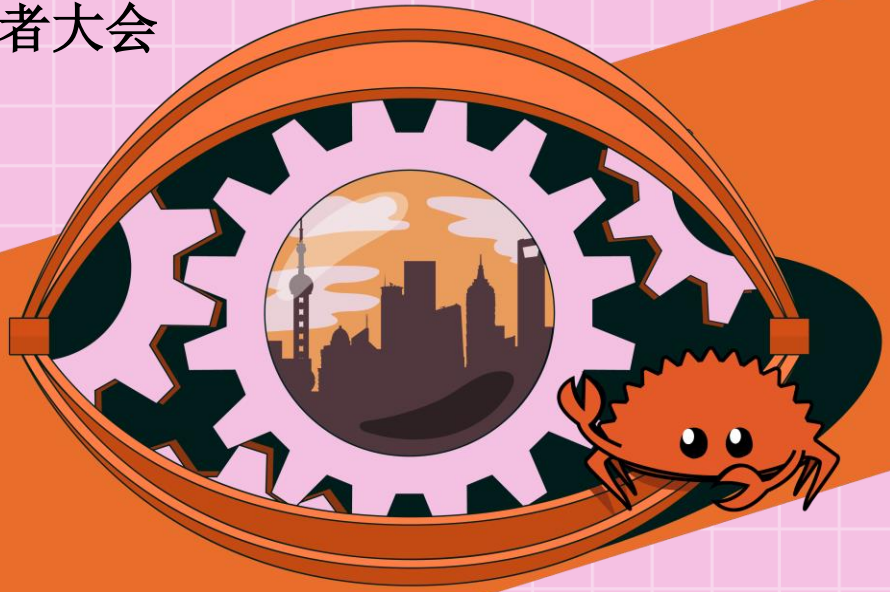
RUST CHINA CONF 2023

第三届中国Rust开发者大会

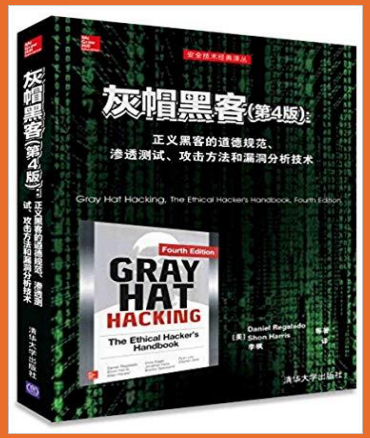


6.17-6.18 @Shanghai

THE FIRST EXPLORATION OF PROJECT SPARROW

Feng Li (李枫)
hkli2013@126.com
Jun 18, 2023

Who Am I



An indie developer from China

- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition» (ISBN: 9787115544384)
- Pure software developmement for ~15 years(~11 years on Mobile dev)
- Actively participating Open Source Communities:
  https://github.com/XianBeiTuoBaFeng2015/MySlides
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, IoT, Programming Languages & Runtimes, Network, Virtualization, RISC-V, EDA, 5G/6G…

Agenda

I. Background
- Tech Stack
- Project Sparrow
- Testbed

II. Practicing Sparrow
- CantripOS(KataOS)
- Rust support in seL4 userspace

III. Sparrow development
- Develop with Renode

IV. Wrap-up

I. Background

1) Tech Stack

1.1 Microkernel

Overview

- **https://en.wikipedia.org/wiki/Microkernel**

In computer science, a **microkernel** (often abbreviated as **μ-kernel**) is the near-minimum amount of software that can provide the mechanisms needed to implement an operating system (OS). These mechanisms include low-level address space management, thread management, and inter-process communication (IPC).

If the hardware provides multiple rings or CPU modes, the microkernel may be the only software executing at the most privileged level, which is generally referred to as supervisor or kernel mode. Traditional operating system functions, such as device drivers, protocol stacks and file systems, are typically removed from the microkernel itself and are instead run in user space.[1]

In terms of the source code size, microkernels are often smaller than monolithic kernels. The MINIX 3 microkernel, for example, has only approximately 12,000 lines of code.[2]

Security

The security benefits of microkernels have been frequently discussed.[29][30] In the context of security the minimality principle of microkernels is, some have argued, a direct consequence of the principle of least privilege, according to which all code should have only the privileges needed to provide required functionality. Minimality requires that a system's trusted computing base (TCB) should be kept minimal. As the kernel (the code that executes in the privileged mode of the hardware) has unvetted access to any data and can thus violate its integrity or confidentiality, the kernel is always part of the TCB. Minimizing it is natural in a security-driven design.

Consequently, microkernel designs have been used for systems designed for high-security applications, including KeyKOS, EROS and military systems. In fact common criteria (CC) at the highest assurance level (Evaluation Assurance Level (EAL) 7) has an explicit requirement that the target of evaluation be "simple", an acknowledgment of the practical impossibility of establishing true trustworthiness for a complex system. Again, the term "simple" is misleading and ill-defined. At least the Department of Defense Trusted Computer System Evaluation Criteria introduced somewhat more precise verbiage at the B3/A1 classes:

> "The TCB shall [implement] complete, conceptually simple protection mechanisms with precisely defined semantics. Significant system engineering shall be directed toward minimizing the complexity of the TCB, as well as excluding from the TCB those modules that are not protection-critical."
>
> — Department of Defense Trusted Computer System Evaluation Criteria

In 2018, a paper presented at the Asia-Pacific Systems Conference claimed that microkernels were demonstrably safer than monolithic kernels by investigating all published critical CVEs for the Linux kernel at the time. The study concluded that 40% of the issues could not occur at all in a formally verified microkernel, and only 4% of the issues would remain entirely unmitigated in such a system.[31]

I. Background

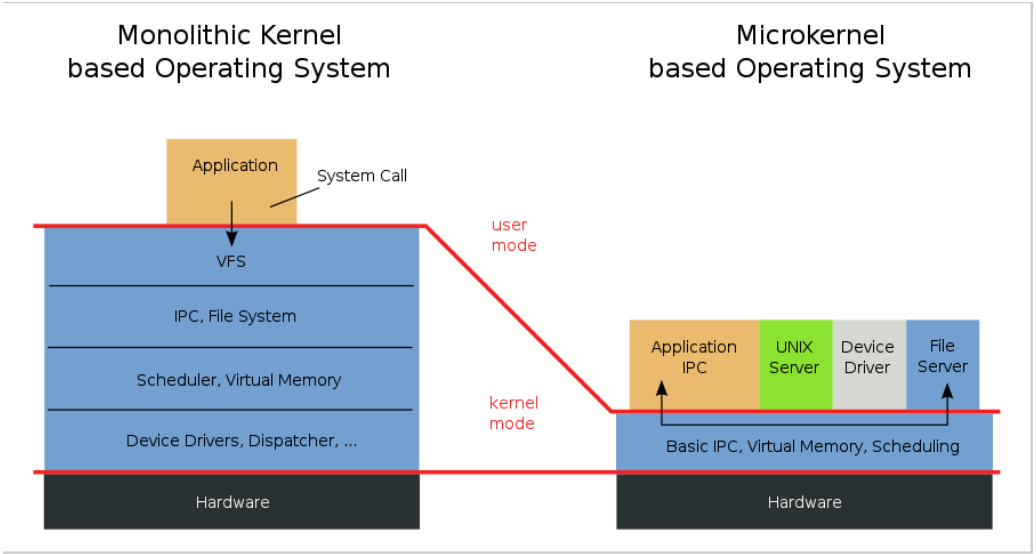
- **Difference Between Microkernel and Monolithic Kernel:**

Parameters	Monolithic kernel	MicroKernel
Basic	It is a large process running in a single address space	It can be broken down into separate processes called servers.
Code	In order to write a monolithic kernel, less code is required.	In order to write a microkernel, more code is required
Security	If a service crashes, the whole system collapses in a monolithic kernel.	If a service crashes, it never affects the working of a microkernel.
Communication	It is a single static binary file	Servers communicate through IPC.
Example	Linux, BSDs, Microsoft Windows (95,98, Me), Solaris, OS-9, AIX, DOS, XTS-400, etc.	L4Linux, QNX, SymbianK42, Mac OS X, Integrity, etc.

Source: https://www.guru99.com/microkernel-in-operating-systems.html

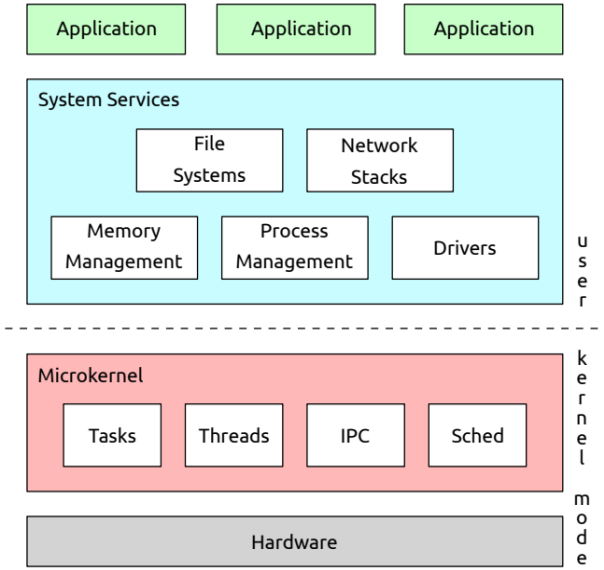
I. Background

Architecture & Design



Structure of monolithic and microkernel-based operating systems, respectively

Source: https://os.inf.tu-dresden.de/Studium/MkK//SS2021/01_intro.pdf



I. Background

1.2 L4

Overview

- **https://en.wikipedia.org/wiki/L4_microkernel_family**

L4 is a family of second-generation microkernels, used to implement a variety of types of operating systems (OS), though mostly for Unix-like, *Portable Operating System Interface* (POSIX) compliant types.

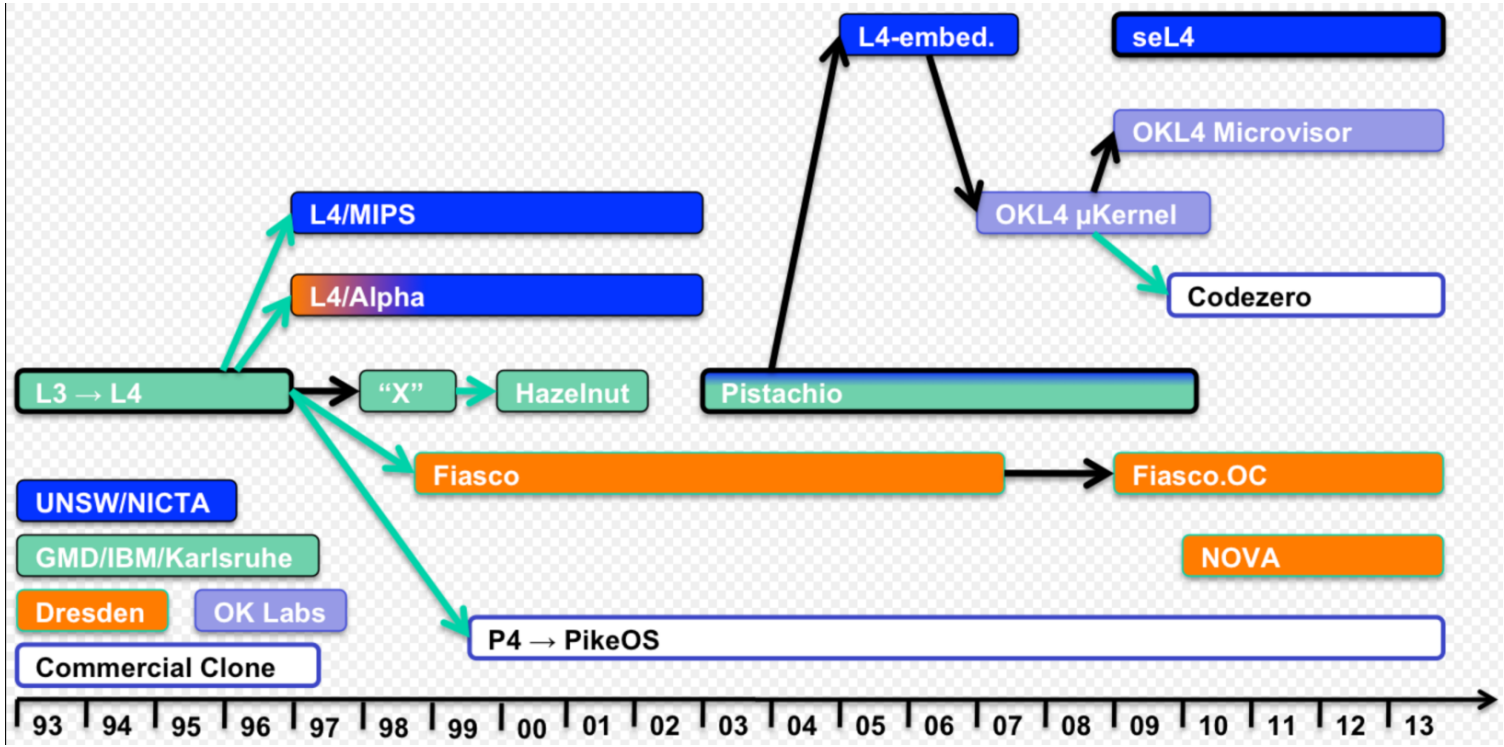
L4, like its predecessor microkernel L3, was created by German computer scientist Jochen Liedtke as a response to the poor performance of earlier microkernel-based OSes. Liedtke felt that a system designed from the start for high performance, rather than other goals, could produce a microkernel of practical use. His original implementation in hand-coded Intel i386-specific assembly language code in 1993 sparked intense interest in the computer industry.[*citation needed*] Since its introduction, L4 has been developed to be cross-platform and to improve security, isolation, and robustness.

There have been various re-implementations of the original binary L4 kernel application binary interface (ABI) and its successors, including *L4Ka::Pistachio* (Karlsruhe Institute of Technology), *L4/MIPS* (University of New South Wales (UNSW)), *Fiasco* (Dresden University of Technology (TU Dresden)). For this reason, the name *L4* has been generalized and no longer refers to only Liedtke's original implementation. It now applies to the whole microkernel family including the L4 kernel interface and its different versions.

L4 is widely deployed. One variant, OKL4 from Open Kernel Labs, shipped in billions of mobile devices.[1][2]

I. Background

L4 family tree:



I. Background

1.2.1 seL4

Overview

- **https://en.wikipedia.org/wiki/L4_microkernel_family#High_assurance:_seL4**

In 2006, the NICTA group commenced a from-scratch design of a third-generation microkernel, named seL4, with the aim of providing a basis for highly secure and reliable systems, suitable for satisfying security requirements such as those of Common Criteria and beyond. From the beginning, development aimed for formal verification of the kernel. To ease meeting the sometimes conflicting requirements of performance and verification, the team used a middle-out software process starting from an executable specification written in Haskell.[16] seL4 uses capability-based security access control to enable formal reasoning about object accessibility.

A formal proof of functional correctness was completed in 2009.[17] The proof provides a guarantee that the kernel's implementation is correct against its specification, and implies that it is free of implementation bugs such as deadlocks, livelocks, buffer overflows, arithmetic exceptions or use of uninitialised variables. seL4 is claimed to be the first-ever general-purpose operating-system kernel that has been verified.[17]

seL4 takes a novel approach to kernel resource management,[18] exporting the management of kernel resources to user level and subjects them to the same capability-based access control as user resources. This model, which was also adopted by Barrelfish, simplifies reasoning about isolation properties, and was an enabler for later proofs that seL4 enforces the core security properties of integrity and confidentiality.[19] The NICTA team also proved correctness of the translation from C to executable machine code, taking the compiler out of the trusted computing base of seL4.[20] This implies that the high-level security proofs hold for the kernel executable. seL4 is also the first published protected-mode OS kernel with a complete and sound worst-case execution-time (WCET) analysis, a prerequisite for its use in hard real-time systems.[19]

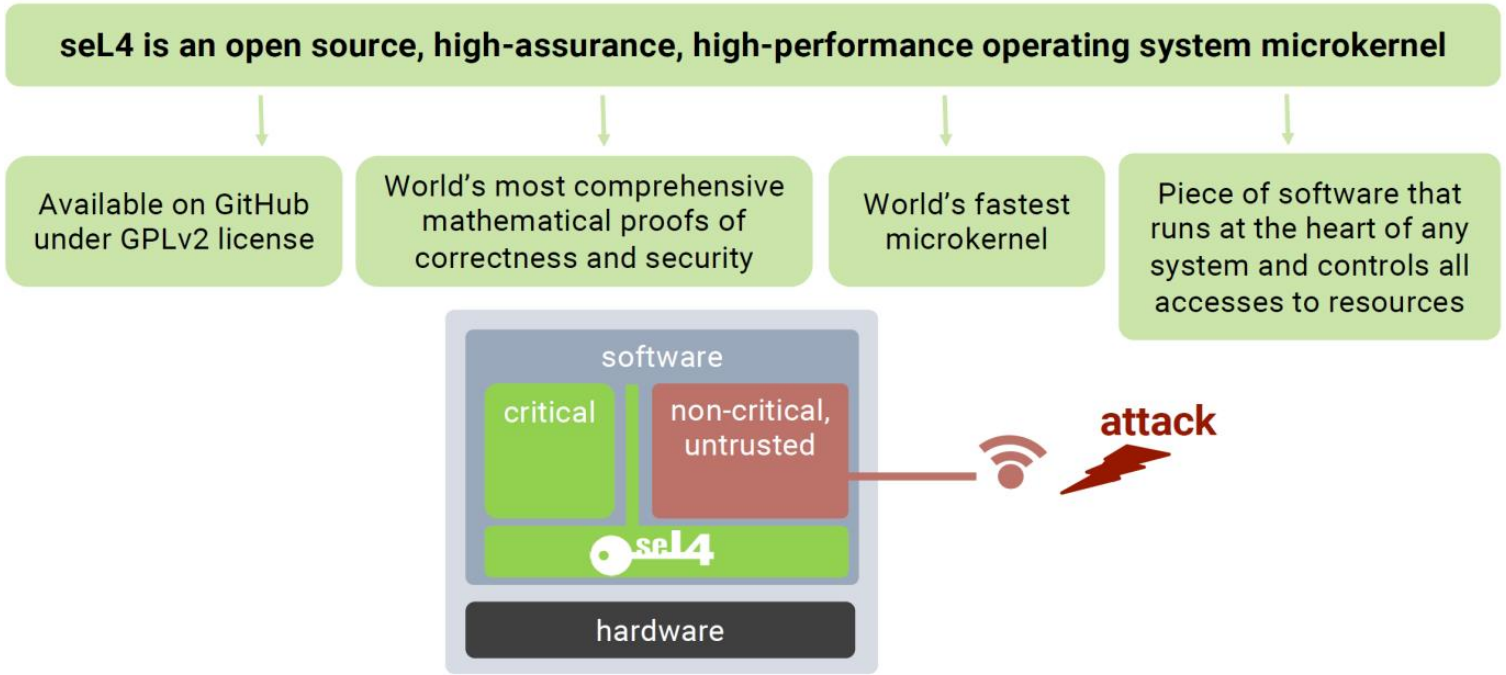
On 29 July 2014, NICTA and General Dynamics C4 Systems announced that seL4, with end to end proofs, was now released under open-source licenses.[21] The kernel source code and proofs are licensed under GNU General Public License version 2 (GPLv2), and most libraries and tools are under the BSD 2-clause. In April 2020, it was announced that the seL4 Foundation was created under the umbrella of the Linux Foundation to accelerate development and deployment of seL4.[22]

The researchers state that the cost of formal software verification is lower than the cost of engineering traditional "high-assurance" software despite providing much more reliable results.[23] Specifically, the cost of one line of code during the development of seL4 was estimated at around US$400, compared to US$1,000 for traditional high-assurance systems.[24]

Under the DARPA High-Assurance Cyber Military Systems (HACMS) program, NICTA together with project partners Rockwell Collins, Galois Inc, the University of Minnesota and Boeing developed a high-assurance drone based on seL4, along with other assurance tools and software, with planned technology transfer onto the optionally piloted autonomous Unmanned Little Bird helicopter under development by Boeing. Final demonstration of the HACMS technology took place in Sterling, VA in April 2017.[25] DARPA also funded several Small Business Innovative Research (SBIR) contracts related to seL4 under a program started by Dr. John Launchbury. Small businesses receiving an seL4-related SBIR included: DornerWorks, Techshot, Wearable Inc, Real Time Innovations, and Critical Technologies.[26]

- **https://sel4.systems**

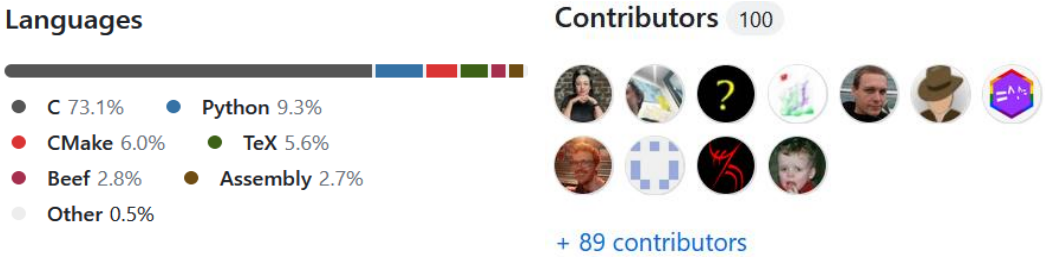
I. Background

- 

Source: "The seL4 Report", Gernot Heiser, Fosdem 2021.

I. Background

- **Src**



I. Background

- **Supported Platforms**
  https://docs.sel4.systems/Hardware/
  e.g., for ARMv7 and ARMv8 Platforms:

Platform	System-on-chip	Core	Arch	Virtualisation	IOMMU	Status	Contributed by	Maintained by
...								
OdroidXU4	Exynos5	Cortex-A15	ARMv7A	ARM HYP	limited System MMU	Unverified	Data61	seL4 Foundation
Raspberry Pi 3-b	BCM2837	Cortex-A53	ARMv8A	ARM HYP	No	Unverified	Data61	seL4 Foundation
Raspberry Pi 4-b	BCM2711	Cortex-A72	ARMv8A	ARM HYP	No	Unverified	Hensoldt and ARM Research IceCap	Hensoldt
Rockpro64	RK3399 hexa-core	Cortex-A53 Quad 1.8 GHz	ARMv8A, AArch64	No	No	Unverified	Data61	seL4 Foundation
...								

I. Background

1.2.1.1 CAmkES

- **https://docs.sel4.systems/projects/camkes/**

CAmkES (component architecture for microkernel-based embedded systems) is a software development and runtime framework for quickly and reliably building microkernel-based multiserver (operating) systems. It follows a component-based software engineering approach to software design, resulting in a system that is modelled as a set of interacting software components. These software components have explicit interaction interfaces and a system design that explicitly details the connections between the components.

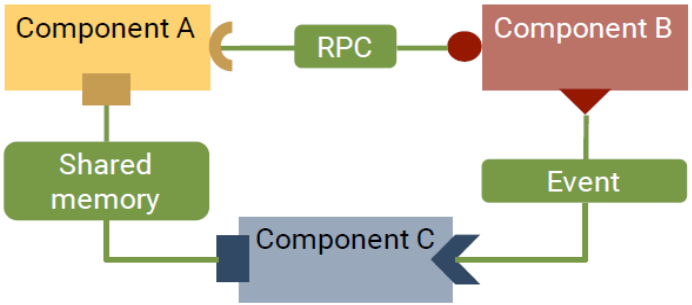
The development framework provides:

- a language to describe component interfaces, components, and whole component-based systems
- a tool that processes these descriptions to combine programmer-provided component code with generated scaffolding and glue code to build a complete, bootable, system image
- full integration in the seL4 environment and build system

https://github.com/seL4/camkes

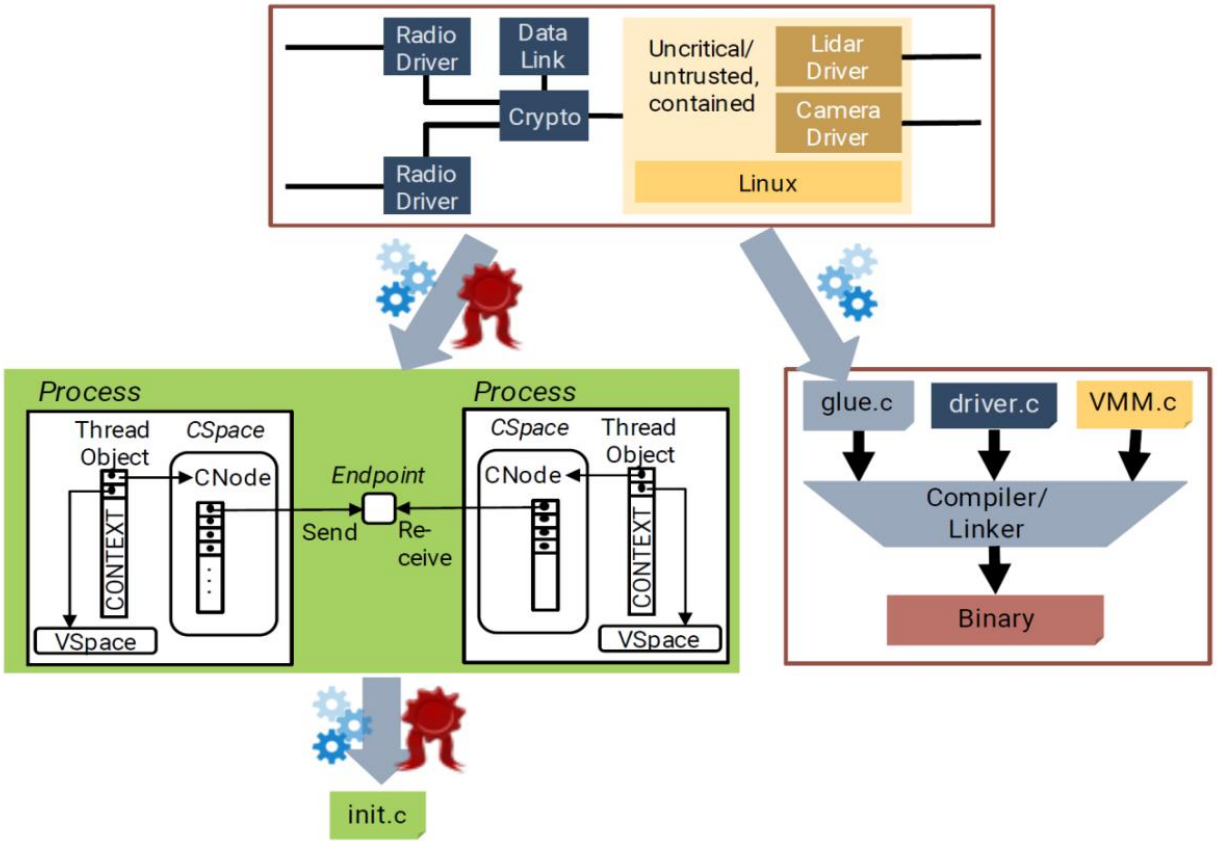
https://github.com/seL4/camkes-vm

- **CAmkES ADL**



Source: https://sel4.systems/About/seL4-whitepaper.pdf

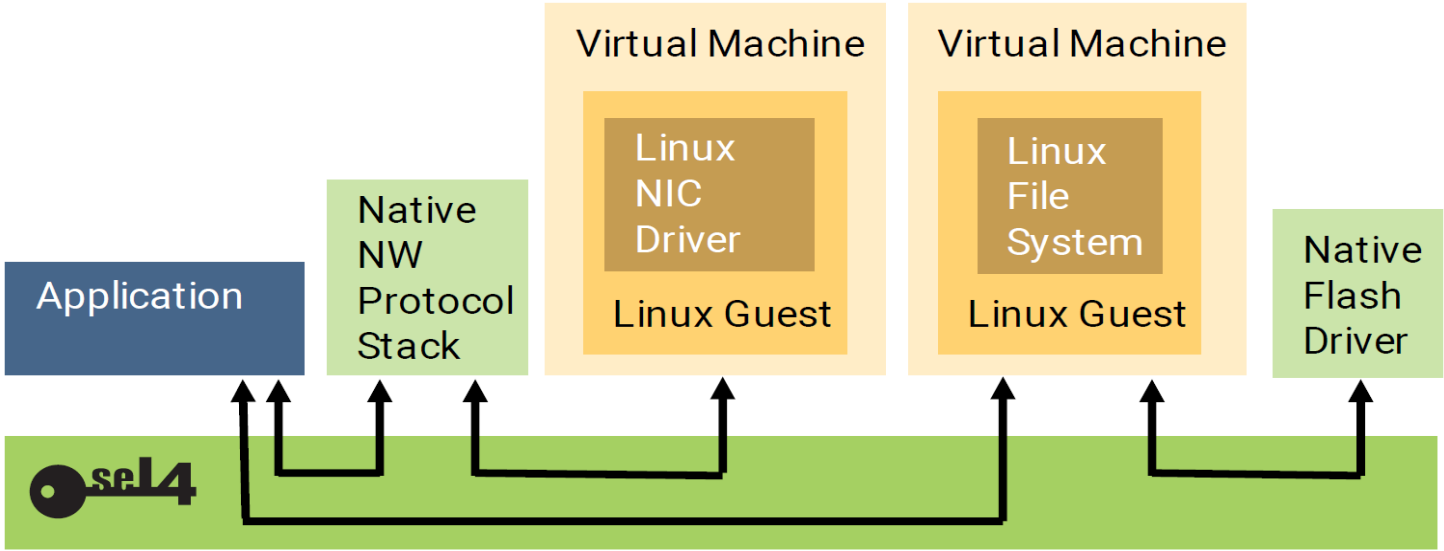
I. Background



Source: https://sel4.systems/About/seL4-whitepaper.pdf

1.2.1.2 Virtualization

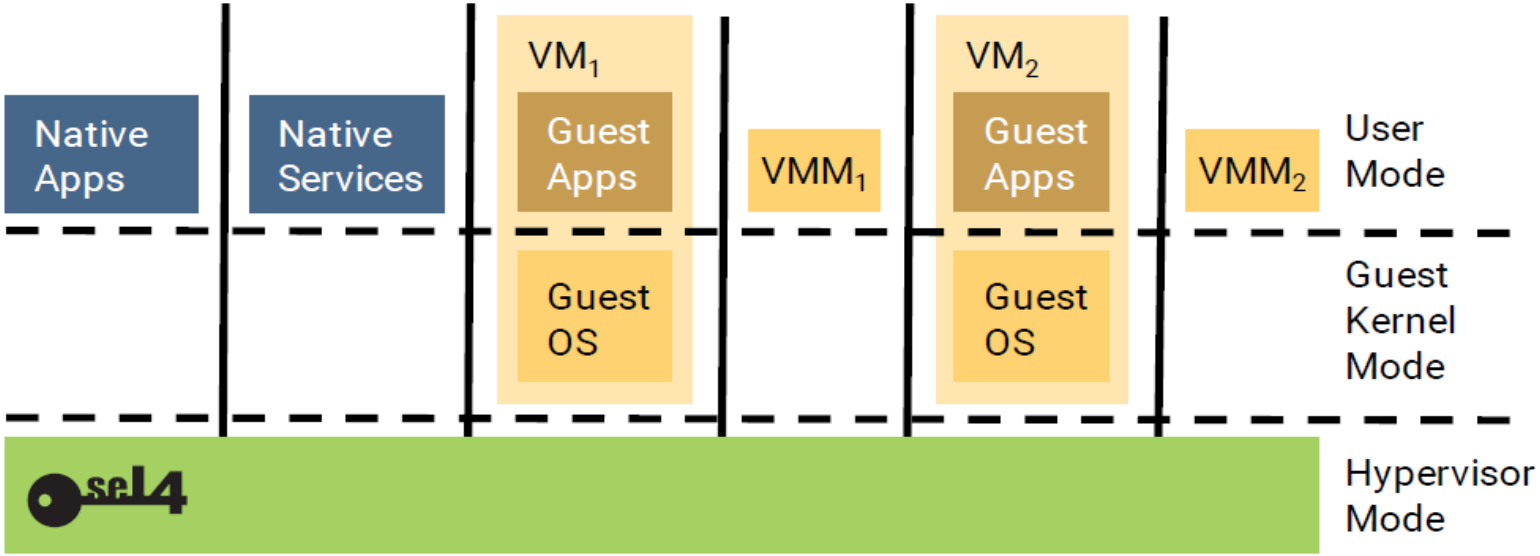
- **seL4 is also a hypervisor**



Source: https://sel4.systems/About/seL4-whitepaper.pdf

I. Background

- **seL4 virtualization support with usermode VMMs**



Source: https://sel4.systems/About/seL4-whitepaper.pdf

1.2.1.3 Provable Security

- seL4 was the world's first OS kernel with a proof of implementation correctness (functional correctness). We then extended the verification down to the binary and up to security-enforcement properties, as explained in Chapter 3.

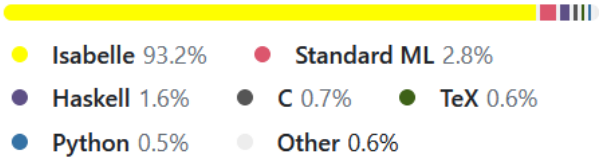
  While by now there are other verified OS kernels, seL4 still defines the state of the art [Heiser, 2019]: It has the most comprehensive verification story, it is still the only capability-based OS that is verified, and it has the most advanced real-time support. And our ongoing research aims to ensure that seL4 will retain its position as the clear leader among security- and safety-oriented OSes, for example by pioneering systematic and principled prevention of information leakage through timing channels [Ge et al., 2019].

  Source: https://sel4.systems/About/seL4-whitepaper.pdf

- **https://docs.sel4.systems/projects/l4v/**

  https://github.com/seL4/l4v

  Languages

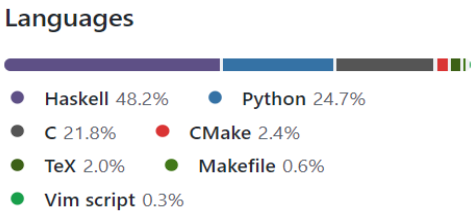


I. Background

CapDL

- **Capability Distribution Language tools for seL4**

  https://github.com/seL4/capdl



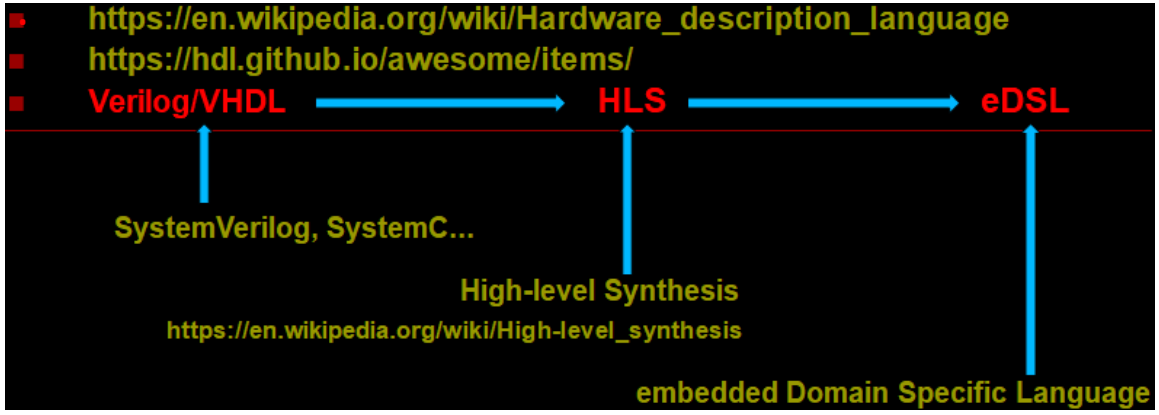
1.3 FOSS EDA

Overview

- https://en.wikipedia.org/wiki/Comparison_of_EDA_software
- https://semiwiki.com/wikis/industry-wikis/eda-open-source-tools-wiki/
- https://fossi-foundation.org/
- https://ieeexplore.ieee.org/document/9398963
- https://ieeexplore.ieee.org/document/9398960
- https://ieeexplore.ieee.org/document/9336682
- https://ieeexplore.ieee.org/document/9105619
- …

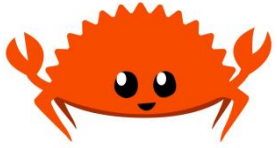
1.3.1 Evolution of HDLs

- https://en.wikipedia.org/wiki/Hardware_description_language
- https://hdl.github.io/awesome/items/
- Verilog/VHDL → HLS → eDSL



Typical eDSLs

- Haskell as host
  Bluespec, Clash...
- Scala as host
  Chisel, SpinalHDL...
- Python as host
  Amaranth/FHDL, PyGears...
- Finally convert to Verilog/VHDL
  https://en.wikipedia.org/wiki/Source-to-source_compiler



is coming!

1.4 RISC-V

Ecosystem

- https://community.riscv.org/events/details/risc-v-foundation-bay-area-risc-v-group-presents-2021-risc-v-ecosystem-updates

Source: https://www.zdnet.com/article/risc-v-the-linux-of-the-chip-world-is-starting-to-produce-technological-breakthroughs/

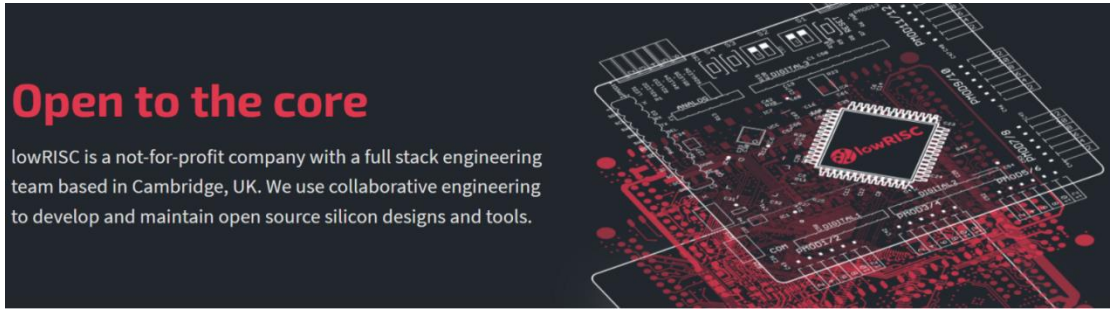
Source: https://zhuanlan.zhihu.com/p/367035973

I. Background

1.4.2 lowRISC

Overview

- https://lowrisc.org/



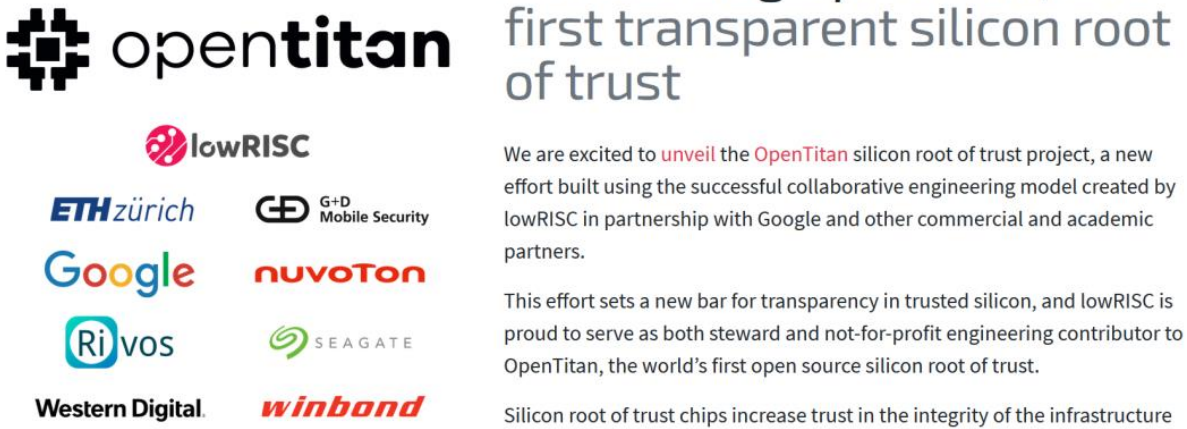
- https://github.com/lowrisc

1.4.2.1 Project OpenTitan

Overview

- **https://opentitan.org/**

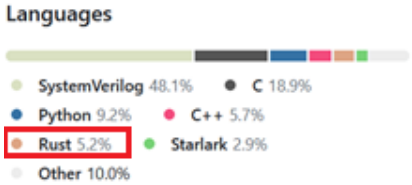
  OpenTitan is the first open source project building a transparent, high-quality reference design and integration guidelines for silicon root of trust (RoT) chips.

- 

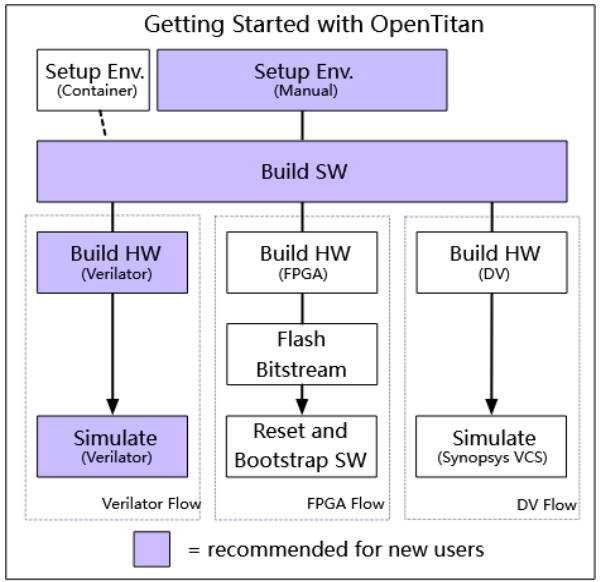
Source: https://lowrisc.org

I. Background

- **https://github.com/lowRISC/opentitan**



- **Workflow**



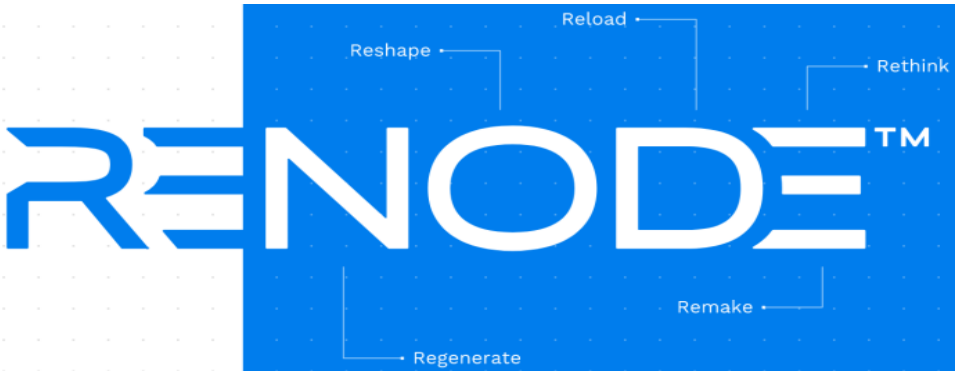
Source: https://opentitan.org/guides/getting_started/index.html

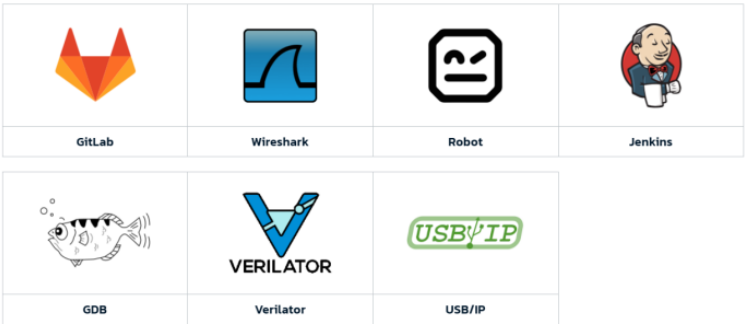
1.5 Renode

Overview

- https://renode.io/

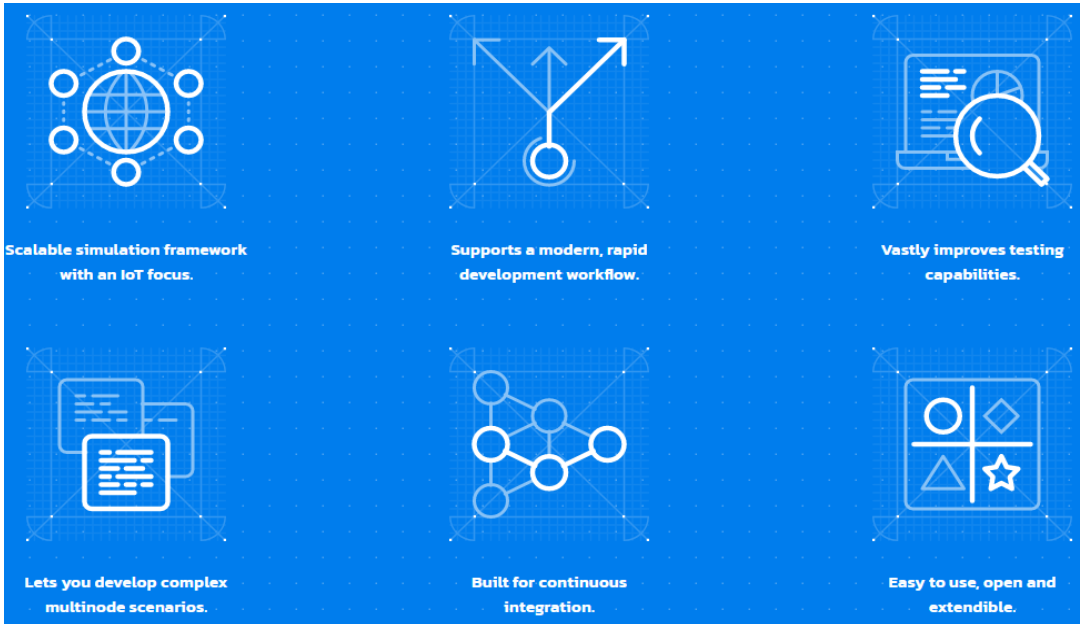
Antmicro's virtual development framework for complex embedded systems.





I. Background

- **Key features:**

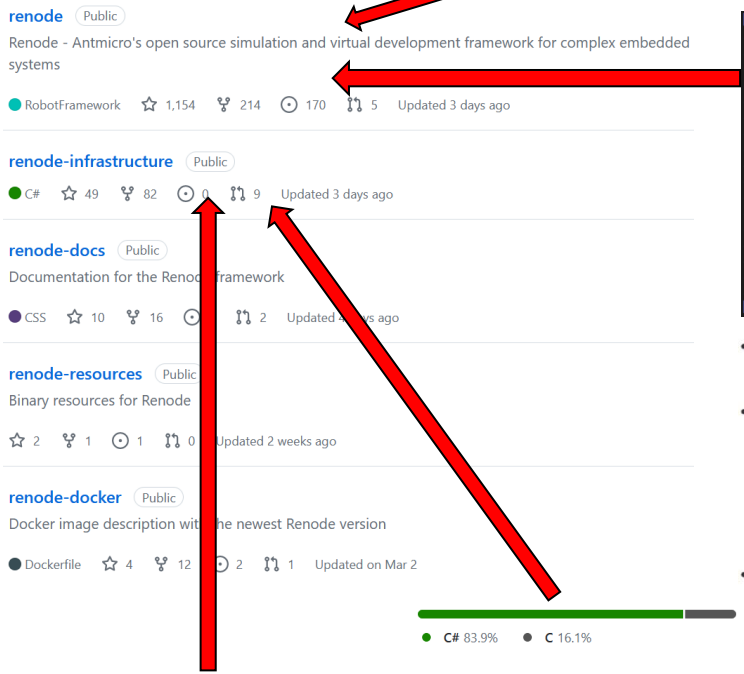


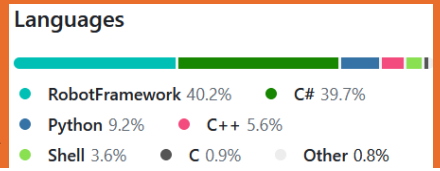
- https://renode.io/about/

I. Background

1.5.1 Src

- https://github.com/renode





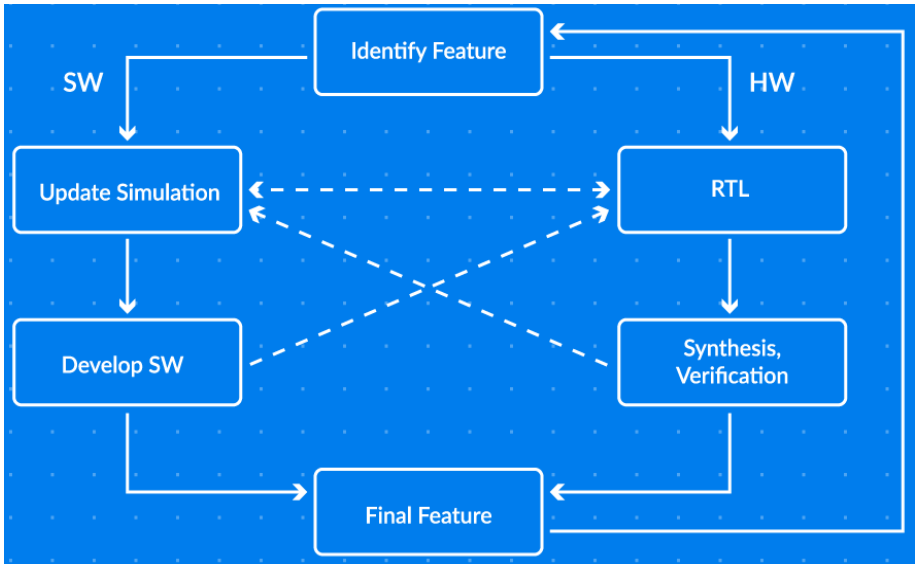

```
[mydev@fedora renode-master]$ find . -name "*.git"
./.git
./lib/AntShell/.git
./lib/BigGustave/.git
./lib/CxxDemangler/.git
./lib/ELFSharp/.git
./lib/FdtSharp/.git
./lib/InpliTftpServer/.git
./lib/Migrant/.git
./lib/Packet.Net/.git
./lib/bc-csharp/.git
./lib/cctask/.git
./lib/options-parser/.git
./lib/termsharp/.git
./lib/termsharp/xwt/.git
./lib/xwt/.git
./src/Infrastructure/.git
./src/Infrastructure/src/Emulator/Cores/tlib/.git
[mydev@fedora renode-master]$
```

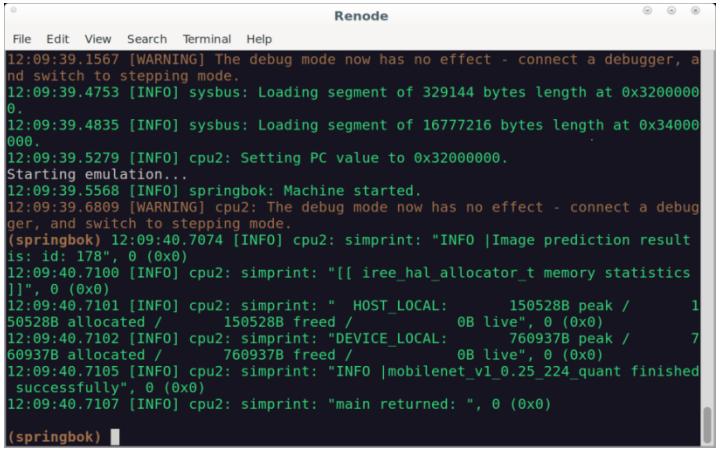
- LiteX generates a platform description in csr.csv output file
- Automatic generation from LiteX configuration (CSV)
  - platform (REPL)
  - script (RESC)
  - Zephyr DTS overlay
- Perfect for CI setup

```
mydev@fedora renode-master]$ tree -L 4 src/Plugins/
src/Plugins/
├── VerilatorPlugin
│   ├── Connection
│   │   ├── IVerilatedPeripheral.cs
│   │   ├── LibraryVerilatorConnection.cs
│   │   ├── Protocols
│   │   │   ├── ActionType.cs
│   │   │   └── ProtocolMessage.cs
│   │   └── SocketVerilatorConnection.cs
│   ├── Verilated
│   │   └── Peripherals
│   │       ├── BaseDoubleWordVerilatedPeripheral.cs
│   │       ├── BaseVerilatedPeripheral.cs
│   │       ├── CFUVerilatedPeripheral.cs
│   │       ├── VerilatedCPU.cs
│   │       ├── VerilatedPeripheral.cs
│   │       ├── VerilatedRiscV32.cs
│   │       ├── VerilatedRiscV32Registers.cs
│   │       └── VerilatedUART.cs
│   ├── VerilatorIntegrationLibrary
│   │   ├── hdl
│   │   │   ├── renode_axi.sv
│   │   │   └── renode.sv
│   │   ├── libs
│   │   │   └── socket-cpp
│   │   └── src
│   │       ├── buses
│   │       ├── communication
│   │       ├── peripherals
│   │       ├── renode_action_enumerators.txt
│   │       ├── renode_bus.cpp
│   │       ├── renode_bus.h
│   │       ├── renode_cfu.cpp
│   │       ├── renode_cfu.h
│   │       ├── renode_dpi.cpp
│   │       ├── renode_dpi.h
│   │       ├── renode.h
│   │       └── verilator-integration-library.cmake
│   ├── VerilatorPlugin.cs
│   ├── VerilatorPlugin.csproj
│   └── VerilatorPlugin_NET.csproj
└── WiresharkPlugin
    ├── BLESniffer.cs
    ├── INetworkLogExtensions.cs
    ├── LinkLayer.cs
    ├── Wireshark.cs
    ├── WiresharkPlugin.cs
    ├── WiresharkPlugin.csproj
    ├── WiresharkPlugin_NET.csproj
    └── WiresharkSender.cs
```

- https://github.com/antmicro/tlib

1.5.2 Co-simulating by Renode





Source: https://opensource.googleblog.com/2022/09/co-simulating-ml-with-springbok-using-renode.html

1.5.3 Renode for RISC-V development

Official

- 

RENODE™ RISC-V®
- **https://riscv.org/announcements/2021/12/risc-v-celebrates-incredible-year-of-growth-and-progress-ratifying-multiple-technical-specifications-launching-new-education-programs-and-accelerating-broad-industry-adoption/**

  Notable achievements of RISC-V adoption in 2021 include:

  - Antmicro added support for the RVV 1.0 vector instructions to its open source Renode simulation framework, allowing users to enhance their machine learning development experience in a purely virtual environment. RISC-V Vector ISA support will be one of the highlights of the upcoming Renode 1.13 release.
- **https://riscv.org/blog/2021/06/antmicro-open-source-portal-launched/**
- **https://riscv.org/blog/2021/03/bringing-the-benefits-of-risc-v-and-renode-to-the-very-efficient-deep-learning-in-iot-project/**
- **…**

2) Project Sparrow
2.1 Overview

- **https://renohttps://github.com/AmbiML/sparrow-manifest**

  Sparrow is a project to build a low-power secure embeded platform for Ambient ML applications. The target platform leverages RISC-V and OpenTitan. The Sparrow software includes a home-grown operating system named CantripOS, that runs on top of seL4 and (ignoring the seL4 kernel) is written almost entirely in Rust.

  Sparrow (and CantripOS) are definitely a work in progress. The CantripOS components are based on an augmented version of seL4's CAmkES framework. Critical system services are CAmkES components that are statically configured. Applications are developed using an AmbiML-focused SDK and dynamically loaded by the system services.

  The components used depends on the target platform. At the moment two platforms are buildable: sparrow and rpi3 (Raspberry Pi BCM2837 running in 64-bit mode). The sparrow platform is not useful other than for reference as building it requires toolchain & simulator support that is not yet released. The rpi3 platform is the intended target platform for public consumption. Contribution of additional platform support is welcomed (e.g. a timer driver for the TimerService).

I. Background

Software repositories:

Sparrow consists of multiple git repositories stitched together with the repo tool. The following git repositories are currently available:

- *camkes-tool*: seL4's camkes-tool repository with additions to support CantripOS services
- *capdl*: seL4's capdl repository with addition for CantripOS services and the CantripOS rootserver (a replacement for capdl-loader-app that is written in Rust and supports hand-off of system resources to the CantripOS MemoryManager service)
- *kernel*: seL4's kernel with drivers for Sparrow's RISC-V platform and support for reclaiming the memory used by the CantripOS rootserver
- *cantrip-full* frameworks for developing in Rust and the CantripOS system services
- *scripts*: support scripts including build-sparrow.sh

2.2 CantripOS(KataOS)

Overview

- **https://github.com/AmbiML/sparrow-cantrip-full**

  Sparrow is a project to build a low-power secure embedded platform for Ambient ML applications. The target platform leverages RISC-V and OpenTitan.

  The Sparrow software includes a home-grown operating system named CantripOS, that runs on top of seL4 and (ignoring the seL4 kernel) is written almost entirely in Rust. CantripOS is comprised of a set of system services that are assembled using CAmkES and applications that are dynamically loaded into a constrained seL4 thread context and communicate with system services through an SDK runtime environment.

  The CAmkES project that assembles the CantripOS system services is found in this git repository. It exists outside the seL4 source trees since it contains code not intended to go to upstream seL4.

  The target-platform-dependent CAmkES assembly description is found in apps/system/platforms. It is built using the standard CAmkES build system and requires the CAmkES dependencies already be installed. Top-level configuration is found in easy-settings.cmake and settings.cmake with build-related configuration in build/cantrip.mk and nearby makefiles.

I. Background

Rust crates

- Most CantripOS Rust crates are in the *cantrip/apps/system/components* directory. Common/shared code is in *cantrip-os-common*:

  - *allocator*: a heap allocator built on the linked-list-allocator crate
  - *camkes*: support for writing CAmkES components in Rust
  - *capdl*: support for reading the capDL specification generated by capDL-tool
  - *copyregion*: a helper for temporarily mapping physical pages into a thread's VSpace
  - *cspace-slot*: an RAII helper for the *slot-allocator*
  - *logger*: seL4 integration with the Rust logger crate
  - *model*: support for processing capDL; used by the cantrip-os-rootserver
  - *panic*: an seL4-specific panic handler
  - *sel4-config*: build glue for seL4 kernel configuration
  - *sel4-sys*: seL4 system interfaces & glue
  - *slot-allocator*: an allocator for slots in the top-level CNode

  CantripOS system services make up the remaining components:

  - *DebugConsole*: a command line interface intended for debug builds
  - *MailboxDriver*: a driver for the Mailbox interface used to communicate betwen the security and management cores [sparow-only]
  - *MemoryManager*: the memory / object manager service that supports dynamic memory management
  - *MlCoordinator*: a service that manages running ML jobs [requires support that is currently on on sparrow]
  - *OpenTitanUARTDriver*: a driver for the UART on the management core [sparrow-only]
  - *ProcessManager*: the service that creates & manages execution of applications
  - *SDKRuntime*: the service that handles application runtime requests
  - *SecurityCoordinator*: the service that provides an interface to the security core (using the MailboxDriver)
  - *TimerService*: a service built on top of the management core timer hardware [requires hardware timer support]

Source: https://github.com/AmbiML/sparrow-manifest

2.3 Springbok

Overview

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

RISC-V 32-Bit Bare-Metal ML Deployment on Springbok via IREE (https://openxla.github.io/iree/).

This project demonstrates how to compile RISC-V 32-bit ML workloads via IREE, and deploy the workloads with IREE's c API to generate the bare-matal executables. The built artifacts are targeted for Springbok, a RISC-V 32-bit bare-metal platform, and can be simulated with Renode.

Languages

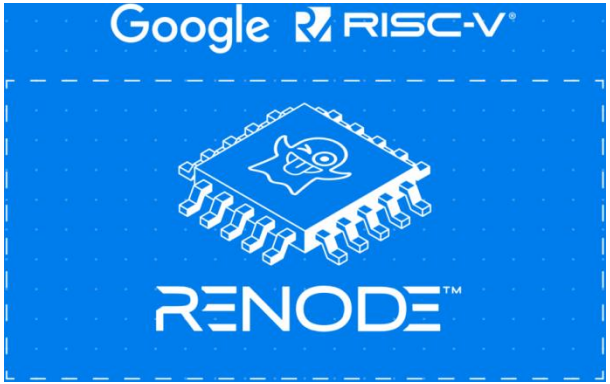
- CMake 29.3%
- C# 20.7%
- Python 17.5%
- C++ 12.7%
- Assembly 8.5%
- C 6.6%
- Shell 4.7%

- **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



Source: https://renode.io/news/co-developing-ml-with-risc-v-using-renode-for-google-research/

Source: https://open-src-soc.org/2022-05/media/slides/4th-RISC-V-Meeting-2022-05-03-14h00-Michael-Gielda-and-Adam-Jesionowski.pdf

3) Testbed

3.1 Intel NUC X15 LAPAC71H(32GB DDR5) with Fedora 38

- **HW**

http://www.intel.com/AC57-Support


```
[mydev11@koonuc15x-1 /]$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  20
  On-line CPU(s) list:   0-19
Vendor ID:               GenuineIntel
  Model name:            12th Gen Intel(R) Core(TM) i7-12700H
```

```
[mydev11@koonuc15x-1 /]$ lspci -k |grep -EA3 'VGA|3D|Display'
00:02.0 VGA compatible controller: Intel Corporation Alder Lake-P Integrated Graphics Controller (rev 0c)
        DeviceName: Onboard - Video
        Subsystem: Intel Corporation Device 3029
        Kernel driver in use: i915
--
03:00.0 Display controller: Intel Corporation DG2 [Arc A730M] (rev 08)
        Subsystem: Intel Corporation Device 3029
        Kernel driver in use: i915
        Kernel modules: i915
[mydev11@koonuc15x-1 /]$
```

- **SW**

```
[mydev11@koonuc15x-1 /]$ uname -a
Linux koonuc15x-1 6.3.6-200.fc38.x86_64 #1 SMP PREEMPT_DYNAMIC Mon Jun  5 15:45:04 UTC 2023 x86_64 GNU/Linux
[mydev11@koonuc15x-1 /]$
[mydev11@koonuc15x-1 /]$ free -m
               total        used        free      shared  buff/cache   available
Mem:           31655         555       29334          80        1765       30623
Swap:           8191           0        8191
[mydev11@koonuc15x-1 /]$
[mydev11@koonuc15x-1 /]$
```

I. Background

```
[mydev11@koonuc15x-1 /]$ clang -v
clang version 16.0.4 (Fedora 16.0.4-1.fc38)
Target: x86_64-redhat-linux-gnu
Thread model: posix
InstalledDir: /bin
Found candidate GCC installation: /bin/../lib/gcc/x86_64-redhat-linux/13
Selected GCC installation: /bin/../lib/gcc/x86_64-redhat-linux/13
Candidate multilib: .;@m64
Candidate multilib: 32;@m32
Selected multilib: .;@m64
[mydev11@koonuc15x-1 /]$
```

```
[mydev11@koonuc15x-1 /]$ python -V
Python 3.11.3
[mydev11@koonuc15x-1 /]$
```

```
[mydev11@koonuc15x-1 /]$ dotnet --version
7.0.302
[mydev11@koonuc15x-1 /]$
```

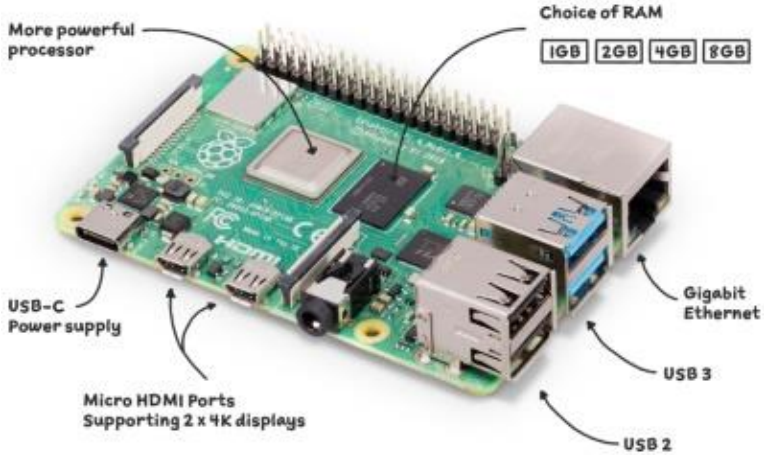
```
[mydev11@koonuc15x-1 /]$ java --version
openjdk 20.0.1 2023-04-18
OpenJDK Runtime Environment GraalVM CE 20.0.1-dev+9.1 (build 20.0.1+9-jvmci-23.1-b02)
OpenJDK 64-Bit Server VM GraalVM CE 20.0.1-dev+9.1 (build 20.0.1+9-jvmci-23.1-b02, mixed mode, sharing)
[mydev11@koonuc15x-1 /]$
[mydev11@koonuc15x-1 /]$ gu list
ComponentId              Version             Component name                Stability
------------------------------------------------------------------------------------------------------
----------
graalvm                  23.1.0-dev          GraalVM Core                  Supported
antlr4                   23.1.0-dev          ANTLR4                        Supported
espresso                 23.1.0-dev          Java on Truffle               Experimental
espresso-llvm            23.1.0-dev          Java on Truffle LLVM Java librExperimental
icu4j                    23.1.0-dev          ICU4J                         Supported
js                       23.1.0-dev          Graal.js                      Supported
llvm                     23.1.0-dev          LLVM Runtime Core             Supported
llvm-toolchain           23.1.0-dev          LLVM.org toolchain            Supported
native-image             23.1.0-dev          Native Image                  Early adopter
native-image-llvm-backend23.1.0-dev          Native Image LLVM Backend     Early adopter (experimental)
nodejs                   23.1.0-dev          Graal.nodejs                  Supported
python                   23.1.0-dev          GraalVM Python                Experimental
regex                    23.1.0-dev          TRegex                        Supported
visualvm                 23.1.0-dev          VisualVM                      Supported
wasm                     23.1.0-dev          GraalWasm                     Experimental
[mydev11@koonuc15x-1 /]$
```

```
[mydev11@koonuc15x-1 /]$ rustup show
Default host: x86_64-unknown-linux-gnu
rustup home:  /home/mydev11/.rustup

installed targets for active toolchain
--------------------------------------

wasm32-unknown-unknown
wasm32-wasi
x86_64-unknown-linux-gnu

active toolchain
----------------

stable-x86_64-unknown-linux-gnu (default)
rustc 1.70.0 (90c541806 2023-05-31)

[mydev11@koonuc15x-1 /]$
```

3.2 Raspberry Pi

RPi4 (8GB LPDDR4) with Fedora 37

- **HW**



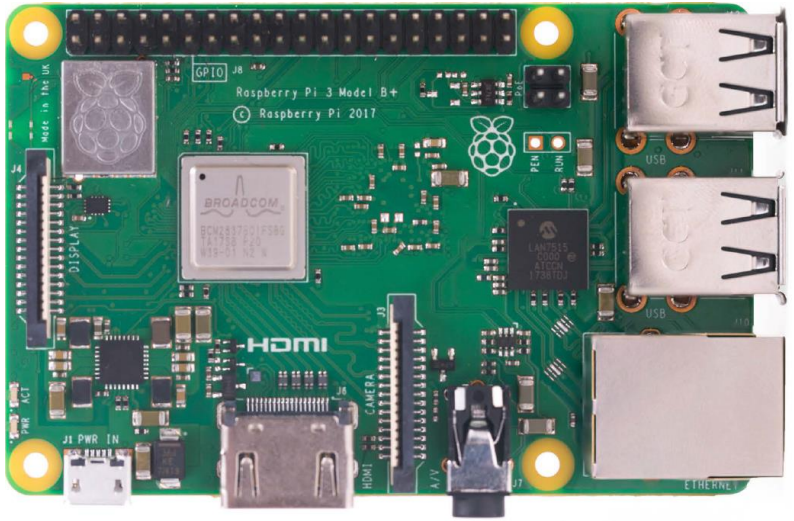
- **SW**

…

I. Background

RPi3 B+ (1G LPDDR2)

- **https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/**
- **HW**



- **SW**

…

II. Practicing Sparrow

1) CantripOS(KataOS)

1.1 Getting started

- **https://github.com/AmbiML/sparrow-cantrip-full/blob/main/docs/GettingStarted.md**

CantripOS includes a multi-platform build framework. This framwork leverages make, cmake, and cargo. To get started follow these steps:

1. Clone the Sparrow project from GitHub using the repo tool We assume below this lands in a top-level directory named "sparrow".
2. Download, build, and boot the system to the Cantrip shell prompt. For now the only target platform that works is "rpi3" (for a raspi3b machine running in simulation on qemu).

```
mkdir sparrow
cd sparrow
repo init -u https://github.com/AmbiML/sparrow-manifest -m sparrow-manifest.xml
repo sync -j$(nproc)
export PLATFORM=rpi3
source build/setup.sh
m simulate-debug
```

Prerequisites.

Note the above assumes you have the follow prerequisites installed on your system and **in your shell's search path**:

1. Gcc (or clang) for the target architecture
2. Rust; any nightly build >=nightly-2021-11-05 should work. A default version is set in the build/setup.sh script; if that is not what you are using either edit the shell script or export `CANTRIP_RUST_VERSION` in each shell where you work. Beware that we use various nightly-only features that are not supported by stable versions of Rust (e.g. to override the default TLS model).
3. The python tempita module.
4. Whichever simulator seL4 expects for your target architecture; e.g. for aarch64 this is qemu-system-aarch64.

Because Sparrow is a CAmkES project you also need CAmkES dependencies.

1.1.1 Testing on LAPAC71H

- **repo**

```
[mydev11@koonuc15x-1 Sparrow]$ which repo
/usr/bin/repo
[mydev11@koonuc15x-1 Sparrow]$ repo --version
repo version v2.34.1
```

- **procedure:**

```
[mydev11@koonuc15x-1 Sparrow]$ repo init -u https://github.com/AmbiML/sparrow-manifest -m sparrow-manifest.xml
Downloading Repo source from https://mirrors.tuna.tsinghua.edu.cn/git/git-repo/
remote: Enumerating objects: 4535, done.
remote: Counting objects: 100% (4535/4535), done.
remote: Compressing objects: 100% (2177/2177), done.
remote: Total 8124 (delta 4025), reused 2358 (delta 2358), pack-reused 3589
Receiving objects: 100% (8124/8124), 3.89 MiB | 5.75 MiB/s, done.
Resolving deltas: 100% (5214/5214), done.
...
repo has been initialized in /opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow
[mydev11@koonuc15x-1 Sparrow]$
```

```
[mydev11@koonuc15x-1 Sparrow]$ repo sync -j$(nproc)
...
Fetching: 100% (25/25), done in 5m14.859s
repo sync has finished successfully.
[mydev11@koonuc15x-1 Sparrow]$
[mydev11@koonuc15x-1 Sparrow]$ du -sh
219M    .
```

...

II. Practicing Sparrow

```
[mydev11@koonuc15x-1 Sparrow]$ export PLATFORM=rpi3
[mydev11@koonuc15x-1 Sparrow]$
[mydev11@koonuc15x-1 Sparrow]$ source build/setup.sh
========================================
ROOTDIR=/opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow
OUT=/opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/out
PLATFORM=rpi3
PYTHON_SPARROW_ENV=/opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/cache/rpi3-venv
========================================

Type 'm [target]' to build.
```

```
Targets available are:

cantrip cantrip-build-debug-prepare cantrip-build-release-prepare
cantrip-builtins cantrip-builtins-debug cantrip-builtins-release
cantrip-bundle-debug cantrip-bundle-release cantrip-clean cantrip-clean-headers
cantrip-flatbuffers cantrip-gen-headers cargo_test_cantrip
cargo_test_cantrip_os_common_logger cargo_test_cantrip_os_common_slot_allocator
cargo_test_cantrip_proc_interface cargo_test_cantrip_proc_manager
cargo_test_debugconsole_zmodem clean collate_cantrip_rust_toolchain
collate_matcha_rust_toolchain collate_rust_toolchains debug-simulation
elfconvert fibonacci_debug fibonacci_debug_c fibonacci_debug_rust
fibonacci_release fibonacci_release_c fibonacci_release_rust flatbuffers
flatbuffers-clean hello_debug hello_debug_c hello_debug_rust hello_release
hello_release_c hello_release_rust install_gcc install_llvm install_rust
keyval_debug keyval_release logtest_debug logtest_release minisel_debug
minisel_release mltest_debug mltest_release panic_debug panic_release
prereqs qemu qemu_clean qemu_presence_check rust_presence_check sel4test
sel4test+wrapper sel4test-bundle-debug sel4test-bundles sel4test-clean
sel4test-wrapper-bundle-debug simulate simulate-debug spike suicide_debug
suicide_release timer_debug timer_release toolchain_clean tools

To get more information on a target, use 'hmm [target]'
```

II. Practicing Sparrow

```
[mydev11@koonuc15x-1 Sparrow]$ m simulate-debug
mkdir -p /opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/out/cantrip/aarch64-unknown-elf/debug
cmake -B /opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/out/cantrip/aarch64-unknown-elf/debug
-G Ninja \
        -DCROSS_COMPILER_PREFIX=aarch64-none-linux-gnu- \
        -DRUST_TARGET=aarch64-unknown-none \
        -DPLATFORM=rpi3 \
        -DSIMULATION=0 \
        -DSEL4_CACHE_DIR=/opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/cache/sel4-debug \
        -DRELEASE=OFF \
        -DRUST_GLOBAL_FEATURES=CONFIG_PLAT_BCM2837 \
        -DCMAKE_C_FLAGS_DEBUG=-g \
        -DAARCH64=1 \
  /opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/cantrip/projects/cantrip
-- Set platform details from PLATFORM=rpi3
--   KernelPlatform: bcm2837
--   KernelARMPlatform: rpi3
-- Setting from flags KernelSel4Arch: aarch64
-- Found seL4: /opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/cantrip/kernel
-- The C compiler identification is unknown
-- The CXX compiler identification is unknown
-- The ASM compiler identification is unknown
-- Found assembler: aarch64-none-linux-gnu-gcc
CMake Error at CMakeLists.txt:9 (project):
  The CMAKE_C_COMPILER:

    aarch64-none-linux-gnu-gcc

  is not a full path and was not found in the PATH.

  Tell CMake where to find the compiler by setting either the environment
  variable "CC" or the CMake cache entry CMAKE_C_COMPILER to the full path to
  the compiler, or to the compiler name if it is in the PATH.


CMake Error at CMakeLists.txt:9 (project):
  The CMAKE_CXX_COMPILER:

    aarch64-none-linux-gnu-g++

  is not a full path and was not found in the PATH.

  Tell CMake where to find the compiler by setting either the environment
  variable "CXX" or the CMake cache entry CMAKE_CXX_COMPILER to the full path
  to the compiler, or to the compiler name if it is in the PATH.


CMake Error at CMakeLists.txt:9 (project):
  The CMAKE_ASM_COMPILER:

    aarch64-none-linux-gnu-gcc

  is not a full path and was not found in the PATH.

  Tell CMake where to find the compiler by setting either the environment
  variable "ASM" or the CMake cache entry CMAKE_ASM_COMPILER to the full path
  to the compiler, or to the compiler name if it is in the PATH.


-- Warning: Did not find file Compiler/-ASM
-- Configuring incomplete, errors occurred!
make: *** [/opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/build/cantrip.mk:125: /opt/MyWorkSpa
ce/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/out/cantrip/aarch64-unknown-elf/debug/kernel/kernel.elf] Erro
r 1
[mydev11@koonuc15x-1 Sparrow]$
```

II. Practicing Sparrow

```
[mydev11@koonuc15x-1 Sparrow]$ grep -ir "aarch64-none-linux-gnu-" .
./build/platforms/rpi3/setup.sh:export C_PREFIX="aarch64-none-linux-gnu-"
./scripts/build-camkes.sh:    CROSS_COMPILER_PREFIX="aarch64-none-linux-gnu-"
./scripts/build-sparrow.sh:    CROSS_COMPILER_PREFIX=${CROSS_COMPILER_PREFIX:-"aarch64-none-linux-gnu-"}
```

```
[mydev11@koonuc15x-1 Sparrow]$ grep -ir "developer.arm.com" .
./cantrip/kernel/src/arch/arm/config.cmake:    # (https://developer.arm.com/documentation/100095/0001/memory-management
-unit/about-the-mmu)
./cantrip/kernel/src/arch/arm/config.cmake:    # According to https://developer.arm.com/documentation/100095/0001/funct
ional-description/about-the-cortex-a72-processor-functions/components-of-the-processor
./cantrip/projects/util_libs/libplatsupport/src/plat/bcm2711/pl011_serial.c: * More information: https://developer.arm.
com/documentation/ddi0183/g/programmers-model/register-descriptions/fractional-baud-rate-register--uartfbrd
./scripts/build-camkes.sh:# wget https://developer.arm.com/-/media/Files/downloads/gnu/11.2-2022.02/binrel/gcc-arm-11.2
-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz
./scripts/build-sparrow.sh:# wget https://developer.arm.com/-/media/Files/downloads/gnu/11.2-2022.02/binrel/gcc-arm-11.
2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz
[mydev11@koonuc15x-1 Sparrow]$
```

install a new toolchain on host as above from https://developer.arm.com/-/media/Files/downloads/gnu/11.2-2022.02/binrel/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz and add it to $PATH temporarily for testing:

```
[mydev11@koonuc15x-1 Sparrow]$ export PATH=/opt/MyWorkSpace/DevSW/Toolchain/gcc-arm-11.2-2022.02-x86_64-aarch64-none-li
nux-gnu/bin:$PATH
```

...

```
-- The C compiler identification is unknown
-- The CXX compiler identification is unknown
-- The ASM compiler identification is GNU
-- Found assembler: /opt/MyWorkSpace/DevSW/Toolchain/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin/aarch64-non
e-linux-gnu-gcc
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - failed
-- Check for working C compiler: /opt/MyWorkSpace/DevSW/Toolchain/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bi
n/aarch64-none-linux-gnu-gcc
-- Check for working C compiler: /opt/MyWorkSpace/DevSW/Toolchain/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bi
n/aarch64-none-linux-gnu-gcc - broken
```

II. Practicing Sparrow

```
CMake Error at /usr/share/cmake/Modules/CMakeTestCCompiler.cmake:67 (message):
  The C compiler

    "/opt/MyWorkSpace/DevSW/Toolchain/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-gcc
"

  is not able to compile a simple test program.

  It fails with the following output:

    Change Dir: /opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/out/cantrip/aarch64-unknown-elf
/debug/CMakeFiles/CMakeScratch/TryCompile-zZFIOB

    Run Build Command(s):/usr/bin/ninja-build -v cmTC_fdbe9 && [1/2] /usr/bin/ccache /opt/MyWorkSpace/DevSW/Toolchain/g
cc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-gcc    -o CMakeFiles/cmTC_fdbe9.dir/testCC
ompiler.c.obj -c /opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/out/cantrip/aarch64-unknown-el
f/debug/CMakeFiles/CMakeScratch/TryCompile-zZFIOB/testCCompiler.c
    FAILED: CMakeFiles/cmTC_fdbe9.dir/testCCompiler.c.obj
    /usr/bin/ccache /opt/MyWorkSpace/DevSW/Toolchain/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/bin/aarch64-non
e-linux-gnu-gcc    -o CMakeFiles/cmTC_fdbe9.dir/testCCompiler.c.obj -c /opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/
AmbiML/Official/Sparrow/out/cantrip/aarch64-unknown-elf/debug/CMakeFiles/CMakeScratch/TryCompile-zZFIOB/testCCompiler.c
    aarch64-none-linux-gnu-gcc: fatal error: cannot execute ‘cc1’: execvp: No such file or directory
    compilation terminated.
    ninja: build stopped: subcommand failed.




  CMake will not be able to correctly generate this project.
Call Stack (most recent call first):
  CMakeLists.txt:9 (project)


-- Configuring incomplete, errors occurred!
make: *** [/opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/build/cantrip.mk:125: /opt/MyWorkSpa
ce/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/out/cantrip/aarch64-unknown-elf/debug/kernel/kernel.elf] Erro
r 1
[mydev11@koonuc15x-1 Sparrow]$
```

1.1.2 Testing on RPi4

- **procedure:**

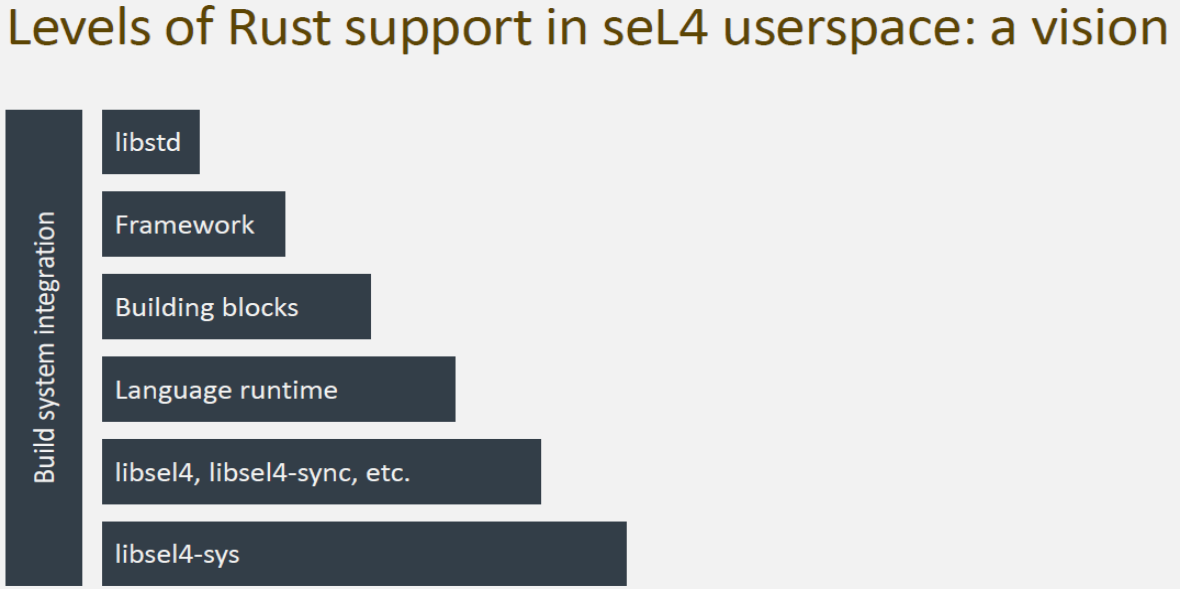
  disable/remove the prefix like ‘C_PREFIX’， ‘CROSS_COMPILER_PREFIX’ in the build system

```
-- The C compiler identification is unknown
-- The CXX compiler identification is unknown
-- The ASM compiler identification is unknown
-- Found assembler: aarch64-linux-gnu-gcc
CMake Error at CMakeLists.txt:9 (project):
  The CMAKE_C_COMPILER:

    aarch64-linux-gnu-gcc

  is not a full path and was not found in the PATH.

  Tell CMake where to find the compiler by setting either the environment
  variable "CC" or the CMake cache entry CMAKE_C_COMPILER to the full path to
  the compiler, or to the compiler name if it is in the PATH.


CMake Error at CMakeLists.txt:9 (project):
  The CMAKE_CXX_COMPILER:

    aarch64-linux-gnu-g++

  is not a full path and was not found in the PATH.

  Tell CMake where to find the compiler by setting either the environment
  variable "CXX" or the CMake cache entry CMAKE_CXX_COMPILER to the full path
  to the compiler, or to the compiler name if it is in the PATH.

CMake Error at CMakeLists.txt:9 (project):
  The CMAKE_ASM_COMPILER:

    aarch64-linux-gnu-gcc

  is not a full path and was not found in the PATH.

  Tell CMake where to find the compiler by setting either the environment
  variable "ASM" or the CMake cache entry CMAKE_ASM_COMPILER to the full path
  to the compiler, or to the compiler name if it is in the PATH.


-- Warning: Did not find file Compiler/-ASM
-- Configuring incomplete, errors occurred!
make: *** [/opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/build/cantrip.mk:125: /opt/MyWorkSpace/MyProjs/Xianbei/SecuritySystem/AmbiML/Official/Sparrow/out/cantrip/aarch64-unknown-elf/debug/kernel/kernel.elf] Error 1
[mydev@fedora Sparrow]$
```

2) Rust support in seL4 userspace

Overview

- **A good summary from "Rust support in seL4 userspace：Present and future", Nick Spinale, seL4 Summit 2022:**



2.1 Rust-centric Pop!_OS

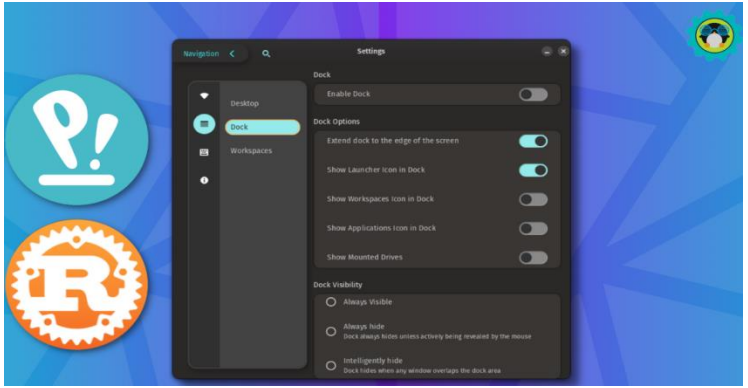
2.1.1 Rust-Written Desktop

- **https://www.phoronix.com/scan.php?page=news_item&px=Pop-OS-New-Rust-Desktop**

Stemming from a Reddit discussion over the possibility of seeing a KDE flavor of Pop!_OS, it was brought up by one of their own engineers they are working on their "own desktop".

System76 engineer and Pop!_OS maintainer Michael Murphy "mmstick" commented that System76 will be its own desktop. When further poked about that whether that means a fork from GNOME, the response was "*No it is its own thing written in Rust.*"

Word of System76 making their "own" desktop not based on GNOME does follow some recent friction between Pop!_OS and GNOME developers over their approach to theming and customizations. Ultimately it will be interesting to see how it plays out. As well, aside from leveraging the Rust programming language, it will be interesting to see ultimately how this plays out and what features are pursued. Additionally, it remains to be seen how quickly they will be able to shift away from a GNOME base for their Linux desktop and whether they plan to use any GNOME components at all as part of their new desktop effort.



Source: https://news.itsfoss.com/system76-rust-cosmic-desktop/qqqqqqqqqqqqqqqqq

2.1.2 System76-Scheduler

- **https://github.com/pop-os/system76-scheduler**

Auto-configure CFS and process priorities for improved desktop responsiveness.

Scheduling service which optimizes Linux's CPU scheduler and automatically assigns process priorities for improved desktop responsiveness. Low latency CPU scheduling will be activated automatically when on AC, and the default scheduling latencies set on battery. Processes are regularly sweeped and assigned process priorities based on configuration files. When combined with pop-shell, foreground processes and their sub-processes will be given higher process priority.

These changes result in a noticeable improvement in the experienced smoothness and performance of applications and games. The improved responsiveness of applications is most noticeable on older systems with budget hardware, whereas games will benefit from higher framerates and reduced jitter. This is because background applications and services will be given a smaller portion of leftover CPU budget after the active process has had the most time on the CPU.

- **https://www.phoronix.com/scan.php?page=news_item&px=System76-Scheduler-1.1**

System76-Scheduler 1.1 was released today for optimizing the Linux CPU scheduler and automatically adjusting process priorities in the name of enhanced desktop responsiveness. This scheduler also takes into account whether running on AC or battery power for laptops to make additional optimizations.

With the new v1.1 release, the scheduler now sets the kernel preempt mode to "full" on the responsive profile while using "voluntary" on battery power.

The updated scheduler also adds new default priorities for common background processes and fixes priority assignments from configurations being overrode by background/foreground priority adjustments.

Among the added "high priority" process defaults are for Steam and X.Org while being set to "low priority" are processes like the daemons for CUPS, Docker, Bluetooth, Avahi, Fwupd, UPower, UDisks, and more. Setting to the "absolute lowest priority" now are tasks like BOINC and the Folding@HOME client.

2.2 Rust for Cloud Native

- **https://rust-cloud-native.github.io/**

Applications and Services

- apache/incubator-teaclave: open source universal secure computing platform, making computation on privacy-s simple
- bottlerocket-os/bottlerocket: an operating system designed for hosting containers
- containers/krunvm: manage lightweight VMs created from OCI images
- containers/youki: a container runtime written in Rust
- datafuselabs/datafuse: A Modern Real-Time Data Processing & Analytics DBMS with Cloud-Native Architecture, built to make the Data Cloud easy
- firecracker-microvm/firecracker: secure and fast microVMs for serverless computing
- infinyon/fluvio: Cloud-native real-time data streaming platform with in-line computation capabilities
- krustlet/krustlet: Kubernetes Rust Kubelet
- kube-rs/controller-rs: a Kubernetes example controller
- kube-rs/version-rs: example Kubernetes reflector and web server
- kubewarden/policy-server: webhook server that evaluates WebAssembly policies to validate Kubernetes requests
- linkerd/linkerd2-proxy: a purpose-built proxy for the Linkerd service mesh
- openebs/mayastor: A cloud native declarative data plane in containers for containers
- rancher-sandbox/lockc: eBPF-based MAC security audit for container workloads
- tikv/tikv: distributed transactional key-value database
- tremor-rs/tremor-runtime: an event processing system that supporting complex workflows such as aggregation, rollups, an ETL language, and a query language
- valeriansaliou/sonic: fast, lightweight & schema-less search backend
- WasmEdge/WasmEdge: WasmEdge is a high-performance WebAssembly (Wasm) Virtual Machine (VM) runtime, which enables serverless functions to be embedded into any software platform; from cloud's edge to SaaS to automobiles

Libraries

- CNI Plugins: crate/framework to write CNI (container networking) plugins in Rust (includes a few custom plugins as well)
- containers/libkrun: a dynamic library providing Virtualization-based process isolation capabilities
- kube-rs/kube-rs: Kubernetes Rust client and async controller runtime
- qovery/engine: Qovery Engine is an open-source abstraction layer library that turns easy apps deployment on AWS, GCP, Azure, and other Cloud providers in just a few minutes
- open-telemetry/opentelemetry-rust: OpenTelemetry is a set of APIs, SDKs, tooling and integrations that are designed for the creation and management of telemetry data such as traces, metrics, and logs.

- **Our new talk "Rust-based Container Runtimes" is coming soon.**

2.3 Unified runtime for eBPF and Wasm

Summary

- Our previous talks **"GraalVM-based unified runtime for eBPF & Wasm"** at GOTC 2021 (Shenzhen) & **"Revisiting GraalVM-based unified runtime for eBPF & Wasm"** at OpenInfra Days China 2021(Beijing), and the third-round discussion of this topic will come in this year, which includes more update for **Rust** related experiments…

2.4 Replace C++ with Rust in AI frameworks

Ideas

- Today, most of the main stream AI frameworks such like Tensorflow, PyTorch and MXNet embrace **Python+C++** for their software layers design.
- **While Rust is coming!**
  https://githubmemory.com/repo/vaaaaanquish/Awesome-Rust-MachineLearning

Ray.Rust

- A reimplementation of project **Ray**(the distributed training framework that behinds many LLMs) by leveraging **Python + Rust** to instead of **Python + C++** within current implementation;
- For some initial design, you may refer to our previous talk **"Ray – A Swiss Army Knife for Distributed Computing & AI"** at COSCon 2022 (Online);
- Our new talk **"The first exploration of Ray.Rust"** will come soon.

1) Develop with Renode

1.1 Rust peripheral support

- It has been possible to design Renode peripheral models in a variety of languages such as C#, Python and C.
- Adding Rust peripherals in Renode now:

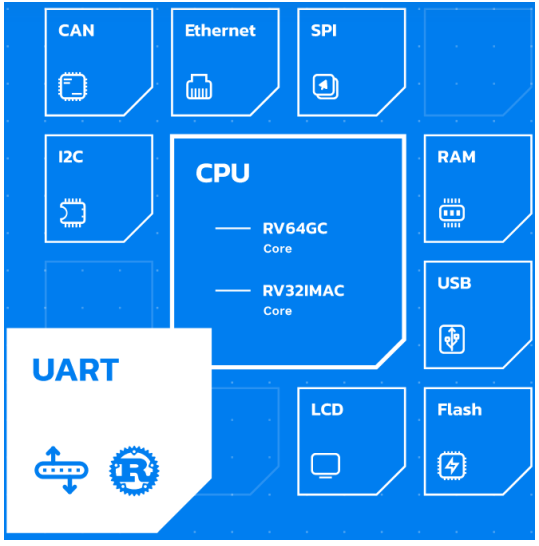
  https://antmicro.com/blog/2021/07/rust-peripheral-support-in-renode/

  A POC sample of Rust UART peripheral.

  https://github.com/antmicro/renode-rust-example

```
[mydev11@koonuc15x-1 Rust]$ tree renode-rust-example/
renode-rust-example/
├── Cargo.toml
├── fe310_rust.repl
├── fe310-rust.resc
├── fe310_rust.robot
├── LICENSE
├── README.md
└── rust_uart.rs
```

```
[mydev11@koonuc15x-1 Rust]$ cat renode-rust-example/Cargo.toml
[package]
name = "rust_uart"
version = "0.1.0"
edition = "2018"

[lib]
crate-type = ["cdylib"]
path = "rust_uart.rs"

[dependencies]
lazy_static = "1.4.0"
bitstream-io = "1.1.0"
[mydev11@koonuc15x-1 Rust]$
```



III. Sparrow Development

The Rust capability was added by integrating WebAssembly into Renode's core infrastructure, and Rust-based peripheral models can have access to all of Renode's functionality by importing and calling methods implemented directly in C#:

```
public abstract class WASMExports {
        public abstract void Reset();
        public abstract void WriteChar(int value);
        public abstract int ReadDoubleWord(long offset);
        public abstract void WriteDoubleWord(long offset, int
value);
}
```

```
#[export_name = "Reset"] // exported to wasm
pub unsafe extern fn reset() {
        clear_buffer();
        reset_registers();
        update_interrupts();
}
```

```
#[link(wasm_import_module = "uart")]
extern { // imported from C#
        #[link_name = "SetIRQ"]
        fn set_irq_inner(value: i32);
        #[link_name = "InvokeCharReceived"]
        fn invoke_char_received_inner(character: i32);
}
```

III. Sparrow Development

Take a look at how the reading from RECEIVE_DATA and writing to TRANSMIT_DATA were implemented in Rust:

```
RECEIVE_DATA_OFFSET => {
        if QUEUE_COUNT == 0 { //EMPTY
                RECEIVE_DATA |= 1 << 31;
        } else {
                RECEIVE_DATA &= ! (1 << 31);
        }
        let mut bytes: [u8; 4] = RECEIVE_DATA.to_le_bytes();
        let output: &mut [u8] = &mut bytes;
        {
                let mut writer = BitWriter::endian(output,
LittleEndian);
                let (success, character) = try_get_character();
                let byts: [u8; 1] = [character];
                if success {
                        writer.write_bytes(&byts).unwrap();
                }
        }
        RECEIVE_DATA = u32::from_le_bytes(bytes);
        return RECEIVE_DATA;
}
```

```
TRANSMIT_DATA_OFFSET => {
        TRANSMIT_DATA = value;
        let bytes = value.to_le_bytes();
        let mut cursor = Cursor::new(&bytes);
        {
                let mut reader = BitReader::endian(&mut cursor,
LittleEndian);
                let character = reader.read(8).unwrap();
                if TRANSMIT_ENABLE {
                        transmit_character(character);
                        update_interrupts();
                }
        }
}
```

III. Sparrow Development

build the Renode Rust UART peripheral example:

cargo build --target wasm32-unknown-unknown --release --lib

```
[mydev11@koonuc15x-1 renode-rust-example]$ cargo build --target wasm32-unknown-unknown --release --lib
    Updating crates.io index
   Compiling bitstream-io v1.6.0
   Compiling lazy_static v1.4.0
   Compiling rust_uart v0.1.0 (/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/Rust/Tests/Renode-Rust-Ex
ample/renode-rust-example)
    Finished release [optimized] target(s) in 1.41s
[mydev11@koonuc15x-1 renode-rust-example]$
```

```
[mydev11@koonuc15x-1 renode-rust-example]$ tree ./target/wasm32-unknown-unknown/release
./target/wasm32-unknown-unknown/release
├── build
├── deps
│   ├── bitstream_io-a2cb41a414fcb70c.d
│   ├── lazy_static-6ec17a74ea9556bd.d
│   ├── libbitstream_io-a2cb41a414fcb70c.rlib
│   ├── libbitstream_io-a2cb41a414fcb70c.rmeta
│   ├── liblazy_static-6ec17a74ea9556bd.rlib
│   ├── liblazy_static-6ec17a74ea9556bd.rmeta
│   ├── rust_uart.d
│   └── rust_uart.wasm
├── examples
├── incremental
├── rust_uart.d
└── rust_uart.wasm
```

```
[mydev11@koonuc15x-1 renode-rust-example]$ file ./target/wasm32-unknown-unknown/release/rust_uart.wasm
./target/wasm32-unknown-unknown/release/rust_uart.wasm: WebAssembly (wasm) binary module version 0x1 (MVP)
[mydev11@koonuc15x-1 renode-rust-example]$
```

III. Sparrow Development

on branch "26999-rust_uart":

https://github.com/renode/renode/commit/8a3b45091022df7d4acf8579754b86f2024606d8

```
[mydev11@koonuc15x-1 renode-master_rust]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
  (commit or discard the untracked or modified content in submodules)
        modified:   src/Infrastructure (modified content, untracked content)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        lib/WebAssembly.dll

no changes added to commit (use "git add" and/or "git commit -a")
[mydev11@koonuc15x-1 renode-master_rust]$
[mydev11@koonuc15x-1 renode-master_rust]$ pushd src/Infrastructure
/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/src/Infrastructure /opt/MyWorkSpace/MyProjs/Si
m-Modeling/IoT-Edge/Renode/renode-master_rust
[mydev11@koonuc15x-1 Infrastructure]$
[mydev11@koonuc15x-1 Infrastructure]$ git status
HEAD detached at 75ed724a
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/Emulator/Peripherals/Peripherals.csproj

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        src/Emulator/Peripherals/Peripherals/UART/RustUART.cs

no changes added to commit (use "git add" and/or "git commit -a")
[mydev11@koonuc15x-1 Infrastructure]$
```

III. Sparrow Development

```
[mydev11@koonuc15x-1 Infrastructure]$ git diff
diff --git a/src/Emulator/Peripherals/Peripherals.csproj b/src/Emulator/Peripherals/Peripherals.csproj
index 8d57eea6..a6f35b80 100644
--- a/src/Emulator/Peripherals/Peripherals.csproj
+++ b/src/Emulator/Peripherals/Peripherals.csproj
@@ -48,6 +48,9 @@
     <Reference Include="FlatBuffers">
       <HintPath>../../../../../lib/resources/libraries/FlatBuffers.dll</HintPath>
     </Reference>
+    <Reference Include="WebAssembly">
+      <HintPath>..\..\..\..\..\lib\WebAssembly.dll</HintPath>
+    </Reference>
   </ItemGroup>
   <ItemGroup>
     <Compile Include="Peripherals\Timers\EFR32_RTCC.cs" />
@@ -557,6 +560,7 @@
     <Compile Include="Peripherals\Timers\EFR32xG2_RTCC.cs" />
     <Compile Include="Peripherals\Timers\EFR32_RTCCCounter.cs" />
     <Compile Include="Peripherals\Miscellaneous\STM32H7_HardwareSemaphore.cs" />
+    <Compile Include="Peripherals\UART\RustUART.cs" />
   </ItemGroup>
   <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />
   <ProjectExtensions>
[mydev11@koonuc15x-1 Infrastructure]$
```

III. Sparrow Development

apply the above patch to master branch and build Renode from source:

https://renode.readthedocs.io/en/latest/advanced/building_from_sources.html

```
Cloning into '/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/tools/building/../../lib/resourc
es'...

>>>> xbuild tool is deprecated and will be removed in future updates, use msbuild instead <<<<

XBuild Engine Version 14.0
Mono, Version 6.12.0.182
Copyright (C) 2005-2013 Various Mono authors
Loading default tasks for ToolsVersion: 14.0 from /usr/lib/mono/xbuild/14.0/bin/Microsoft.Common.tasks

Build started 6/14/2023 4:08:40 AM.
__________________________________________________
/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/src/Renode/Renode.csproj: Importing project /o
pt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/output/properties.csproj
/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/src/Renode/Renode.csproj: Importing project /u
sr/lib/mono/xbuild/14.0/bin/Microsoft.CSharp.targets from extension path /usr/lib/mono/xbuild
/usr/lib/mono/xbuild/14.0/bin/Microsoft.CSharp.targets: Importing project /usr/lib/mono/xbuild/14.0/bin/Microsoft.Commo
n.targets
/usr/lib/mono/xbuild/14.0/bin/Microsoft.Common.targets: Importing project /usr/lib/mono/xbuild/14.0/Microsoft.Common.pr
ops from extension path /usr/lib/mono/xbuild
```

...

```
                Done building target "Build" in project "/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-m
aster_rust/tests/unit-tests/RenodeTests/RenodeTests.csproj" ("/usr/lib/mono/xbuild/14.0/bin/Microsoft.Common.targets").
                Done building project "/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/tests/u
nit-tests/RenodeTests/RenodeTests.csproj".
        Done executing task "MSBuild"
        Task "CallTarget"
                Using task CallTarget from Microsoft.Build.Tasks.CallTarget, Microsoft.Build.Tasks.Core, Version=14.0.0
.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
        Done executing task "CallTarget"
        Done building target "Build" in project "/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_ru
st/Renode.sln".
Done building target "Build" in project "/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/Renod
e.sln" ("/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/Renode.sln").
Done building project "/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/Renode.sln".

Build succeeded.

Warnings:
```

III. Sparrow Development

...

```
ERROR: License copying failed!

Required file not found: 'WebAssembly-license'. Provide it or exclude 'WebAssembly.dll'.
```

add a WebAssembly-license file to $SRC_RENODE/lib/resources/libraries/ and rebuild:

```
[mydev11@koonuc15x-1 resources]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        libraries/WebAssembly-license

nothing added to commit but untracked files present (use "git add" to track)
[mydev11@koonuc15x-1 resources]$
```

```
[mydev11@koonuc15x-1 renode-master_rust]$ ./build.sh -pn -v
Required renode-resources repository already downloaded. To repeat the process remove .renode_libs_fetched file.

>>>> xbuild tool is deprecated and will be removed in future updates, use msbuild instead <<<<

XBuild Engine Version 14.0
Mono, Version 6.12.0.182
Copyright (C) 2005-2013 Various Mono authors
Loading default tasks for ToolsVersion: 14.0 from /usr/lib/mono/xbuild/14.0/bin/Microsoft.Common.tasks

Build started 6/14/2023 2:02:02 PM.
__________________________________________________
/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/src/Renode/Renode.csproj: Importing project /o
pt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/output/properties.csproj
/opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/src/Renode/Renode.csproj: Importing project /u
sr/lib/mono/xbuild/14.0/bin/Microsoft.CSharp.targets from extension path /usr/lib/mono/xbuild
```

III. Sparrow Development

```
[mydev11@koonuc15x-1 renode-master_rust]$ tree -L 1 tools/packaging/
tools/packaging/
├── common_copy_files.sh
├── common_copy_licenses.sh
├── common_make_linux_portable.sh
├── common_make_packages.sh
├── conda
├── export_linux_workdir.sh
├── linux
├── linux_portable
├── macos
├── make_linux_packages.sh
├── make_linux_portable_dotnet.sh
├── make_linux_portable.sh
├── make_osx_packages.sh
├── make_source_package.sh
├── make_windows_packages.sh
├── renode_1.13.3
├── renode_1.13.3+20230614gitd6908da
└── windows
```

```
[mydev11@koonuc15x-1 renode-master_rust]$ tree -L 1 tools/packaging/renode_1.13.3+20230614gitd6908da
tools/packaging/renode_1.13.3+20230614gitd6908da
├── bin
├── licenses
├── platforms
├── plugins
├── scripts
├── tests
└── tools
```

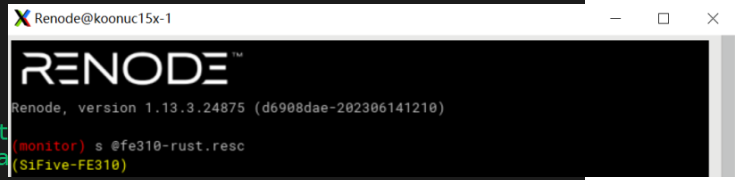
```
[mydev11@koonuc15x-1 renode-master_rust]$ file tools/packaging/renode_1.13.3+20230614gitd6908da/bin/Renode.exe
tools/packaging/renode_1.13.3+20230614gitd6908da/bin/Renode.exe: PE32+ executable (console) x86-64 Mono/.Net assembly,
for MS Windows, 4 sections
```

III. Sparrow Development

simulating:

```
[mydev11@koonuc15x-1 renode-rust-example]$ mono /opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/tools/packaging/renode_1.13.3+20230614gitd6908da/bin/Renode.exe
14:54:52.1174 [INFO] Loaded monitor commands from: /opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/renode-master_rust/tools/packaging/renode_1.13.3+20230614gitd6908da/scripts/monitor.py
```

```
14:56:05.4484 [INFO] Including script: /opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/Rust/Tests/Renode-Rust-Example/renode-rust-example/fe310-rust.resc
14:56:05.4549 [INFO] System bus created.
14:56:05.8206 [INFO] sysbus: Loaded SVD: /tmp/renode-37138/8deccda2-c3fc-4152-abb6-3d82df563d1c.tmp. Name: FE310. Description: E31 CPU Coreplex, high-performance, 32-bit RV32IMAC core
      .
14:56:05.8789 [INFO] uart0: Initialized the peripheral to /opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/Rust/Tests/Renode-Rust-Example/renode-rust-example/target/wasm32-unknown-unknown/release/rust_uart.wasm
14:56:05.8856 [INFO] uart1: Initialized the peripheral to /opt/MyWorkSpace/MyProjs/Sim-Modeling/IoT-Edge/Renode/Rust/Tests/Renode-Rust-Example/renode-rust-example/target/wasm32-unknown-unknown/release/rust_uart.wasm
14:56:05.8869 [INFO] Downloading https://dl.antmicro.com/projects/renode/zephyr-fe310-shell.elf-s_323068-cf87169150ecdb30ad5a14c87ae209c53dd3eca2.
14:56:08.6679 [INFO] Downloading: https://dl.antmicro.com/projects/renode/zephyr-fe310-shell.elf-s_323068-cf87169150ecdb30ad5a14c87ae209c53dd3eca2
    Progress: 60% (192KiB/315.5KiB)
    Speed: 396.82KB/s.
14:56:09.2681 [INFO] Download done.
14:56:09.3111 [INFO] sysbus: Loading segment of 20520 bytes length
14:56:09.3207 [INFO] sysbus: Loading segment of 324 bytes length at
14:56:09.3207 [INFO] sysbus: Loading segment of 7944 bytes length a
14:56:09.3297 [INFO] cpu: Setting PC value to 0x20401BC4.
14:56:09.4672 [INFO] SiFive-FE310: Machine started.
14:56:09.4924 [WARNING] plic: Unhandled write to offset 0x200000, value 0x0.
14:56:09.4925 [WARNING] gpioInputs: Unhandled write to offset 0x38, value 0x0.
14:56:09.4930 [WARNING] gpioInputs: Unhandled read from offset 0x3C.
14:56:09.4930 [WARNING] gpioInputs: Unhandled write to offset 0x3C, value 0x0.
```



III. Sparrow Development

...

```
14:56:09.4935 [WARNING] gpioInputs: Unhandled read from offset 0x38.
14:56:09.4936 [WARNING] gpioInputs: Unhandled write to offset 0x38, value 0x400.
14:56:09.4942 [WARNING] sysbus: Write of value 0x60000 to an unimplemented register PRIC:PLLCFG (0x10008008) g
enerated from SVD.
14:56:09.4943 [WARNING] sysbus: Write of value 0x100 to an unimplemented register PRIC:PLLOUTDIV (0x1000800C)
generated from SVD.
14:56:09.4949 [WARNING] sysbus: [cpu: 0x204030E0] (tag: 'PRCI_PLLCFG') ReadDoubleWord from non existing periph
eral at 0x10008008, returning 0xFFFFFFFF.
14:56:09.4949 [WARNING] sysbus: Write of value 0xFFFFFFFF to an unimplemented register PRIC:PLLCFG (0x10008008
) generated from SVD.
14:56:09.4950 [WARNING] sysbus: [cpu: 0x204030F4] (tag: 'PRCI_HFROSCCFG') ReadDoubleWord from non existing per
ipheral at 0x10008000, returning 0xFFFFFFFF.
14:56:09.4950 [WARNING] sysbus: Write of value 0xBFFFFFFF to an unimplemented register PRIC:HFROSCCFG (0x10008
000) generated from SVD.
14:56:09.4991 [WARNING] gpioInputs: Unhandled write to offset 0x10, value 0x0.
14:56:09.4993 [WARNING] gpioInputs: Unhandled write to offset 0x40, value 0x0.
```

This will run a Zephyr RTOS shell example on RISC-V based SiFive FE310 with a Rust implementation of UART:

III. Sparrow Development

1.2 renode-run crate

- https://github.com/jonlamb-gh/renode-run

  A custom Cargo runner that runs Rust firmware in the Renode emulator.

- Installation

```
[mydev11@koonuc15x-1 Renode-Run]$ cargo install renode-run
    Updating crates.io index
  Downloaded renode-run v0.1.2
  Downloaded 1 crate (19.3 KB) in 2.86s
  Installing renode-run v0.1.2
    Updating crates.io index
  Downloaded anstyle v1.0.0
  Downloaded quote v1.0.28
...
Downloaded linux-raw-sys v0.3.8
Downloaded 61 crates (5.1 MB) in 6m 23s (largest was `linux-raw-sys` at 1013.8 KB)
 Compiling proc-macro2 v1.0.60
 Compiling unicode-ident v1.0.9
 Compiling quote v1.0.28
 Compiling serde v1.0.164
 Compiling libc v0.2.146
...
   Compiling cargo_metadata v0.15.4
   Compiling toml v0.7.4
   Compiling cargo_toml v0.15.3
   Compiling renode-run v0.1.2
    Finished release [optimized] target(s) in 6m 43s
  Installing /home/mydev11/.cargo/bin/renode-run
   Installed package `renode-run v0.1.2` (executable `renode-run`)
[mydev11@koonuc15x-1 Renode-Run]$
```

1. Set the Cargo runner

Set `renode-run` as your Cargo runner (`.cargo/config.toml`).

```
[target.'cfg(all(target_arch = "arm", target_os = "none"))']
runner = "renode-run"
```

Languages

Rust 100.0%

III. Sparrow Development

- **an example of Configuration**

```
[package.metadata.renode]
name = 'my-script'
description = 'my renode script - ${FOOKEY} works'
machine-name = 'my-machine'
using-sysbus = true
renode = '${HOME}/repos/forks/renode/renode'
environment-variables = [
    ['FOOKEY', 'FOOVAL'],
    ["MYENV", "MYVAL"],
]
init-commands = [
    'logLevel -1 i2c2',
]
variables = [
    '$tap?="renode-tap0"',
    # Set random board UNIQUE ID
    '''
    python "import _random"
    python "rand = _random.Random()"

    $id1 = `python "print rand.getrandbits(32)"`
    $id2 = `python "print rand.getrandbits(32)"`
    $id3 = `python "print rand.getrandbits(32)"`
    ''',
]
```

```
platform-descriptions = [
    '@platforms/boards/stm32f4_discovery-kit.repl',
    'path/to/dev_board.repl',
    '< ${SOMETHING}/other_dev_board.repl',
    '''
    phy3: Network.EthernetPhysicalLayer @ ethernet 3
        Id1: 0x0000
        Id2: 0x0000
    ''',
    '''
    wss: Python.PythonPeripheral @ sysbus 0x50070000
        size: 0x10
        initable: true
        filename: "${ORIGIN}/sensor_models/wss.py"
    ''',
]
pre-start-commands = [
    '''
    emulation CreateSwitch "switch"
    connector Connect sysbus.ethernet switch
    emulation CreateTap $tap "tap"
    connector Connect host.tap switch
    ''',
    '''
    logFile @/tmp/logfile.log true
    logLevel 3 file
    ''',
    'emulation LogEthernetTraffic',
    'machine StartGdbServer 3333',
]
reset = '''
sysbus LoadELF $bin
sysbus WriteDoubleWord 0x1FFF7A10 $id1
sysbus WriteDoubleWord 0x1FFF7A14 $id2
sysbus WriteDoubleWord 0x1FFF7A18 $id3
'''
```

IV. Wrap-up

- **Rust is playing an more and more important role in HW-SW collaboration! You may look forward to our upcoming follow-ups "The first exploration of Rust for HW-SW collaboration", "Rust for Formal Verification, Coq and SMT" and "Revisiting Renode -- a Swiss Army Knife for RISC-V based HW-SW co-developed system".**

- **Our third-round and perhaps the forth-round discussion of "The eBPF-centric new approach for Hyper-Converged Infrastructure & Edge Computing" will be divided into two series:**

  "ARM + xPython + Rust + Lua + GraalVM + …" and

  "RISC-V + xPython +  **+ Zig + Lua + SmartRuntime…"**

  according to different technology roadmap.

Q & A

鲜卑拓跋枫

扫一扫上面的二维码图案，加我微信

Thank you!