



Challenges in Automatic Device Driver Synthesis

Leonid Ryzhyk

Adam Walker

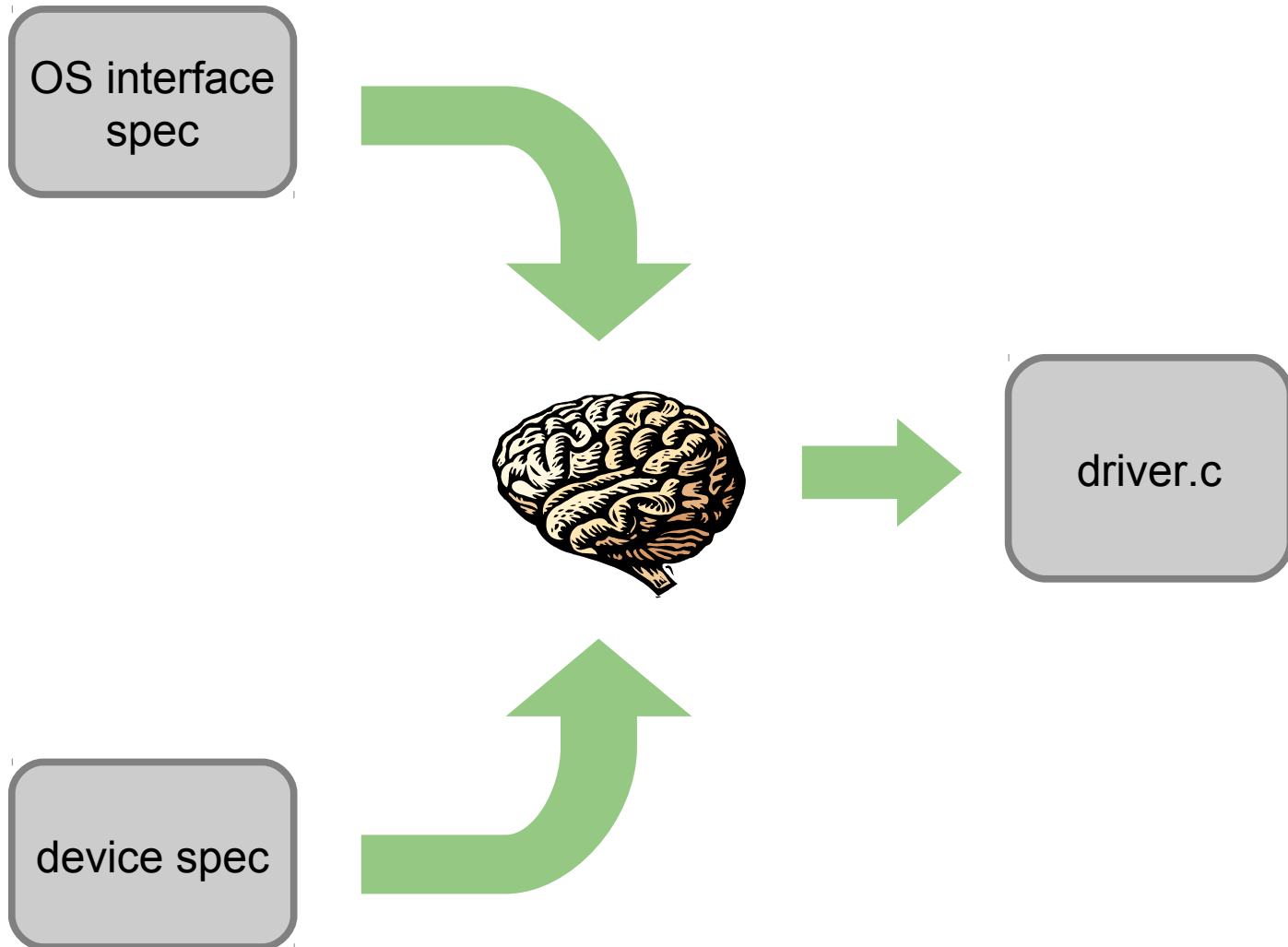


Australian Government
Department of Broadband, Communications
and the Digital Economy
Australian Research Council

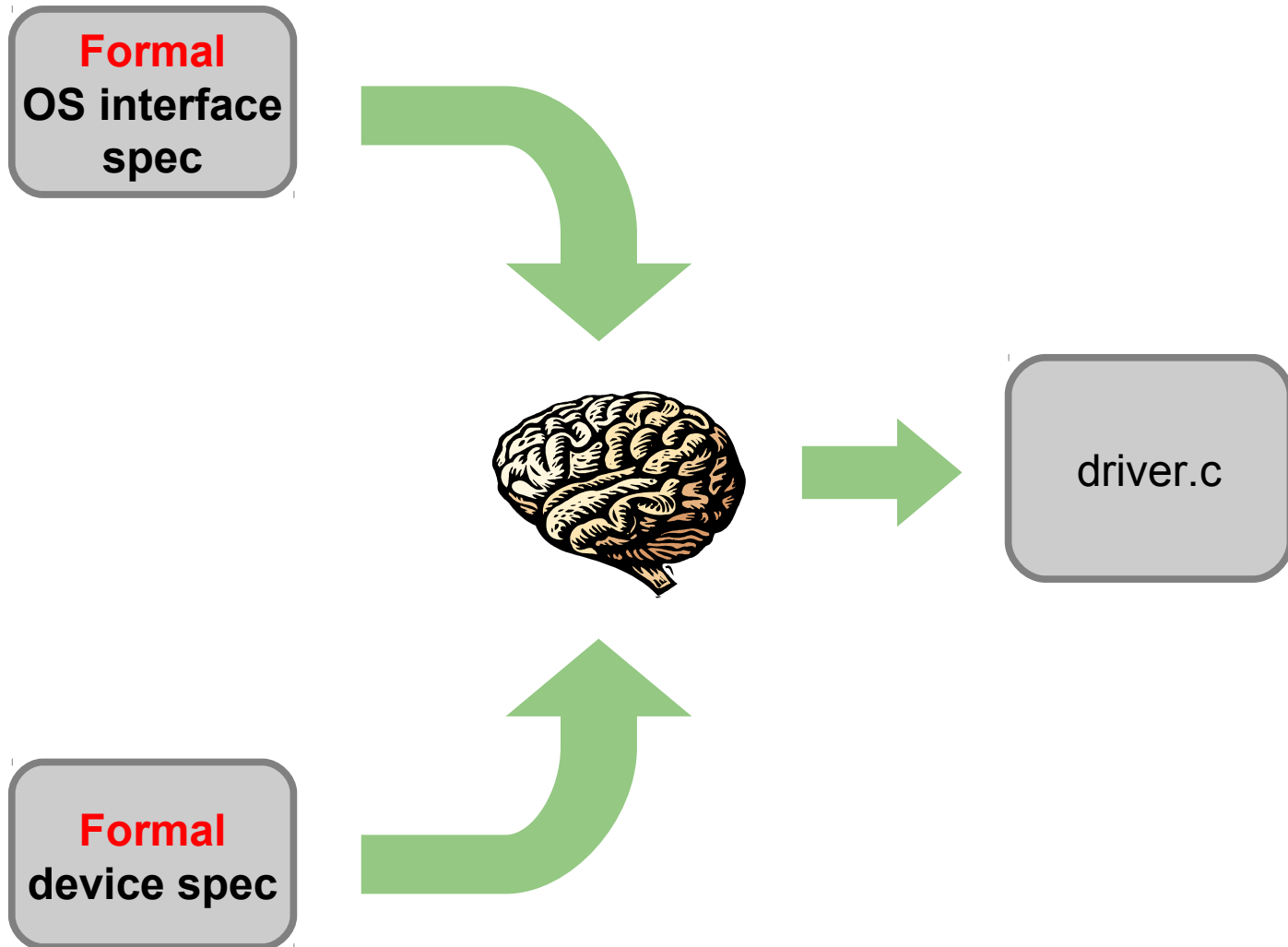
NICTA Funding and Supporting Members and Partners



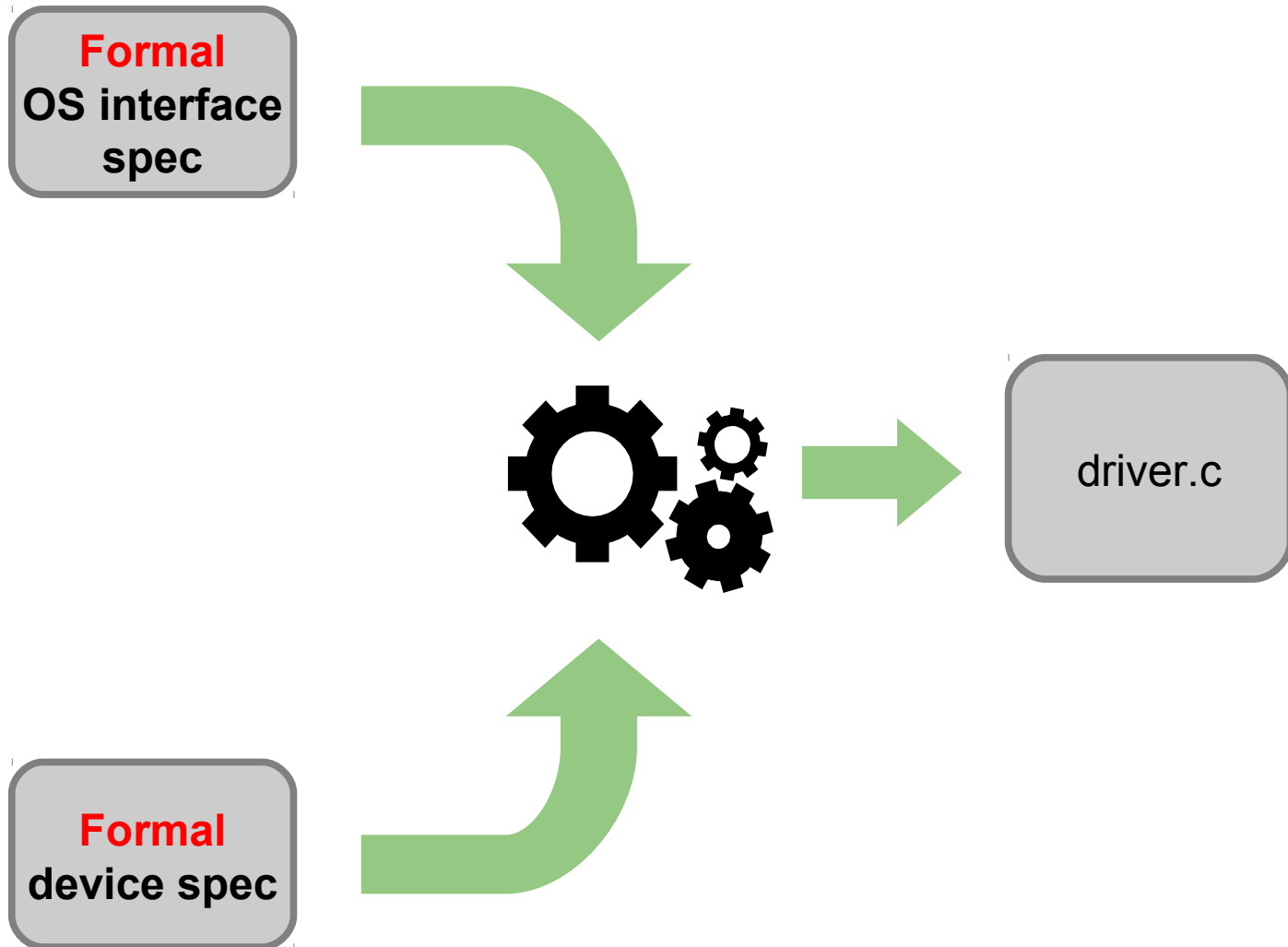
Driver synthesis



Driver synthesis



Driver synthesis

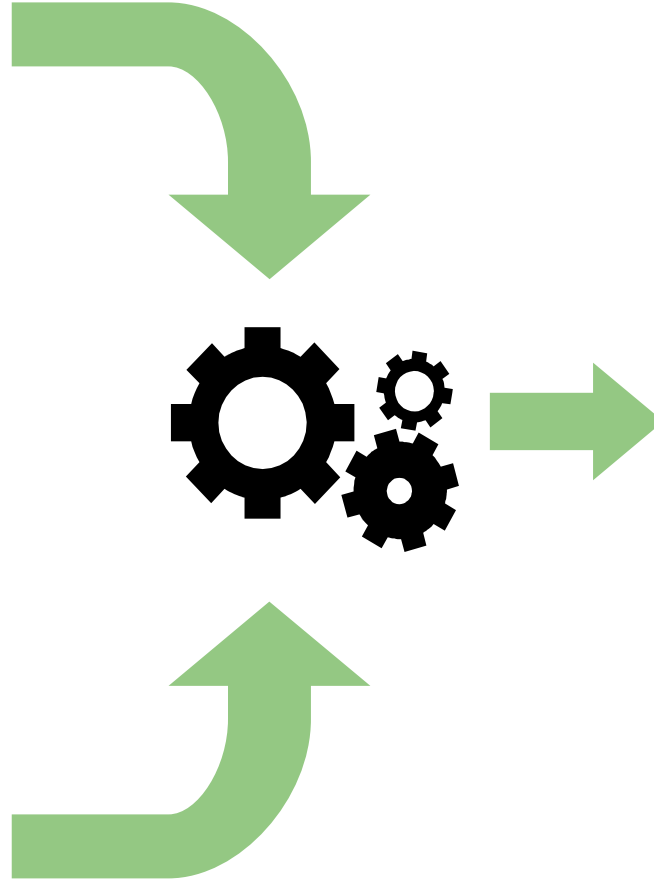


Driver synthesis

Formal
OS interface
spec

Developed once for a
class of devices

Formal
device spec



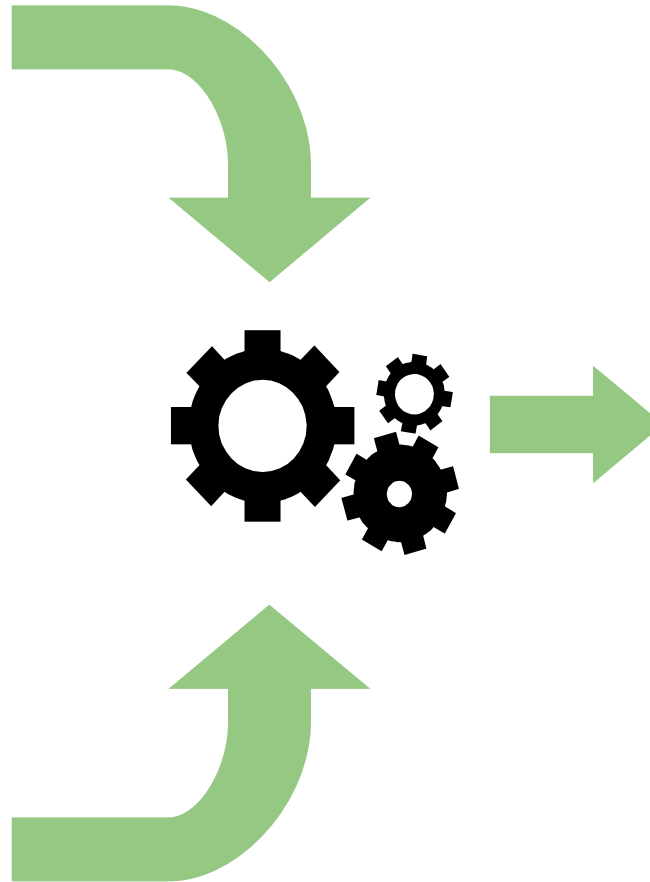
Driver synthesis

Formal
OS interface
spec

Developed once for a
class of devices

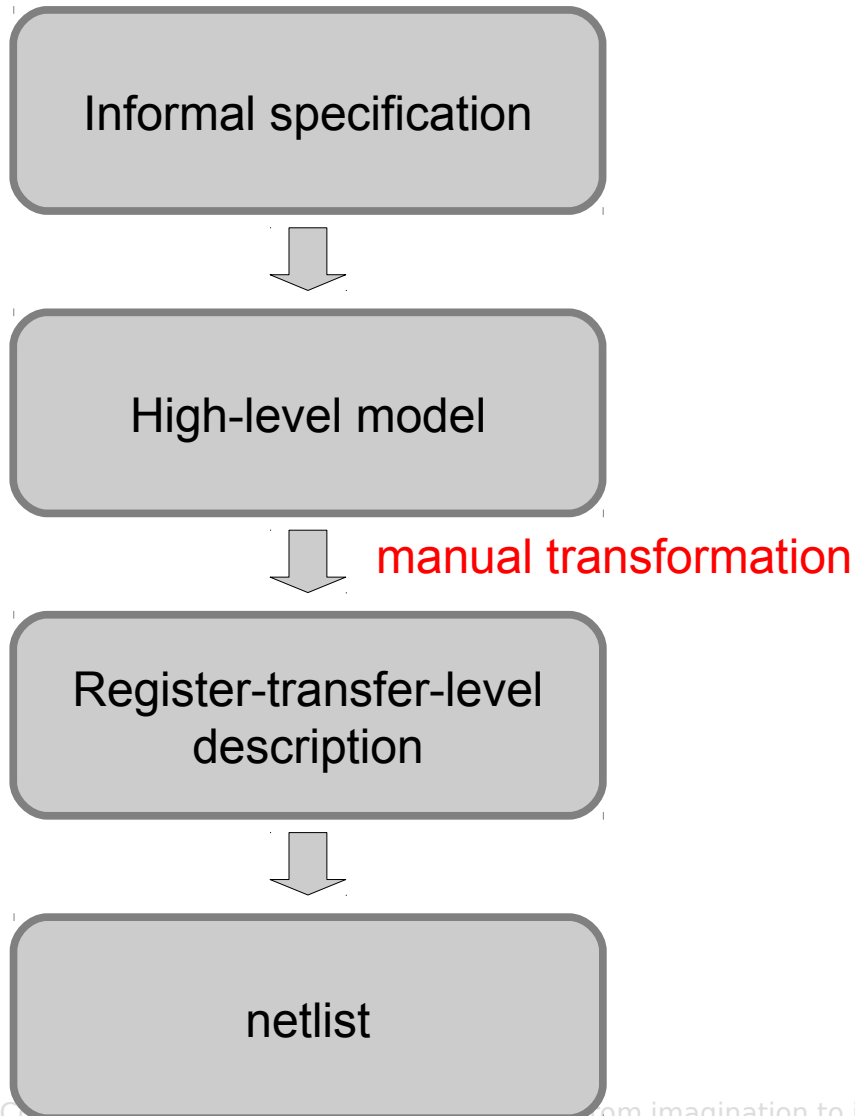
Use existing hardware
specs

Formal
device spec

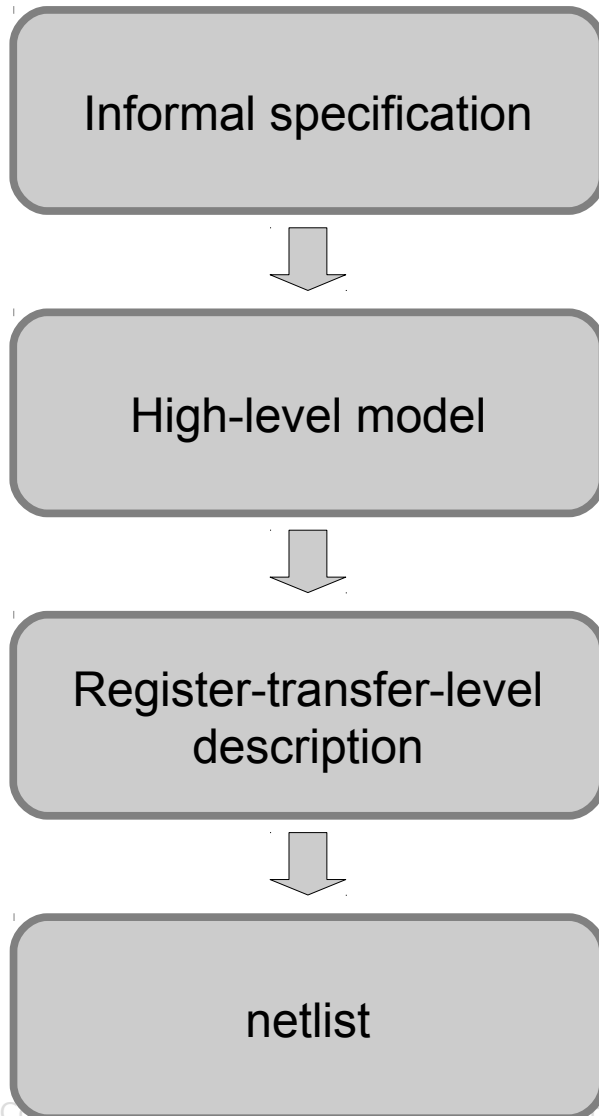


driver.c

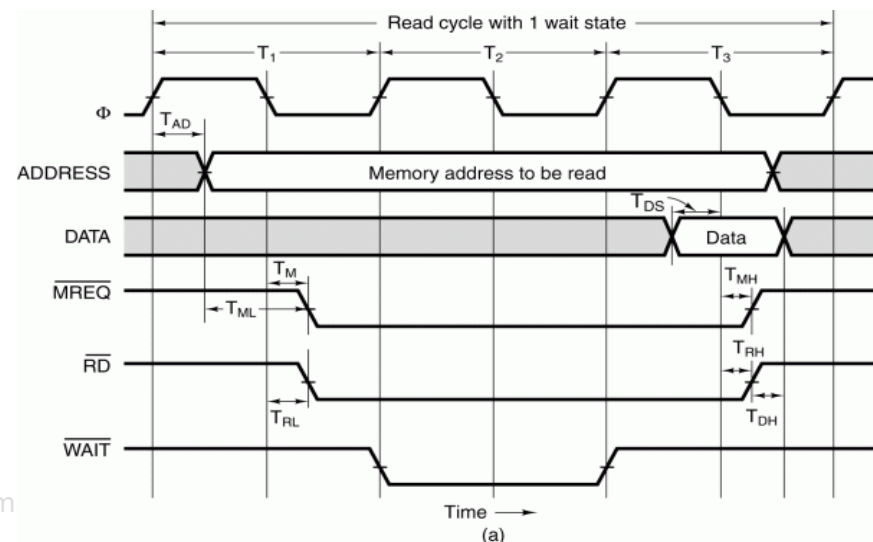
Hardware design workflow



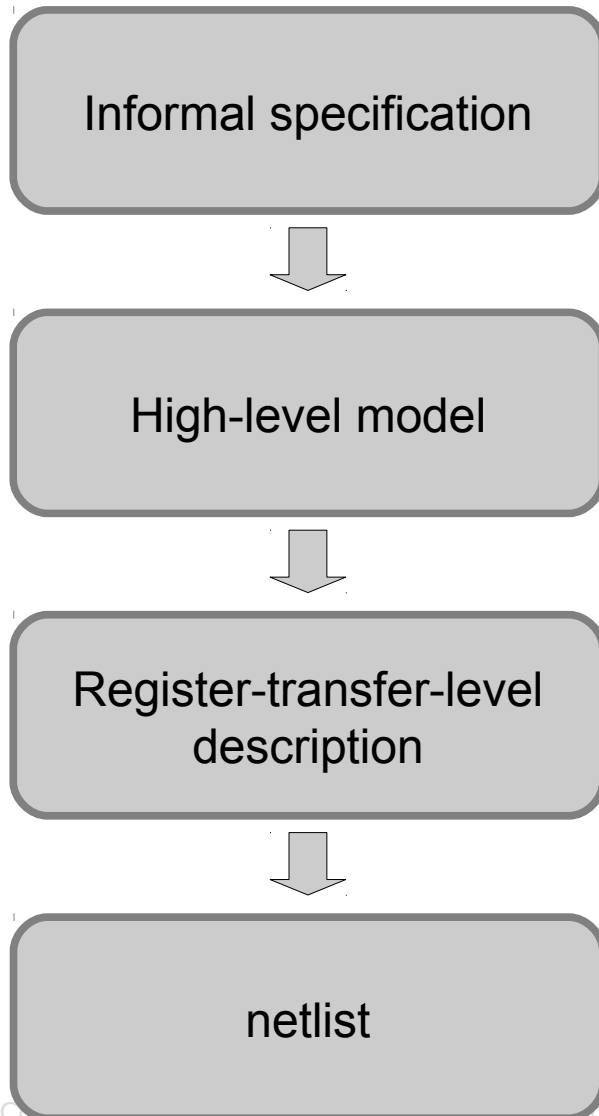
Hardware design workflow



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



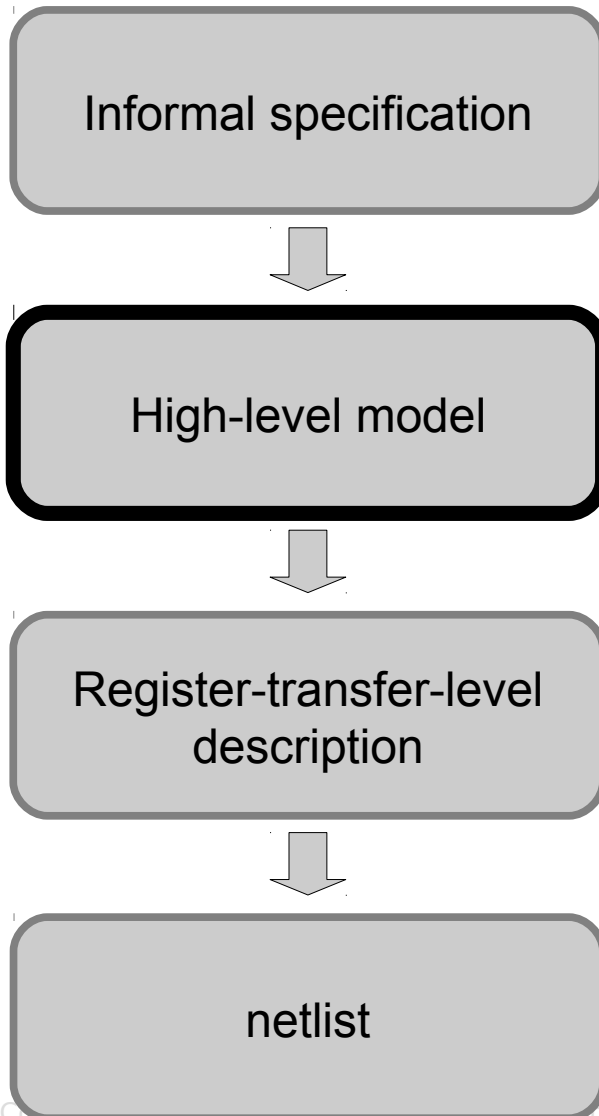
Hardware design workflow



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

```
bus_write(u32 addr, u32 val)
{
    ...
}
```

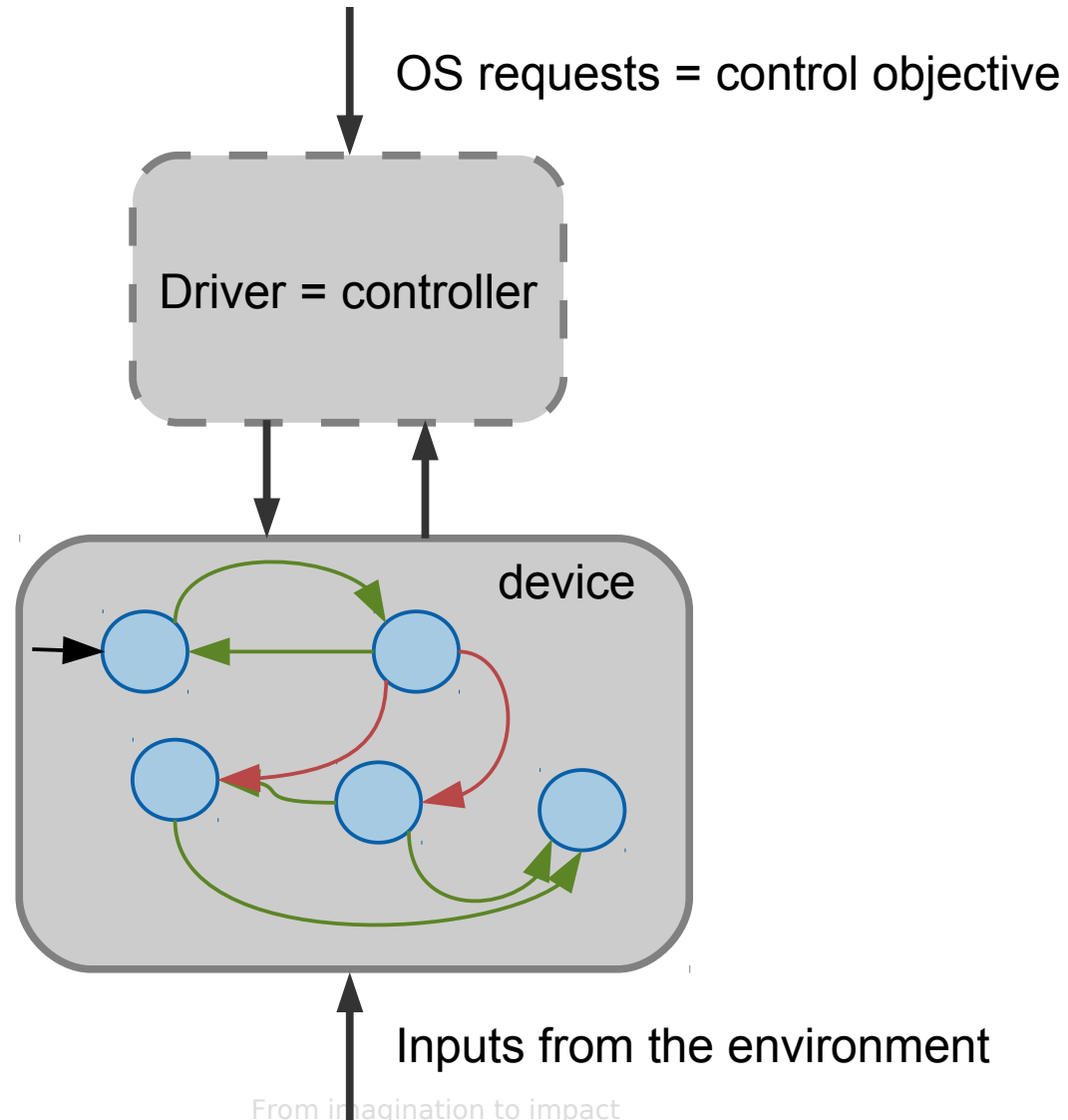
Hardware design workflow



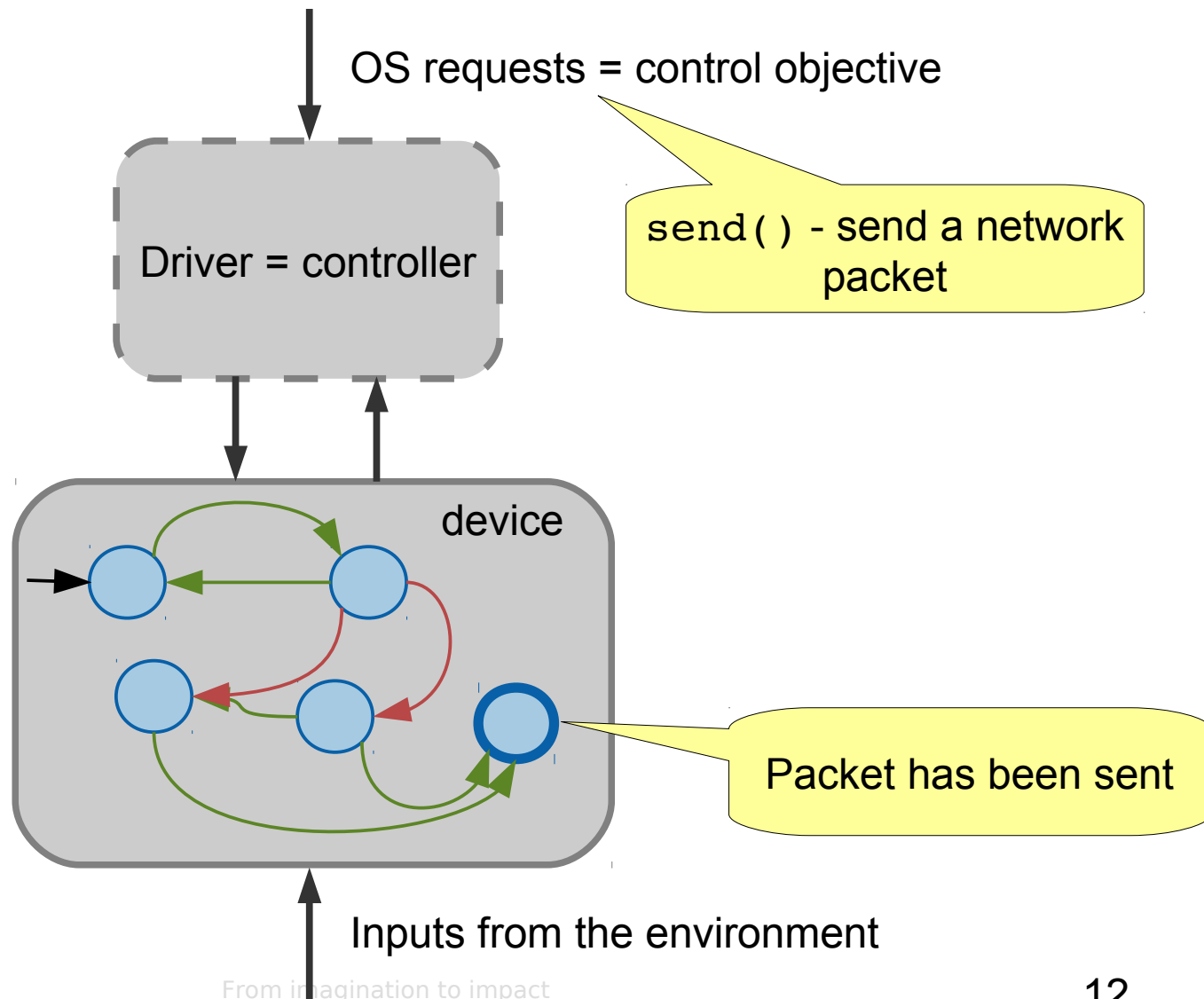
- Captures external behaviour
- Abstracts away structure and timing
- Abstracts away the low-level interface

```
bus_write(u32 addr, u32 val)
{
    ...
}
```

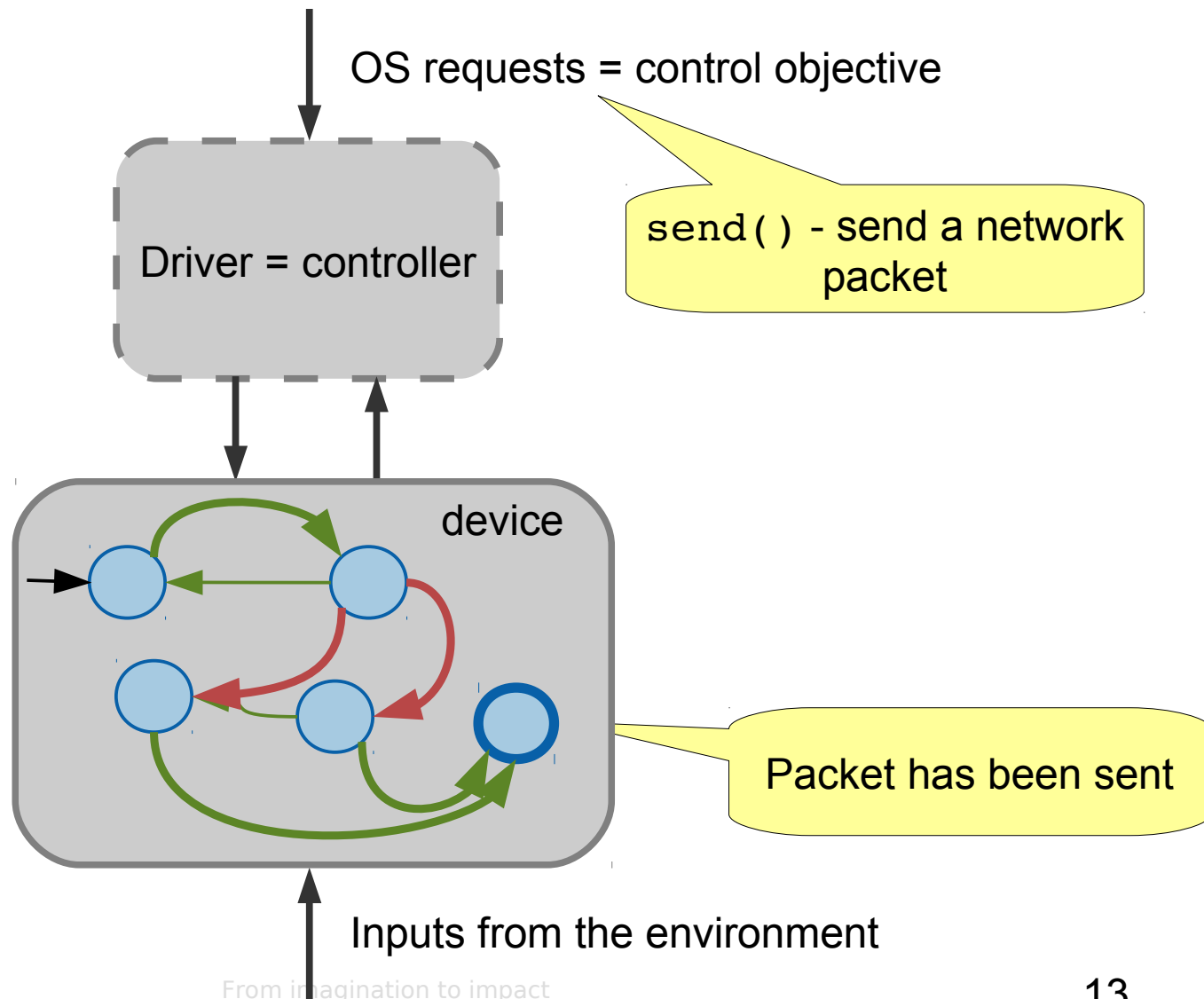
Driver synthesis as controller synthesis



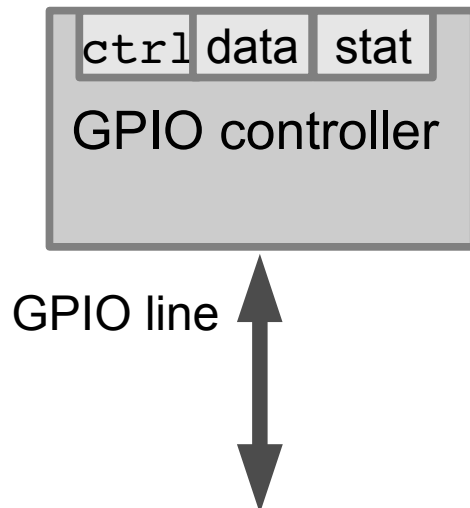
Driver synthesis as controller synthesis



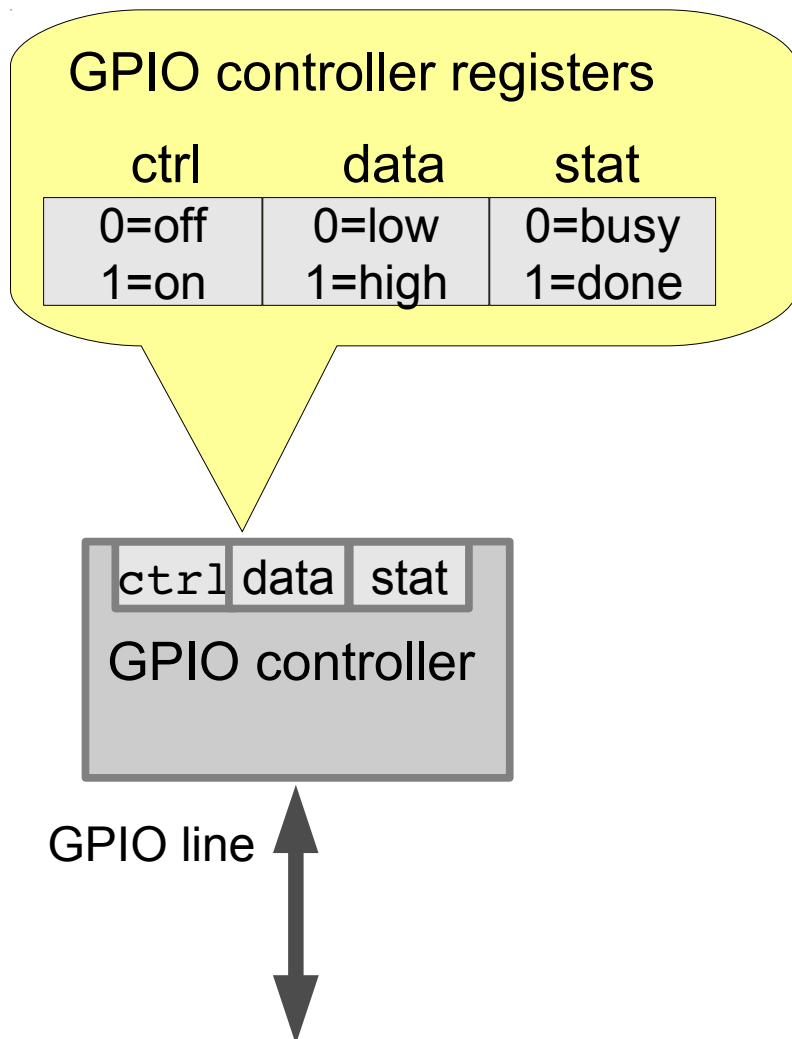
Driver synthesis as controller synthesis



Example: GPIO controller



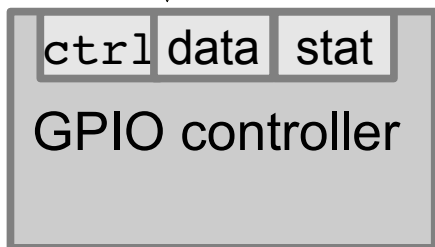
Example: GPIO controller



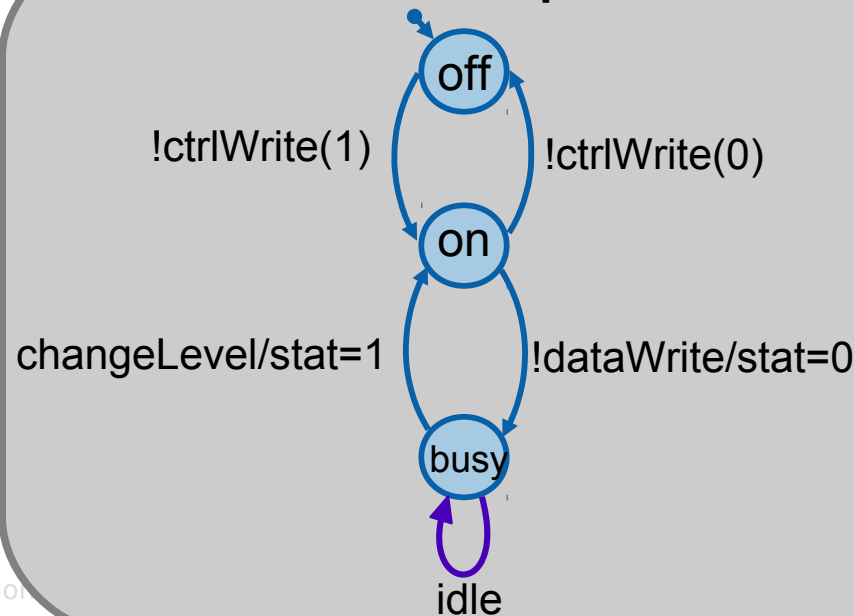
Example: GPIO controller

GPIO controller registers

ctrl	data	stat
0=off	0=low	0=busy
1=on	1=high	1=done



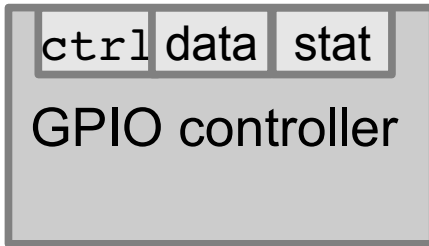
Device spec



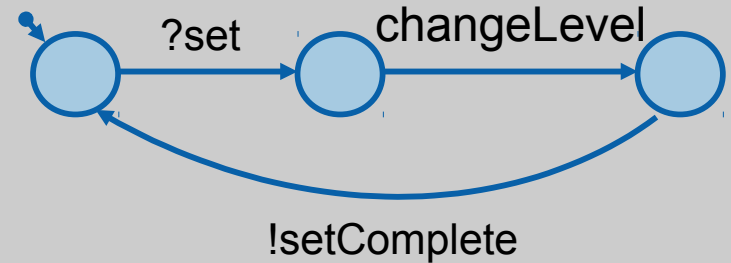
Example: GPIO controller

GPIO controller registers

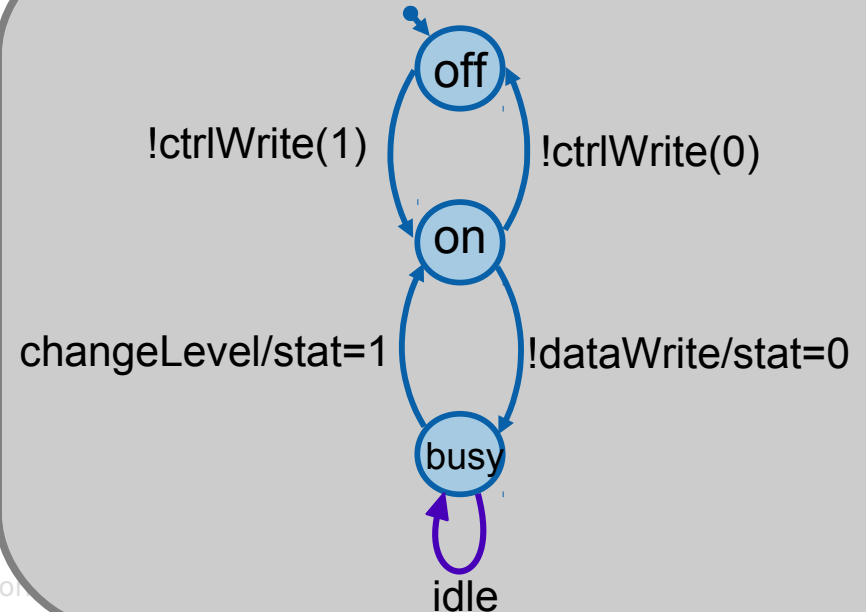
ctrl	data	stat
0=off	0=low	0=busy
1=on	1=high	1=done



OS interface spec



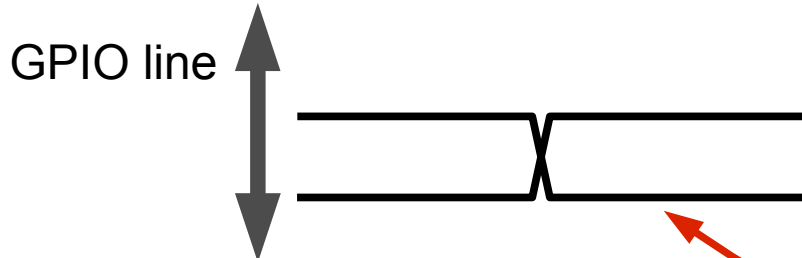
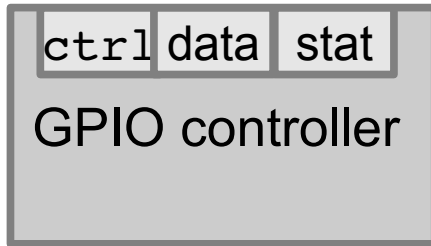
Device spec



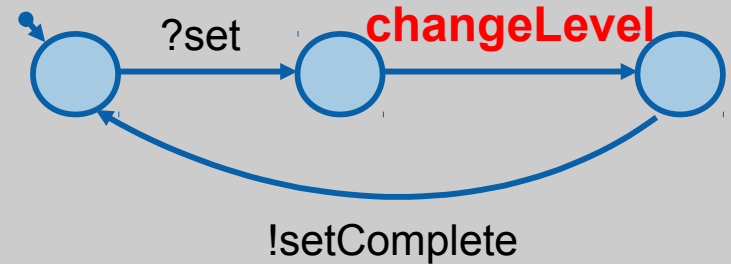
Example: GPIO controller

GPIO controller registers

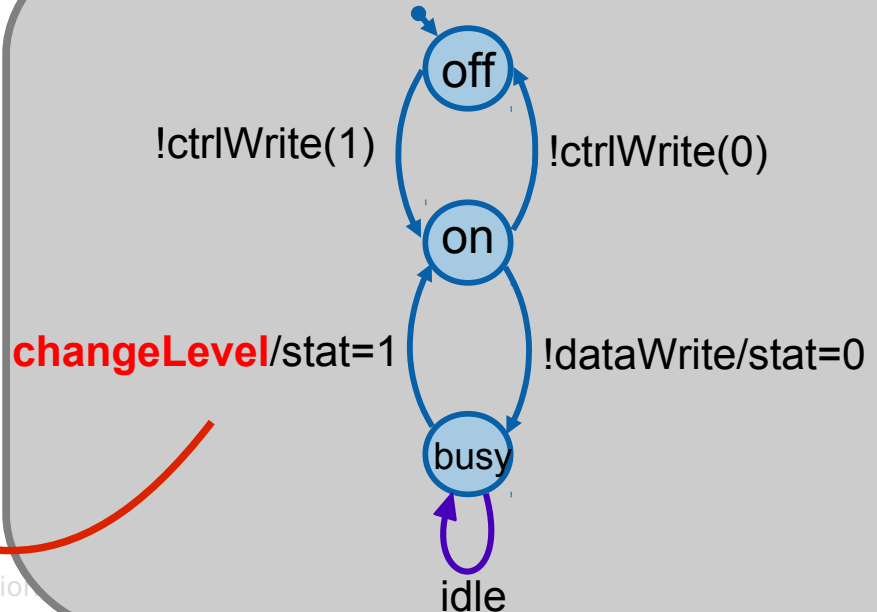
ctrl	data	stat
0=off	0=low	0=busy
1=on	1=high	1=done



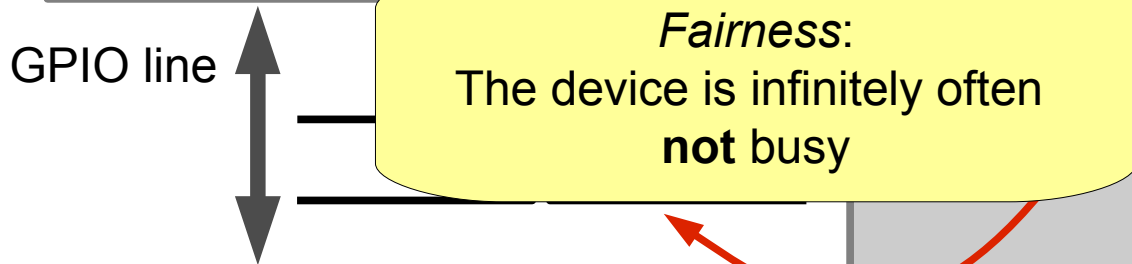
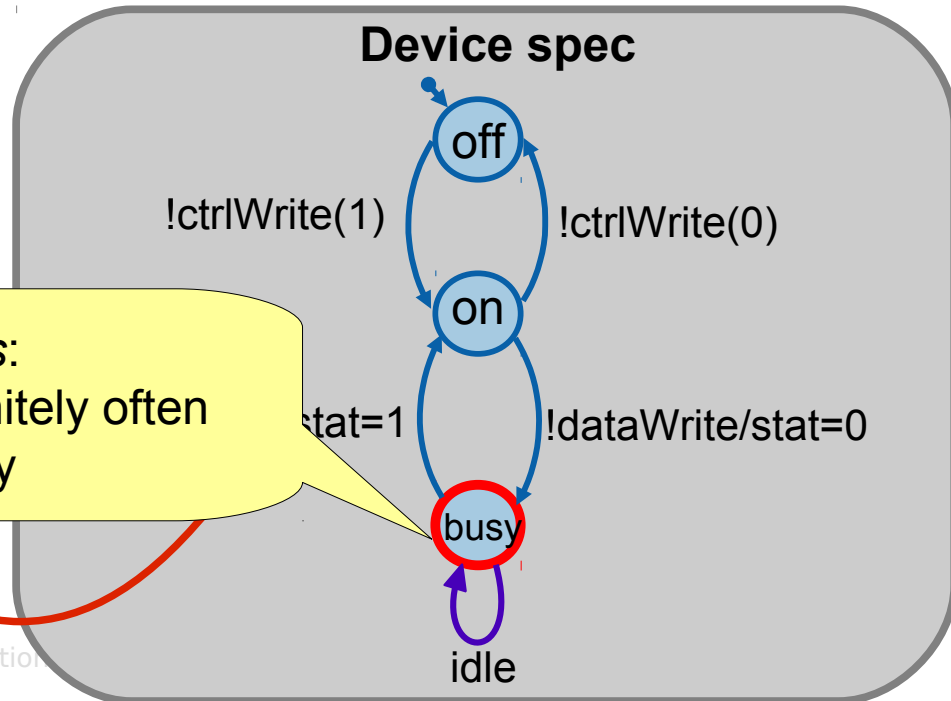
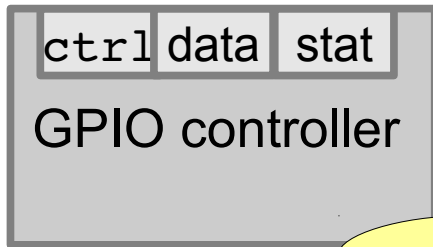
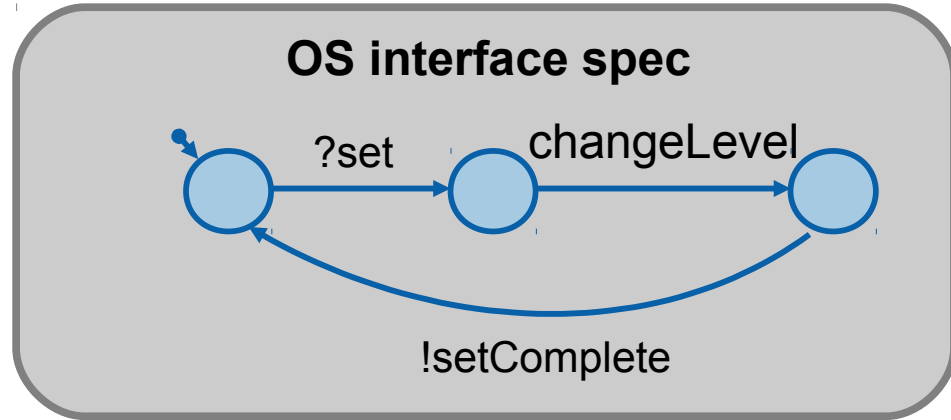
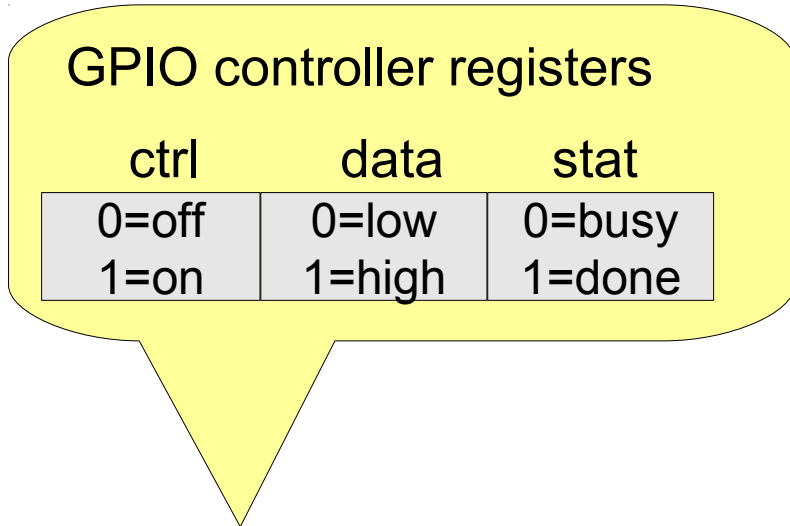
OS interface spec



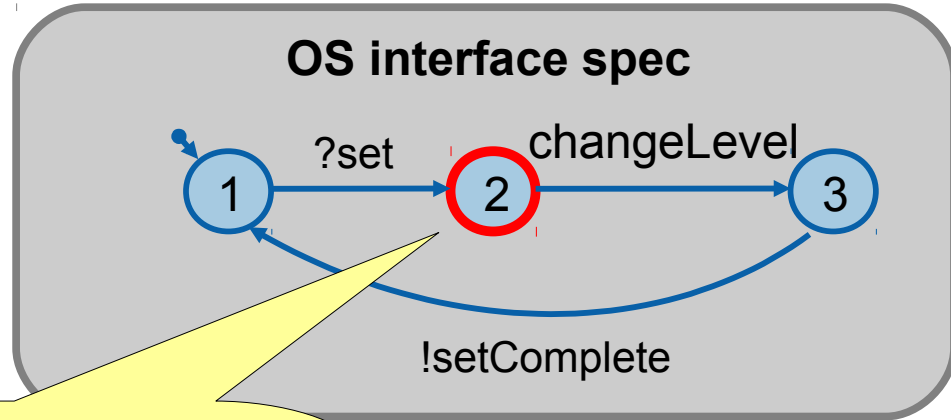
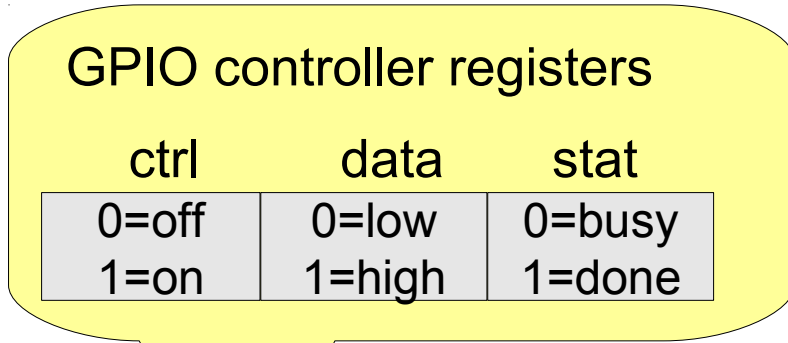
Device spec



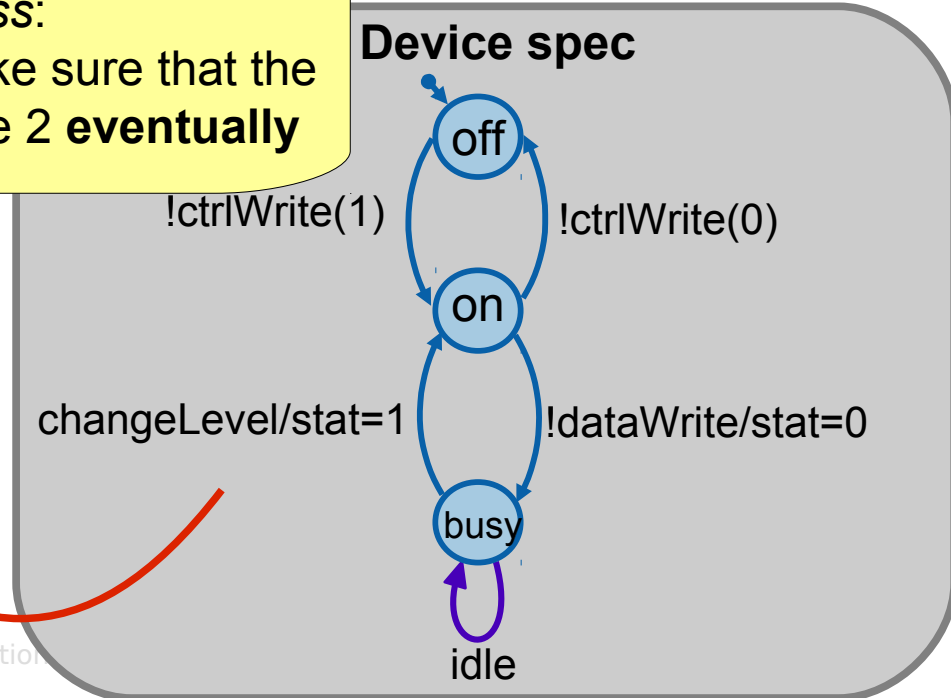
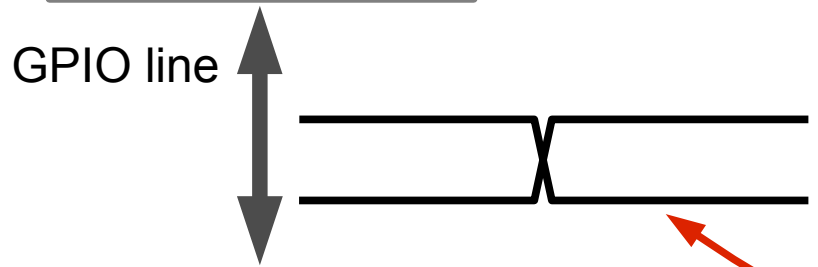
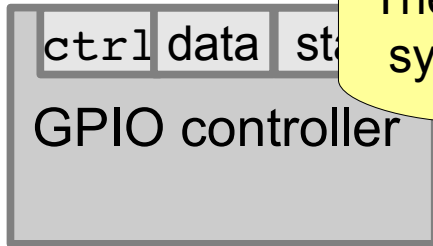
Example: GPIO controller



Example: GPIO controller

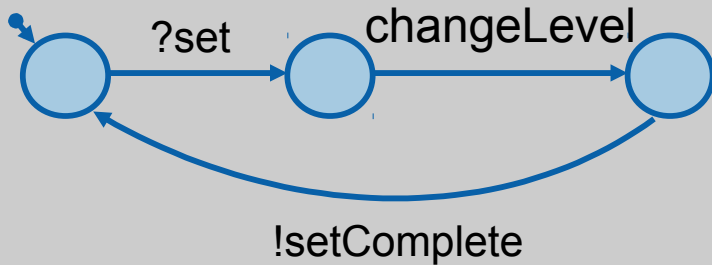


Liveness:
The driver must make sure that the system leaves state 2 **eventually**

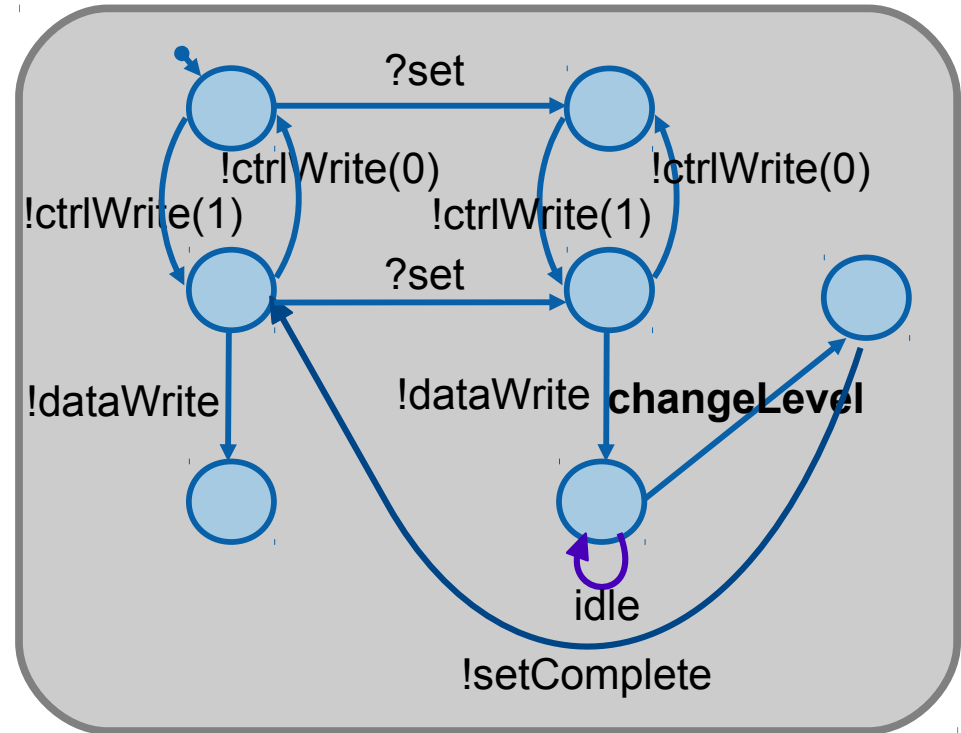
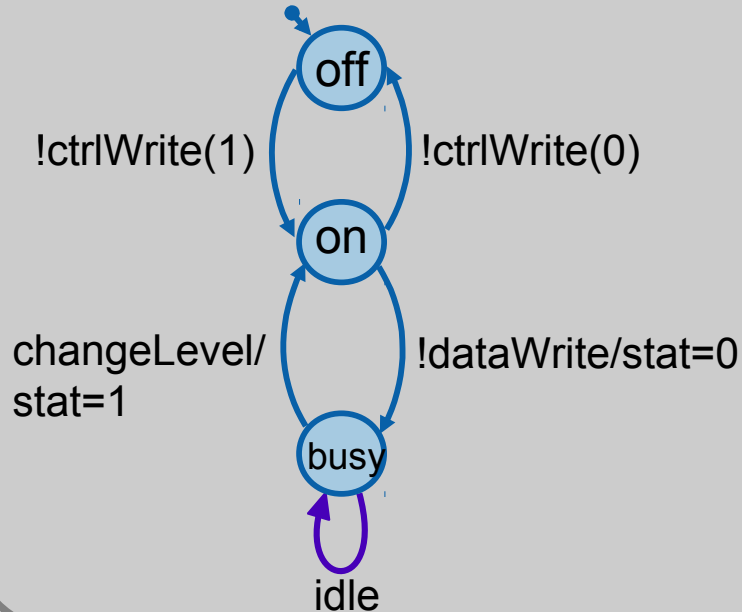


Example: GPIO controller

OS interface spec

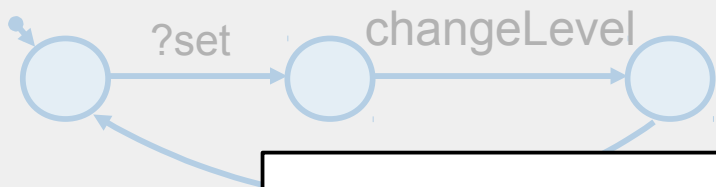


Device spec



Example: GPIO controller

OS interface spec



Control objective:

$$\langle\langle driver \rangle\rangle \left(\bigwedge_{\phi \in \Phi} GF \phi \rightarrow \bigwedge_{\gamma \in \Gamma} GF \gamma \right)$$

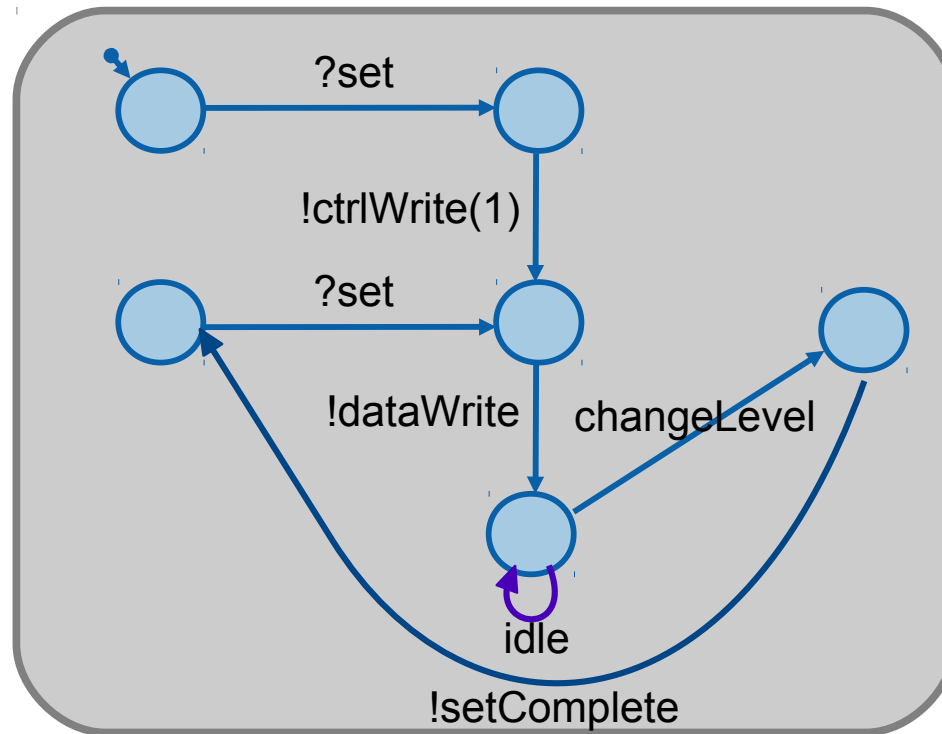
changeLevel/
stat=1

busy

idle

!dataWrite/stat=0

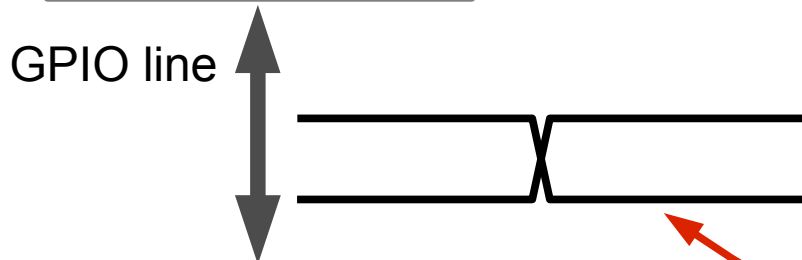
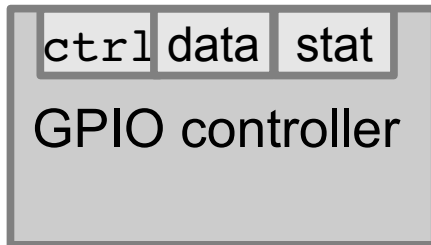
Example: GPIO controller



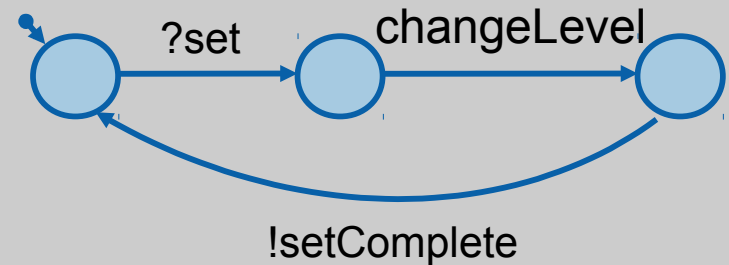
Example: GPIO controller

GPIO controller registers

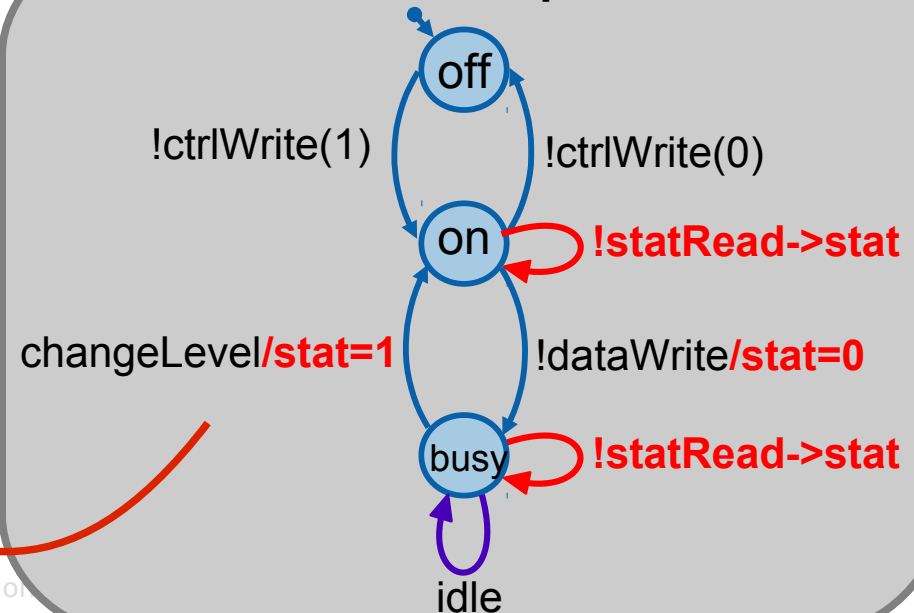
ctrl	data	stat
0=off	0=low	0=busy
1=on	1=high	1=done



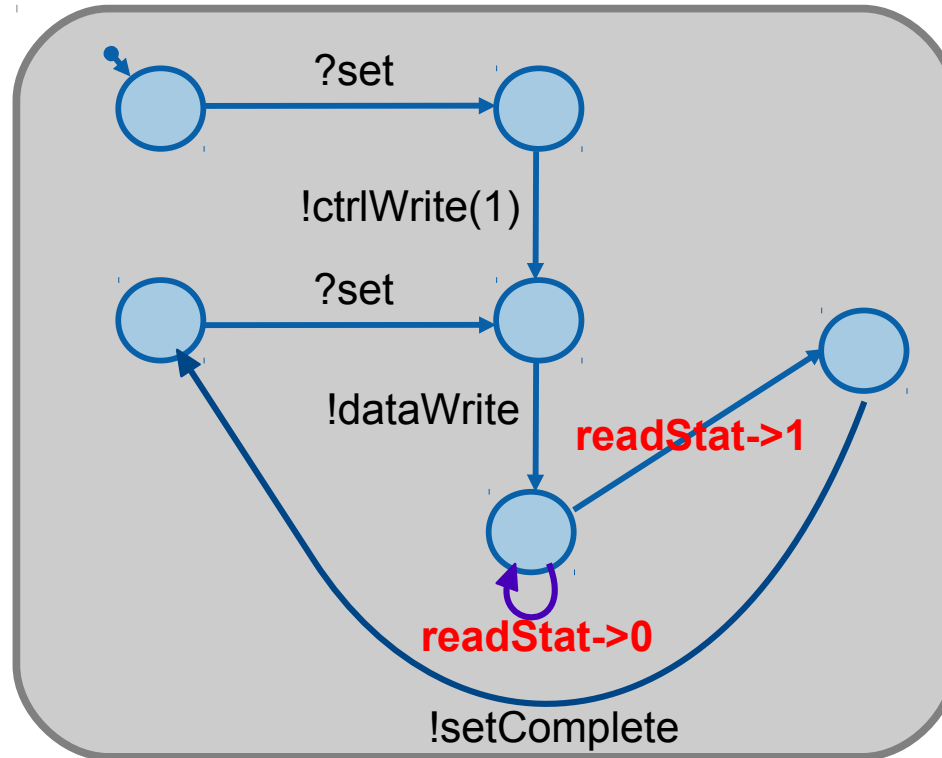
OS interface spec



Device spec



Example: GPIO controller



- Challenge #1: State Explosion
 - Real devices contain dozens of registers, internal FIFOs, support multiple modes of operation and failure modes etc.
 - An efficient synthesis algorithm must rely on abstraction

[1] Henzinger, Jhala, Majumdar, *Counterexample-guided control*, 2003

[2] de Alfaro, Roy, *Solving games via three-valued abstraction Refinement*, 2007

- Challenge #2: Partial Information
 - The driver cannot directly observe device states and state transitions
 - Partial observations can be obtained by reading register values
 - Device state can change between two register accesses

[1] Chatterjee, Doyen, Henzinger, Raskin, *Algorithms for omega-regular games with imperfect information*, 2007

[2] Dimitrova, Finkbeiner, *Abstraction refinement for games with incomplete information*, 2008

Challenges

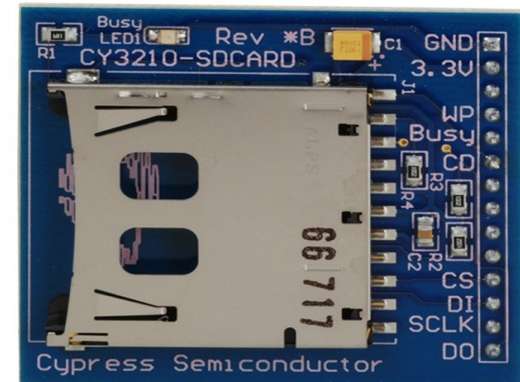


- Challenge #3: Driver synthesis for devices with the DMA capability
- Challenge #4: Efficient C code generation
- Challenge #5: Synthesising concurrent implementations
- Challenge #6: Optimal synthesis
- Challenge #7: Verifying synthesised drivers

Results

Proof-of-concept implementation:

- Successfully synthesised drivers for real devices based on manually written specifications:
 - Asix AX88772 USB-to-Ethernet adapter
 - Ricoh R5C822 SD host controller
- Performance on par with manually developed drivers



Conclusions



- Automatic driver synthesis is a radical approach to improving OS reliability
- Driver synthesis poses interesting challenges in the area of controller synthesis algorithms