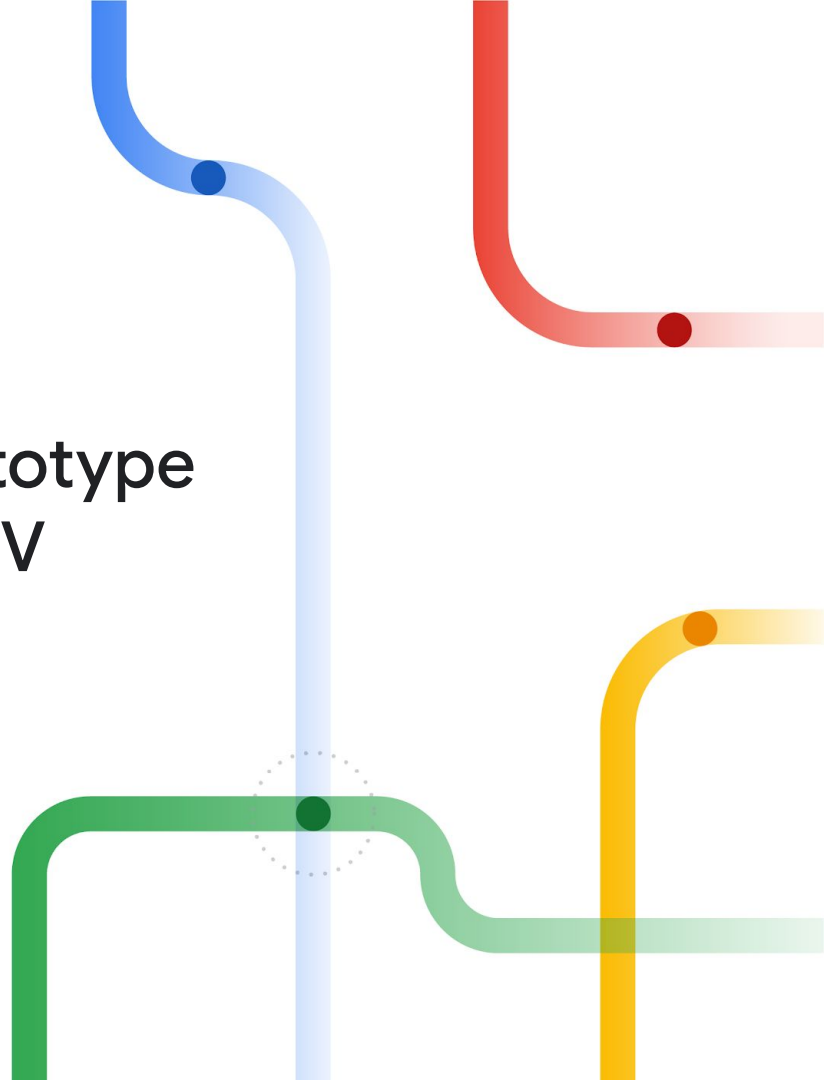


# Springbok

## Using Renode and IREE to prototype and develop ML models on RVV

Michael Gielda (Antmicro), Adam Jesionowski (Google)



# Agenda

- 01 Low Power ML on RISC-V
- 02 Introduction to Renode
- 03 System Co-design  
with Renode

01

# Low Power Machine Learning on RISC-V

# Springbok

Springbok is an RISC-V core with the Vector extension (RVV) that runs machine learning (ML) workloads

Part of the AmbiML project to create an open-source ML development ecosystem centered on privacy and security

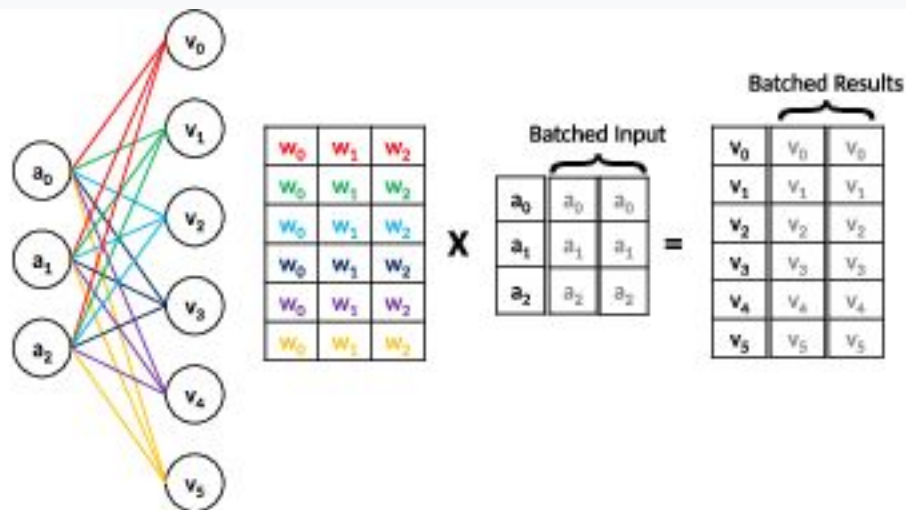
<https://github.com/AmbiML/iree-rv32-springbok>



# RVV for ML Acceleration

Machine learning relies heavily on matrix multiply and add operations suitable for running with a vector unit

Springbok runs the ML models as well as other vectorizable components (e.g. image manipulation)



# Python to RVV

The majority of machine learning modelling is performed in Python using frameworks like PyTorch, Tensorflow, or JAX

But Springbok is a bare-metal environment, we can't run a Python interpreter!

Solution: IREE



# IREE

ML toolchain capable of transforming Python models through a series of intermediate representations (IR) down into LLVM

These transformations enable optimizations and the ability to target and scale across heterogeneous architectures, from servers with GPUs to embedded environments

<https://github.com/google/iree>

## Intermediate Representation Execution Environment

# First step: MLIR

Multi-Level Intermediate Representation

Element-wise multiply of two 1024-element i32 vectors:

```
func @simple_mul(%arg0: tensor<1024xi32>, %arg1: tensor<1024xi32>) -> tensor<1024xi32>
{
    %0 = "mhlo.multiply"(%arg0, %arg1) : (tensor<1024xi32>, tensor<1024xi32>) ->
    tensor<1024xi32>
    return %0 : tensor<1024xi32>
}
```



# Invoke IREE with RVV flags

IREE compiler LLVM flags:

```
-iree-llvm-target-triple=riscv32-pc-linux-elf  
-iree-llvm-target-cpu=generic-rv32  
-iree-llvm-target-cpu-features=+m,+f,+zv1512b,+zve32x  
-iree-llvm-target-abi=ilp32  
-riscv-v-vector-bits-min=512  
-riscv-v-fixed-length-vector-lmul-max=8
```

Runtime LLVM RISC-V flags:

```
-march=rv32imf_zv1512b_zve32x
```

# Output: RVV

```
vsetivli      zero,16,e32,m1,ta,mu  
vle32.v v8,(a4)  
add      a4,a3,a1  
vle32.v v9,(a4)  
vmul.vv v8,v9,v8  
add      a4,a0,a1  
vse32.v v8,(a4)  
...
```

# Springbok HAL

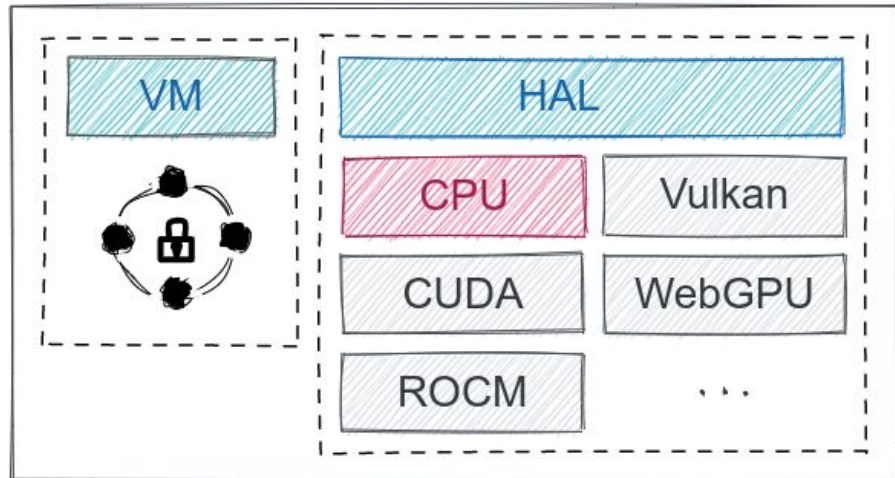
IREE's output consists of a virtual machine and the compiled ML output

It needs a Hardware Abstraction Layer (HAL) to operate on RISC-V and a scheduler

Our code provides an example of bare-metal execution on RISC-V

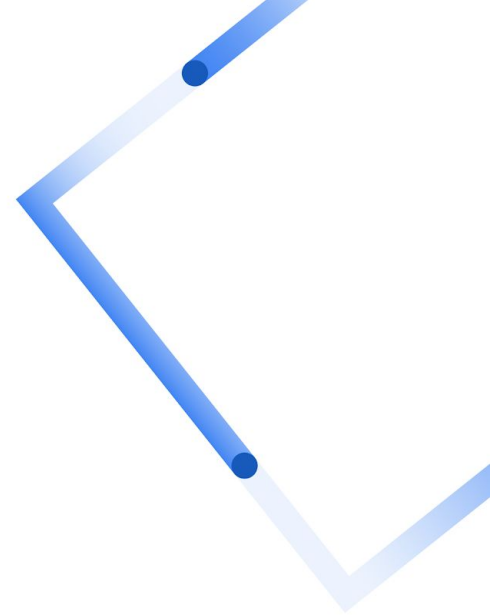
## IREE Runtime

~25-150KB



02

# Introduction to Renode



# What is Renode

Renode is an open source simulation framework by Antmicro focusing on developer productivity and flexibility.

It simulates whole SoCs and boards, allowing you to run the same software as on hardware.

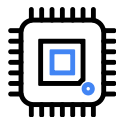
<https://www.renode.io>



# What can you do with Renode



IoT development,  
operating systems porting



Architectural exploration,  
pre-silicon development



Network protocols  
implementation and validation



ML development

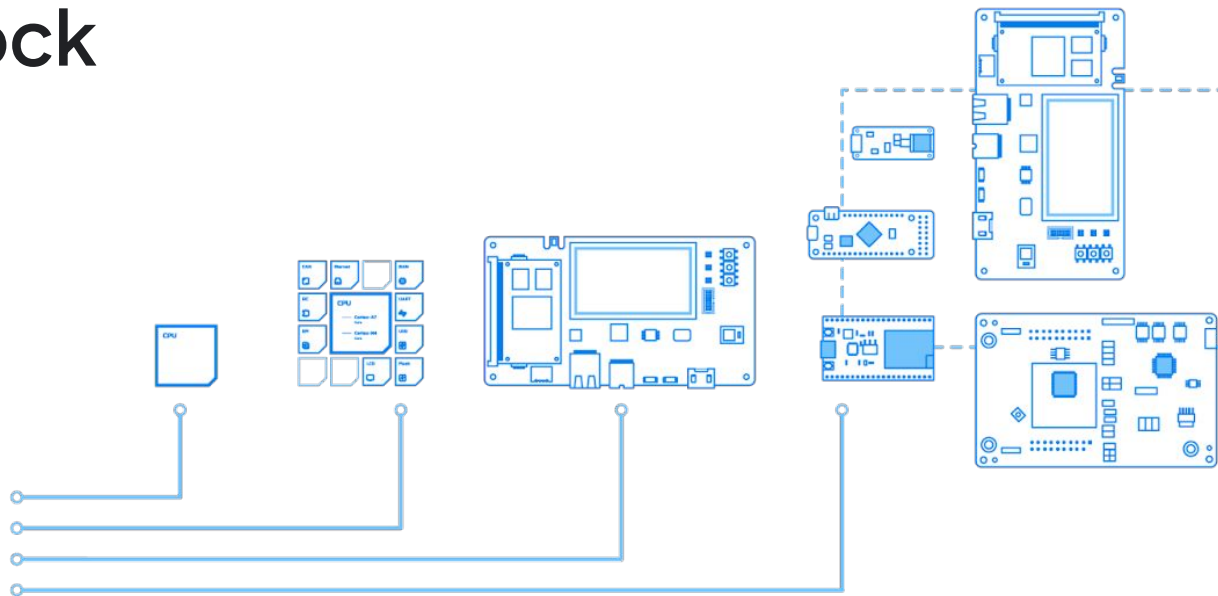


Continuous Integration,  
testing



Security analysis

# Building block nature



# Textual platform description

Renode assembles platforms from building blocks using text-based, layered .repl files:

- Great for prototyping: just edit a text file and reload (no need to rebuild)
- Enables easy support for lines of similar products
- Can be easily auto-generated - ideal for soft SoC support and ongoing development projects like Springbok

```
nvic: IRQControllers.NVIC @ sysbus  
0xE000E000  
-> cpu@0
```

```
cpu: CPU.CortexM @ sysbus  
cpuType: "cortex-m4"  
nvic: nvic
```

```
spi2: SPI.NRF52840_SPI @ sysbus  
0x40023000  
-> nvic@0x23
```

```
gpio0: GPIOPort.NRF52840_GPIO @ sysbus  
0x50000000
```

```
uart0: UART.NRF52840_UART @ sysbus  
0x40002000  
easyDMA: true  
-> nvic@2
```



# Model stubs

To enable needs-based, iterative platform development Renode supports model stubs in Python.

- Model parts that you really need
- Log or mock everything else
- Implement Python peripherals as one liners or in separate files

```
rcc: Python.PythonPeripheral @ sysbus  
0x40023800
```

```
    size: 0x400
```

```
    initable: true
```

```
    script: "0xFFFFFFFF if  
request.offset != 0x8 else  
0xFFFFFFFFFA"
```

```
pwrCr1: Python.PythonPeripheral @  
sysbus 0x40007000
```

```
    size: 0x4
```

```
    initable: true
```

```
    filename:  
"scripts/pydev/flipflop.py"
```

# Internal scripting language

Renode allows you to interact with every detail of the emulation via its CLI - the Monitor

- Monitor commands can be run as scripts
- Access to all peripherals and settings
- Control the emulation and tracing options
- Add your own commands on the fly

```
using sysbus
mach create $name

machine LoadPlatformDescription
    @platform.repl

emulation CreateSwitch "switch"
connector Connect ethmac switch
emulation CreateNetworkServer "server"
    "192.168.100.100"
connector Connect server switch

server StartTFTP 6069
server.tftp ServeFile $micropython
    "boot.bin"

showAnalyzer uart

macro reset
    ""
        sysbus LoadBinary $bios 0x0
        cpu PC 0x0
    ""
runMacro $reset
```

# Python support

Renode has a built-in Python runtime (IronPython)

- Complex event hooks with flow control
- Access to all emulation details
- Hook on:
  - Blocks of code
  - PC value, watchpoints
  - Interrupts
  - Memory/peripheral access
  - Network packets
  - Serial data
  - Whatever you want

```
(machine) include @notification_helper.py
(machine) set py_notification_hook
> """
> # recipient and get_recipients defined
  in external file
> for recipient not get_recipients():
>
  recipient.send_notification(self.line)
> """
(machine) uart AddLineHook "interesting
value" $py_notification_hook
```

```
(machine) cpu AddHookAtInterruptBegin
  "self.DebugLog('exception %d' %
  exceptionIndex)"
```

# Debugging with GDB

Renode allows you to debug applications running on emulated machines using GDB

- Uses the GDB remote protocol
- Breakpoints, watchpoints, stepping, memory access etc
- Virtual time does not progress when the emulated CPU is halted
- Multi-core debugging
- Disassembly via LLVM for runtime code analysis



# Logging & tracing

Extensive and customisable logging and tracing capabilities

- Easily log [executed functions](#) or [peripheral accesses](#)
- Precise filtering depending on the log source and target: console or log file
- Built-in graphical log analyser
- Various data sources - executed software, peripherals accesses / watchpoints, interrupts, network/UART data, framework events, user-defined events

Total results: 20

Id	Type	Timestamp	Source	Machine	Text
945	INFO	15:44:54.9808			Including script ./home/antmicro/hq-master/renoc
946	INFO	15:44:54.9935	sysbus	Mi-V	System bus created.
949	DEBUG	15:44:55.3968	debugArea	Mi-V	Segment size automatically calculated to value 64K
951	DEBUG	15:44:55.3979	smallRom	Mi-V	Segment size automatically calculated to value 64K
953	DEBUG	15:44:55.3979	flash	Mi-V	Segment size automatically calculated to value 64K
955	DEBUG	15:44:55.3979	ddr	Mi-V	Segment size automatically calculated to value 4Mi
1103	WARNING	15:44:55.5161	gpioInputs	Mi-V	Writing to an output GPIO pin #0
1104	WARNING	15:44:55.5161	gpioInputs	Mi-V	Writing to an output GPIO pin #1
1105	WARNING	15:44:55.5161	gpioInputs	Mi-V	Writing to an output GPIO pin #2
1252	DEBUG	15:44:55.7512	curbus	Mi-V	Loading FILE /tmp/rocode-593769/1a21fdaef1a81-645

Filters: Noisy, Debug, Info, Warning, Error

Search: [Search] [Reset]

Renode's flexible GDB support enables use IDEs like Visual Studio Code.

- Google Research



# OS-aware debugging

Developed with Google for this project, allows system-level awareness in debugging workflow.

Includes:

- system threads awareness (automatically handle context switches)
- context aware breakpoints
- debug symbols auto-reload on context switch
- awareness of virtual memory mapping changes on context switch

Relatively simple to port to other OSs (Zephyr port on the way now).

```
RENODE™

Renode, version 1.12.0.21189 (f1326194-202204071121)

(monitor) $bin=build/lnages/capdl-loader-lnage-arm-zynq7000; t @run.resc; sysbus
LoadSymbolsFrom @build/ kernel.elf; machine StarttdbServer
3333; cpu !shalt True;
(machine-0)
```

```
No DTB passed in from boot loader.
Looking for DTB in CP10 archive...found at 6ac484.
Loaded DTB from 6ac484,
paddr=[3c000..3efff]
ELF-loading lnage 'kernel' to 0
paddr=[0..3bfff]
vaddr=[e0000002..e003bfff]
virt_entry=e0000000
ELF-loading lnage 'capdl-loader' to 3f000
paddr=[3f000..3bfff]
vaddr=[10000..1b0fff]
virt_entry=10004
Enabling MMU and paging
Bootstrapping kernel
available phys memory regions: 1
[0..40000000]
reserve virt address space regions: 4
[e0000000..e003c000]
[e003c000..e003ee47]
[e003f000..e01e0000]
[ff000000..ffff0000]
Booting all finished, dropped to user space
client: what's the answer to 342 + 74 + 283 + 37 + 534 ?
```

```
(gdb) set4 wait-for-thread rootserver
Program received signal SIGTRAP, Trace/breakpoint trap.
0xe001eaa8 in ?? ()
(gdb) set4 switch-symbols rootserver
Reading symbols from /storage/canikes/build/capdl-loader...
(gdb) set4 tbreak rootserver main
(gdb) c
Continuing

Program received signal SIGTRAP, Trace/breakpoint trap.
main () at /storage/canikes/projects/capdl/capdl-loader-app/src/main.c:2124
2124 {
(gdb) l
2119     platsupport_serial_setup_boot_info_fallsafe();
2120 }
2121 #endif
2122 int main(void)
2123 {
2124     ZF LOGI("Starting CapDL Loader...");
2125     InitSystem(&capdl_spec);
2126     ZF LOGI("A RESET A_FG_G 'capdl Loader done, suspending...' A_RESET");
2127     set4_tcb_suspend( set4_capdl_thread_tcb);
2128     set4_tcb_tbreak rootserver main.c:2128
(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
main () at /storage/canikes/projects/capdl/capdl-loader-app/src/main.c:2128
2128     set4_tcb_suspend( set4_capdl_thread_tcb);
(gdb) set4 tbreak adder_... adder_adder_0_fault_handler_tcb adder_adder_0_0000_tcb
(gdb) set4 tbreak adder_adder_0_control_tcb
(gdb) c
Continuing

Program received signal SIGTRAP, Trace/breakpoint trap.
main () in init_code_slot (spec=0x0, mode=MOVE, cnode_id=0, cnode_slot=0x0) at /storage/canikes/projects/capdl/capdl-loader-app/src/main.c:1836
1836     ZF LOGI("ERR(error, "");
(gdb) set4 tbreak kernel
(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0xe001e05c in ?? ()
(gdb) bt
#0  arm_svc_syscall () at /storage/canikes/kernel/src/arch/arm32/traps.S:53
#1  0xe001e05c in ?? ()
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
(gdb) set4 thread
kernel
```

# Renode RISC-V support

Renode supports RV32 and RV64 with standard extensions, with multicore AMP and SMP processing.

Added support for Vector v1.0 extensions while working on Springbok support.

Support for custom instructions and CSRs, implemented natively in Renode, in Python or even in Verilog via Verilator!

## Python

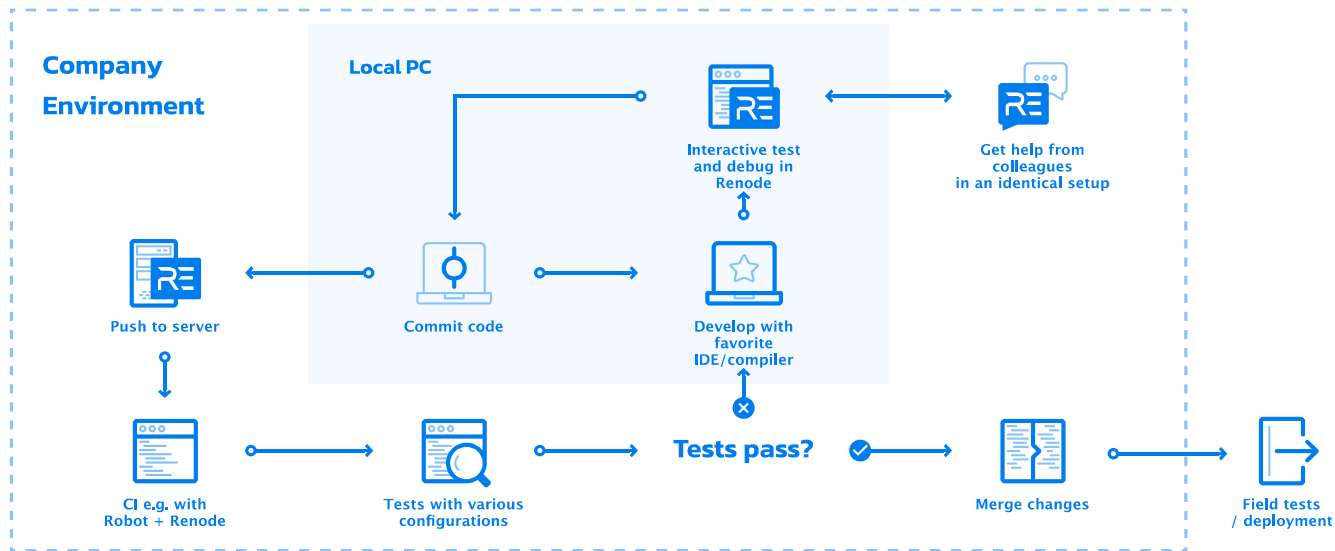
```
cpu InstallCustomInstructionHandlerFromString  
    "0001010001110000000000010010011"  
    "cpu.DebugLog('I'm running Python  
        here!')"
```

## C#

```
RegisterCSR((ulong)0x3e1,  
            () => counterValue,  
            value => UpdateCounter(value));  
  
InstallCustomInstruction(  
    pattern: "0000011-----sssss---dddd0001011",  
    handler: HandleMaskIrqInstruction);
```



# Development flow - CI



# Example CI - Zephyr Dashboard

[Renode Zephyr Dashboard](#) — massive automated CI system testing Zephyr targets running standard demos in Renode,

- Uses publicly available data to generate thousands of test cases
- Based on our open [dts2repl tool](#) for converting device trees into Renode's .repl files
- We are now at almost 140 passing boards!

Q Search...					
85 PASSED					
120 PASSED					
124 PASSED					
134 PASSED					
138 PASSED					
BOARD NAME	MICROPYTHON	TENSORFLOW LITE MICRO	PHILOSOPHERS	SHELL MODULE	HELLO WORLD
MM MM-FEATHER	BUILT	BUILT	BUILT	BUILT	BUILT
MM MM-SWIFTIO	BUILT	BUILT	BUILT	BUILT	BUILT
ARM V2M MPS2	BUILT	BUILT	BUILT	BUILT	BUILT
ARM V2M MPS2-AN521	BUILT	BUILT	BUILT	BUILT	BUILT
ARM V2M MPS2-AN521_ns	BUILT	BUILT	BUILT	BUILT	BUILT
ARM V2M MPS2-AN521_remote	BUILT	BUILT	BUILT	BUILT	BUILT
Arm MPS3-AN547	BUILT	BUILT	BUILT	BUILT	BUILT
Arm MPS3-AN547_ns	BUILT	BUILT	BUILT	BUILT	BUILT
MSP-EXP432P401R-LAUNCHXL	BUILT	BUILT	BUILT	BUILT	BUILT
Nuvoton NPCX7M6FB EVB	NOT BUILT	NOT BUILT	BUILT	BUILT	BUILT
Nuvoton NPCX9M6F EVB	NOT BUILT	NOT BUILT	BUILT	BUILT	BUILT
nRF21540-DK-NRF52840	PASSED	PASSED	PASSED	PASSED	PASSED
BLE400	NOT BUILT	PASSED	PASSED	PASSED	PASSED
BLE Nano	NOT BUILT	PASSED	PASSED	PASSED	PASSED
nRF51-VBLUno51	NOT BUILT	PASSED	PASSED	PASSED	PASSED
nRF51-DK-NRF51422	NOT BUILT	PASSED	PASSED	PASSED	PASSED
nRF51-Dongle-nRF51422	NOT BUILT	PASSED	PASSED	PASSED	PASSED
nRF52832-MDK	PASSED	PASSED	PASSED	PASSED	PASSED
nRF52833-DK-NRF52820	NOT BUILT	PASSED	PASSED	PASSED	PASSED
nRF52833-DK-NRF52833	PASSED	PASSED	PASSED	PASSED	PASSED
Electronut Labs Blip	PASSED	PASSED	PASSED	PASSED	PASSED
nRF52840-MDK	PASSED	PASSED	PASSED	PASSED	PASSED
Electronut Labs Papyr	PASSED	PASSED	PASSED	PASSED	PASSED
nRF52840-DK-NRF52811	NOT BUILT	PASSED	PASSED	PASSED	PASSED
nRF52840-DK-NRF52840	PASSED	PASSED	PASSED	PASSED	PASSED
nRF52840-Dongle-NRF52840	NOT BUILT	BUILT	BUILT	PASSED	BUILT
nRF52 Adafruit Feather	PASSED	PASSED	PASSED	PASSED	PASSED
BLE Nano 2	PASSED	PASSED	PASSED	PASSED	PASSED
Sparkfun nRF52832 breakout	PASSED	PASSED	PASSED	PASSED	PASSED
nRF52-VBLUno52	PASSED	PASSED	PASSED	PASSED	PASSED
nRF52-DK-NRF52805	NOT BUILT	PASSED	PASSED	PASSED	PASSED
nRF52-DK-NRF52810	NOT BUILT	PASSED	PASSED	PASSED	PASSED
nRF52-DK-NRF52832	PASSED	PASSED	PASSED	PASSED	PASSED
NRF5340-DK-NRF5340-application-MCU	BUILT	BUILT	BUILT	BUILT	BUILT
NRF5340-DK-NRF5340-application-MCU-Non-Secure	NOT BUILT	BUILT	BUILT	BUILT	BUILT

# Example CI - Springbok

The screenshot shows the GitHub interface for the repository `Ambiml / iree-rv32-springbok`. The repository is public. The navigation bar includes links for `<> Code`, `Issues`, `Pull requests`, `Actions`, `Projects`, `Wiki`, `Security`, and `Insights`. Below the navigation bar, there are buttons for `main` (selected), `1 branch`, and `0 tags`. To the right are buttons for `Go to file`, `Add file`, and `Code`. The main content area shows a commit by `PiotrZierhoffer` titled `Add GitHub actions (#3)`. The commit message is `7639879 3 days ago` with `8 commits`. The commit details show a list of files: `.github/workflow`, `build_tools`, `cmake`, and `samples`. A tooltip is displayed over the `build_tools` file, showing the message `All checks have passed` and `1 successful check`. The tooltip also shows a green checkmark and the text `Springbok README tests / test (push)` with a `Details` link. The commit message is `Initial Springbok commit.` and the commit date is `6 days ago`.

Ambiml / iree-rv32-springbok Public

<> Code Issues Pull requests Actions Projects Wiki Security Insights

main 1 branch 0 tags Go to file Add file Code

PiotrZierhoffer Add GitHub actions (#3) 7639879 3 days ago 8 commits

.github/workflow 3 days ago

build\_tools 3 days ago

cmake 6 days ago

samples Initial Springbok commit. 6 days ago

All checks have passed  
1 successful check

Springbok README tests / test (push) Successful... Details

03

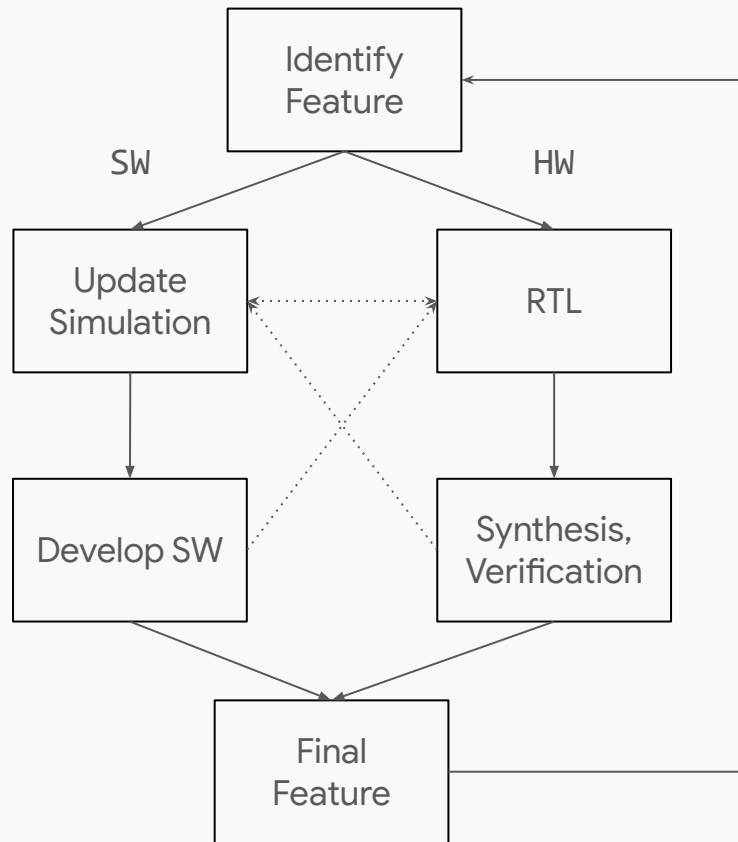
# System Co-design with Renode

# Hardware/Software Co-design for ML

ML operates on a wide variety of inputs and at a wide range of scales

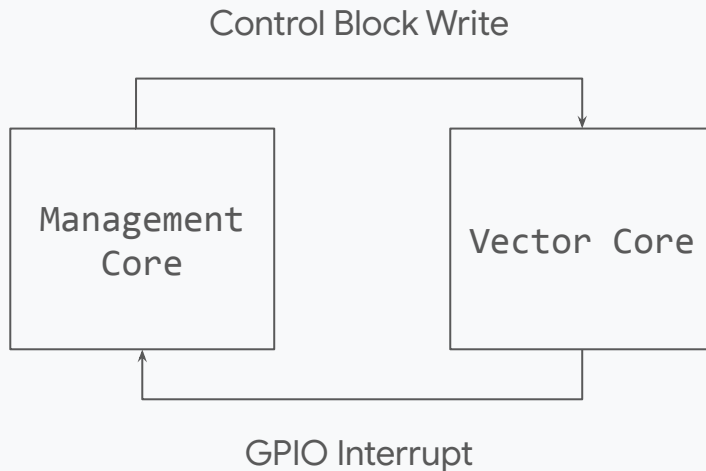
Co-design enables us to speed up the iteration loops on both hardware and software

Simulation is crucial here as it enables us to modify hardware at the speed of software



# Motivating Example

Springbok acts like a DSP in the larger system. We start it off by writing to an enable register, it runs the model, it halts. When it halts, we want to interrupt another core.



# Custom HALT (SW)

We utilize RISC-V's CUSTOM-3 (1111011) opcode for several purposes.

Our HALT is CUSTOM-3 where func3 is 3.

In our C runtime, the last instruction executed is the HALT.

```
_finish:  
...  
    .word 0x0000307B # custom3<func3=3>
```

# Custom HALT (Renode)

Renode provides an API for installing handlers when we hit a custom instruction.

In code we halt the core and trigger an interrupt.

## SpringbokRV32.cs

```
InstallCustomInstruction(  
    pattern: "-----1111011",  
    handler: HandleSpringbokCustom3);
```

```
// HandleSpringbokCustom3, func3=3  
Core.IsHalted = true;  
mode = Mode.Freeze | Mode.SwReset;  
irqsPending |= InterruptBits.Finish;  
IrqUpdate();
```



# Springbok MobileNetv1 Demo on Renode





# Thank You

Michael Gielda and Adam Jesionowski

<https://github.com/AmbiML/iree-rv32-springbok>

<https://www.renode.io>