FirmGuide: Boosting the Capability of Rehosting Embedded Linux Kernels through Model-Guided Kernel Execution

Qiang Liu[1*] Cen Zhang[2*] Lin Ma[1] Muhui Jiang[1,3] Yajin Zhou[1] Lei Wu[1] Wenbo Shen[1] Xiapu Luo[3] Yang Liu[2] Kui Ren[1]

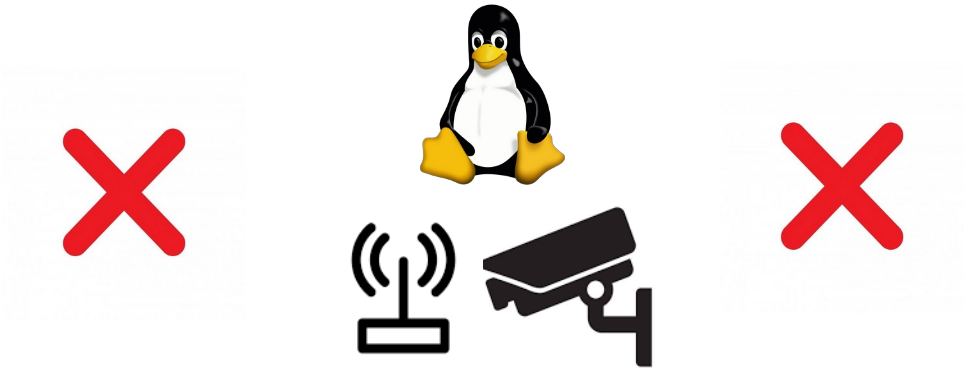
[1]Zhejiang University [2]Nanyang Technological University [3]The Hong Kong Polytechnic University

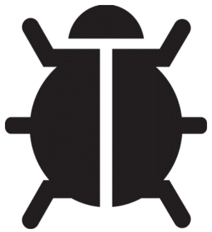
[*]The first two authors contributed equally to this work.

ASE2021

Motivation

Dynamic Bug or Vulnerability Understanding





Dynamic Bug or Vulnerability Mining

- Linux kernel with drivers inside high-end embedded firmware
- Understanding and testing abilities not easily and scalably due to hardware requirement
- **Rehosting the embedded Linux kernel with the best effort**

Challenge and Observation 1

SoC: plxtech,nas782x	
CPU	Arm11MPCore
Memory	up to 512M
Interrupt Controller	gic
Time-related	rps, oscillator, sysclk, plla, pllb, stdclk, twdclk
UART	ns16550a
Others	gmacclk, pcie, watchdog, sata, nand, ethernet, ehci, leds

High fidelity to make the Linux kernel functional-correct

Low fidelity for successful boot

High-fidelity Virtual Device

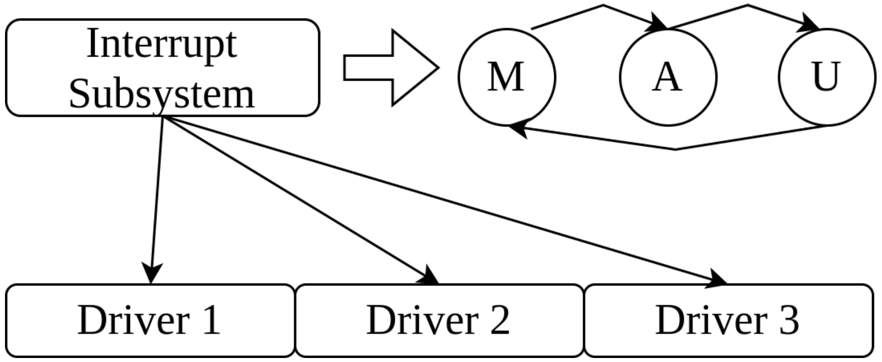
Dummy Virtual Device

- Numerous peripherals: Type-I High Fidelity Type-II Low Fidelity
- **Classifying peripherals for a minimum best effort**

Challenge and Observation 2

Multiple models for interrupt controllers

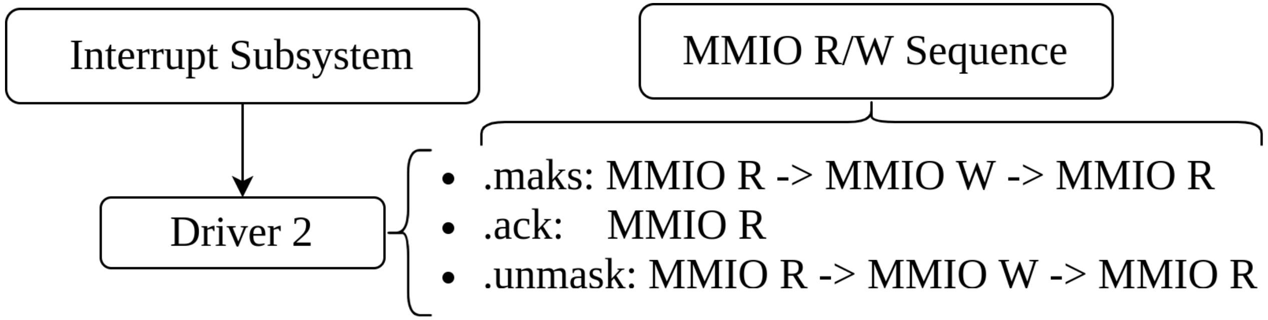
ralink-rt2880-intc
qca,ar7240-intc
marvell,orion-intc
marvell,orion-bridge-intc
arm,cortex-a9-gic
…



- ***Diverse models***: Linux subsystems that hide implementation details
- **Extracting state machines from the Linux subsystems (Type-I)**

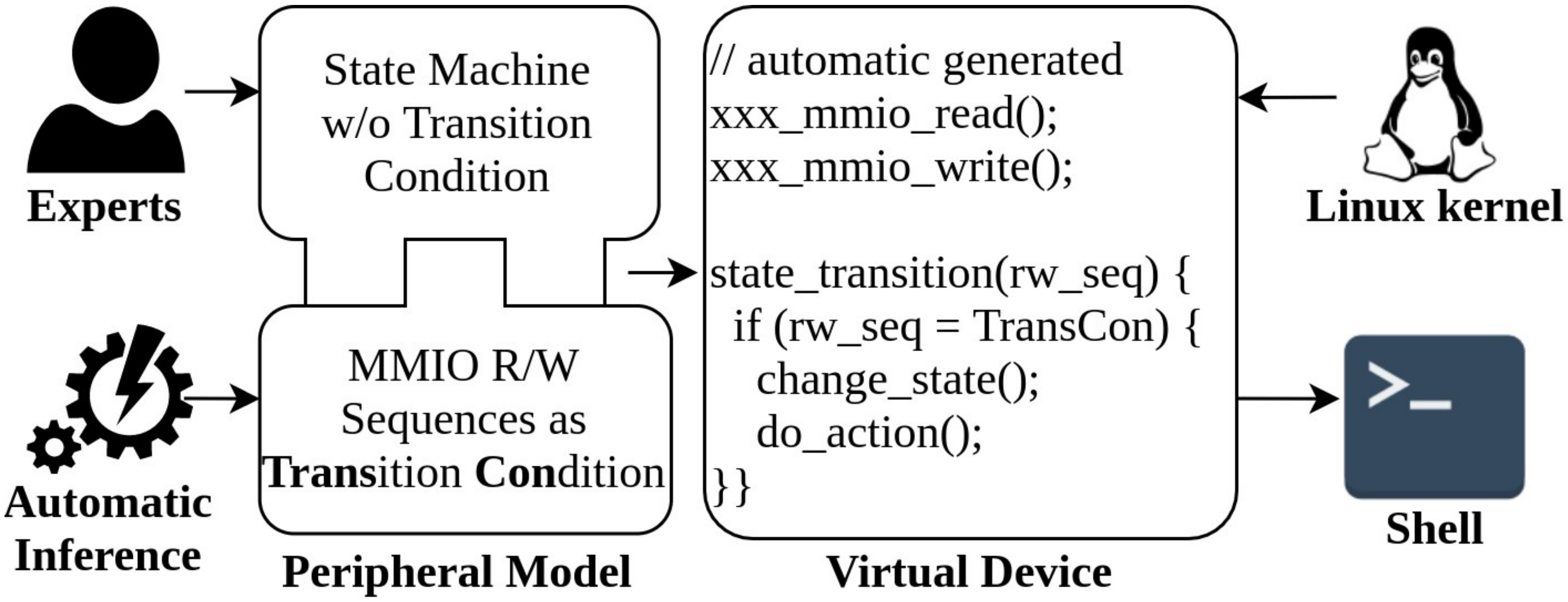
Challenge and Observation 3

Mask Interrupt
MMIO Read M -> a
a &= flags
MMIO Write a -> M
Load IRQ number
MMIO Read I -> b
switch(b)
…



- *Complex semantics*: Specific driver interface callbacks that embed complex semantics
- **Extracting MMIO R/W sequences**

Core Technique: Model-guided Kernel Execution



- Peripheral model = the model template (a state machine) + the model parameters (MMIO R/W sequences as transition conditions)

Model-guided Kernel Execution: Running Example

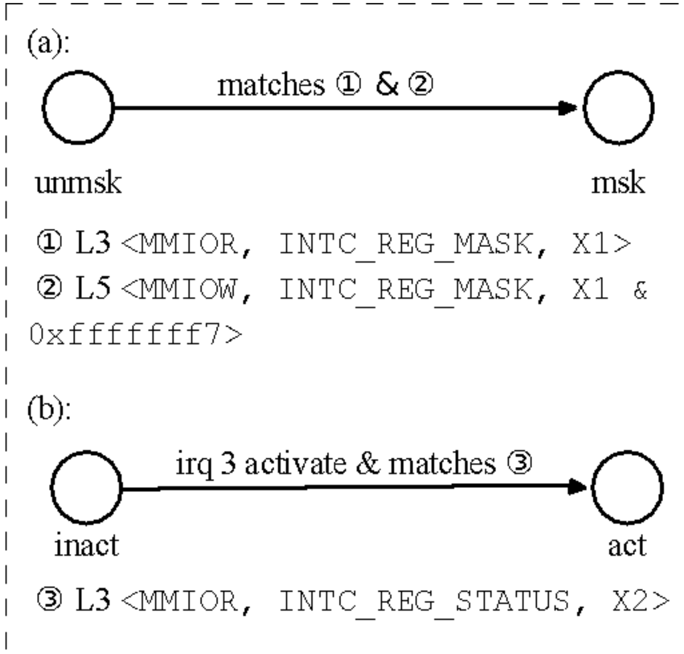
```
1 static void irq_mask_callback(u32 irq)
2 {
3   u32 mask = readl(INTC_REG_MASK);
4   mask &= ~(1 << (irq & 0x1f))
5   writel(mask, INTC_REG_MASK);
6 }
```

(a)

```
1 static void handle_irq_callback(...)
2 {
3   u32 pending = readl(INTC_REG_STATUS);
4   while(pending) {
5     u32 irq = __ffs(pending);
6     generic_handle_irq(irq);
7     pending |= ^(1 << irq);
8   }
9 }
```

Linux kernel driver code

(b)

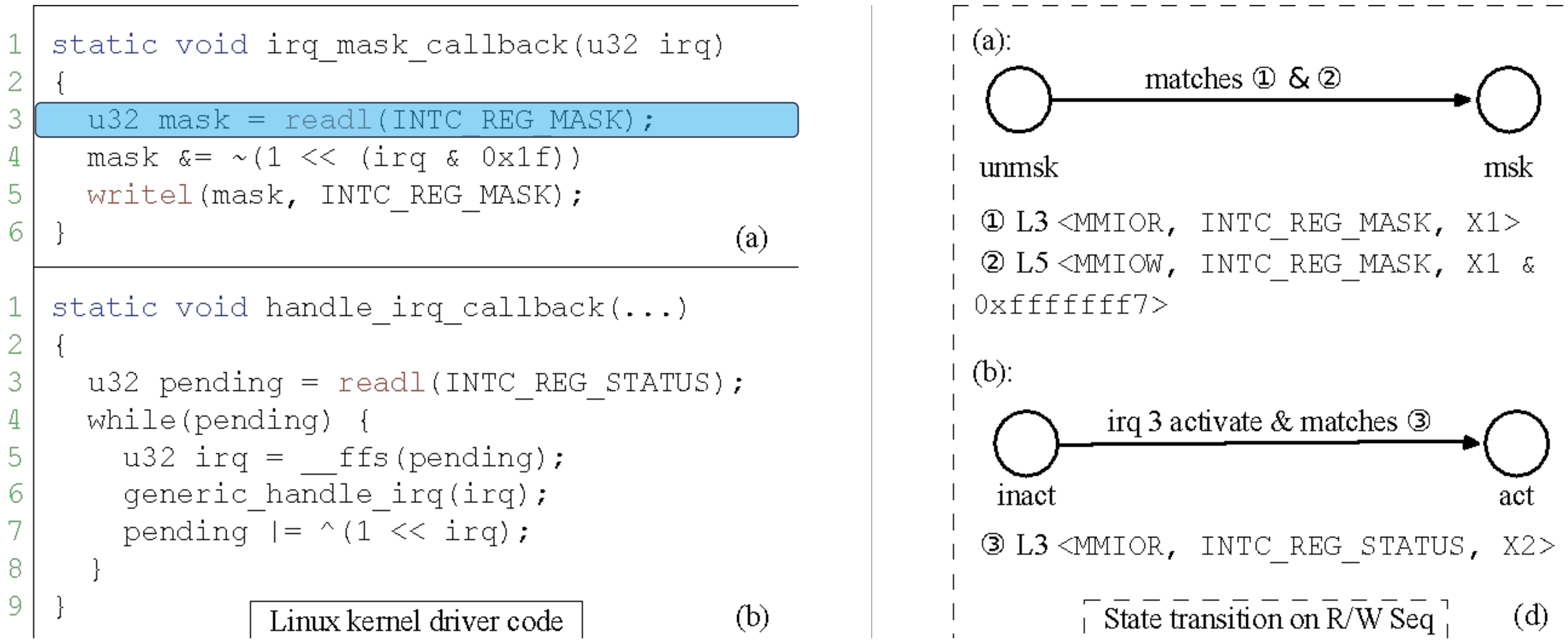


State transition on R/W Seq

(d)

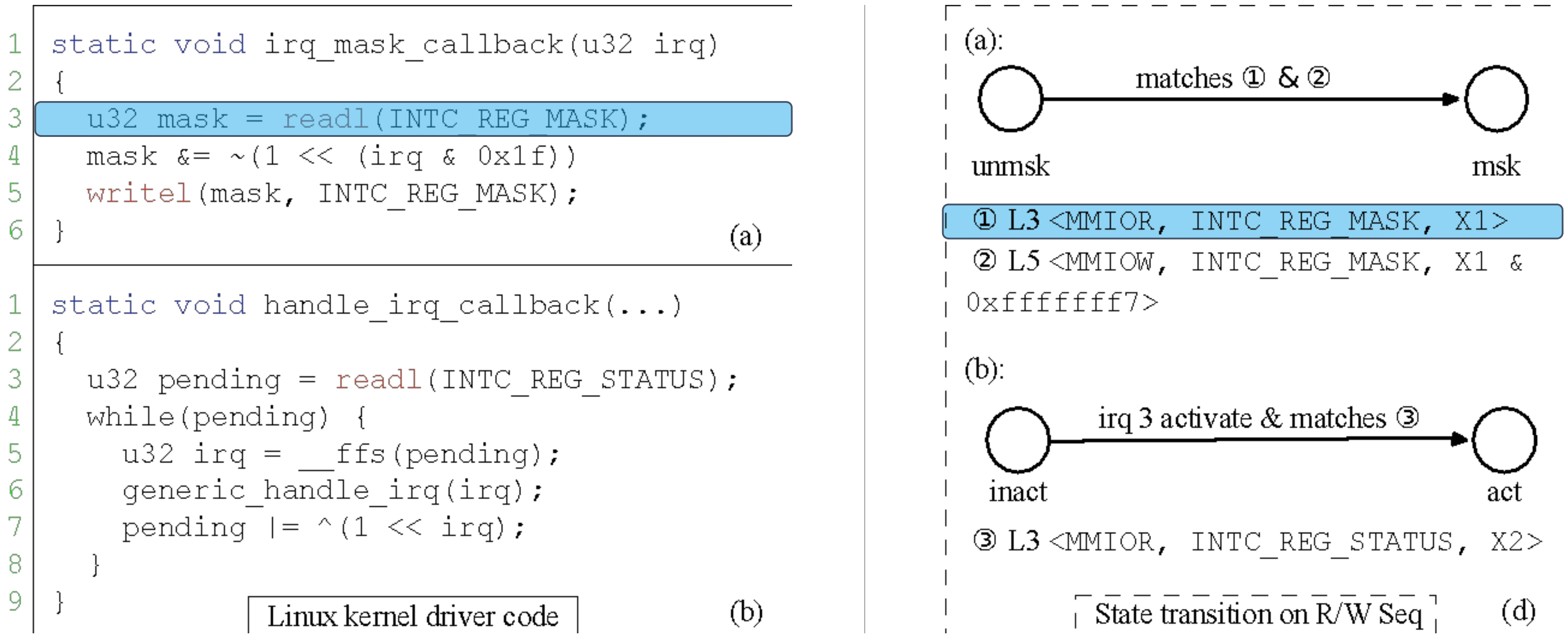
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



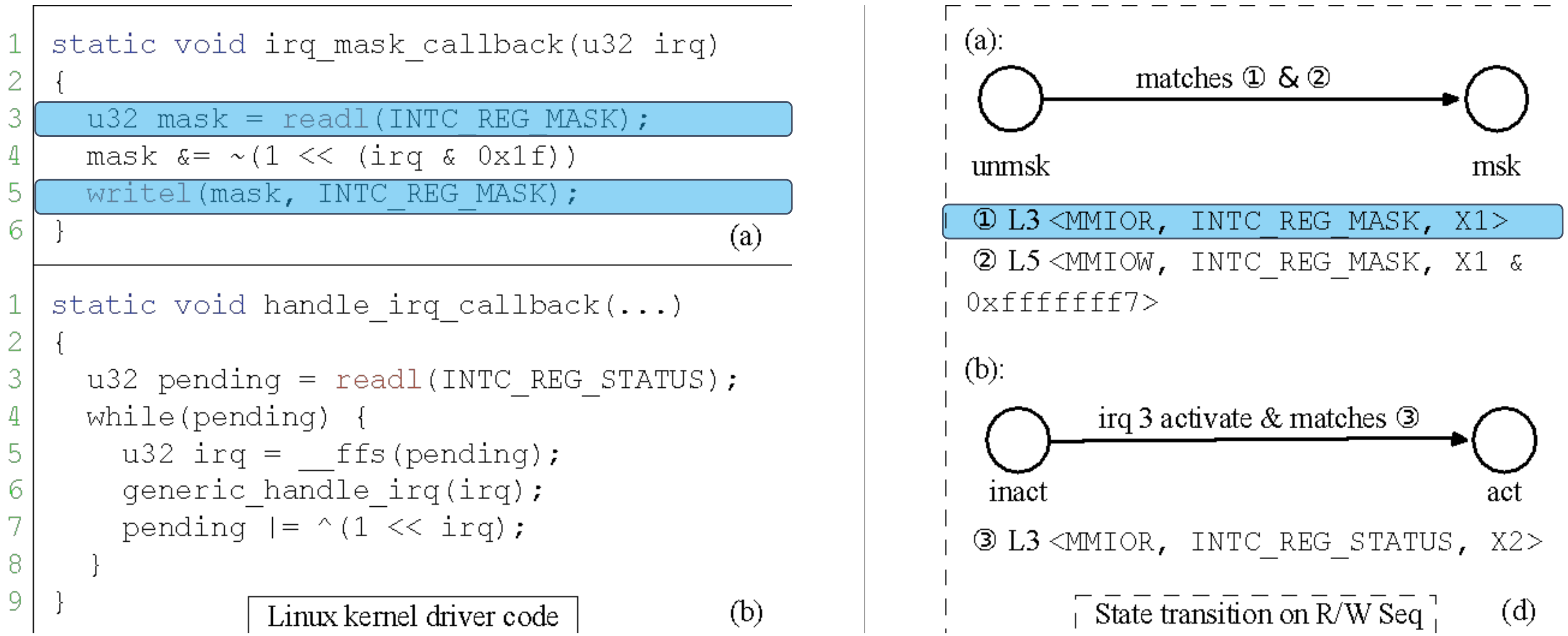
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



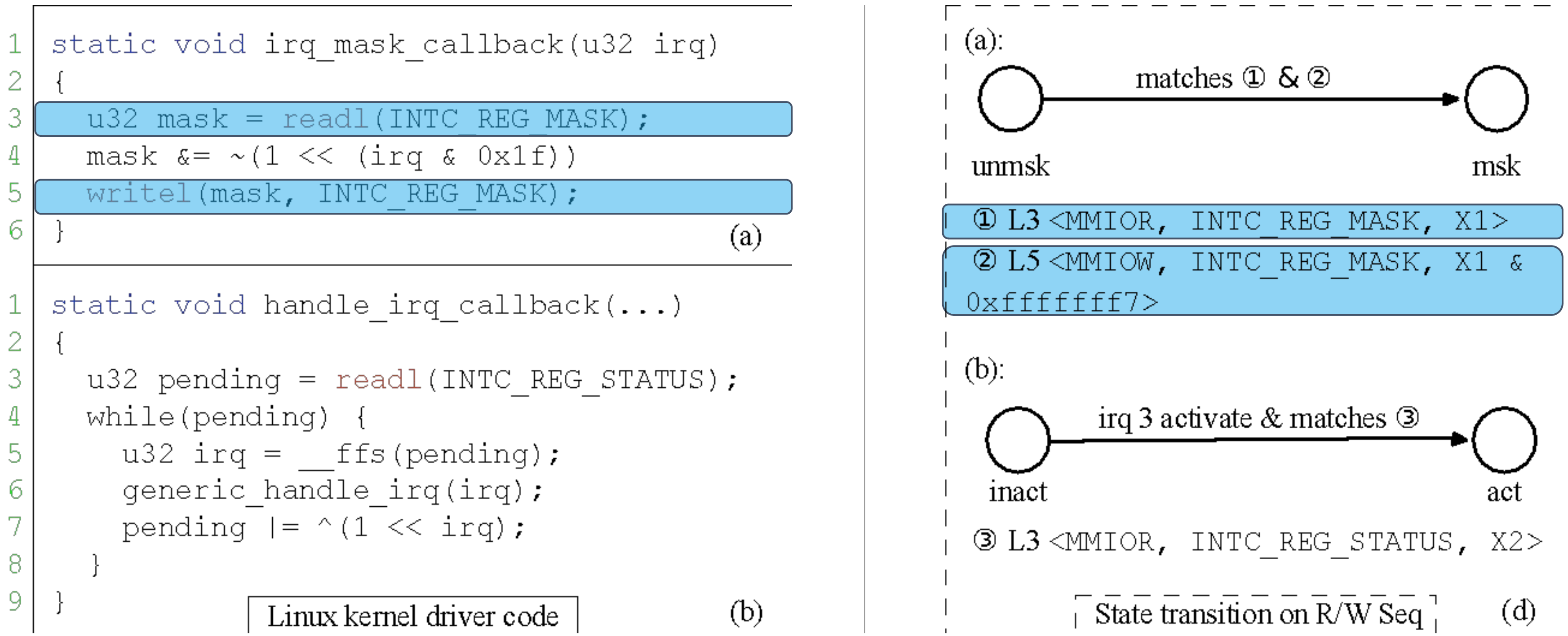
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



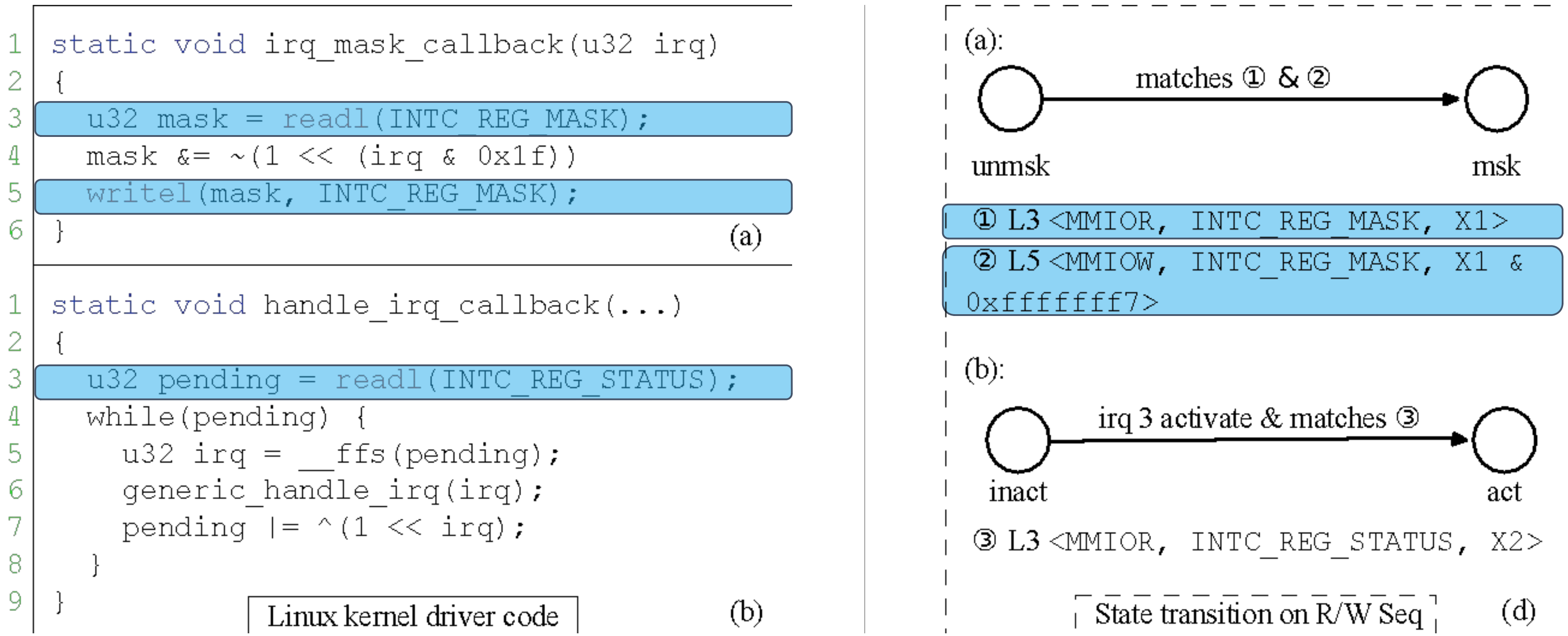
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



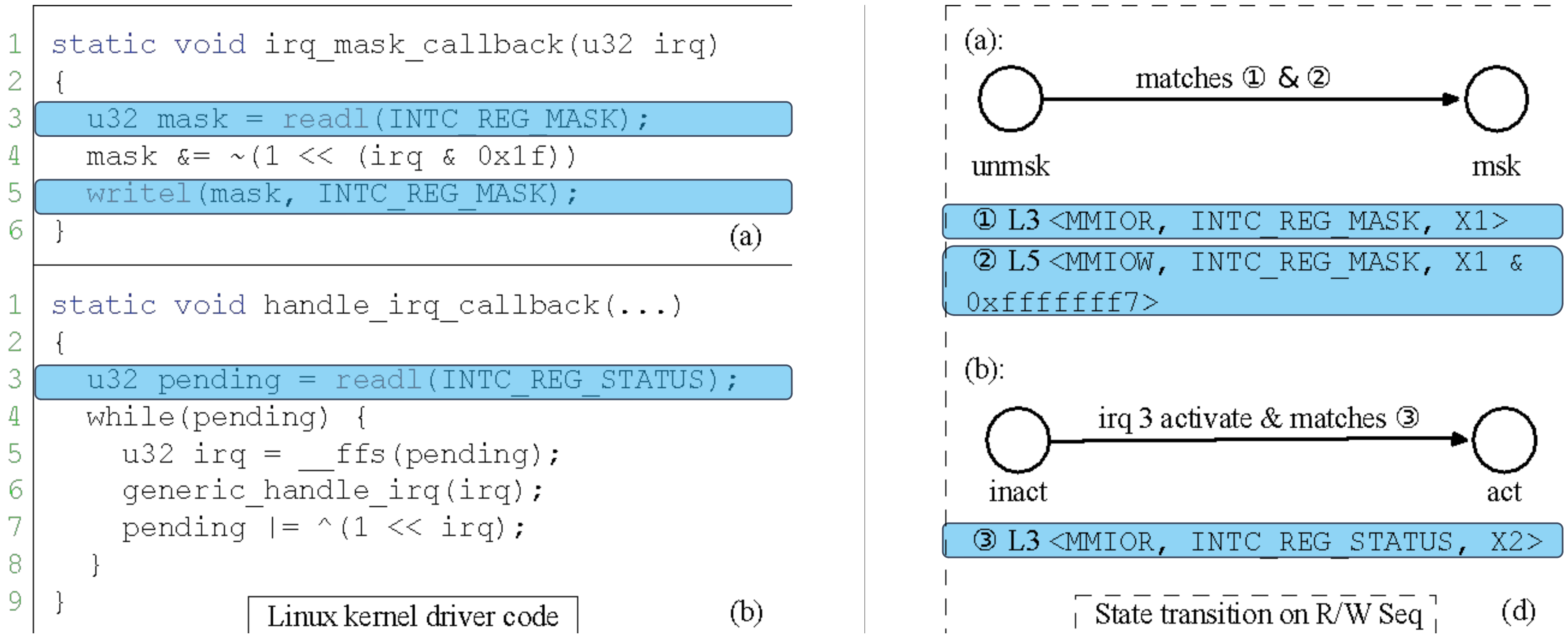
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



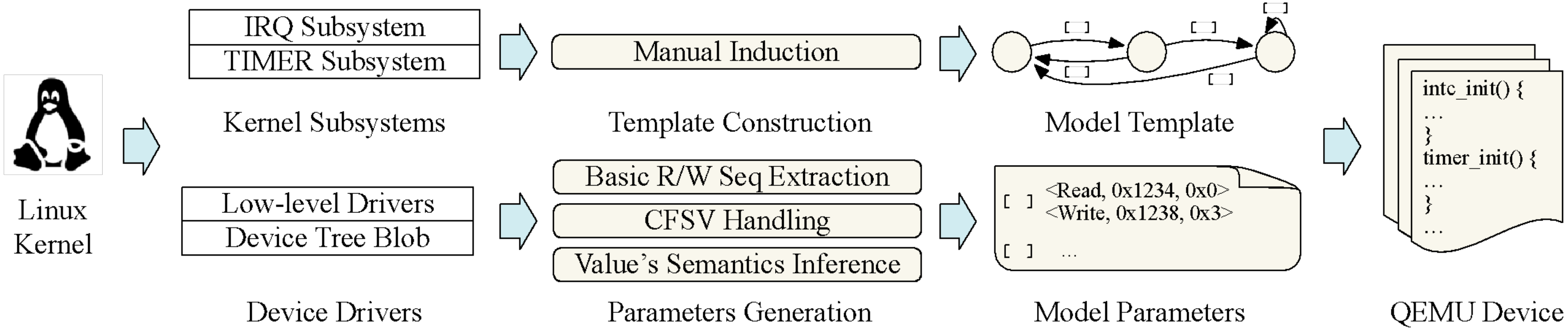
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



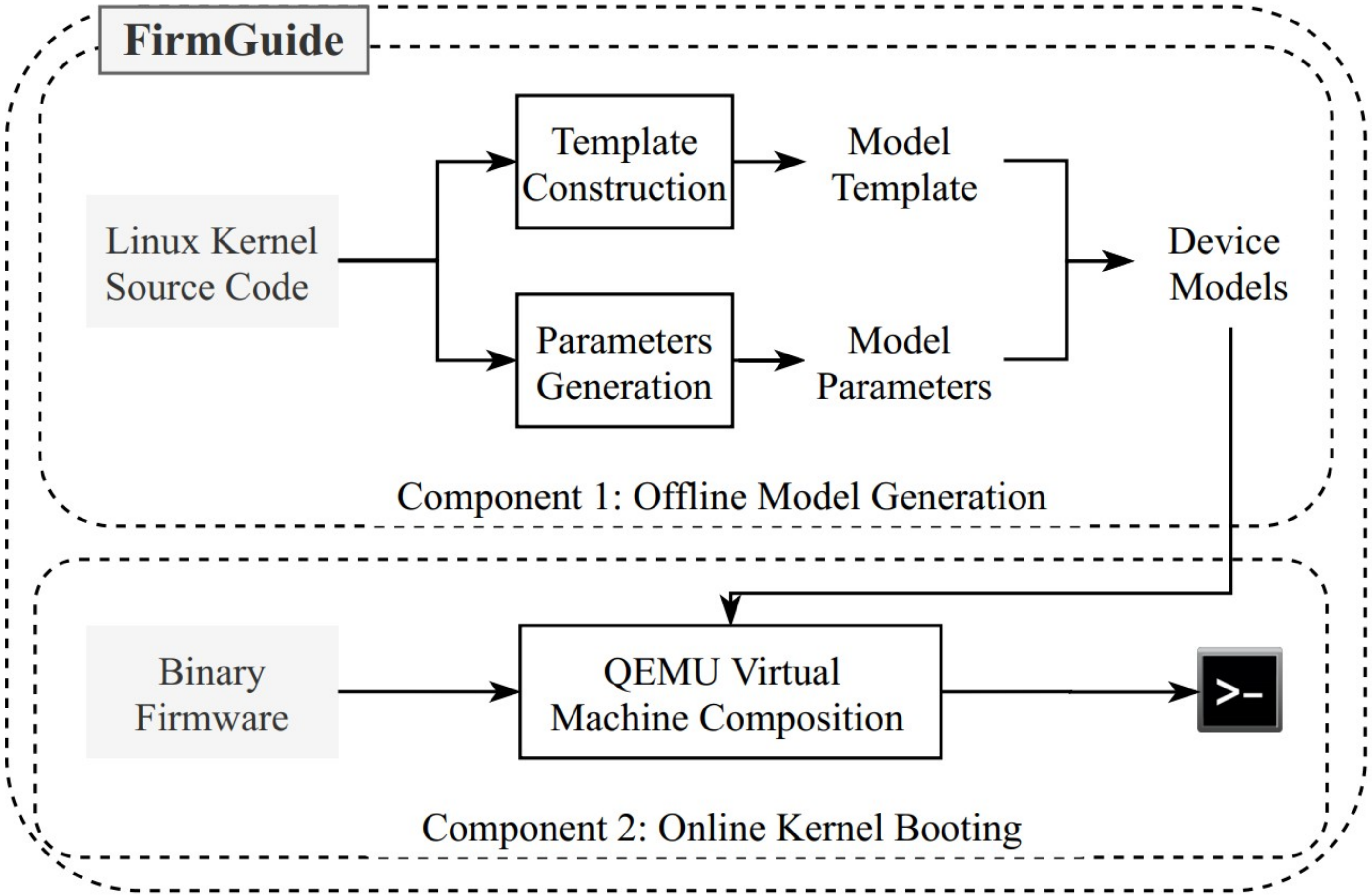
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Methodology



- **We semi-automatically build the state machine of each peripheral: a general model template (manually) plus model parameters (automatically)**

System Design and Implementation



LLVM pass for preprocess
KLEE for MMIO R/W Seq
Python for glues

Python for main logic
Template-render pattern

Evaluation

RQ 1: What peripheral models can we generate?

Type I

Family of SoCs	Interrupt Controller	Timer	First Solution (Second)	Exists CSVF (y/n)	Timer Semantics
ramips/rt305x	ralink-rt2880-intc	not necessary	1	n	-
ath79/generic	qca,ar7240-intc	not necessary	5	n	-
kirkwood/generic	marvell,orion-intc marvell,orion-bridge-intc	marvell,orion-timer	2	y	y=~x
bcm53xx/generic	arm,cortex-a9-gic	arm,cortex-a9-global-timer arm,cortex-a9-twd-timer	2,207	y	y=x1<<32+x2
oxnas/generic	arm,arm11mp-gic	plxtech,nas782x-rps-timer	914	y	y=x

Type II: # of initial values/# of Type II peripherals

Family of SoCs	ramips/rt305x	ath79/generic	kirkwood/generic	bcm53xx/generic	oxnas/generic
count	1/10	2/15	3/26	2/4	2/9

Evaluation

RQ 2: What embedded Linux kernels can we rehost?

Subtarget	Unpack	Kernel	User Space	Shell
ramips/rt3050	4784	4784	4743 (99.14%)	4345 (90.80%)
ath79/generic	541	541	444 (82.07%)	444 (82.07%)
bcm53xx/generic	388	388	388 (100.00%)	388 (100.00%)
kirkwood/generic	330	326	324 (99.39%)	244 (74.85%)
oxnas/generic	149	149	48^ (32.21%)	48^ (32.21 %)
Overall	6192	6188	5947 (96.11%)	5469 (88.38%)

Given 6K+ firmware crossing 10 vendors, 3 architectures, and 22 Linux kernel versions, FirmGuide can successfully rehost more than 96% of them.

^The successful rate to support oxnas/generic is low because it cannot recognize our ramfs due to a unset flag.

Evaluation

RQ 3: What about the functionality of the rehosted embedded Linux kernels?

Linux Test Project: Syscall Testing

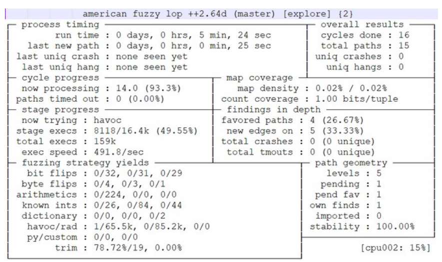
Models	Pass	Skipped	Failed	Total
Fully Generated	1049	164	46	1259
Ground Truth	1049	164	46	1259

RQ 4: What are application of FirmGuide?

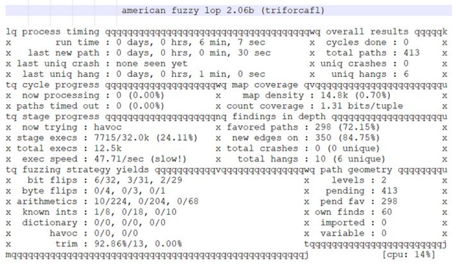
CVE Reproduction and Exploit Development

CVE ID	CVE Type	Triggering	Exploitation
CVE-2016-5195	Race Condition	N	N
CVE-2016-8655	Race Condition	Yes	Y
CVE-2016-9793	Integer Overflow	Y	N
CVE-2017-7038	Integer Overflow	Y	Y
CVD-2017-1000112	Buffer Overflow	Y	Y
CVE-2018-5333	NULL Pointer Dereference	Y	Y

Fuzzing



UnicoreFuzz



TriforceAFL

Summary

Conclusion

- A novel technique “Model-Guided Kernel Execution” for peripheral modeling
- The first semi-automatic framework for embedded Linux kernel rehosting
- Feasible dynamically understanding and mining vulnerability in embedded kernels

Discussion

Limitation and future work

- Manual state machine construction for more complex peripherals
- High fidelity of Type-II peripherals

Q & A

qiangliu@zju.edu.cn, cen001@e.ntu.edu.sg