Asynchronous Communication Mechanisms (ACMs)

Fei Xia
EECE, Newcastle University
Data communication between two asynchronous processes

- Point to point connection

Data is a stream of items of a set type. Writer and reader are cyclic processes. Writer provides one item of data per cycle. Reader uses one item per cycle.
Synchronization

- Simple approach

![Diagram of synchronization with states 'a', 't', and 'b' and transitions between them, indicating a simple approach to synchronization.]
ACMs

- An ACM divides temporal domains
ACM model

- Each process updates its own control variables
- A process can only read the value of a control variable of another process
  - Concurrent processes executing on different timing domains
- 1-bit control variables
  - Does not require any synchronization primitives
  - Order of execution does not interfere on the correctness of the result

- The shared memory is organized as a set of $n$ cells
- A cell is a memory location that can store one data item
- Each process has its own set of control variables
History

- Atomic register [Lamport 86]
  - The problem of asynchronous communication can be solved using shared registers
  - Complex registers are built from simple ones
- ACMs [Simpson 87-03]
  - Defined a fully ACM that preserves data-coherence with 4 slots
  - Classification of ACMs
- Systematic synthesis of ACMs [Yakovlev Group 96-08]
  - ACMs are modelled using MATLAB for use in higher level systems
- Automatic synthesis of ACMs [Gorgônio, Xia 05-08]
  - State space generation (BFS based)
  - Petri net synthesis using theory of regions
  - Modular approach using Petri nets modules as building blocks
  - Modelling and verification using CPN
ACM classification

<table>
<thead>
<tr>
<th>No re-reading</th>
<th>Re-reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>No overwriting</td>
<td>BB</td>
</tr>
<tr>
<td>Overwriting</td>
<td>RRBB</td>
</tr>
<tr>
<td>OW overwriting</td>
<td>OWBB</td>
</tr>
<tr>
<td>OW re-reading</td>
<td>OWRRBB</td>
</tr>
</tbody>
</table>

- Overwriting enables the writer to be fully asynchronous
- Rereading enables the reader to be fully asynchronous
- BB stands for bounded buffer
ACMs with buffering

Writer → ACM → Control variables → Shared memory

data

Reader

data

enough memory to contain an arbitrary number of items
Buffering

$n$ cells in a ring … …
Implementation example

- RRBB with three cells from an interleaving specification
**RRBB**

- Adding “silent actions”

λ: writer silent actions;  
µ: reader silent actions

prepare to reread
RRBB synthesis

var \( w: 0..n-1; r: 0..n-1 \); initialized sensibly (say \( r = w-1 \)) and initialize data items in the cells.

<table>
<thead>
<tr>
<th></th>
<th>writer</th>
<th>reader</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{wr: write cell } w; )</td>
<td>( \text{r0: if } (r+1 \mod n) \neq w )</td>
</tr>
<tr>
<td></td>
<td>( \text{ww: wait until } r \neq w+1; )</td>
<td>then ( r := (r+1 \mod n); )</td>
</tr>
<tr>
<td></td>
<td>( w0: w := (w+1 \mod n); )</td>
<td>( \text{rd: read cell } r; )</td>
</tr>
</tbody>
</table>

School of Electrical, Electronic and Computer Engineering

Newcastle University
RRBB modular design

Writer

data

advance

Writer

data

Reader

data

advance

Reader

ACM cell $i$

$w$ and $r$

cell memory

ACM cell $i+1$

$w$ and $r$

cell memory
Modular design algorithm

```plaintext
var w: 0..1; r: 0..1; initialized sensibly (one cell has w=1 and another has r=1, all others being 0) and initialize data in the cells.

<table>
<thead>
<tr>
<th>Writer</th>
<th>Reader</th>
</tr>
</thead>
<tbody>
<tr>
<td>wr: write;</td>
<td>r0: if wnext=0 then</td>
</tr>
<tr>
<td>w0: w:=0; wnext:=1;</td>
<td>begin r:=0; mext:=1;</td>
</tr>
<tr>
<td>ww: wait until mext=0;</td>
<td>advance to next end</td>
</tr>
<tr>
<td>wa: advance to next;</td>
<td>rd: read;</td>
</tr>
</tbody>
</table>
```
Overwriting

- Needs more than one “slot” per cell
- This will allow the writer to “overtake” the reader if overwriting the oldest item in the buffer
- Item to be overwritten can be
  - Oldest in the buffer
  - Newest in the buffer
  - Some other item (does not make as much sense as the previous choices)
- Also has to do with if the ACM is organized as a FIFO, a stack or a random bag
3-Cell OWRRBB
3-cell OWRRBB

writer

Control variables

cell 0  cell 1  cell 2

slot 0

$X_{00}$  $X_{10}$  $X_{20}$

slot 1

$X_{01}$

reader

One-hot encoding (1-bit variable)

Control variables

School of Electrical, Electronic and Computer Engineering

Newcastle University
Modular OWRRBB algorithm

Var $w: 0..1; r: 0..1; over: 0..1$ (this cell is overwritten); $release: 0..1$ (writer releases cell for reader);

<table>
<thead>
<tr>
<th>Writer</th>
<th>Reader</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w0: over_{\text{next}}:=0;$</td>
<td>$r0: \text{if } over=1 \text{ then}$</td>
</tr>
<tr>
<td>if $w \neq r$ then $over:=1;$</td>
<td>begin $over:=0;$</td>
</tr>
<tr>
<td>$w=(\text{not})r;$</td>
<td>advance to next end</td>
</tr>
<tr>
<td>$wr: \text{write slot } w;$</td>
<td>$r:=w;$</td>
</tr>
<tr>
<td>$wa: release:=1;$</td>
<td>$rd: \text{read slot } r;$</td>
</tr>
<tr>
<td>$\text{release}_{\text{next}}:=0;$</td>
<td>$ra: \text{if } release_{\text{next}}=1 \text{ then}$</td>
</tr>
<tr>
<td>advance to next;</td>
<td>advance to next</td>
</tr>
</tbody>
</table>
Automatic synthesis

Functional specification (3-cell re-reading ACM)

```java
writer() {
    while(true){
        write cell w;
        w = w+1 mod 3;
        wait until r != w;
    }
}
```

```java
reader() {
    while(true){
        if((r+1 mod 3) != w { 
            r = r+1 mod 3;
        } read cell r;
    }
}
```
Acknowledgements

Alex Yakovlev
Hugo Simpson
Ian Clark
Eric Campbell
Jordi Cortadella
Kyller Gorgonio