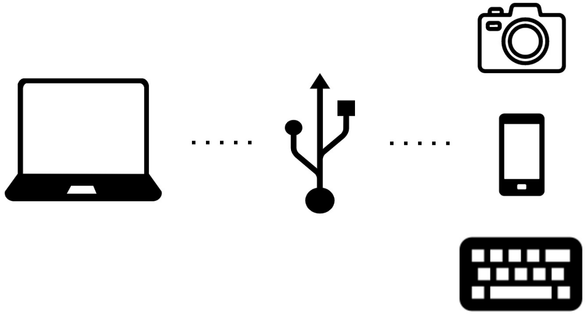
Refinement-Based Verification of Device-to-Device Information Flow

Ning Dong, Roberto Guanciale, Mads Dam
KTH Royal Institute of Technology
Email: {dongn, robertog, mfd}@kth.se

Formal Methods in Computer-Aided Design 2021

Background



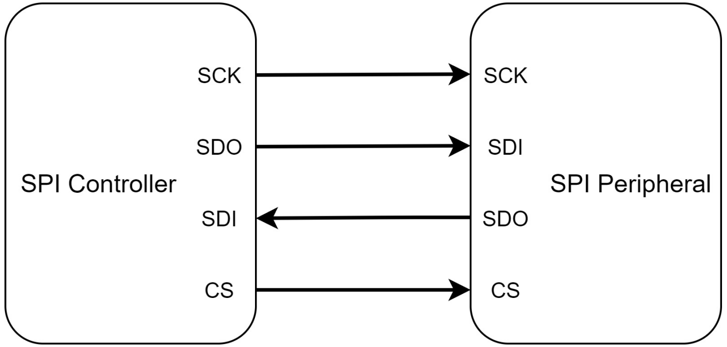
I/O devices: a bridge but bring challenges

- poorly written drivers cause OS crashes [1]
- vulnerable to side-channel attacks [2]
- threats for data stream integrity and confidentiality

1. Chou, Andy, et al. "An empirical study of operating systems errors." Proceedings of the eighteenth ACM symposium on Operating systems principles. 2001.

2. Schmidt, Jörn-Marc, et al. "Side-channel leakage across borders." International Conference on Smart Card Research and Advanced Applications. Springer, Berlin, Heidelberg, 2010.

Serial Peripheral Interface

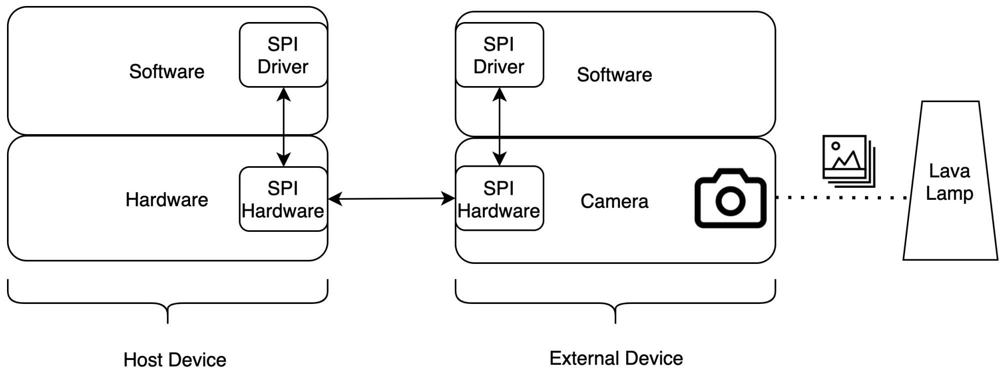


Serial Peripheral Interface (SPI)

- a single controller – multiple peripherals architecture
- full-duplex and half-duplex data transmissions

SCK: serial clock
SDO/SDI: serial data output/input
CS: chip select

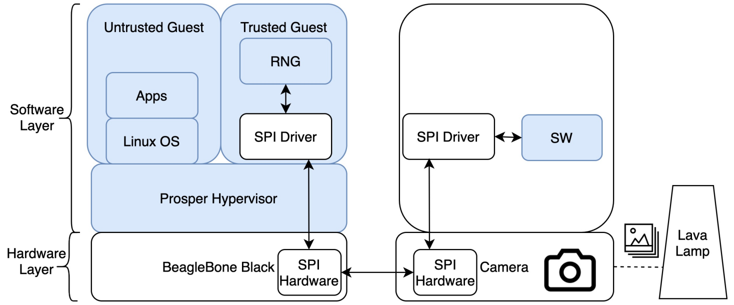
Motivating Example



Hardened random number generator using an external source of physical randomness[1]

1. J. Liebow-Feeser, "Lavarand in production: The nitty-gritty technical details," Apr 2021. [Online]. Available: https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details

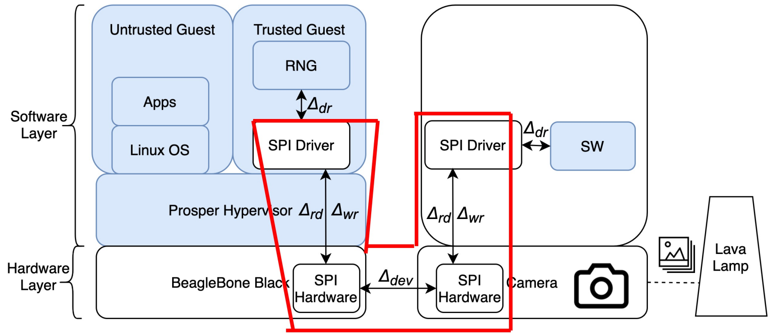
Motivating Example



Threats in the host device, (external device applied also)

- data exfiltration and tampering by the environment
- integrity/confidentiality violation by driver or hardware

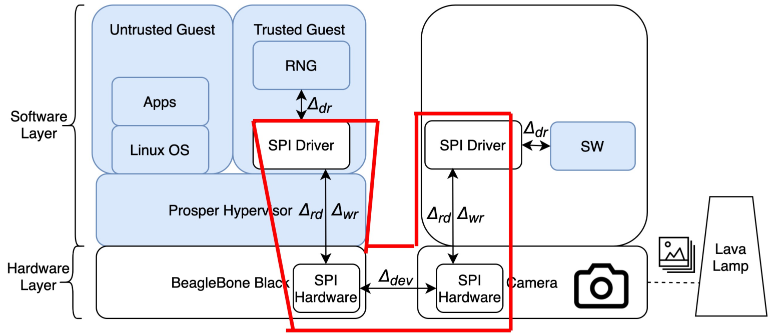
Motivating Example



Threats in the host device, (external device applied also)

- data exfiltration and tampering by the environment
- integrity/confidentiality violation by driver or hardware

Motivating Example



Threats in the host device, (external device applied also)

- data exfiltration and tampering by the environment
- integrity/confidentiality violation by driver or hardware

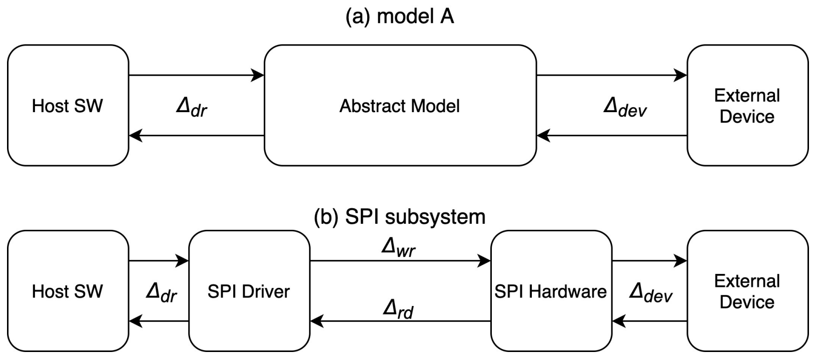
Q: how to protect data streams from integrity and confidentiality attacks considering SPI?

Contribution

Our work has been carried out in HOL4

- Formal model of SPI and its driver
- Abstract model for I/O devices and their drivers
- Establish a weak bisimulation relation between models at different levels
- System properties
- Information flow analysis

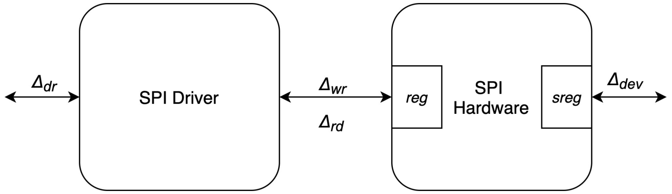
Model Overview



Abstract and concrete models: CCS style (e.g., $rd\ a\ v, \overline{rd\ a\ v}$)

- memory operations (driver-to-device): $\Delta_{wr}\ \Delta_{rd}$
- data transmissions (device-to-device): Δ_{dev}
- driver's interface (software-to-driver): Δ_{dr}

Formal Models



Concrete model:

$s = (regs, sreg, c) \xrightarrow{l} s'$

$d = (b_1, b_2, idx, last\_read\_v, c) \xrightarrow{l} d'$

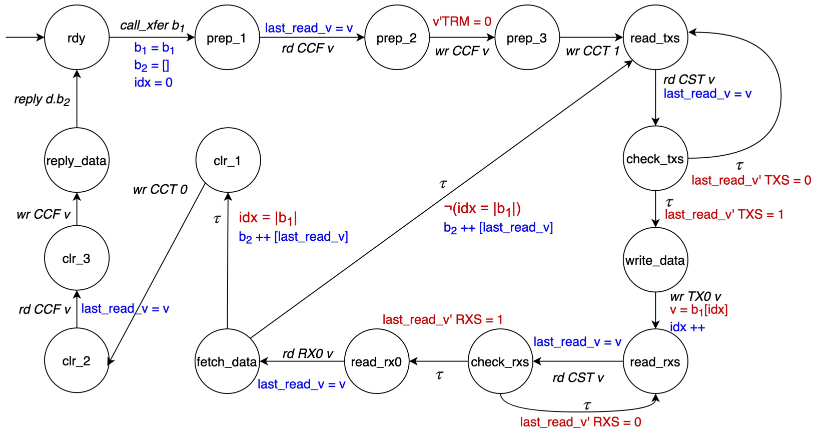
$(d|s)\backslash(\Delta_{wr} \cup \Delta_{rd})$

Abstract model: describe visible effects of the subsystem

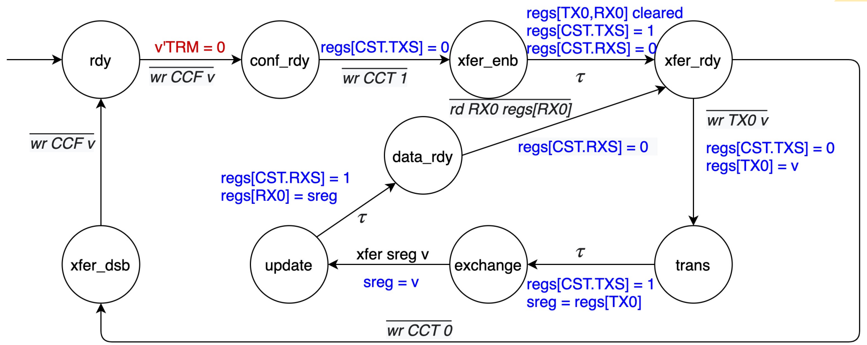
$a = (t, c) \xrightarrow{l} a' \ where \ t = (b_1, b_2, idx, v)$

Functionalities: initialization, transmission, reception, full-duplex synchronous transfer

Control Automaton



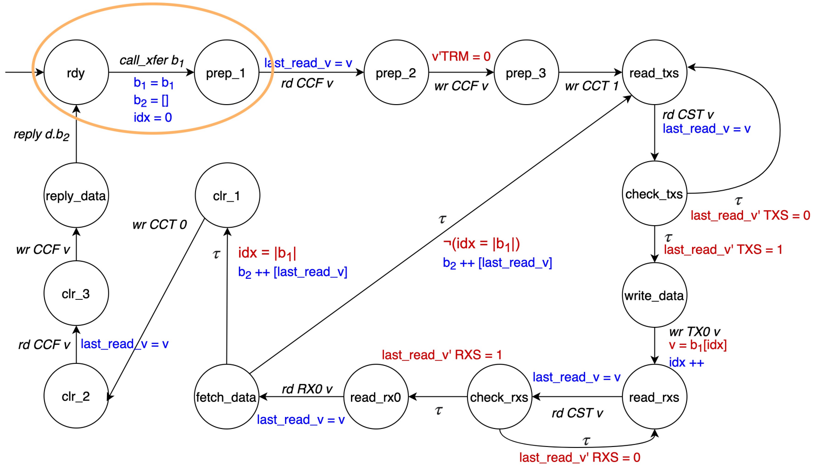
Driver synchronous transfer automaton



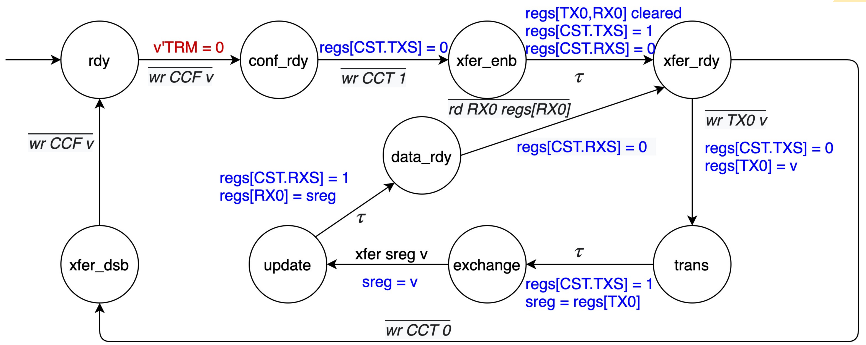
Hardware synchronous transfer automaton

Full-duplex synchronous transfer: data sending and receiving simultaneously
black: transition labels
blue: side effects
red: enabling conditions

Control Automaton



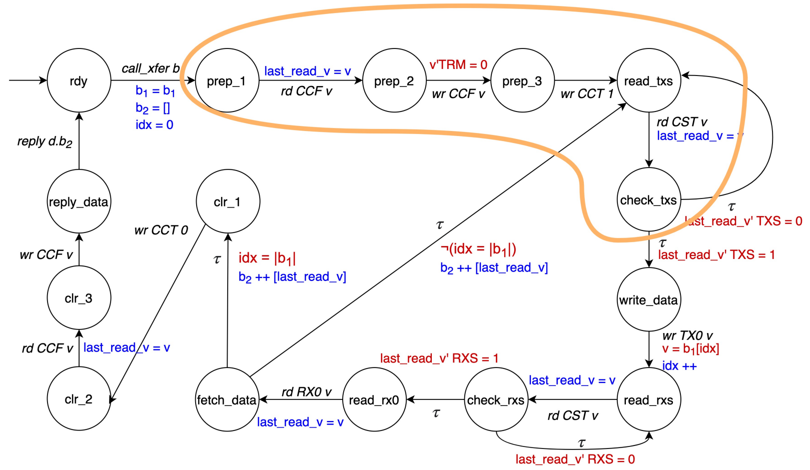
Driver synchronous transfer automaton



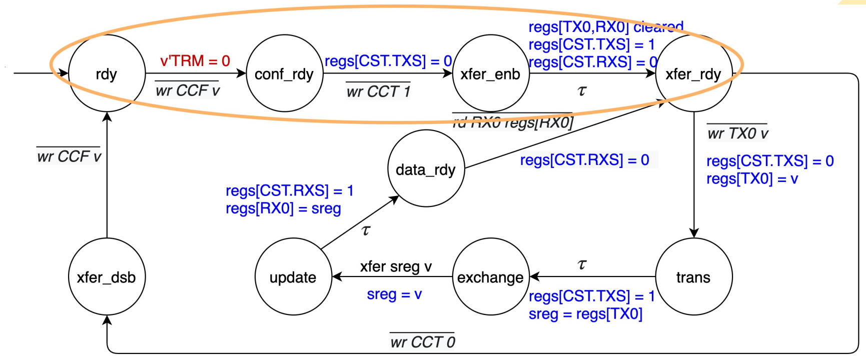
Hardware synchronous transfer automaton

Full-duplex synchronous transfer: data sending and receiving simultaneously
black: transition labels
blue: side effects
red: enabling conditions

Control Automaton



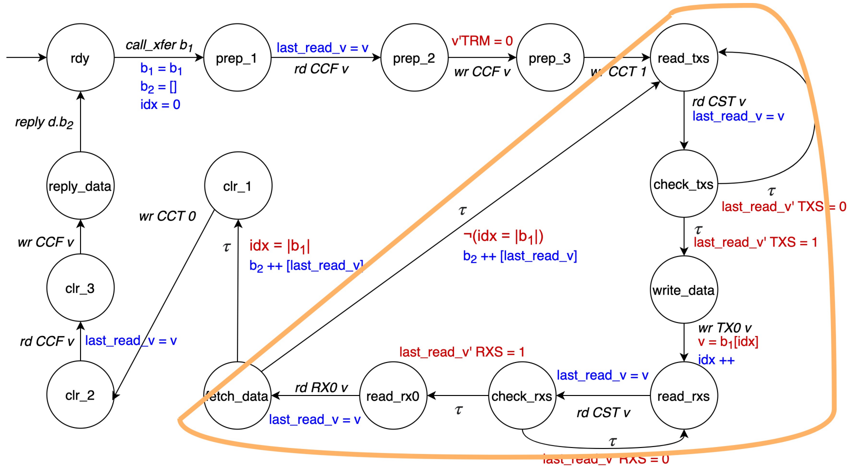
Driver synchronous transfer automaton



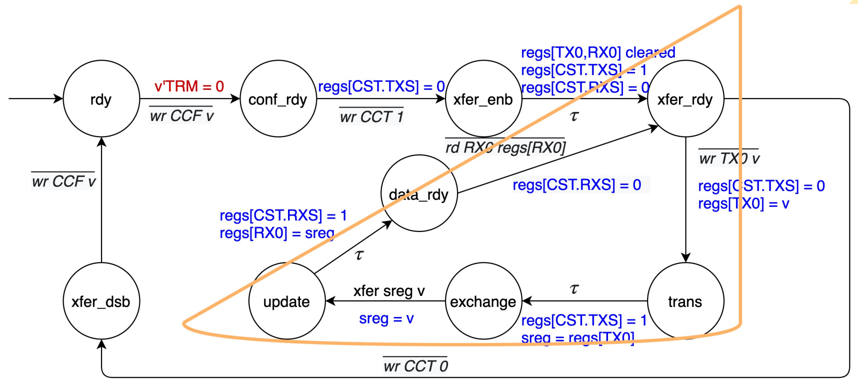
Hardware synchronous transfer automaton

Full-duplex synchronous transfer: data sending and receiving simultaneously
black: transition labels
blue: side effects
red: enabling conditions

Control Automaton



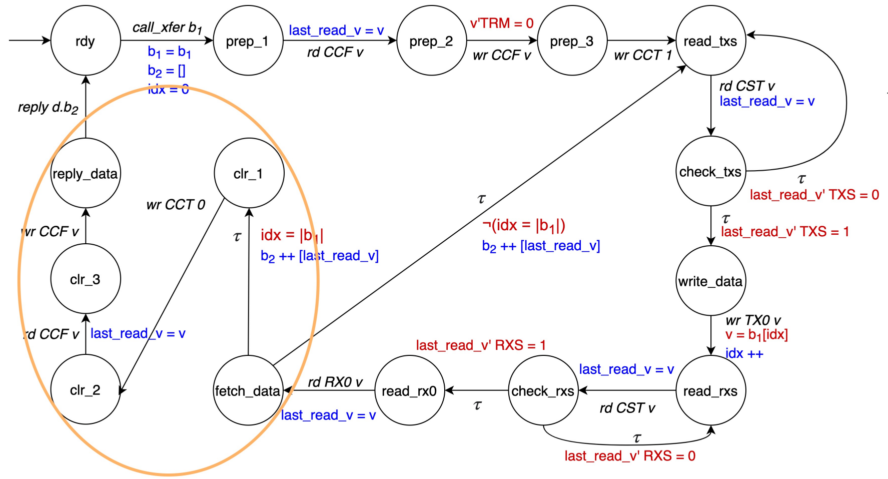
Driver synchronous transfer automaton



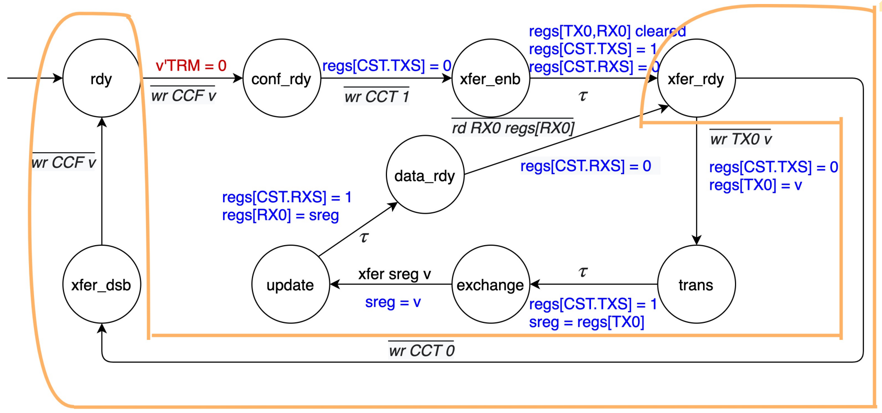
Hardware synchronous transfer automaton

Full-duplex synchronous transfer: data sending and receiving simultaneously
black: transition labels
blue: side effects
red: enabling conditions

Control Automaton



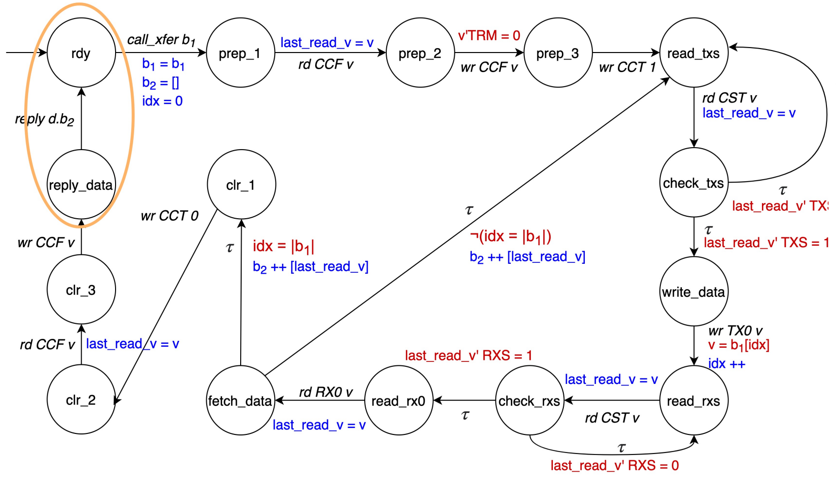
Driver synchronous transfer automaton



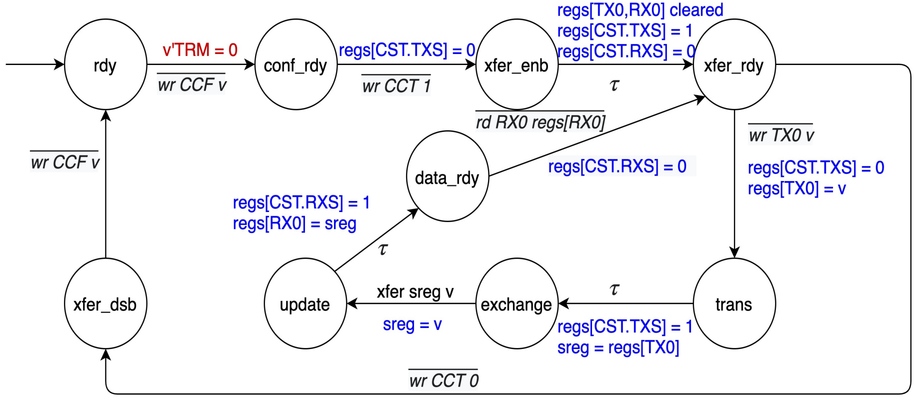
Hardware synchronous transfer automaton

Full-duplex synchronous transfer: data sending and receiving simultaneously
black: transition labels
blue: side effects
red: enabling conditions

Control Automaton



Driver synchronous transfer automaton



Hardware synchronous transfer automaton

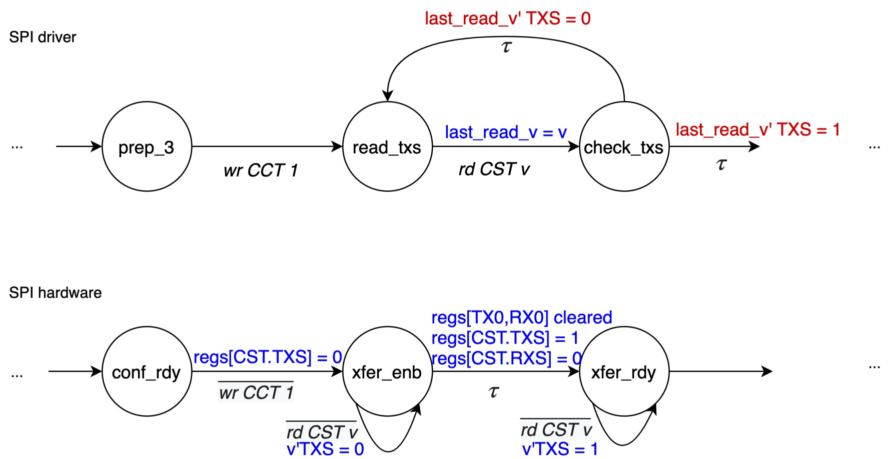
Full-duplex synchronous transfer: data sending and receiving simultaneously

black: transition labels

blue: side effects

red: enabling conditions

Control Automaton

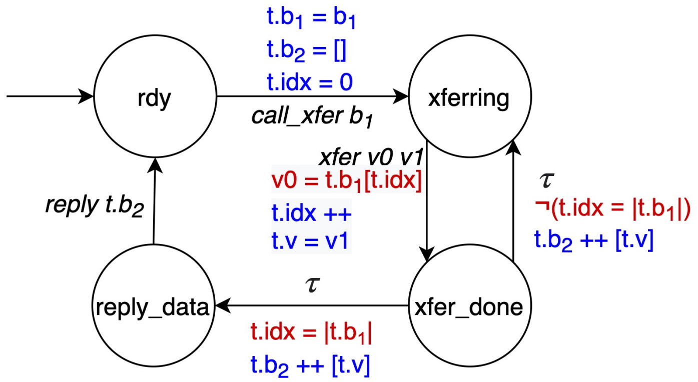


Memory-mapped registers: **CCT** (channel control), **CST** (channel status), **TX0** (transmit buffer), **RX0** (receive buffer)

Bits of **CST**: **TXS**(transmitter register status), **RXS**(receiver register status)

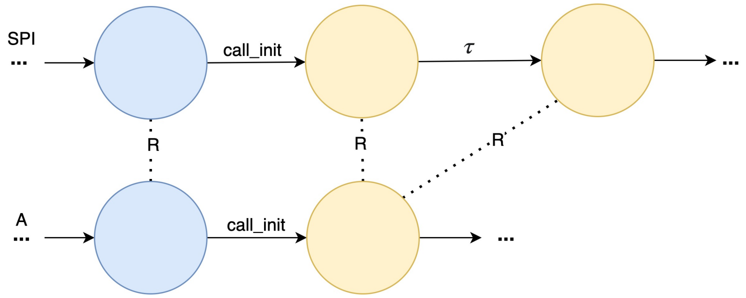
τ: internal operations

Control Automaton



Same automaton in the abstract model:
describe the expected results of I/O device and its driver's executions

Refinement



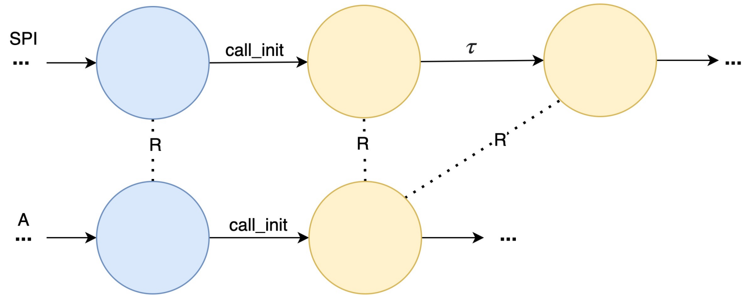
τ: driver/hardware internal, read and write operations.

Weak simulation:
given two transition systems $(S, \rightarrow_1)$ and $(T, \rightarrow_2)$, a binary relation R is a weak simulation if for every $(p, q) \in R$

- if $p \overset{a}{\rightarrow}_1 p'$, then $\exists q'. q \overset{\tau^* a}{\longrightarrow}_2 q' \wedge (p', q') \in R$
- if $p \overset{\tau}{\rightarrow}_1 p'$, then $\exists q'. p \overset{\tau^*}{\rightarrow}_2 p' \wedge (p', q') \in R$

R is a weak bisimulation if both R and R^{-1} are weak simulations, $S \sim_R T$

Refinement



τ: driver/hardware internal, read and write operations.

Weak simulation:
given two transition systems $(S, \rightarrow_1)$ and $(T, \rightarrow_2)$, a binary relation R is a weak simulation if for every $(p, q) \in R$

- if $p \overset{a}{\rightarrow}_1 p'$, then $\exists q'. q \overset{\tau^* a}{\longrightarrow}_2 q' \wedge (p', q') \in R$
- if $p \overset{\tau}{\rightarrow}_1 p'$, then $\exists q'. p \overset{\tau^*}{\rightarrow}_2 p' \wedge (p', q') \in R$

R is a weak bisimulation if both R and R^{-1} are weak simulations, $S \sim_R T$

Weak bisimulation is transitive, and compositional (parallel composition).

Refinement

How to prove it?

$$(d|s)\backslash\{\Delta_{wr} \cup \Delta_{rd}\} \sim_R a$$

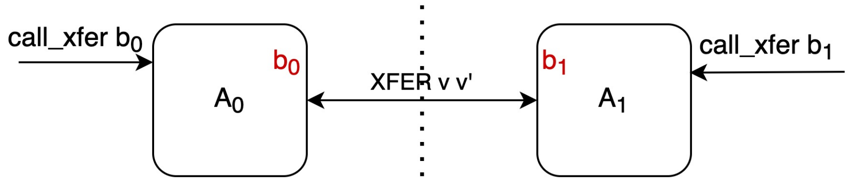
An intermediate model $\mathbf{b}$:
removes memory-mapped registers and related operations, but keeps the shift register
Lines of code: ~1100 (SPI), ~580 ($\mathbf{b}$), ~330 (a).

$$(d|s)\backslash\{\Delta_{wr} \cup \Delta_{rd}\} \sim_{R_1} b$$
$$b \sim_{R_2} a$$

Therefore

$$(d|s)\backslash\{\Delta_{wr} \cup \Delta_{rd}\} \sim_{R_1 \circ R_2} a$$

System Properties of Abstract Model

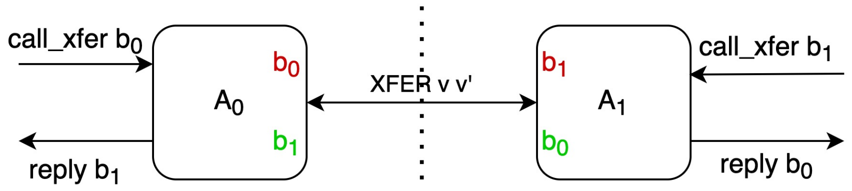


1. transmission, reception, full-duplex synchronous transfer: data flow is correct

2. initialization: done properly

3. no erroneous state: never enters the state $\perp$

System Properties of Abstract Model

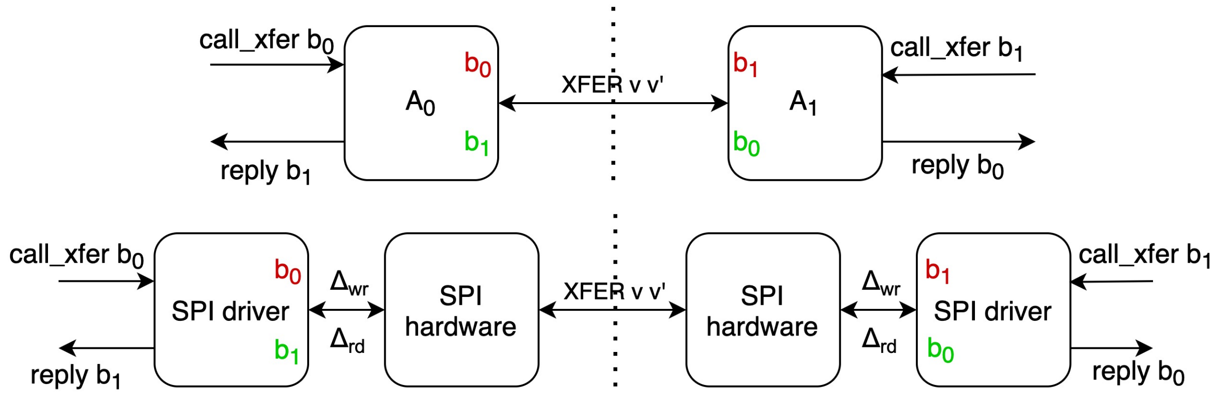


1. transmission, reception, full-duplex synchronous transfer: data flow is correct

2. initialization: done properly

3. no erroneous state: never enters the state ⊥

System Properties



System properties are lifted from the abstract model to the SPI subsystem via weak bisimulation.

Information Flow Analysis

(weak) bisimulation VS (weak) simulation ?

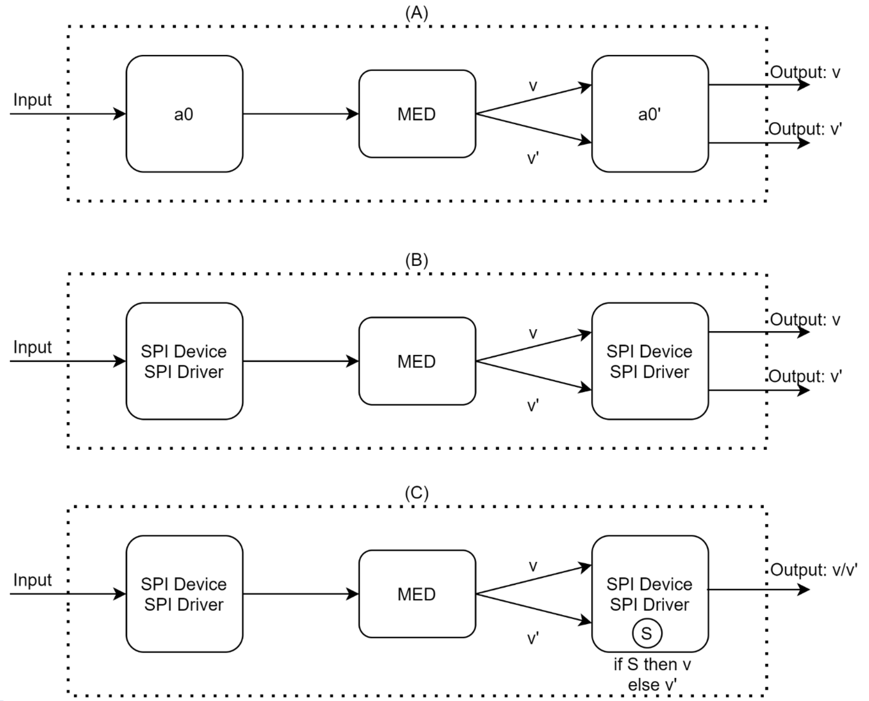
1. progress-sensitive noninterference (PSNI)[1]:
if for every complete run π_1 starting from s, there exists a complete run π_2 starting from t such that $O(\pi_1) = O(\pi_2)$, and vice versa, then s and t are PSNI. Here $O(\pi)$ extracts observable transition labels from π.

Abstract model and SPI subsystem:
avoid malicious driver behaviours, e.g., leak sensitive data by nonterminating

```
while (s)
{/* empty body */}
```

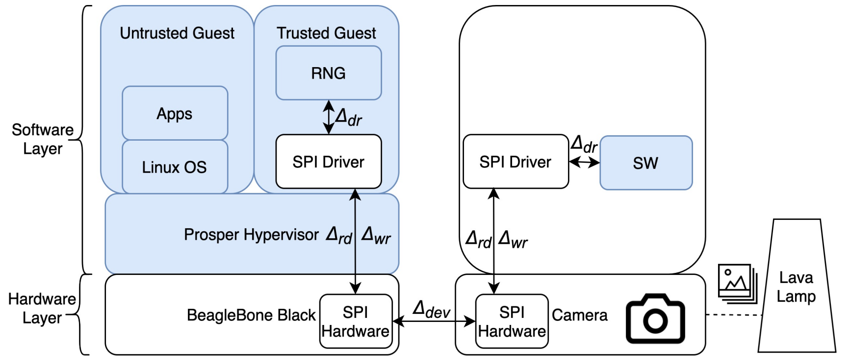
1. D. Hedin and A. Sabelfeld, “A perspective on information-flow control,” in Software safety and security. IOS Press, 2012, pp. 319–347.

Information Flow Analysis



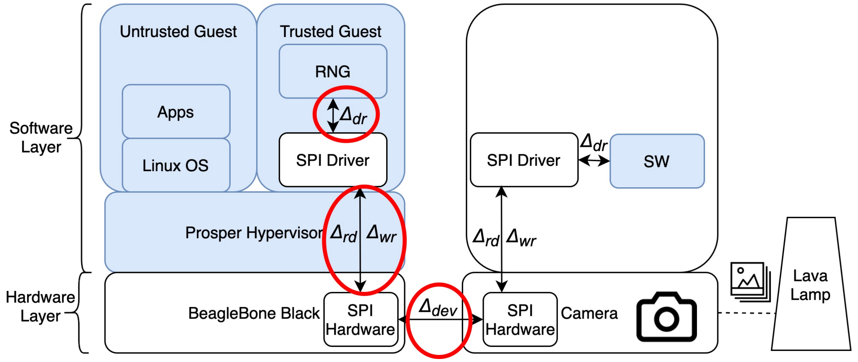
2. compose with non-deterministic components safely, for example a faulty communication medium

Application



Random number generator

Application



Random number generator

Evaluation

Lines of source code in HOL4:

- SPI model: ~1150 (hardware ~560, driver ~590)
- Abstract model: ~330
- Proofs: ~6700

Time: ~6 minutes on a 2,2 GHz 6-Core Intel Core i7 CPU with 16GB RAM

10 man-months of work

Hardware reference manual: not enough for the hardware model, e.g., order of writing SPI registers is unclear.

Conclusion and Future work

- ✓ Modelled and verified an SPI subsystem
- ✓ Proposed a correct-by-construction abstract model
- ✓ Established a weak bisimulation relation between our models at different levels
- ✓ Information flow analysis, PSNI and safe composition

- ➢ Binary verification of the device driver
- ➢ DMA and interrupts
- ➢ Information flow analysis for side channels, e.g., timing

Source code: https://github.com/kth-step/sw-spi-cam-model/releases/tag/fmcad