School of Computer Science & Engineering

**Trustworthy Systems Group**

# High-Fidelity Specification of Real-World Devices

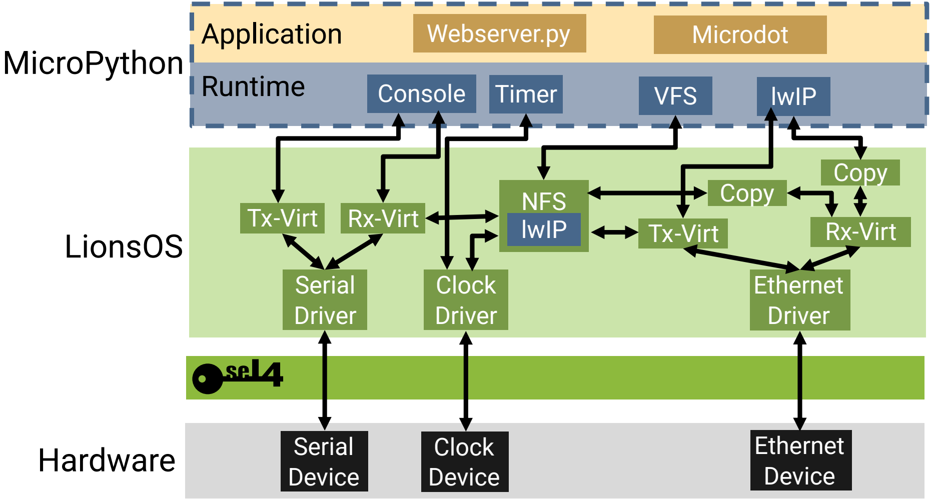
Liam Murphy [1] Albert Rizaldi [2] Lesley Rossouw [1]
Chen George [3] James Treloar [1] Hammond Pearce [1]
Miki Tanaka [1] Gernot Heiser [1]

[1]UNSW Sydney [2]PlanV GmbH [3]University of Wisconsin - Madison

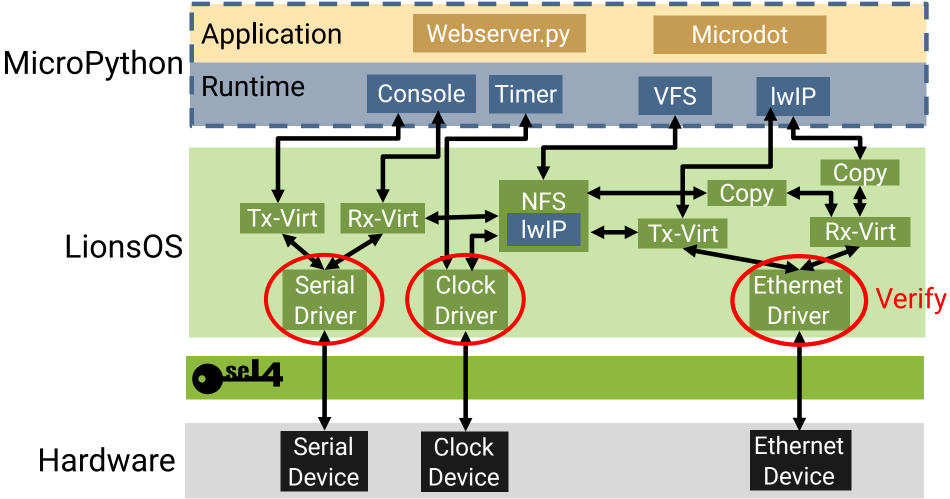
**PLOS ’25**

# Background

# Who Are We?


MicroPython
Application
Webserver.py
Microdot
Runtime
Console
Timer
VFS
lwIP
LionsOS
Tx-Virt
Rx-Virt
NFS
lwIP
Copy
Copy
Tx-Virt
Rx-Virt
Serial Driver
Clock Driver
Ethernet Driver
sel4
Hardware
Serial Device
Clock Device
Ethernet Device

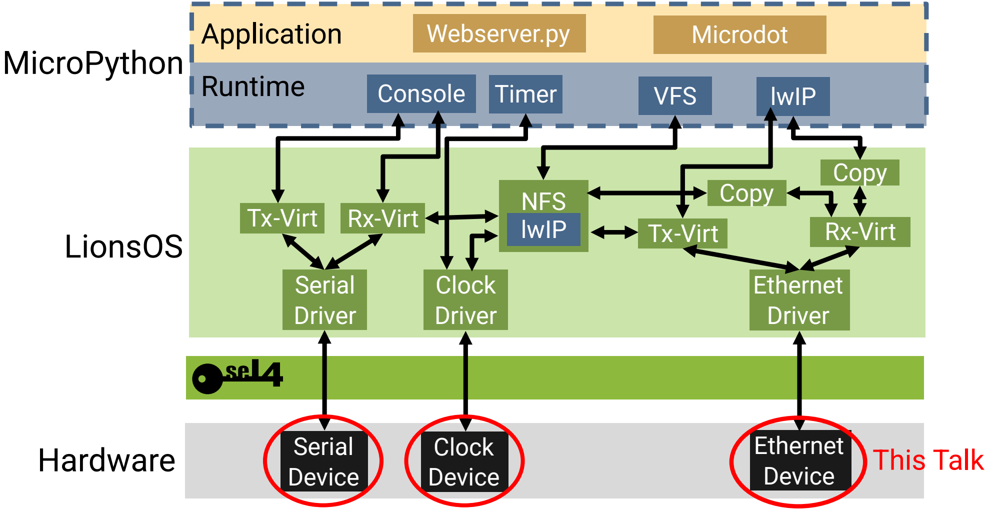
# Who Are We?



MicroPython
Application
Webserver.py
Microdot
Runtime
Console
Timer
VFS
lwIP
LionsOS
Tx-Virt
Rx-Virt
NFS
lwIP
Copy
Copy
Tx-Virt
Rx-Virt
Serial Driver
Clock Driver
Ethernet Driver
Verify
sel4
Hardware
Serial Device
Clock Device
Ethernet Device

# Who Are We?



MicroPython
Application
Webserver.py
Microdot
Runtime
Console
Timer
VFS
lwIP
LionsOS
Copy
Copy
NFS
lwIP
Tx-Virt
Rx-Virt
Tx-Virt
Rx-Virt
Serial Driver
Clock Driver
Ethernet Driver
seL4
Hardware
Serial Device
Clock Device
Ethernet Device
This Talk

# Why Drivers?

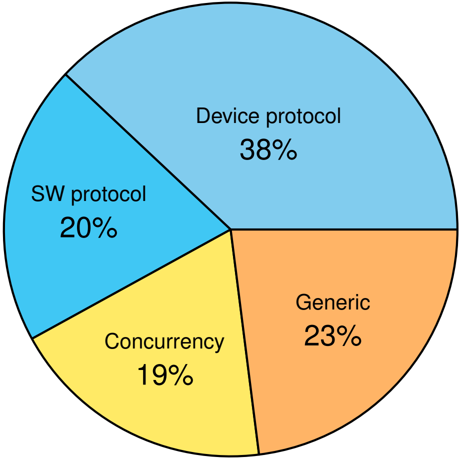
- Device driver bugs are a major source of OS vulnerabilities

# Why Drivers?

- Device driver bugs are a major source of OS vulnerabilities
  - Cause of majority of Linux CVEs from 2018-2022 [Pohjola et al. 2023]

# Why Drivers?

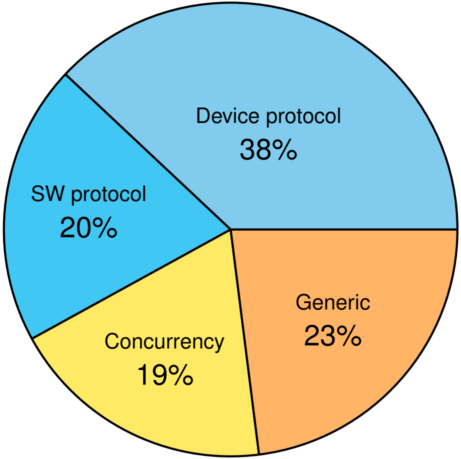
- Device driver bugs are a major source of OS vulnerabilities
  - Cause of majority of Linux CVEs from 2018-2022 [Pohjola et al. 2023]
- Dominant cause of Linux driver bugs is device-protocol violations [Ryzhyk et al. 2009]




*Causes of Linux driver bugs (2002–2008)*

# Why Drivers?

- Device driver bugs are a major source of OS vulnerabilities
  - Cause of majority of Linux CVEs from 2018-2022 [Pohjola et al. 2023]
- Dominant cause of Linux driver bugs is device-protocol violations [Ryzhyk et al. 2009]
  - Without knowledge of the device protocol, formal verification cannot prevent these




*Causes of Linux driver bugs (2002–2008)*

# Aim

- Prevent device-protocol bugs by:

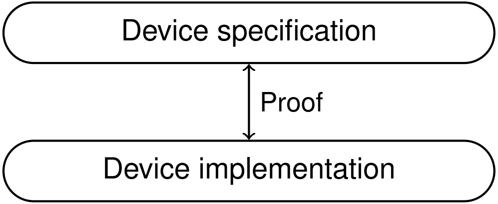
# Aim

- Prevent device-protocol bugs by:
    - Formally specifying device interfaces

Device specification

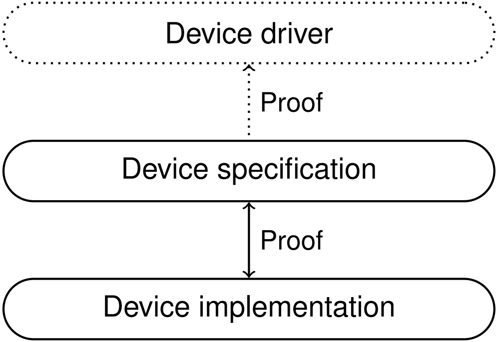
# Aim

- Prevent device-protocol bugs by:
  - Formally specifying device interfaces
  - Verifying these specifications against the device implementation



# Aim

- Prevent device-protocol bugs by:
  - Formally specifying device interfaces
  - Verifying these specifications against the device implementation
- Future work: driver verification

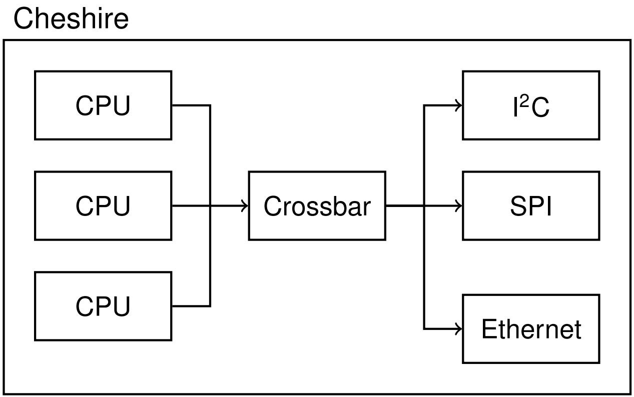


# Overview

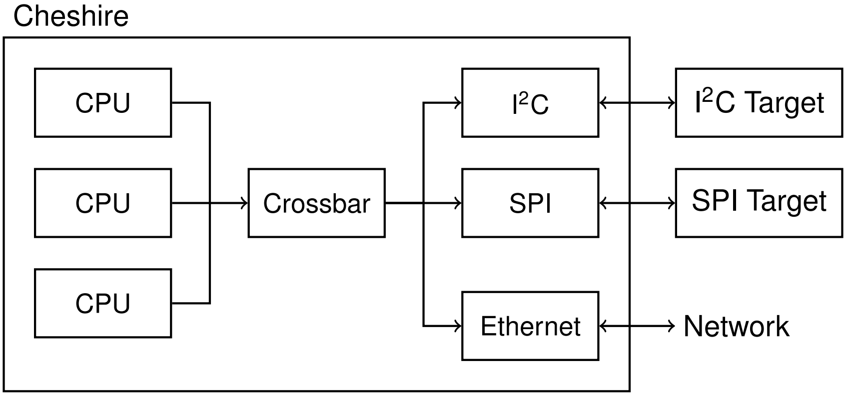
- Target hardware
- Verification approach
- Specification format
- Application
- Status

# Target Hardware

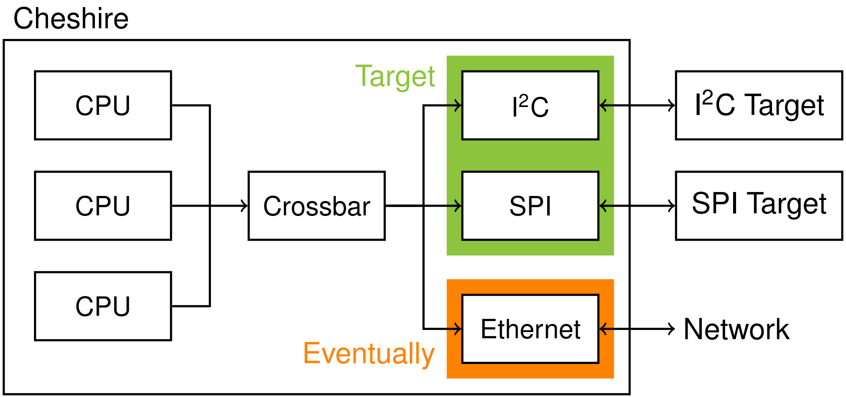
# Target Hardware



# Target Hardware



# Target Hardware



# Target Hardware

| | Verilog | Driver | HOL spec |
|---|---|---|---|
| **I$^2$C** | 5993 | 713 | 1414 |
| **SPI** | 4609 | 864 | 1235 |
| **Ethernet** | 3987 | 641 | N/A |

# Verification Approach

# Tools

- HOL4 theorem prover

# Tools

- HOL4 theorem prover
- HOL4 Verilog formalisation [Lööw and Myreen 2019]:

# Tools

- HOL4 theorem prover
- HOL4 Verilog formalisation [Lööw and Myreen 2019]:
  - Semantics for Verilog AST

Verilog AST

# Tools

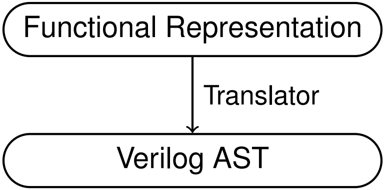
- HOL4 theorem prover
- HOL4 Verilog formalisation [Lööw and Myreen 2019]:
  - Semantics for Verilog AST
  - Semantics for functional representation

Functional Representation

Verilog AST

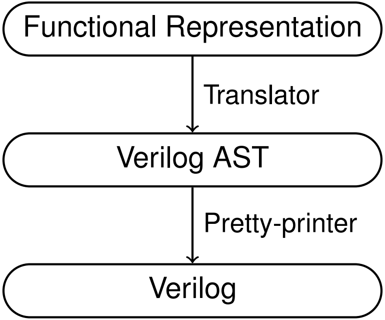
# Tools

- HOL4 theorem prover
- HOL4 Verilog formalisation [Lööw and Myreen 2019]:
  - Semantics for Verilog AST
  - Semantics for functional representation
  - Proof-producing translator from functional to AST



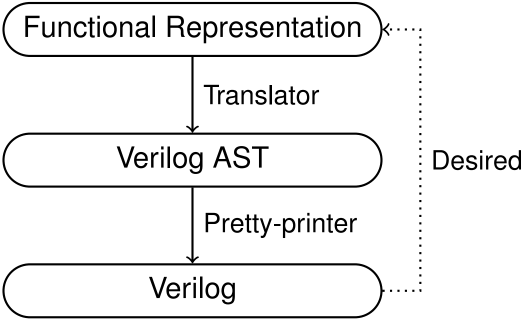
# Tools

- HOL4 theorem prover
- HOL4 Verilog formalisation [Lööw and Myreen 2019]:
    - Semantics for Verilog AST
    - Semantics for functional representation
    - Proof-producing translator from functional to AST
    - Pretty-printer for deep embedding



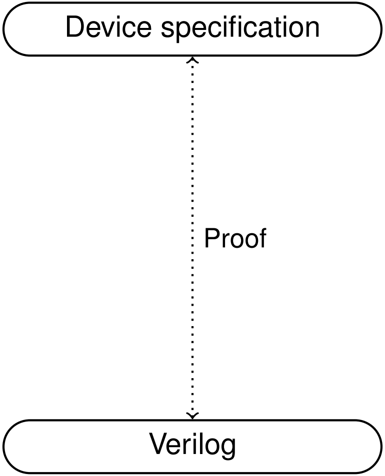
# Tools

- HOL4 theorem prover
- HOL4 Verilog formalisation [Lööw and Myreen 2019]:
  - Semantics for Verilog AST
  - Semantics for functional representation
  - Proof-producing translator from functional to AST
  - Pretty-printer for deep embedding
- Problem: need opposite direction



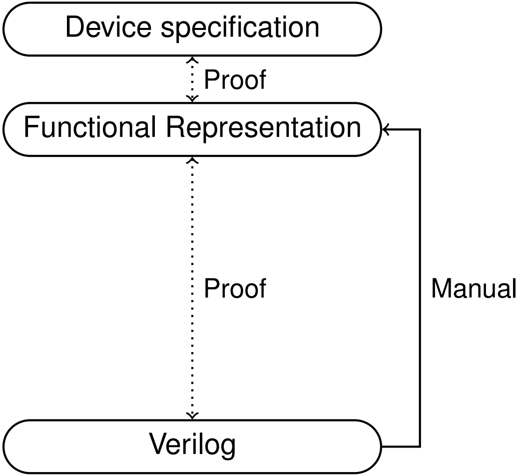
# Workflow

- Workaround:



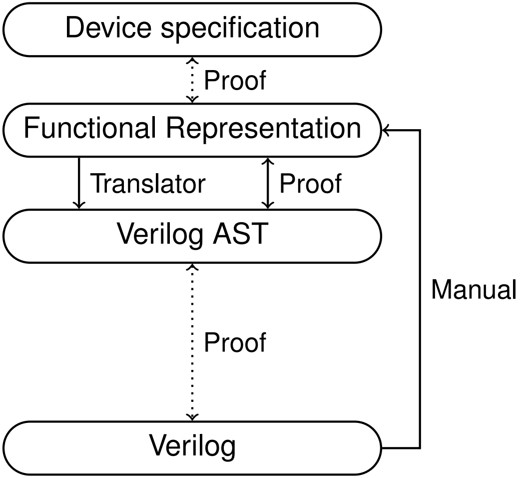
# Workflow

- Workaround:
  - Manually translate Verilog to functional representation



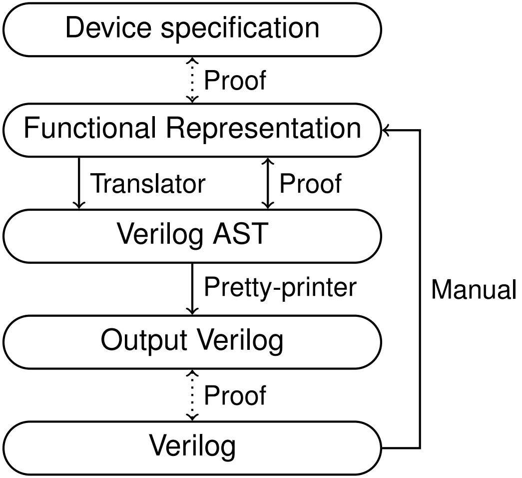
# Workflow

- Workaround:
  - Manually translate Verilog to functional representation
  - Translate and export into Verilog



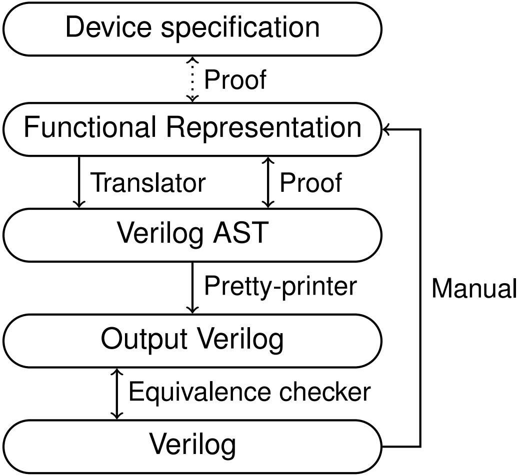
# Workflow

- Workaround:
  - Manually translate Verilog to functional representation
  - Translate and export into Verilog



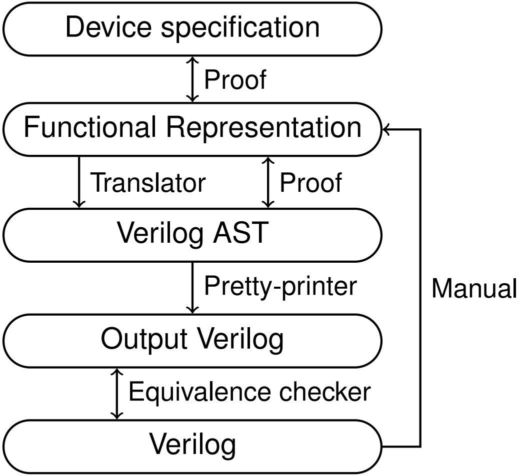
# Workflow

- Workaround:
  - Manually translate Verilog to functional representation
  - Translate and export into Verilog
  - Use equivalence-checker (`eqy`) to prove equivalence to original Verilog



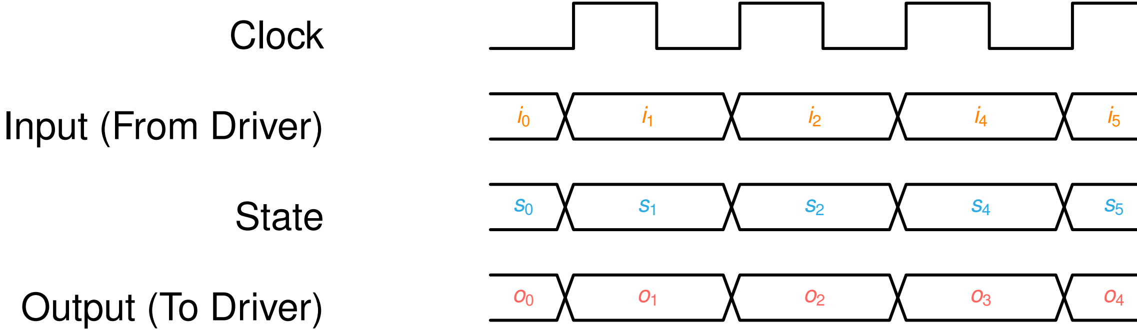
# Workflow

- Workaround:
  - Manually translate Verilog to functional representation
  - Translate and export into Verilog
  - Use equivalence-checker (`eqy`) to prove equivalence to original Verilog
- Finally, prove that functional representation refines specification

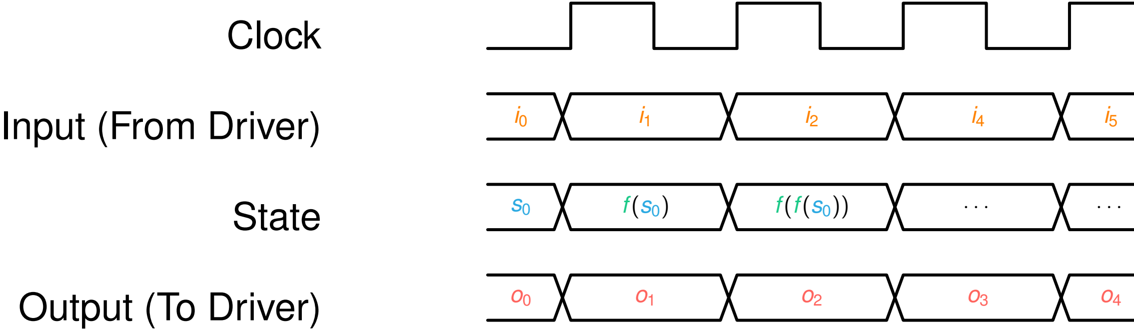


# Specification Format

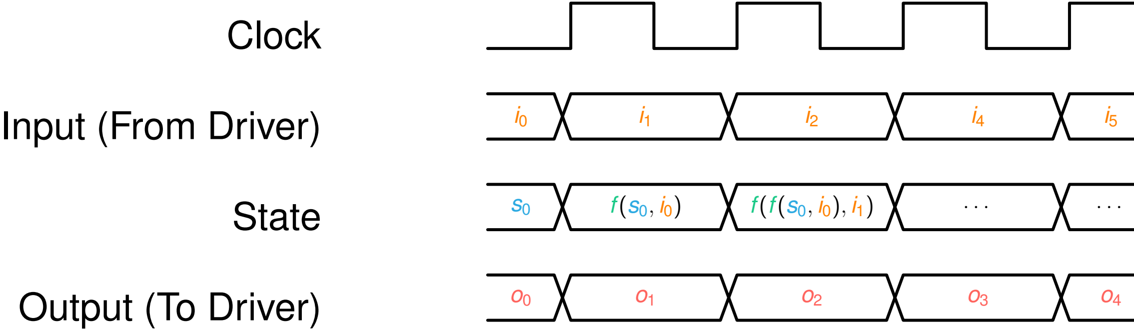
# Specification Format



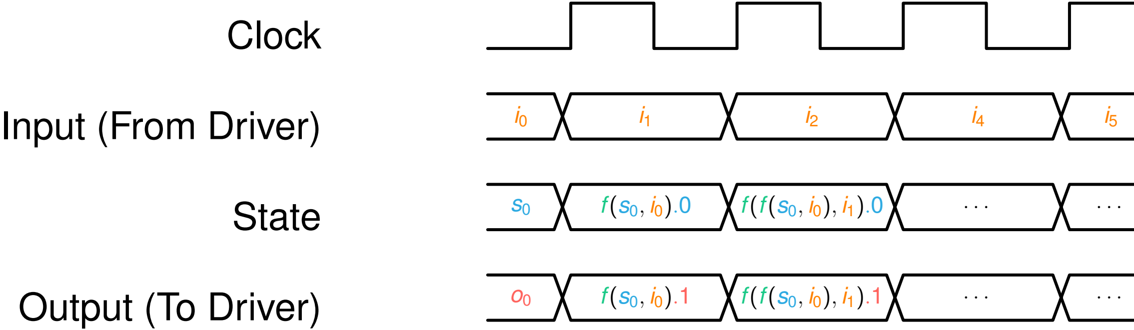
# Specification Format



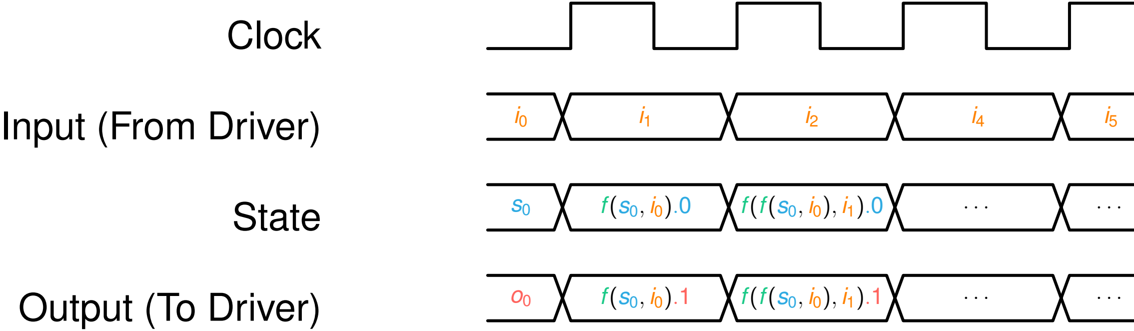
# Specification Format



# Specification Format



# Specification Format




$$f : s \to i \to s \times o$$

# Non-Determinism

- Model can be non-deterministic

# Non-Determinism

- Model can be non-deterministic
  - Inputs not controlled by driver

# Non-Determinism

- Model can be non-deterministic
  - Inputs not controlled by driver
  - Underspecification

# Non-Determinism

- Model can be non-deterministic
  - Inputs not controlled by driver
  - Underspecification
- $f$ only returns one possible state

# Non-Determinism

- Model can be non-deterministic
  - Inputs not controlled by driver
  - Underspecification
- $f$ only returns one possible state
- Determined by `fnums` - infinite stream of numbers

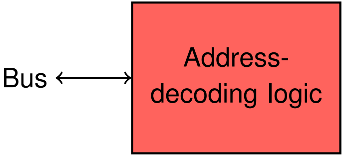
# Non-Determinism

- Model can be non-deterministic
  - Inputs not controlled by driver
  - Underspecification
- *f* only returns one possible state
- Determined by `fnums` - infinite stream of numbers
- To obtain set of all possible states, try all `fnums`

# Application to Cheshire

# Structure of Cheshire Peripherals

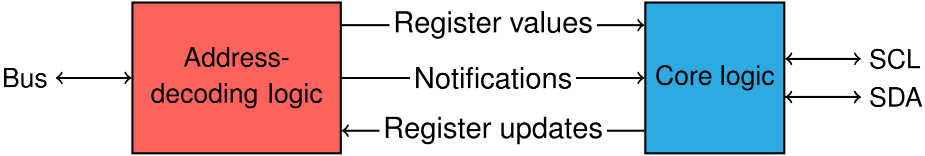
# Structure of Cheshire Peripherals



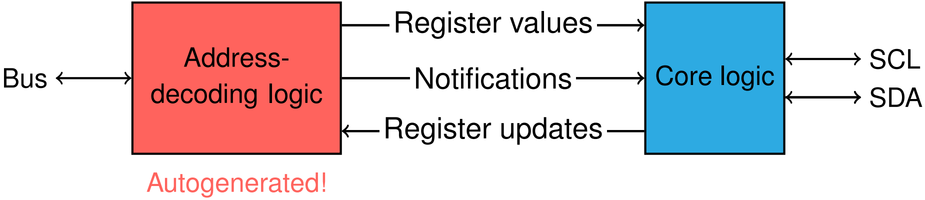

 

# Structure of Cheshire Peripherals

# Structure of Cheshire Peripherals



# Structure of Cheshire Peripherals



# Cheshire Specifications

- Format of Cheshire peripheral specifications:

```
cheshire_run tick read write:
  state -> req option list ->
  ffi_outcome + state # word32 option list
```

# Cheshire Specifications

- Format of Cheshire peripheral specifications:

```
cheshire_run tick read write:
  state -> req option list ->
  ffi_outcome + state # word32 option list
```

- Input instantiated with optional MMIO request

# Cheshire Specifications

- Format of Cheshire peripheral specifications:

```
cheshire_run tick read write:
  state -> req option list ->
  ffi_outcome + state # word32 option list
```

- Input instantiated with optional MMIO request
- Output instantiated with response to read requests

# Cheshire Specifications

- Format of Cheshire peripheral specifications:

```
cheshire_run tick read write:
  state -> req option list ->
  ffi_outcome + state # word32 option list
```

- Input instantiated with optional MMIO request
- Output instantiated with response to read requests
- Executes multiple cycles

# Cheshire Specifications

```
cheshire_run tick read write:
  state -> req option list ->
  ffi_outcome + state # word32 option list
```

- `tick` represents core logic

# Cheshire Specifications

```
cheshire_run tick read write:
  state -> req option list ->
  ffi_outcome + state # word32 option list
```

- `tick` represents core logic
- `read`, `write` and `cheshire_run` represent address-decoding logic

# Status

# Status

| | $I^2C$ | | SPI | |
|---|---|---|---|---|
| | Address-decoding | Core | Address-decoding | Core |
| Specification | ✓ | ✓ | ✓ | ✓ |
| Functional Representation | ✓ | ✓ | ✓ | WIP |
| Equivalence-checking | ✓ | WIP | ✓ | WIP |
| Correctness proof | ✓ | Almost! | ✓ | WIP |

# Questions?


**Note**

UNSW is recruiting OS faculty members - talk to Gernot if you're interested

# References


- Pohjola, Johannes Åman et al. (Oct. 2023). "Pancake: Verified Systems Programming Made Sweeter". In: *Workshop on Programming Languages and Operating Systems (PLOS)*. Koblenz, DE.
- Ryzhyk, Leonid et al. (Apr. 2009). "Dingo: Taming Device Drivers". In: *EuroSys Conference*. Nuremberg, DE, pp. 275–288.
- Lööw, Andreas and Magnus O. Myreen (2019). "A proof-producing translator for Verilog development in HOL". In: *Proceedings of the International Workshop on Formal Methods in Software Engineering (FormaliSE@ICSE)*, pp. 99–108.