Modelling and analysis of Asynchronous Communication Mechanisms

Ian Clark (*)    Fei Xia (**)    Alex Yakovlev (**)    

* King’s College London
** University of Newcastle upon Tyne

Collaborative EPSRC research project

Ian.Clark@kcl.ac.uk

ACiD–WG Workshop             Neuchatel   12th February 2001
Objectives

- To study ACM algorithms and test their operating properties against desired operating properties.

- Petri net methods can be used to test all existing and new solutions to the problem.

- In this case the method is used to independently test the algorithms produced by synthesis using regions presented in the previous talk.
Outline

• Basic modelling techniques

• Data communication properties

• Monitoring subnets and Enable places

• Metastability

• ‘OR’ arcs

• Example solutions

• Conclusions
Petri Nets

Model each ACM instruction separately rather than use an abstract model.

Example $A := \neg B$

- **B=1**
  - $A=0$
  - $A=1$
- **B=0**
  - $A=1$
  - $A=0$

- **Set of places for each variable – only 1 marked**
- **Reference arcs because 2 processes**
- **Control loop for sequential operation**

![Petri Net Diagram](image)

- **Start**
- **A=0, B=1**
- **A=1, B=0**
- **Finish**
Atomic models are not always accurate.

In hardware latches have set-up and hold times, plus propagation delay.

\[ A := \text{not } B \]
Behavioural Properties

Coherence — in a record with multiple fields, the reader must obtain a version in which all or none are updated

Sequencing — data items must be read in the order written, even though not every item need be read

Freshness — data items must not be ‘out of date’ - the reader must get the most recently available data

Full/Conditional Asynchrony — each accessing process may or may not be blocked by the data state, i.e. read/unread or full/empty or by the other processes access to the slots
Monitoring subnets

**freshness and sequencing**

<table>
<thead>
<tr>
<th>wr</th>
<th>w0&amp;1</th>
</tr>
</thead>
</table>

| r0 | rd   |

| inst a | inst b |

| inst a | mon   | inst b |

**coherence**

- writing 1
- reading 1 (lost)

**asynchrony**

- each state in the state space should have 2 next states
- for conditional asynchrony some states will have only one next state – these can be checked
Operational Properties

Metastability has to be considered

sync. domain A

sync. domain B

D Q

mid-value

unknown period
usually many times the quoted propagation delay

clock edge

setup time

propagation delay

1

0

M

1
OR arcs in Petri Nets

When modelling a system with 2 or more concurrent elements shared variables are usually referenced and not produced and consumed.

Normal PN arcs are AND

for example \( A := B \) \( A, B \in \{0, 1\} \)

However with OR arcs

50% saving with binary
However the real savings come with variables which are > binary

i.e. \( A := B \quad A, B = \{0, 1, 2, 3\} \)

Whole model 4 transitions as opposed to 16 transitions normally

75% Saving in model size

There will only ever be one token in the set and therefore safe

\[
(B = 0 \land (A = 0 \lor A = 1 \lor A = 2 \lor A = 3)) = (B = 0 \land (A = 0))
\]
2 slot signal algorithm

writer

\[ g := 1; \]
\[ \text{write slot } w; \]
\[ g := 0; \]
\[ w := \neg r; \]

Or

writer

\[ \text{write slot } w; \]
\[ w := \neg r; \]

reader

\[ r := \neg r; \]
\[ \text{while } w = r \text{ or } g = 1 \text{ then} \]
\[ \text{wait} \]
\[ \text{read slot } r; \]

Or

reader

\[ r := \neg r; \]
\[ \text{while } w = r \text{ then} \]
\[ \text{wait} \]
\[ \text{read slot } r; \]
Data latency

\[ w=0 \quad r=0 \]
write slot 0
\[ w:=\neg r \ (1) \]
\[ r:=\neg r (1) \]
while \( w=r \)
write slot 1
\[ w:=\neg r \ (0) \]
read slot 1

Check for reading/re-reading
3 slot signal algorithm

writer

write slot w;
1 := w;
w := differ(l,r)
((2,3,2),(3,3,1),(2,1,1))

reader

while l==r then
    wait
r:=l;
read slot r;
2 slot message/command

Dual of the signal

writer

while \( w = r \) or \( g = 1 \) then
  wait;
  write slot \( w \);
  \( w := \neg w \);

OR

while \( w = r \) then
  wait;
  write slot \( w \);
  \( w := \neg w \);

reader

\( r := \neg w; \)
\( g := 1; \)
read slot \( r \);
\( g := 0; \)
Simpsons 3 slot pool algorithm

\[
d[n] := \text{input}; \\
l := n; \\
n := \text{differ}(l,r); \\
r := 1; \\
\text{output} := d[r];
\]

PN analysis showed that this 3 slot fully asynchronous algorithm only fails because of the interleaving between the 2 stage hardware used to implement the 3 control statements.

Tests were performed to see what happened if the 3 control statements were implemented atomically, i.e. each control variable was mutually exclusively accessed by each process.

This proved successful and maintained all properties, therefore hardware implementation using mutexs proved valid.
Conclusions

PN’s provide a method to model and analyse ACM algorithms

Can be used on new and existing designs

Helps to show operating modes clearly during behavioural property tests

Used here to prove that the solutions produced by synthesis using regions are correct