Formal Co-Validation of Low-Level Hardware/Software Interfaces

Alex Horn[1] Michael Tautschnig[1] Celina Val[2] Lihao Liang[1]
Tom Melham[1] Jim Grundy[3] Daniel Kroening[1]

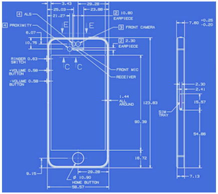
[1]University of Oxford

[2]University of British Columbia

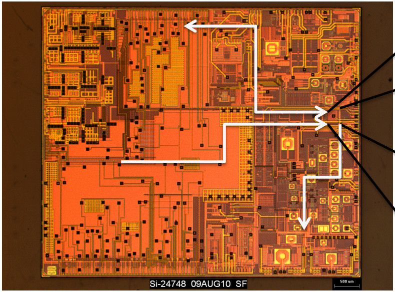
[3]Intel Corporation

October 22, 2013

Motivation



The New Product



Firmware

Consider this scenario:

- The product won't function unless there is firmware! So ideally ...
- But the hardware won't be available until shortly before release.

Our focus: how can we formalize hardware/software interfaces?

Current techniques

Well-known firmware development techniques in industry include:

- Using an older version of the hardware (if any!)
  - hard to debug, can hide latent firmware bugs, no guarantee
- Virtual platforms
  - faster turnaround times, easier to debug and test
  - but generally too big to formally analyze

Current techniques

Well-known firmware development techniques in industry include:

- Using an older version of the hardware (if any!)
  - hard to debug, can hide latent firmware bugs, no guarantee
- Virtual platforms
  - faster turnaround times, easier to debug and test
  - but generally too big to formally analyze

Idea: *Verifiable* Virtual Platform

Current techniques

Well-known firmware development techniques in industry include:

- Using an older version of the hardware (if any!)
  - hard to debug, can hide latent firmware bugs, no guarantee
- Virtual platforms
  - faster turnaround times, easier to debug and test
  - but generally too big to formally analyze

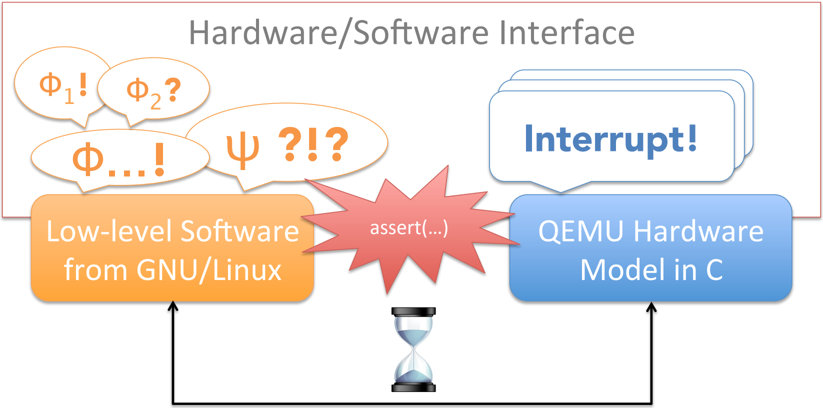
Idea: *Verifiable* Virtual Platform

> *How to model hardware/software interfaces so existing software engineering principles apply but also formal methods*

(See also question posed by Per Bjesse (Synopsys) during FMCAD 2010)

VVP: Verifiable Virtual Platform

realistic • open source • concurrent



Software threads model untimed delays

Outline

An Ethernet MAC concurrency bug

A real bug firmware developers care about

Problem statement and contribution

Technical details

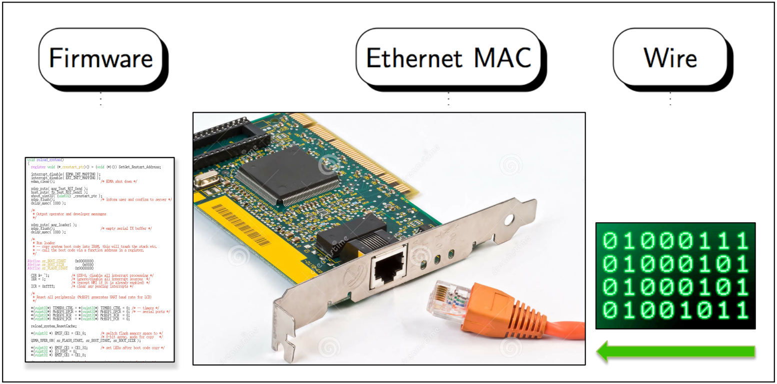
Few glimpses at a verifiable virtual platform

Experiments and conclusion

Download Me!

GNU/Linux + Open Cores Ethernet MAC

Explain known kernel bug due to concurrency (i.e. asynchronous operations) in the hardware/software interface.



Concurrency bug

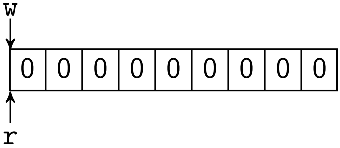
`Interrupt source:`

$\boxed{0_a}$

`Interrupt mode?`

$\boxed{0_x}$

`Buffer descriptors:`



Initially, assume the firmware is in polling mode (i.e. 0_x) and “there are no new RX frames” (yet).

Concurrency bug

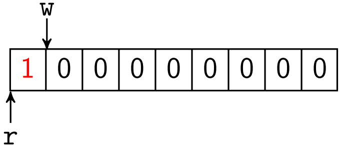
`Interrupt source`:

$\boxed{1_b}$

`Interrupt mode?`

$\boxed{0_x}$

`Buffer descriptors`:



A new RX frame arrives changing the interrupt source from 0_a to 1_b.
The arrival of an RX frame gives us a “nonempty” buffer descriptor.

Concurrency bug

`Interrupt source`:

$\boxed{1_c}$

`Interrupt mode?`

$\boxed{0_x}$

`Buffer descriptors`:

w

1	1	0	0	0	0	0	0	0

r

Repeat but notice that the Open Cores Ethernet MAC always sets the interrupt source register as new RX frames arrive (1_b has become 1_c).

Concurrency bug

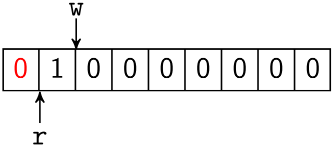
Interrupt source:

$\boxed{1_c}$

Interrupt mode?

$\boxed{0_x}$

Buffer descriptors:



The firmware reads one "nonempty" buffer descriptor changing it to be "empty" again.

Concurrency bug

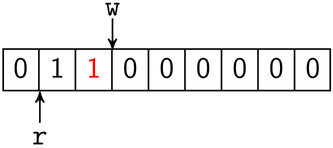
`Interrupt source:`

$\boxed{1_d}$

`Interrupt mode?`

$\boxed{0_x}$

`Buffer descriptors:`



But simultaneously new RX frames can arrive.

Concurrency bug

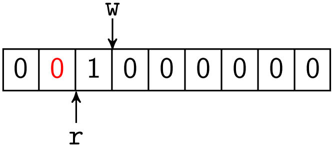
`Interrupt source`:

$\boxed{1_d}$

`Interrupt mode?`

$\boxed{0_x}$

`Buffer descriptors`:



As before, the firmware continues to consume these ...

Concurrency bug

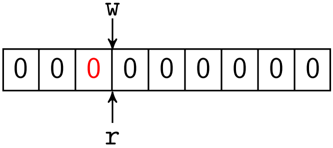
`Interrupt source`:

$\boxed{1_d}$

`Interrupt mode?`

$\boxed{0_x}$

`Buffer descriptors`:



... until it detects that there aren't any more RX frames to consume.
So assume it initiates a procedure now to switch to interrupt mode.

Concurrency bug

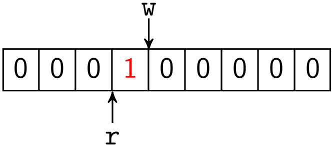
`Interrupt source`:

$\boxed{1_e}$

`Interrupt mode?`

$\boxed{0_x}$

`Buffer descriptors`:



Asynchronous operation: a fraction of a second later a new RX frame arrives and changes 1_d to 1_e as well as the buffer descriptor.

Concurrency bug

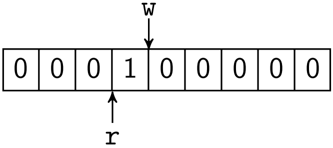
`Interrupt source:`

$\boxed{0_f}$

`Interrupt mode?`

$\boxed{0_x}$

`Buffer descriptors:`



Since firmware is not in interrupt mode yet, it fails to detect the intermittent RX frame; it continues by clearing interrupt sources (0_f).

Concurrency bug

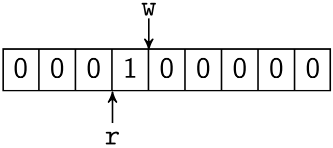
`Interrupt source:`

$\boxed{0_f}$

`Interrupt mode?`

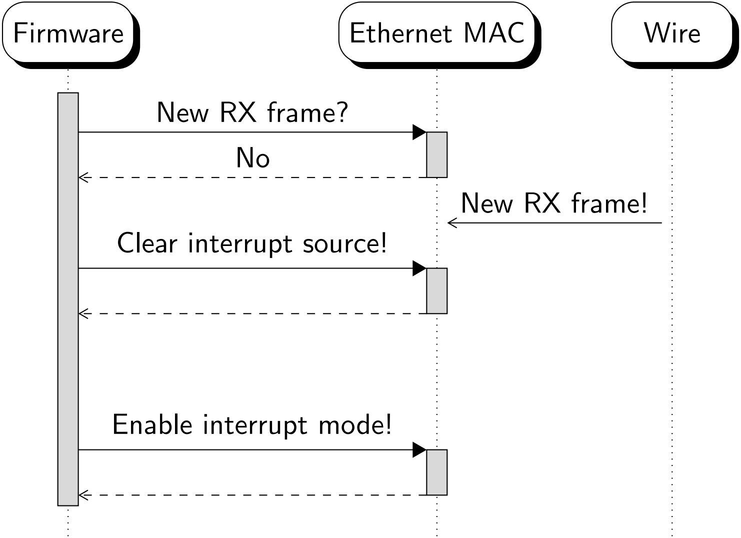
$\boxed{1_y}$

`Buffer descriptors:`

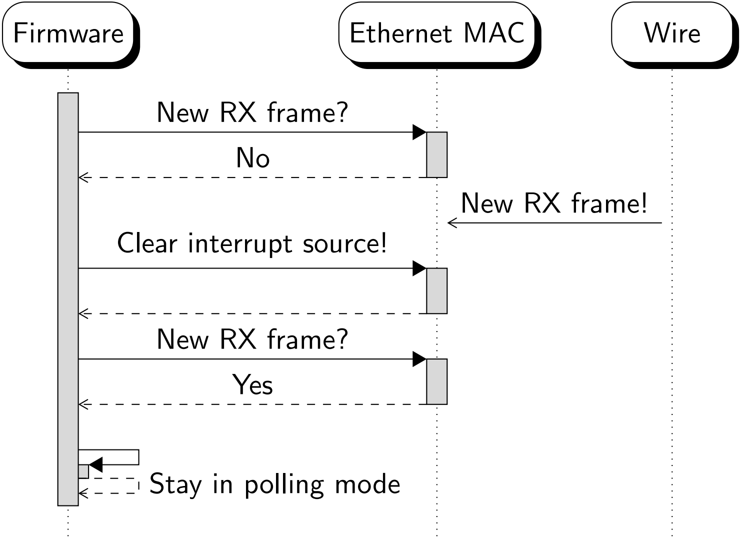


The firmware continues by enabling interrupts (1_y). But an interrupt is only raised once another RX frame arrives, *problem*.

Polling to interrupt mode switch (bug)



Polling to interrupt mode switch (fix)



Problem statement and contribution

Problem:

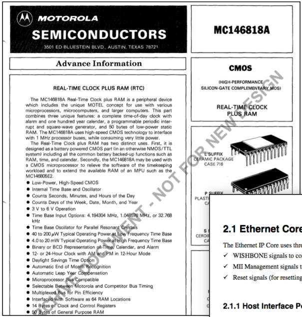
- Many firmware bugs can go undetected when hardware and software are verified in isolation.

Contribution:

- Three realistic and open-source benchmarks to **scientifically study firmware verification**.
- Practical evidence that a verifiable virtual platform is a **feasible concept** to verify hardware/software interfaces.

Benchmarks to study firmware verification

MC146818A: Real-time clock



MOTOROLA
SEMICONDUCTORS
3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

MC146818A

Advance Information

CMOS

(HIGH-PERFORMANCE SILICON-GATE COMPLEMENTARY MOS)

REAL-TIME CLOCK PLUS RAM

REAL-TIME CLOCK PLUS RAM (RTC)

The MC146818A Real-Time Clock plus RAM is a peripheral device which includes the unique MOTEL concept for use with various microprocessors, microcomputers, and larger computers. This part combines three unique features: a complete time-of-day clock with alarm and one hundred year calendar, a programmable periodic interrupt and square-wave generator, and 50 bytes of low-power static RAM. The MC146818A uses high-speed CMOS technology to interface with 1 MHz processor buses, while consuming very little power.

The Real-Time Clock plus RAM has two distinct uses. First, it is designed as a battery powered CMOS part (in an otherwise NMOS/TTL system) including all the common battery backed-up functions such as RAM, time, and calendar. Secondly, the MC146818A may be used with a CMOS microprocessor to relieve the software of the timekeeping workload and to extend the available RAM of an MPU such as the MC146805E2.

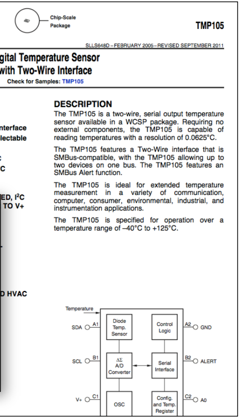
- Low-Power, High-Speed CMOS
- Internal Time Base and Oscillator
- Counts Seconds, Minutes, and Hours of the Day
- Counts Days of the Week, Date, Month, and Year
- 3 V to 6 V Operation
- Time Base Input Options: 4.194304 MHz, 1.048576 MHz, or 32.768 kHz
- Time Base Oscillator for Parallel Resonant Crystals
- 40 to 200 μW Typical Operating Power at Low Frequency Time Base
- 4.0 to 20 mW Typical Operating Power at High Frequency Time Base
- Binary or BCD Representation of Time, Calendar, and Alarm
- 12- or 24-Hour Clock with AM and PM in 12-Hour Mode
- Daylight Savings Time Option
- Automatic End of Month Recognition
- Automatic Leap Year Compensation
- Microprocessor Bus Compatible
- Selectable Between Motorola and Competitor Bus Timing
- Multiplexed Bus for Pin Efficiency
- Interfaced with Software as 64 RAM Locations
- 14 Bytes of Clock and Control Registers
- 50 Bytes of General Purpose RAM

L SUFFIX
CERAMIC PACKAGE
CASE 716

P SUFFIX

NOT FOR NEW DESIGN

TMP105: Temperature Sensor



TEXAS INSTRUMENTS

Chip-Scale Package

TMP105

www.ti.com

SLLS648D – FEBRUARY 2005 – REVISED SEPTEMBER 2011

Digital Temperature Sensor
with Two-Wire Interface

Check for Samples: TMP105

FEATURES

- SUPPORTS 1.8V I²C™ BUS
- TWO ADDRESSES
- DIGITAL OUTPUT: Two-Wire Serial Interface
- RESOLUTION: 9- to 12-Bits, User-Selectable
- ACCURACY:
  - ±2.0°C (max) from –25°C to +85°C
  - ±3.0°C (max) from –40°C to +125°C
- LOW QUIESCENT CURRENT: 50μA, 1.5μA Standby

ED, I²C
TO V+

D HVAC

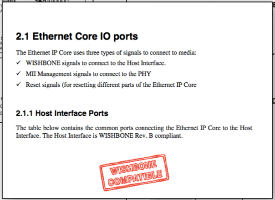
DESCRIPTION

The TMP105 is a two-wire, serial output temperature sensor available in a WCSP package. Requiring no external components, the TMP105 is capable of reading temperatures with a resolution of 0.0625°C.

The TMP105 features a Two-Wire interface that is SMBus-compatible, with the TMP105 allowing up to two devices on one bus. The TMP105 features an SMBus Alert function.

The TMP105 is ideal for extended temperature measurement in a variety of communication, computer, consumer, environmental, industrial, and instrumentation applications.

The TMP105 is specified for operation over a temperature range of –40°C to +125°C.



2.1 Ethernet Core IO ports

The Ethernet IP Core uses three types of signals to connect to media:

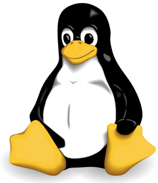
- ✓ WISHBONE signals to connect to the Host Interface.
- ✓ MII Management signals to connect to the PHY
- ✓ Reset signals (for resetting different parts of the Ethernet IP Core

2.1.1 Host Interface Ports

The table below contains the common ports connecting the Ethernet IP Core to the Host Interface. The Host Interface is WISHBONE Rev. B compliant.

WISHBONE COMPATIBLE

Ethernet MAC



CBMC Download

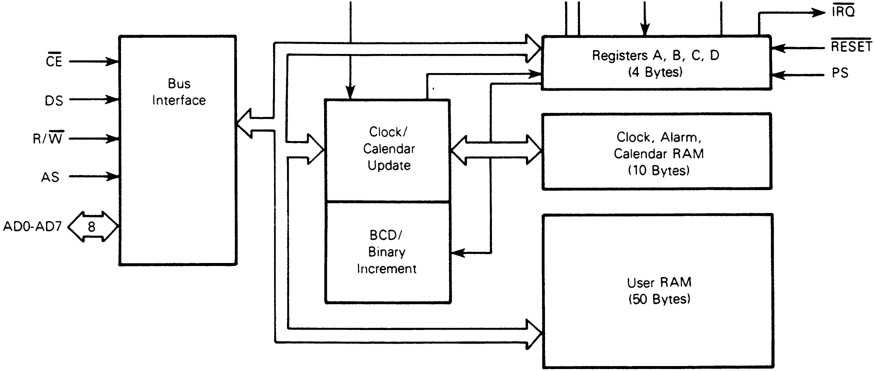
QEMU
open source processor emulator

Experimental setup

Overview of work flow:

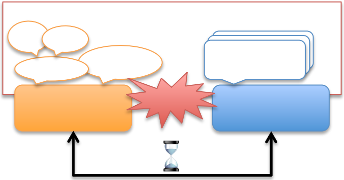
1. Extract QEMU hardware model and Linux driver
2. Manually add runtime assertions in C
3. If necessary, introduce concurrency in QEMU + Linux code
   - Use new CBMC concurrency source code annotations
   - Encode any concurrency as symbolic partial orders (CAV'13)
4. SAT solver finds satisfying assignment (i.e. bug) or not.

Real-time clock (RTC)



Special-purpose registers that require an ancillary manipulation of bits to read and write time, date and alarm data.

RTC benchmark code



Project	Files		LOC
Linux Kernel 3.6	$\sim 14,000$ $\sim 17,000$	(.h) (.c)	$\sim 10^7$
QEMU 1.2	~ 600 $\sim 1,500$	(.h) (.c)	$\sim 700,000$
QEMU hardware model of RTC	5 5	(.h) (.c)	$\sim 1,000$
Linux x86 RTC driver and model	~ 300 8	(.h) (.c)	$\sim 20,000$
Combined RTC benchmark	0 1	(.h) (.c)	$\sim 6,000$

Example Assertion

assert(...)

```
void cmos_ioport_write(void *opaque,
                       uint32_t addr, uint32_t data)
{
    RTCState *s = opaque;
    if ((addr & 1) == 0) {
      s->io_info = OUTB_0x70; // for temporal property
      s->cmos_index = data & 0x7f;
    } else {
        switch(s->cmos_index) {
        case RTC_SECONDS_ALARM:

#ifdef RTC_BENCHMARK_PROP_9
            assert((s->cmos_data[RTC_REG_B] & REG_B_SET)
                     == REG_B_SET);
#endif

          ...
```

Example Assertion


```
void cmos_ioport_write(void *opaque,
                       uint32_t addr, uint32_t data)
{
    RTCState *s = opaque;
    if ((addr & 1) == 0) {
      s->io_info = OUTB_0x70; // for temporal property
      s->cmos_index = data & 0x7f;
    } else {
        switch(s->cmos_index) {
        case RTC_SECONDS_ALARM:

#ifdef RTC_BENCHMARK_PROP_9
            assert((s->cmos_data[RTC_REG_B] & REG_B_SET)
                   == REG_B_SET);
#endif

        ...
```

Example Assertion

assert(...)

```
void cmos_ioport_write(void *opaque,
                       uint32_t addr, uint32_t data)
{
    RTCState *s = opaque;
    if ((addr & 1) == 0) {
      s->io_info = OUTB_0x70; // for temporal property
      s->cmos_index = data & 0x7f;
    } else {
        switch(s->cmos_index) {
        case RTC_SECONDS_ALARM:

#ifdef RTC_BENCHMARK_PROP_9
            assert((s->cmos_data[RTC_REG_B] & REG_B_SET)
                   == REG_B_SET);
#endif


        ...
```

Found bug in RTC hardware model

rtc: Only call rtc_set_cmos when Register B SET flag is disabled.

```
author     Alex Horn <alex.horn@cs.ox.ac.uk>
           Mon, 26 Nov 2012 16:32:54 +0000 (17:32 +0100)
committer  Anthony Liguori <aliguori@us.ibm.com>
           Tue, 27 Nov 2012 17:04:33 +0000 (11:04 -0600)
commit     02c6ccc6dde90dcbf5975b1cfe2ab199e525ec11
tree       0a4286587fa357224cdaebe6c14ff2255b9b84ef      tree | snapshot
parent     03a36f17d7788e4a1e07b3341b18028aa0206845      commit | diff
```

```
rtc: Only call rtc_set_cmos when Register B SET flag is disabled.

This bug occurs when the SET flag of Register B is enabled. When an RTC
data register (i.e. any of the ten time/calender CMOS bytes) is set, the
data is (as expected) correctly stored in the cmos_data array. However,
since the SET flag is enabled, the function rtc_set_time is not invoked.
As a result, the field base_rtc in RTCState remains uninitialized. This
causes a problem on subsequent writes which can end up overwriting data.
To see this, consider writing data to Register A after having written
data to any of the RTC data registers; the following figure illustrates
the call stack for the Register A write operation:

 +- cmos_io_port_write
 +-- check_update_timer
 +---- get_next_alarm
 +------ rtc_update_time

In rtc_update_time, get_guest_rtc calculates the wrong time and
overwrites the previously written RTC data register values.

Signed-off-by: Alex Horn <alex.horn@cs.ox.ac.uk>
Signed-off-by: Paolo Bonzini <pbonzini@redhat.com>
Signed-off-by: Anthony Liguori <aliguori@us.ibm.com>
```

Also found bug in TMP105 hardware model

tmp105: Fix I2C protocol bug

```
author     Andreas Färber <andreas.faerber@web.de>
           Wed, 16 Jan 2013 00:57:56 +0000 (01:57 +0100)
committer  Anthony Liguori <aliguori@us.ibm.com>
           Wed, 16 Jan 2013 18:14:20 +0000 (12:14 -0600)
commit     cb5ef3fa1871522a0886627033459e94bd537fb7
tree       ec94c5b0f0514297227eb6de0a0e432f2affe5a2      tree | snapshot
parent     6d0b430176e3571af0e1596276078f05bfe1c5a5      commit | diff
```

```
tmp105: Fix I2C protocol bug

An early length postincrement in the TMP105's I2C TX path led to
transfers of more than one byte to place the second byte in the third
byte's place within the buffer and the third byte to get discarded.

Fix this by explictly incrementing the length after the checks but
before the callback is called, which again checks the length.

Adjust the Coding Style while at it.

Signed-off-by: Alex Horn <alex.horn@cs.ox.ac.uk>
Signed-off-by: Andreas Färber <andreas.faerber@web.de>
Reviewed-by: Anthony Liguori <aliguori@us.ibm.com>
Signed-off-by: Anthony Liguori <aliguori@us.ibm.com>
```

Experiments

Hardware/software interface properties formally checked on an individual basis:

- 11 RTC properties within a few minutes
- 17 TMP105 properties in less than 15 minutes
- 3 Ethernet MAC properties in sequential code within a few minutes
- 7 Ethernet MAC properties in concurrent code within a few hours[1]

[1] After publication, we found a bug in CBMC's implementation of the partial order concurrency encoding but continue to improve the code. At the present time, we cannot reproduce the results with CBMC for the concurrent model of the Ethernet MAC.

Download Me!

Conclusion:

- Formal verification of hardware/software interface properties written in C code
  - Executable code leverages well-established testing principles in industry
  - Apply multi-path (i.e. CBMC-style) symbolic execution and symbolic partial order encodings to handle concurrency in hardware/software
- Open-source prototype of a verifiable virtual platform (VVP)
  - Provides an object of study for software engineers

All code and documentation is openly available *now*.

Source code, data sheets and experiments can be downloaded at `http://www.cprover.org/firmware/`.

Download Me!

Conclusion:

- Formal verification of hardware/software interface properties written in C code
  - Executable code leverages well-established testing principles in industry
  - Apply multi-path (i.e. CBMC-style) symbolic execution and symbolic partial order encodings to handle concurrency in hardware/software
- Open-source prototype of a verifiable virtual platform (VVP)
  - Provides an object of study for software engineers

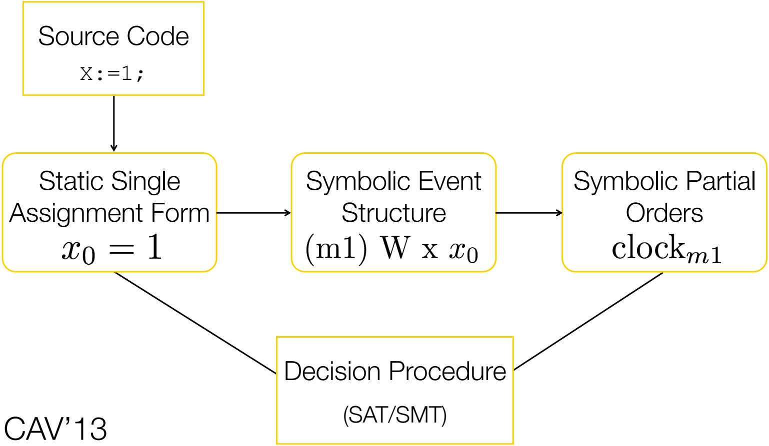
All code and documentation is openly available *now*.

Source code, data sheets and experiments can be downloaded at `http://www.cprover.org/firmware/`.

Thank you.

Symbolic partial order encoding with CBMC

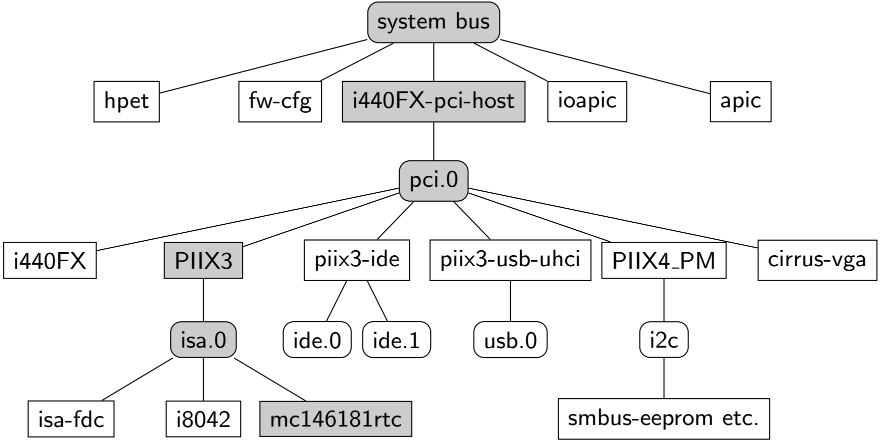
(Not to be confused with partial order reduction.)



CAV'13

RTC/QEMU: QDev and QOM Simplifications

Domain knowledge required:



Related work by Kai Cong et al.

Recent Kai Cong, Fei Xie, and Li Lei publications:

- Symbolic Execution of Virtual Devices (QSIC, 2013).
  - Single-path (KLEE-style) symbolic execution
  - Doesn't symbolically co-execute virtual device and driver
  - Suggests a way to automatically extract QEMU hardware models
- Automatic Concolic Test Generation with Virtual Prototypes for Post-silicon Validation (ICCAD, 2013).
  - Uses QEMU and KLEE
  - Records concrete hardware/software interactions