Towards safer Concurrent Device Drivers
Modeling RMoX Drivers in CSP

Martin Ellis

School of Computing
University of Kent

Communicating Process Architectures, 2011
1 Motivation
   - Making Safer Concurrent Device Drivers.
   - Previous Work

2 The Problem

3 Our Technique
   - Resource Driver
   - Extending CSP generation.
Outline

1 Motivation
   - Making Safer Concurrent Device Drivers.
   - Previous Work

2 The Problem

3 Our Technique
   - Resource Driver
   - Extending CSP generation.
The World So Far...

We want "safe" and "correct" concurrent device drivers.

- Device Driver / Kernel interface well understood.
- Device Driver / Hardware interface less so.
1 Motivation
   - Making Safer Concurrent Device Drivers.
   - Previous Work

2 The Problem

3 Our Technique
   - Resource Driver
   - Extending CSP generation.
Most previous work done on kernel/driver interfaces.

- Slam.
  - Static analysis of Windows drivers.
  - Tried to help prevent kernel crashes (BSOD).

- DDVERIFY
  - Static analysis of Linux drivers.
  - Handles concurrent Linux drivers.

- Fred Barne’s work on modeling drivers is CSP.
  - Prove deadlock freedom of RMoX drivers.
  - Only considered the Driver/Kernel interface.

- Driver synthesis.
  - Chinook.
  - Mattias I’Nils’ and Axel Jantsch’s work with ProGram.
Device Driver Complexities.

- Memory mapped IO vs port mapped IO.
- Overloaded addresses.
- Bitfields.
- Concurrent access.
### Motivation
Making Safer Concurrent Device Drivers.

### Previous Work
The Problem

### Our Technique
Resource Driver Extending CSP generation.

### Summary
Select and Access ports

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdy</td>
<td>parity</td>
<td>data</td>
<td>reserved</td>
<td></td>
</tr>
</tbody>
</table>

#### bitfield port

<table>
<thead>
<tr>
<th>0x200</th>
<th>0x216</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register Select</td>
<td>Access</td>
</tr>
</tbody>
</table>
### Motivation

Making Safer Concurrent Device Drivers.

### Previous Work

The Problem

Our Technique

Resource Driver Extending CSP generation.

Modelling Drivers?

### Summary

<table>
<thead>
<tr>
<th>bitfield port</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>rdy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Select and Access ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>Register Select</td>
</tr>
<tr>
<td>Access</td>
</tr>
</tbody>
</table>

\{ 0x200 \}

\{ 0x216 \}
Concurrent Access

- Placed memory/channels
- Circumvents parallel usage checking
- All the usual issues with data aliasing.
Outline

1 Motivation
   - Making Safer Concurrent Device Drivers.
   - Previous Work

2 The Problem

3 Our Technique
   - Resource Driver
   - Extending CSP generation.
- Kernel / Driver interface made of well understood occam channels.
- Hardware / Driver interface made of "magic".
- Abstract things into nice occam channels.
- Kernel / Driver interface made of well understood occam channels.
- Hardware / Driver interface made of "magic".
- Abstract things into nice occam channels.
Resource Driver

So what does the resource driver give us?

- Primitives for reading registers "correctly"
- Sanity checks (no use before declaration etc)

- These runtime checks are slow though. 😞
So what does the resource driver give us?

- Primitives for reading registers "correctly"
- Sanity checks (no use before declaration etc)
- These runtime checks are slow though. 😞
Motivation

- Making Safer Concurrent Device Drivers.
- Previous Work

The Problem

Our Technique

- Resource Driver
- Extending CSP generation.
Extending KRoC

KRoC’s CSP model generation has been extended.

- Now includes details of variant channels.
- Number of parameters.
- Values known known at compile time.
Protocol $\rightarrow$ CSP

**occam**

```
PROTOCOL P.RES
CASE
  a; INT
  b; INT; BYTE
;
```

**CSP**

```
U = (−999)
NUMBER = \{U\} \cup \{0..99\}
channelres : a.NUMBER | b.NUMBER\_NUMBER
```
Protocol → CSP

**occam**

```
PROTOCOL P.RES
CASE
  a; INT
  b; INT; BYTE
:
```

**CSP**

```
U = (-999)
NUMBER = \{U\} \cup \{0..99\}
channelres : a.NUMBER | b.NUMBER.NUMBER
```
Concurrent Device Drivers

Martin Ellis

Motivation
Making Safer Concurrent Device Drivers.

Previous Work
The Problem

Our Technique
Resource Driver
Extending CSP generation.
Modelling Drivers?

Summary

Communication → CSP

**occam**

```plaintext
PROC x (chan P.RES res!)
SEQ
  res ! a; x
  res ! b; y; z
:
```

**CSP**

```plaintext
X(res) = res.a!(U) →
  res.b!(U).(U) →
  SKIP
```
## Concurrent Device Drivers

**Martin Ellis**

### Motivation
Making Safer Concurrent Device Drivers.

### Previous Work
The Problem

### Our Technique
Resource Driver Extending CSP generation.

### Modelling Drivers?

### Summary

---

**occam**

```occam
PROC x (chan P.RES res!)
SEQ
  res ! a; x
  res ! b; y; z
:
```

**CSP**

```csp
X(res) = res.a!(U) \rightarrow
       res.b!(U).(U) \rightarrow
       SKIP
```
Concurrent Device Drivers
Martin Ellis

Motivation
Making Safer Concurrent Device Drivers.

Previous Work
The Problem
Our Technique
Resource Driver
Extending CSP generation.
Modelling Drivers?

Summary

Constant Propagation

**occam**

```
PROC x (chan P.RES res!)
  SEQ
    res ! a; 42
    res ! b; y; z
```

**CSP**

```
X(res) = res.a!(42) \rightarrow
        res.b!(U).(U) \rightarrow
        SKIP
```
Generated CSP

\[
PPORT\_HANDLER_g = \\
(srv.\text{InPResResInDeclare}?vv.pa \rightarrow PPORT\_HANDLER \sqcap STOP) \sqcap \\
(srv.\text{InPResPortInDeclare}?vv.pa.pc \rightarrow PPORT\_HANDLER \sqcap STOP) \sqcap \\
(srv.\text{other1} \sqcap srv.\text{other2} \sqcap \ldots) ; PPORT\_HANDLER
\]
Externalising internal choice.

"Tweeked" CSP

\[
PPORT\_HANDLER_t(RESS, PORTS) = (srv.InPResResInDeclare?vv.pa \to \\
PPORT\_HANDLER(RESS, \{pc\} \cup PORTS) \\
\langle \text{pa} \notin \text{RESS} \rangle \triangleright \text{STOP} \rangle\]

\[
PPORT\_HANDLER(RESS, \{pc\} \cup PORTS) \\
\langle \text{pa} \in \text{RESS} \rangle \land (\text{pc} \notin PORTS) \triangleright \text{STOP} \rangle\]

\[
(sv.srv.other1 \Box sv.srv.other2 \Box \ldots) ; PP\_HANDLER(RESS, PORTS)
\]
Refinement

\[ \text{SYSTEM\_PRES\_DRIVER}_t \sqsubseteq_T \text{SYSTEM\_PRES\_DRIVER}_g \]

\[(\text{SYSTEM\_PRES\_DRIVER}_t \parallel \text{DEVICE\_DRIVER}) \text{ deadlocks?}\]
Refinement

\[ \text{SYSTEM\_PRES\_DRIVER}_t \sqsubseteq_T \text{SYSTEM\_PRES\_DRIVER}_g \]

\[ (\text{SYSTEM\_PRES\_DRIVER}_t \parallel \text{DEVICE\_DRIVER}) \text{ deadlocks?} \]
And It Works...

This works... "ish"

- The state space is huge.
- The $NUMBER$ type has to be narrow.
And It Works...

This works..."ish"

- The state space is huge.
- The `NUMBER` type has to be narrow.
And It Works...

This works..."ish"

- The state space is huge.
- The \textit{NUMBER} type has to be narrow.
Can we write specification for drivers in CSP?

\[\begin{align*}
\text{READ} &= 0 \\
\text{WRITE} &= 1 \\
\text{DEFINE}_{DSP} &= (\text{port.InPResPortInDefineRes!0.220} \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{port.InPResPortInDefineRes!0.240}) \rightarrow \text{DECLARE \cdot PORTS}_{DSP} \rightarrow \text{RESET}_{DSP} \rightarrow \text{SKIP} \\
\text{DECLARE \cdot PORTS}_{DSP} &= (\text{port.InPResPortInDeclare!0.U.1.6.7.8.WRITE} \mid \mid \mid \text{port.InPResPortInDeclare!0.U.1.A.7.8.READ} \mid \mid \mid \text{port.InPResPortInDeclare!0.U.2.C.7.8.WRITE} \mid \mid \mid \text{port.InPResPortInDeclare!0.U.3.C.7.8.READ} \mid \mid \mid \text{port.InPResPortInDeclare!0.U.4.E.7.8.READ}) \rightarrow \text{SKIP} \\
\text{RESET}_{DSP} &= \text{port.write!0.1} \rightarrow \text{port.setDelay!3} \rightarrow \text{port.write!0.0} \rightarrow \text{port.wait!1.\#AA.100.3} \rightarrow \text{SKIP}
\end{align*}\]
Summary

- We can produce nice models of device drivers.
- We have an issue with state space.
- How can we model drivers’ expected behaviour?
- How do we deal with the state space issues in FDR?
For Further Reading I

Ball, T. and Cook, B. and Levin, V. and Rajamani, S.K.  
*SLAM and Static Driver Verifier: Technology transfer of formal methods inside Microsoft.*  

Barnes, F.R.M. and Ritson, C.G.  
Checking process-oriented operating system behaviour using CSP and refinement.  