RISC-V® Summit

# Tutorial: Getting Started With seL4 on RISC-V

## With Renode's Help

Jesse Millwood
Embedded Engineer
DornerWorks
jesse.millwood@dornerworks.com

# Jesse Millwood


- Embedded Engineer
- DornerWorks
- Advanced Embedded Platform Group

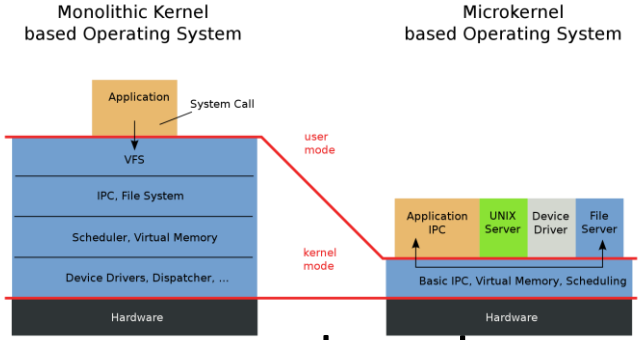
- Objectives
  - seL4 Introduction
  - Using seL4 Test
  - Developing with CAmkES
  - Emulating with Renode

# Secure Your System with seL4

# Why Use seL4?

- It is a microkernel
- It is backed by formal methods
- It is Open Source
- It focuses on security

# seL4: Microkernel




- Microkernel Architecture
  - Very small kernel
  - Pushes driver logic and OS logic to user space threads
- Main Responsibilities
  - Enforce flow control
  - Enforce separation and access control
    - Requires the address translation as specified by the Supervisor Privilege Mode
  - Scheduling

# seL4: Proved Correct

- The core kernel has been expressed in formal specifications
  - Proves:
    - No Overflows
    - No Deadlocks
    - IPC Fastpath gives same results as regular IPC
    - Isolation
    - Capability Access Control
    - Information Flow
- Refinement proofs show that
  - The C code implements only what is specified in the higher level model
  - The binary only implements what is expressed in the C code
- Does not extend to User Space!

# seL4: Open Source

- Originally developed by UNSW/NICTA
- Functional Correctness proof completed in 2009
- Open sourced code and proofs in 2014
  - Huge Undertaking
    - ~20 Person years to complete kernel development and reusable proof frameworks
      - ~ 11.5 Person years to design, implement, and verify the kernel
  - GPL'd kernel
  - Data61 userspace libraries are a mix of MIT,BSD, and GPL (no dynamic linking mechanism in user space)
- Development and Maintenance is taken care of by Data61 (Formed by merging of NICTA and CSRIO)
- Kernel and Libraries are there for you to alter to your needs!
  - Upstreaming increases quality
- Your advancement advances everyone!
  - https://dornerworks.com/tag/open

# seL4: Secure

- Achieves secure access control by using capabilities for operations on all kernel objects / resources
- Kernel Objects:
  - Untyped Memory
  - VSpaces
  - Cspaces
  - IPC
  - Endpoints/Notification
  - Threads

# Sounds Great!

- What can I use it for?
  - seL4 Desktop? Can I check my e-mails?
    - NO, not directly
    - On the horizon?
      - UX/RT https://gitlab.com/uxrt
      - Genode: https://genode.org/index
  - Highly secure static embedded systems?
    - YES
  - No POSIX
    - Possible to implement subsets

# System Architecture and Tools

- Architectures
  - Bare seL4 Applications
  - Hypervisor
  - ADL Based
- Tools
  - CAmkES
  - Cogent
  - Rump Kernels

# Let’s Build A System

- Data61’s seL4 Test
  - Great for testing seL4 on new platforms
  - Great for getting familiar with the surrounding tools and project layouts

# seL4 Test: Platform

- Don’t Have Hardware?
  - Renode
    - Aims to be able to run the same binary as the hardware
    - Supported RISC-V CPUs with supervisor modes:
      - Microchip Polarfire SoC
      - SiFive fu540

# seL4 Test: Get The Tools

- Host Build Setup
  - Requires many packages, differing across distributions and versions
  - https://docs.sel4.systems/HostDependencies
- Install Google's Repo tool
  - https://gerrit.googlesource.com/git-repo/
- Use Data61 Docker setup
  - Just need docker
    - https://docs.docker.com/install/linux/docker-ce/ubuntu/
  - Everything is packaged up for you
  - Just point it to your build directory
  - Can still use external tools (editors, etc..)

# Renode: Install

- Install “mono-complete”
  - https://www.mono-project.com/download/stable/#download-lin
- Install Renode Package from Github Release Page
  - https://github.com/renode/renode/releases/tag/v1.8.2
- Install robot framework python packages (Python 2.7)

```
python -m pip install robotframework==3.0.4 netifaces requests psutil
```

# seL4 Test: Get The Code

- Use the Google Repo Tool
  - Point to an seL4 manifest repository

```
$ mkdir sel4test
$ cd sel4test
$ repo init -u https://github.com/dornerworks/riscvsummit-manifests -m sel4test.xml
$ repo sync
```

jesse@dwbuntu: ~/sel4-projects/sel4test

```
jesse@dwbuntu ~/sel4-projects/sel4test
  % repo init -u https://bitbucket.org/riscvsummit/sel4manfiests                                                    !5567
```

# seL4 Test: Project Layout

```
<manifest>
  <remote fetch="https://github.com/nanopb" name="nanopb"/>
  <remote fetch="https://github.com/seL4" name="upstream-seL4"/>
  <remote fetch="https://github.com/sel4proj" name="upstream-sel4proj"/>
  <remote fetch="https://github.com/Jesse-Millwood" name="jesse-github"/>
  <remote fetch="https://bitbucket.org/riscvsummit/" name="riscvsummit"/>
  <default remote="upstream-seL4" revision="master"/>

  <!-- Core Data61 Repos -->
  <project name="musllibc.git"
           path="projects/musllibc"
           revision="1c6350ea6149e8a6b7de8890c34ba3ed615afe60"/>
  <project name="seL4_libs.git"
           path="projects/seL4_libs"
           revision="c4e061572c9acd2915fa5f1dee1b353db391e45a"/>
  <project name="seL4_tools.git"
           path="tools/seL4"
           revision="afbad19d98166ff6c2b92696f528e92dc75da6ee">
    <linkfile dest="init-build.sh"
              src="cmake-tool/init-build.sh"/>
    <linkfile dest="griddle"
              src="cmake-tool/griddle"/>
  </project>
  <project name="sel4_projects_libs"
           path="projects/sel4_projects_libs"
           remote="upstream-sel4proj"
           revision="33cf99b5ebd36c0ecd2032a4b76f6fb992ad1016"/>
  <project name="sel4runtime.git"
           path="projects/sel4runtime"
           revision="7b71e82ca6802a712475d0e523e689f190303bd3"
           remote="upstream-sel4proj"/>
  <project name="util_libs.git"
           path="projects/util_libs"
           revision="4e14660568aa82b62402a4bdfea234f9dd0c3140"/>
  <project name="nanopb"
```

```
├── easy-settings.cmake -> projects/sel4test/easy-settings.cmake
├── griddle -> tools/seL4/cmake-tool/griddle
├── init-build.sh -> tools/seL4/cmake-tool/init-build.sh
├── kernel
├── nanopb
├── projects
│   ├── musllibc
│   ├── seL4_libs
│   ├── sel4_projects_libs
│   ├── sel4runtime
│   ├── sel4test
│   └── util_libs
├── run-build.sh -> tools/sel4Extra/run-build.sh
├── run-renode.sh -> tools/sel4Extra/run-renode.sh
├── run-robot.sh -> tools/sel4Extra/run-robot.sh
├── start-container.sh -> tools/sel4Extra/start-container.sh
└── tools
    ├── DockerFiles
    ├── riscv-pk
    ├── seL4
    └── sel4Extra
```

# seL4 Test: Enter Build Environment

- If using the Docker Environment, Enter it with make
  - Some developers make aliases for it
- My “start-container.sh” script will start the RISC-V specific container
  - By default it doesn’t pull in the RISC-V tools

jesse@dwbuntu: ~/DornerWorksProjects/RISC-V Summit 2019/Code/Repo-sel4test

```
jesse@dwbuntu ~/DornerWorksProjects/RISC-V Summit 2019/Code/Repo-sel4test
 %                                                                                                                !5396
```

# seL4 Test: Configure The Build

- CMake/Ninja
- Project Root: init-build.sh
  - Pass CMake Variables to this script
    - What Cmake Variables Exist?
      - Look for these files: CMakeLists.txt, settings.cmake, easy-settings.cmake
      - Run “ccmake .” from the build directory
- Make a build directory in the Project Root
  - Run the CMake commands from here

# seL4 Test: Build The Code!

- Simply run “ninja” in the build directory
  - Must have a successful CMake Configuration

# seL4 Test: Scripting Build

- “run-build.sh” script
  - Provided by my tools in “tools/seL4Extra”
  - Creates a directory based on the platform and architecture
  - Changes directory into it
  - Calls “init-build.sh” with the following arguments:
    - -DPLATFORM=hifive
    - -DSIMULATION=FALSE
    - -DCROSS_COMPILER_PREFIX=riscv64-linux-gnu-
    - -DElfloaderImage=elf
    - -DLibSel4PlatSupportUseDebugPutChar=OFF
    - -DSel4testAllowSettingsOverride=True
    - -DCMAKE_BUILD_TYPE=Debug
    - -DRELEASE=False
    - -DLibSel4TestPrintXML=ON
    - -DRISCV64=TRUE

./start-container.sh

```
Step 1/7 : ARG EXTRAS_IMG=extras
Step 2/7 : FROM $EXTRAS_IMG
 ---> a84ed673b16d
Step 3/7 : ARG UID
 ---> Using cache
 ---> 01bdfd46c8df
Step 4/7 : ARG UNAME
 ---> Using cache
 ---> 7cde3f5a7c3e
Step 5/7 : RUN useradd -u ${UID} ${UNAME}     && adduser ${UNAME} sudo     && passwd -d ${UNAME}     && echo 'Defaults        lecture_file = /etc/sudoers.lecture' >> /etc/sudoers     && echo 'Defaults
        lecture = always' >> /etc/sudoers     && echo '##################### Warning! ####################################' > /etc/sudoers.lecture     && echo 'This is an ephemeral docker container!
You can do things to it using' >> /etc/sudoers.lecture     && echo 'sudo, but when you exit, changes made outside of the /host directory' >> /etc/sudoers.lecture     && echo 'will be lost.' >> /etc/su
doers.lecture     && echo 'If you want your changes to be permanent, add them to the ' >> /etc/sudoers.lecture     && echo '    extras.dockerfile' >> /etc/sudoers.lecture     && echo 'in the seL4-CAmk
ES-L4v dockerfiles repo.' >> /etc/sudoers.lecture     && echo '##################################################################' >> /etc/sudoers.lecture     && echo '' >> /etc/sudoers.lecture
&& mkdir /home/${UNAME}     && echo 'echo "___                                  "' >> /home/${UNAME}/.bashrc     && echo 'echo " |    _        _ |_        _    _ |_ |_      "' >> /home/${UNAME}/.bashrc
&& echo 'echo " |  |  |_| _) |_ \)/ (_) |  |_ | ) \/ "' >> /home/${UNAME}/.bashrc     && echo 'echo "                                  /  "' >> /home/${UNAME}/.bashrc     && echo 'echo " __
                    "' >> /home/${UNAME}/.bashrc     && echo 'echo "(_      _ |_  _  _    _                "' >> /home/${UNAME}/.bashrc     && echo 'echo "__) \/ _) |_ (- ||| _)              "'
>> /home/${UNAME}/.bashrc     && echo 'echo "    /                                  "' >> /home/${UNAME}/.bashrc     && echo 'echo "Hello, welcome to the sel4/CAmkES/L4v docker build environment"' >> /
home/${UNAME}/.bashrc     && echo 'export PATH=/scripts/repo:$PATH' >> /home/${UNAME}/.bashrc     && echo 'cd /host' >> /home/${UNAME}/.bashrc     && mkdir -p /isabelle     && chown -R ${UNAME}:${UNAM
E} /isabelle     && ln -s /isabelle /home/${UNAME}/.isabelle     && chown -R ${UNAME}:${UNAME} /home/${UNAME}     && chmod -R ug+rw /home/${UNAME}
 ---> Using cache
 ---> e7bf0c9b0915
Step 6/7 : VOLUME /home/${UNAME}
 ---> Using cache
 ---> eb3aa4cf7c23
Step 7/7 : VOLUME /isabelle
 ---> Using cache
 ---> fdce38e6daa4
Successfully built fdce38e6daa4
Successfully tagged user_img--riscv:latest
docker run \
        -it \
        --hostname in-container \
        --rm \
        -u 1000:1000 \
        -v /home/jesse/DornerWorksProjects/RISC-V\ Summit\ 2019/Code/Repo-sel4test/:/host \
        -v jesse-home:/home/jesse \
        -v /etc/localtime:/etc/localtime:ro \
        user_img--riscv bash
___
 |    _        _ |_        _    _ |_ |_
 |  |  |_| _) |_ \)/ (_) |  |_ | ) \/
                                  /
 __
(_       |_  _  _
__) \/ _) |_ (- |||  _)
    /
Hello, welcome to the sel4/CAmkES/L4v docker build environment
jesse@in-container:/host$ ls
easy-settings.cmake  griddle  init-build.sh  kernel  nanopb  projects  riscv-build.sh  run-build.sh  run-repo.sh  start-container.sh  tools
jesse@in-container:/host$
```

# Renode: Configure Emulation

- Use a Renode Monitor script (*.resc)
  - Provide
    - The seL4 binary
    - The platforms DTB (Gets passed to the bootloader, BBL)

sel4test.resc ×

```
:name seL4 Test
:description Runs seL4 Test on Renode

$name?="sel4test"

using sysbus
mach create $name
$bin?=@./riscv-build-hifive-64/images/sel4test-driver-image-riscv-hifive
$dtb?=@./tools/sel4Extra/renode_support/hifive-renode.dtb
machine LoadPlatformDescription @./tools/sel4Extra/renode_support/sifive-fu540-dw.repl
showAnalyzer uart0

macro reset
"""
   sysbus LoadELF $bin
   sysbus LoadFdt $dtb 0x81000000 "earlyconsole mem=256M@0x80000000"
   e51 SetRegisterUnsafe 11 0x81000000
   u54_1 SetRegisterUnsafe 11 0x81000000
"""
runMacro $reset
```

sel4test.resc × sifive-fu540-dw.repl ×

```
e51: CPU.RiscV64 @ sysbus
    cpuType: "rv64imac"
    hartId: 0
    privilegeArchitecture: PrivilegeArchitecture.Priv1_10
    timeProvider: clint

u54_1: CPU.RiscV64 @ sysbus
    cpuType: "rv64gc"
    hartId: 1
    privilegeArchitecture: PrivilegeArchitecture.Priv1_10
    timeProvider: clint

u54_2: CPU.RiscV64 @ sysbus
    cpuType: "rv64gc"
    hartId: 2
    privilegeArchitecture: PrivilegeArchitecture.Priv1_10
    timeProvider: clint

u54_3: CPU.RiscV64 @ sysbus
    cpuType: "rv64gc"
    hartId: 3
    privilegeArchitecture: PrivilegeArchitecture.Priv1_10
    timeProvider: clint

u54_4: CPU.RiscV64 @ sysbus
    cpuType: "rv64gc"
    hartId: 4
    privilegeArchitecture: PrivilegeArchitecture.Priv1_10
    timeProvider: clint

debug: Memory.MappedMemory @sysbus 0x0
    size: 0x1000

e51DTim: Memory.MappedMemory @ sysbus 0x01000000
    size: 0x2000
```

sel4test.resc × sifive-fu540-dw.repl ×

```

u54Hart1ITim: Memory.MappedMemory @ sysbus 0x01808000
    size: 0x7000

u54Hart2ITim: Memory.MappedMemory @ sysbus 0x01810000
    size: 0x7000

u54Hart3ITim: Memory.MappedMemory @ sysbus 0x01818000
    size: 0x7000

u54Hart4ITim: Memory.MappedMemory @ sysbus 0x01820000
    size: 0x7000

clint: IRQControllers.CoreLevelInterruptor  @ sysbus 0x2000000
    frequency: 1000000
    numberOfTargets: 5
    [0, 1] -> e51@[3, 7]
    [2, 3] -> u54_1@[3, 7]
    [4, 5] -> u54_2@[3, 7]
    [6, 7] -> u54_3@[3, 7]
    [8, 9] -> u54_4@[3, 7]

plic: IRQControllers.PlatformLevelInterruptController @ sysbus 0xc000000
    [0-3] -> e51@[8-11]
    [4-7] -> u54_1@[8-11]
    [8-11] -> u54_2@[8-11]
    [12-15] -> u54_3@[8-11]
    [16-19] -> u54_4@[8-11]
    numberOfSources: 53
    numberOfTargets: 5
    prioritiesEnabled : false

uart0: UART.SiFive_UART @ sysbus 0x10010000
    IRQ -> plic@4
```

# seL4 Test: Run The Code

- Pass your Renode Monitor script to the Renode application
- Caveat: Kernel Modification Needed
  - Renode doesn’t handle “sfence” with arguments
  - The kernel provided in this manifest removes the arguments in order to run

jesse@dwbuntu: ~/sel4-projects/sel4test

```
jesse@dwbuntu ~/sel4-projects/sel4test
  %                                                                                                                    !5583
```

# seL4 Test: Output

```
 <testcase classname="sel4test" name="TLS0001">
 </testcase>
 <testcase classname="sel4test" name="TLS0002">
 </testcase>
 <testcase classname="sel4test" name="TLS0006">
 </testcase>
 <testcase classname="sel4test" name="TRIVIAL0000">
 </testcase>
 <testcase classname="sel4test" name="TRIVIAL0001">
 </testcase>
 <testcase classname="sel4test" name="TRIVIAL0002">
 </testcase>
 <testcase classname="sel4test" name="VSPACE0000">
 </testcase>
 <testcase classname="sel4test" name="VSPACE0002">
 </testcase>
 <testcase classname="sel4test" name="VSPACE0003">
 </testcase>
 <testcase classname="sel4test" name="VSPACE0004">
 </testcase>
 <testcase classname="sel4test" name="VSPACE0005">
 </testcase>
 <testcase classname="sel4test" name="VSPACE0006">
 </testcase>
 <testcase classname="sel4test" name="Test all tests ran">
 </testcase>
</testsuite>
All is well in the universe
```

# Renode Features

- Declaratively Specify Boards, SoC, Connected Environments (RF, Ethernet, etc ..)
  - Means no recompilation needed for adding devices or changing cores
  - Heterogeneous system support
- Scripting Interfaces (Python, Renode Monitor, Robot)
- Instruction Level Emulation Granularity
- Virtual Time
- Networking (Ethernet, Wireless, Wireshark)
- Peripherals and Cores in C#
  - Can include custom peripherals at run time

# Renode: Robot Framework

- Acceptance Based Testing
- Test inputs and outputs to your emulated system
- Renode integration provides direct interaction with the emulated platform

# seL4 Test: Robot Framework

- “run-robot.sh” will run the following script:

```
*** Keywords ***
Prepare Machine
    Execute Command          using sysbus
    Execute Command          mach create "sel4test"
    Execute Command          \$bin?=@${CURDIR}/../../../riscv-build-hifive-64/images/sel4test-driver-image-riscv-hifive
    Execute Command          \$dtb?=@${CURDIR}/hifive-renode.dtb
    Execute Command          machine LoadPlatformDescription @${CURDIR}/sifive-fu540-dw.repl
    Execute Command          showAnalyzer uart0

    Execute Command          sysbus LoadELF $bin
    Execute Command          sysbus LoadFdt $dtb 0x81000000 "earlyconsole mem=256M@0x80000000"
    Execute Command          e51 SetRegisterUnsafe 11 0x81000000
    Execute Command          u54_1 SetRegisterUnsafe 11 0x81000000
```

```
*** Test Cases ***
Should Boot seL4
    [Documentation]           Boots seL4 on SiFive Freedom U540 platform.
    [Tags]                    sel4  uart
    Prepare Machine

    Create Terminal Tester    ${UART}
    Start Emulation

    Wait For Line On Uart   bbl loader                                    timeout=120
    Wait For Line On Uart   ELF-loader started on (HART 1) (NODES 1)      timeout=120

    Provides                  booted-sel4

Should Test There Are Tests
    [Documentation]           Check for the first test from sel4test
    [Tags]                    sel4 test uart

    Requires                  booted-sel4
    Wait For Line On Uart   <testcase classname="sel4test" name="Test that there are tests">    timeout=9999

   Provides                   there-are-tests
```

# seL4 Test: Robot Framework

```
jesse@dwbuntu ~/sel4-projects/sel4test
  % ./run-robot.sh
Preparing suites
Starting suites
Running tools/sel4Extra/renode_support/sel4test.robot
==============================================================================
sel4test
==============================================================================
Should Boot seL4 :: Boots seL4 on SiFive Freedom U540 platform.       | PASS |
------------------------------------------------------------------------------
Should Test There Are Tests :: Check for the first test from sel4test | PASS |
------------------------------------------------------------------------------
sel4test                                                              | PASS |
2 critical tests, 2 passed, 0 failed
2 tests total, 2 passed, 0 failed
==============================================================================
Output:  /home/jesse/sel4-projects/sel4test/sel4test.xml
Cleaning up suites
Aggregating all robot results
Output:  /home/jesse/sel4-projects/sel4test/robot_output.xml
Log:     /home/jesse/sel4-projects/sel4test/log.html
Report:  /home/jesse/sel4-projects/sel4test/report.html
Tests finished successfully :)
```

# TEST SUITE TEST REPORT

LOG

20191123 05:15:08 GMT-05:00 | 3 minutes 6 seconds ago

## Summary Information

| | | | |
|---|---|---|---|
| Status: | All tests passed | HotSpot Action: | - |
| Start Time: | 20191123 05:10:16.362 | End Time: | 20191123 05:15:07.976 |
| Elapsed Time: | 00:04:51.614 | Log File: | log.html |

## Test Statistics

| Total Statistics | Total | Pass | Fail | Elapsed | Pass / Fail |
|---|---|---|---|---|---|
| Critical Tests | 2 | 2 | 0 | 00:04:46 | |
| All Tests | 2 | 2 | 0 | 00:04:46 | |

| Statistics by Tag | Total | Pass | Fail | Elapsed | Pass / Fail |
|---|---|---|---|---|---|
| sel4 | 1 | 1 | 0 | 00:00:07 | |
| sel4 test uart | 1 | 1 | 0 | 00:04:39 | |
| uart | 1 | 1 | 0 | 00:00:07 | |

SUITE **Test Suite** 00:04:51.614

| | |
|---|---|
| **Full Name:** | Test Suite |
| **HotSpot Action:** | - |
| **Source:** | /home/jesse/sel4-projects/sel4test/tools/sel4Extra/renode_support/sel4test.robot |
| **Start / End / Elapsed:** | 20191123 05:10:16.362 / 20191123 05:15:07.976 / 00:04:51.614 |
| **Status:** | 2 critical test, 2 passed, 0 failed<br>2 test total, 2 passed, 0 failed |

SETUP renode-keywords.**Setup** 00:00:05.662

TEARDOWN renode-keywords.**Teardown** 00:00:00.004

TEST **Should Boot seL4** 00:00:07.055

TEST **Should Test There Are Tests** 00:04:38.784

| | |
|---|---|
| **Full Name:** | Test Suite.Should Test There Are Tests |
| **Documentation:** | Check for the first test from sel4test |
| **Tags:** | sel4 test uart |
| **Start / End / Elapsed:** | 20191123 05:10:29.186 / 20191123 05:15:07.970 / 00:04:38.784 |
| **Status:** | PASS (critical) |

SETUP Remote.**Reset Emulation** 00:00:00.090

KEYWORD Remote.**Requires** booted-sel4 00:00:01.742

KEYWORD Remote.**Wait For Line On Uart** <testcase classname="sel4test" name="Test that there are tests">, timeout=9999 00:04:36.938

KEYWORD Remote.**Provides** there-are-tests 00:00:00.008

# Building More Complex Systems

- As systems get more complicated, there are way more resources to keep track of
  - This makes it difficult to make sure the system is configured in the correct, secure way
- CAmkES to the rescue
  - Manual: https://github.com/seL4/camkes-tool/blob/master/docs/index.md

# CAmkES: Get The Code

- Similar command as before

```
$ mkdir camkes
$ cd camkes
$ repo init -u https://github.com/dornerworks/riscvsummit-manifests -m camkes.xml
$ repo sync
```

- Start from a known working system
- Note about docker environment (need camkes target)

# CAmkES: Project Layout

```
.
├── easy-settings.cmake -> projects/camkes/easy-settings.cmake
├── griddle -> tools/seL4/cmake-tool/griddle
├── init-build.sh -> tools/seL4/cmake-tool/init-build.sh
├── README.md -> projects/camkes-tool/docs/index.md
├── run-build.sh -> tools/sel4Extra/run-build.sh
├── run-renode.sh -> tools/sel4Extra/run-renode.sh
├── run-robot.sh -> tools/sel4Extra/run-robot.sh
├── start-container.sh -> tools/sel4Extra/start-container.sh
├── kernel
├── projects
│   ├── cakeml_libs
│   ├── camkes
│   ├── camkes-tool
│   ├── capdl
│   ├── global-components
│   ├── musllibc
│   ├── picotcp
│   ├── projects_libs
│   ├── seL4_libs
│   ├── seL4_projects_libs
│   ├── sel4runtime
│   └── util_libs
└── tools
    ├── cogent
    ├── DockerFiles
    ├── pruner
    ├── riscv-pk
    ├── rumprun
    ├── seL4
    └── sel4Extra
```

# CAmkES: Main Application Directory

```
projects/camkes/apps/serialcom$
├── CMakeLists.txt$
├── components$
│   └── SerialApplication$
│       └── src$
│           └── application.c$
├── README$
└── serialcom.camkes$
```

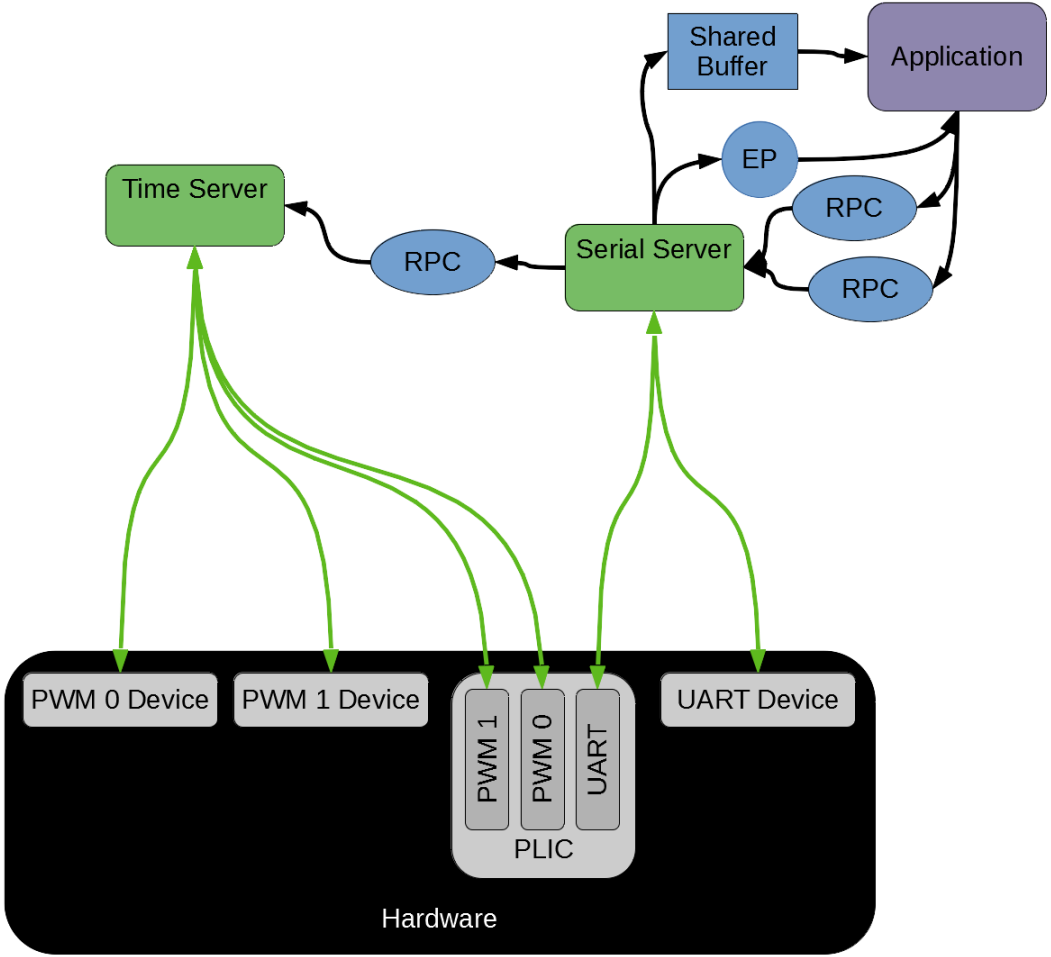
# CAmkES: Serial Application

- Goal of this application:
  - Print and read from the serial device
  - Split up functionality among separated components
- Sounds Trivial
  - Consider taking advantage of the separation enforced by seL4 and the connections available by CAmkES
  - This won’t be in one address space

# CAmkES: Serial Application

- Every CAmkES Application has a Root CAmkES file
  - This defines the architecture of the whole system
    - Instantiates Components and Connections
    - Defines connections
      - RPC interfaces
      - Shared Memory
    - Configures Components

```
assembly {
    composition {
        component SerialApplication serial_app;
        component SerialServer serial;
        component TimeServer time_server;

        connection seL4RPCCall serial_time_server(from time_server.putchar, to serial.processed_putchar);
        connection seL4RPCCall serial_client(from serial_app.putchar, to serial.processed_putchar);

        connection seL4SerialServer serial_input(from serial_app.getchar, to serial.getchar);
        connection seL4TimeServer serialserver_timer(from serial.timeout, to time_server.the_timer);

        connection seL4GlobalAsynchCallback global_callback(from serial_app.self, to serial_app.wait);
    }
    configuration {
        serial_app.putchar_attributes = 0;
        serial_app.getchar_global_endpoint = "serial_app";
        serial_app.getchar_badge = "1";
        serial_app.getchar_attributes = "0";
        serial_app.getchar_shmem_size = 0x1000;

        serial.timeout_attributes = 1;
        time_server.putchar_attributes = 0;

        time_server.timers_per_client = 1;

        serial.wait_global_endpoint = "serial";
        serial.self_global_endpoint = "serial";

        serial_app.wait_global_endpoint = "serial_app";
        serial_app.self_global_endpoint = "serial_app";

    }
}
```


Shared Buffer
Application
EP
RPC
RPC
Time Server
RPC
Serial Server
PWM 0 Device
PWM 1 Device
PWM 1
PWM 0
UART
PLIC
UART Device
Hardware

# CAmkES: Serial Application

```
component SerialApplication {
    control;
    attribute int simple = true;
    uses PutChar putchar;
    uses GetChar getchar;
    emits Callback self;
    consumes Callback wait;
}
```

# CAmkES: Serial Server (Snippet)

```
component SerialServer {
    control;
    has mutex serial;
    provides PutChar processed_putchar;
    provides PutChar raw_putchar;
    provides GetChar getchar;
    uses Timer timeout;

    /* Virtqueue connection to faciliate read and write */
    maybe uses VirtQueueDev read;
    maybe uses VirtQueueDev write;
    //emits Callback self;
    emits Callback self_read;
    emits Callback self_write;

    //maybe consumes Callback serial_wait;
    maybe consumes Callback serial_write_wait;
    maybe consumes Callback serial_read_wait;
```

# CAmkES: RPC Call

- Must have an interface defined (procedure keyword)
  - At least one component “uses” the interface
  - One component “provides” the interface

# CAmkES: RPC Calls (Procedure)

```
procedure PutChar {
    void putchar(in int c);
};
```

RPC Function Name

# CAmkES: RPC Calls (Provider)

Procedure Interface Instance Name

RPC Function Name

```
void processed_putchar_putchar(int c)
{
    seL4_Word n = processed_putchar_get_sender_id();
    internal_putchar((int)n, c);
    if (c == '\n') {
        internal_putchar(n, '\r');
    }
}
```

# CAmkES: RPC Calls (User)

```
void print_string(char* string, uint32_t len)
{
    for (uint32_t i=0; i<len; i++)
    {
        putchar_putchar(string[i]);
    }
}
```

**Interface Instance Name**

**RPC Function Name**

# CAmKES: Why?

- But I don’t trust generated code!
- There is too much indirection!
- Capability Calculation
  - Serial application generated roughly 1000 capabilities
    - Look for the *.cdl file in your build directory

# CAmkES: Configure The Build

- Provided “run-build.sh” calls init-build.sh with the following CMake settings:
  - -DPLATFORM=hifive
  - -DSIMULATION=FALSE
  - -DElfloaderImage=elf
  - -DLibSel4PlatSupportUseDebugPutChar=OFF
  - -DCMAKE_BUILD_TYPE=Debug
  - -DRELEASE=False
  - -DCAMKES_APP=serialcom
  - -DCAmkESDTS=ON
  - -DRISCV64=TRUE

# CAmkES: Run The Build

- Same process as before
- Run “run-build.sh” script or “init-build.sh” with your flags

./start-container.sh

./start-container.sh | jesse@dwbuntu: ~/DornerWorksProjects/RISC-V_Summit_2019/C.

```
Step 6/7 : VOLUME /home/${UNAME}
 ---> Using cache
 ---> 97f4e25ddbb0
Step 7/7 : VOLUME /isabelle
 ---> Using cache
 ---> f9dc9fb90881
Successfully built f9dc9fb90881
Successfully tagged user_img--riscv:latest
docker run \
        -it \
        --hostname in-container \
        --rm \
        -u 1000:1000 \
        -v /home/jesse/DornerWorksProjects/RISC-V_Summit_2019/Code/Repo-camkes:/host \
        -v jesse-home:/home/jesse \
        -v /etc/localtime:/etc/localtime:ro \
        user_img--riscv bash
___
 |  _      _ |_      _   _ |_ |_
 |  |  |_| _) |_ \)/ (_) |  |_ | ) \/
                                  /

 __
(_      _ |_  _  _  _
__) \/ _) |_ (- ||| _)
    /
Hello, welcome to the sel4/CAmkES/L4v docker build environment
jesse@in-container:/host$
```

# CAmkES: Emulate It

```
:name seL4 Test
:description Runs seL4 Test on Renode

$name?="sel4test"

using sysbus
mach create $name
$bin?=@./riscv-build-hifive-64/images/capdl-loader-image-riscv-hifive
$dtb?=@./tools/sel4Extra/renode_support/hifive-renode.dtb
machine LoadPlatformDescription @./tools/sel4Extra/renode_support/sifive-fu540-dw.repl
showAnalyzer uart0

macro reset
"""
   sysbus LoadELF $bin
   sysbus LoadFdt $dtb 0x81000000 "earlyconsole mem=256M@0x80000000"
   e51 SetRegisterUnsafe 11 0x81000000
   u54_1 SetRegisterUnsafe 11 0x81000000
"""
runMacro $reset
```

jesse@dwbuntu: ~/DornerWorksProjects/RISC-V_Summit_2019/Code/Repo-camkes

./start-container.sh | jesse@dwbuntu: ~/DornerWorksProjects/RISC-V_Summit_2019/C.

```
16:35:00.1250 [INFO] sel4test: Machine paused.data from empty receive fifo (7)-V_Summit_2019/Code/Re
16:35:00.1321 [DEBUG] sel4test: Disposing sysbus.e51.@ [Hart #1 at Supervisor level (connection outp
16:35:00.1358 [DEBUG] sel4test: Disposing sysbus.u54_1.t 0x1000. (20) 0x80000000.
16:35:00.1373 [DEBUG] sel4test: Disposing sysbus.u54_2. [Hart #1 at Supervisor level (connection out
16:35:00.1385 [DEBUG] sel4test: Disposing sysbus.u54_3.t 0x1000. (18) 0x80007000.
16:35:00.1398 [DEBUG] sel4test: Disposing sysbus.u54_4. [Hart #1 at Supervisor level (connection out
16:35:00.1412 [DEBUG] sel4test: Disposing sysbus. offset 0x1000. (16)at 0x80200000.
16:35:00.1418 [DEBUG] sel4test: Disposing sysbus.debug. empty receive fifo
16:35:00.1420 [DEBUG] sel4test: Disposing sysbus.e51DTim.0x1000. (14)
16:35:00.1421 [DEBUG] sel4test: Disposing sysbus.u54Hart1ITim.0. (13)
16:35:00.1421 [DEBUG] sel4test: Disposing sysbus.u54Hart2ITim.0. (12)
16:35:00.1421 [DEBUG] sel4test: Disposing sysbus.u54Hart3ITim.0. (11)
16:35:00.1421 [DEBUG] sel4test: Disposing sysbus.u54Hart4ITim.0. (10)
16:35:00.1421 [DEBUG] sel4test: Disposing sysbus.qspi0Flash.000. (9)ole mem=256M@0x80000000'.
16:35:00.1421 [DEBUG] sel4test: Disposing sysbus.ddr.set 0x1000. (8)
16:35:00.1728 [DEBUG] sel4test: Disposing sysbus.qspi0.t 0x1000. (7)lue 0x1.
16:35:00.1737 [DEBUG] sel4test: Disposing sysbus.qspi1.t 0x1000. (6)lue 0x1.
16:35:00.1738 [DEBUG] sel4test: Disposing sysbus.qspi2.t 0x1000. (5)lue 0x0.
16:35:00.1738 [DEBUG] sel4test: Disposing sysbus.i2c.set 0x1000. (4)lue 0x1.
16:35:00.1745 [INFO] sel4test: Disposed.read from offset 0x1000. (3)lue 0x0.

jesse@dwbuntu ~/DornerWorksProjects/RISC-V_Summit_2019/Code/Repo-camkes
  %                                                                          !6432
```

# Using seL4 in your projects

- Start from a manifest to get the build process correct
- Set up a testing environment
  - There are a lot of moving parts, it's important to know when things go wrong
- Explore Userspace architecture systems:
  - Cogent
  - Rump Kernels
  - CAmkES
- Join the community
  - Mailing List: https://sel4.systems/lists/listinfo/
  - Github: https://github.com/seL4 and https://github.com/SEL4PROJ
  - Discourse: https://sel4.discourse.group/
- Checkout other Data61 Content
  - Tutorials: https://docs.sel4.systems/Tutorials/
  - FAQ: https://docs.sel4.systems/FrequentlyAskedQuestions.html
- DornerWorks can help with system design and board support
  - https://dornerworks.com/contact-us

# Questions?

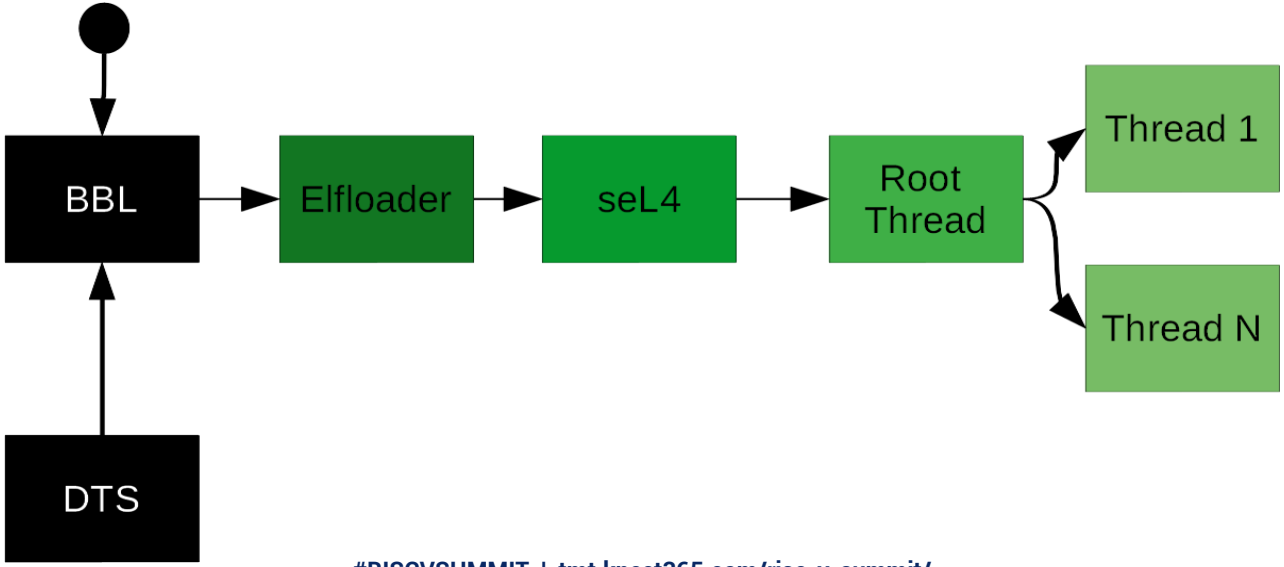
# Extra Slides

# seL4 Test: Root Thread

- Once seL4 has booted, it hands off the following to the “Root Thread”
  - All of the remaining “Un-typed” memory
  - Boot Info Structure
- Remember, the proofs don’t apply to code here
  - It is up to the programmer to make sure you don’t have overflows or any of the other runtime bugs not present in the kernel
  - There are tools to help here, such as Data61’s Cogent
- This thread has the ability to “Re-Type” this memory into different kernel objects
- It can create and delegate capabilities to the rest of the system
- In the seL4 test project, the root thread is responsible for:
  - Creating Virtual memory spaces
  - Creating Memory allocators
  - Creating Capability Spaces
  - Initialized needed hardware such as timers and interrupts
  - Creates a larger stack
  - Creates threads and hands out capabilities to tests

# seL4 Test: Boot Process

- Berkley Boot Loader (riscv-pk project)
- Elfloader
- seL4 kernel
- Userspace Root Thread



# seL4 Test: ELF Loader

- Build process archives the userspace image and the kernel in the same CPIO Archive
- ELF Loader extracts the images and places them at the proper spots