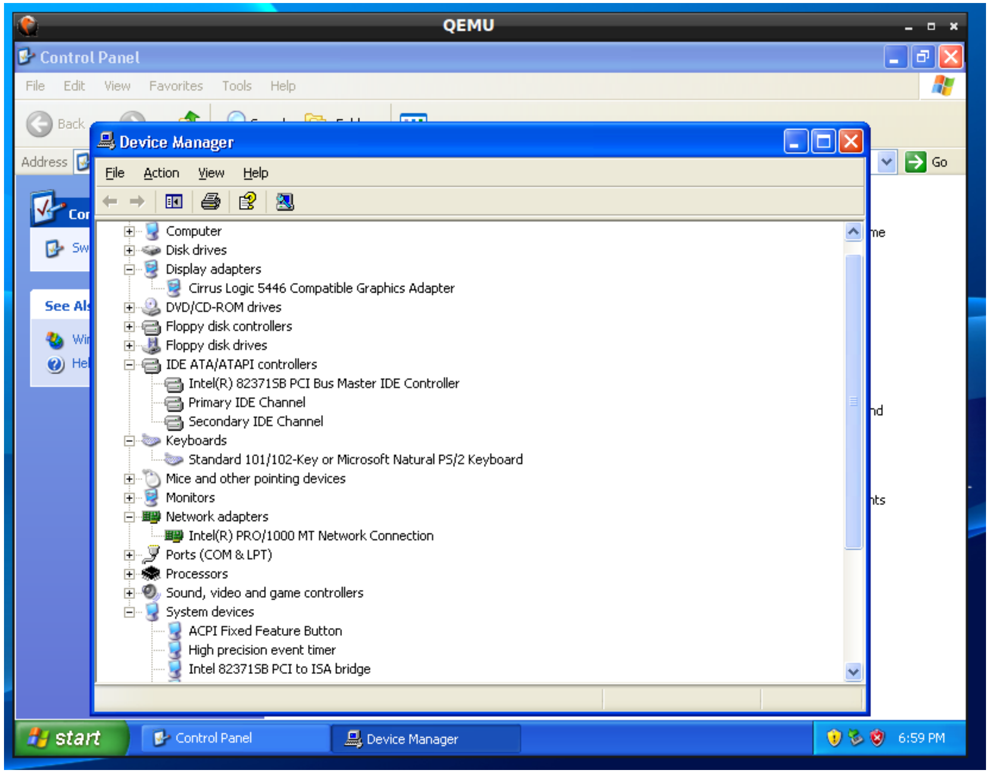
Detecting Vulnerabilities in Virtual Devices with Conformance Testing

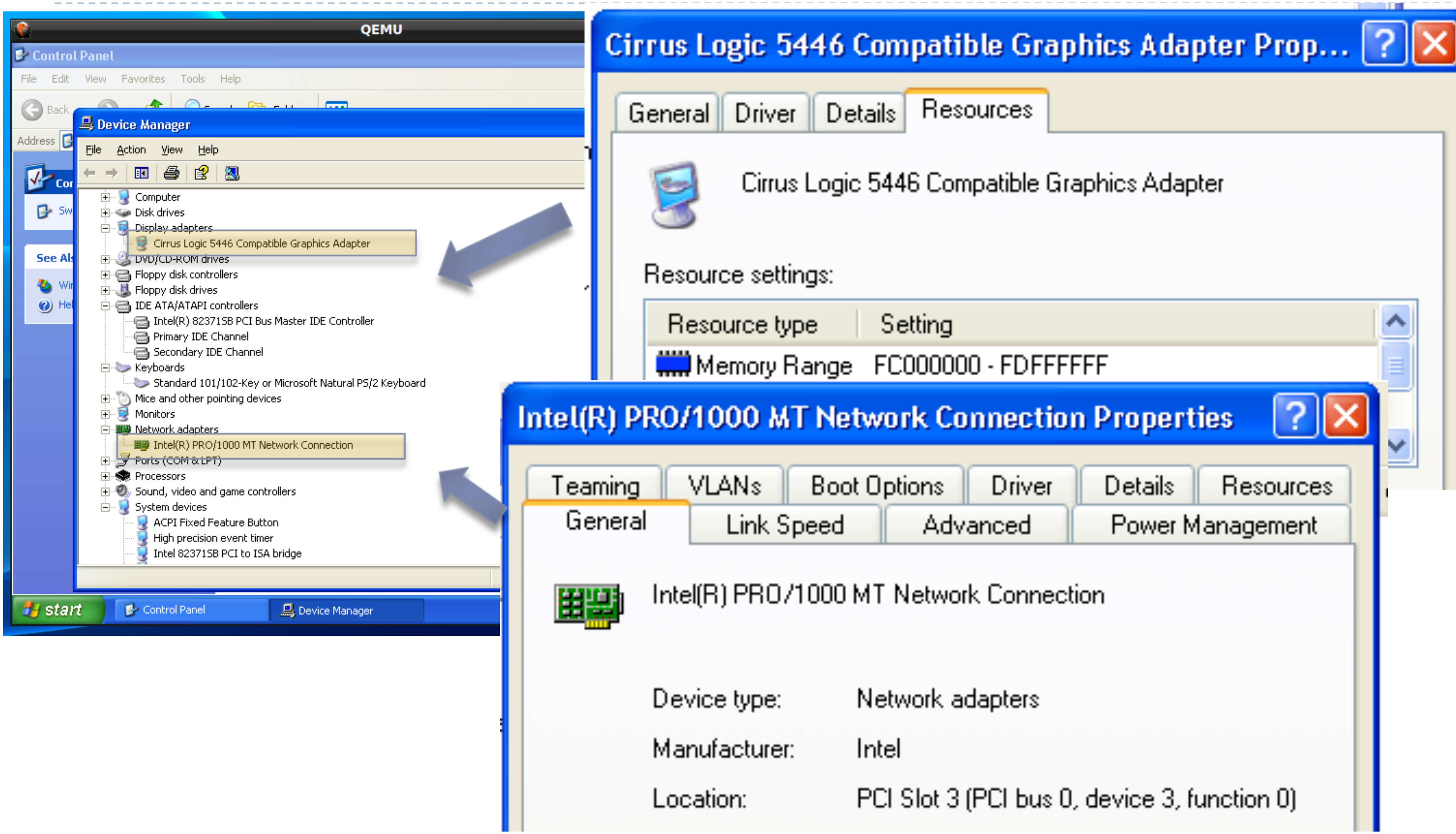
-- Turning old hardware Into gold

Kang Li kodos @disekt.org
Michael Contreras maikol@disekt.org

VM and Virtual Devices



QEMU and its Virtual Devices



QEMU and its Virtual Devices

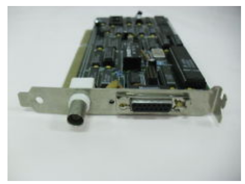
- QEMU
  - Open-source hypervisor with hardware virtualization
  - Share common components with KVM and VirtualBox
- Many types of devices
  - Graphics
  - Network
  - Disk Controller
  - USB
  - …

QEMU and its Virtual Devices

- Multiple devices in the same category
  - Network interfaces

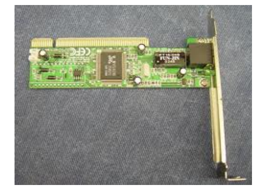
Intel
E1000

Intel
eepro100



Novell
NE2000

AMD
PC-NET II



Realtek
RTL8139

SMCS
LAN9118

- What's common about these devices?
  - All pictures copied from ebay, all below $10.
  - Old devices that you have probably thrown away!
  - VMs (and Cloud) are "faking" these old devices.

VM and Cloud provide an illusion

When virtual devices are done right

What if the virtual device goes wrong?

Ensuring virtual devices are done right

Conformance Testing --

compare a virtual device with its physical counterpart

Conformance Testing is like Comparing …

Original (**OLD**)

Remake (**NEW**)
/ Mimic

Example: Mishandled Jumbo Frames

- We discovered memory corruption bug
  - Reported on 12/02/2012
  - Confirmed on 12/06/2012 (CVE 2012-6075)
- Assumptions about the hardware
  - Large packet should be dropped when LPE is disabled.
  - A misunderstanding between driver and device
- Driver optimizes allocations based on this assumption
  - Driver allocates 1522 bytes for buffers, expecting anything larger will be dropped by the NIC
  - Physical hardware respects this limit but virtual device does not
- Mismatching behavior causes overwrite in Guest OS memory

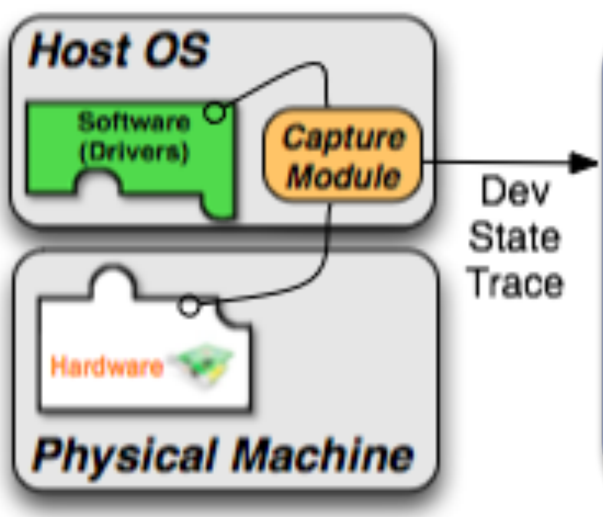
Example: Mishandled Jumbo Frames

- Fuzzed a physical Intel E1000 device
- Captured some of the device MMIO registers during the fuzz
  - E.g. GPRC, RLEC
- Replayed the I/O events (traces) on the QEMU virtual device
- No immediate crash
- However, detected the following inconsistencies
  - **RLEC** @ 0x04040 – Receive Length Error Count
  - **PRC** @ 0x0405C – Packets Received ([64-1522] Bytes) Count
  - **BPRC** @ 0x04078 – Broadcast Packets Received Count
  - **MPRC** @ 0x0407C – Multicast Packets Received Count
  - **GPRC** @ 0x04074 – Good Packet Received Count

What can be observed/compared

- Full visibility of virtual devices
- But limited visibility of physical devices

- Observed by Capturing Traces

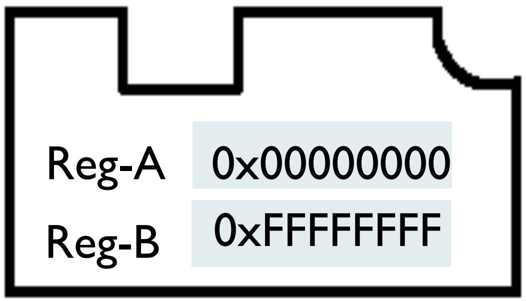


Trace:

Event: mmio_write(reg, value)
Device State: $[R_1,R_2, \ldots, R_N]$
Event: mmio_read(reg)
Device State: $[R_1,R_2, \ldots, R_N]$
… …
Event: mmio_write(reg, value)
Device States: $[R_1,R_2, \ldots, R_N]$
… …

Conformance Testing (Ideally)

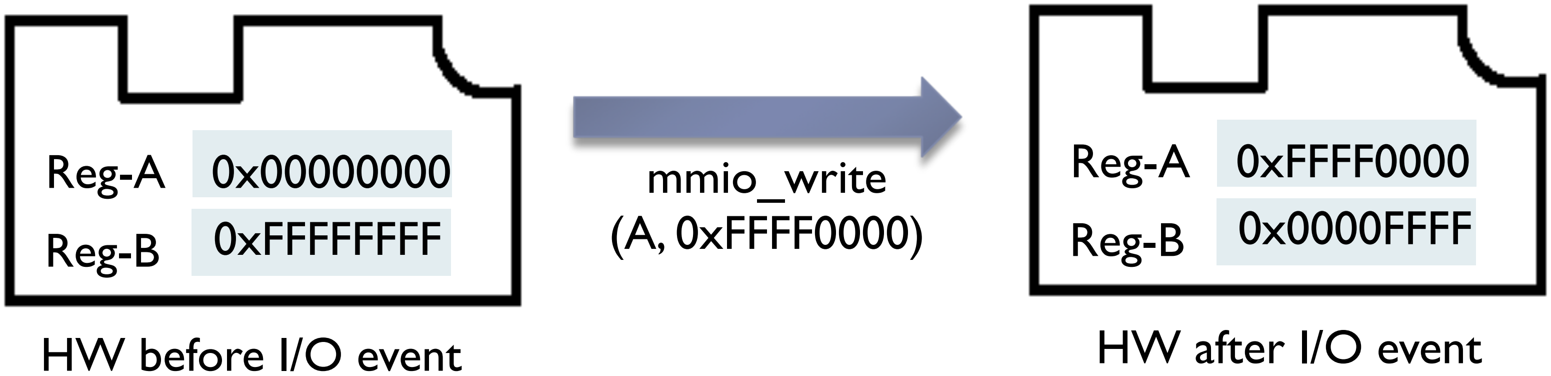
Spec: Reg-A is a mask register for Reg-B.
An update to A causes B to change to $V_B\&{\sim}V_A$



HW before I/O event

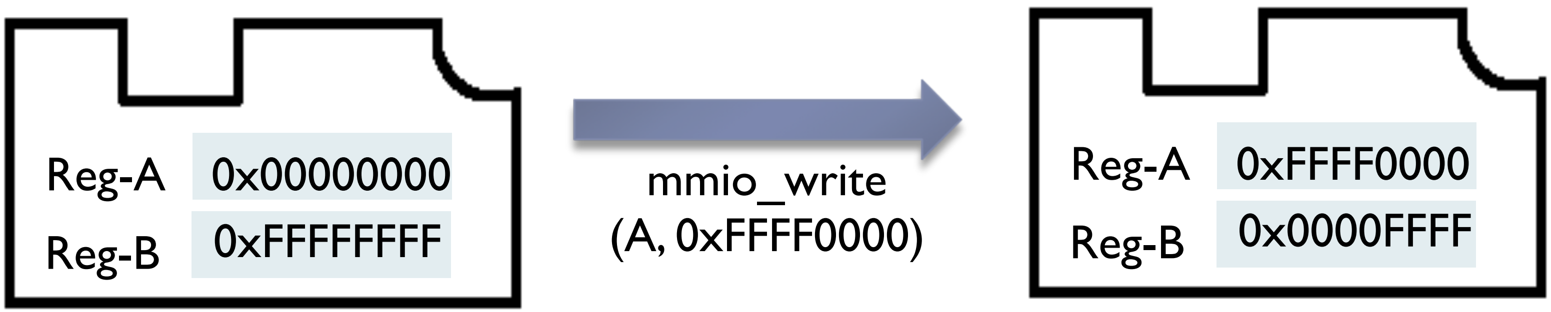
Conformance Testing (Ideally)

Spec: Reg-A is a mask register for Reg-B.
An update to A causes B to change to $V_B\&\sim V_A$



Conformance Testing (Ideally)

Spec: Reg-A is a mask register for Reg-B.
An update to A causes B to change to $V_B\&{\sim}V_A$



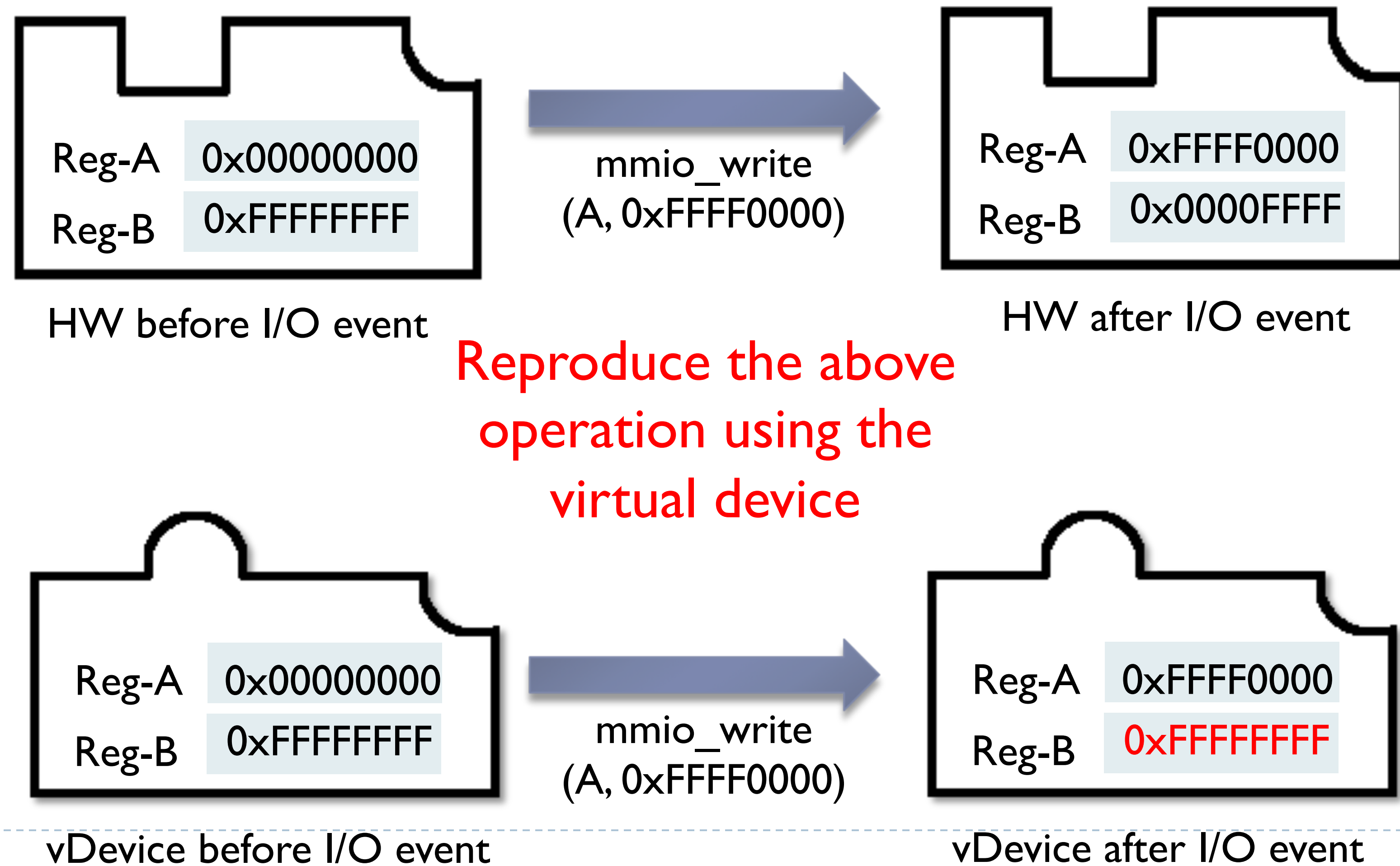
HW before I/O event

HW after I/O event

Reproduce the above operation using the virtual device

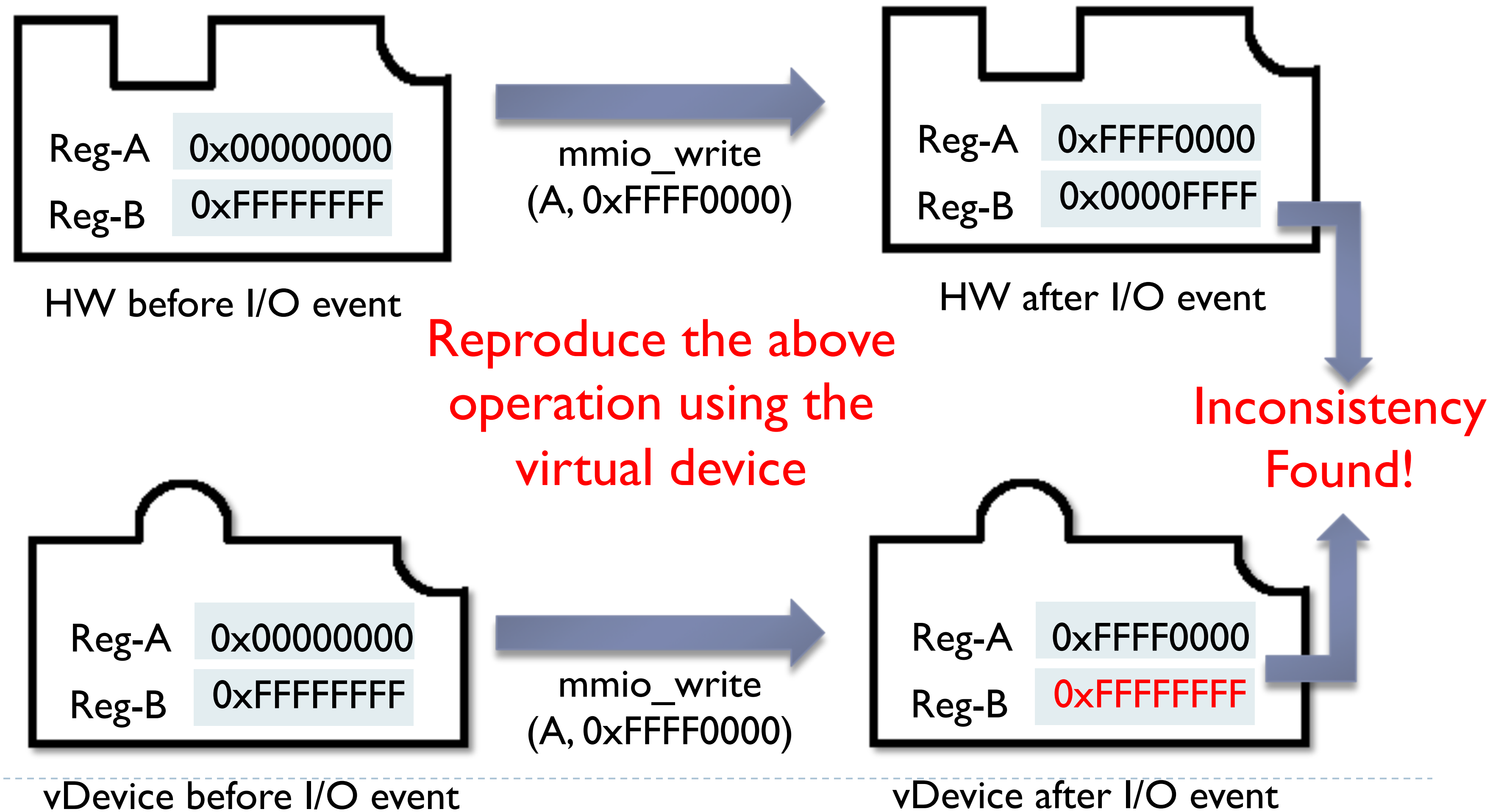
Conformance Testing (Ideally)

Spec: Reg-A is a mask register for Reg-B.
An update to A causes B to change to $V_B \& {\sim} V_A$



Conformance Testing (Ideally)

Spec: Reg-A is a mask register for Reg-B.
An update to A causes B to change to $V_B \& {\sim} V_A$



Conformance Testing (In Reality)

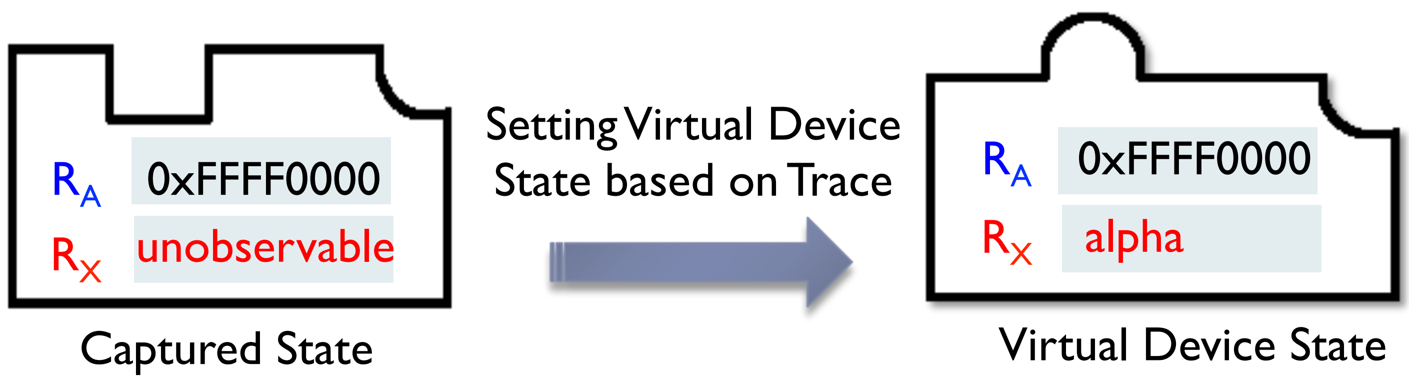
- Dump and replay only works in simple cases
  - Not all physical registers are accessible (readable)
  - Some events are difficult to observe or expensive to capture
    - E.g. DMA
  - Some registers are accessible, but have side effects
    - E.g., clear-on-read

Symbolic Conformance Testing

- Symbolic execution is getting popular for bug finding
  - KLEE symbolic execution engine
    - http://klee.llvm.org/
    - OSDI Paper from Cristian Cadar, Daniel Dunbar, and Dawson Engler
  - Users of symbolic execution
    - Coverity, ForAllSecure, and bugchecker.net …
- Our approach to deal with unobservable device state
  - Construct the virtual device state by setting
    - observable register values based on the trace
    - missing registers with symbolic values

Symbolic Register Values

- Example:
  - For a dumb device with only 2 registers:
    - R_A (observable) and R_X (unobservable)
  - The device state in a trace looks like this: [R_A== 0xFFFF0000]



How to Run with Symbolic Values?

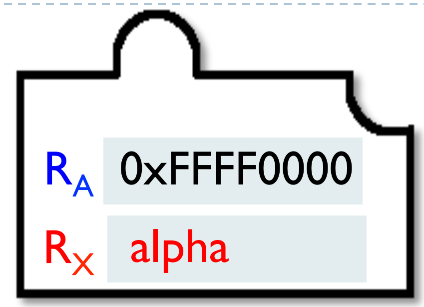
- Consider the following virtual device program:

Virtual Device Code Snippet

```
…
mmio_write_update_RA (value)
{
	if (RX == 0)
		RA = value;
	else
		RA  = value & 0x0000FFFF;
}
```

How to Run with Symbolic Values?

- Consider the following virtual device program:



Virtual Device State

+ Event I:
write (R_A, 0xC0FFEE)

Virtual Device Code Snippet

…
mmio_write_update_RA (value)
{
    if (R_X == 0)
        R_A = value;
    else
        R_A = value & 0x0000FFFF;
}

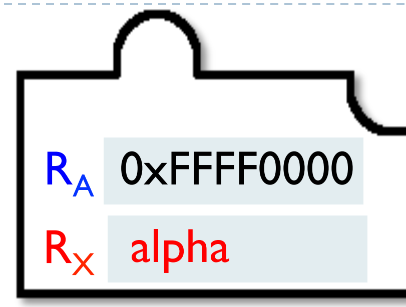
Suppose we have the above initial state and a given event …

How to Run with Symbolic Values?

- Consider the following virtual device program:

Virtual Device Code Snippet

```
 …
mmio_write_update_RA (value)
{
        if (RX == 0)
                RA = value;
        else
                RA  = value & 0x0000FFFF;
}
```



Virtual Device State

+ **Event I:**
write (R_A, 0xC0FFEE)

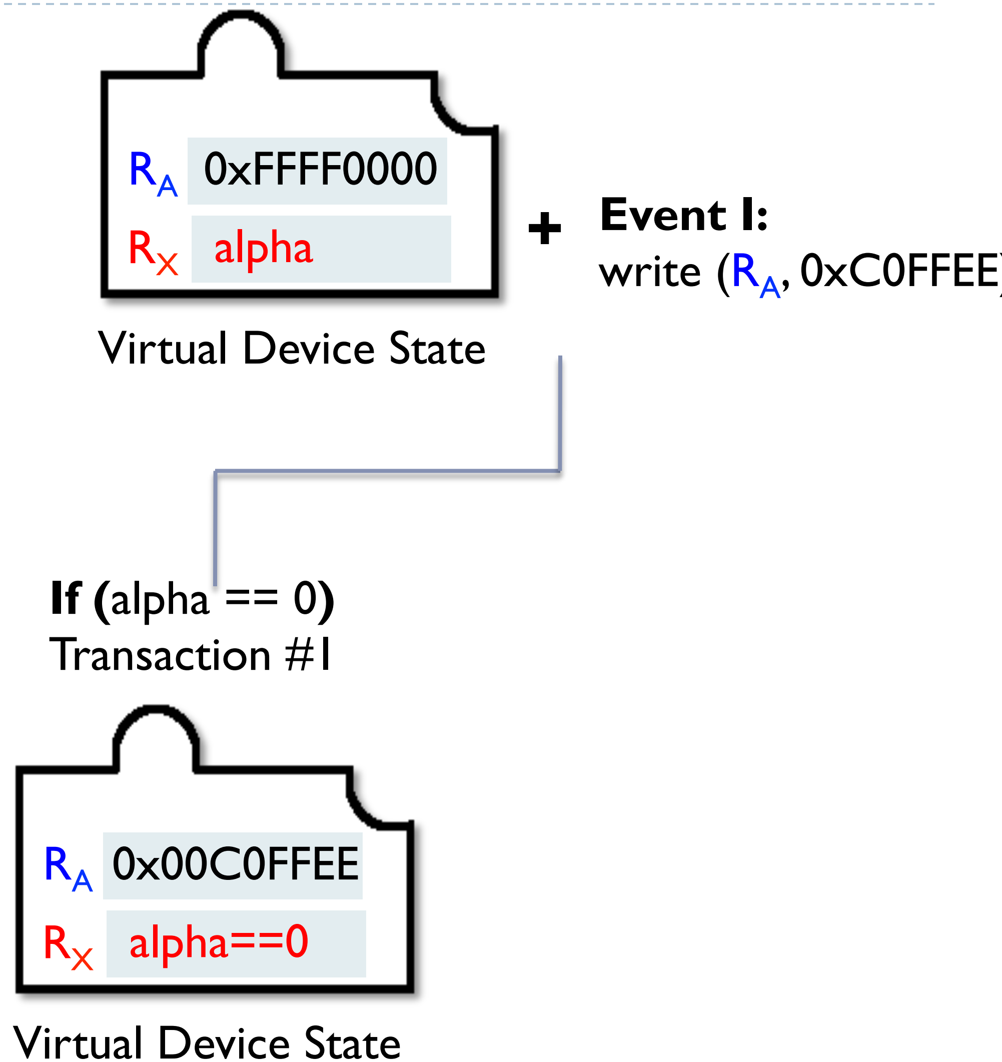
What will the virtual device state be after Event I?

Symbolic Execution

- Consider the following virtual device program:

Virtual Device Code Snippet

```
…
mmio_write_update_RA (value)
{
        if (R_X == 0)
                R_A = value;
        else
                R_A  = value & 0x0000FFFF;
}
```

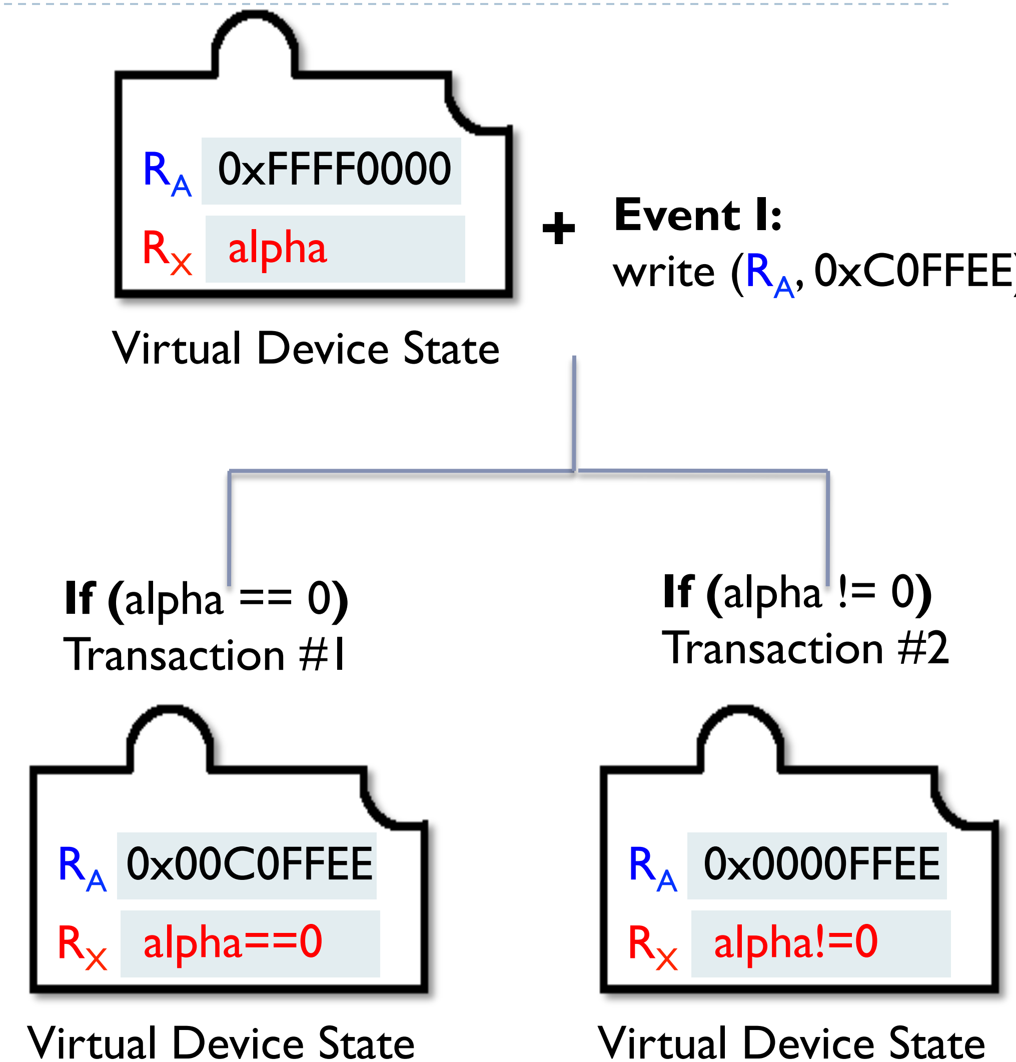


Symbolic Execution

- Consider the following virtual device program:

Virtual Device Code Snippet

```
…
mmio_write_update_RA (value)
{
        if (RX == 0)
                RA = value;
        else
                RA  = value & 0x0000FFFF;
}
```



Searching for Inconsistencies

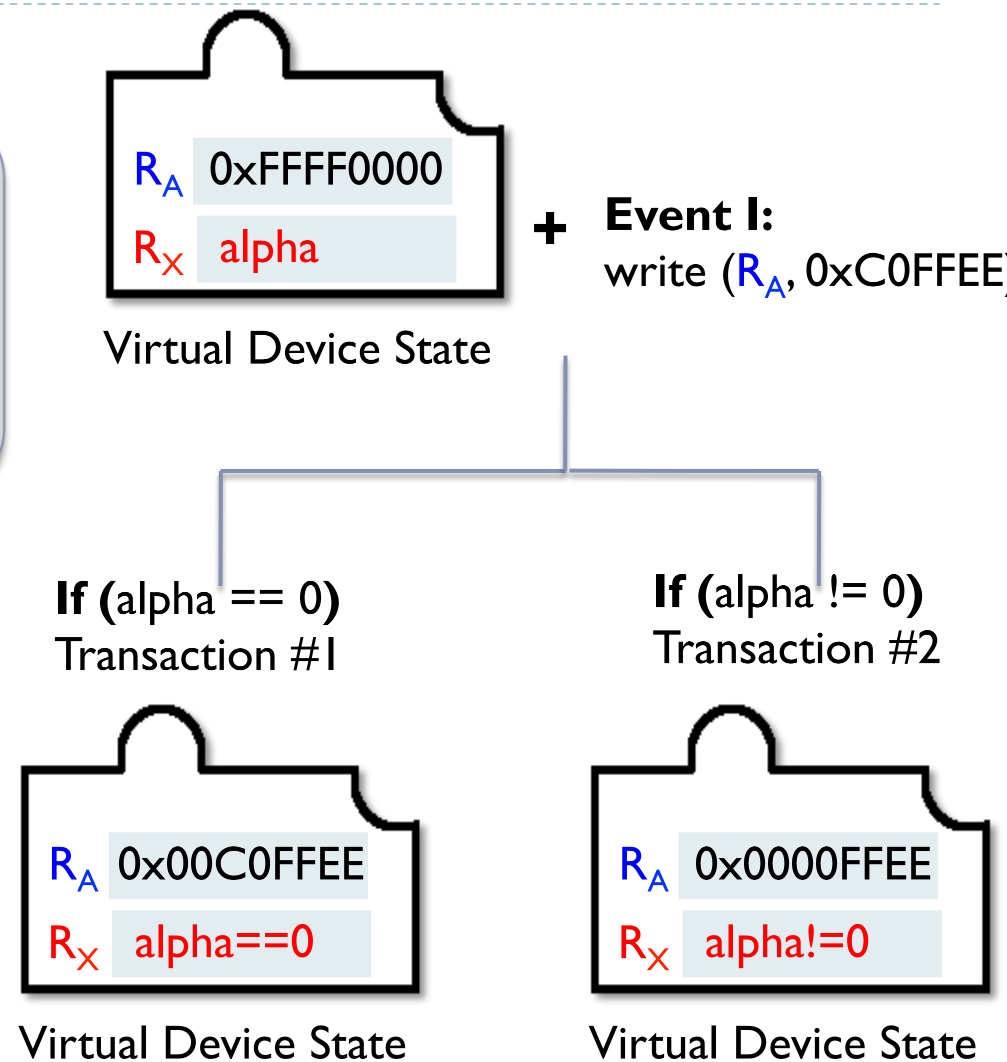
Given this Captured Trace:

…
Device State: [R_A== 0xFFFF0000]

Event 1: mmio_write (R_A, 0xC0FFEE)
Device State: [R_A== 0xFFEE]

…

- Does one of the output virtual device states match the captured device state?



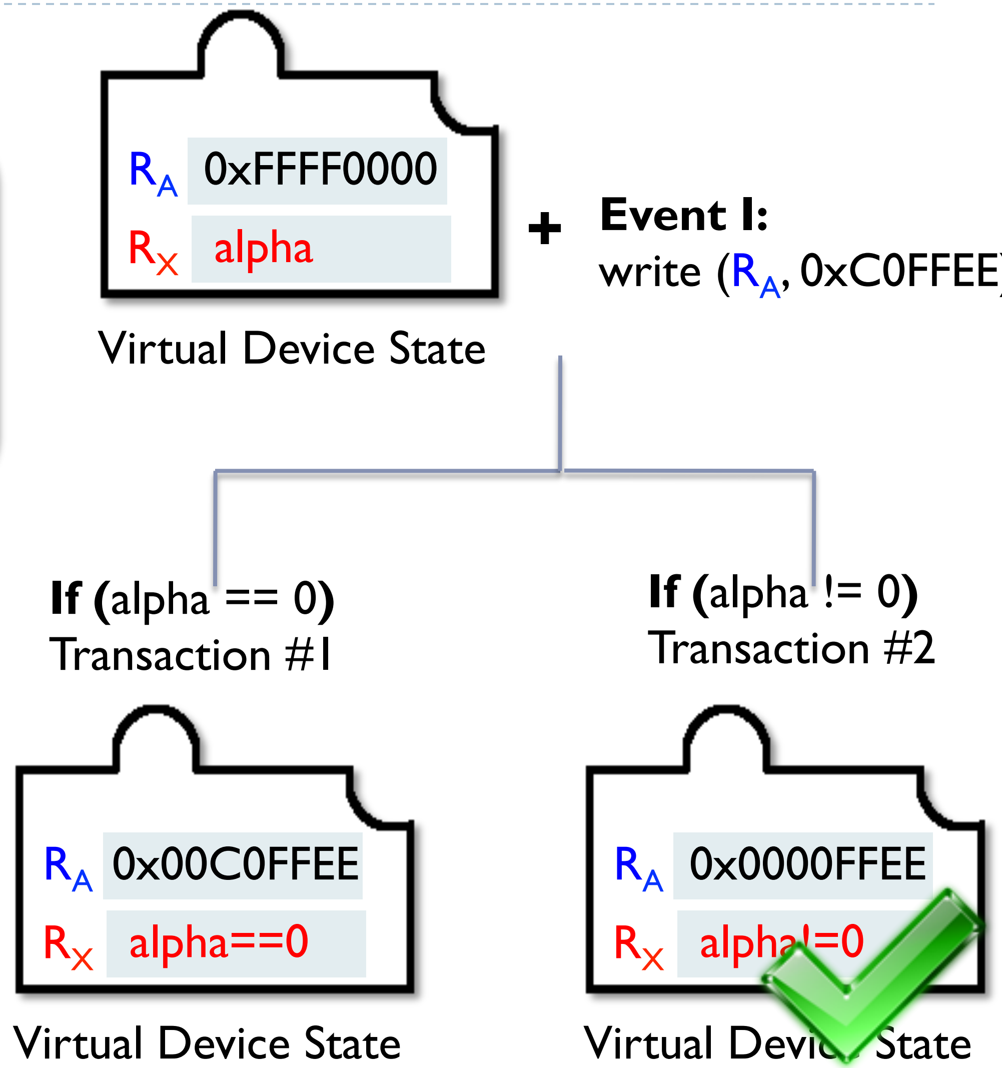
Searching for Inconsistencies

Given this Captured Trace:

...
Device State: [R_A== 0xFFFF0000]

Event I: mmio_write (R_A, 0xC0FFEE)
Device State: [R_A== 0xFFEE]

...



- Found a match, continue with the Transaction.
- If multiple matches found, follow each one.

Searching for Inconsistencies (cont.)

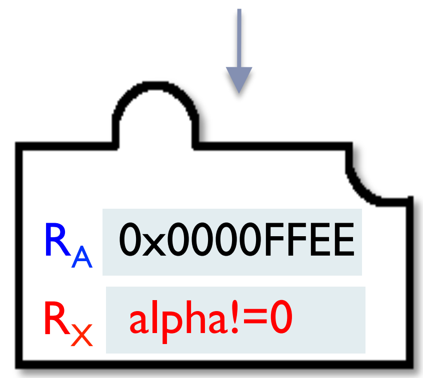
Given this Captured Trace:

...
Device State: [R_A== 0xFFFF0000]

Event I: mmio_write (R_A, 0xC0FFEE)
Device State: [R_A== 0xFFEE]

Event II: mmio_write (R_A, 0x00BEFF00)
Device State: [R_A== 0xBEFF00]
...

Follow from previous transaction



Virtual Device State

+ Event II:
write (R_A, 0x00BEFF00)

- Checking a trace with consecutive events

Searching for Inconsistencies (cont.)

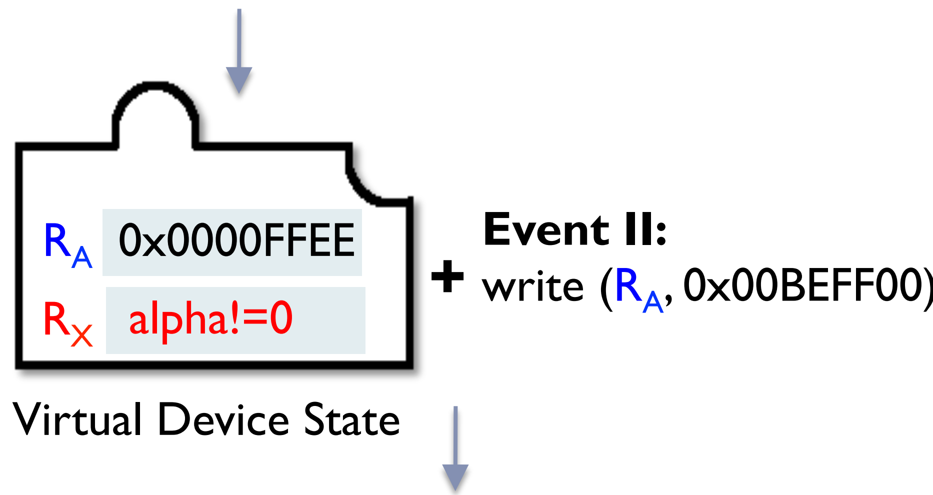
Given this Captured Trace:

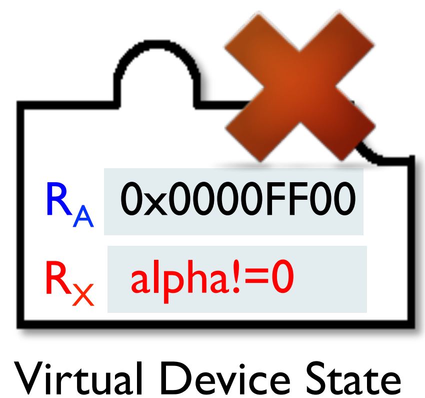
…
Device State: [R_A== 0xFFFF0000]

Event I: mmio_write (R_A, 0xC0FFEE)
Device State: [R_A== 0xFFEE]

Event II: mmio_write (R_A, 0x00BEFF00)
Device State: [R_A== 0xBEFF00]
…

Follow from previous transaction





- Checking a trace with consecutive events
- No candidate match ➔ Inconsistency Found!

Summary

- Conformance Testing 
  - A unique way to test virtual device implementations.
- Symbolic Execution helps to deal with partial states.
- QEMU bugs can be detected in this way.
  - Many inconsistencies found in EEPRO100, E1000, and other virtual devices.