What Your Firmware Tells You Is Not How You Should Emulate It: A Specification-Guided Approach for Firmware Emulation (CCS 2022)

Wei Zhou[†], Lan Zhang[†], Le Guan, Peng Liu, Yuqing Zhang

Motivation

- Emulation is important for firmware dynamic analysis, but emulating firmware of microcontrollers is challenging due to the lack of peripheral models.
- Existing works P2IM, Laelaps, Jetset, Fuzzware, uEmu try to infer peripheral access patterns from firmware itself: **Emulator should generate responses that meet the expectations of firmware so that it does not crash or hang.** However, incomplete and misleading information in firmware undermines the effectiveness, efficiency and applicability of these approaches.
- State diagrams and state tables are widely adopted in chip manuals to specify the peripheral state machines. These information can be helpful for firmware emulation.

Contributions

- A specification-guided firmware emulation to more accurately emulate firmware. The core technique is to leverage NLP techniques to automatically extract useful Condition-Action rules from chip manuals.
- Incorporating invalidity-guided emulation to identify missing or faulty C-A rules extracted by SEmu.
- A new method based on modified edit distance that measures trace similarity.
- Experiment on **STM32F1**, **STM32F4**, **STM32L1**, **NXP K64 series**, and **Atmel SMART series** shows that SEmu achieves much better trace fidelity compared with firmware-guided approaches.
- Successfully uncovered real non-compliance bugs in firmware.

Background

MCU Reference Manuals

Chip vendors commonly publish reference manuals for each chip written in natural language, providing essential information on how to use every peripheral, including its registers, memory map, hardware interface, and behaviors. A reference manual typically includes:

- Register memory map

- Field description
    - Name
    - Bits
    - Access permissions
    - Functions: **Conditions** and **Actions**
- Interrupt vector table
- DMA channel assignment

Natural Language Processing

Although reference manuals (mainly in PDF format) are unstructured data, there are observable characteristics in the format and text. Hence, NLP techniques can be used to understand the naturally expressed sentences and extract condition-action logic from chip manuals.

```
RDRF is set when the number of datawords in the
receive buffer is equal to or more than the number
indicated by RWFIFO[RXWATER].
```

Part-of-speech (POS) tagging to recognize three named entities: RDRF, receive buffer, RWFIFO[RXWATER].
Constituency Analysis to generate a parse tree to identify conditions and actions.

- Actions: RDRF is set
- Condition: the rest of the sentence

Typed Dependencies Analysis to analyze the grammatical structure, matching the verbs and their corresponding subjects or objects

Approach

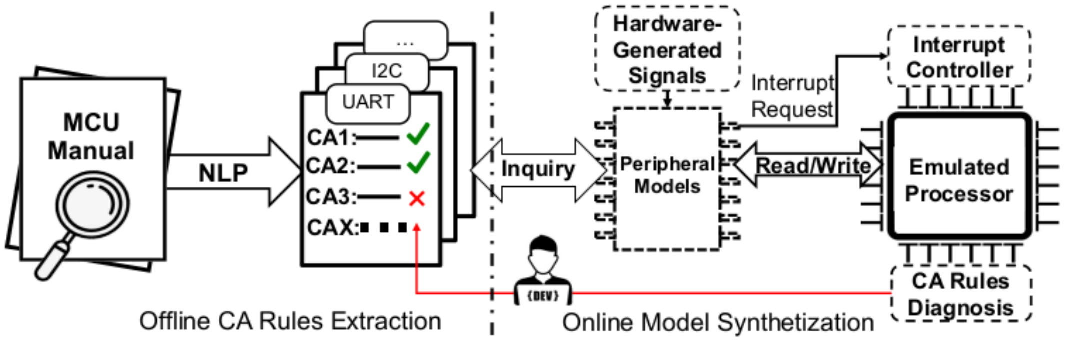


Figure 2: *SEmu* overview

EXTRACTING C-A RULES FROM CHIP MANUALS

Condition-Action (CA) rule describes how important events such as accessing an MMIO register influence the peripheral state.
A **condition** comprises one or more predicates combined using the Boolean and operator.

- Type-1 Condition: via external hardware generated signals

- Type-2 Condition: via firmware generated signals
- Type-3 Condition: via internal generated signals

An **action** is one or more assignment functions.

- Type-1 Action: MMIO register related
- Type-2 Action: Interrupt related
- Type-3 Action: DMA related

Extracting C-A rules has several challenges:
How to identify relevant sentences?
Solution: Collect sentences that contain certain named entities. First collect relevant named entities from MMIO Register Map. Then scan sentences from field description part that contain at least one named entities. Extend the set of named entities with newly encountered subjects/objects.
How to identify and handle co-references?
Solution: Matching the random noun phrases with the initial set of named entities extracted from the register memory map. Use approximate string matching to measure the similarity between an unknown noun phrase with each of the initial set of named entities to identify possible co-references.
How to identify conditions and actions?
Solution: Use Stanford Constituency Parser to analyze the grammatical structure of sentences.
How to represent C-A rules?
Solution: First translate conditions and actions into predicates and assignment functions using Stanford Dependency Parser. Then formulate rule triggers and actions. Last, formulate the C-A rules. For example:

$$B\#D[R] \geq RWFIFO[RXWATER] \rightarrow S1[RDRF] := 1 \tag{1}$$

Synthesizing Peripheral Models with C-A rules

Use QEMU to emulate the basic ARM ISA and core peripherals (e.g., NVIC). During firmware emulation, we dynamically build a model for each peripheral, taking the intercepted firmware-peripheral interactions and the extracted C-A rules as input. Three types of conditions should be checked during firmware execution. Type-2 and Type-3 condition can be easily checked. For type-1 condition (external hardware triggered conditions), SEmu can only emulate buffer-based and timer-based hardware signals.

Diagnosing Faulty/Missing C-A rules

Reasons why C-A rules may be imprecise:

- SEmu cannot emulate certain hardware
- NLP techniques cannot handle some complex centences

Solution: Use symbolic execution to dectect root cause for emulation failure.

- Select a firmware and corresponding valid test cases that run normally on real devices.

- Use symbolic execution to detect invalid states during emulation with synthesized peripherals.

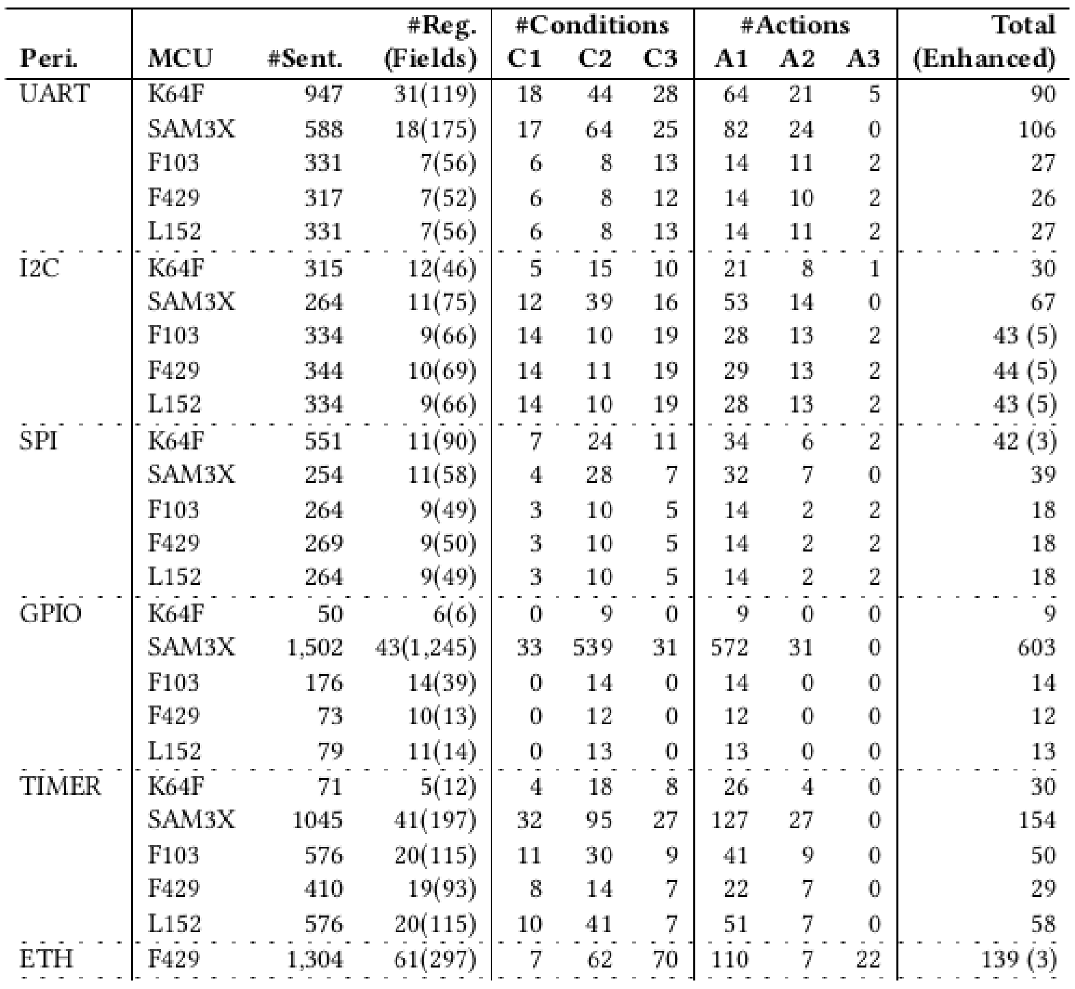
Evaluation

Target five chip manuals that cover more than twenty popular MCU series

- STM32F103, STM32F429, STM32L152
- NXP K64F series
- Atmel SAM3X series

RQ1: Can our NLP engine automatically extract C-A rules to describe peripheral behaviors?
For each manual, we extracted C-A rules for 26 popular peripherals, including ADC, I2C, SPI, GPIO, UART, Ethernet, etc.

Table 5: C-A Rule Statistics

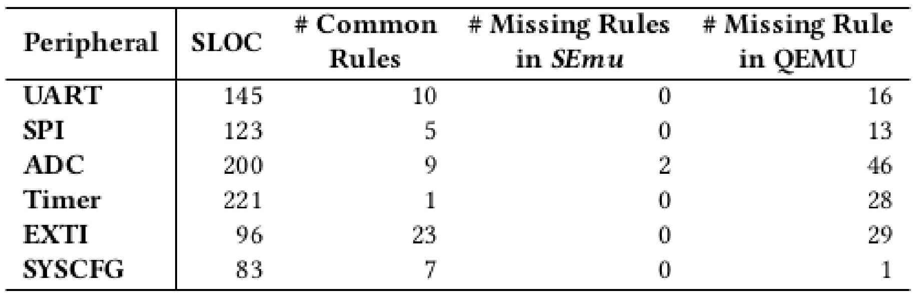


			#Reg.	#Conditions			#Actions			Total
Peri.	MCU	#Sent.	(Fields)	C1	C2	C3	A1	A2	A3	(Enhanced)
UART	K64F	947	31(119)	18	44	28	64	21	5	90
	SAM3X	588	18(175)	17	64	25	82	24	0	106
	F103	331	7(56)	6	8	13	14	11	2	27
	F429	317	7(52)	6	8	12	14	10	2	26
	L152	331	7(56)	6	8	13	14	11	2	27
I2C	K64F	315	12(46)	5	15	10	21	8	1	30
	SAM3X	264	11(75)	12	39	16	53	14	0	67
	F103	334	9(66)	14	10	19	28	13	2	43 (5)
	F429	344	10(69)	14	11	19	29	13	2	44 (5)
	L152	334	9(66)	14	10	19	28	13	2	43 (5)
SPI	K64F	551	11(90)	7	24	11	34	6	2	42 (3)
	SAM3X	254	11(58)	4	28	7	32	7	0	39
	F103	264	9(49)	3	10	5	14	2	2	18
	F429	269	9(50)	3	10	5	14	2	2	18
	L152	264	9(49)	3	10	5	14	2	2	18
GPIO	K64F	50	6(6)	0	9	0	9	0	0	9
	SAM3X	1,502	43(1,245)	33	539	31	572	31	0	603
	F103	176	14(39)	0	14	0	14	0	0	14
	F429	73	10(13)	0	12	0	12	0	0	12
	L152	79	11(14)	0	13	0	13	0	0	13
TIMER	K64F	71	5(12)	4	18	8	26	4	0	30
	SAM3X	1045	41(197)	32	95	27	127	27	0	154
	F103	576	20(115)	11	30	9	41	9	0	50
	F429	410	19(93)	8	14	7	22	7	0	29
	L152	576	20(115)	10	41	7	51	7	0	58
ETH	F429	1,304	61(297)	7	62	70	110	7	22	139 (3)

RQ2: Can the diagnosis tool help correct incorrect rules?
In total, we added 26 rules (0.6%) and fixed 21 rules (0.5%) as shown in the brackets in the last columns of Table 5. To be specific, we added 5 C-A rules to I2C (F103, F429 and L152, 15 rules in total), 3 rules to ADC DMA mode (SAM3X), 3 rules for Ethernet (F429), 4 rules for MCG (K64), and 1 rule for PMC (SMA3); we also modified 6 rules for RCC (F103, F429 and L152, 18 rules in total) and remove 3 useless rules for SPI debug mode (K64).

RQ3: Are the extracted C-A rules complete and sound?

Table 1: C-A Rule Faithfulness Compared with Rules Used in QEMU Models (STM32F405)



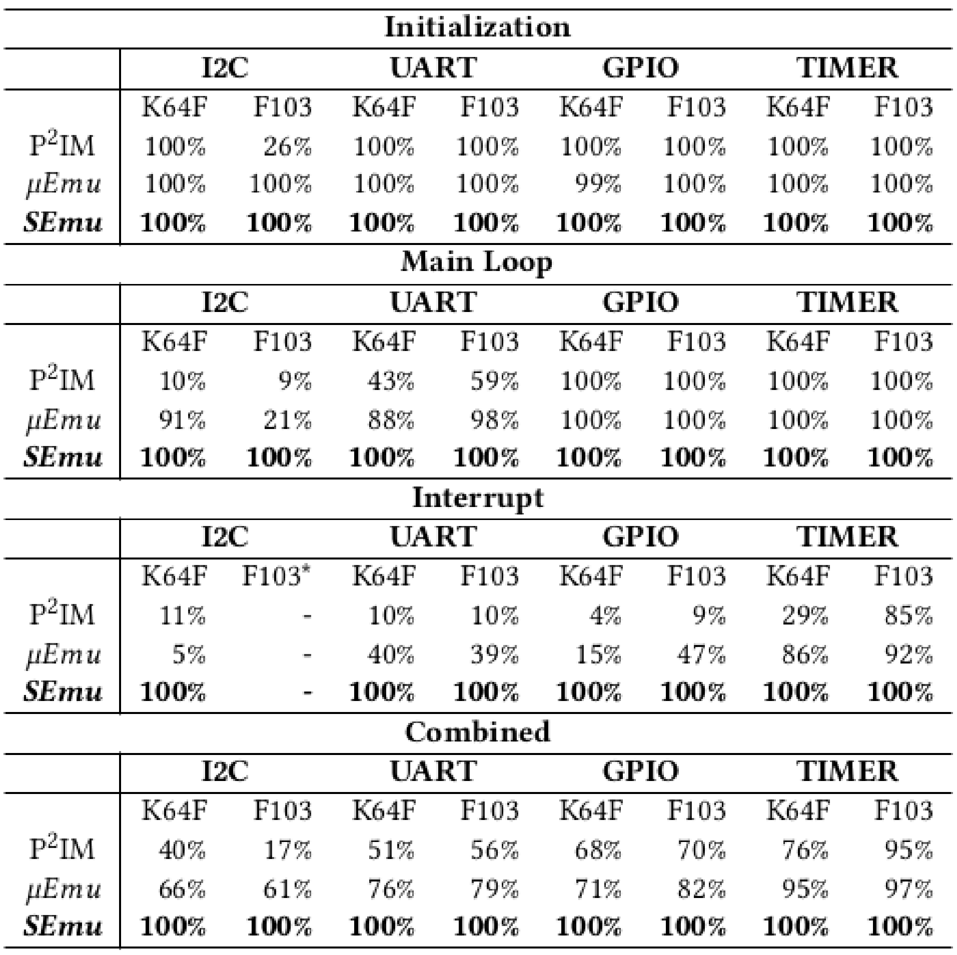
Peripheral	SLOC	# Common Rules	# Missing Rules in *SEmu*	# Missing Rule in QEMU
UART	145	10	0	16
SPI	123	5	0	13
ADC	200	9	2	46
Timer	221	1	0	28
EXTI	96	23	0	29
SYSCFG	83	7	0	1

RQ4: Can peripheral models dynamically built with C-A rules provide higher fidelity compared with firmware-guided approaches?

To collect traces on real devices, we used OpenOCD and an external debugging dongle to connect the target boards to the remote gdbserver provided by the chip vendors.

Three types of traces are considered: initialization trace, main loop trace and interrupt trace. Use edit distance to measure trace fidelity.

Table 6: Trace Fidelity Score



Initialization	I2C		UART		GPIO		TIMER	
	K64F	F103	K64F	F103	K64F	F103	K64F	F103
P^2IM	100%	26%	100%	100%	100%	100%	100%	100%
μEmu	100%	100%	100%	100%	99%	100%	100%	100%
SEmu	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
Main Loop	**I2C**		**UART**		**GPIO**		**TIMER**	
	K64F	F103	K64F	F103	K64F	F103	K64F	F103
P^2IM	10%	9%	43%	59%	100%	100%	100%	100%
μEmu	91%	21%	88%	98%	100%	100%	100%	100%
SEmu	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
Interrupt	**I2C**		**UART**		**GPIO**		**TIMER**	
	K64F	F103*	K64F	F103	K64F	F103	K64F	F103
P^2IM	11%	-	10%	10%	4%	9%	29%	85%
μEmu	5%	-	40%	39%	15%	47%	86%	92%
SEmu	**100%**	-	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
Combined	**I2C**		**UART**		**GPIO**		**TIMER**	
	K64F	F103	K64F	F103	K64F	F103	K64F	F103
P^2IM	40%	17%	51%	56%	68%	70%	76%	95%
μEmu	66%	61%	76%	79%	71%	82%	95%	97%
SEmu	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**

*: The F103 I2C unit-test sample does not involve any external interrupts. Note that all three emulators have the 100% score with the ADC and SPI unit test samples which are not listed in the Table.

RQ5: Will the improved emulation fidelity provide better performance?
SEmu successfully reproduced all the bugs mentioned in previous work, but did not report any false crashes or hangs.

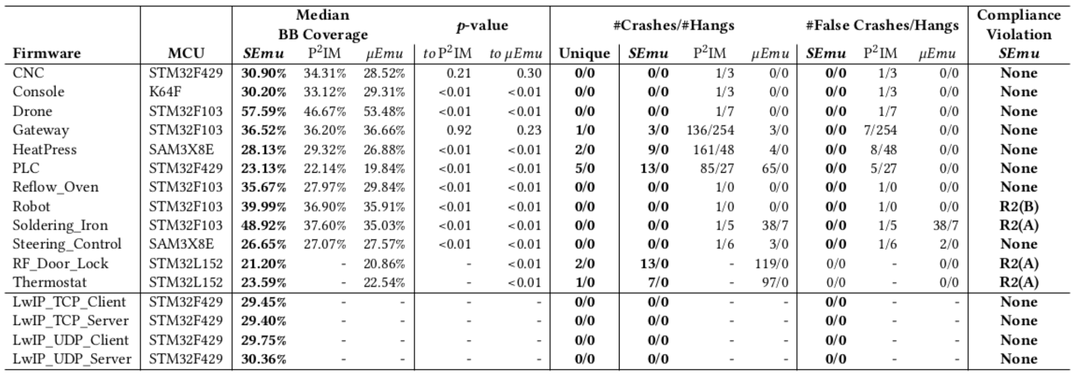


Table 2: Fuzzing and Compliance Check Results (- indicates the emulator cannot support fuzzing that firmware)

		Median BB Coverage			***p*-value**		**#Crashes/#Hangs**				**#False Crashes/Hangs**			**Compliance Violation**
Firmware	**MCU**	***SEmu***	P^2IM	*μEmu*	*to* P^2IM	*to μEmu*	**Unique**	***SEmu***	P^2IM	*μEmu*	***SEmu***	P^2IM	*μEmu*	***SEmu***
CNC	STM32F429	**30.90%**	34.31%	28.52%	0.21	0.30	**0/0**	**0/0**	1/3	0/0	**0/0**	1/3	0/0	**None**
Console	K64F	**30.20%**	33.12%	29.31%	<0.01	<0.01	**0/0**	**0/0**	1/3	0/0	**0/0**	1/3	0/0	**None**
Drone	STM32F103	**57.59%**	46.67%	53.48%	<0.01	<0.01	**0/0**	**0/0**	1/7	0/0	**0/0**	1/7	0/0	**None**
Gateway	STM32F103	**36.52%**	36.20%	36.66%	0.92	0.23	**1/0**	**3/0**	136/254	3/0	**0/0**	7/254	0/0	**None**
HeatPress	SAM3X8E	**28.13%**	29.32%	26.88%	<0.01	<0.01	**2/0**	**9/0**	161/48	4/0	**0/0**	8/48	0/0	**None**
PLC	STM32F429	**23.13%**	22.14%	19.84%	<0.01	<0.01	**5/0**	**13/0**	85/27	65/0	**0/0**	5/27	0/0	**None**
Reflow_Oven	STM32F103	**35.67%**	27.97%	29.84%	<0.01	<0.01	**0/0**	**0/0**	1/0	0/0	**0/0**	1/0	0/0	**None**
Robot	STM32F103	**39.99%**	36.90%	35.91%	<0.01	<0.01	**0/0**	**0/0**	1/0	0/0	**0/0**	1/0	0/0	**R2(B)**
Soldering_Iron	STM32F103	**48.92%**	37.60%	35.03%	<0.01	<0.01	**0/0**	**0/0**	1/5	38/7	**0/0**	1/5	38/7	**R2(A)**
Steering_Control	SAM3X8E	**26.65%**	27.07%	27.57%	<0.01	<0.01	**0/0**	**0/0**	1/6	3/0	**0/0**	1/6	2/0	**None**
RF_Door_Lock	STM32L152	**21.20%**	-	20.86%	-	<0.01	**2/0**	**13/0**	-	119/0	0/0	-	0/0	**R2(A)**
Thermostat	STM32L152	**23.59%**	-	22.54%	-	<0.01	**1/0**	**7/0**	-	97/0	0/0	-	0/0	**R2(A)**
LwIP_TCP_Client	STM32F429	**29.45%**	-	-	-	-	**0/0**	**0/0**	-	-	**0/0**	-	-	**None**
LwIP_TCP_Server	STM32F429	**29.40%**	-	-	-	-	**0/0**	**0/0**	-	-	**0/0**	-	-	**None**
LwIP_UDP_Client	STM32F429	**29.75%**	-	-	-	-	**0/0**	**0/0**	-	-	**0/0**	-	-	**None**
LwIP_UDP_Server	STM32F429	**30.36%**	-	-	-	-	**0/0**	**0/0**	-	-	**0/0**	-	-	**None**

Conclusion

Instead of proposing yet another firmware-guided emulation solution, in this work we propose the first specification-based firmware emulation solution. The new approach leverages NLP techniques to translate peripheral behaviors (specified) in human language (documented in chip manuals) into a set of structured condition-action rules. By properly executing and chaining these rules at runtime, we can dynamically synthesize a peripheral model for each peripheral accessed during firmware execution. We found some non-compliance which we later confirmed to be bugs caused by race condition.