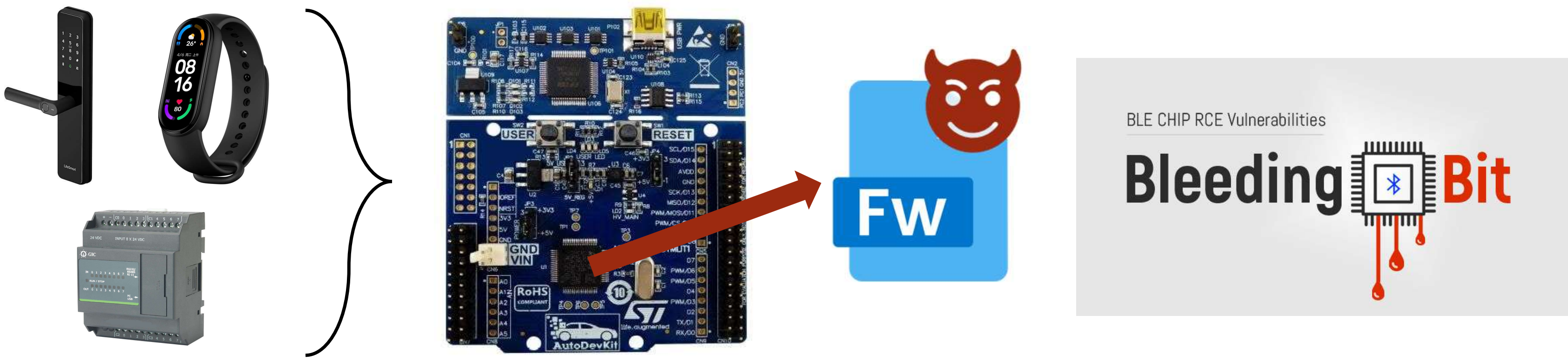
A Friend's Eye is A Good Mirror: Synthesizing MCU Peripheral Models from Peripheral Drivers

Chongqing Lei, Zhen Ling, Yan Yang, Junzhou Luo, *Southeast University*
Yue Zhang, *Drexel University*
Xinwen Fu, *University of Massachusetts Lowell*

東南大學
SOUTHEAST UNIVERSITY
Drexel
UNIVERSITY
UMASS
LOWELL

Background – MCU

- Microcontroller units (MCUs) are widely used in embedded systems
- MCU firmware controls the MCU
- MCU firmware can be vulnerable
- **Security analysis of MCU firmware is essential**
  - Static analysis
  - **Dynamic analysis**: requires execution environment



Background – Rehosting

- **Rehosting**
  - Creating virtual execution environment for dynamic firmware analysis
- **Challenge**
  - Sea of hardware (CPUs + *peripherals*) ⟶ **Manual emulation is unscalable**
- **Existing solutions**
  - Hardware oriented – requires actual hardware, not always available
    - Hardware-in-the-loop: Avatar[NDSS'14], Surrogates[WOOT'15], Inception[Security'18]
    - Record-and-replay: Pretender[RAID'19], Conware[AsiaCCS'21]
  - Firmware oriented – limited fidelity and generality
    - Function level: HALucinator[Security'20], Para-rehosting[NDSS'21], BaseSAFE[WiSec'20]
    - Register level: Laelaps[ACSAC'20], P2IM[Security'20], μEmu[Security'21], Fuzzware[Security'22]

Can we automatically emulate hardware with high fidelity and generality?

Motivating Example

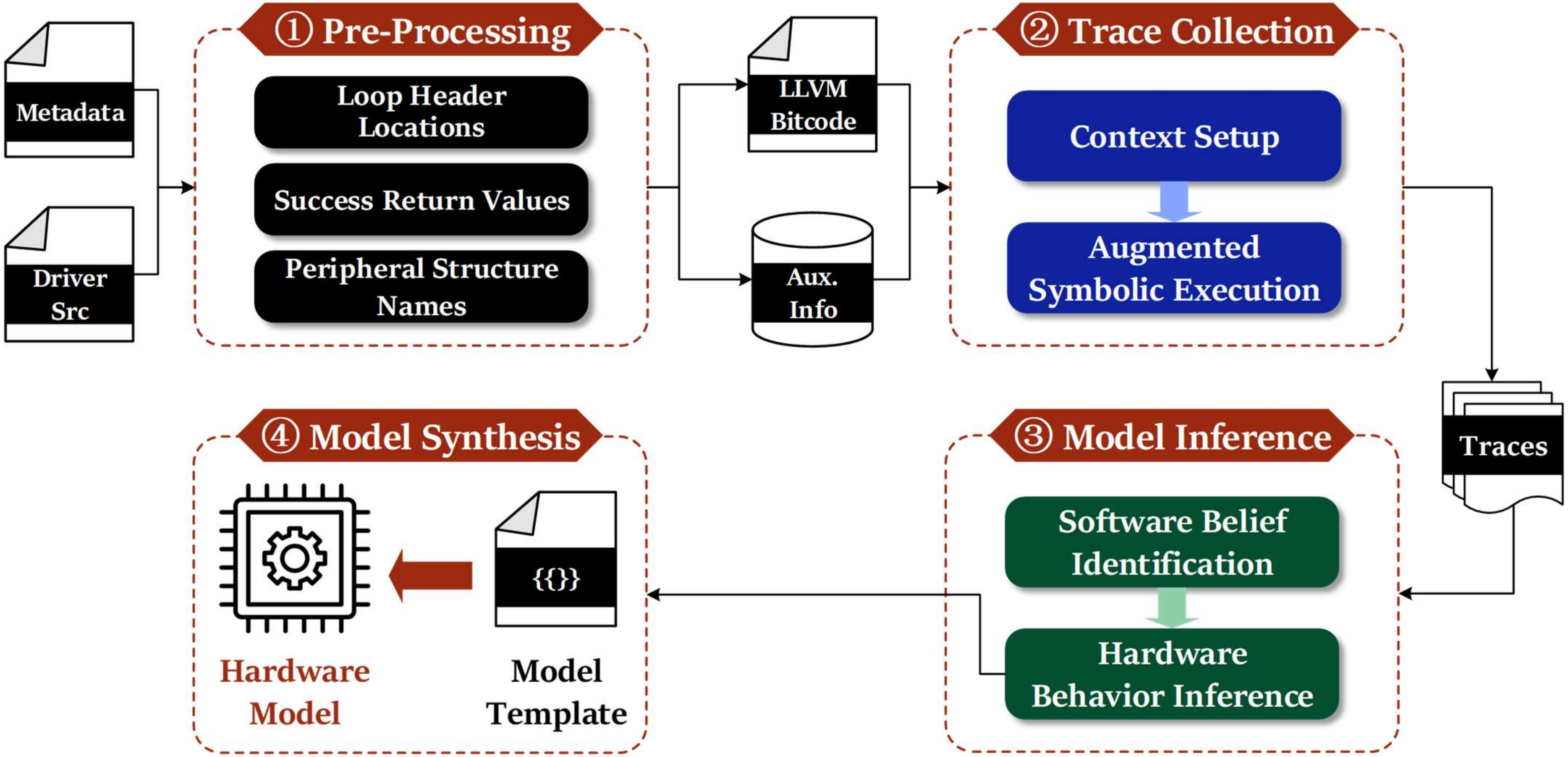
- Drivers contain rich information about hardware behaviors

```
// uart.c
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef
*huart, uint8_t *pData, uint16_t Size, uint32_t
Timeout) {
  ...
  while (Size > 0) {
    /* wait until UART_FLAG_RXNE flag in the ISR
register is set by the hardware */
    if (UART_WaitOnFlagUntilTimeout(huart,
UART_FLAG_RXNE, RESET, tickstart, Timeout) != HAL_OK)
{
      return HAL_TIMEOUT;
    }
    /* read incoming data from the TDR register */
    *pData = (uint8_t)(huart->Instance->RDR &
(uint8_t)uhMask);
    ++pData;
    --Size;
  }
  ...
  return HAL_OK;
}
```

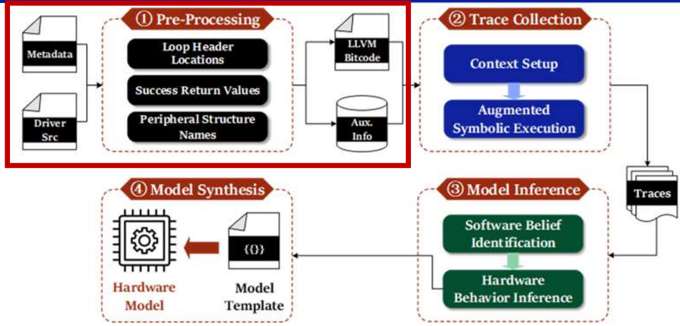
Software Belief: Data register RDR can only be read when RXNE is set

Hardware Behavior: RXNE is set by hardware when the incoming data stored in RDR is ready to be read

PERRY Design – Overview



PERRY Design – Pre-processing



- Extract useful information for later use

```c
// uart.c
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef
*huart, uint8_t *pData, uint16_t Size, uint32_t
Timeout) {
  ...
  while (Size > 0) {
    /* wait until UART_FLAG_RXNE flag in the ISR
register is set by the hardware */
    if (UART_WaitOnFlagUntilTimeout(huart,
UART_FLAG_RXNE, RESET, tickstart, Timeout) != HAL_OK)
{
      return HAL_TIMEOUT;
    }
    /* read incoming data from the TDR register */
    *pData = (uint8_t)(huart->Instance->RDR &
(uint8_t)uhMask);
    ++pData;
    --Size;
  }
  ...
  return HAL_OK;
}
```

A loop waiting for RXNE to be set by hardware

➡ **Loop Header**

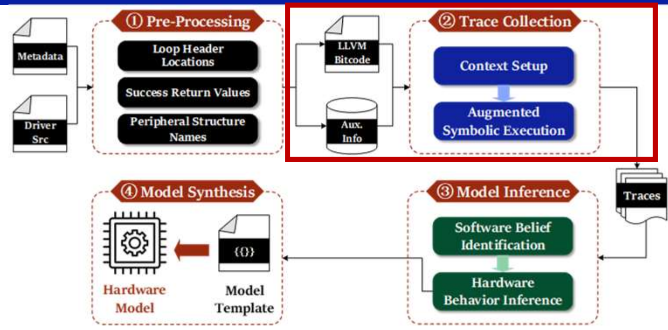
huart->Instance contains peripheral registers

➡ **Peripheral Structure Name**

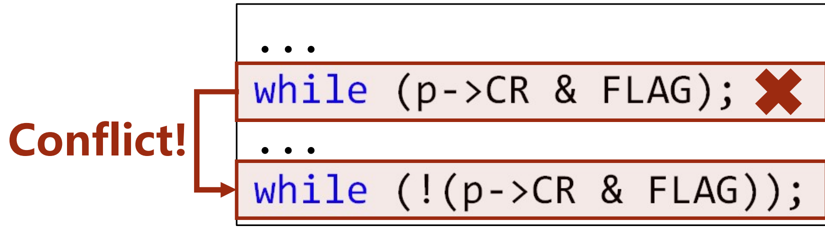
HAL_OK represents success return

➡ **Success Return Value**

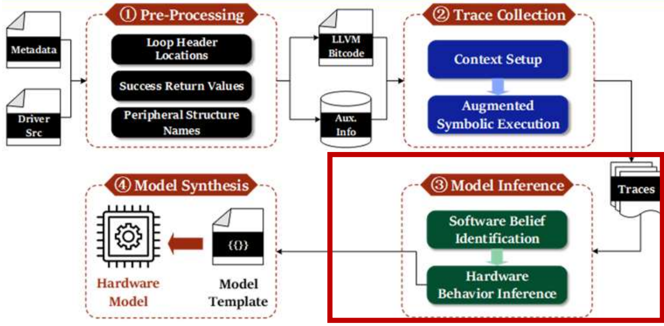
PERRY Design – Trace Collection



- Collect useful information for model inference
- **Context setup**
  - Top-level driver functions as entry points
  - Symbolize MMIO regions, global variables and parameters
  - Taint MMIO registers and data buffers
  - Hook callbacks (unresolved function pointers)
- **Symbolic execution**
  - Taint propagation
  - Jump out of loops actively
  - Remove conflicting register-related constraints using check-points
- **Collected information**
  - Symbolic execution exit status, function return value, path constraints, register accesses, taints, callback invocations



PERRY Design – Model Inference



- Reading data registers
  - Registers whose taints flow into data buffers

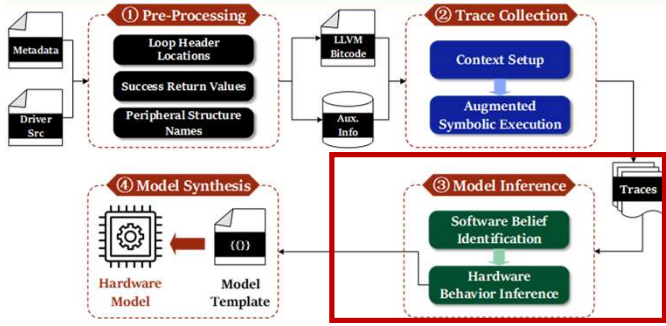
```
int rx_func(u8 *data) {
    ...
    while (!(p->SR & RXNE));
    *data = p->DR;
    ...
}
```

Software Belief: Data register DR can only be read when register-related path constraints C_{reg} is satisfied

↓

Hardware Behavior: Hardware updates registers such that C_{reg} is satisfied when incoming data stored in DR is ready to be read

PERRY Design – Model Inference



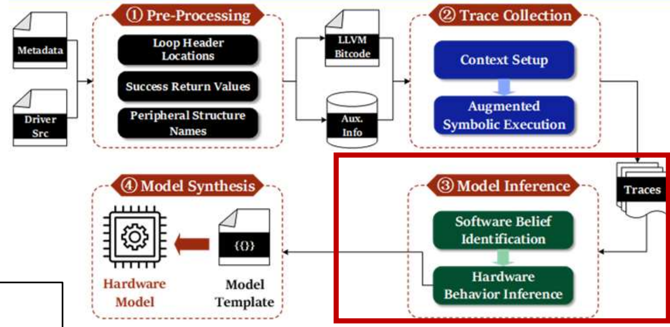
- Writing data registers
  - Registers tainted by data buffers
  - Success return value

```
int tx_func(int *data, int len) {
    for (int i = 0 ; i < len; ++i) {
        while (!(p->SR & TXE));
        p->DR = data;
    }
    while (!(p->SR & TC));
    return success;
}
```

Software Belief: Data transmission through data register DR only succeeds when register-related path constraints C_{reg} is satisfied

Hardware Behavior: Hardware updates registers such that C_{reg} is satisfied when outgoing data stored in DR is successfully transmitted

PERRY Design – Model Inference



- Updating non-data registers
  - Waiting registers to be updated after reads/writes

```
void non_data_update() {
  p->R1 |= FLAG_1;              // software update, C_R1
  while (!(p->R2 & FLAG_2));    // wait hardware update, C_R2
  ...
  if (p->R1 & FLAG_1) {         // check software update, C_R1
    while (!(p->R2 & FLAG_2)); // wait hardware update, C_R2
  }
}
```

Case 1 (first highlighted block)

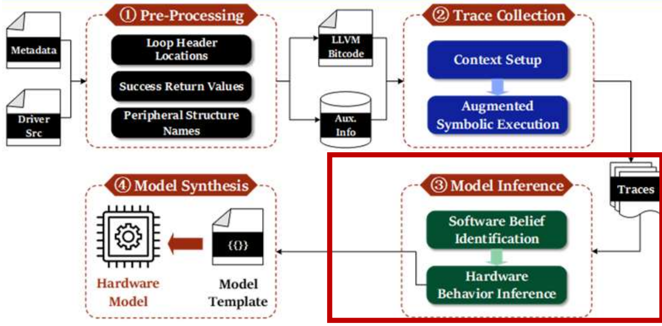
Case 2 (second highlighted block)

Software Belief: The value of register R_2 must satisfy constraint C_{R_2} after the value of register R_1 is updated in a way that constraint C_{R_1} is satisfied

↓

Hardware Behavior: Hardware updates R_2 such that C_{R_2} is satisfied when R_1 is updated in a way that constraint C_{R_1} is satisfied

PERRY Design – Model Inference



- Handling interrupts
  - Reading/writing data registers
  - Invoking callbacks

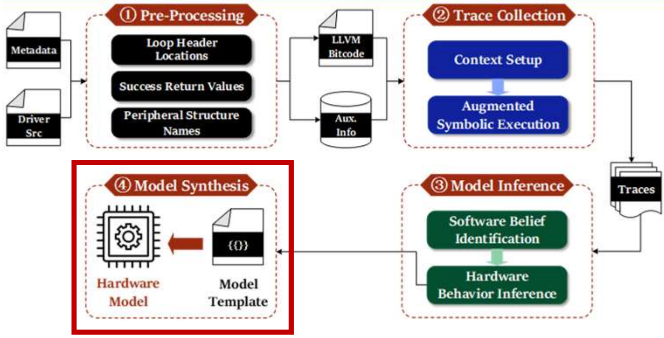
```
void isr_func() {
    if ((p->CR & TXEIE) && (p->SR & TXE)) {
        // handle TXE interrupt
        txe_callback();
    } else if ((p->CR & RXNEIE) && (p->SR & RXNE)) {
        // handle RXNE interrupt
        rxne_callback();
    }
}
```

Software Belief: Interrupts are handled only when register-related path constraint $C_{reg}^{1} \vee C_{reg}^{2} \vee \cdots \vee C_{reg}^{N}$ is satisfied

Hardware Behavior: Hardware fires interrupts when $C_{reg}^{1} \vee C_{reg}^{2} \vee \cdots \vee C_{reg}^{N}$ is satisfied

PERRY Design – Model Synthesis

- Template-based model synthesis
  - Fills-in holes with inferred hardware behaviors
  - Generates source files that can be integrated into QEMU




```
def on_recv(data):
  store(DR, data)
  # update related regs
    ...
  # fire interrupts
  if should_interrupt():
    fire_interrupt()
```

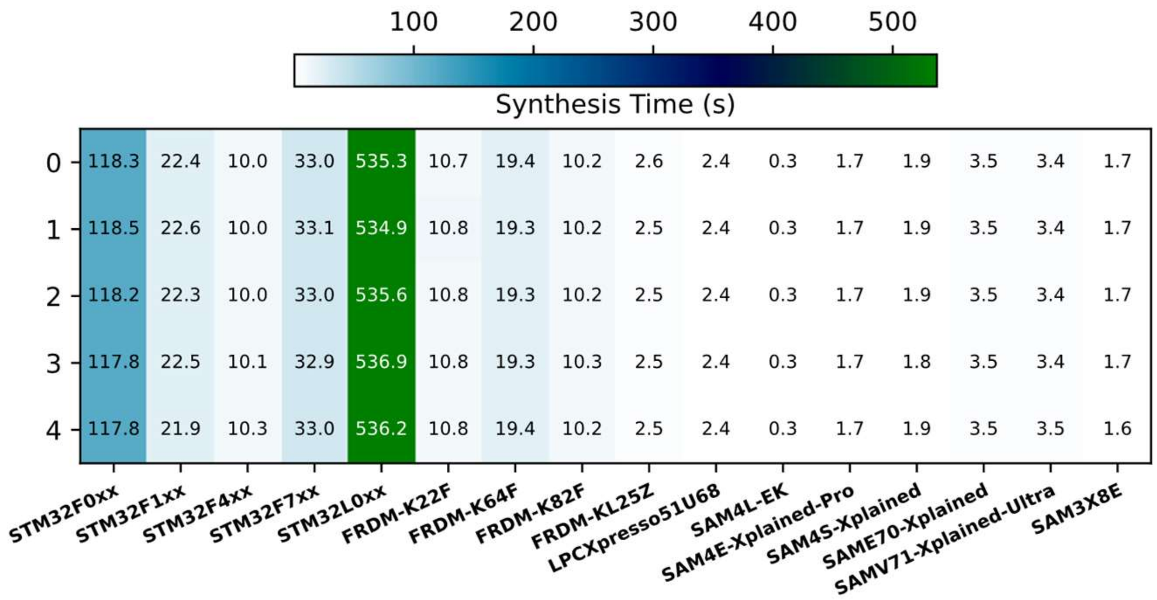
```
def on_send():
  send(load(DR))
  # update related regs
  ...
  # fire interrupts
  if should_interrupt():
    fire_interrupt()
```

```
def on_update(r1, data1):
  store(r1, data1)
  # update related registers
  r2, data2 = get_related(r1, data1)
  store(r2, data2)
  # fire interrupts
  if should_interrupt():
    fire_interrupt()
```

Evaluation – Efficiency

RQ1: How effective is PERRY?

- 10 driver libraries from 3 top MCU vendors (ST, NXP, Microchip)
- >30 MCUs are covered
- Synthesis time ranges from 0.3 seconds to ~9 hours



Evaluation – Efficiency

RQ2: Can PERRY infer consistent hardware behaviors?

- Dataset: P^2IM unit tests (66 in total)
- 49/66 (74.24%) passed without manual intervention ⟺ 0% for SEmu[CCS'22]
- All passed after fixing wrong/missing hardware behaviors with 6 LoC

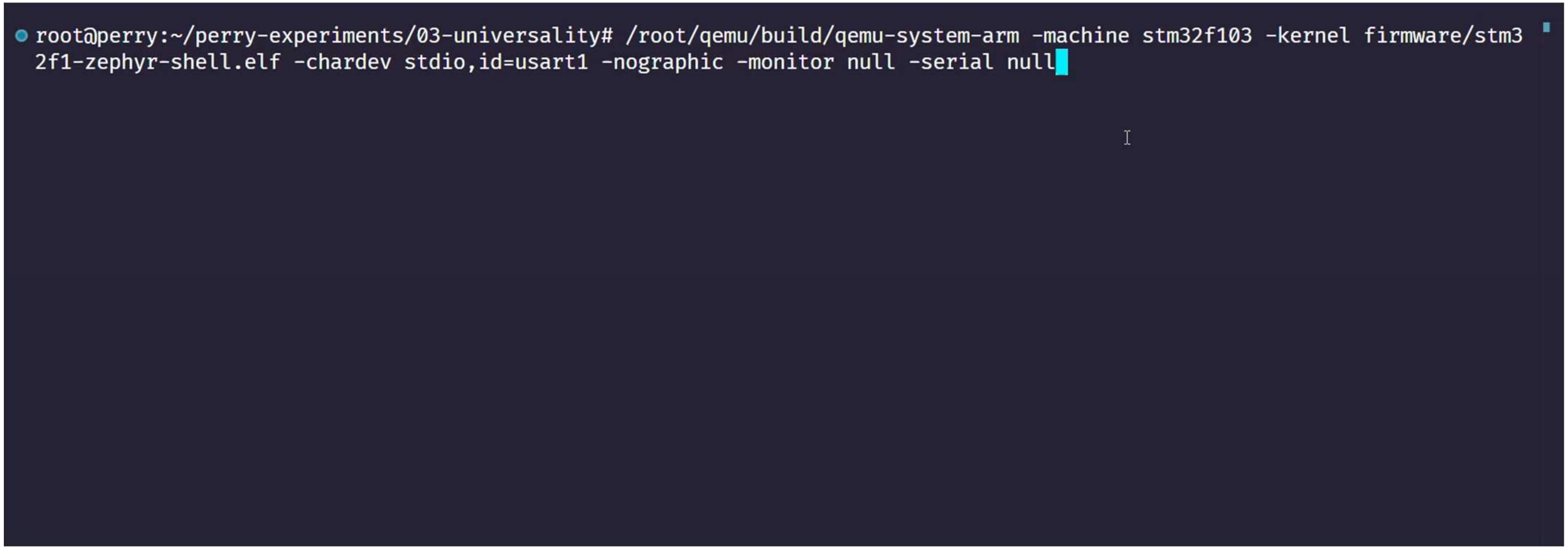
Peri.	Unit test	STM32F103			FRDM-K64F	ATSAM3X8E		Passing Rate
		Arduino	RIOT*	NUTTX	RIOT	Arduino	RIOT	
ADC	read converted values	✓	-	✓	✓	✓	✓	5/5
DAC	write values for conversion	-	-	-	-	✓	✓	2/2
GPIO	execute the interrupt callback	✓	✗(RCC ✧)	✓	✓	✓	✓	5/6
	read a pin	✓	✗(RCC ✧)	✓	✓	✓	✓	5/6
	set/clear a pin	✓	✗(RCC ✧)	✓	✓	✓	✓	5/6
PWM	perform basic configuration	✓	-	✓	✓	✓	✓	5/5
I2C	receive bytes	✗(▲)	-	✗(▲)	✗(✧)	✓	-	1/4
	send bytes	✗(▲)	-	-	✗(✧)	✓	-	1/3
UART	receive bytes	✓	✗(RCC ✧)	✓	✓	✗(✧)	✗(✧)	3/6
	transmit bytes	✓	✗(RCC ✧)	✓	✓	✗(✧)	✓	4/6
SPI	receive bytes	✓	✗(RCC ✧)	✓	✓	✓	✓	5/6
	transmit bytes	✓	✗(RCC ✧)	-	✓	✓	✓	4/5
TIMER	execute the interrupt callback	-	✗(RCC ✧)	-	✓	-	✓	2/3
	read counter values	-	✗(RCC ✧)	-	✓	-	✓	2/3
LoC to Fix		3 (1 for RCC, 2 for I2C)			1 (for I2C)	2 (for UART)		**49/66(74.24%)**

* All STM32F103/RIOT unit tests failed due to a single wrong behavior in the RCC peripheral. Unit tests marked with "-" do not exist. ✧ represents implicit assumptions on hardware and ▲ represents in-context register operations.

Evaluation – Universality

RQ3: Can hardware models generated by PERRY emulate various firmware?

- Dataset: P^{2}IM real world firmware samples (10) + shell firmware from LiteOS and Zephyr (19)
- 20/29 (68.97%) are emulated without manual intervention



Evaluation – Scalability

RQ4: Can hardware models generated by PERRY be easily fixed?

- 35 LoC to fix all generated models, at most 4 LoC to fix one model
- Can emulate various firmware once fixed

MCUs	Firmware	# Miss. Behaviors	# Wrong Behaviors	LoC to Fix
STM32F0 series	Zephyr-Shell LiteOS-Shell	1 (☆)	1 (▲)	4
STM32F1 series	Zephyr-Shell LiteOS-Shell Drone Gateway Reflow_Oven Robot Soldering_Iron	0	0	0
STM32F4 series	Zephyr-Shell LiteOS-Shell CNC PLC	0	1 (▲)	1
STM32F7 series	Zephyr-Shell LiteOS-Shell	0	1 (▲)	1
STM32L0 series	Zephyr-Shell LiteOS-Shell	1 (☆)	0	3
FRDM-K22F	Zephyr-Shell	0	0	0
FRDM-K64F	Zephyr-Shell Console	0	0	0
FRDM-K82F	Zephyr-Shell	0	0	0
FRDM-KL25Z	Zephyr-Shell	0	0	0
SAM4L-EK	Zephyr-Shell	2 (✧)	0	4
SAM4E Xplained Pro	Zephyr-Shell	2 (✧)	0	4
SAM4S Xplained	Zephyr-Shell	2 (✧)	0	4
SAM E70 Xplained	Zephyr-Shell	2 (✧)	0	4
SAM V71 Xplained Ultra	Zephyr-Shell	2 (✧)	0	4
SAM3X8E	Heat_Press Steering_Control	0	0	0

Note ☆: Non-trivial hardware functionalities. ✧: Implicit assumptions on Hardware. ▲: In-context register operations.

Security Application – Mining Specification Violation Bugs

- Drivers may interact with peripherals without following protocols defined by the specification
- Cross-checking hardware behaviors inferred by PERRY with those defined in the specification
- 2 specification violation bugs in ST and NXP drivers

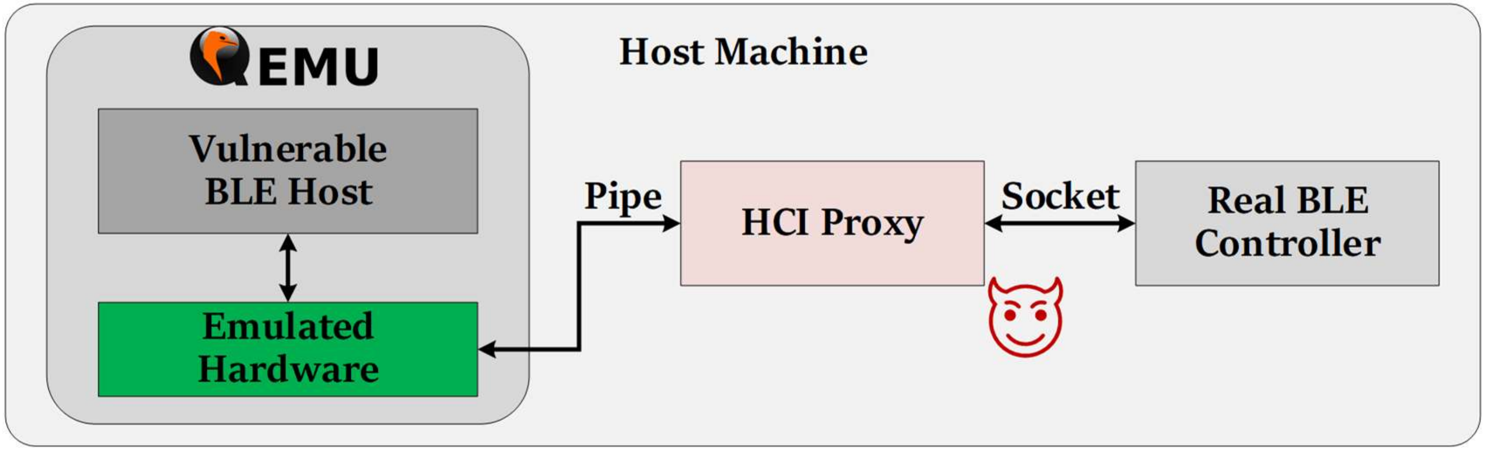
```
#define __HAL_RCC_HSI48_ENABLE()  \
    SET_BIT(RCC->CR2, RCC_CR2_HSI48ON)
#define RCC_FLAG_HSI48RDY ((uint8_t)((CR2_REG_INDEX << 5U) |
RCC_CR2_HSI48RDY_BitNumber))
#define RCC_CR2_HSI48RDY_BitNumber 16
#define RCC_CR2_HSI48ON_Pos (16U)
#define RCC_CR2_HSI48RDY_Pos (17U)

__HAL_RCC_HSI48_ENABLE();
tickstart = HAL_GetTick();
while(__HAL_RCC_GET_FLAG(RCC_FLAG_HSI48RDY) == RESET) { ... }
```

Conflicted bit number for HSI48RDY

Security Application – Reproducing Firmware Vulnerabilities

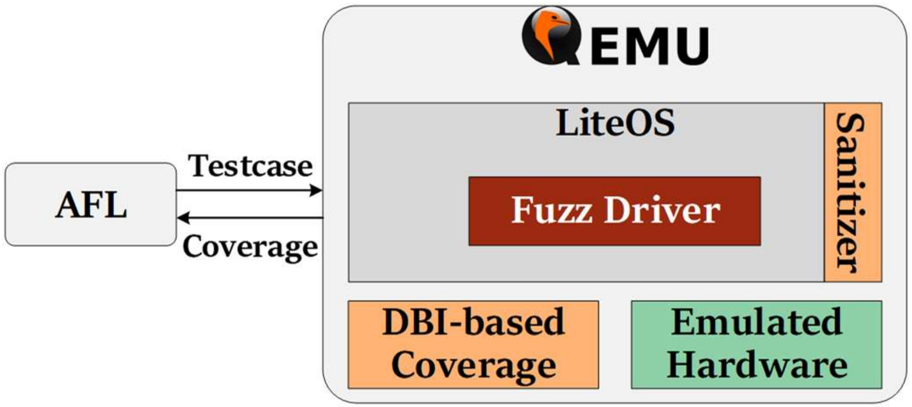
- CVE-2022-1041, CVE-2022-1042
  - Vulnerabilities in Zephyr's BLE host protocol stack
  - Triggered through malformed HCI packets transferred over UART/USB
  - UART on the emulated hardware is connected to a *real* BLE controller
  - Inject payload after BLE connection establishment



Security Application – Fuzzing RTOS

- 11 fuzz drivers for the MQTT and LWM2M protocol stacks of LiteOS
- Implement various sanitizers
  - UBSAN, KASAN, KMSAN, KCSAN
  - FLASH/RAM are too small to contain instrumented code/shadow memory…
  - Emulated hardware, just increase sizes of FLASH/RAM regions!
- 7 0-day vulnerabilities and 3 N-day vulnerabilities

Component	Target	Speed (#/sec)	# Exec.	# Path	# Vuln.
MQTT	Deserialize_ack	518.99	11,400,193	9	0
	Deserialize_connack	729.21	15,757,024	11	0
	Deserialize_connect	1233.25	26,547,383	35	1
	Deserialize_publish	634.81	13,632,707	14	1
	Deserialize_suback	517.32	11,380,177	13	1
	Deserialize_subscribe	969.45	20,505,021	12	1
	Deserialize_unsuback	469.20	10,261,209	10	0
	Deserialize_unsubscribe	537.79	11,910,801	11	1
LWM2M	coap_parse_message	331.79	7,331,274	4,112	3
	lwm2m_data_parse(TLV)	271.10	6,261,404	6,000	0
	lwm2m_data_parse(JSON)	10.84	1,638,116	3,160	2



Conclusion

- Drivers help infer hardware behaviors
- We introduce PERRY, a tool that effectively synthesizes hardware models from hardware drivers
- PERRY generates hardware models that can faithfully emulate various firmware
- PERRY can boost various security-focused tasks

Thank You!

leicq@seu.edu.cn

PERRY is available on GitHub

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Evaluation – Scalability

RQ4: Can hardware models generated by PERRY be easily fixed?

- 35 LoC to fix all generated models, at most 4 LoC to fix one model
- Causes of failing cases:
  - Non-trivial hardware functionalities
    - e.g., interrupt table relocation via non-standard peripherals
  - Implicit assumptions on hardware
    - Cannot capture behaviors not presented in driver code
  - In-context register operations

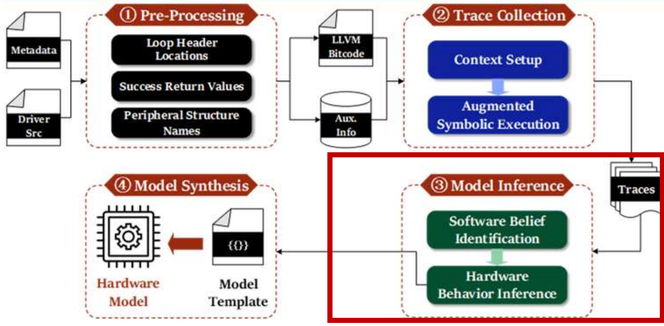
Expected

```
CLEAR_BIT(RCC->CR, RCC_CR_HSEON);
CLEAR_BIT(RCC->CR, RCC_CR_HSEBYP);
while(__HAL_RCC_GET_FLAG(RCC_FLAG_HSERDY) != RESET) { ... }
```

Inferred

PERRY Design – Model Synthesis

- Complex software beliefs and hardware behaviors (e.g., DMA)
  - Extend PERRY with on-demand analysis



- Parameter
  - registers receiving taints

```
void EDMA_TcdSetTransferConfig(edma_tcd_t *tcd, const
edma_transfer_config_t *config, ...)
{
    tcd->SADDR = config->srcAddr;
    tcd->DADDR = config->destAddr;
    tcd->NBYTES = config->minorLoopBytes;
}
```

- Function
  - register updates

```
void EDMA_StartTransfer(edma_handle_t *handle) {
    handle->base->SERQ = DMA_SERQ_SERQ(handle-
>channel);
}
```

- Callback
  - invocation constraints

```
void EDMA_HandleIRQ(edma_tcd_t *tcd) {
    bool transfer_done = tcd->CSR &
DMA_CSR_DONE_MASK) != 0U);
    if (transfer_done) {
        transfer_done_callback()
    }
}
```