RISC-V VP++: Fast Open-Source Virtual Platforms - Powering Research Innovation and Industrial Use

Daniel Große
Institute for Complex Systems (ICS)
Web: jku.at/ics
Email: daniel.grosse@jku.at

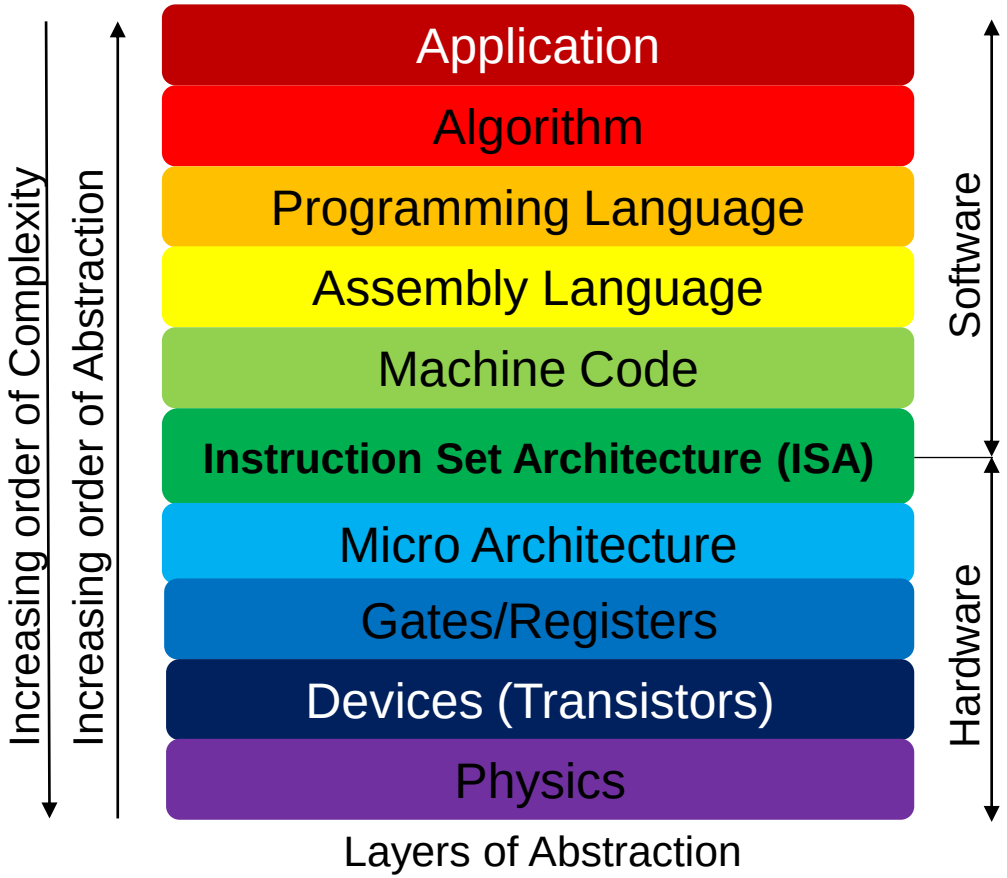
JOHANNES KEPLER UNIVERSITY LINZ

Outline

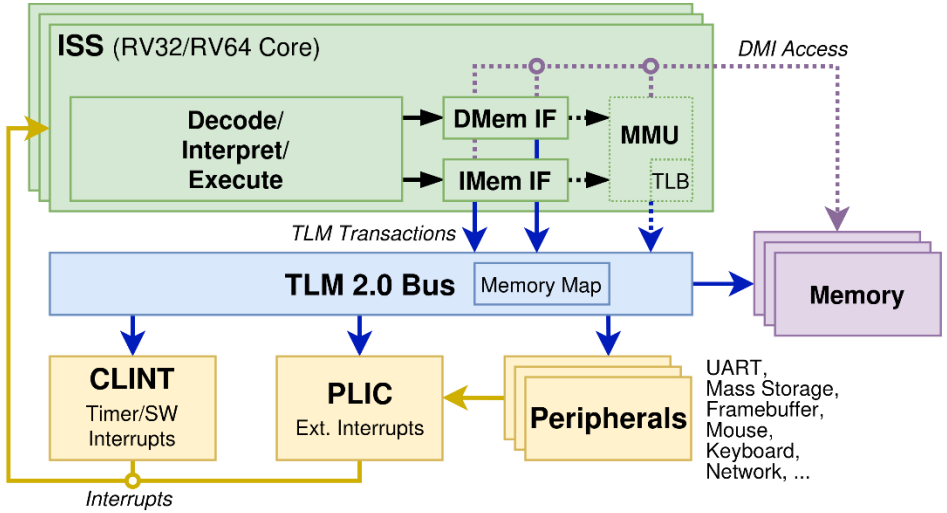
- RISC-V
- RISC-V VP++ Overview & Verification Approaches
  - Bare Metal Example & Metamorphic Testing
  - Linux: GUI-VP Kit / 3D-Games & FastISS
  - Vector Extension & RVVTS
- Conclusions

RISC-V

- Open and royalty-free ISA
- Focus on simplicity and modularity
- Base Integer Instruction Set
  - Mandatory
  - 32, 64 and 128 bit configurations
  - ~40 Instructions
- Extensions:
  - M .. Multiply/Divide
  - A .. Atomic
  - F, D, Q .. Floating Point (Single, Double, Quad)
  - C .. Compressed
  - …



RISC-V VP++ Overview



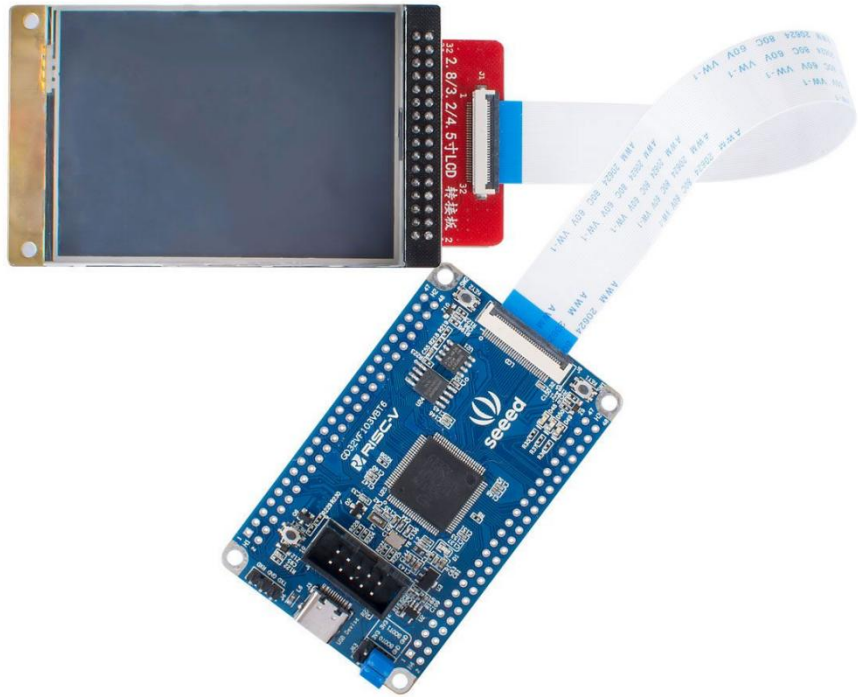
- Open source on GitHub
  - https://github.com/ics-jku/riscv-vp-plusplus
- Key features:
  - SystemC/C++ TLM-2.0
  - 32 & 64 bit cores (RV32GCV + SUN, RV64GCV + SUN)
  - Bare metal/Small operating systems configurations, including:
    - SiFive HiFive1 - FE310
    - GD32VF103VBT6 microcontroller (Nuclei N205) including UI
  - Linux RV32 and RV64, single and quad-core VPs (SiFive FE540)
  - Full integration of GUI-VP, which enables simulation of interactive graphical Linux applications
  - Support for *RISC-V "V" Vector Extension* (RVV) version 1.0
  - Support for *Capability Hardware Enhanced RISC Instructions* (CHERI)
  - Based on RISC-V VP introduced in 2018

Bare Metal Example

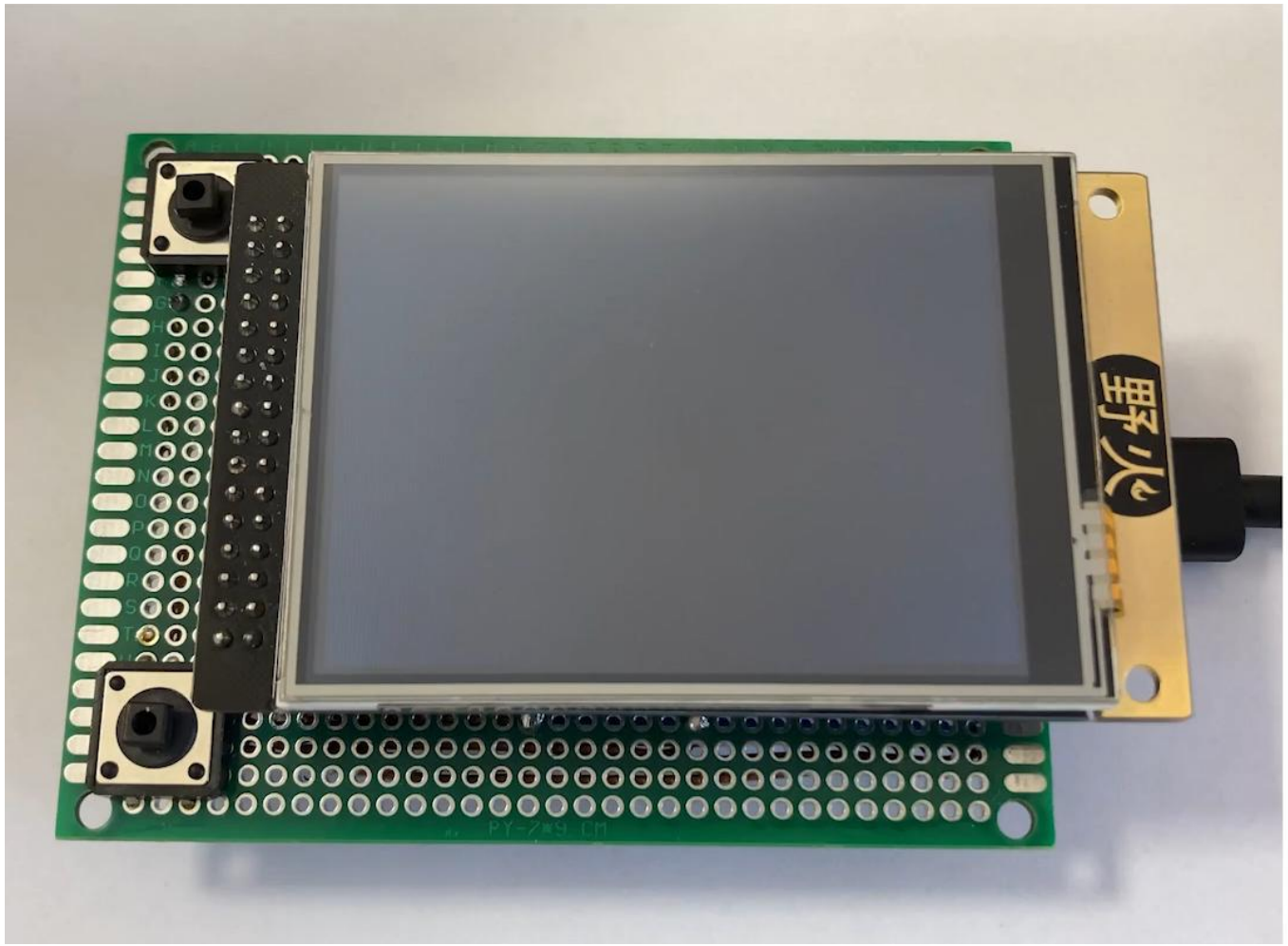
RISC-V VP++: GD32V

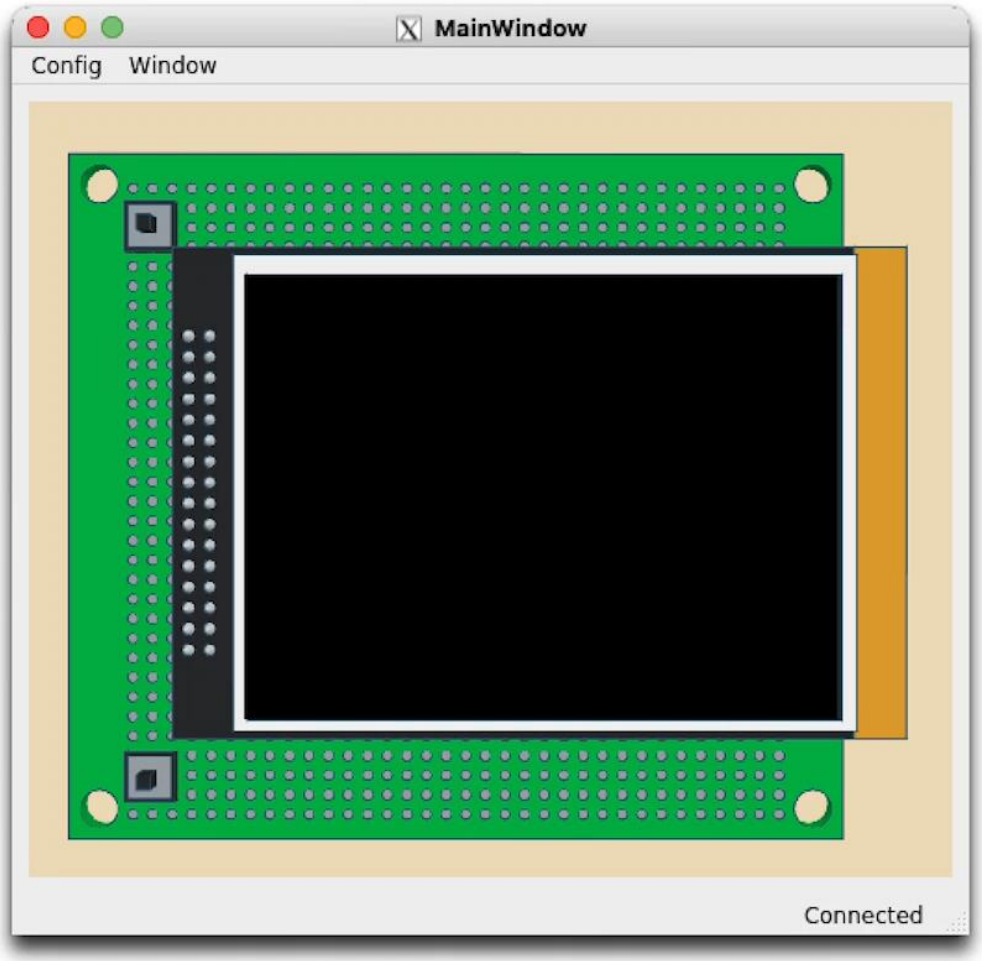
- GD32VF103VBT6 microcontroller (Nuclei N205)

- Implemented peripherals in RISC-V VP++
  - GPIO, AFIO, EXTI, SPI, EXMC, RCU
- UI
  - ILI9341 display w XPT2046 touch controller
  - TFT_eSPI
    - Widley used embedded graphics library
    - Adapted for RISC-V



RISC-V VP++: GD32V Demo





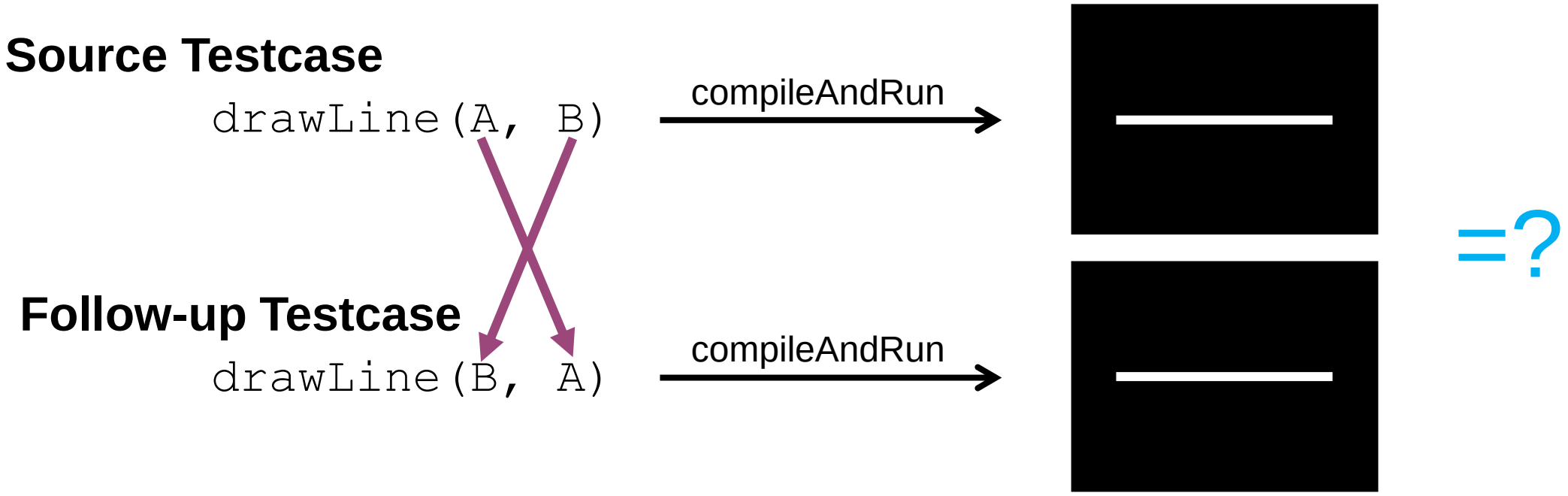
RISC-V VP++: TFT_eSPI Verification Challenge

- **Current verification:**
  Use TFT_eSPI and visually check the result

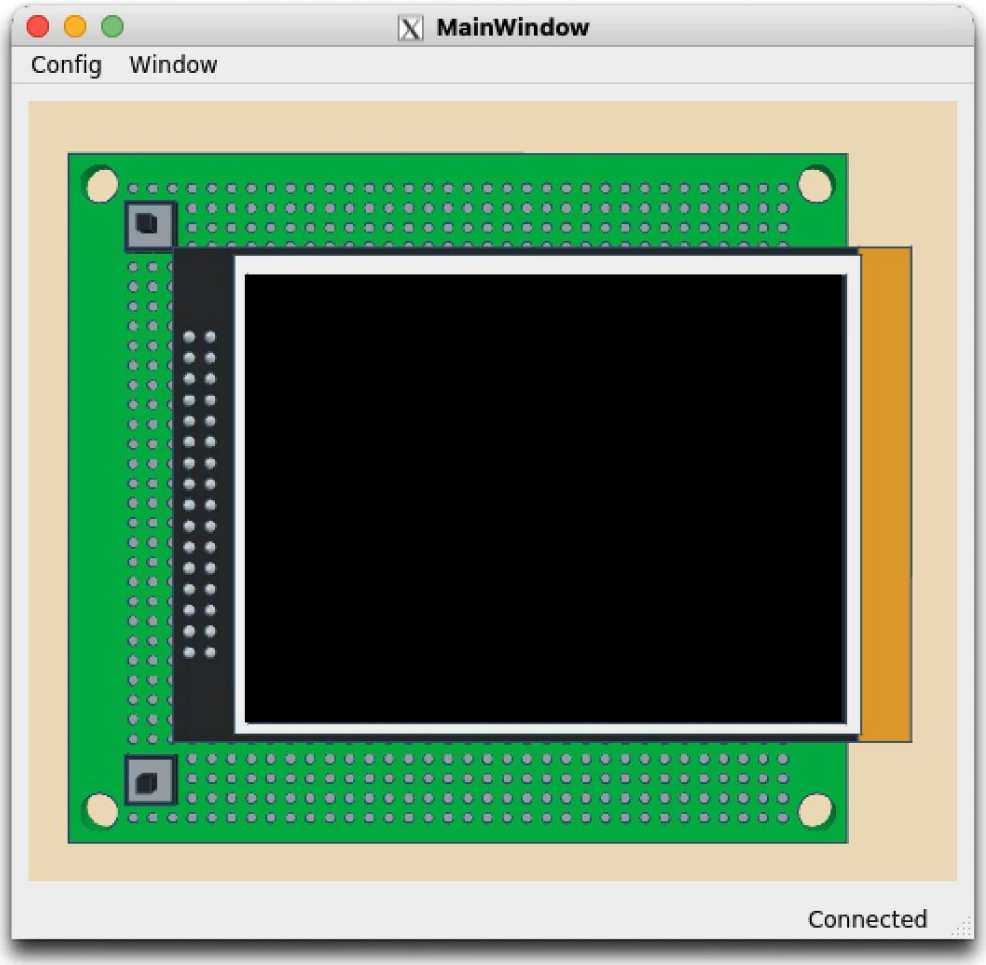
- **Our approach:**
  Overcome need of physical HW via *Metamorphic Testing*
  - Relates multiple program executions via Metamorphic Relations (MRs)
  - If relation is violated, bug is found

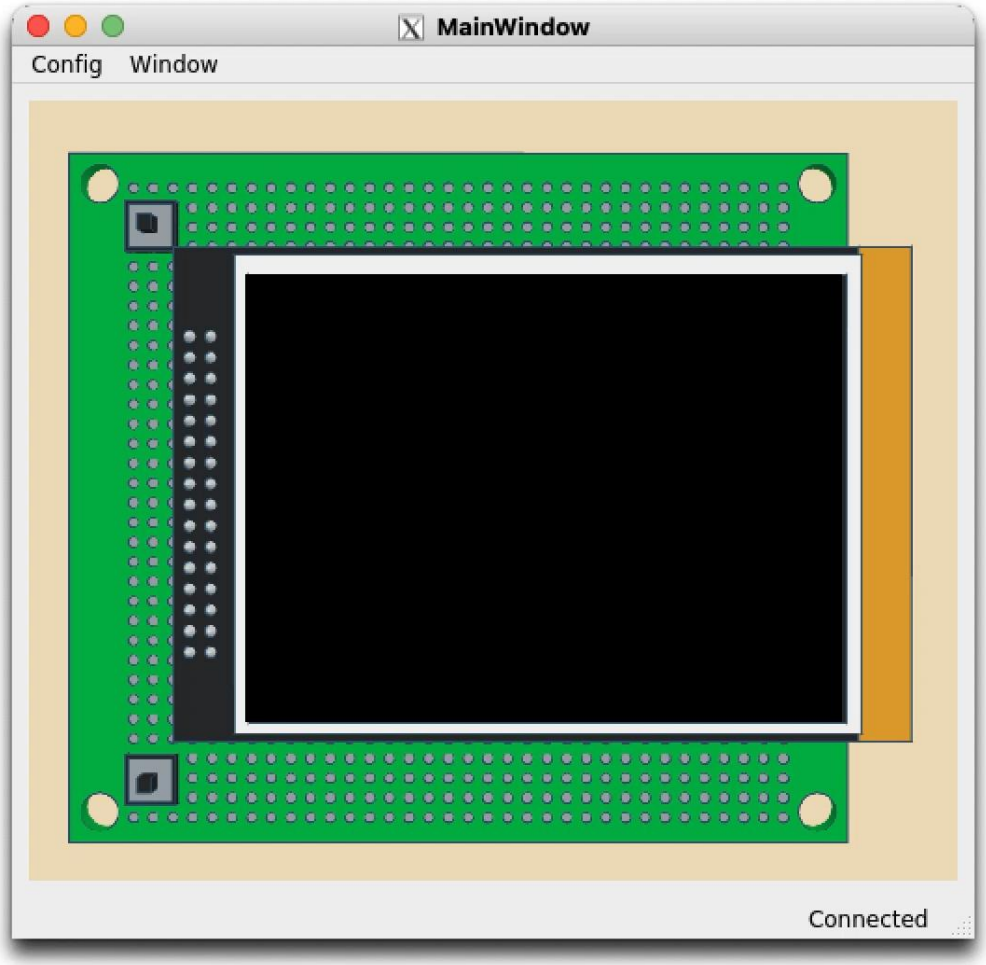
Metamorphic Testing for TFT_eSPI leveraging VPs

Simple **Metamorphic Relation (MR)** example to draw a line on a display



Metamorphic Testing Demo





Approach & Results

- Effective Firmware Testing Approach
  - No need for physical Hardware
  - No need for an Oracle
  - Highly automated

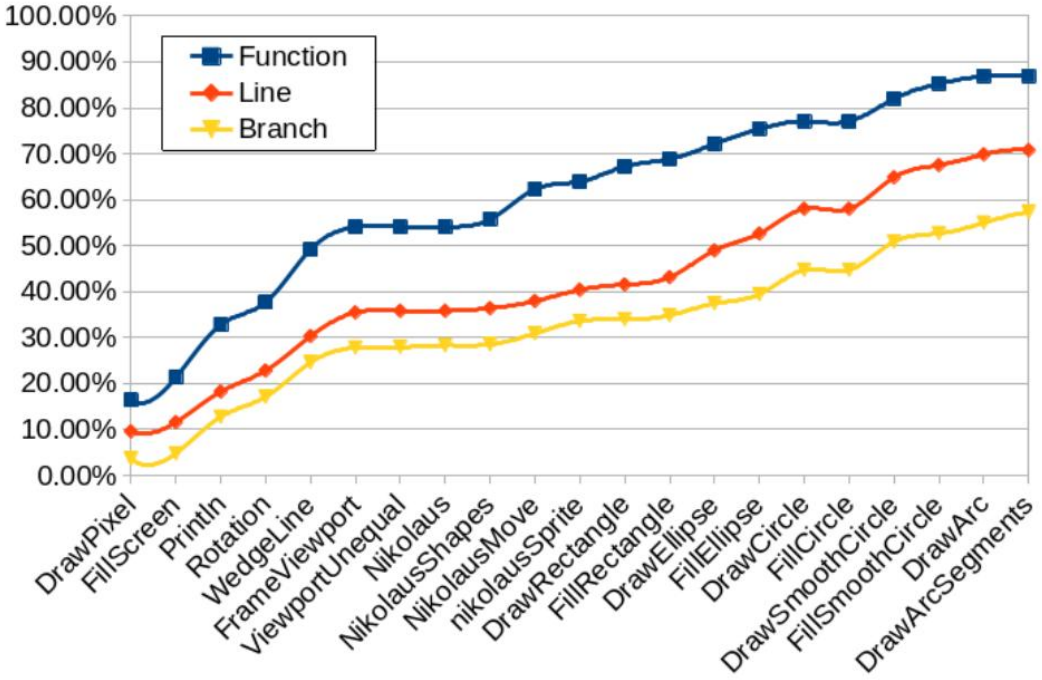
- Exposed 15 unknown bugs in TFT_eSPI library

Number of MRs	21
∅ Testcases per MR	4300
Failed MRs	11
Total Bugs	15

Christoph Hazott, Florian Stögmüller, and Daniel Große. Verifying embedded graphics libraries leveraging virtual prototypes and metamorphic testing. In *ASP-DAC*, 2024.
https://ics.jku.at/files/2024ASPDAC_Verifying_Embedded_Graphics_Libraries_leveraging_VPs_and_MT.pdf

Extension: LLMs for MR generation

- Further improved coverage high coverage



	Manual	LLM
Number of MRs	21	160
Total Bugs	15	26

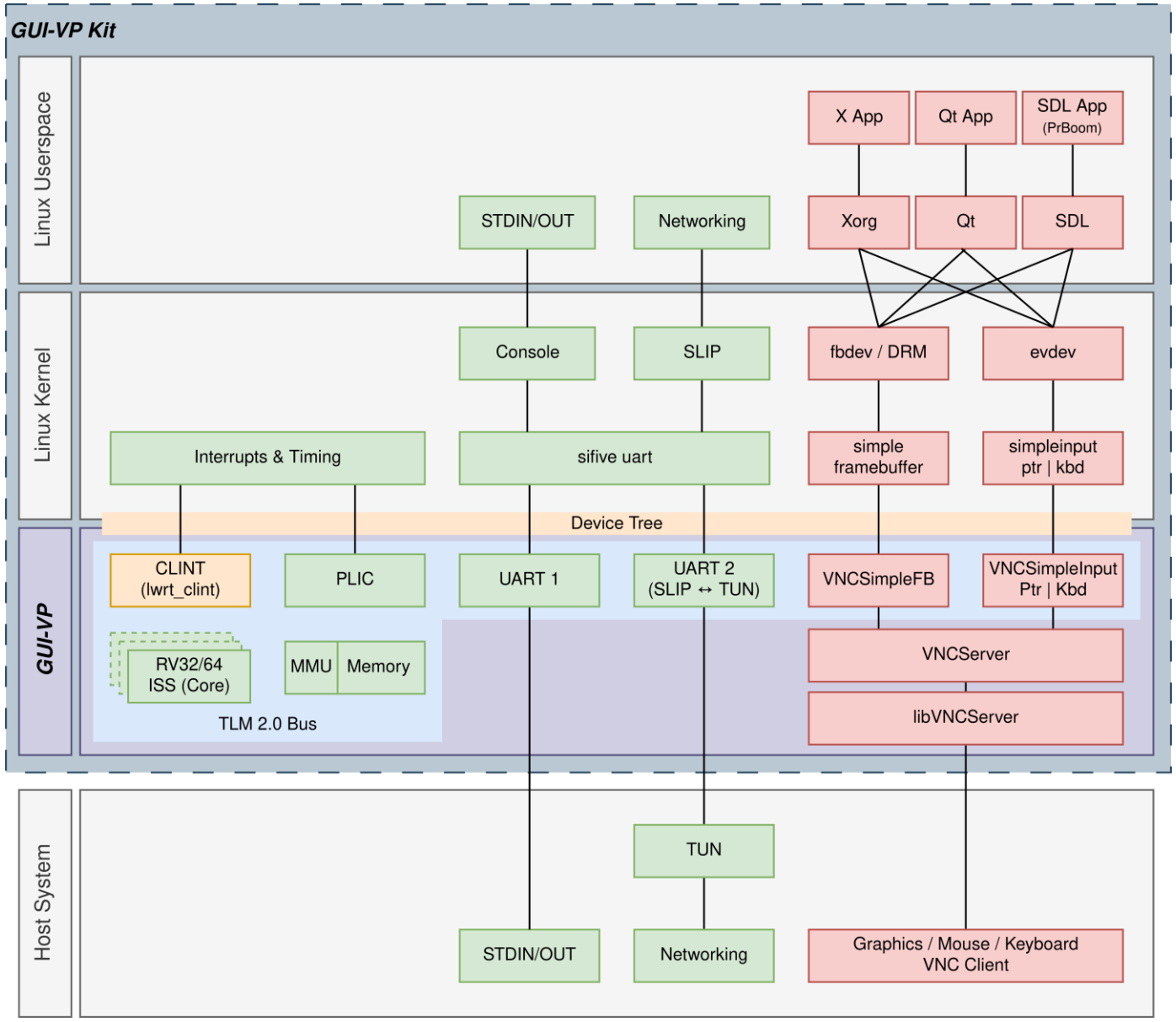
C. Hazott and D. Große, LLM-assisted Metamorphic Testing of Embedded Graphics Libraries, in *FDL*, 2025
https://ics.jku.at/files/2025FDL_LLM-assisted_Metamorphic_Testing_of_Embedded_Graphics_Libraries.pdf

Linux & GUI-VP Kit

GUI-VP Kit

Main Parts:

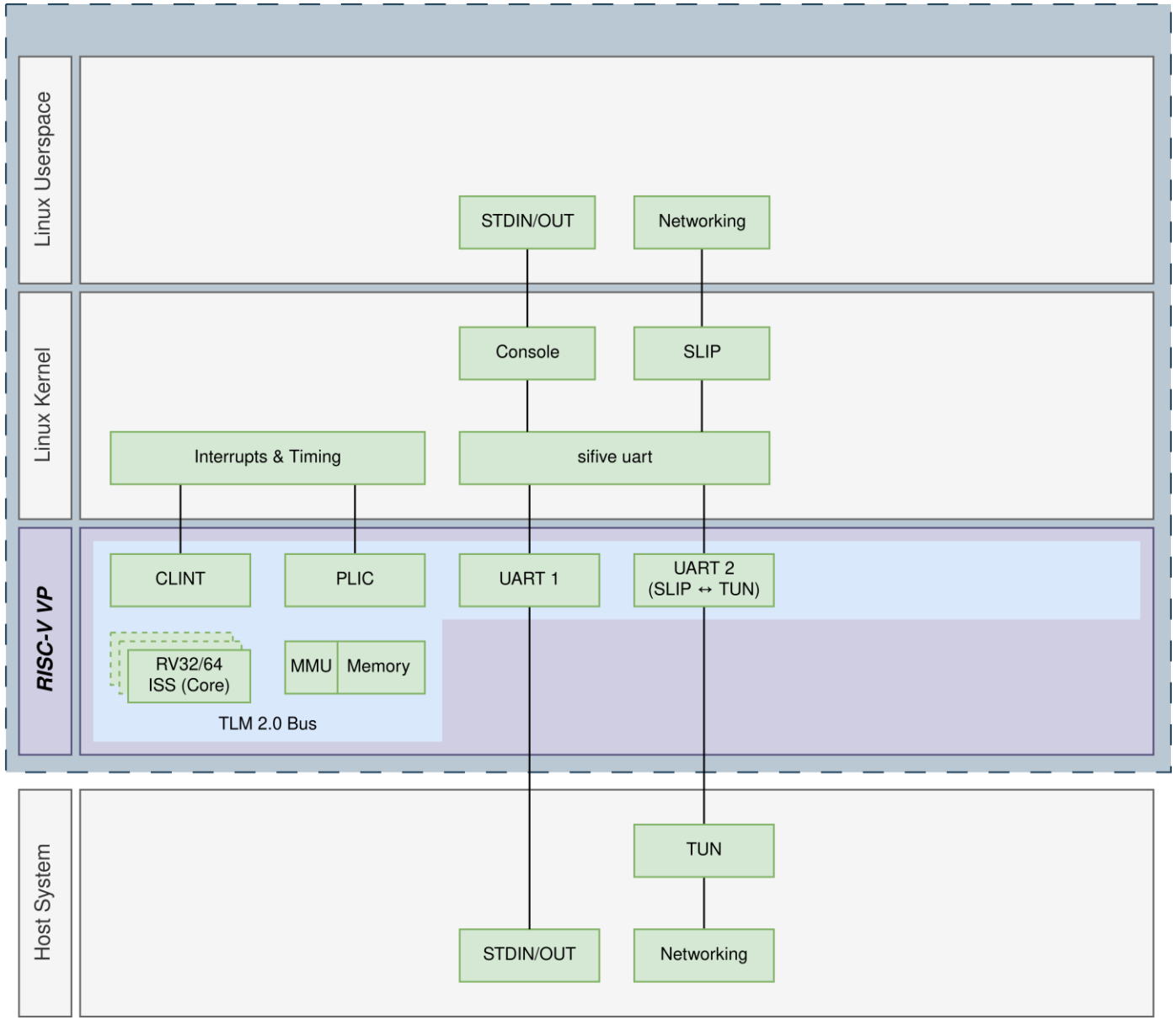
- **GUI-VP**
  - RISC-V VP
  - Real-Time behavior
  - Graphics output
  - Mouse/Keyboard input
  - VNC server
- **Linux Buildsystem**
  - Configurations
  - Linux drivers
  - Devicetree



Manfred Schlägl and Daniel Große. GUI-VP Kit: A RISC-V VP meets Linux graphics - enabling interactive graphical application development. In *GLSVLSI*, 2023. https://ics.jku.at/files/2023GLSVLSI_GUI-VP_Kit.pdf

GUI-VP Kit Simulation Stack

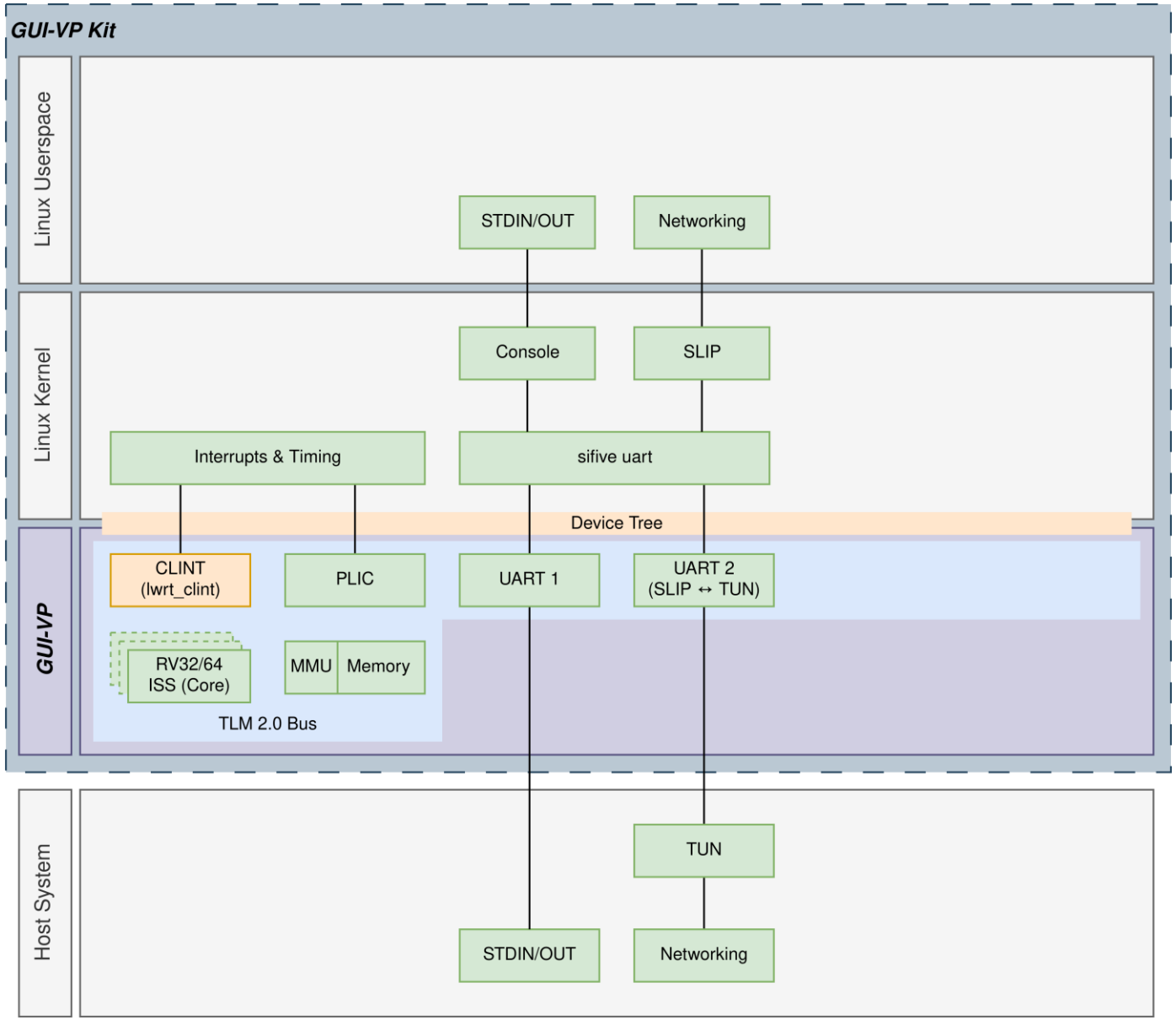
- Starting point



GUI-VP Kit Simulation Stack

- Starting point

- Modifications / Replacements
  - Linux Bring-Up ➞ Device Tree
  - Real-Time Behavior ➞ *lwrt_clint*

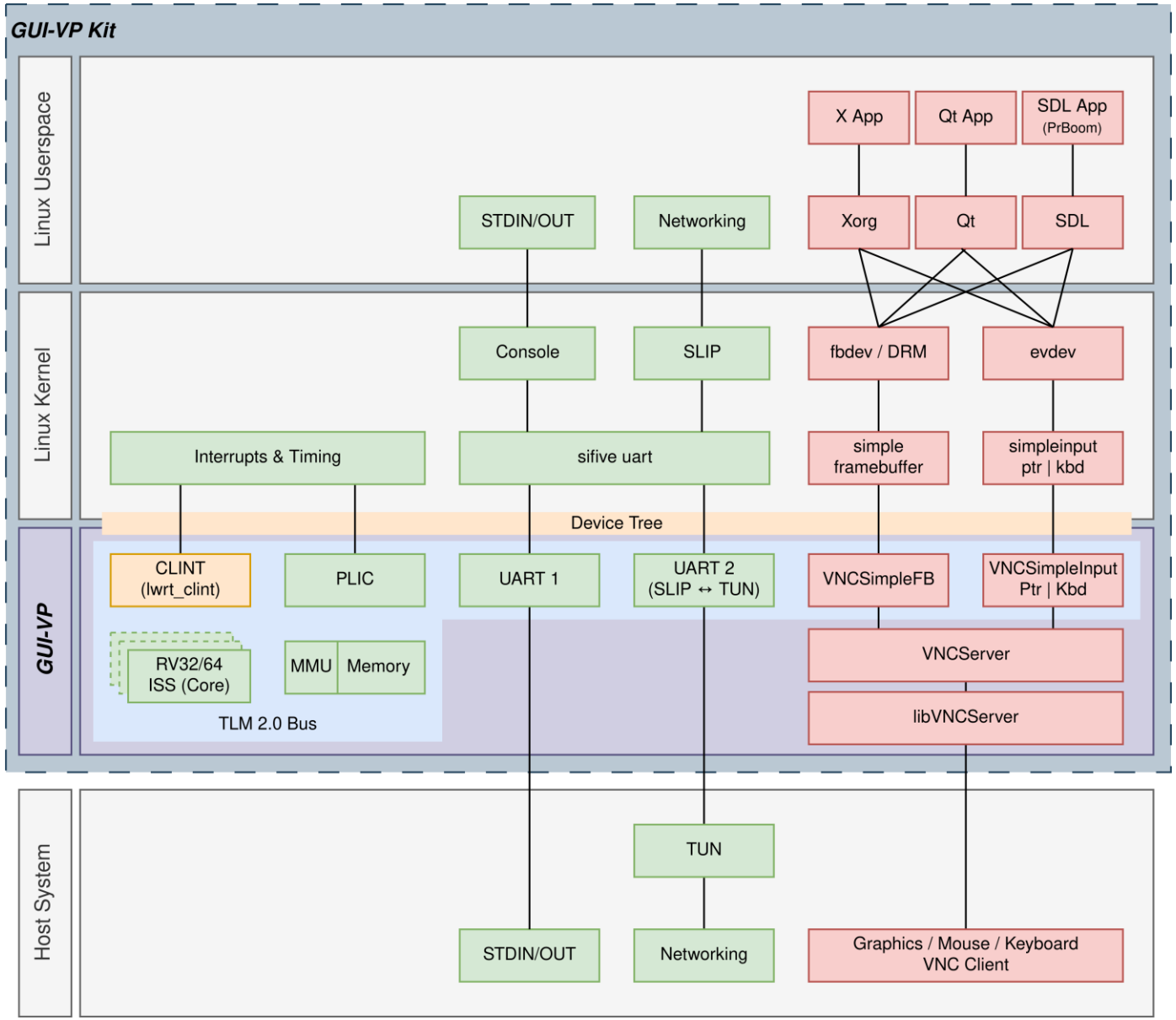


GUI-VP Kit Simulation Stack

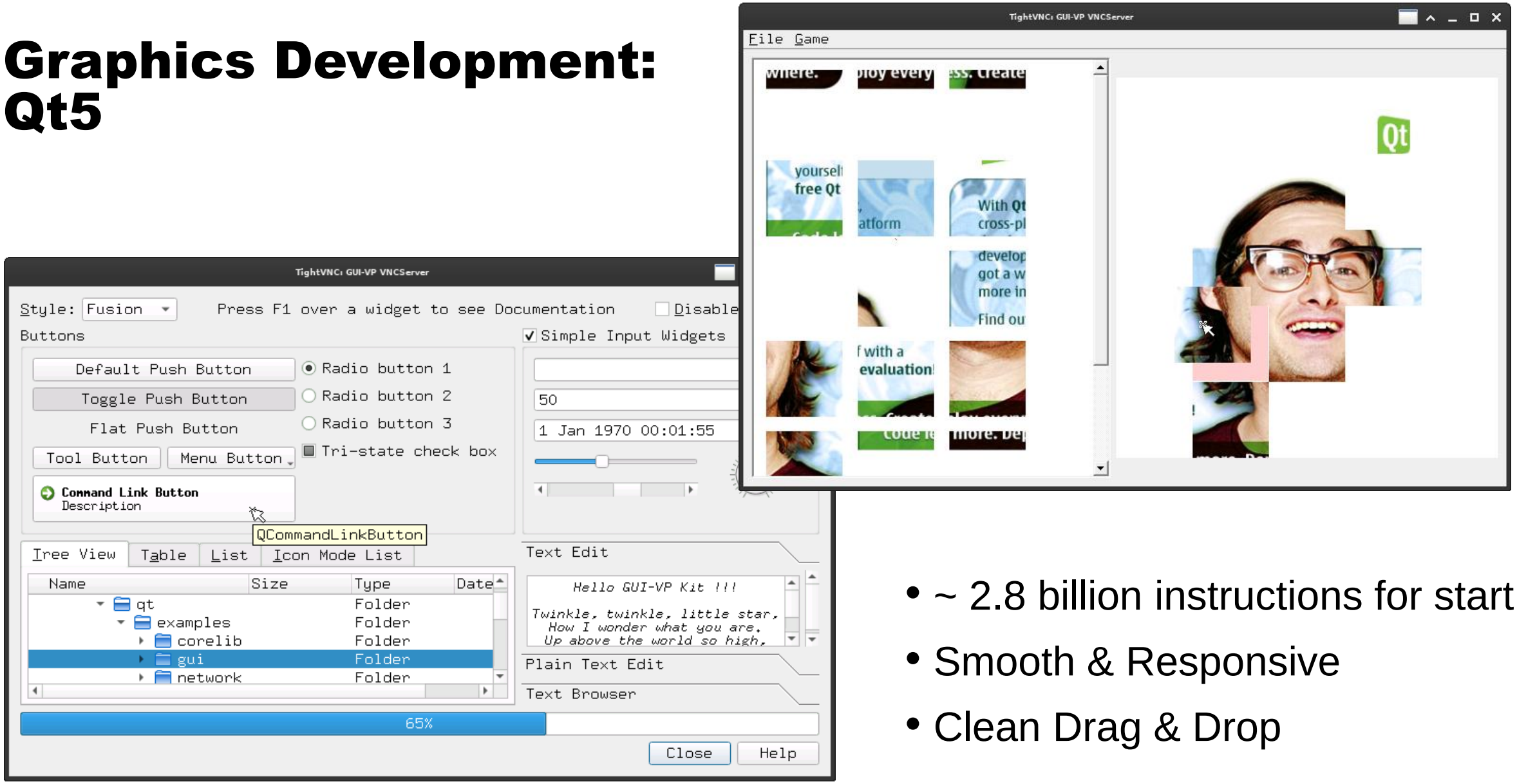
- Starting point

- Modifications / Replacements
  - Linux Bring-Up ➞ Device Tree
  - Real-Time Behavior ➞ *lwrt_clint*

- Extensions
  - VNC Server
  - Graphics Output
  - Mouse Input (Ptr)
  - Keyboard Input (Kbd)

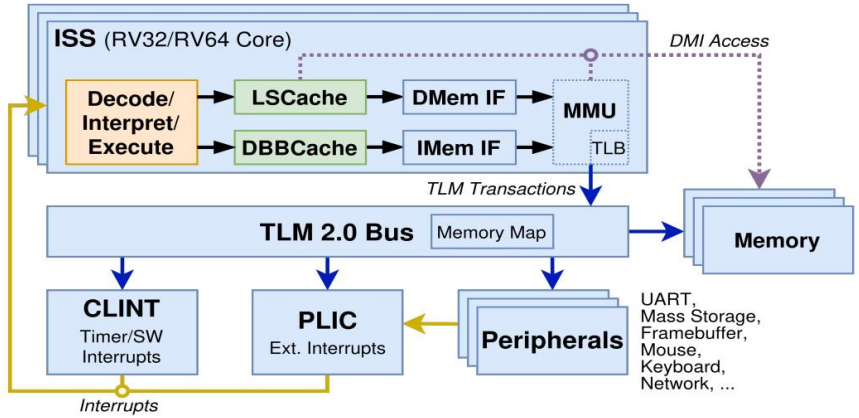


Graphics Development: Qt5



- ~ 2.8 billion instructions for start
- Smooth & Responsive
- Clean Drag & Drop

Fast Interpreter-Based ISS [1]



Goal: Improve performance while preserving comprehensibility and adaptability of an interpreter ISS

- **Dynamic Basic Block Cache (DBBCache)**
  - Speed up execution
  - Extracts Dynamic Basic Blocks (DBBs) at runtime
  - Caches data for ISS instruction processing
- **Load/Store Cache (LSCache)**
  - Speed up memory access
  - Direct translation of in-simulation virtual addresses to host system memory addresses

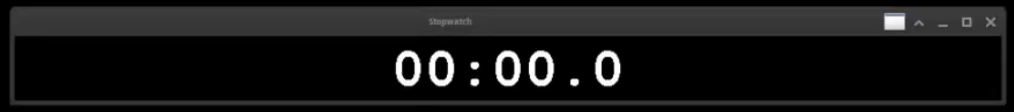
→ up to 406 MIPS

→ avg. 8.98x performance improvement over the unoptimized RISC-V VP++

→ avg. 1.65x performance improvement over the Spike Simulator

[1] Manfred Schlägl and Daniel Große. Fast Interpreter-Based Instruction Set Simulation for Virtual Prototypes. In *DATE,* 2025 (https://ics.jku.at/files/2025DATE_Fast_Interpreter-based_ISS.pdf)

Linux Boot


Stopwatch
00:00.0


Terminal - ame@pulsar: /srv/ext/VP/GUI-VP_Kit
ame@pulsar:/srv/ext/VP/GUI-VP_Kit$ make run_rv64_sc

Linux PrBoom Demo Run


```
 found /usr/share/games/doom/doom1.wad
IWAD found: /usr/share/games/doom/doom1.wad
PrBoom (built Feb  3 2025), playing: DOOM Shareware
PrBoom is released under the GNU General Public license v2.0.
You are welcome to redistribute it under certain conditions.
It comes with ABSOLUTELY NO WARRANTY. See the file COPYING for details.
V_Init: allocate screens.
 found /usr/share/games/doom/prboom.wad
D_InitNetGame: Checking for network game.
W_Init: Init WADfiles.
 adding /usr/share/games/doom/doom1.wad
 adding /usr/share/games/doom/prboom.wad
W_InitCache

M_Init: Init miscellaneous info.
R_Init: Init DOOM refresh daemon -
R_LoadTrigTables: Endianness...ok.
R_InitData: Textures Flats Sprites Tranmap build [........]
R_Init: R_InitPlanes R_InitLightTables R_InitSkyMap R_InitTranslationsTables R_InitPatches
P_Init: Init Playloop state.
I_Init: Setting up machine state.
I_InitSound: couldn't open audio with desired format
S_Init: Setting up sound.
S_Init: default sfx volume 8
HU_Init: Setting up heads up display.
I_InitGraphics: 640x480
I_UpdateVideoMode: 640x480 (fullscreen)
V_InitMode: using 8 bit video mode
I_SetRes: Using resolution 640x480
I_UpdateVideoMode: 0xe0000000, SDL buffer, direct access
ST_Init: Init status bar.
```

~12 FPS @ 640x480
~12 Mio per Frame
~144 MIPS
(each pixel is a TLM transaction)

Vector Extension (RVV)

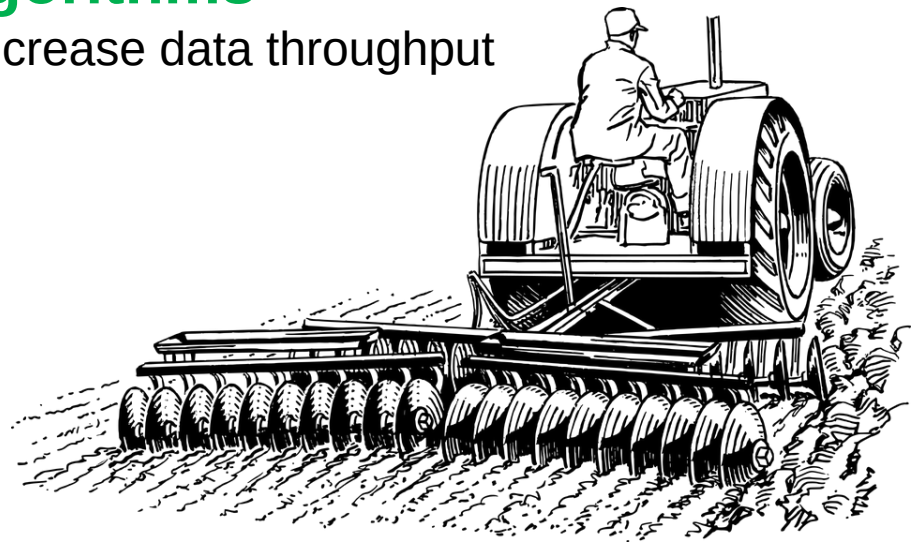
RVV Motivation

Acceleration of signal processing, multimedia, machine learning, …

→ Exploiting data-level parallelism of algorithms

Working on multiple elements of data simultaneously to increase data throughput

→



SISD = Single Instruction, Single Data

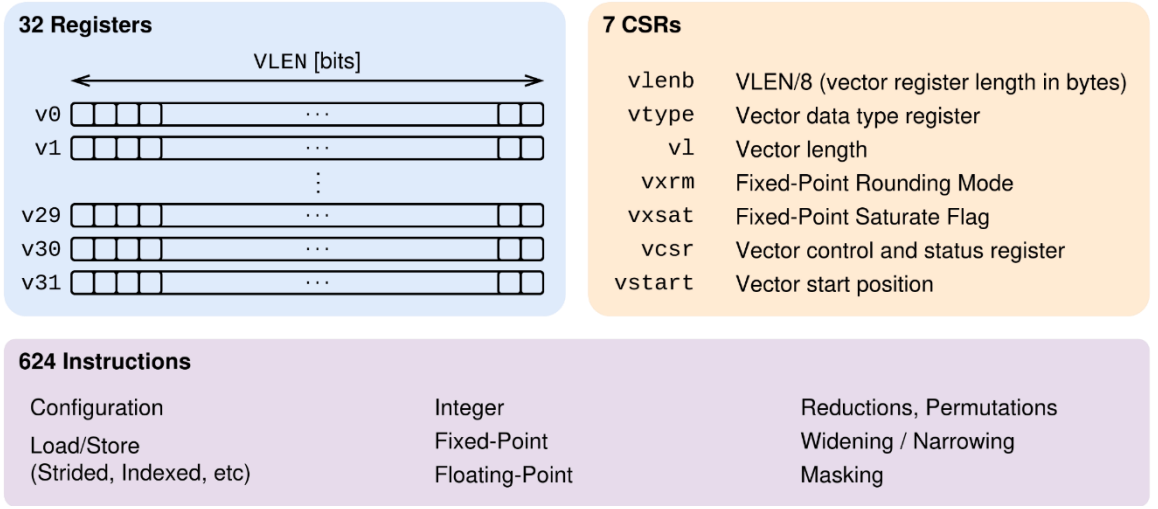
- Execute a single instruction to operate on a single data element
- Classic von-Neuman architecture

SIMD = Single Instruction, Multiple Data

- Execute a single instruction to operate on **multiple** data elements **simultaneously**
- **Classic SIMD and Vector architectures**

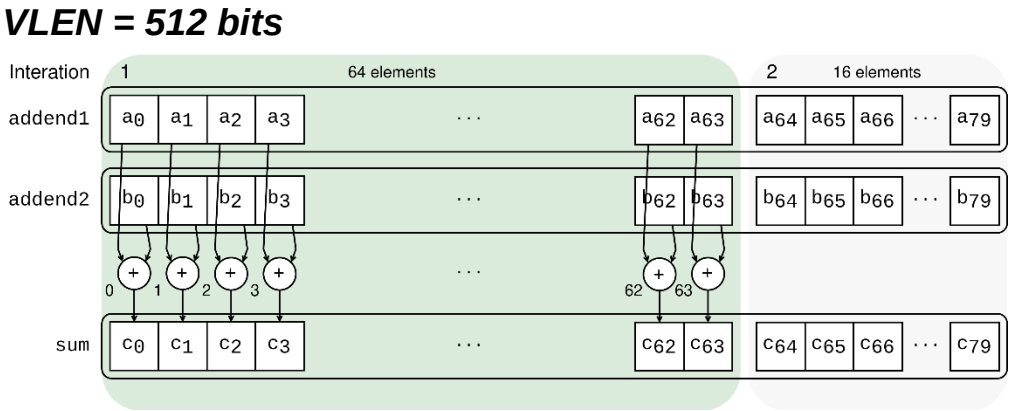
→ RISC-V Vector Extension (RVV)

RISC-V Vector (RVV)



- Large number of instructions
- Complex instruction behavior
  (e.g. *strided load/store*)
- Instruction behavior highly dependent on **dynamic configuration**
  - Element type, vector length, register grouping, …

High dimensional parameter space!
→ Verification and Testing is hard!



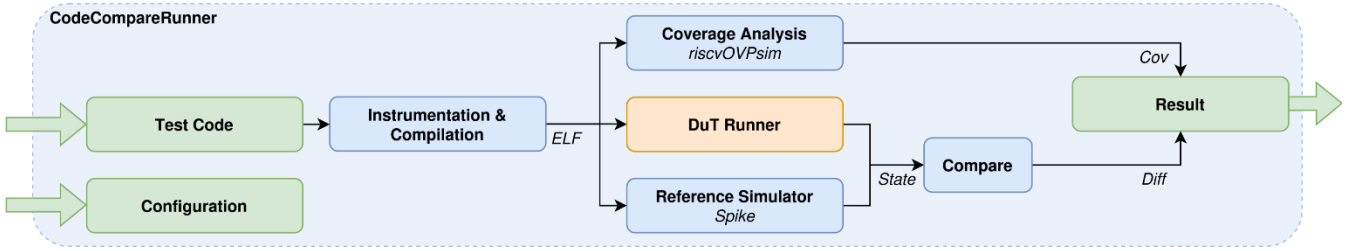
Verification of RISC-V Vector
RVVTS Framework

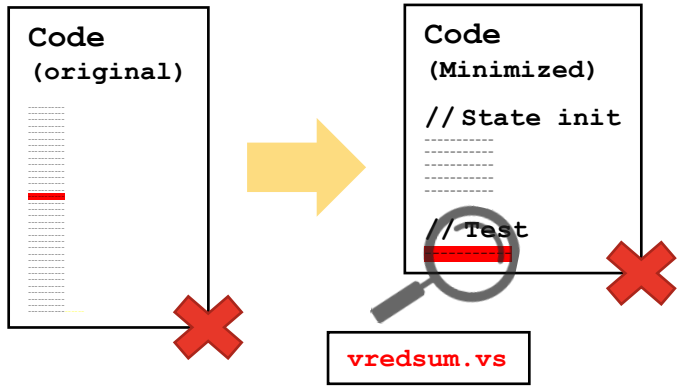
Modular, automated, open-source framework for positive and negative testing of RVV

- **Grammar-based, coverage-guided *Instruction Sequence Generator* (ISG)**
- Code instrumentation and build

High test throughput → **signature NOT after each instruction**

- Execution on reference simulator and *Device under Test* (DUT), and extraction of architectural states + Measurement of functional coverage
  → Architectural state differences → Potential fails
- **Single Instruction Isolation:** Minimization of failing test cases and isolation of failing instructions





Verification of RISC-V Vector
RVVTS Framework

Pre-generated, ready-to-use test sets

- RV32/RV64 RVV with 512 bit vector length
- ***Pure positive*** (valid sequences only) and ***positive/negative*** testing (valid&invalid sequences)
- Each set >1.25 million RVV instructions with **>94.94%** functional coverage

RISC-V VP++ Virtual Prototype

~1.8k failures → 10 isolated instructions

→ **3 Bugs confirmed**

QEMU emulator

~19k failures → 168 isolated instructions

→ **2 Bugs confirmed** (7 more to analyze)

→ **Adaptable to other Extensions**

→ **Adaptable to Simulators, Emulators and real HW**

→ **https://github.com/ics-jku/RVVTS**

JOHANNES KEPLER UNIVERSITY LINZ

M. Schlägl and D. Große. Single instruction isolation for RISC-V vector test failures. In *ICCAD*, 2024
https://ics.jku.at/files/2024ICCAD_Single-Instruction-Isolation-for-RISC-V-Vector-Test-Failures.pdf

Conclusions

RISC-V VP++

- https://github.com/ics-jku/riscv-vp-plusplus
- Fast and highly configurable RISC-V VP
- Research: Enabler for advanced verification and analysis approaches
  (e.g. Metamorphic Testing, RVVTS, HW/SW Coverage, function life-time diagrams, …)
- Industrial use: COSIDE integration, (and several others)

RISC-V VP++: Fast Open-Source Virtual Platforms - Powering Research Innovation and Industrial Use

Daniel Große
Institute for Complex Systems (ICS)
Web: jku.at/ics
Email: daniel.grosse@jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Selected RISC-V VP++ publications

VP Model

- A RISC-V CHERI VP: Enabling system-level evaluation of the capability-based CHERI architecture (ASP-DAC 2026)
- Fast interpreter-based instruction set simulation for virtual prototypes (DATE 2025)
- A RISC-V “V” VP: Unlocking Vector Processing for Evaluation at the System Level (DATE 2024)
- RISC-V VP++: Next generation open-source virtual prototype (OSDA 2024)
- GUI-VP Kit: A RISC-V VP meets Linux graphics - enabling interactive graphical application development (GLSVLSI 2023)

Verification

- LLM-assisted metamorphic testing of embedded graphics libraries (FDL 2025)
- ProtoLens: dynamic transaction visualization in virtual prototypes (FDL 2025)
- Single instruction isolation for RISC-V vector test failures (ICCAD 2024)
- Using virtual prototypes and metamorphic testing to verify the hardware/software-stack of embedded graphics libraries. (Integration 2025)
- Boosting SW development efficiency with function lifetime diagrams (DDECS 2025)
- Relation coverage: A new paradigm for hardware/software testing (ETS 2024)
- Verifying embedded graphics libraries leveraging virtual prototypes and metamorphic testing (ASP-DAC 2024)