



浙江大學  
ZHEJIANG UNIVERSITY



NANYANG  
TECHNOLOGICAL  
UNIVERSITY  
SINGAPORE



THE HONG KONG  
POLYTECHNIC UNIVERSITY  
香港理工大學

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

Qiang Liu<sup>1\*</sup> Cen Zhang<sup>2\*</sup> Lin Ma<sup>1</sup> Muhui Jiang<sup>1,3</sup> Yajin Zhou<sup>1</sup> Lei Wu<sup>1</sup> Wenbo Shen<sup>1</sup> Xiapu Luo<sup>3</sup>  
Yang Liu<sup>2</sup> Kui Ren<sup>1</sup>

<sup>1</sup>Zhejiang University <sup>2</sup>Nanyang Technological University <sup>3</sup>The Hong Kong Polytechnic University

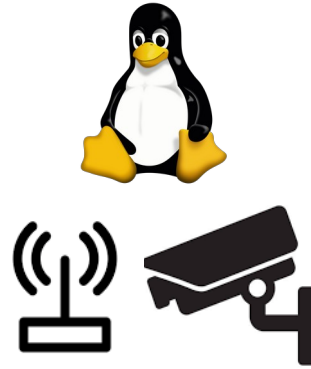
\*The first two authors contributed equally to this work.

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, pll, 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

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

# Challenge and Observation 2

Multiple models for interrupt controllers

ralink-rt2880-intc

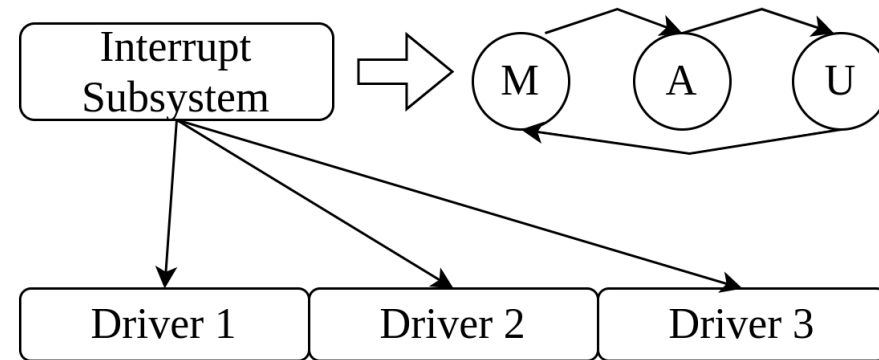
qca,ar7240-intc

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

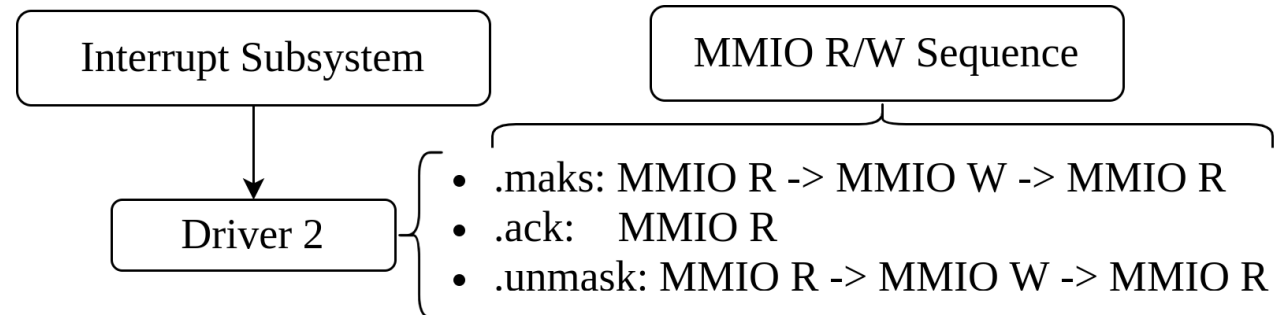
MMIO Write a -> M

Load IRQ number

MMIO Read I -> b

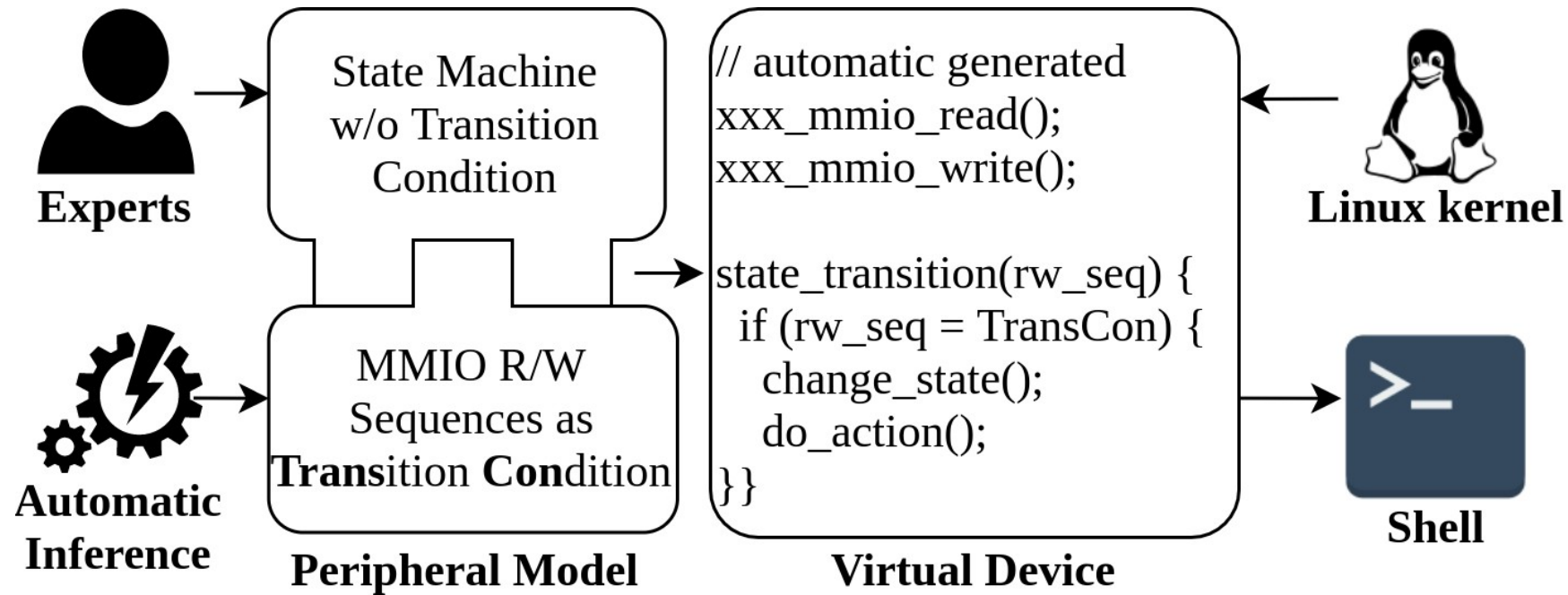
switch(b)

...



- **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

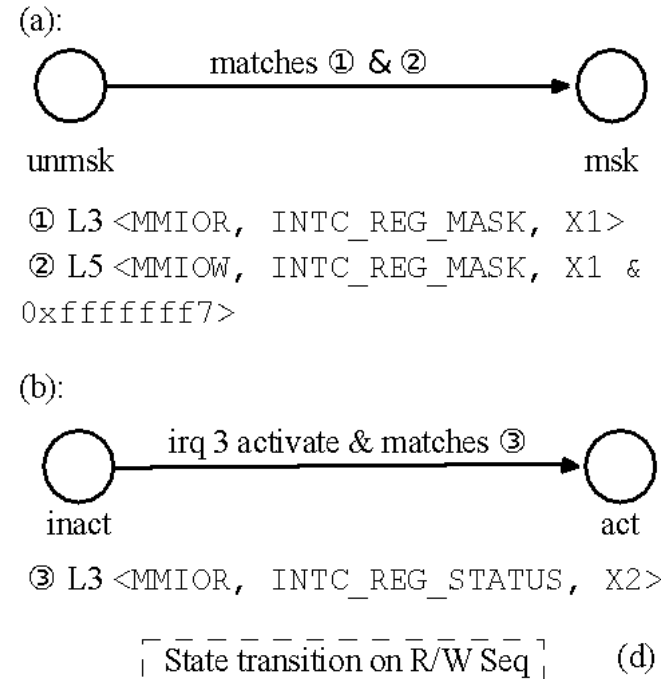
```
1 static void irq_mask_callback(u32 irq)
2 {
3     u32 mask = readl(INTC_REG_MASK);
4     mask &= ~(1 << (irq & 0x1f))
5     writel(mask, INTC_REG_MASK);
6 }
```

(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)



- **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

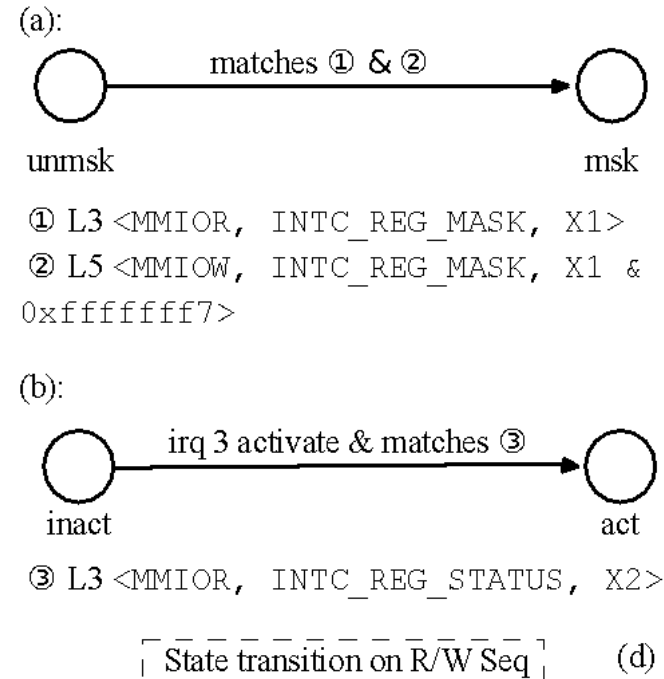
```
1 static void irq_mask_callback(u32 irq)
2 {
3   u32 mask = readl(INTC_REG_MASK);
4   mask &= ~(1 << (irq & 0x1f))
5   writel(mask, INTC_REG_MASK);
6 }
```

(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)



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



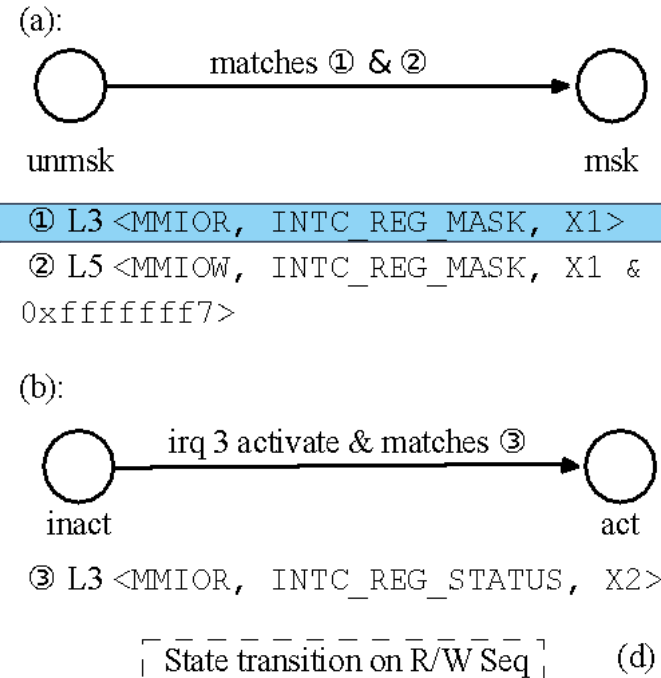
# Model-guided Kernel Execution: Running Example

```
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)



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

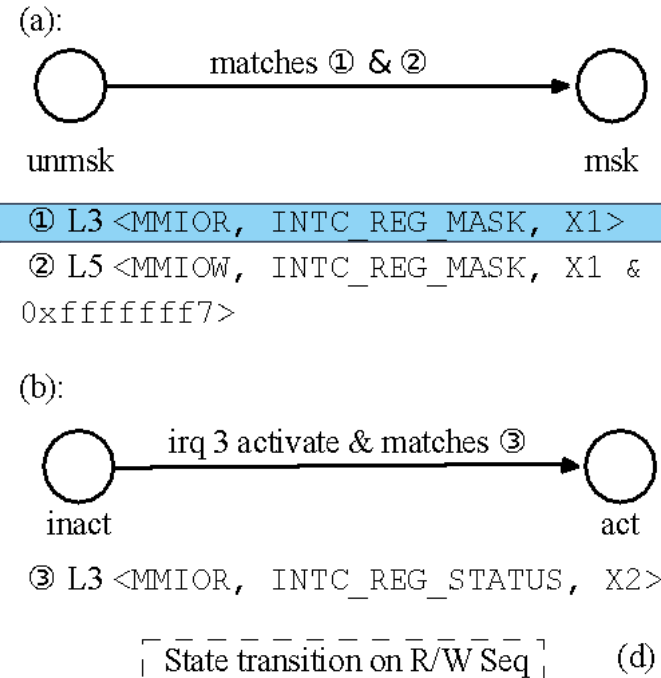
# Model-guided Kernel Execution: Running Example

```
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)



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

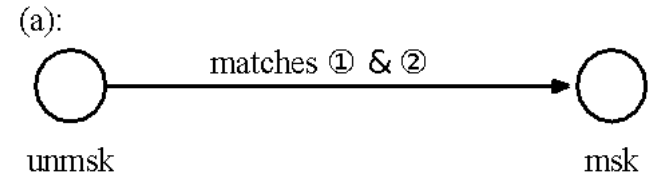
# Model-guided Kernel Execution: Running Example

```
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 }
```

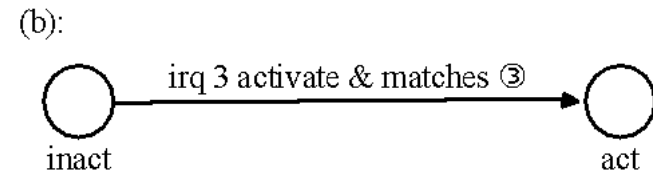
(a)

```
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)



- ① L3 <MMIOR, INTC\_REG\_MASK, X1>
- ② L5 <MMIOW, INTC\_REG\_MASK, X1 & 0xffffffff7>



- ③ L3 <MMIOR, INTC\_REG\_STATUS, X2>
- 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**

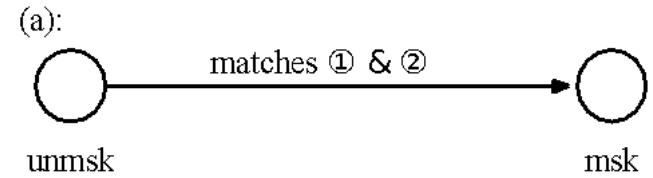
# Model-guided Kernel Execution: Running Example

```
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 }
```

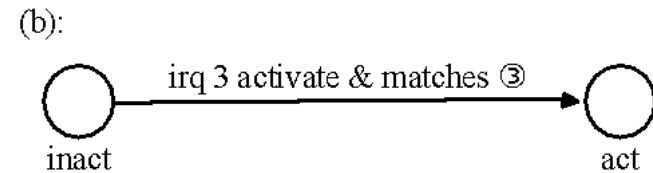
(a)

```
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)



- ① L3 <MMIOR, INTC\_REG\_MASK, X1>
- ② L5 <MMIOW, INTC\_REG\_MASK, X1 & 0xffffffff7>



- ③ L3 <MMIOR, INTC\_REG\_STATUS, X2>

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**

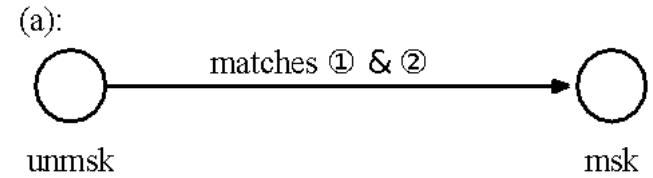
# Model-guided Kernel Execution: Running Example

```
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 }
```

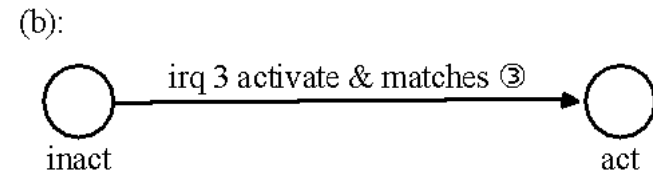
(a)

```
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)



① L3 <MMIOR, INTC\_REG\_MASK, X1>  
② L5 <MMIOW, INTC\_REG\_MASK, X1 & 0xffffffff7>

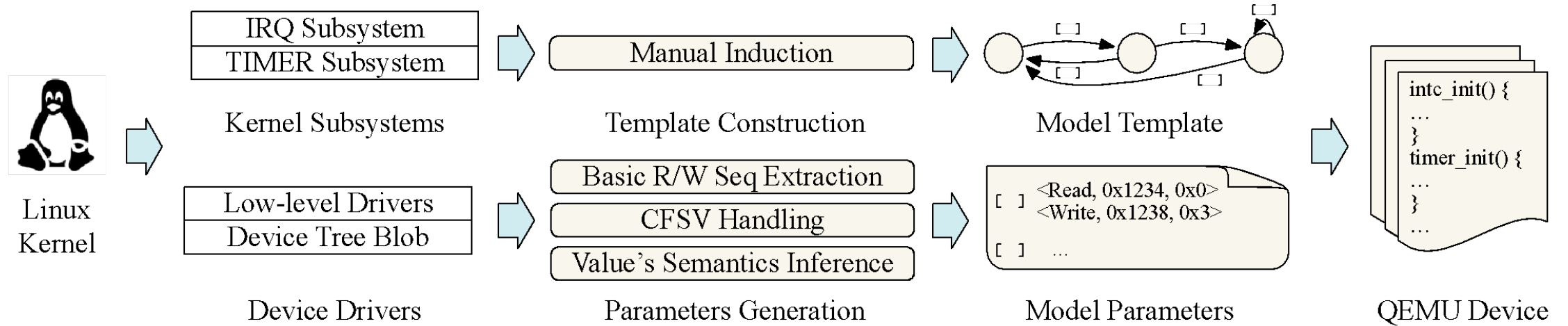


③ L3 <MMIOR, INTC\_REG\_STATUS, X2>

State transition on R/W Seq (d)

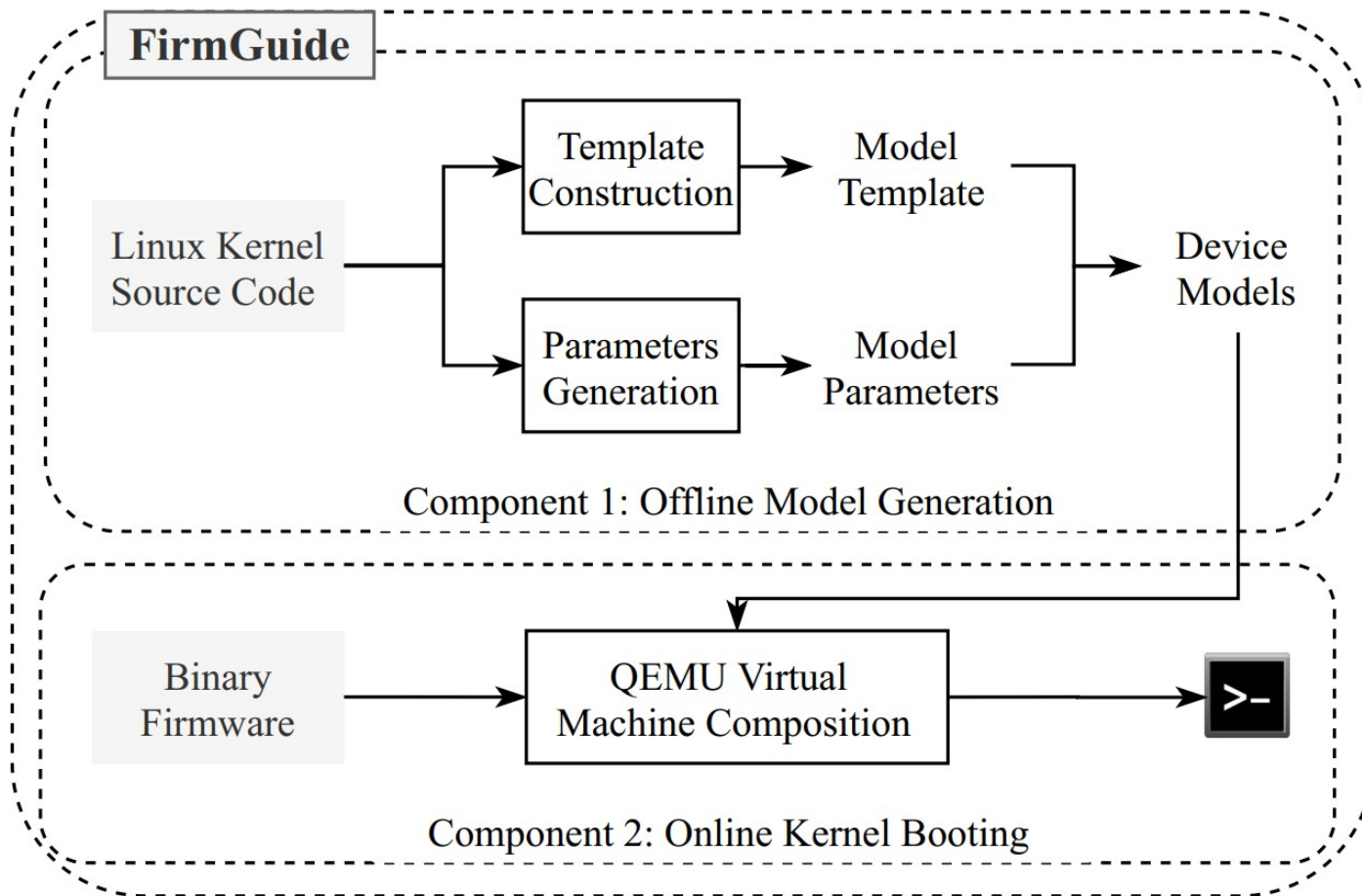
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

# Model-guided Kernel Execution: 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	$y=\sim x$
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

```
american fuzzy lop ++2.64d (master) [explore] [2]
process timing ----- overall results -----
run time : 0 days, 0 hrs, 5 min, 24 sec    cycles done : 16
last new path : 0 days, 0 hrs, 0 min, 25 sec  total paths : 15
last uniq crash : none seen yet             uniq crashes : 0
last uniq hang : none seen yet              uniq hangs : 0
cycle progress -----
now processing : 14.0 (53.3%)                map coverage : 0.02% / 0.02%
paths timed out : 0 (0.00%)                  count coverage : 1.00 bits/tuple
stage progress -----
now trying : havoc                            favored paths : 4 (26.67%)
stage execs : 8189/16.4k (49.55%)            new edges on : 5 (33.33%)
total execs : 159k                            total crashes : 0 (0 unique)
exec speed : 491.8/sec                         total tmouts : 0 (0 unique)
fuzzing strategy yields -----
bit flips : 0/32, 0/31, 0/29                path geometry
byte flips : 0/4, 0/3, 0/1                    levels : 5
arithmetic : 0/224, 0/0, 0/0                 pending : 1
known ints : 0/26, 0/84, 0/44                pend fav : 1
dictionary : 0/0, 0/0, 0/2                    own finds : 1
havoc/read : 1/65.5k, 0/85.2k, 0/0           imported : 0
py/custom : 0/0, 0/0                           dictionary : 0/0, 0/0, 0/0
trim : 78.72k/19, 0.00%                       stability : 100.00%
[cpu002: 15%]
```

UnicoreFuzz

```
american fuzzy lop 2.06h (triforceafl)
lq process timing ----- overall results -----
run time : 0 days, 0 hrs, 6 min, 7 sec    cycles done : 0
last new path : 0 days, 0 hrs, 0 min, 30 sec  total paths : 413
last uniq crash : none seen yet             uniq crashes : 0
last uniq hang : 0 days, 0 hrs, 1 min, 0 sec  uniq hangs : 6
cycle progress -----
now processing : 0 (0.00%)                    map coverage : 0.00% / 0.00%
paths timed out : 0 (0.00%)                  count coverage : 1.31 bits/tuple
stage progress -----
now trying : havoc                            favored paths : 298 (72.15%)
stage execs : 7115/32.0k (24.11%)            new edges on : 350 (84.75%)
total execs : 12.5k                            total crashes : 0 (0 unique)
exec speed : 47.71/sec (slow!)                total tmouts : 10 (6 unique)
fuzzing strategy yields -----
bit flips : 6/32, 3/31, 2/29                path geometry
byte flips : 0/4, 0/3, 0/1                    levels : 2
arithmetic : 10/224, 0/204, 0/68             pending : 413
known ints : 1/26, 0/15, 0/10                 pend fav : 298
dictionary : 0/0, 0/0, 0/0                    own finds : 60
py/custom : 0/0, 0/0                           imported : 0
trim : 92.56k/13, 0.00%                       dictionary : 0/0, 0/0, 0/0
[cpu: 14%]
```

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