Institute for Communication Technologies and Embedded Systems

RWTH AACHEN UNIVERSITY

MACHINEWARE

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Open-Source Virtual Platforms for Industry and Research

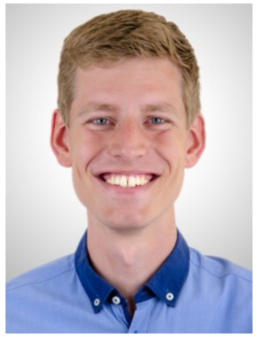
Nils Bosbach, RWTH Aachen University

Lukas Jünger, MachineWare GmbH

Rainer Leupers, RWTH Aachen University

accellera SYSTEMS INITIATIVE

About Us



Nils Bosbach

- PhD student at *Chair for Software for Systems on Silicon (SSS)*, RWTH Aachen University

Lukas Jünger

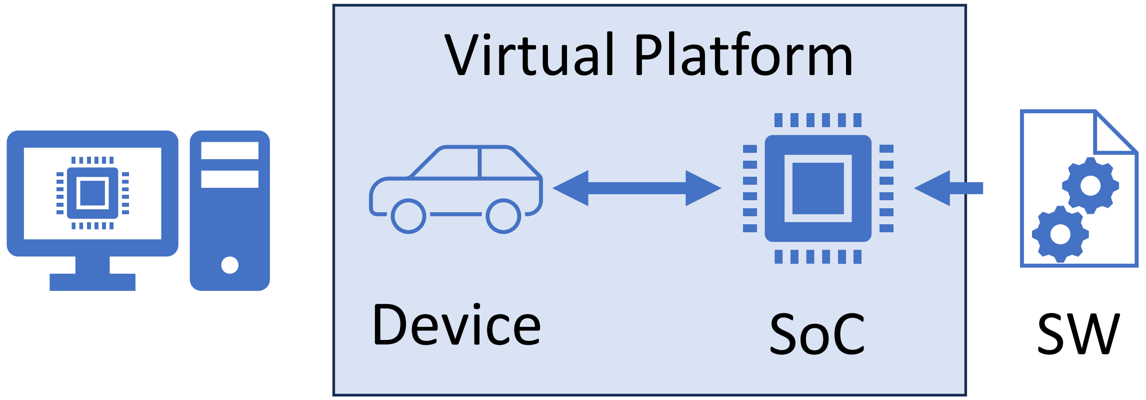
- Co-founder of *MachineWare GmbH*
- Chair of the *SystemC Configuration, Control and Inspection (CCI)* Working Group

What to Expect?

Open-Source Virtual Platforms for Industry and Research

What to Expect?

Open-Source Virtual Platforms for Industry and Research



"A virtual platform is a functional representation of a digital system written entirely in software."
- Cadence

What to Expect?

Open-Source Virtual Platforms for Industry and Research

- Prototyping
- Develop & test models
- Embedded SW development

- Prototyping of new architectures
- Architecture exploration
- Improve simulation performance

What to Expect?

Open-Source Virtual Platforms for Industry and Research

Many commercial solutions available

Solution we will talk about today:

Available on GitHub

- ✓ Clone
- ✓ Play around
- ✓ Extend

Free and/or Open-Source Software

- Disclaimer: I am not a lawyer!
- "Free of charge" or "free as in freedom"?
- Licenses

	Apache 2.0	GPLv2
Closed-source	Allowed	Not-allowed
Commercialization	Easy (no submarine patents)	Difficult
IP Protection	Easy (derived works under another license)	Difficult

- Are Apache 2.0 and GPLv2 compatible? Maybe

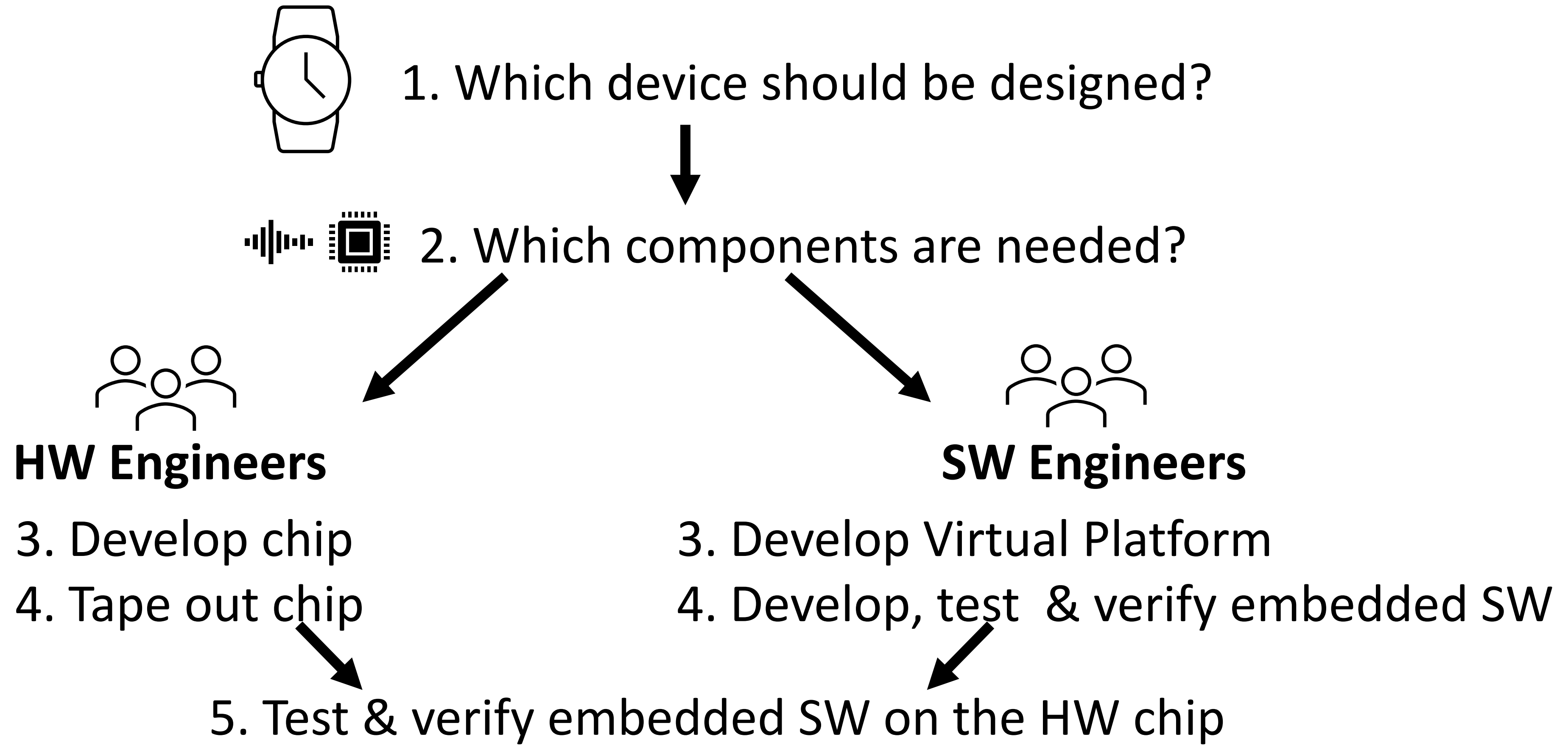
2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Virtual Platforms

A brief introduction

accellera SYSTEMS INITIATIVE

Why Do We Need Virtual Platforms?



1. Which device should be designed?

2. Which components are needed?

HW Engineers

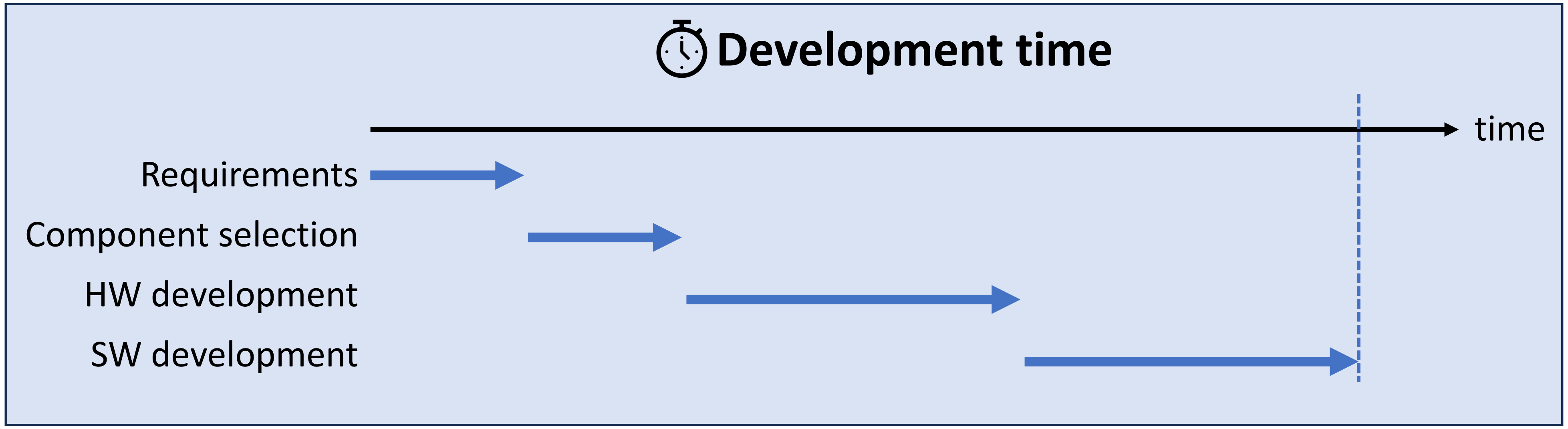
3. Develop chip
4. Tape out chip

SW Engineers

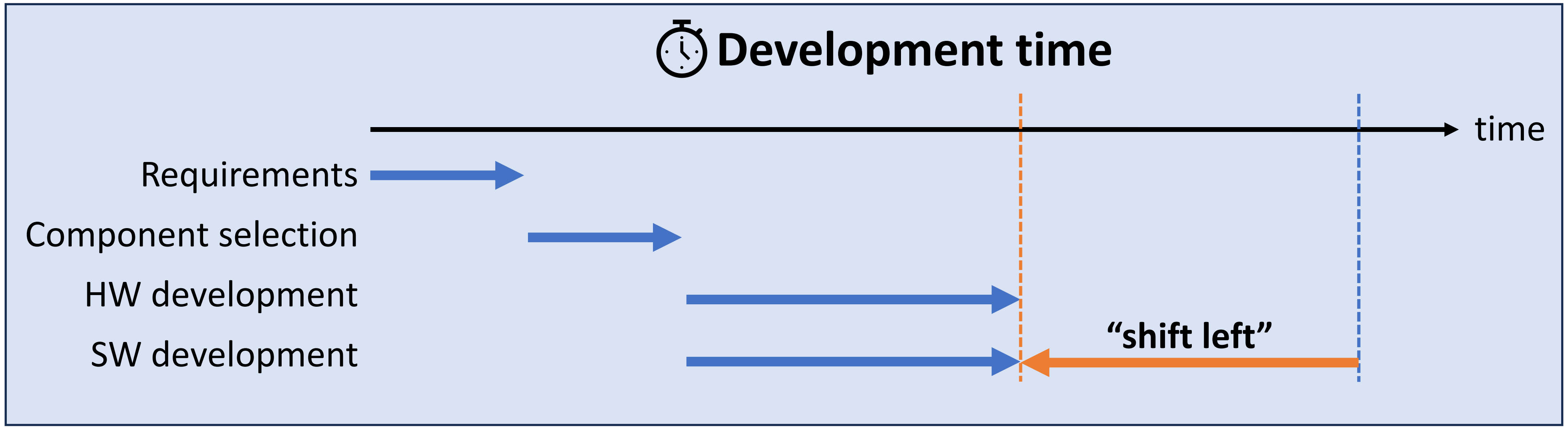
3. Develop Virtual Platform
4. Develop, test & verify embedded SW

5. Test & verify embedded SW on the HW chip

Why Should You Use Virtual Platforms?



Why Should You Use Virtual Platforms?



Scalability

- Easy distribution
- Integrates with CI

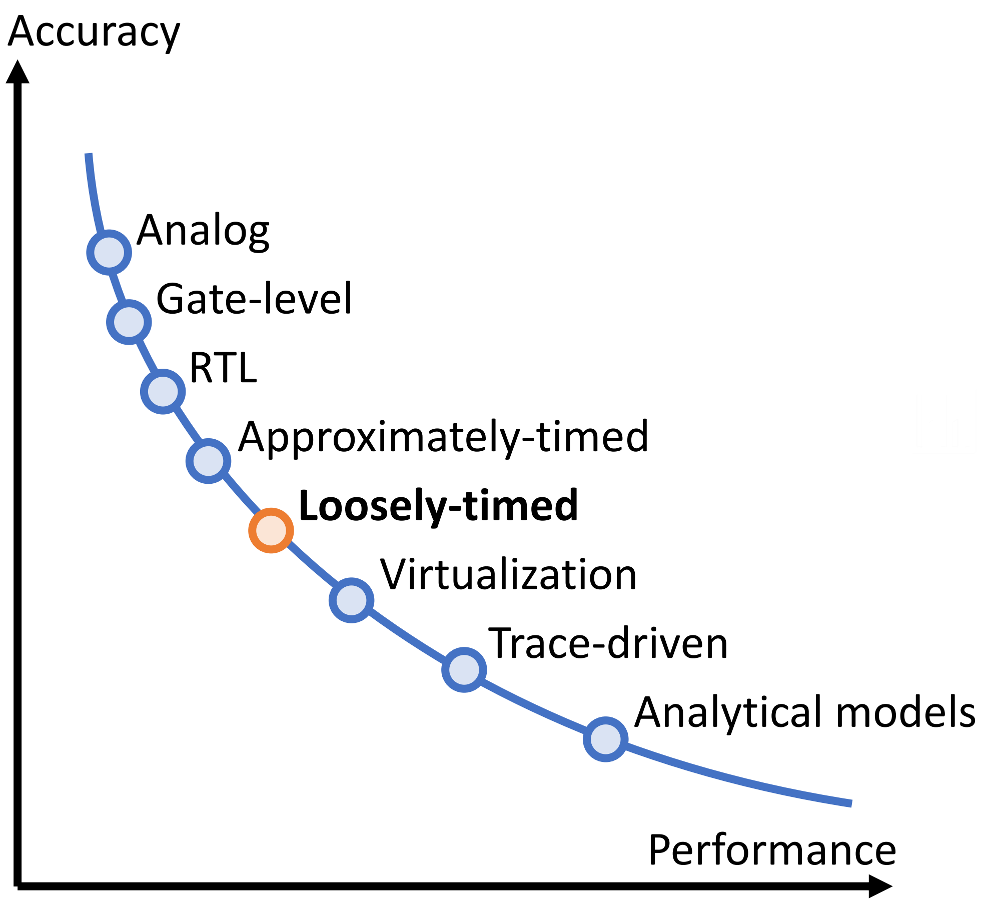
Introspection

- Stop at any time
- Access all register values

Tracing

- Record tracing data
- Analyze SW behavior

Abstraction Level



- Many different abstraction levels
- Tradeoff between performance and accuracy

Choose the level that fits your needs

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Virtual Platform Design

Let’s get started!

accellera SYSTEMS INITIATIVE

How to Start?

Specification

- List of components
- Interface description
- Functional behavior

Virtual Platform

- Simulates the behavior
- Can execute target SW

(Custom) Models — VCML-based models to mimic the behavior of the SoC

VCML — Virtual Components Modeling Library – adds modeling primitives

SYSTEM C™ — Standardized “design and verification language” (IEEE Std. 1666™-2011)

Why Do We Need a Modeling Library?

- ☑ Concept of time
- ☑ Simulated parallelism
- ☑ Standardized interfaces (ports, TLM sockets)
- ☑ Hierarchy (modules)
- ✗ Models (e.g., buses)
- ✗ Frequently-needed parts (e.g., register model)
- ✗ Often-used communication-protocol implementations (SPI, CAN, I²C, …)
- ✗ TLM logging/tracing, (parametrization, configuration)

} **VCML**

Virtual Components Modeling Library (VCML)

- Loosely-timed simulation framework based on SystemC TLM-2.0
- Apache-2.0 license
- Windows, Linux, MacOS
  - x86, arm64 CI builds
- Provides
  - Commonly used features (registers, peripherals, etc.)
  - Abstract protocols based on TLM-2.0 (interrupt, SPI, I^{2}C, etc.)
  - Models (memory, memory-mapped buses, UARTs, etc.)
- Extensive unit test suite

https://github.com/machineware-gmbh/vcml

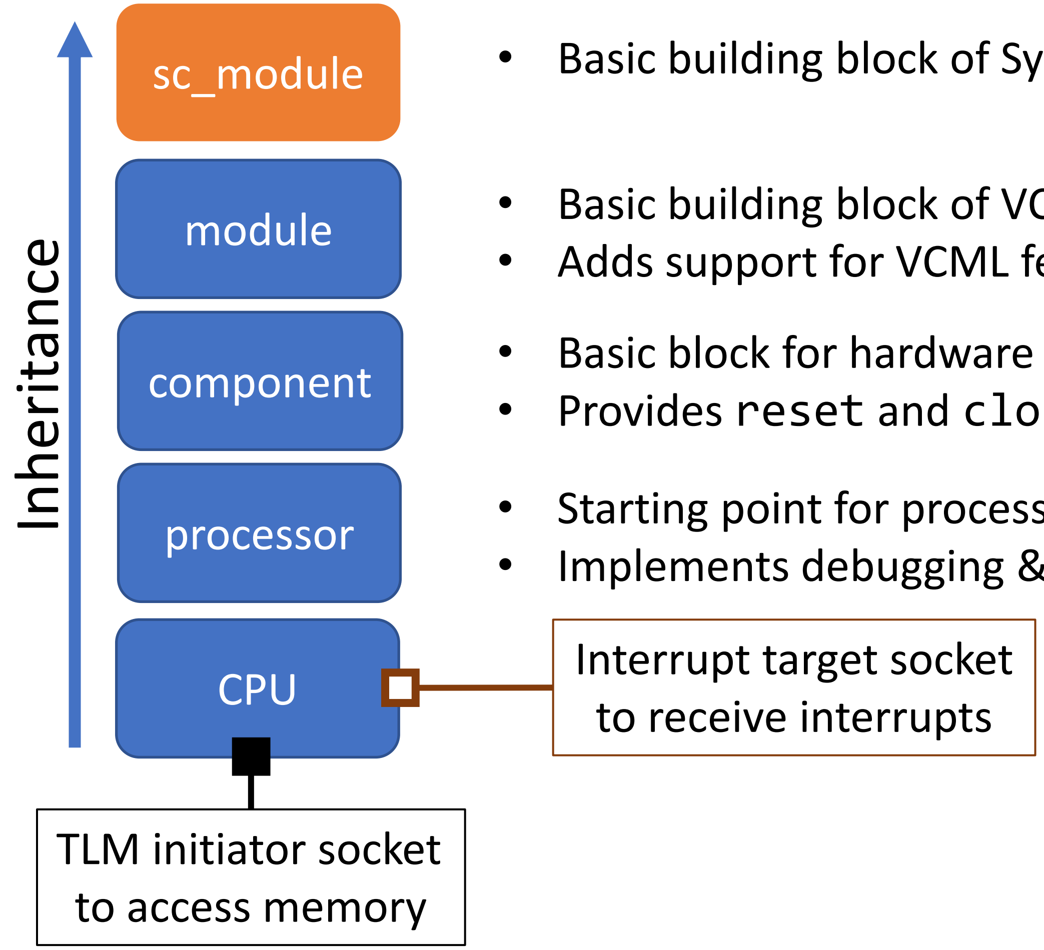
Why Should You Use VCML?

- Completely standard SystemC TLM-2.0 compatible
- Saves your time when construction a new VP
  - Reuse VCML features and models
- Supports all major Systems: Windows, Linux, MacOS, x86 and arm64
- Easy commercialization through Apache 2.0 license
- Example VPs available open-source (ARM and OpenRISC)
- Commercial support available through MachineWare

VCML-Highlights

- Modeling primitives
- Tracing
- Debugging support
- Backends
- Models
- Configurability
- Session Protocol
- Scripting

Implementation of the CPU Model



Inheritance
sc_module
module
component
processor
CPU
Interrupt target socket to receive interrupts
TLM initiator socket to access memory

- Basic building block of SystemC

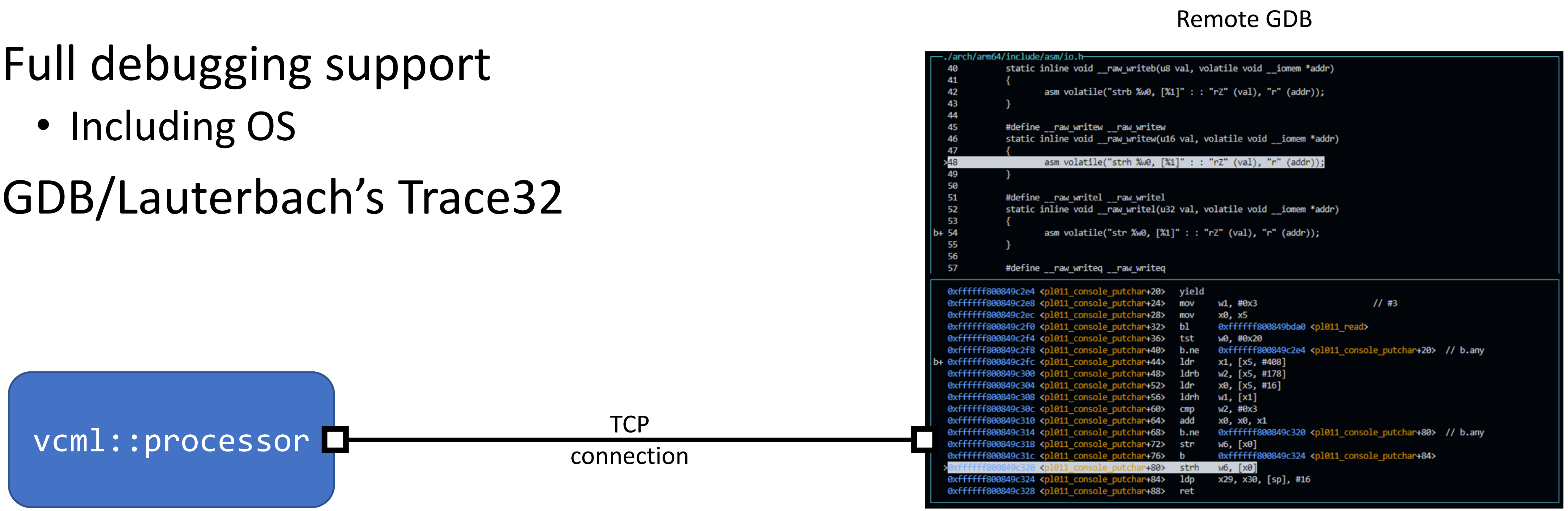
- Basic building block of VCML
- Adds support for VCML features (sockets, properties, commands, session)

- Basic block for hardware models
- Provides `reset` and `clock` sockets

- Starting point for processor implementation
- Implements debugging & temporal decoupling

Target Software Debugging

- Full debugging support
  - Including OS
- GDB/Lauterbach's Trace32



```
./arch/arm64/include/asm/io.h
        static inline void __raw_writeb(u8 val, volatile void __iomem *addr)
        {
                asm volatile("strb %w0, [%1]" : : "rZ" (val), "r" (addr));
        }

        #define __raw_writew __raw_writew
        static inline void __raw_writew(u16 val, volatile void __iomem *addr)
        {
>               asm volatile("strh %w0, [%1]" : : "rZ" (val), "r" (addr));
        }

        #define __raw_writel __raw_writel
        static inline void __raw_writel(u32 val, volatile void __iomem *addr)
        {
b+              asm volatile("str %w0, [%1]" : : "rZ" (val), "r" (addr));
        }

        #define __raw_writeq __raw_writeq
```

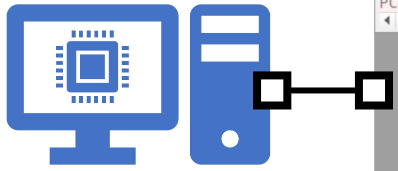
```
   0xffffff800849c2e4 <pl011_console_putchar+20>   yield
   0xffffff800849c2e8 <pl011_console_putchar+24>   mov    w1, #0x3                    // #3
   0xffffff800849c2ec <pl011_console_putchar+28>   mov    x0, x5
   0xffffff800849c2f0 <pl011_console_putchar+32>   bl     0xffffff800849bda0 <pl011_read>
   0xffffff800849c2f4 <pl011_console_putchar+36>   tst    w0, #0x20
   0xffffff800849c2f8 <pl011_console_putchar+40>   b.ne   0xffffff800849c2e4 <pl011_console_putchar+20>  // b.any
b+ 0xffffff800849c2fc <pl011_console_putchar+44>   ldr    x1, [x5, #408]
   0xffffff800849c300 <pl011_console_putchar+48>   ldrb   w2, [x5, #178]
   0xffffff800849c304 <pl011_console_putchar+52>   ldr    x0, [x5, #16]
   0xffffff800849c308 <pl011_console_putchar+56>   ldrh   w1, [x1]
   0xffffff800849c30c <pl011_console_putchar+60>   cmp    w2, #0x3
   0xffffff800849c310 <pl011_console_putchar+64>   add    x0, x0, x1
   0xffffff800849c314 <pl011_console_putchar+68>   b.ne   0xffffff800849c320 <pl011_console_putchar+80>  // b.any
   0xffffff800849c318 <pl011_console_putchar+72>   str    w6, [x0]
   0xffffff800849c31c <pl011_console_putchar+76>   b      0xffffff800849c324 <pl011_console_putchar+84>
  >0xffffff800849c320 <pl011_console_putchar+80>   strh   w6, [x0]
   0xffffff800849c324 <pl011_console_putchar+84>   ldp    x29, x30, [sp], #16
   0xffffff800849c328 <pl011_console_putchar+88>   ret
```

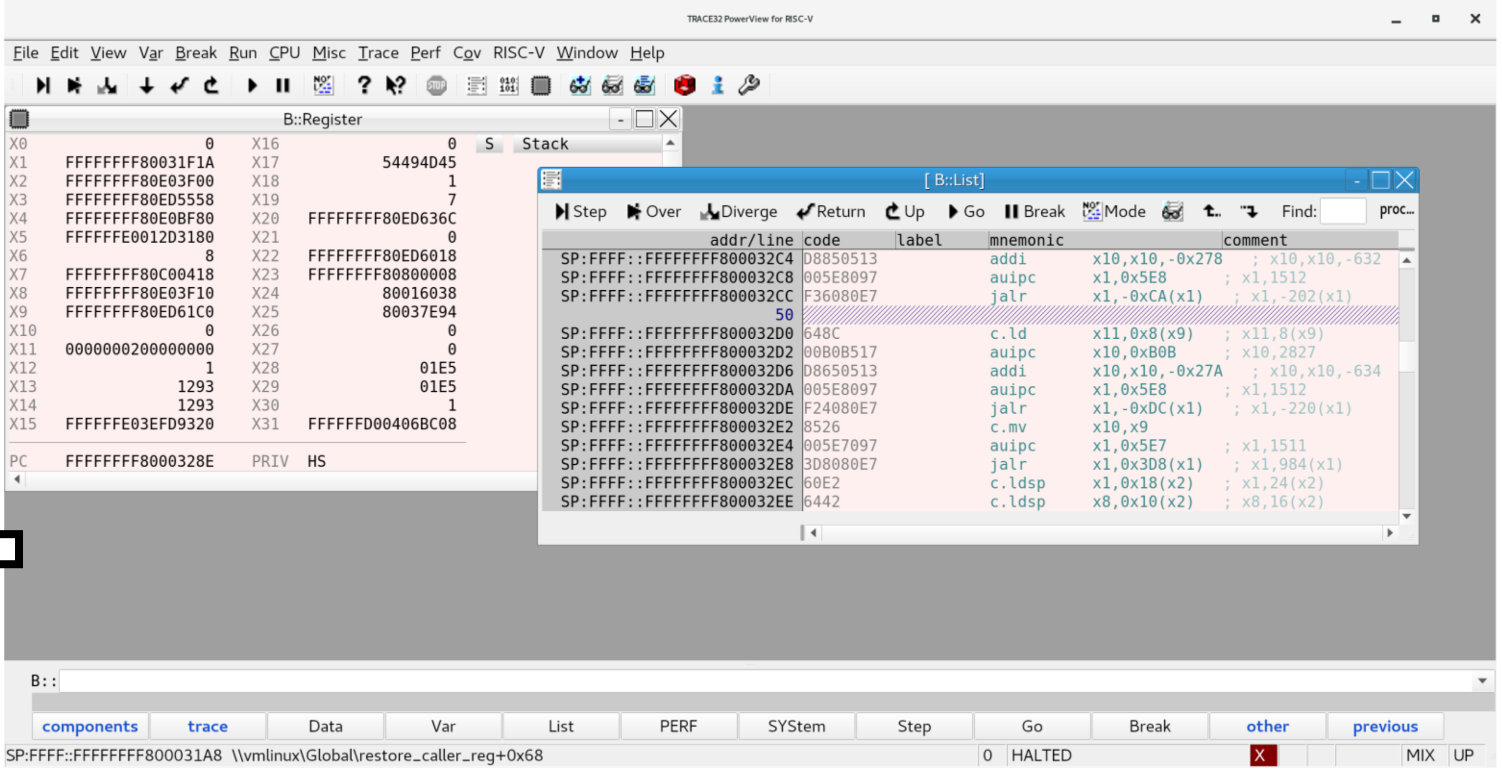
Lauterbach‘s Trace32

- Multicore debugging
- OS awareness
- User-space debugging

Watch our Trace32 demo:

https://youtu.be/B6zys6_M-k4




TRACE32 PowerView for RISC-V
File Edit View Var Break Run CPU Misc Trace Perf Cov RISC-V Window Help
Step Over Diverge Return Up Go Break Mode Find: proc...
components trace Data Var List PERF SYStem Step Go Break other previous

B::Register

```
X0                 0    X16                0   S   Stack
X1  FFFFFFFF80031F1A    X17         54494D45
X2  FFFFFFFF80E03F00    X18                1
X3  FFFFFFFF80ED5558    X19                7
X4  FFFFFFFF80E0BF80    X20 FFFFFFFF80ED636C
X5  FFFFFFE0012D3180    X21                0
X6                 8    X22 FFFFFFFF80ED6018
X7  FFFFFFFF80C00418    X23 FFFFFFFF80800008
X8  FFFFFFFF80E03F10    X24         80016038
X9  FFFFFFFF80ED61C0    X25         80037E94
X10                0    X26                0
X11 0000000200000000    X27                0
X12                1    X28             01E5
X13             1293    X29             01E5
X14             1293    X30                1
X15 FFFFFFE03EFD9320    X31 FFFFFFD00406BC08

PC  FFFFFFFF8000328E    PRIV HS
```

[B::List]

```
           addr/line code     label mnemonic                      comment
SP:FFFF::FFFFFFFF800032C4 D8850513  addi   x10,x10,-0x278   ; x10,x10,-632
SP:FFFF::FFFFFFFF800032C8 005E8097  auipc  x1,0x5E8         ; x1,1512
SP:FFFF::FFFFFFFF800032CC F36080E7  jalr   x1,-0xCA(x1)     ; x1,-202(x1)
                       50
SP:FFFF::FFFFFFFF800032D0 648C      c.ld   x11,0x8(x9)      ; x11,8(x9)
SP:FFFF::FFFFFFFF800032D2 00B0B517  auipc  x10,0xB0B        ; x10,2827
SP:FFFF::FFFFFFFF800032D6 D8650513  addi   x10,x10,-0x27A   ; x10,x10,-634
SP:FFFF::FFFFFFFF800032DA 005E8097  auipc  x1,0x5E8         ; x1,1512
SP:FFFF::FFFFFFFF800032DE F24080E7  jalr   x1,-0xDC(x1)     ; x1,-220(x1)
SP:FFFF::FFFFFFFF800032E2 8526      c.mv   x10,x9
SP:FFFF::FFFFFFFF800032E4 005E7097  auipc  x1,0x5E7         ; x1,1511
SP:FFFF::FFFFFFFF800032E8 3D8080E7  jalr   x1,0x3D8(x1)     ; x1,984(x1)
SP:FFFF::FFFFFFFF800032EC 60E2      c.ldsp x1,0x18(x2)      ; x1,24(x2)
SP:FFFF::FFFFFFFF800032EE 6442      c.ldsp x8,0x10(x2)      ; x8,16(x2)
```

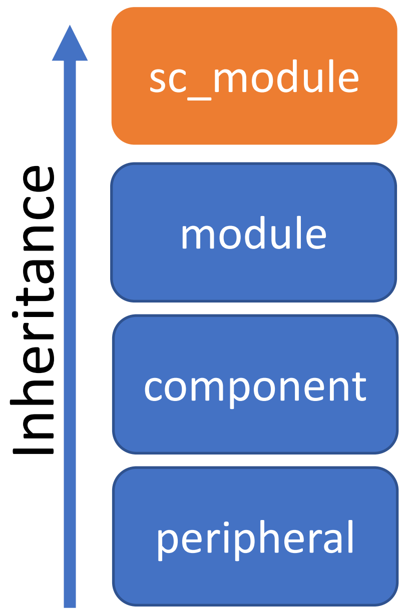
```
B::
SP:FFFF::FFFFFFFF800031A8  \\vmlinux\Global\restore_caller_reg+0x68    0  HALTED   MIX  UP
```

SIM-V and Lauterbach Trace32

Like real hardware, just better

MACHINEWARE

How to Implement Peripherals?

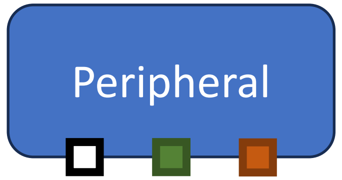


- Basic building block of SystemC

- Basic building block of VCML
- Adds support for VCML features (sockets, properties, commands, …)
- Basic block for hardware models
- Provides `reset` and `clock` sockets
- Starting point for custom I/O peripheral models
- Adds support for registers

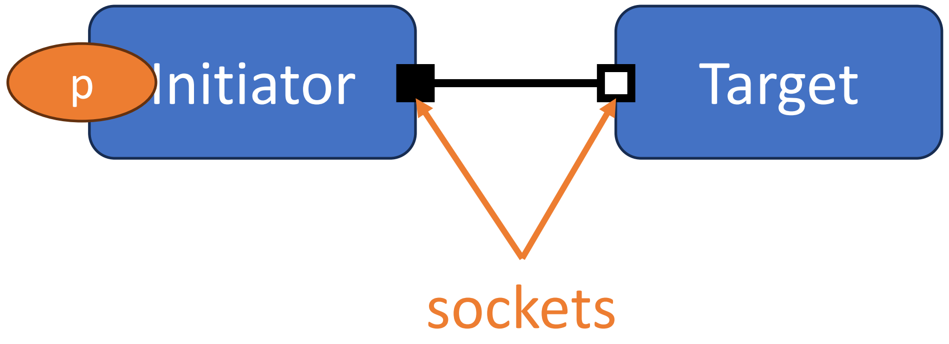
Peripheral Interfaces – TLM Protocols

Interfaces



- Memory accesses
- Interrupt
- Communication/data transfer (e.g., SPI, I²C, …)

Based on TLM blocking transports

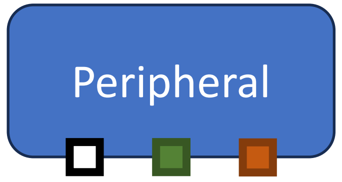


Payload

- Operation (R/W)
- Data pointer
- Attributes

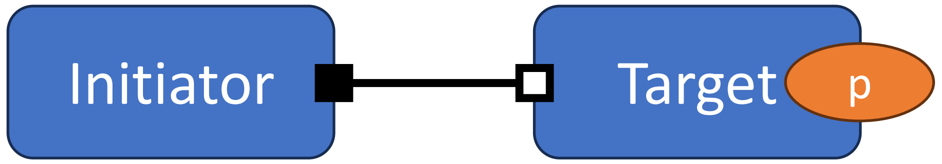
Peripheral Interfaces – TLM Protocols

Interfaces



- Memory accesses
- Interrupt
- Communication/data transfer (e.g., SPI, I²C, …)

Based on TLM blocking transports

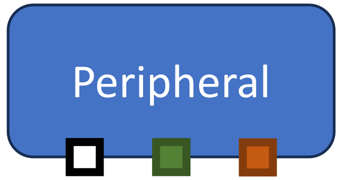


Process payload

- Execute operation (R/W)

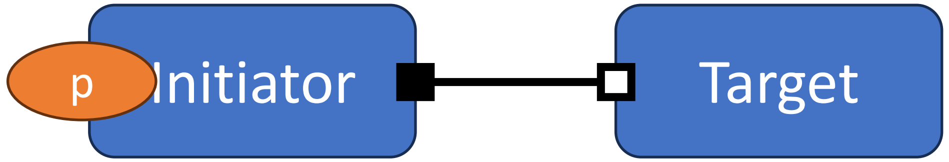
Peripheral Interfaces – TLM Protocols

Interfaces



- Memory accesses
- Interrupt
- Communication/data transfer (e.g., SPI, I^{2}C, …)

Based on TLM blocking transports

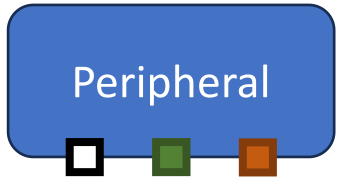


Send payload back

- Analyze returned payload

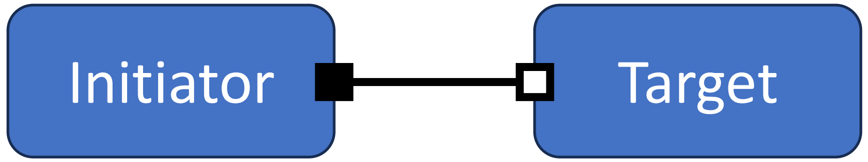
Peripheral Interfaces – TLM Protocols

Interfaces



- Memory accesses
- Interrupt
- Communication/data transfer (e.g., SPI, I²C, …)

Based on TLM blocking transports



Implemented protocols

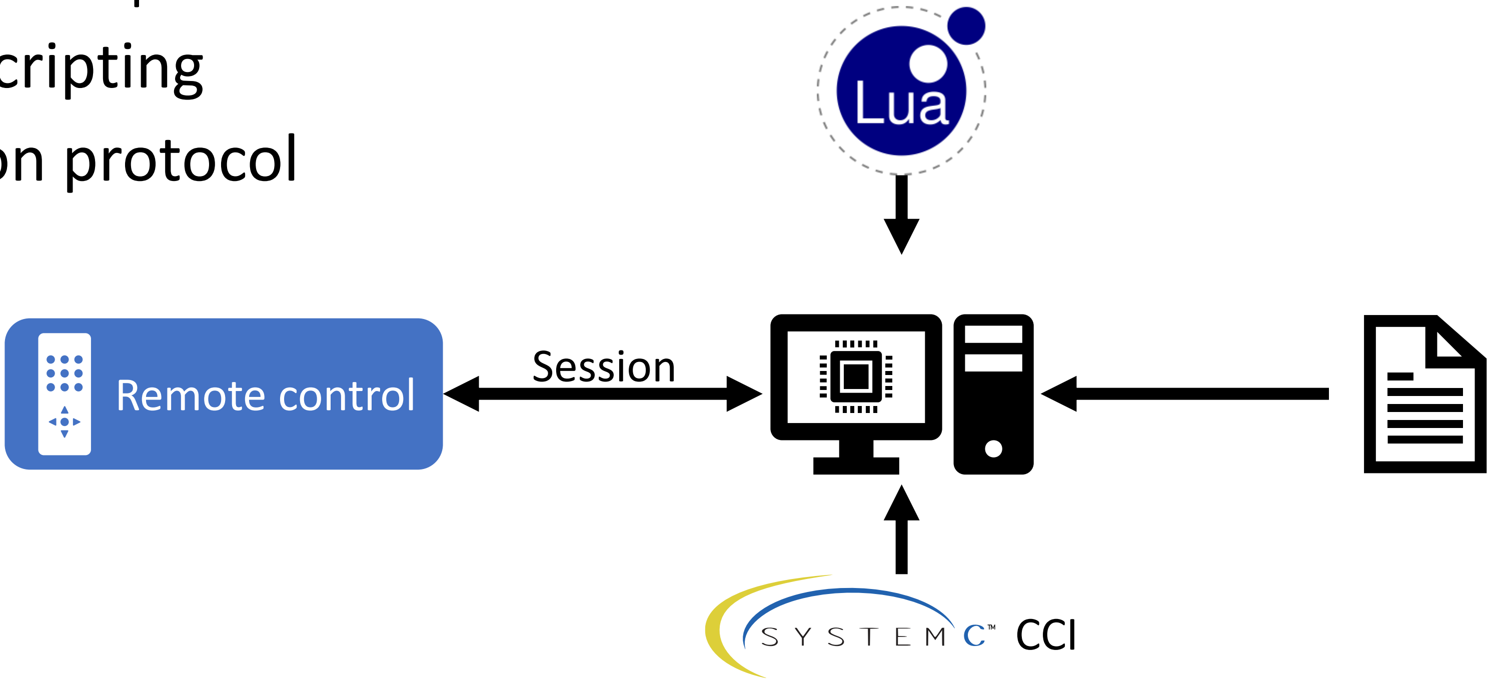
- CAN
- Clock
- Ethernet
- GPIO
- I²C
- PCI
- SD
- Serial
- SPI
- VirtIO

Predefined Peripherals in VCML

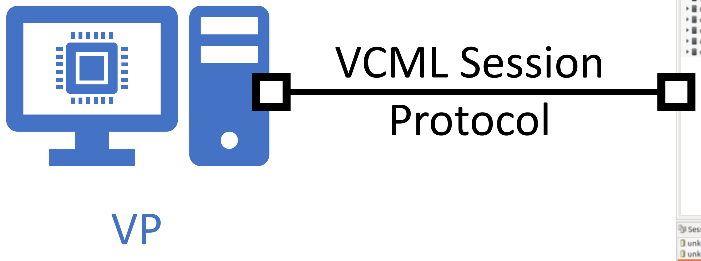
- Memory
- Interrupt controllers (ARM, RISC-V)
- Ethernet, SPI, I^2C controller
- SD controller
- UARTs (e.g., PL011)
- Sensors (SPI/I^2C-based)
- VirtIO (controller, console, RNG, block, ...)
- CAN (bridge & bus)

Configurability

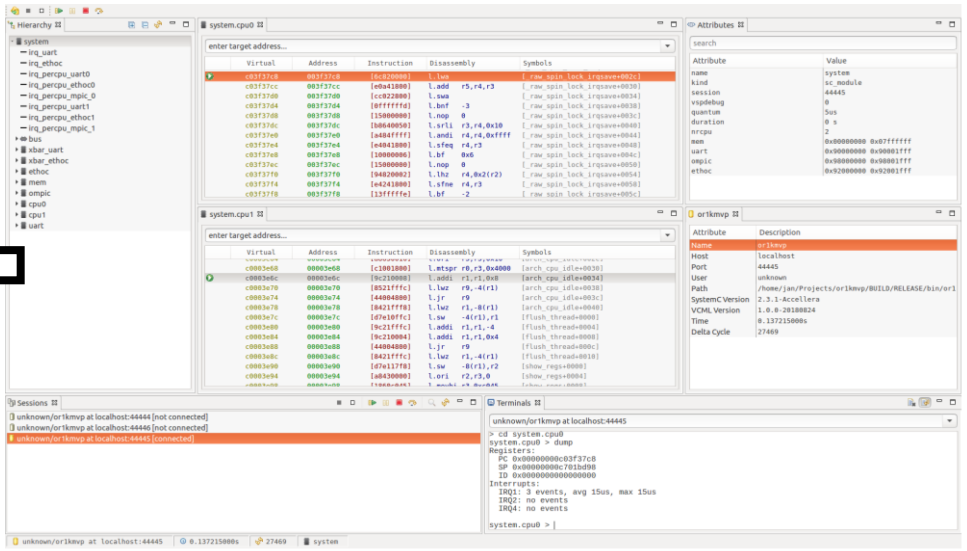
- VCML provides properties to configure modules
  - CCI-compatible
- LUA scripting
- Session protocol



Session



Virtual Platform Explorer (ViPER)



- system
  - irq_uart
  - irq_ethoc
  - irq_percpu_uart0
  - irq_percpu_ethoc0
  - irq_percpu_mpic_0
  - irq_percpu_uart1
  - irq_percpu_ethoc1
  - irq_percpu_mpic_1
  - bus
  - xbar_uart
  - xbar_ethoc
  - ethoc
  - mem
  - ompic
  - cpu0
  - cpu1
  - uart

system.cpu0

enter target address...

```
Virtual   Address   Instruction  Disassembly          Symbols
c03f37c8  003f37c8  [6c820000]   l.lwa                [_raw_spin_lock_irqsave+002c]
c03f37cc  003f37cc  [e0a41800]   l.add   r5,r4,r3     [_raw_spin_lock_irqsave+0030]
c03f37d0  003f37d0  [cc022800]   l.swa                [_raw_spin_lock_irqsave+0034]
c03f37d4  003f37d4  [0ffffffd]   l.bnf   -3           [_raw_spin_lock_irqsave+0038]
c03f37d8  003f37d8  [15000000]   l.nop   0            [_raw_spin_lock_irqsave+003c]
c03f37dc  003f37dc  [b8640050]   l.srli  r3,r4,0x10   [_raw_spin_lock_irqsave+0040]
c03f37e0  003f37e0  [a484ffff]   l.andi  r4,r4,0xffff [_raw_spin_lock_irqsave+0044]
c03f37e4  003f37e4  [e4041800]   l.sfeq  r4,r3        [_raw_spin_lock_irqsave+0048]
c03f37e8  003f37e8  [10000006]   l.bf    0x6          [_raw_spin_lock_irqsave+004c]
c03f37ec  003f37ec  [15000000]   l.nop   0            [_raw_spin_lock_irqsave+0050]
c03f37f0  003f37f0  [94820002]   l.lhz   r4,0x2(r2)   [_raw_spin_lock_irqsave+0054]
c03f37f4  003f37f4  [e4241800]   l.sfne  r4,r3        [_raw_spin_lock_irqsave+0058]
c03f37f8  003f37f8  [13fffffe]   l.bf    -2           [_raw_spin_lock_irqsave+005c]
```

Attribute	Value
name	system
kind	sc_module
session	44445
vspdebug	0
quantum	5us
duration	0 s
nrcpu	2
mem	0x00000000 0x07ffffff
uart	0x90000000 0x90001fff
ompic	0x98000000 0x98001fff
ethoc	0x92000000 0x92001fff

system.cpu1

enter target address...

```
Virtual   Address   Instruction  Disassembly            Symbols
c0003e68  00003e68  [c1001800]   l.mtspr r0,r3,0x4000   [arch_cpu_idle+0030]
c0003e6c  00003e6c  [9c210008]   l.addi  r1,r1,0x8      [arch_cpu_idle+0034]
c0003e70  00003e70  [8521fffc]   l.lwz   r9,-4(r1)      [arch_cpu_idle+0038]
c0003e74  00003e74  [44004800]   l.jr    r9             [arch_cpu_idle+003c]
c0003e78  00003e78  [8421fff8]   l.lwz   r1,-8(r1)      [arch_cpu_idle+0040]
c0003e7c  00003e7c  [d7e10ffc]   l.sw    -4(r1),r1      [flush_thread+0000]
c0003e80  00003e80  [9c21fffc]   l.addi  r1,r1,-4       [flush_thread+0004]
c0003e84  00003e84  [9c210004]   l.addi  r1,r1,0x4      [flush_thread+0008]
c0003e88  00003e88  [44004800]   l.jr    r9             [flush_thread+000c]
c0003e8c  00003e8c  [8421fffc]   l.lwz   r1,-4(r1)      [flush_thread+0010]
c0003e90  00003e90  [d7e117f8]   l.sw    -8(r1),r2      [show_regs+0000]
c0003e94  00003e94  [a8430000]   l.ori   r2,r3,0        [show_regs+0004]
```

or1kmvp

Attribute	Description
Name	or1kmvp
Host	localhost
Port	44445
User	unknown
Path	/home/jan/Projects/or1kmvp/BUILD/RELEASE/bin/or1
SystemC Version	2.3.1-Accellera
VCML Version	1.0.0-20180824
Time	0.137215000s
Delta Cycle	27469

unknown/or1kmvp at localhost:44444 [not connected]
unknown/or1kmvp at localhost:44446 [not connected]
unknown/or1kmvp at localhost:44445 [connected]

unknown/or1kmvp at localhost:44445

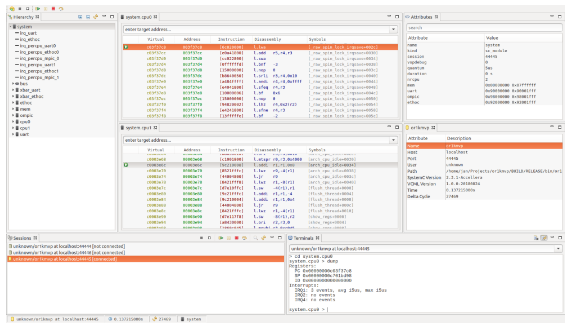
```
> cd system.cpu0
system.cpu0 > dump
Registers:
  PC 0x00000000c03f37c8
  SP 0x00000000c701bd98
  ID 0x0000000000000000
Interrupts:
  IRQ1: 3 events, avg 15us, max 15us
  IRQ2: no events
  IRQ4: no events

system.cpu0 > |
```

unknown/or1kmvp at localhost:44445 | 0.137215000s | 27469 | system

- Control the simulation
- Access/inspect models
- Open, TCP-based protocol

Session

Virtual Platform Explorer (ViPER)



```
Virtual    Address    Instruction  Disassembly          Symbols
c03f37c8   003f37c8   [6c820000]   l.lwa                [_raw_spin_lock_irqsave+002c]
c03f37cc   003f37cc   [e0a41800]   l.add   r5,r4,r3     [_raw_spin_lock_irqsave+0030]
c03f37d0   003f37d0   [cc022800]   l.swa                [_raw_spin_lock_irqsave+0034]
c03f37d4   003f37d4   [0ffffffd]   l.bnf   -3           [_raw_spin_lock_irqsave+0038]
c03f37d8   003f37d8   [15000000]   l.nop   0            [_raw_spin_lock_irqsave+003c]
c03f37dc   003f37dc   [b8640050]   l.srli  r3,r4,0x10   [_raw_spin_lock_irqsave+0040]
c03f37e0   003f37e0   [a484ffff]   l.andi  r4,r4,0xffff [_raw_spin_lock_irqsave+0044]
c03f37e4   003f37e4   [e4041800]   l.sfeq  r4,r3        [_raw_spin_lock_irqsave+0048]
c03f37e8   003f37e8   [10000006]   l.bf    0x6          [_raw_spin_lock_irqsave+004c]
c03f37ec   003f37ec   [15000000]   l.nop   0            [_raw_spin_lock_irqsave+0050]
c03f37f0   003f37f0   [94820002]   l.lhz   r4,0x2(r2)   [_raw_spin_lock_irqsave+0054]
c03f37f4   003f37f4   [e4241800]   l.sfne  r4,r3        [_raw_spin_lock_irqsave+0058]
c03f37f8   003f37f8   [13fffffe]   l.bf    -2           [_raw_spin_lock_irqsave+005c]
```

Attribute	Value
name	system
kind	sc_module
session	44445
vspdebug	0
quantum	5us
duration	0 s
nrcpu	2
mem	0x00000000 0x07ffffff
uart	0x90000000 0x90001fff
ompic	0x98000000 0x98001fff
ethoc	0x92000000 0x92001fff

```
Virtual    Address    Instruction  Disassembly           Symbols
c0003e68   00003e68   [c1001800]   l.mtspr r0,r3,0x4000  [arch_cpu_idle+0030]
c0003e6c   00003e6c   [9c210008]   l.addi  r1,r1,0x8     [arch_cpu_idle+0034]
c0003e70   00003e70   [8521fffc]   l.lwz   r9,-4(r1)     [arch_cpu_idle+0038]
c0003e74   00003e74   [44004800]   l.jr    r9            [arch_cpu_idle+003c]
c0003e78   00003e78   [8421fff8]   l.lwz   r1,-8(r1)     [arch_cpu_idle+0040]
c0003e7c   00003e7c   [d7e10ffc]   l.sw    -4(r1),r1     [flush_thread+0000]
c0003e80   00003e80   [9c21fffc]   l.addi  r1,r1,-4      [flush_thread+0004]
c0003e84   00003e84   [9c210004]   l.addi  r1,r1,0x4     [flush_thread+0008]
c0003e88   00003e88   [44004800]   l.jr    r9            [flush_thread+000c]
c0003e8c   00003e8c   [8421fffc]   l.lwz   r1,-4(r1)     [flush_thread+0010]
c0003e90   00003e90   [d7e117f8]   l.sw    -8(r1),r2     [show_regs+0000]
c0003e94   00003e94   [a8430000]   l.ori   r2,r3,0       [show_regs+0004]
```

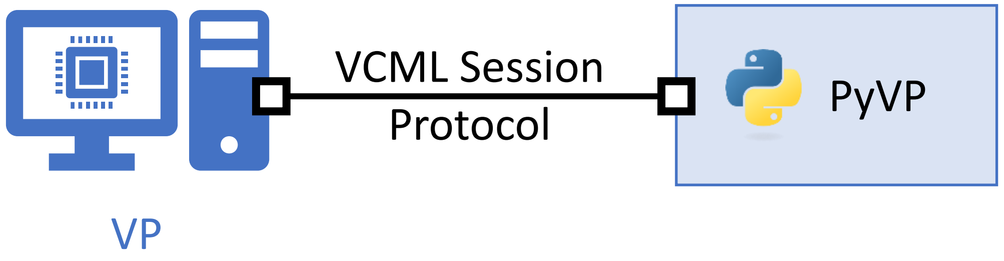
Attribute	Description
Name	or1kmvp
Host	localhost
Port	44445
User	unknown
Path	/home/jan/Projects/or1kmvp/BUILD/RELEASE/bin/or1
SystemC Version	2.3.1-Accellera
VCML Version	1.0.0-20180824
Time	0.137215000s
Delta Cycle	27469

unknown/or1kmvp at localhost:44444 [not connected]
unknown/or1kmvp at localhost:44446 [not connected]
unknown/or1kmvp at localhost:44445 [connected]

unknown/or1kmvp at localhost:44445

```
> cd system.cpu0
system.cpu0 > dump
Registers:
  PC 0x00000000c03f37c8
  SP 0x00000000c701bd98
  ID 0x0000000000000000
Interrupts:
  IRQ1: 3 events, avg 15us, max 15us
  IRQ2: no events
  IRQ4: no events

system.cpu0 > |
```

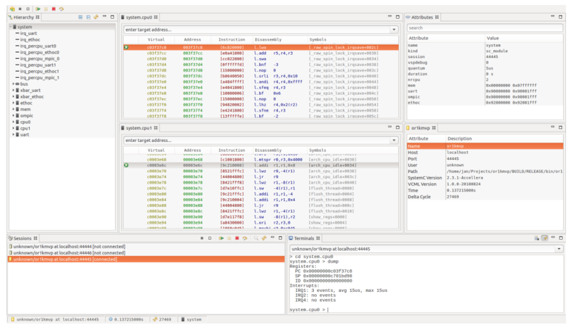
unknown/or1kmvp at localhost:44445 | 0.137215000s | 27469 | system

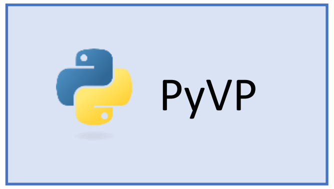


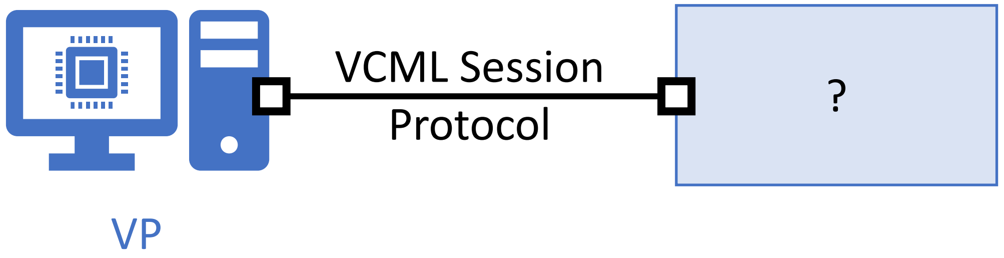
- Python-based implementation of a Session client
- Use Python to script the simulation

Session

Virtual Platform Explorer (ViPER)







- Infinite number of use cases
- Very powerful interface
- Simple usage

Free Implementations

- OpenRISC 1000 Multicore Virtual Platform (OR1KMVP)
  - Embeds the open-source OR1KISS into `vcml::processor`
    - Simple interpreter-based instruction-set simulator (ISS)
  - https://github.com/janweinstock/or1kmvp

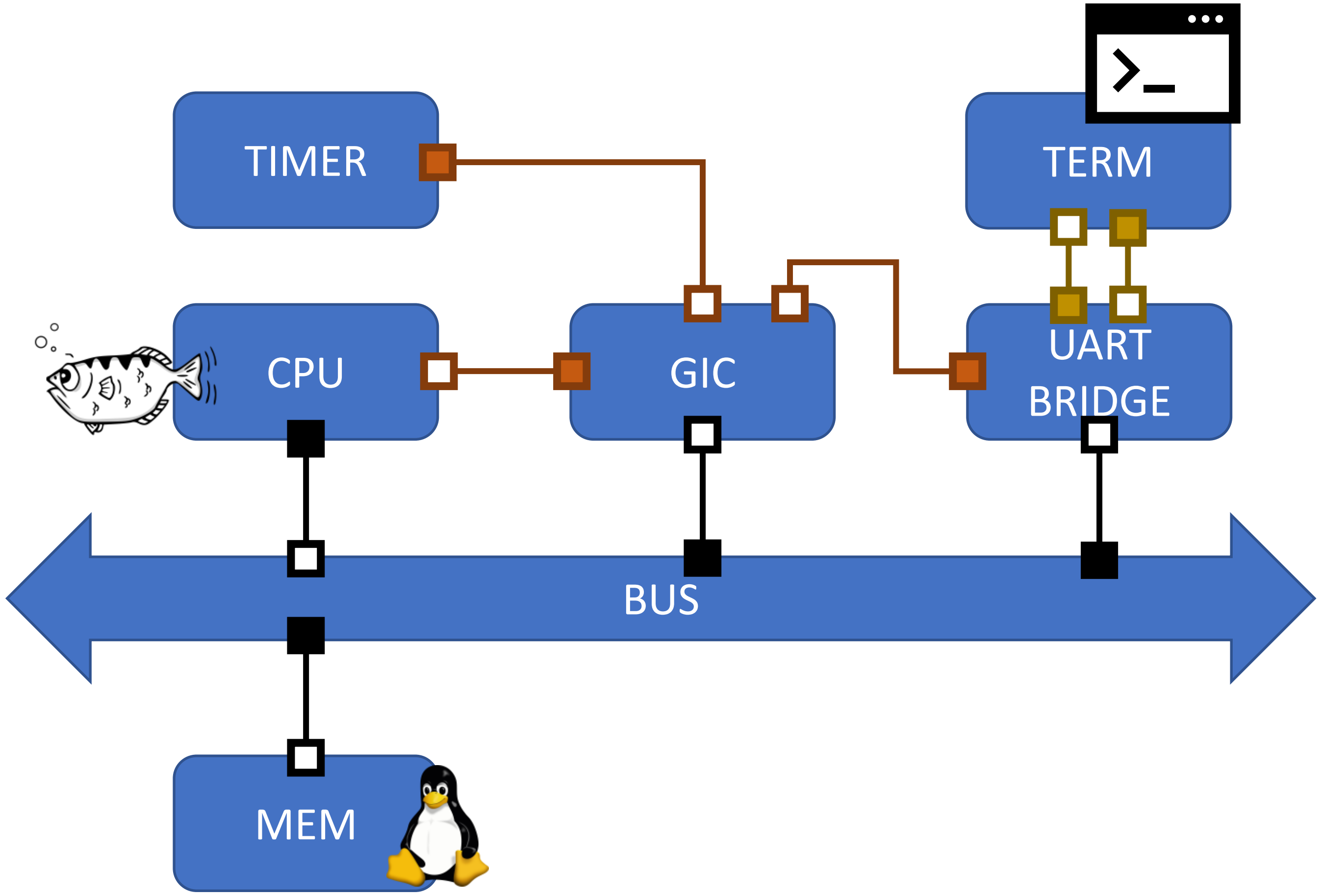
- An ARMv8 Virtual Platform (AVP64)
  - Embeds the ARMv8 unicorn implementation (QEMU-based) into `vcml::processor`
    - Dynamic-binary-translation (DBT)-based ISS
  - https://github.com/aut0/avp64

Commercial Implementations

- SIM-V
  - Ultra-fast RISC-V simulator
  - Supports state-of-the-art extensions
  - Custom-instruction API
- SIM-A
  - Ultra-fast ARM Cortex-M simulator
- QBox
  - QEMU in SystemC TLM-2.0 through VCML

Full-System Integration

- Instantiate models
- Bind sockets
- Start simulation



SW Development

Demo - Find the tutorial on the AVP64 GitHub page

https://github.com/aut0/avp64/tree/master/vscode

Summary

From spec to chip

Slides: https://mwa.re/vcml-tutorial

Summary

VP Benefits

- Development Time
- Introspection
- Scalability
- Tracing

SW Stack

- (Custom) Models
- VCML
- C++ SYSTEMC™