finished with slot 1, so writes slot 0 again
  - Cannot wait
- Reader finishes reading slot 1, uses data item, comes back, finds writer still not finished with slot 0, so reads slot 1 again
  - Cannot wait
- Repeat same events till end of time

Thinking a bit more

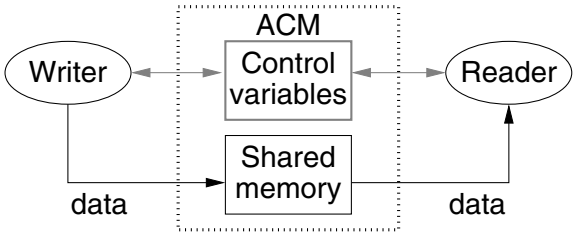
- Writer comes in and settles on slot 0 to write
- During this writing access reader comes in, has to go to slot 1
- Writer finishes writing slot 0, prepares next data item, comes back, finds reader still not finished with slot 1, so writes slot 0 again
  - Cannot wait
- Reader finishes reading slot 1, uses data item, comes back, finds writer still not finished with slot 0, so reads slot 1 again
  - Cannot wait
- Repeat same events till end of time

In this scenario, neither data coherence nor data freshness is violated, but there is no data communication!

Two slots are not enough.

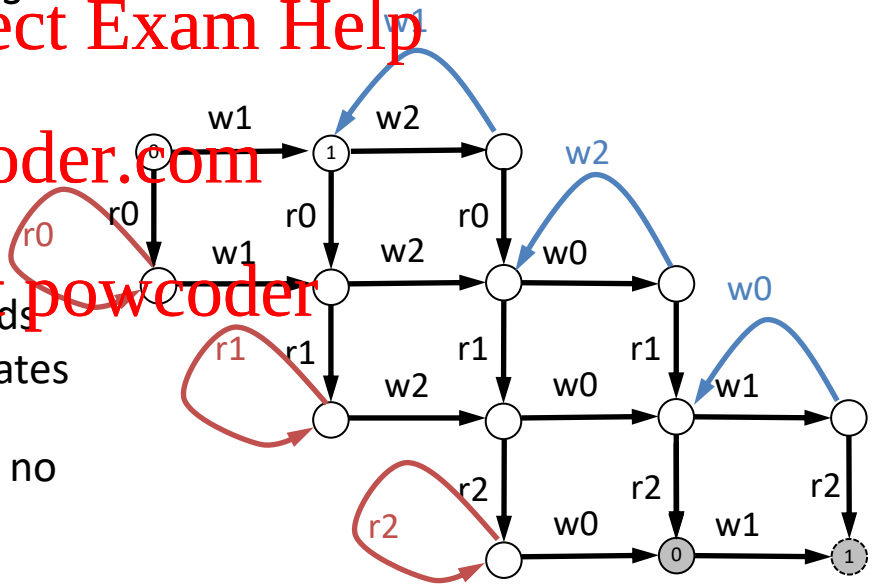
More than just data slots

- Reader needs to tell writer which slot it is accessing
- Writer needs to tell reader which slot it is accessing, and which slot contains the newest completely written data
- These can be accomplished through *control variables*
- Control variables are typically small-size and the data slots are typically large-size
  - Managing the access of large memory by passing information through small memory, i.e. single-bit latches
  - This means that you no longer care about which value metastable signal settles to, you just make sure it has settled when you use the value



Adding slots

- Reading, writing and latest slots
- Indicating we potentially can solve this with 3 slots?
- State graph of a 3-slot pool
  - No simultaneous access of the same slot by writer/reader
  - Reader always follows writer
  - If writer is stuck on a slot, reader re-reads
  - If reader is stuck on a slot, writer alternates in the other two slots
  - Not possible to have both sides live and no communication (no livelock)



Three-slots pool

- In theory, 3 data slots in the ACM memory are enough to ensure correct pool behaviour under most circumstances
- However, this requires 3-value (ternary) control variables, which do not have the correct metastability resistance
- Also, under extreme asynchrony, the 3-slot pool may fail the data requirements
- For instance, if you can fit an entire cycle of one side into the setting of a control variable of the other, data coherence may not be guaranteed
  - Setting a control variable is the fastest action compared with data access (reading/writing) and the preparation and use of data

Four-slot pool

- Therefore, in practice, we use 4-slot pools
- Control variables are in vectors of binaries, of which any time only a single bit is modified (back to Boolean, cf. ternary for 3-slot)
- Metastability-resistance proven and
- All properties verified with reachability analysis
  - Even if you fit an arbitrary number of cycles on one side into the setting of a binary control variable on the other you won't be able to violate data coherence or freshness
- All failure modes are to do with metastability leak on some of the control variables
  - When a control variable is used, it has a half value
  - Can be mitigated by lengthening the time between the reading of a control variable and its use, sort of similar to multi-stage synchronizers

Four-slot pool algorithm

Hugo Simpson's 4-slot pool:

- ACM storage is a 2x2 matrix of slots, called $d[x,y]$ with Boolean x, y
- Boolean control variables n, l and r
- Boolean two-element vectors $v[x]$ and $s[x]$
- := is assignment

Writing	Reading
wr: $d[n, \overline{s[n]}] := input$	$r0$: $r := l$
$w0$: $s[n] := \overline{s[n]}$	$r1$: $v := s$
$w1$: $l := n \Vert n := \bar{r}$	rd: $output := d[r, v[r]]$

Other types of ACMs

- Question
  - Can you think of other types of ACMs, based on allowing and not allowing overwriting and allowing and not allowing re-reading?