Toward the Analysis of Embedded Firmware through Automated Re-hosting

Eric Gustafson, Marius Muench, Chad Spensky, Nilo Redini, Aravind Machiry, Aurelien Francillon, Davide Balzarotti, Yung Ryn Choe, Christopher Kruegel, Giovanni Vigna

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Let's secure the IoT!

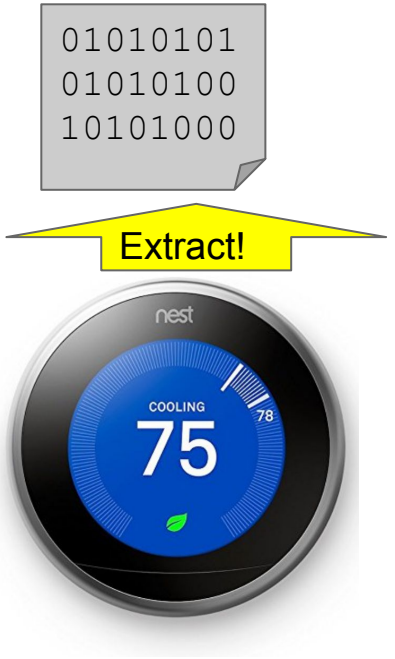
....but there's all this crazy hardware...

Our Analysis Goals:

- Fuzzing
  - Feed the program with lots of inputs until something bad happens
  - Make lots of copies of the code and its environment to make it feasible
- Symbolic Execution
  - Used to understand how data affects program behavior, and detect possible invalid behaviors
  - Needs a strong model of the code's environment (software and hardware) to be tractable.

What if…

What if…

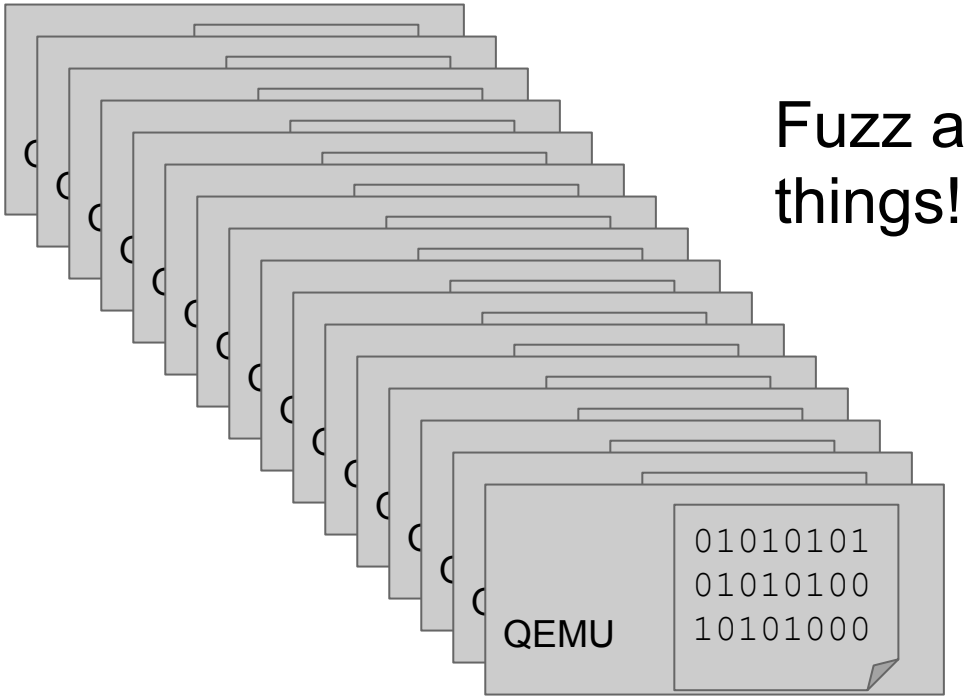


What if...


QEMU
01010101
01010100
10101000
Virtualize!


nest
COOLING
75
78

What if...



Fuzz all the things!!!

Re-hosting to the Rescue?

"**Re-hosting**": the act of transferring a piece of software from one execution environment into another, such as from a hardware device to a software emulator

....but there's all this crazy hardware...

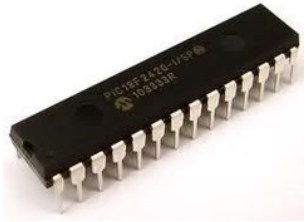
Uh oh...


Atmel
AVR


Ambarella
A9-A0-RH
NZV1X
S0101212
A999


Qualcomm
snapdragon

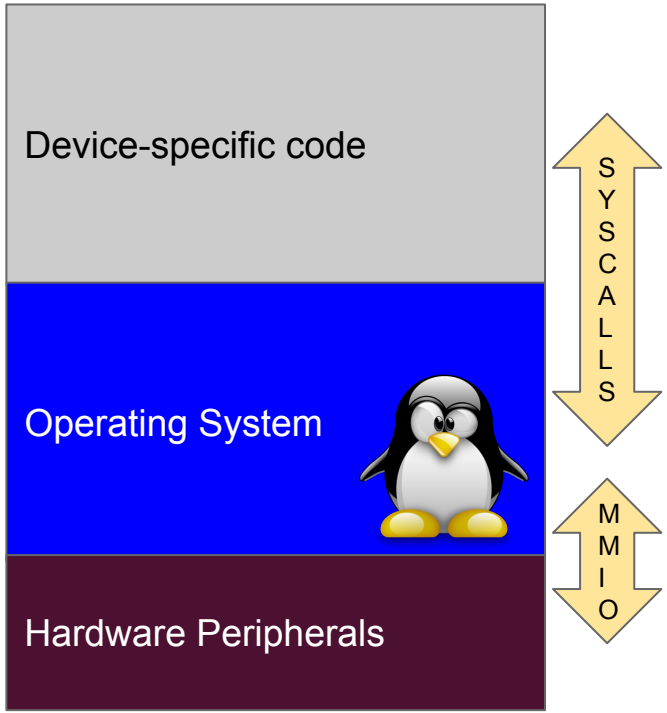

M430F1232
76TG4
DYCRD



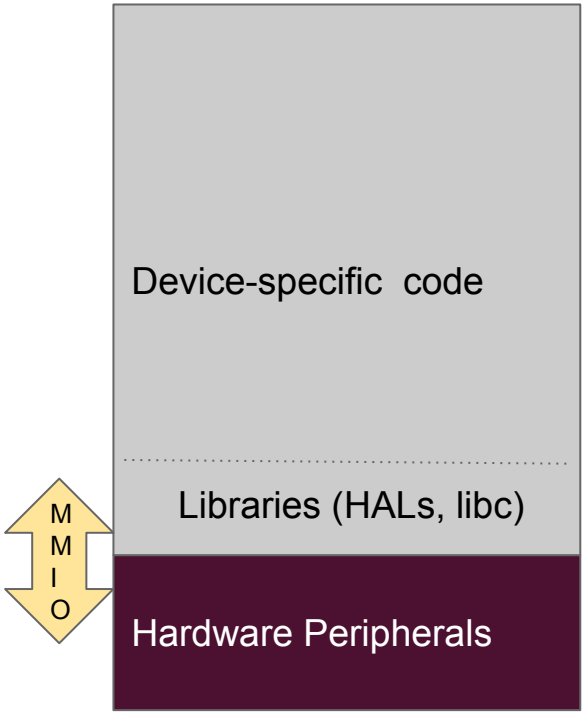

STM32F103
VCT6
ARM


NVIDIA
TEGRA K1
NVIDIA
TEGRA X1

Firmware is hard!

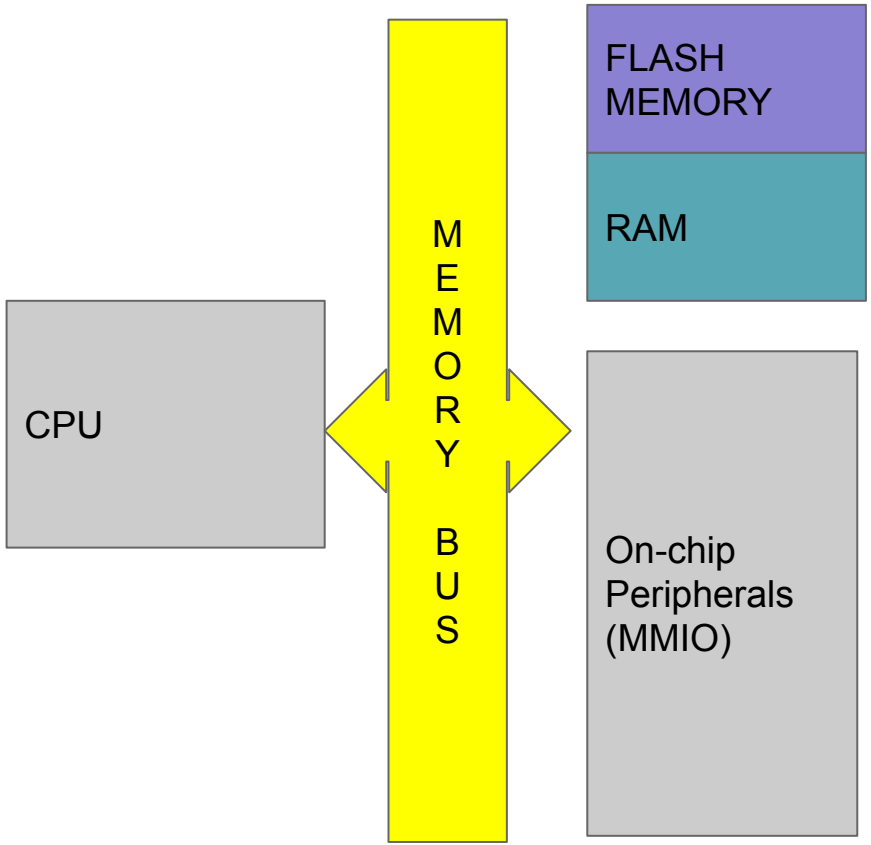


OS-based firmware

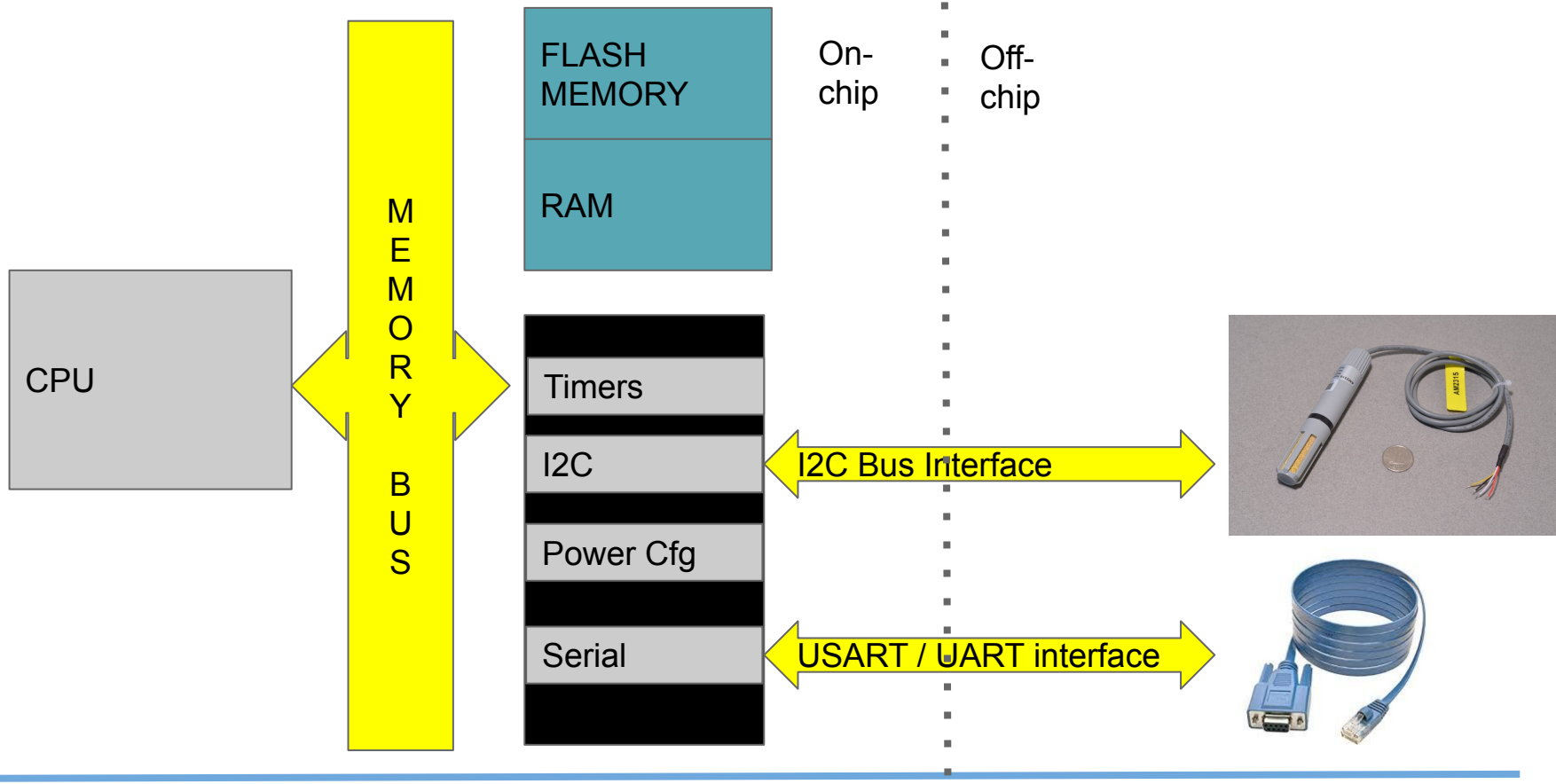


Blobs

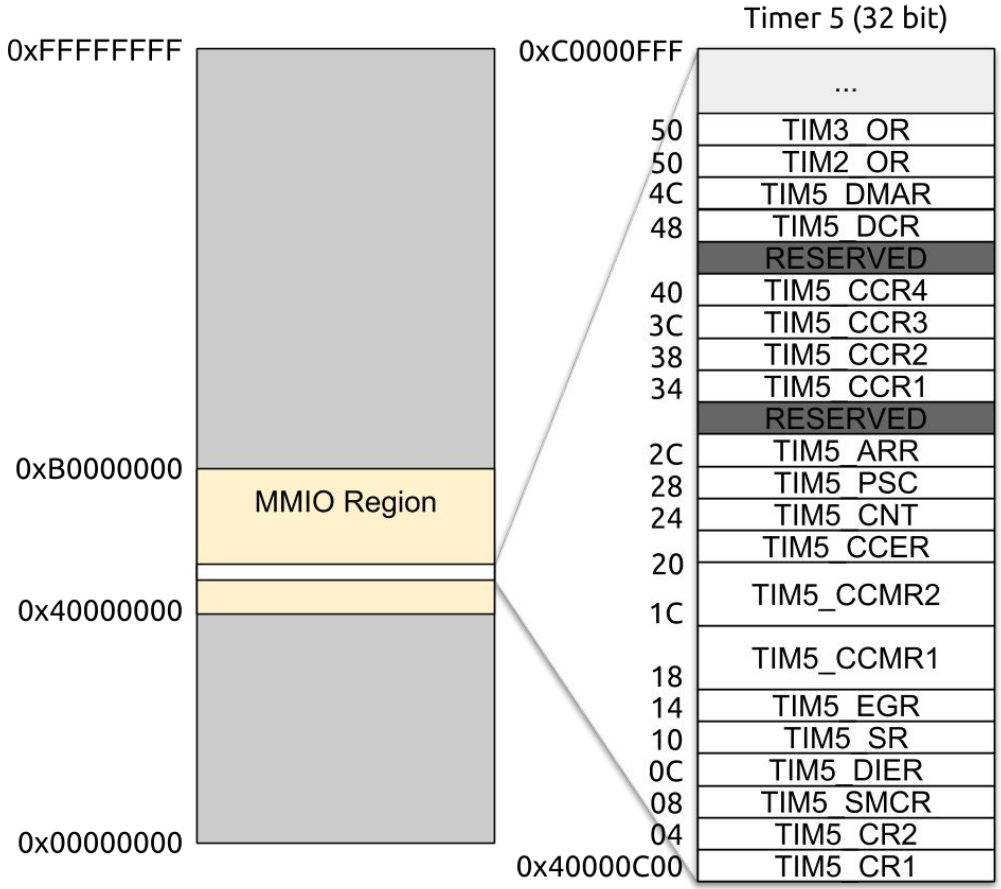
Peripherals are Hard!



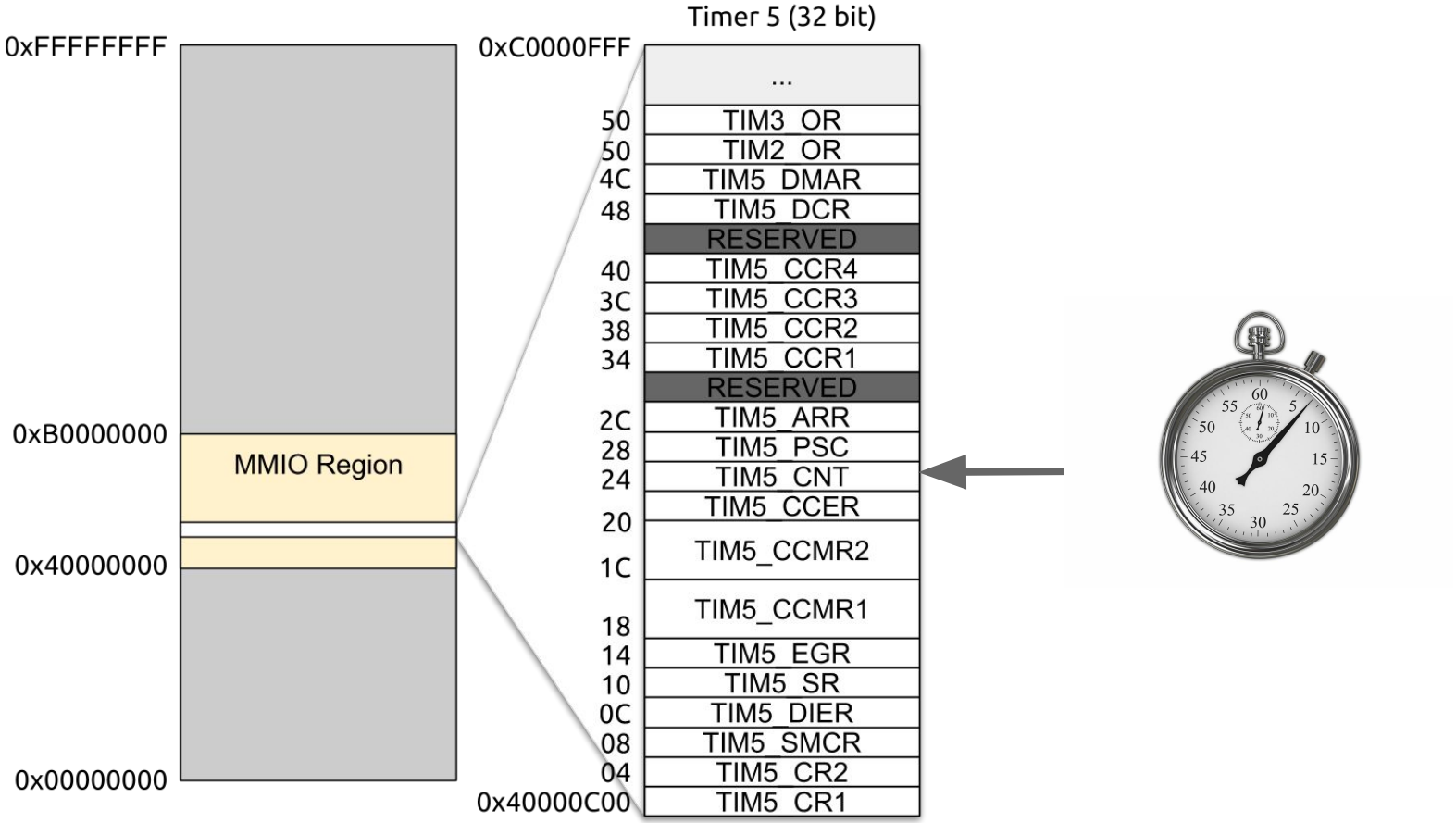
Peripherals are Hard!



Peripherals are Hard!



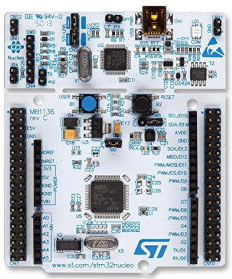
Peripherals are Hard!

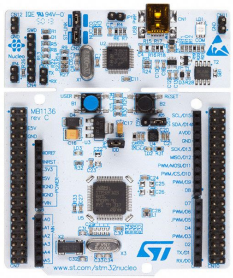


Peripherals are Hard!

Offset	Register Name
0x0	Status
0x4	Data (RX and TX)
0x8	Baud Rate
0xC	Control 1
0x10	Control 2
0x14	Control 3
0x18	GTPR

STM32L152 Serial port





Offset	Register Name
0x0	Control 1
0x4	Control 2
0x8	Control 3
0xC	Baud Rate
0x10	GTPR
0x14	RTOR
… …	… … …
0x20	Data RX
0x24	Data TX

STM32F072 Serial port

Peripherals are Hard!

- Obtained a dataset of Cortex-M memory layouts as used by debuggers (SVD files)
- Data self-published by vendors (and is therefore extremely incomplete)
- 463 distinct chip models, 13 vendors, 1592 unique peripherals
- Mainline QEMU supports 3 Cortex-M CPUs, and zero of the above dataset!

Emulation is Hard!

- Hardware-in-the-loop isn’t sufficient
  - One thread per device
  - One device reboot per execution

- Replay is not sufficient!
  - Can’t do fuzzing without input

Four Attributes of Ideal Re-Hosting

- **Virtual**
  - Does not require hardware at the time of emulation

Four Attributes of Ideal Re-Hosting

- Virtual
  - Does not require hardware at the time of emulation
- **Abstraction-less**
  - Does not rely on any aspect of the program

Four Attributes of Ideal Re-Hosting

- Virtual
  - Does not require hardware at the time of emulation
- Abstraction-less
  - Does not rely on any aspect of the program
- **Interactive**
  - Responds to stimulus as the original hardware would

Four Attributes of Ideal Re-Hosting

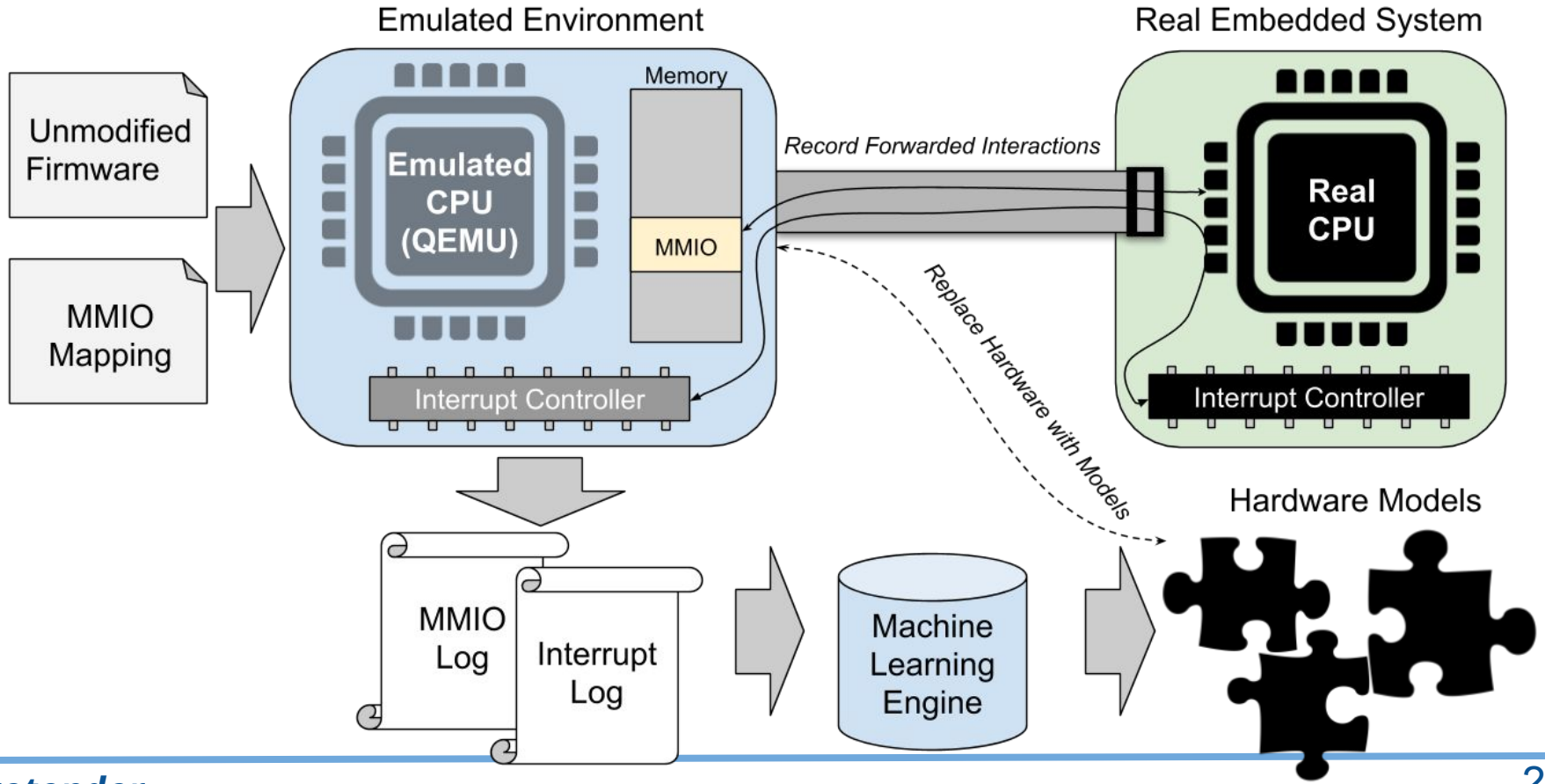
- Virtual
  - Does not require hardware at the time of emulation
- Abstraction-less
  - Does not rely on any aspect of the program
- Interactive
  - Responds to stimulus as the original hardware would
- **Automatic**
  - Requires a minimum of human intervention

Re-hosting is hard!

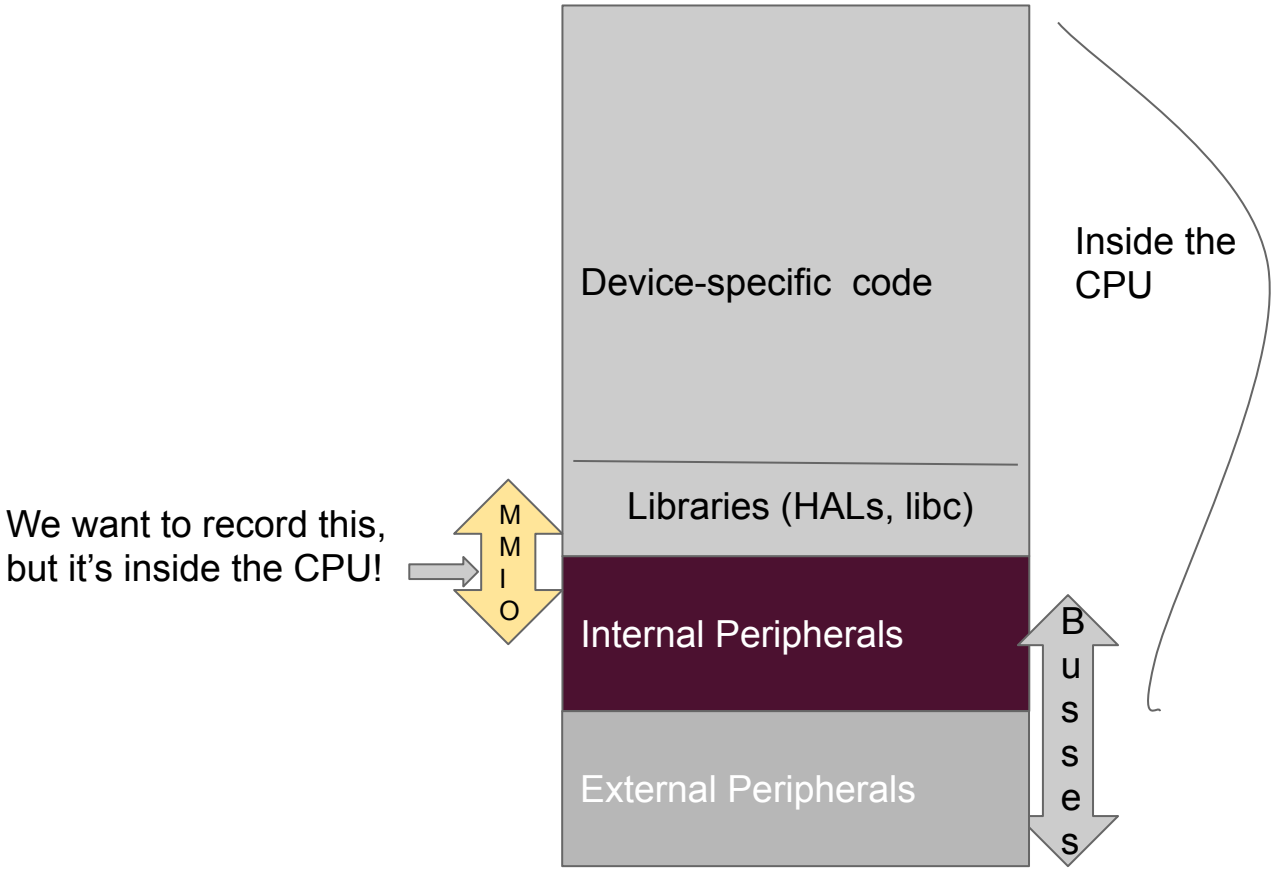
But are we doomed? Not yet.

Can we observe the real hardware, to build models for an emulator?

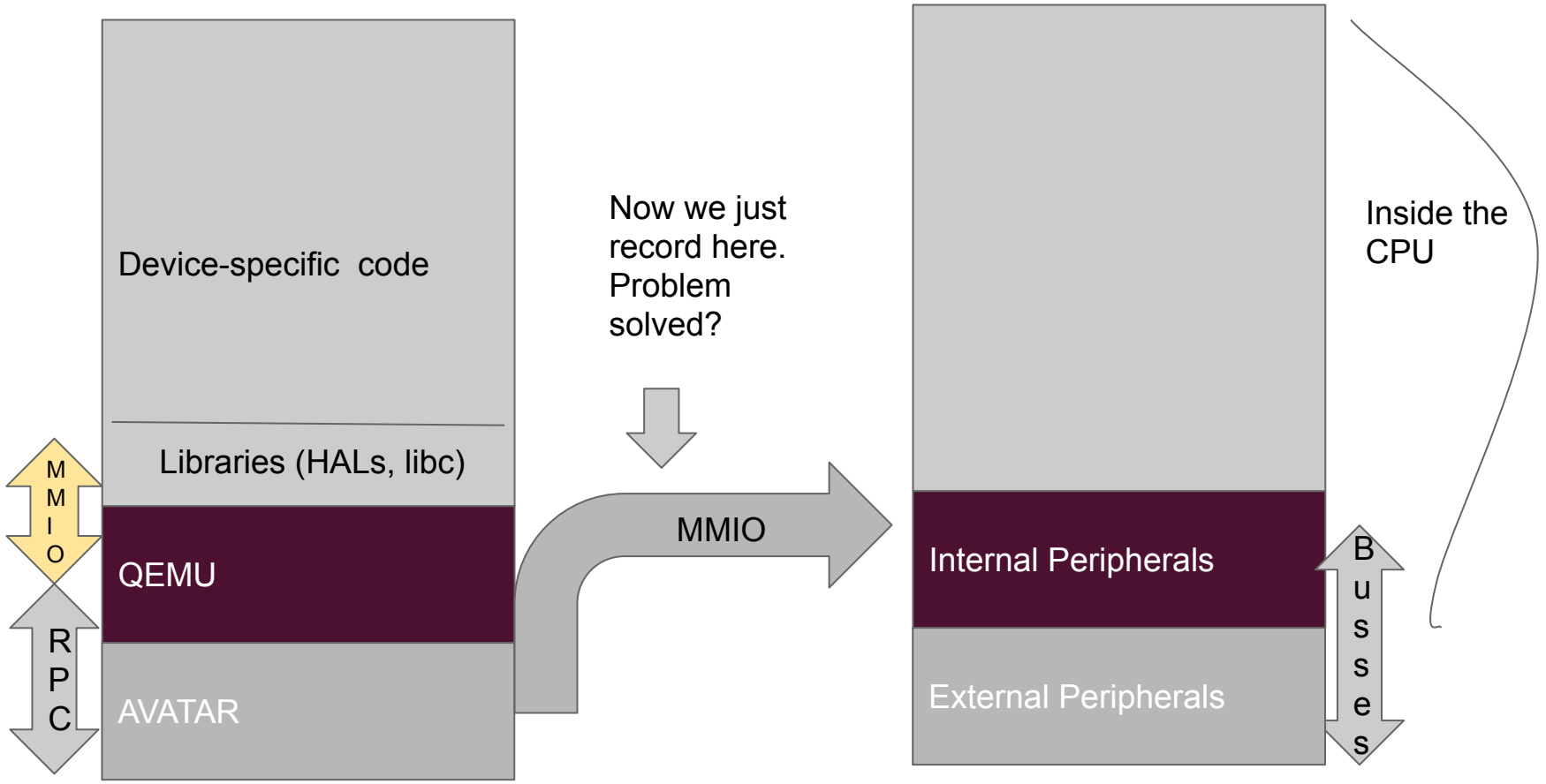
Pretender


Emulated Environment
Real Embedded System
Unmodified Firmware
MMIO Mapping
Emulated CPU (QEMU)
Memory
MMIO
Interrupt Controller
Record Forwarded Interactions
Real CPU
Interrupt Controller
Replace Hardware with Models
Hardware Models
MMIO Log
Interrupt Log
Machine Learning Engine

Recording


Device-specific code
Libraries (HALs, libc)
Internal Peripherals
External Peripherals
Inside the CPU
We want to record this, but it's inside the CPU!
MMIO
Busses

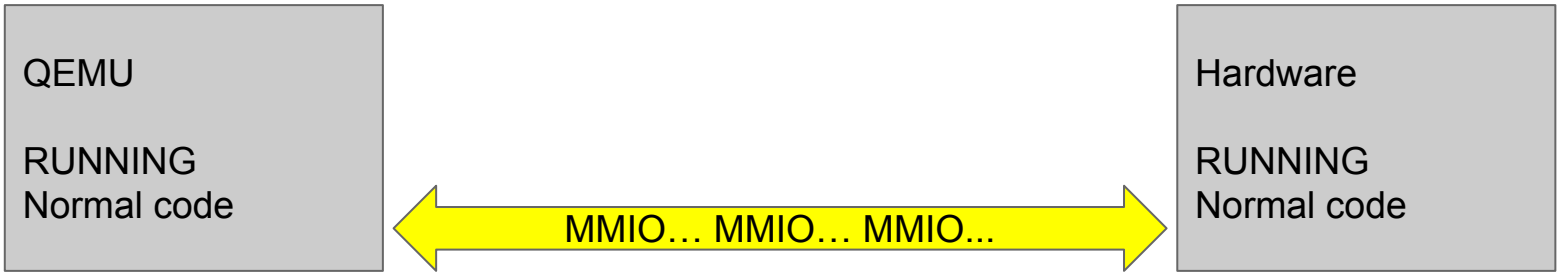
Recording


Device-specific code
Libraries (HALs, libc)
QEMU
AVATAR
MMIO
RPC
Now we just record here. Problem solved?
MMIO
Internal Peripherals
External Peripherals
Busses
Inside the CPU

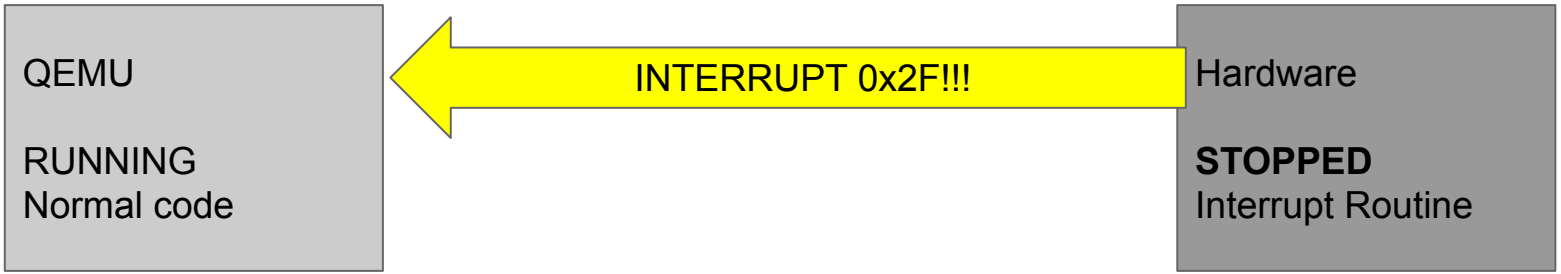
Interrupts

- The current version of Avatar does not handle interrupts at all, but almost every firmware requires them
- Previous approaches leverage chip-specific hardware to observe interrupts
- Timing, masking, ordering, …. Cause extreme complications

Interrupt Recording



Interrupt Recording



Interrupt Recording

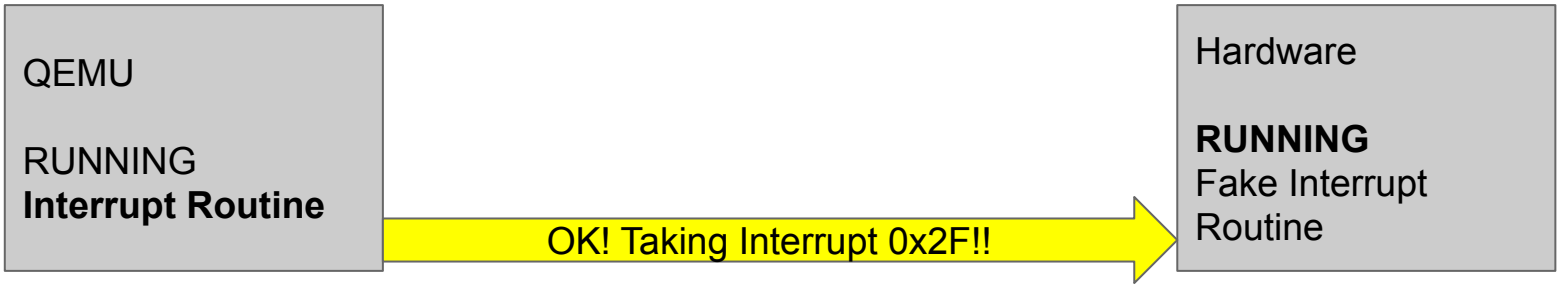
QEMU

RUNNING
Normal code

Hardware

STOPPED
Fake Interrupt Routine

Interrupt Recording



Interrupt Recording

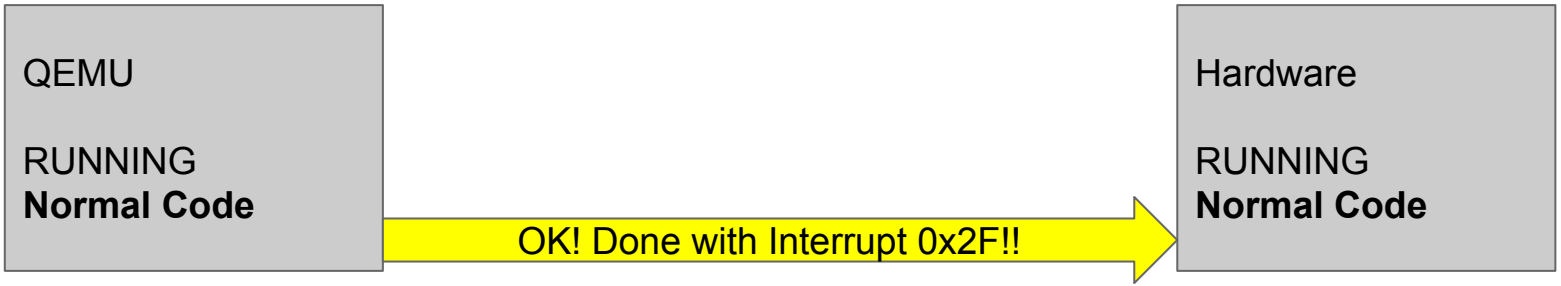
QEMU

RUNNING
Interrupt Routine

Hardware

RUNNING
Fake Interrupt
Routine

Interrupt Recording



Modeling

1. Figure out which groups of memory locations are distinct “peripherals”
2. Figure out which interrupts those peripherals fire, and under which conditions
3. Assign a model to each location within the peripheral

Grouping Peripherals

Op.	Address	Value
READ	0x40000004	0x1000
WRITE	0x40010024	0x0
READ	0x40002000	0x8000
WRITE	0x40020004	0x1
READ	0x40000008	0x8
READ	0x40003000	0x10
…	… … …	… … …

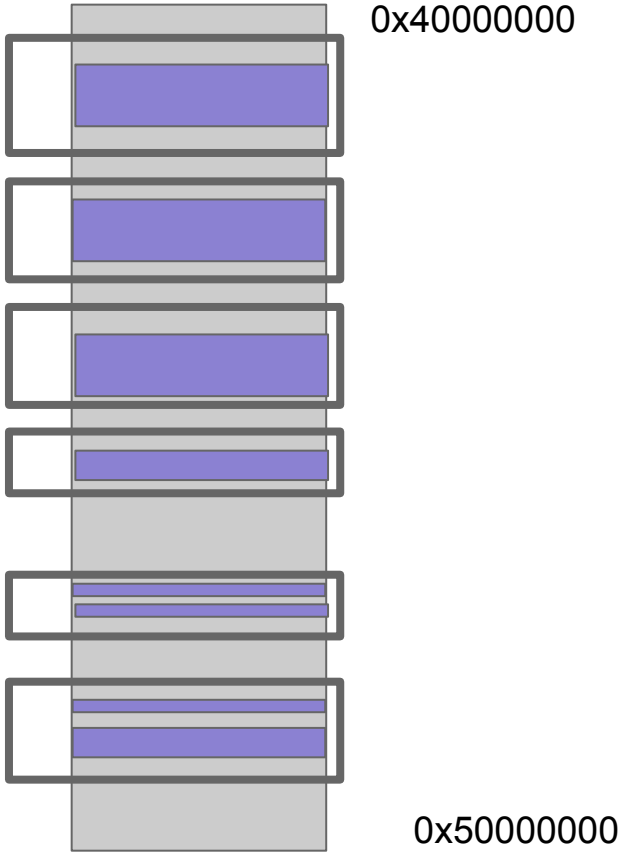
Grouping Peripherals

Op.	Address	Value
READ	**0x40000004**	0x1000
WRITE	**0x40010024**	0x0
READ	**0x40002000**	0x8000
WRITE	**0x40020004**	0x1
READ	**0x40000008**	0x8
READ	**0x40003000**	0x10
…	… … …	… … …

Grouping Peripherals

Op.	Address	Value
READ	**0x40000004**	0x1000
WRITE	**0x40010024**	0x0
READ	**0x40002000**	0x8000
WRITE	**0x40020004**	0x1
READ	**0x40000008**	0x8
READ	**0x40003000**	0x10
…	… … …	… … …

Clustering:



Associating Interrupts

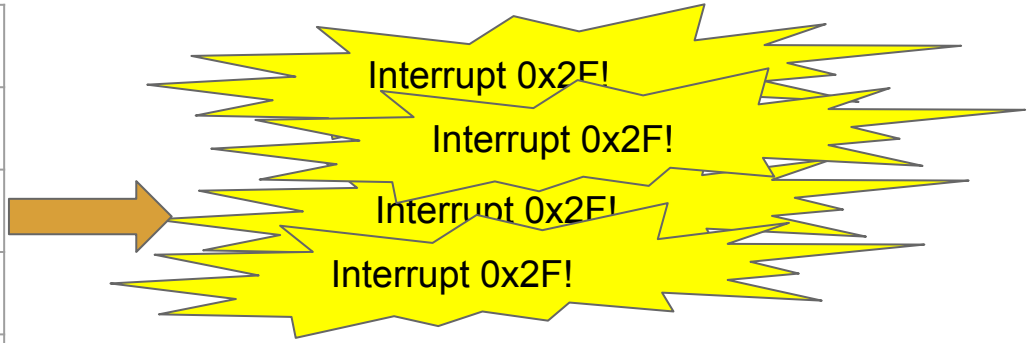
Offset	Value
0x0	????????
0x4	????????
0x8	????????
0xC	????????
0x10	????????

Associating Interrupts

Offset	Value
0x0	????????
0x4	**0xDEADBEEF**
0x8	????????
0xC	????????
0x10	????????

Associating Interrupts

Offset	Value
0x0	????????
0x4	0xDEADBEEF
0x8	????????
0xC	????????
0x10	????????



Associating Interrupts

ISR ENTER	**0x2F**
READ	Peripheral 1
WRITE	Peripheral 4
READ	Peripheral 4
WRITE	Peripheral 1
READ	Peripheral 4
READ	Peripheral 4
READ	Peripheral 4
WRITE	Peripheral 4
WRITE	Peripheral 1
ISR EXIT	**0x2F**

Associating Interrupts

ISR ENTER	**0x2F**
READ	Peripheral 1
WRITE	Peripheral 4
READ	Peripheral 4
WRITE	Peripheral 1
READ	Peripheral 4
READ	Peripheral 4
READ	Peripheral 4
WRITE	Peripheral 4
WRITE	Peripheral 1
ISR EXIT	**0x2F**

Associating Interrupts

ISR ENTER	**0x2F**
READ	Peripheral 1
WRITE	Peripheral 4
READ	Peripheral 4
WRITE	Peripheral 1
READ	Peripheral 4
READ	Peripheral 4
READ	Peripheral 4
WRITE	Peripheral 4
WRITE	Peripheral 1
ISR EXIT	**0x2F**

Peripheral 4 generates Interrupt 0x2F!

Interrupt Trigger Inference

Op.	Offset	Value
WRITE	0x4	0xDEADBEEF
...	...	… … ...
ENTER		0x2F

Interrupt Trigger Inference

Op.	Offset	Value
WRITE	0x4	0xDEADBEEF
...	...	… … ...
ISR	ENTER	0x2F

WRITE	0x4	0xFACEBEEF
… ...	...	… … ...
ISR	ENTER	0x2F

WRITE	0x4	0x0000BEEF
… ...	...	… … ...
ISR	ENTER	0x2F

ISR	EXIT	0x2F
… ...	...	… … ...
WRITE	0x4	0xDEAD0000

Interrupt Trigger Inference

Op.	Offset	Value
WRITE	0x4	0xDEADBEEF
...	...	… … ...
ISR	ENTER	0x2F

WRITE	0x4	0xFACEBEEF
… ...	...	… … ...
ISR	ENTER	0x2F

The trigger for Interrupt 0x2F is 0x0000BEEF in offset 0x4!

WRITE	0x4	0x0000BEEF
… ...	...	… … ...
ISR	ENTER	0x2F

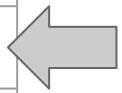
ISR	EXIT	0x2F
… ...	...	… … ...
WRITE	0x4	0xDEAD0000

Modeling MMIO

Offset	Register Model
0x0	????????
0x4	????????
0x8	????????
0xC	????????
0x10	????????

Modeling MMIO

Offset	Register Model
0x0	????????
0x4	????????
0x8	????????
0xC	????????
0x10	????????



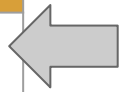
Offset	**Op.**	**Value**
0x0	READ	1
0x0	WRITE	42
0x0	READ	42
0x0	WRITE	56
0x0	READ	56

Modeling MMIO

Offset	Register Model
0x0	Storage Model
0x4	????????
0x8	????????
0xC	????????
0x10	????????

Modeling MMIO

Offset	Register Model
0x0	Storage Model
0x4	????????
0x8	????????
0xC	????????
0x10	????????



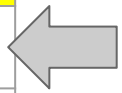
Offset	**Op.**	**Value**
0x4	WRITE	0x400
0x4	WRITE	0x800
0x4	WRITE	0x600
0x4	WRITE	0x1234
0x4	WRITE	0x5432

Modeling MMIO

Offset	Register Model
0x0	Storage Model
0x4	Write-Only Model
0x8	????????
0xC	????????
0x10	????????

Modeling MMIO

Offset	Register Model
0x0	Storage Model
0x4	Write-Only Model
0x8	????????
0xC	????????
0x10	????????



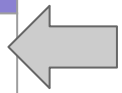
Offset	Op.	Value
0x8	READ	0x1
0x8	READ	0x2
0x8	READ	0x4
0x8	READ	0x1
0x8	READ	0x2
0x8	READ	0x4

Modeling MMIO

Offset	Register Model
0x0	Storage Model
0x4	Write-Only Model
0x8	Pattern Model
0xC	????????
0x10	????????

Modeling MMIO

Offset	Register Model
0x0	Storage Model
0x4	Write-Only Model
0x8	Pattern Model
0xC	????????
0x10	????????



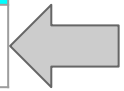
Offset	**Op.**	**Value**
0xC	READ	0x12
0xC	READ	0x48
0xC	READ	0x96
0xC	READ	0x123
0xC	WRITE	0
0XC	READ	0x24
0xC	READ	0x48
0xC	READ	0x96

Modeling MMIO

Offset	Register Model
0x0	Storage Model
0x4	Write-Only Model
0x8	Pattern Model
0xC	Increasing Model
0x10	????????

Modeling MMIO

Offset	Register Model
0x0	Storage Model
0x4	Write-Only Model
0x8	Pattern Model
0xC	Increasing Model
0x10	????????



Offset	Op.	Value
0x10	READ	“I”
0x10	READ	“L”
0x10	READ	“o”
0x10	READ	“v”
0x10	READ	“e”
0x10	READ	“D”
0x10	READ	“o”
0x10	READ	“l”
0x10	READ	“p”
0x10	READ	“h”
0x10	READ	“i”
0x10	READ	“n”
0x10	READ	“s”
0x10	READ	“!”
0x10	WRITE	“O”
0x10	WRITE	“K”

Modeling MMIO

Offset	Register Model
0x0	Storage Model
0x4	Write-Only Model
0x8	Pattern Model
0xC	Increasing Model
0x10	State Approximation

State Approximation

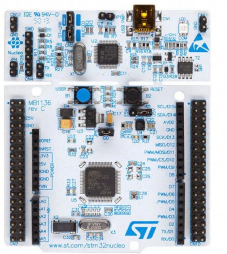
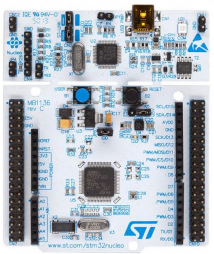
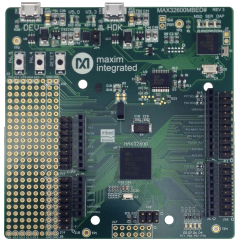
- Remaining locations typically represent state held by the hardware or physical world
- Can we recover the state machine? No:
  - No countable states, no state transitions, no state probabiliites
- Can we just guess? No.
  - Many firmware samples and libraries will not tolerate errors!

Fake it until we make it!

- Consider writes to the peripheral to change its “state”.
- When a value is read, return the next value of that location, except if it is in a different “state”
- When a write occurs, move to the next state where the same value was written
  - Seek backward if we don’t find one
  - Missing values are filled in from the most recent value

Evaluation

- Constructed 6 test firmware samples based on the mbed framework
- Used w/ 3 different boards
- Mixes of interrupts, stateful peripherals, etc

Evaluation

- 3 samples are fully-interactive, and have functionality not seen during recording, as well as synthetic vulnerabilities
- Replace analyst-chosen source of input with external input source
- Now we can drive the firmware like console programs

Evaluation

Firmware Name	Peripherals	Blocks Executed			
		Rec.	Null Model	SA	Fuzzing
Nucleo L152RE					
blink_led					
read_hyperterminal					
i2c_master					
button_interrupt					
thermostat *(custom)*					
rf_door_lock *(custom)*					
Nucleo F072RB					
blink_led					
read_hyperterminal					
i2c_master					
button_interrupt					
thermostat *(custom)*					
rf_door_lock *(custom)*					
MAX32600MBED					
blink_led					
read_hyperterminal					
i2c_master					
button_interrupt					
thermostat *(custom)*					
rf_door_lock *(custom)*					

Evaluation

Firmware Name	Peripherals	Blocks Executed			
		Rec.	Null Model	SA	Fuzzing
Nucleo L152RE					
blink_led	Timer, GPIO				
read_hyperterminal	Timer, GPIO, UART				
i2c_master	Timer, I2C, AM3215				
button_interrupt	Timer, GPIO, Button				
thermostat *(custom)*	Timer, I2C, AM3215				
rf_door_lock *(custom)*	Timer, GPIO, Radio,				
Nucleo F072RB					
blink_led	Timer, GPIO				
read_hyperterminal	Timer, GPIO, UART				
i2c_master	Timer, I2C, AM3215				
button_interrupt	Timer, GPIO, Button				
thermostat *(custom)*	Timer, I2C, AM3215				
rf_door_lock *(custom)*	Timer, GPIO, Radio,				
MAX32600MBED					
blink_led	Timer, GPIO				
read_hyperterminal	Timer, GPIO, UART				
i2c_master	Timer, I2C, AM3215				
button_interrupt	Timer, GPIO, Button				
thermostat *(custom)*	Timer, I2C, AM3215				
rf_door_lock *(custom)*	Timer, GPIO, Radio,				

Evaluation

Firmware Name	Peripherals	Blocks Executed			
		Rec.	Null Model	SA	Fuzzing
Nucleo L152RE					
blink_led	Timer, GPIO	218	86		
read_hyperterminal	Timer, GPIO, UART	545	85		
i2c_master	Timer, I2C, AM3215	1185	61		
button_interrupt	Timer, GPIO, Button	344	68		
thermostat *(custom)*	Timer, I2C, AM3215	1263	62		
rf_door_lock *(custom)*	Timer, GPIO, Radio,	665	87		
Nucleo F072RB					
blink_led	Timer, GPIO	405	117		
read_hyperterminal	Timer, GPIO, UART	828	102		
i2c_master	Timer, I2C, AM3215	1572	103		
button_interrupt	Timer, GPIO, Button	362	103		
thermostat *(custom)*	Timer, I2C, AM3215	1662	103		
rf_door_lock *(custom)*	Timer, GPIO, Radio,	960	102		
MAX32600MBED					
blink_led	Timer, GPIO	280	9		
read_hyperterminal	Timer, GPIO, UART	514	8		
i2c_master	Timer, I2C, AM3215	941	8		
button_interrupt	Timer, GPIO, Button	188	8		
thermostat *(custom)*	Timer, I2C, AM3215	1009	8		
rf_door_lock *(custom)*	Timer, GPIO, Radio,	692	8		

Evaluation

Firmware Name	Peripherals	Blocks Executed			
		Rec.	Null Model	SA	Fuzzing
Nucleo L152RE					
blink_led	Timer, GPIO	218	86	218	
read_hyperterminal	Timer, GPIO, UART	545	85	545	
i2c_master	Timer, I2C, AM3215	1185	61	1185	
button_interrupt	Timer, GPIO, Button	344	68	314	
thermostat *(custom)*	Timer, I2C, AM3215	1263	62	1261	
rf_door_lock *(custom)*	Timer, GPIO, Radio,	665	87	665	
Nucleo F072RB					
blink_led	Timer, GPIO	405	117	405	
read_hyperterminal	Timer, GPIO, UART	828	102	828	
i2c_master	Timer, I2C, AM3215	1572	103	1572	
button_interrupt	Timer, GPIO, Button	362	103	362	
thermostat *(custom)*	Timer, I2C, AM3215	1662	103	1662	
rf_door_lock *(custom)*	Timer, GPIO, Radio,	960	102	960	
MAX32600MBED					
blink_led	Timer, GPIO	280	9	280	
read_hyperterminal	Timer, GPIO, UART	514	8	514	
i2c_master	Timer, I2C, AM3215	941	8	942	
button_interrupt	Timer, GPIO, Button	188	8	188	
thermostat *(custom)*	Timer, I2C, AM3215	1009	8	1009	
rf_door_lock *(custom)*	Timer, GPIO, Radio,	692	8	692	

Evaluation

Firmware Name	Peripherals	Blocks Executed			
		Rec.	Null Model	SA	Fuzzing
Nucleo L152RE					
blink_led	Timer, GPIO	218	86	218	n/a
read_hyperterminal	Timer, GPIO, UART	545	85	545	636
i2c_master	Timer, I2C, AM3215	1185	61	1185	n/a
button_interrupt	Timer, GPIO, Button	344	68	314	n/a
thermostat *(custom)*	Timer, I2C, AM3215	1263	62	1261	1276
rf_door_lock *(custom)*	Timer, GPIO, Radio,	665	87	665	758
Nucleo F072RB					
blink_led	Timer, GPIO	405	117	405	n/a
read_hyperterminal	Timer, GPIO, UART	828	102	828	999
i2c_master	Timer, I2C, AM3215	1572	103	1572	n/a
button_interrupt	Timer, GPIO, Button	362	103	362	n/a
thermostat *(custom)*	Timer, I2C, AM3215	1662	103	1662	1918
rf_door_lock *(custom)*	Timer, GPIO, Radio,	960	102	960	972
MAX32600MBED					
blink_led	Timer, GPIO	280	9	280	n/a
read_hyperterminal	Timer, GPIO, UART	514	8	514	668
i2c_master	Timer, I2C, AM3215	941	8	942	n/a
button_interrupt	Timer, GPIO, Button	188	8	188	n/a
thermostat *(custom)*	Timer, I2C, AM3215	1009	8	1009	1066
rf_door_lock *(custom)*	Timer, GPIO, Radio,	692	8	692	712

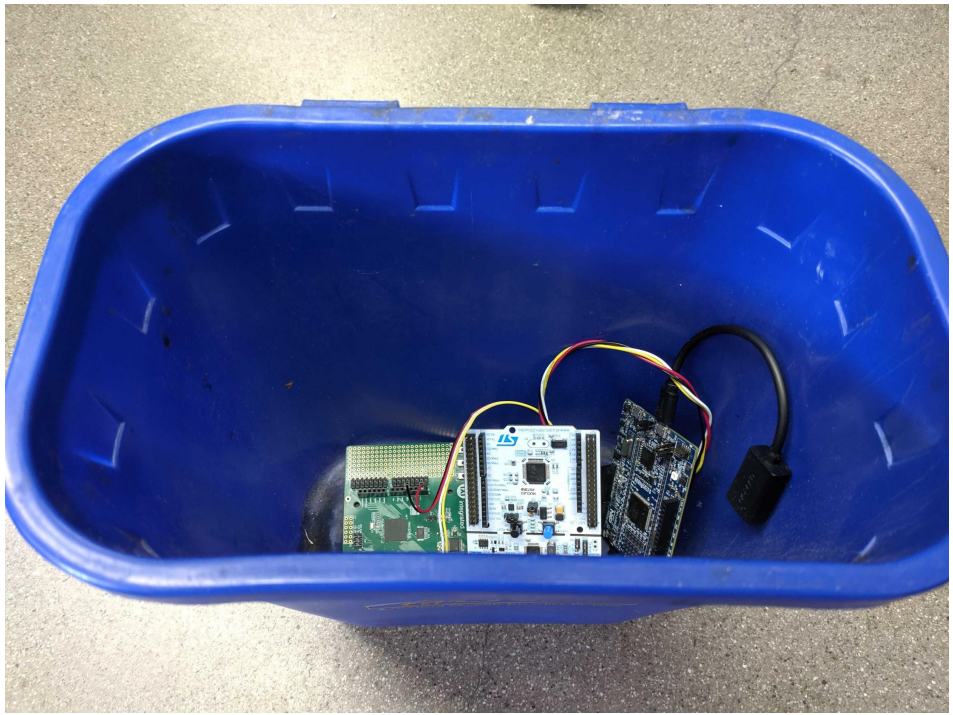
Limitations

- DMA: We can’t record what we can’t observe
- The limits of state approximation:
- Frequent interrupts cause recording issues

Next Steps

- Recording is tricky, can we go without?
- Static analysis to locate DMA and disambiguate internal/external peripherals
- Relax “abstraction-less”, find abstractions in blobs
  -

Thank you!



https://github.com/ucsb-seclab/pretender