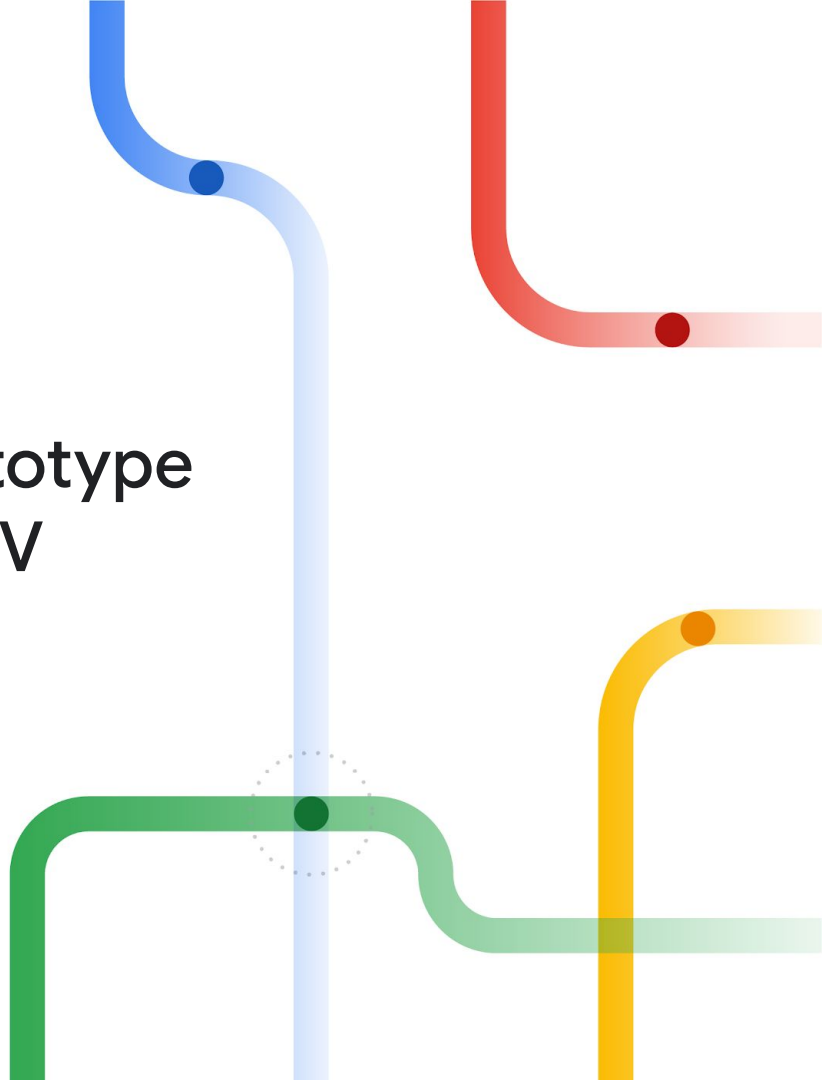
Springbok

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

Michael Gielda (Antmicro), Adam Jesionowski (Google)

Google Research



Agenda

01 Low Power ML on RISC-V

02 Introduction to Renode

03 System Co-design with Renode

Google Research

01

Low Power Machine Learning on RISC-V

Google Research

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

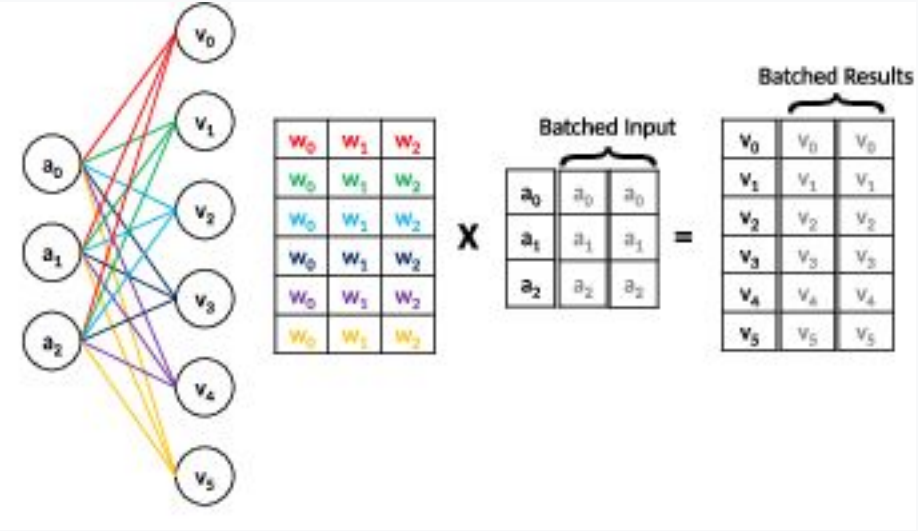
https://github.com/AmbiML/iree-rv32-springbok

Google Research

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)



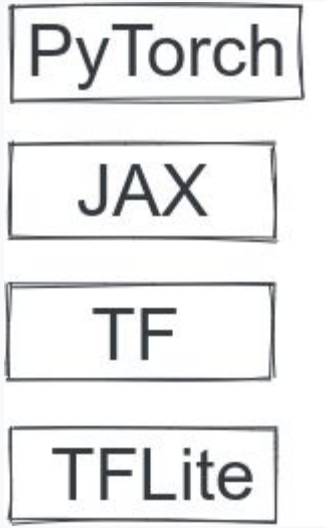
Google Research

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



Google Research

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

Google Research

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>
}
```

Google Research

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,+zvl512b,+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_zvl512b_zve32x
```

Google Research

Output: RVV

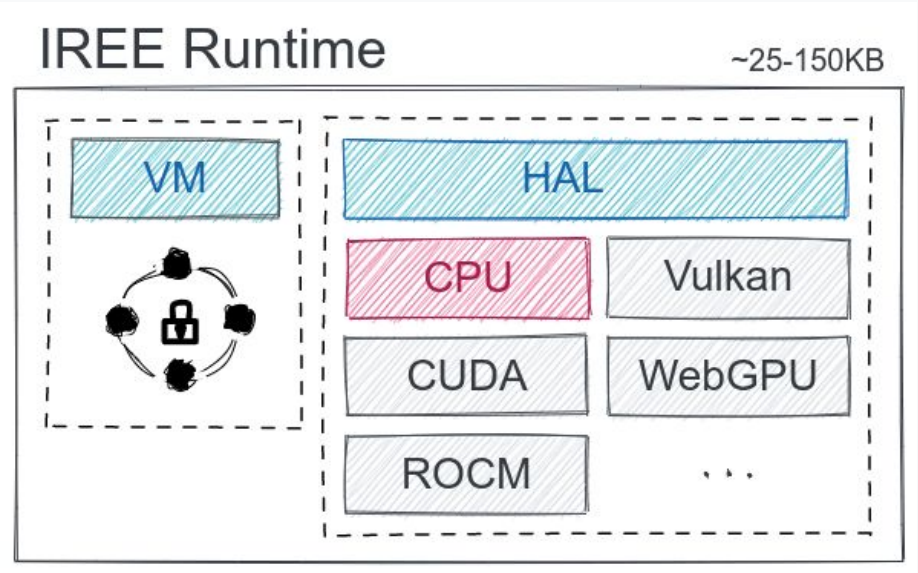
```
vsetivli        zero,16,e32,m1,ta,mu
vle32.v v8,(a4)
add     a4,a3,a1
vle32.v v9,(a4)
vmul.vv v8,v9,v8
add     a4,a0,a1
vse32.v v8,(a4)
…
```

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



Google Research

02

Introduction to Renode

Google Research

What is Renode

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

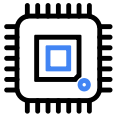
It simulates whole SoCs and boards, allowing you to run the same software as on hardware.

https://www.renode.io

Google Research

What can you do with Renode

IoT development,
operating systems porting



Architectural exploration,
pre-silicon development

Network protocols
implementation and validation

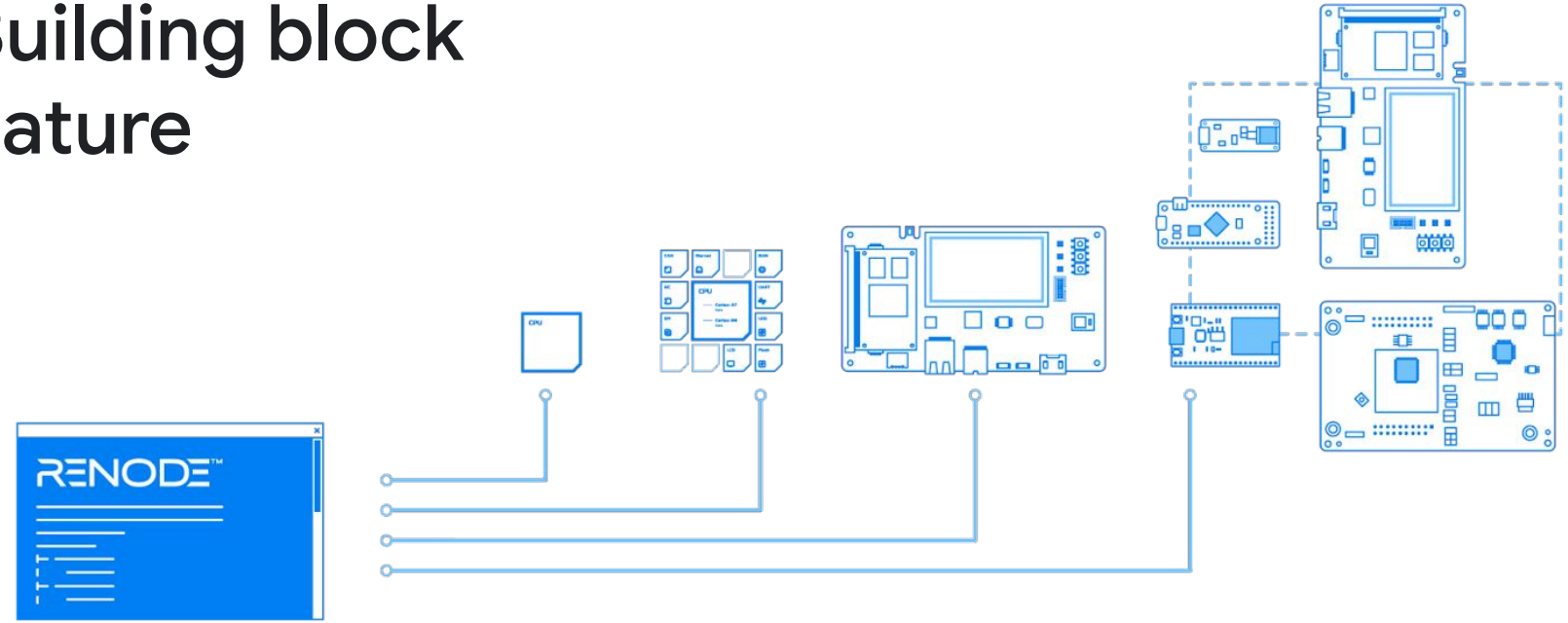
ML development

Continuous Integration,
testing

Security analysis

Google Research

Building block nature



Google Research

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
```

Google Research

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
0xFFFFFFFA”

pwrCr1: Python.PythonPeripheral @
sysbus 0x40007000
    size: 0x4
    initable: true
    filename:
"scripts/pydev/flipflop.py"
```

Google Research

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
```

Google Research

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)"
```

Google Research

Debugging with GDB

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

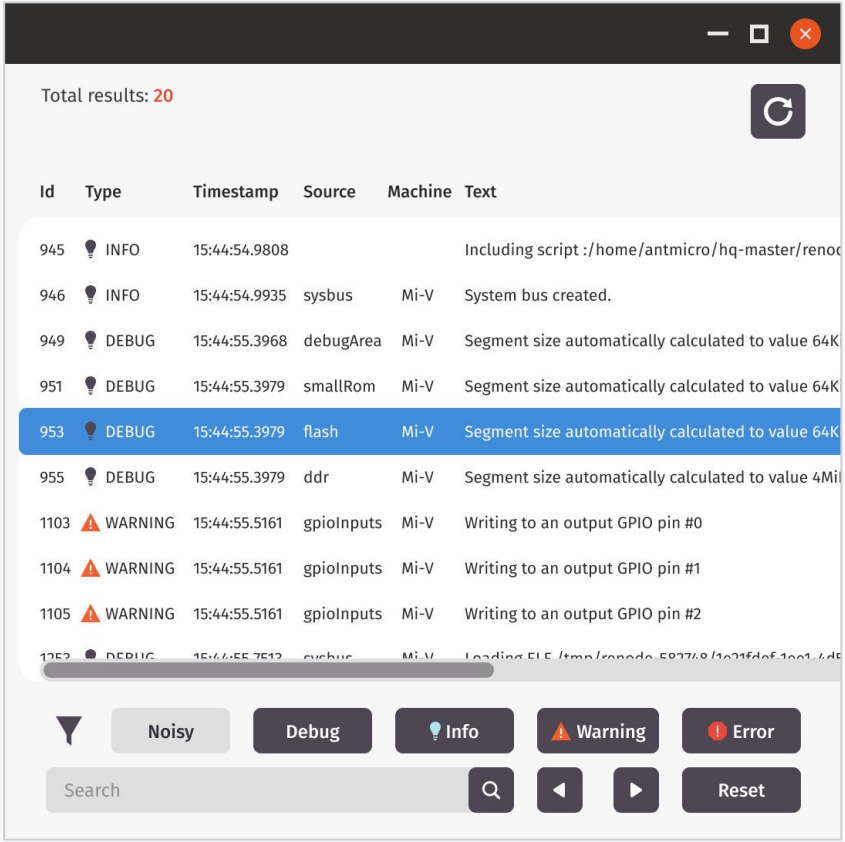
- Uses the GDB remote protocol
- Breakpoints, watchpoints, stepping, memory access etc
- Virtual time does not progress when the emulated CPU is halted
- Multi-core debugging
- Disassembly via LLVM for runtime code analysis

Google Research

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

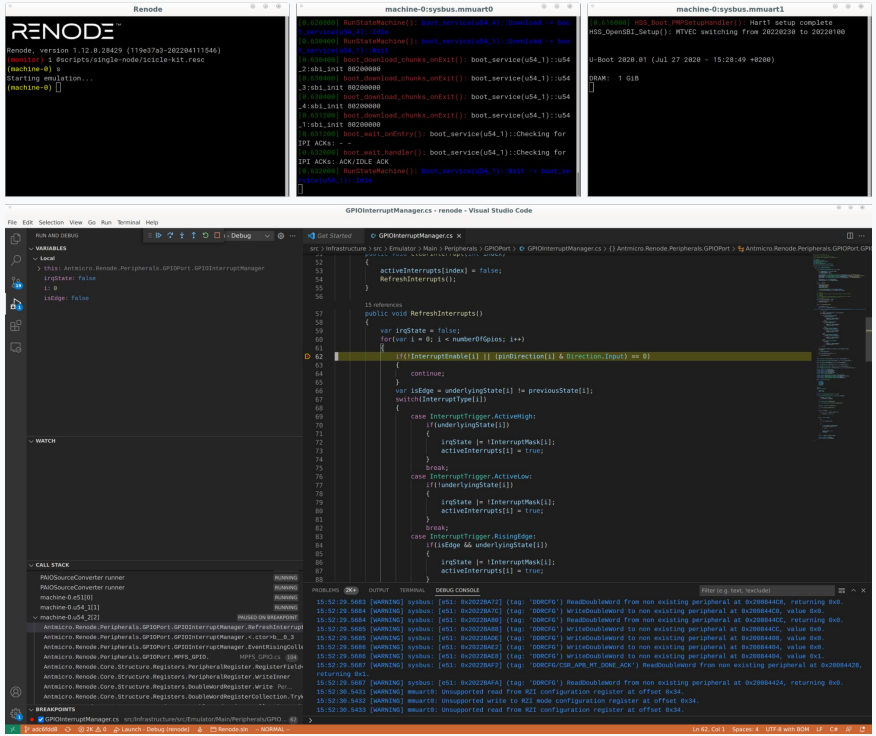


Google Research

IDE support

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

- We provide configuration files to easily run Renode in debug mode with VS code
- Debug interactively with full and precise knowledge of both HW and SW, e.g. how specific parts of drivers affect Renode models



Google Research

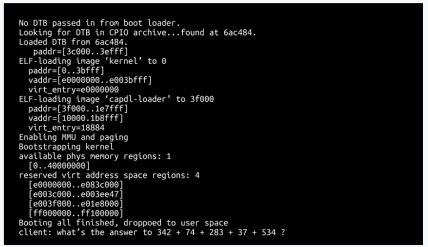
OS-aware debugging

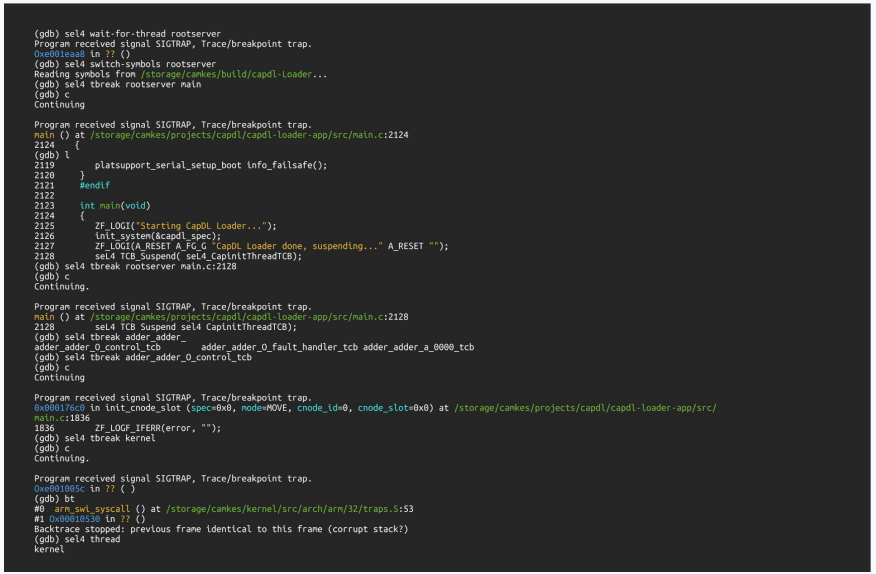
Developed with Google for this project, allows system-level awareness in debugging workflow.

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).





Google Research

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
        "00010100011100000000000010010011"
        "cpu.DebugLog('I’m running Python
                       here!')"
```

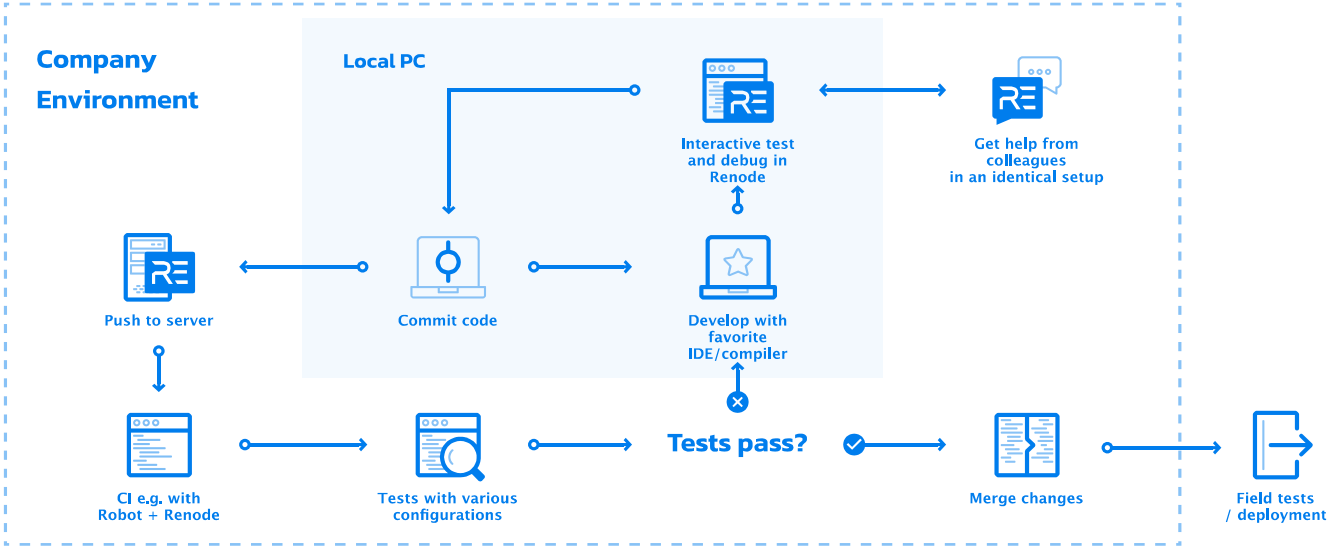
C#

```
RegisterCSR((ulong)0x3e1,
            () => counterValue,
            value => UpdateCounter(value));

InstallCustomInstruction(
 pattern: "0000011-----sssss---ddddd0001011",
 handler: HandleMaskIrqInstruction);
```

Google Research

Development flow - CI



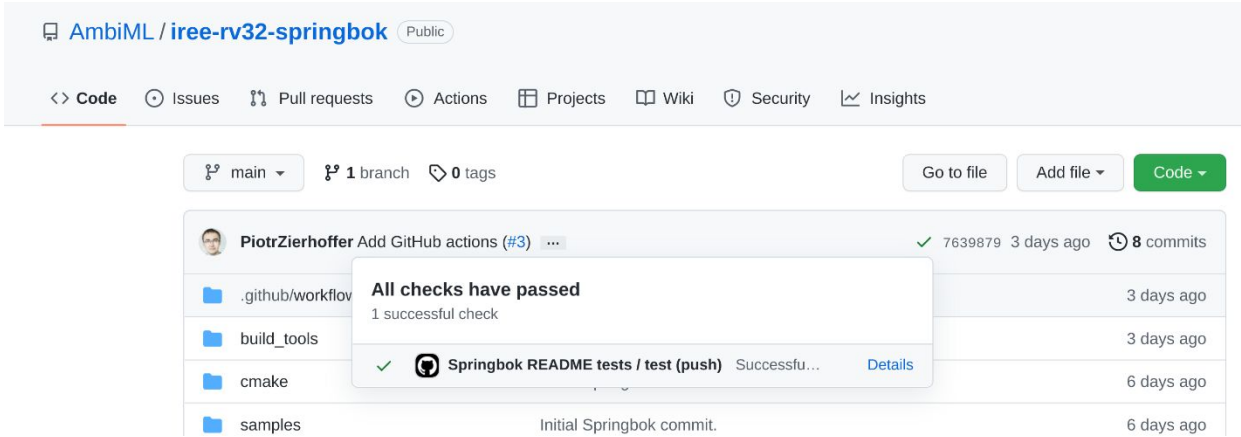
Google Research

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!

Example CI - Springbok



Google Research

03

System Co-design with Renode

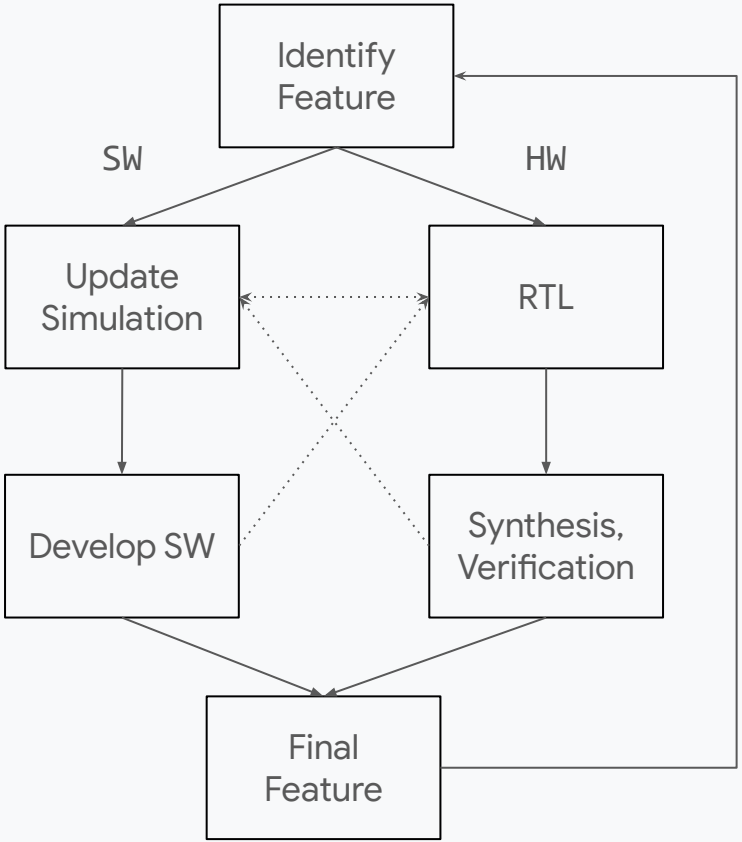
Google Research

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

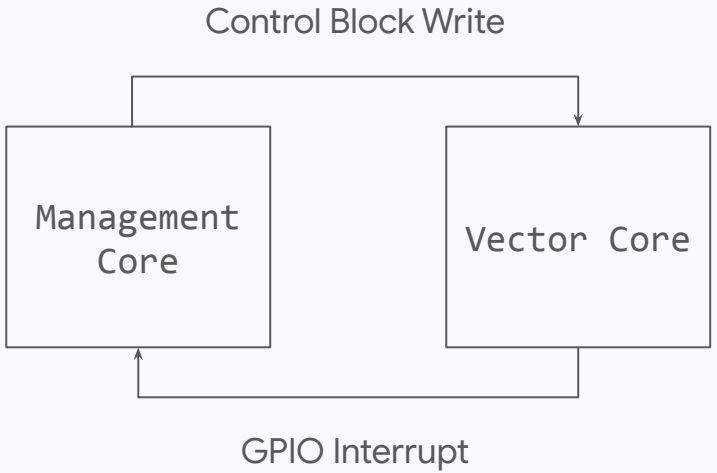
Simulation is crucial here as it enables us to modify hardware at the speed of software



Google Research

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.



Google Research

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>
```

Google Research

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

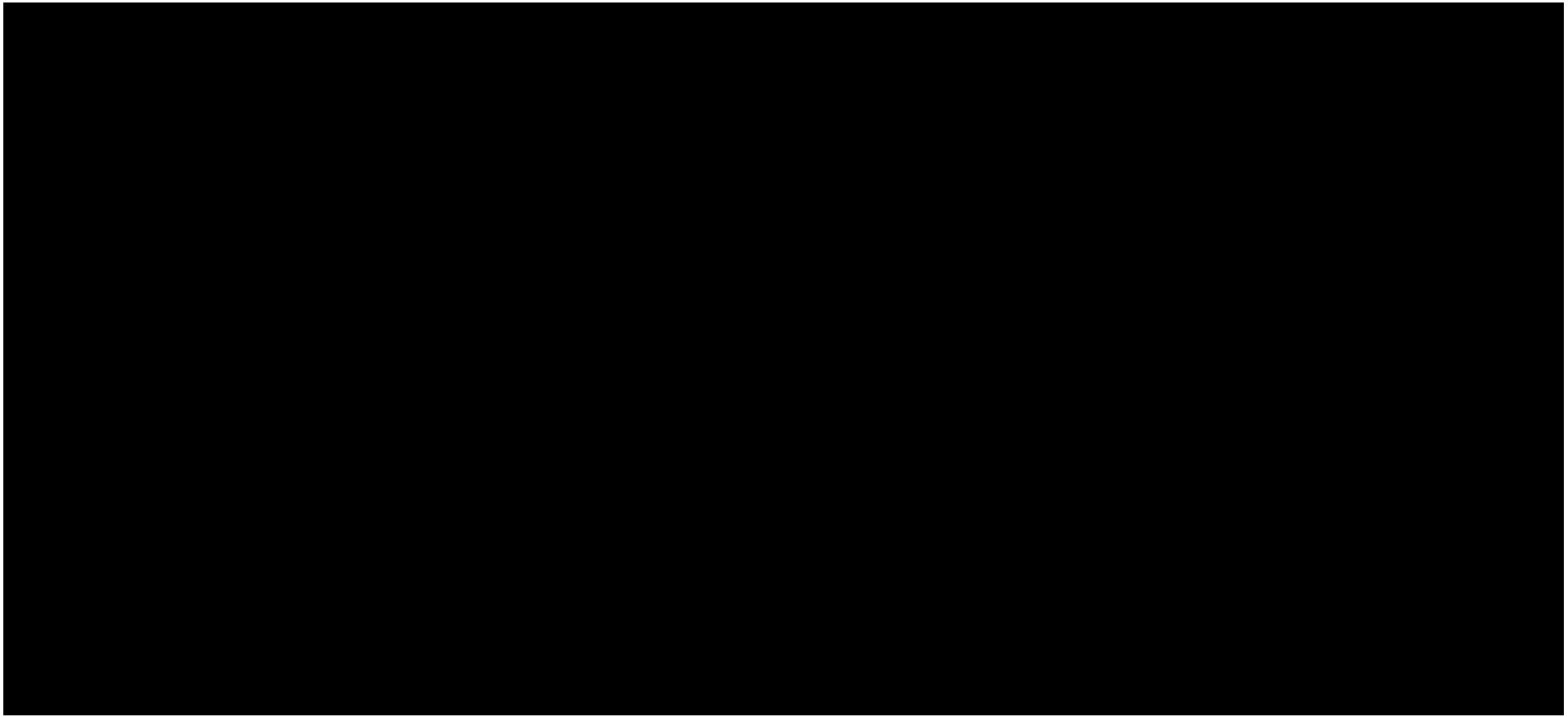
```
InstallCustomInstruction(
 pattern: "-------------------------1111011",
 handler: HandleSpringbokCustom3);


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

Google Research

Springbok MobileNetv1 Demo on Renode

Google Research



Google Research

Thank You

Michael Gielda and Adam Jesionowski

https://github.com/AmbiML/iree-rv32-springbok
https://www.renode.io

Google Research