Motivation

• The joys of driver development
  – Poor hardware documentation
  – Poor OS documentation
  – Debugging drivers is hard
A problem has been detected and Windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c
Conventional driver development

OS interface spec

device spec

driver.c
Driver synthesis with Termite

Formal OS interface spec

Formal device spec

driver.c
Driver synthesis with Termite

Formal OS interface spec

Formal device spec

driver.c
Termite is *not* ...

- ... a domain-specific language for drivers

```plaintext
FEATURE device_read {
    OUTPUT char data;
    poll LSR.dr until (LSR.dr == 1) data = RBR
    ERROR (LSR.oe == 0x1 || LSR.pe == 0x1);
}
```

- ... a register description language

```plaintext
device logitech_busmouse(base : bit[8] port@{0..3}) {
    register sig_reg = base @ 1 : bit[8];
    variable signature = sig_reg, ... : int(8);
}
```

- ... a theoretical exercise
  - the goal is to synthesise efficient drivers for real-world devices (network, storage, audio, etc.)
Driver synthesis with Termite

- Where do specifications come from?
  - A device spec can be as complex as the driver
  - Use existing device specifications developed by hardware designers
Hardware Design Workflow

- Informal specification
- High-level model
- Register-transfer-level description
- netlist
Hardware Design Workflow

- Informal specification
  - High-level model
    - Register-transfer-level description
      - netlist

- Low-level description: registers, gates, wires.
- Cycle-accurate
- Precisely models internal device architecture and interfaces
Hardware Design Workflow

- Informal specification
  - Captures external behaviour
  - Abstracts away structure and timing
  - Abstracts away the low-level interface

- High-level model
- Register-transfer-level description
  ```c
  bus_write(u32 addr, u32 val) {
      ...
  }
  ```
- netlist
Driver synthesis as controller synthesis

Driver = controller

OS requests = control objective

send() - send a network packet
Driver synthesis as controller synthesis

Driver = controller

OS requests = control objective

send() - send a network packet

Packet has been sent
• Game theory
  – Provides a theoretical framework for verification and synthesis of reactive systems
  – Provides a classification of games
  – Complexity bounds for various types of games
  – Algorithms for finding winning strategies
Example: trivial network adapter

- **off**
  - write(ctl,1)
  - write(ctl,0)

- **on**
  - write(dat,...)

- **bsy**
  - write(ctl,1)

- **done**
  - send

**Transition Legend**

- **Controllable transition**
- **Uncontrollable transition**
Computing the winning set

```
write(ctl,0)
write(ctl,1)
write(dat,...)
write(ctl,1)
send
done
```

INIT

GOAL
Computing the winning set

\[
\begin{align*}
\text{write}(ctl, 0) & \rightarrow \text{write}(ctl, 1) \\
\text{write}(dat, \ldots) & \rightarrow \text{Cpre} \{ \text{done} \} = \{ \text{bsy} \}
\end{align*}
\]

send

\[
\begin{align*}
\text{write}(ctl, 1) & \rightarrow \text{bsy} \\
& \rightarrow \text{done} \\
& \rightarrow \text{GOAL}
\end{align*}
\]
Computing the winning set

\[
C_{\text{pre}}(\{\text{done}, \text{bsy}\}) = \{\text{bsy, on}\}
\]

\[
C_{\text{pre}}(\{\text{done}\}) = \{\text{bsy}\}
\]

write(ctl,0) → off

write(ctl,1) → on

write(dat,...) → bsy

send → done

GOAL

INIT
Computing the winning set

\[
\text{Cpre}\{\text{done,bsy,on}\} = \{\text{off,done,bsy,on}\}
\]

\[
\text{Cpre}\{\text{done,bsy}\} = \{\text{bsy,on}\}
\]

\[
\text{Cpre}\{\text{done}\} = \{\text{bsy}\}
\]

\[
\text{write}(\text{ctl},1)
\]

\[
\text{write}(\text{ctl},0)
\]

\[
\text{write}(\text{dat},...)
\]

\[
\text{send}
\]

\[
\text{GOAL}
\]

\[
\text{INIT}
\]
OS specification

Game objective:
The driver must be in state 0 infinitely often (aka Büchi objective)
Game automaton

- **off 0**
  - write(ctl,1)
  - write(ctl,0) write(ctl,1)
- **on 0**
  - write(ctl,1)
  - write(ctl,0) write(ctl,1)
  - write(dat,...)
- **bsy 0**
  - write(ctl,1)
  - write(ctl,0)
- **done 0**
  - write(ctl,1)
  - xmit
  - xmit_complete
  - send
- **off 1**
  - write(ctl,1)
  - write(ctl,0) write(ctl,1)
  - xmit
  - xmit_complete
- **on 1**
  - write(ctl,1)
  - write(ctl,0) write(ctl,1)
  - write(dat,...)
  - xmit
  - xmit_complete
- **bsy 1**
  - write(ctl,1)
  - write(ctl,0)
  - write(dat,...)
  - xmit
  - xmit_complete
- **done 1**
  - write(ctl,1)
  - xmit
  - xmit_complete
- **off 2**
  - write(ctl,1)
  - write(ctl,0)
  - write(ctl,0)
  - write(ctl,1)
  - xmit_complete
- **on 2**
  - write(ctl,1)
  - write(ctl,0) write(ctl,1)
  - write(dat,...)
  - xmit
  - xmit_complete
- **bsy 2**
  - write(ctl,1)
  - write(ctl,0)
  - write(dat,...)
  - xmit
  - xmit_complete
Winning strategy

off 0 → xmit → off 1

write(ctl,1) → on 1

write(dat,...) → bsy 1

write(ctl,1) → send

Unobservable event

done 0 → xmit → done 1

xmit_complete → done 2
Synthesis with imperfect information

- `write(ctl,1)`
- `stat := 0`
- `write(ctl,0)`
- `write(dat,...)`
- `read(stat) -> 0`
- `read(stat) -> 1`
- `stat := 1`
- `send`
- `stat := 0`
- `stat := 1`
- `done`
Synthesis with imperfect information

```
write(ctl, 0) -> {off}
write(ctl, 1) -> {on}
write(dat, ...) -> {bsy, done}
read(stat) -> 0 -> {bsy, done}
read(stat) -> 1 -> {done}
write(ctl, 1) -> {on}
write(ctl, 0) -> {off}
```
Strategy with imperfect information

write(ctl,1)
write(dat,...)
write(ctl,1)
xmit
read(stat) -> 0
write(ctl,1)
read(stat) -> 1
xmit

xmit_complete
Challenges

- State explosion ($2^{320}$ states in the IDE controller device)
  - Classical game theory algorithms do not scale
  - Predicate abstraction ($2^{48}$ states after abstraction)
  - Symbolic state space representation (BDD size: 3226 nodes)
Challenges

• Synthesis with imperfect information
  – Generalisation of a perfect information strategy
• Efficient C code generation
  – Control flow graph minimisation
• Support for DMA
  – wip
Termite toolkit

- DML Compiler
- Statechart compiler
- Other frontends
- Device.dml
- OS.sc

Arrows indicate flow:
- Device.tsl
- Abstract.tsl
- Game automaton
- Counterexample strategy
- Driver strategy

- Debugger
- Code Generator
- TSL Compiler
- Synthesis
- Code Generator

NICTA Copyright 2012 From imagination to impact
Key Research Outcomes (to date)

Successfully synthesised drivers:

- IDE disk controller
- W5100 Eth shield
- Asix AX88772 USB-to-Eth adapter
- SD host controller
Synthesis of provably correct drivers

- Synthesised => Correct ??
  - Bugs in the tool
  - Bugs in the input spec
- Dealing with bugs in the synthesis tool
  - Generate proof of correctness along with the driver
- Dealing with buggy specs
Bridging the gap between hardware and driver synthesis

Transaction-level model -> synthesis

manual refinement

Register-transfer-level description -> synthesis

driver.c
Bridging the gap between hardware and driver synthesis

High-level model (Bluespec/Catapult/Synphony)

synthesis

driver.c

synthesis
Conclusions

• Automatic device-driver synthesis:
  – correct-by-construction device drivers at a fraction of the cost of manual development
  – practical alternative to traditional driver development